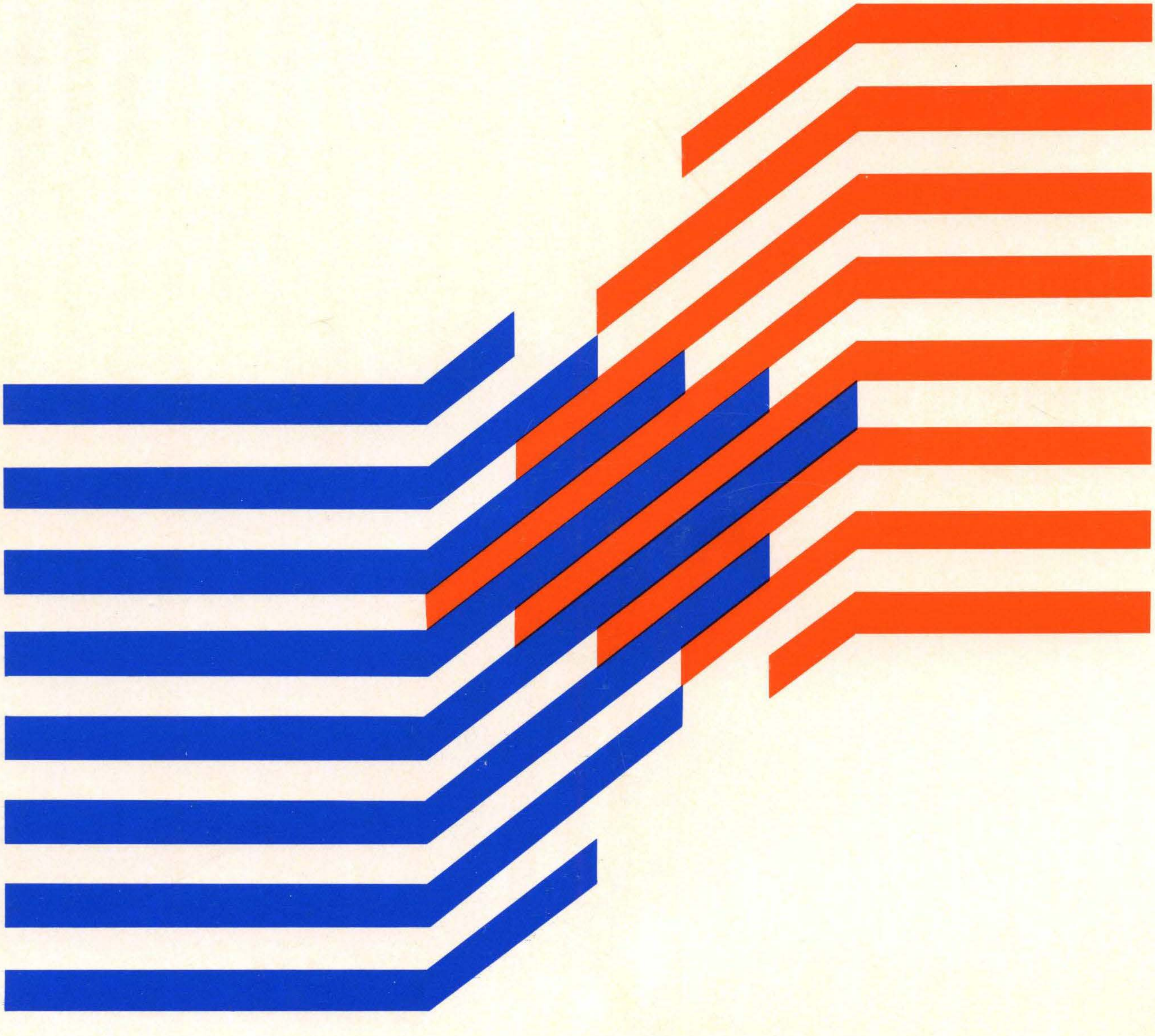
IBM

LY28-1006-2

MVS/ESA JES2 Logic

MVS/System Product: JES2 Version 3



IBM

LY28-1006-2

MVS/ESA JES2 Logic

MVS/System Product: JES2 Version 3

Production of This Book

This book was prepared and formatted using the IBM BookMaster document markup language.

Keep your Books

Do not replace your existing documentation until you install the JES2 component of MVS/SP Version 3 Release 1.3 on ALL members of your multi-access spool system.

Third Edition (February, 1990)

This is a major revision of LY28-1006-1. See the Summary of Changes for a summary of changes made to this manual. Technical changes or additions to the text and illustrations are indicated by a vertical line to the left of the change.

This edition applies to Version 3 Release 1.3 of MVS/System Product -- JES2 5685-001, and to all subsequent releases until otherwise indicated in new editions or Technical Newsletters. The previous edition still applies to Version 3 Release 1.0 of MVS/System Product-JES2 and may be ordered using temporary order number LT00-3539. Changes are made periodically to the information herein; before using this publication in connection with the operation of IBM systems, consult the *System/370 Bibliography*, GC20-0001, for the editions that are applicable and current.

References in this publication to IBM products or services do not imply that IBM intends to make these available in all countries in which IBM operates. References to IBM products in this document do not imply that functionally equivalent products may be used. The security certification of the trusted computing base that includes the products discussed herein covers certain IBM products. Please contact the manufacturer of any product you may consider to be functionally equivalent for information on that product's security classification. This statement does not expressly or implicitly waive any intellectual property right IBM may hold in any product mentioned herein.

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

A form for reader's comments is provided at the back of this publication. If the form has been removed, comments may be addressed to IBM Corporation, Information Development, Department D58, Building 921-2, PO Box 950, Poughkeepsie, N.Y. 12602. IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

PROGRAMMING INTERFACES

This book consists entirely of diagnostic information. Such information should never be used as programming interface information.

Contents

Figures

About This Book

Who This Book Is For

Anyone interested in determining sources of errors within JES2 should read this publication. Readers must be familiar with programming techniques and the operating principles of MVS.

This manual describes the logic of JES2 as it applies MVS/ESA systems. Use of this manual does not replace use of the program listings; the manual supplements the listings and makes the information in them more accessible.

This publication uses several terms to describe various aspects of JES2 function. These terms include:

- JES2 - describes the basic functions of JES2 where no specific reference to network job entry capabilities is required.
- NJE - describes the network job entry capabilities of JES2. This term is used where no differentiation needs to be made between binary synchronous communications (BSC) networking capabilities and Systems Network Architecture (SNA) networking capabilities.
- SNA NJE - describes the network job entry capabilities specifically when using Systems Network Architecture (SNA) protocols.

Trademarks

The following are trademarks of International Business Machines Corporation:

- MVS/ESA™
- MVS/XA™
- MVS/370™
- BookMaster®

How This Book Is Organized

The organization and content of each chapter are:

- "Chapter 1. Introduction", describes the general characteristics of JES2.
- "Chapter 2. Method of Operation", contains HIPO (hierarchy plus input-process-output) diagrams that describe the operation of JES2. The diagrams are high-level and are intended to guide the reader to a particular area of the program listing or to more detailed descriptions contained in Chapter 3 of this manual.
- "Chapter 3. Program Organization", describes the general program organization of JES2 and its modules. It includes detailed descriptions of the routines and subroutines that comprise the modules.
- "Chapter 4. Directory", lists main entry points in JES2 modules and provides a brief description of each.

- "Chapter 5. Diagnostic Aids", provides error analysis techniques, describes JES2 structure and processing from a diagnostic perspective, and presents details about control block usage.

The appendixes contain the following additional information:

- "Appendix A. Multileaving", describes the basic principles of multileaving.
- "Appendix B. External Writer", describes the external writer program used to produce SYSOUT data sets on magnetic tape or direct-access devices.
- "Appendix C. Subsystem Vector Table", lists the various subsystem vector table (SSVT) fields and describes their use.
- "Appendix D. JES2 Acronyms", lists the various acronyms used throughout this manual and gives their individual meanings.

The following information to aid you in understanding and trouble-shooting JES2 is available on microfiche:

MVS/ESA Information	**Microfiche Order Number**
Data Areas	LYB8-1850

Where to Find More Information

This book references other publications for further details about specific topics. The following table lists these publications, the abbreviated forms of their titles used throughout this book, and their order numbers.

Short Title Used in This Book	Title	Order Number
JES2 Commands	*MVS/ESA Operations: JES2 Commands*	SC28-1039
JES2 Initialization and Tuning	*MVS/ESA System Programming Library: JES2 Initialization and Tuning*	SC28-1038
JES2 Customization	*MVS/ESA System Programming Library: JES2 Customization*	LY28-1010
JES2 Messages	*MVS/ESA Message Library: JES2 Messages*	SC28-1040
System Management Facilities (SMF)	*MVS/ESA System Programming Library: System Management Facilities*	GC28-1819
VSAM Programmer's Guide	*MVS/ESA VSAM Programmer's Guide*	GC26-4015
Assembler Reference	*Disk System Basic Assembler Reference Manual*	SC21-7509
IBM 3800 Printer Guide	*IBM 3800 Printing Subsystem* Programmer's Guide	GC26-3846
Remote Workstation Generation	*MVS/ESA Remote Workstation Generation*	GC28-1842
System Messages	*MVS/ESA Message Library: System Messages Volume 1 and 2*	GC28-1812 and GC28-1813

Do You Have Problems, Comments, or Suggestions?

Your suggestions and ideas can contribute to the quality and the usability of this book. If you have problems using this book, or suggestions for improving it, complete and mail the Reader's Comment Form found at the back of this book.

Summary of Changes

Summary of Changes
for LY28-1006-2

A JES2 multi-access spool complex must install this release of JES2 on **all** members of the complex if it is installed on any member. All prior releases of MVS/System Product JES2 Version 1 (MVS/370), Version 2 (MVS/XA), and MVS/System Product Version 3 (MVS/ESA) are incompatible with MVS/System Product JES2 Version 3 Release 1.3 in a multi-access spool environment.

This major revision, which supports MVS/System Product Version 3 Release 1.3 - JES2 (5685-001), reflects the following product enhancements:

- Protection for, and auditing of, access to SYSIN and SYSOUT data sets provided by security authorization facility (SAF) calls.
- User verification and authority checking for jobs, commands, and SYSOUT received from other nodes or remote workstations.
- Security information about users and data sets passed to the print services facility (PSF).
- Security information added to network records and/or security information created for jobs/data sets entering a node.
- Command acceptance, rejection, and access control; auditing of these events.
- User validation processing is changed from the scheduler call to a SAF RACROUTE call.
- Spool access is protected by new SAF calls in JES2 and by security level assignments restricting the use of TSO commands that permit spool access.
- Subtask environment enhancements, such as the added general subtasking facility.
- Macro and control block changes.
- Printer FCB selection processing enhancements.

  Note: The data area DCT.DCTDFCB in the JES2 $DCT control block is renamed to DCT.DCTNIFCB (for the non-impact printer default FCB value). A new data area, DCT.DCTDDFCB, exists for the device default FCB value.

Summary of Changes for LY28-1006-1

This major revision, which supports MVS/System Product Version 3 Release 1.1 - JES2 (5685-001), reflects the following product enhancements:

- Restructure of HASPSSSM
- Support for up to 32,767 jobs in the system
- Support for SWA for JES2-managed data to reside above 16 megabytes in virtual storage
- Support for continuous system operation
- Changes in warm start processing
- Support for reusable spin IOTs
- Diagnosis support for routines shipped without source code
- Tracing enhancements
- All information relating to remote workstation generation is now contained in *Remote Workstation Generation* (GC28-1842).

Chapter 1. Introduction

The job entry subsystem 2 (JES2) serves as the entry point for jobs in MVS. JES2 is responsible for reading jobs into the system, scheduling the jobs, executing them, and handling their output from the system.

Configuration

JES2 performs its functions within the following environments, as illustrated in Figure 1-1.

1. A **single JES2 configuration** consists of one MVS/JES2 system supporting an installation's jobs that are located on the spool.
2. A **multi-access spool configuration** (also called shared spool) consists of from 2-7 MVS/JES2 systems supporting an installation's jobs that are located on the shared spool.
3. A **network job entry (NJE) configuration** consists of from 2-1000 single-access or multi-access spool configurations (called nodes) joined together via communications lines. All MVS/JES2 nodes transmit among themselves jobs, messages, and SYSOUT data sets.

The JES2 program logic that is described in this manual covers all the functional capability of the NJE configuration, which is a superset of the functions in the other two configurations.

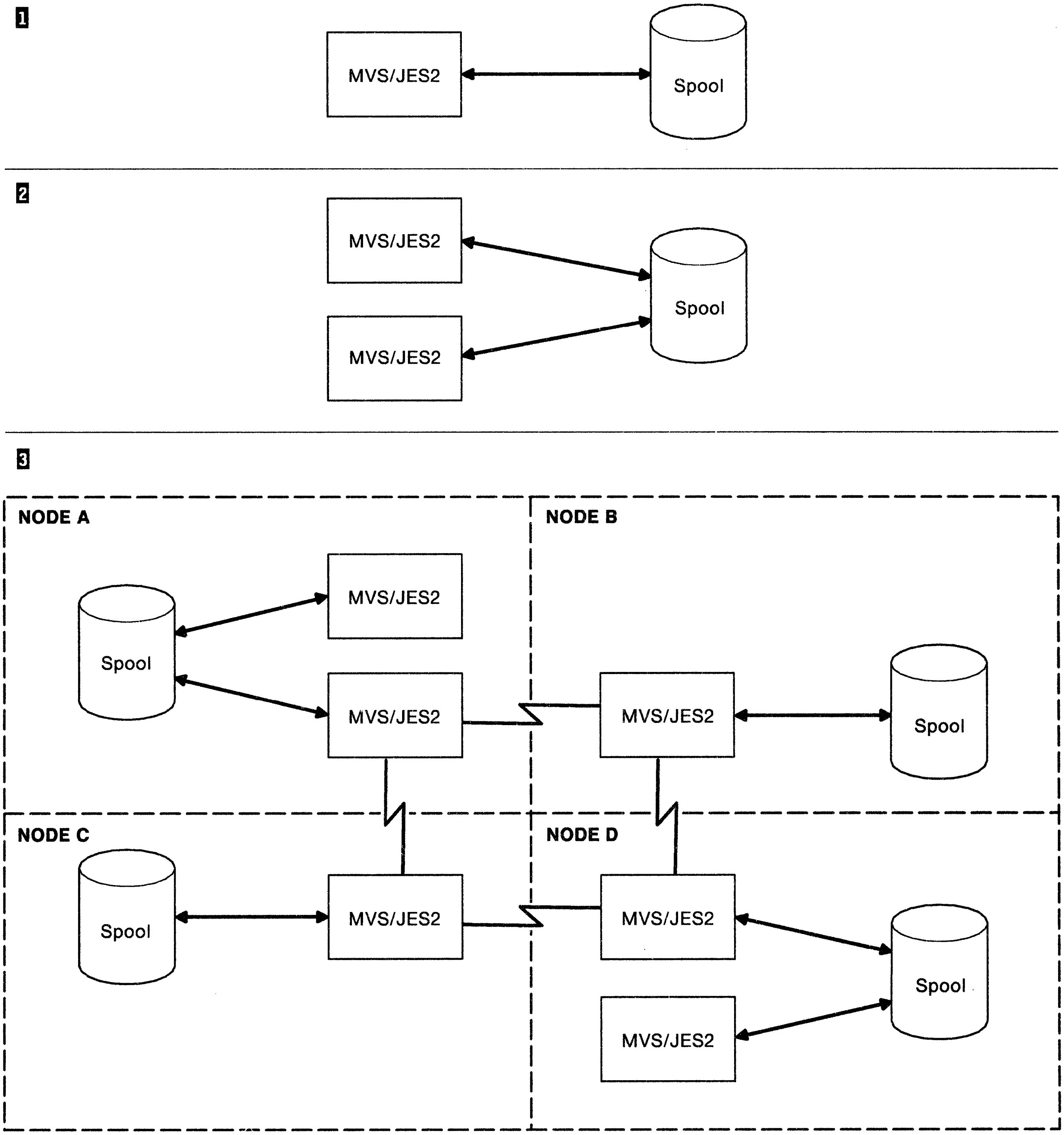


Figure 1-1. JES2 Configurations

Environment

The MVS/JES2 system, isolated in Figure 1-2, consists of the following pieces:

- **JES2** - that set of program services that perform JES2 functions and belong to the JES2 component. These programs read jobs into the system, place them on the spool, supply them to MVS for execution, and process their output following execution.
- **MVS** - that set of program services that perform supervisory functions and belong to the MVS control program. Examples of these programs are the interrupt handlers, initiators, contents supervisor, paging supervisor, and others.
- **SSI** - the subsystem interface (SSI) through which JES2 and MVS communicate. The SSI consists of service routines that run in common storage. Users running in other address spaces use the IEFSSREQ macro to invoke these SSI service routines to ultimately communicate with JES2.
- **Spool** - that collection of direct access volumes that contain the jobs that JES2 processes and the various job-related control blocks, such as the job control table (JCT) and input/output table (IOT).
- **Checkpoint data set** - that data set that contains information to allow communication among the processors of a multi-access spool complex; this data set is also used to restart JES2.

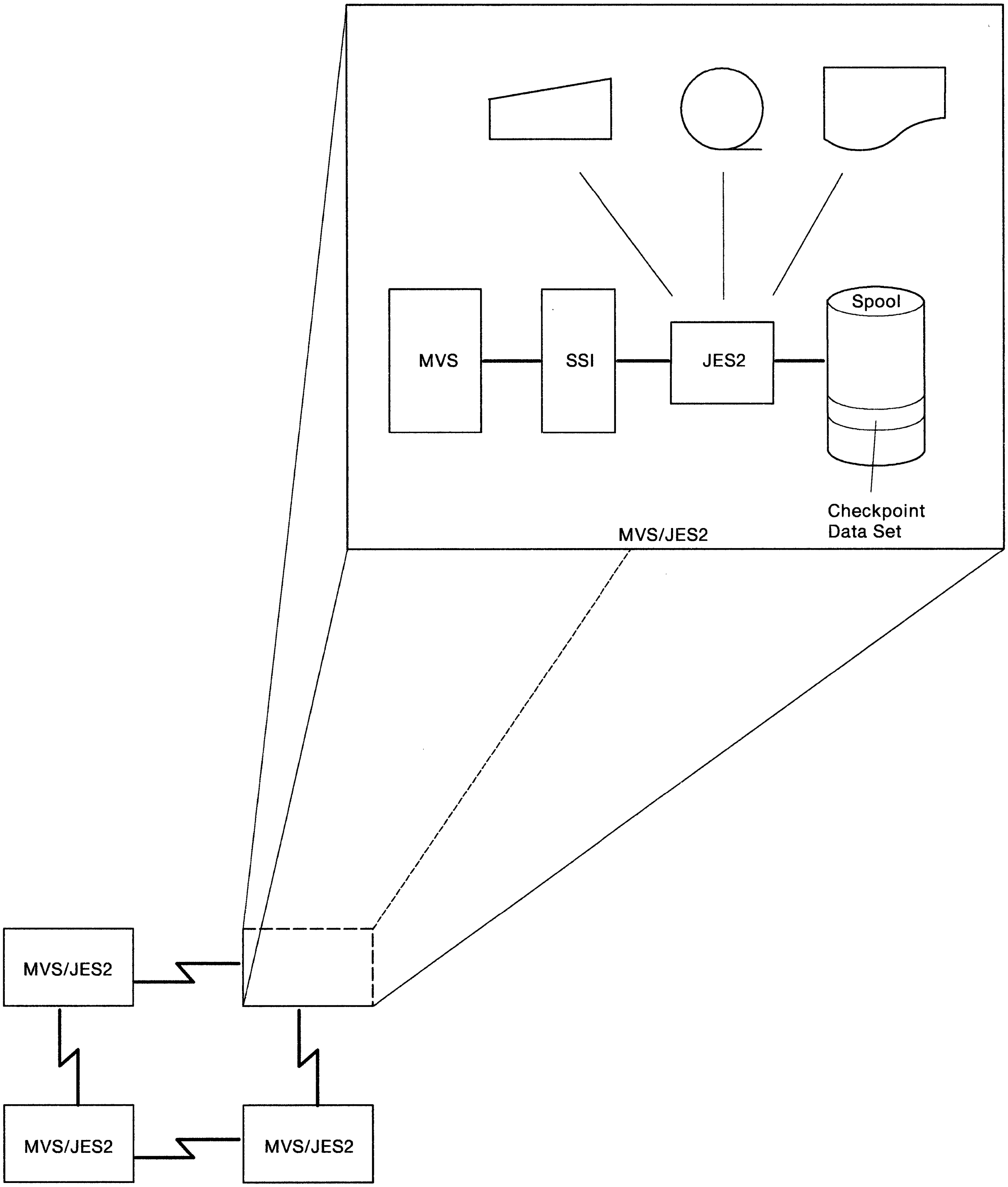


Figure 1-2. JES2 Environment

JES2 Structure

JES2 operates in its own address space within the system. The JES2 address space is created when the system is initialized and JES2 is started. The four JES2 load modules are HASPSSSM, HASPINIT, HASPFSSM and HASJES20. HASPSSSM may reside either in the link pack area (LPA) or be directly loaded into common storage. HASJES20 resides in the private area of the JES2 address space. The HASPINIT load module is loaded into the JES2 private area during initialization and deleted at the end of the JES2 initialization process. Figure 1-3 illustrates the placement of JES2 code and control blocks in storage for a typical MVS/JES2 system after it has been initialized.

1. MVS modules that require the support of JES2 use a subsystem option block (SSOB) and a subsystem identification block (SSIB) to request a particular function of JES2 and then issue the IEFSSREQ macro instruction.

2. MVS SSI support routines get control as a result of the IEFSSREQ macro and index into the subsystem vector table (SSVT) for the entry point into HASPSSSM that performs the requested function. HASPSSSM then gets control at that entry point; the SSVT is created when JES2 is initialized and is located in subpool 228 of the common service area (CSA).

3. HASPSSSM performs the requested function, if necessary communicating with HASJES20 via a cross memory post to complete the requested function. HASJES20 completes its processing utilizing various JES2 control blocks and then cross memory posts HASPSSSM. HASPSSSM then returns to the issuer of IEFSSREQ.

4. HASPFSSM is loaded into the functional subsystem (FSS) address space. HASJES20 communicates with HASPFSSM in support of the functional subsystem interface (FSI). The FSI exists between JES2 and the FSS to provide control of and data set services for the FSS.

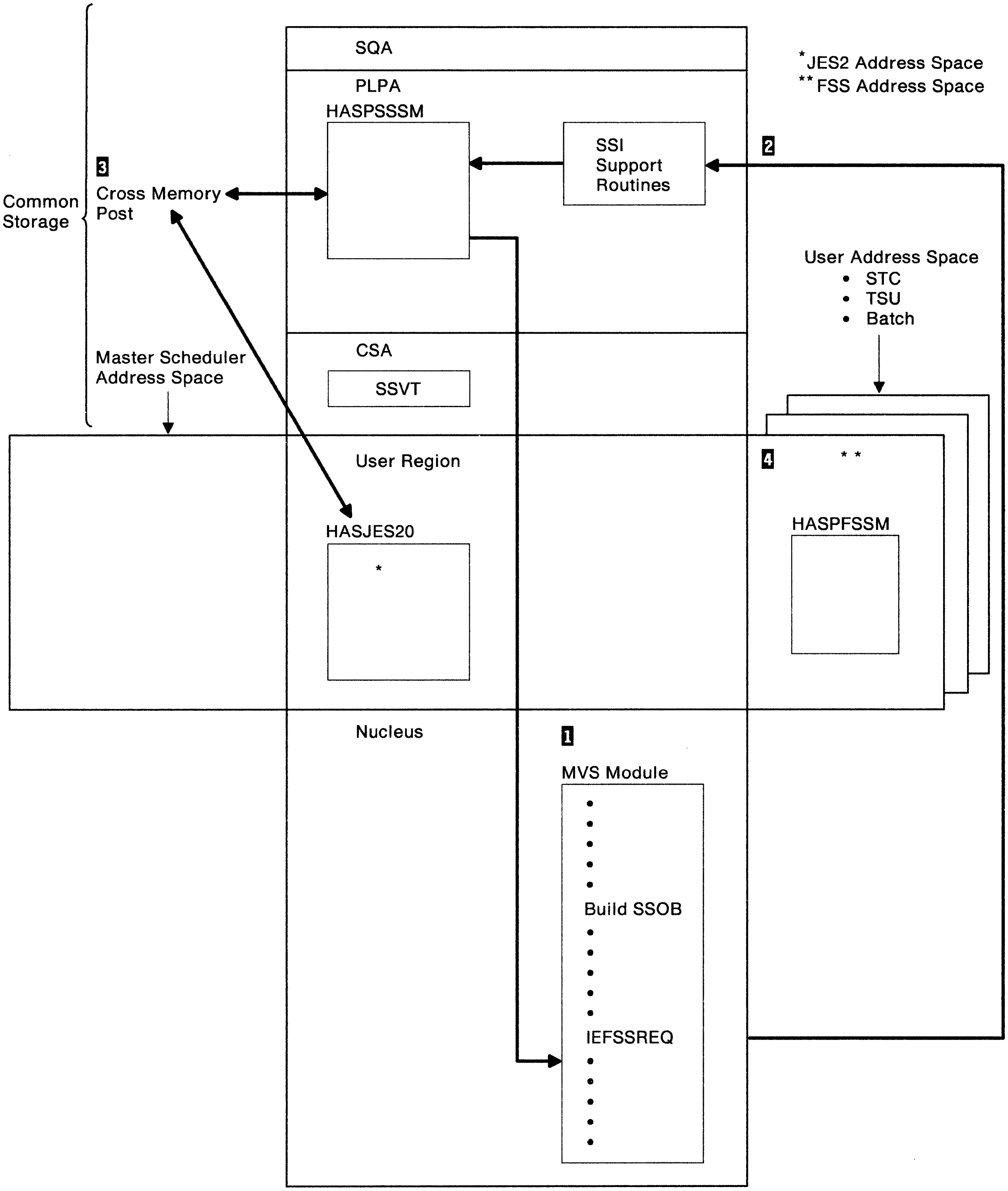


Figure 1-3. JES2 Structure

HASPSSSM

MVS interfaces directly with HASPSSSM through the SSI to provide job scheduling, data management (SYSIN and SYSOUT), functional subsystem connect and disconnect, and other subsystem functions and operator communication functions. (See the Subsystem Interface section of this chapter for more information.) These functions are incorporated into HASPSSSM as function routines that are invoked through the use of addresses in the SSVT as shown in Figure 1-5.

1. The SSI support routines get control after MVS issues the IEFSSREQ macro instruction. The SSOB is passed as input. In the SSOB is a function code that identifies the function of JES2 that MVS wants to execute. The SSI support routines subtract one from the function code and add to the result the address of the function code matrix in the SSVT (label SSVTFCOD, mapped by the MVS macro, IEFJSSVT). The result of this addition is a specific byte address in the function code matrix.
2. The specific byte (or index) contains a zero or a nonzero value. If the value is zero, the function requested is not supported by JES2. A nonzero value in this byte indicates that the function requested is supported by JES2. In this case, the SSI support routines subtract one from this byte value, multiply the result by 4, and add the result of the multiplication to the address of the JES2 SSVT pointer area (label SSVTFRTN, mapped by the MVS macro, IEFJSSVT.) The resulting address points to a fullword, containing the address of the function routine in HASPSSSM that is to get control to support the requested JES2 function. (Appendix C lists the specific contents of the SSVT, identifying the various function routines in HASPSSSM that can get control.)

Note that JES2 maps the function code matrix with the $SVTC macro and the function routine section with the $SENTRYS macro.

The HASPSSSM load module is made up of the source modules listed in Figure 1-4.

Figure 1-4. HASPSSSM Source Modules

HASPSSSM Source Modules

Source Module	**Function**
HASCDSAL	Data set allocation/unallocation services
HASCDSOC	Data set open/close services
HASCDSS	Data space services
HASCJBST	Job select/termination services
HASCJBTR	Job end-of-task/end-of-memory services
HASCLINK	Nucleus of the common storage routines
HASCSRIC	Services and control subroutines
HASCSIRQ	Miscellaneous SSI routines
HASCSRDS	Data-set-related subroutines
HASCSRJB	Job-related subroutines
HASPAM	Spool data set access method

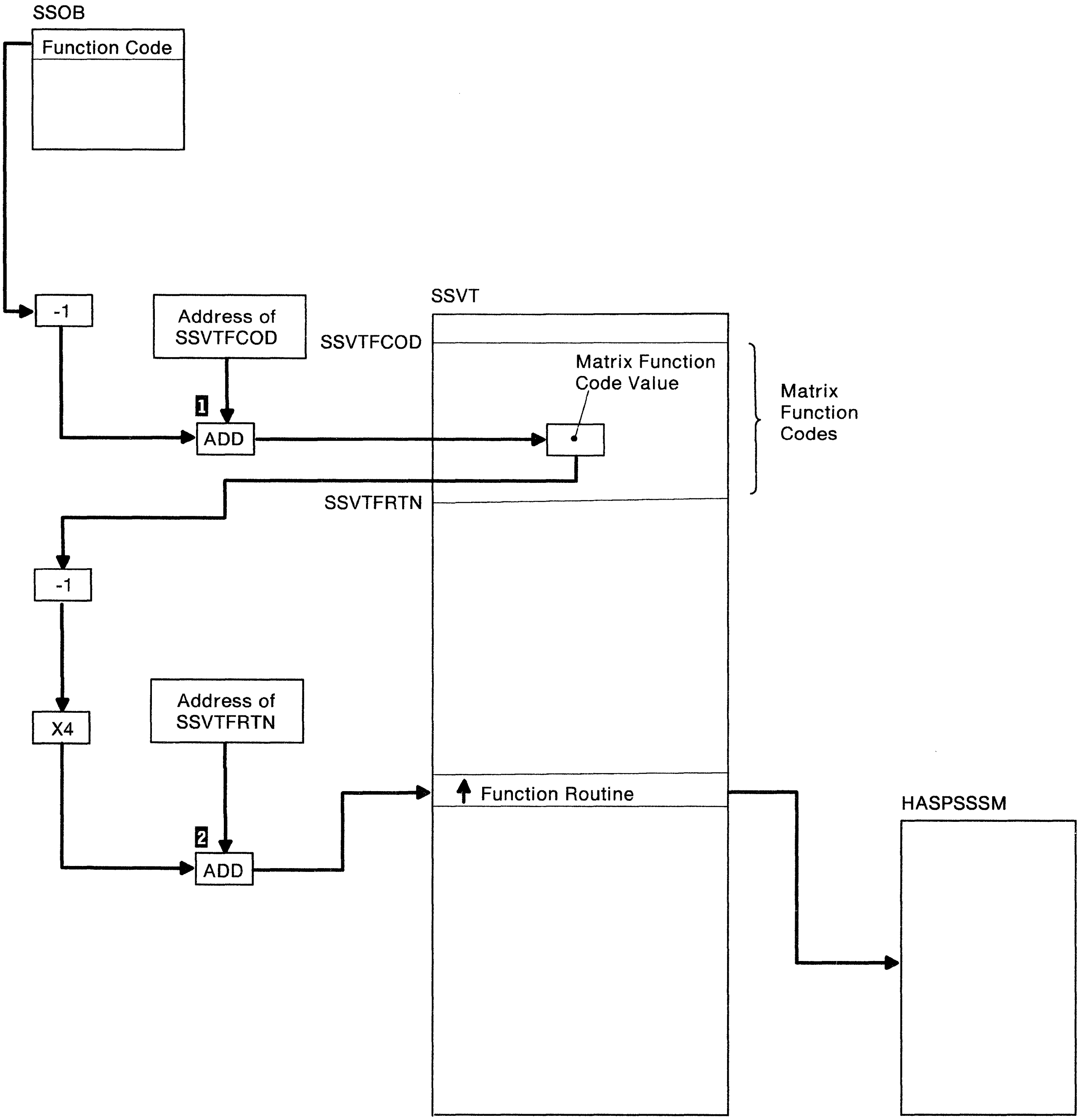


Figure 1-5. JES2 Function Routine Addressing

HASPINIT

The HASPINIT load module is made up of the source modules illustrated in Figure 1-6. HASPINIT resides in the JES2 private area during JES2 initialization. The initialization routine administrator (HASPIRA) is entered first, performs some preliminary initialization and invokes the other JES2 initialization modules. When the other JES2 initialization modules (HASPIRMA, HASPIRPL, HASPIRDA, and HASPIRRE) have been called, HASPIRA completes the initialization.

Figure 1-6. HASPINIT Source Modules

HASPINIT Source Modules

Source Module	**Description**
HASPIRA	Initialization routine administrator
HASPIRDA	DASD (checkpoint/spool) initialization
HASPIRMA	Miscellaneous allocations and initializations
HASPIRPL	Parameter initialization
HASPIRRE	RJE/NJE initialization

HASJES20

The HASJES20 module is made up of the source modules illustrated in Figure 1-7. These modules perform JES2 main task and subtask processing. The JES2 **main task** provides the basic functions of reading and spooling job input, converting JCL, selecting jobs for execution from the JES2 job queue, spooling and writing job output, and purging jobs — all accomplished with a set of programs called **processors**. These processors use numerous subroutines called **control service routines**. The JES2 main task controls the activation and deactivation of these processors through a dispatcher. Figure 1-7 also illustrates that HASJES20 is made up of subtasks. These subtasks provide services to the JES2 main task; these services may involve MVS waits, and as such, are performed in subtasks, because the JES2 main task is not permitted to enter an MVS wait state until all currently outstanding JES2 functions are completed.

The HASP communication table (HCT), which is found in HASPNUC, is the global directory for HASJES20. The HCT contains the address of the JES2 module map ($HASPMAP which resides in HASPTABS), which is initialized with a symbolic name and address for each of the JES2 modules that make up HASJES20 and HASPSSSM. (Figure 1-9 illustrates an initialized HCT residing in HASPNUC and HASPINIT.)

HASPFSSM

HASPFSSM is made up of a single source module. It is loaded into the private area of the functional subsystem (FSS) address space during FSS connect processing. HASPFSSM contains service routines that support the functional subsystem interface (FSI).

Figure 1-7 (Page 1 of 3). HASJES20 Source Modules and Task Structure

Source Module and Function‡	Subtask	Subtask Description
HASPNUC (Main Task) * HCT * MODCHECK * Dispatcher * I/O Supervisor * Service Routines		
* Dynamic Allocation/ Deallocation Subtask	HOSALLOC	Dynamic Allocation/Deallocation subtask (HOSALLOC) — uses SVC 99 to dynamically allocate a device to JES2 or to dynamically deallocate a JES2 device.
HASPSTAB * Scan Tables HASPSCAN * $SCAN Facility HASPSXIT * $SCAN Facility Table Exits HASPMSG * Message Building Routines HASPTABS * JES2 Tables HASPRAS * Error Service Routines HASPTERM * Recovery/Termination Services HASPNPM * Network Path Manager * Network Services HASPNET * Job Transmitter Processor * SYSOUT Receiver Processor * SYSOUT Transmitter Processor HASPCON * Service Routines		
* Communication Subtask	HASPWTO	Communications subtask (HASPWTO) — issues SVC 34 and SVC 35 for the main task when operator communications take place.
HASPDYN * DCT, DTE, PCE build HASPTRAK * Spool Track Management HASPCOMM * Command Processor HASPSERV * Command Services and Work Selection Modify Service * $L Support HASPRDR * Input Service Processor HASPCNVT * Conversion Processor HASPCNVS		
* Conversion Subtask	HOSCNVT	Conversion subtask (HOSCNVT) — links to the MVS converter to convert a job's JCL to converter/interpreter text.

‡ Unless otherwise indicated as a subtask, the functions itemized in this column are performed as part of the JES2 main task.

* The JES2 dispatcher schedules and dispatches various processors under the single TCB of the JES2 main task. Because the processing that runs under the JES main task is not permitted to enter a wait state, such processing is isolated as subtasks under the JES2 main task.

Figure 1-7 (Page 2 of 3). HASJES20 Source Modules and Task Structure

Source Module and Function‡	Subtask	Subtask Description
HASPXEQ * Execution Processor		
HASPPSO * Process SYSOUT Processor		
HASPPRPU * Print/Punch Processor		
* Image Loading Subtask	HASPIMAG	Image loader subtask (HASPIMAG) — loads the universal character set (UCS) and forms control buffer (FCB) images for processing job output.
HASPSTAC * STATUS/CANCEL Processor		
HASPSSRV * Subtask Services		
HASPSUBS * General Subtask	HA$PSUBS	Initialize subtask; remove and process work from $STWORK.
* ESTAE routines for General Subtask		
HASPHOPE * Output Processor		
HASPFSSP * Functional Subsystem Service Processor		
HASPJOS * Job Output Services		
HASPRTAM * Line Manager Processor (BSC and SNA) * Remote Console Processor		
HASPBSC * BSC Service Routine		
HASPSNA * SNA Service Routines		
* VTAM Subtask (ACB OPEN, CLOSE)	HASPVTAM	SNA subtask (HASPVTAM) — initializes JES2's use of the VTAM and OPEN ACB interface. OPEN/CLOSE processing.
* API Routines		
HASPCKPT * Checkpoint Processor		
HASPCKDS * Checkpoint Dialog and Service Routines	HASPCKAP	The HASP checkpoint application subtask provides for a third copy of the checkpoint so that authorized application programs (such as SDSF) can use it.
HASPWARM * Warm Start Processor		
HASPMISC * Resource Management Processor * Priority Aging Processor		
* SMF Accounting Subtask	HASPACCT	System management facility (SMF) subtask (HASPACCT) — issues SVC 83 to write accounting records for the main task.

‡ Unless otherwise indicated as a subtask, the functions itemized in this column are performed as part of the JES2 main task.

* The JES2 dispatcher schedules and dispatches various processors under the single TCB of the JES2 main task. Because the processing that runs under the JES main task is not permitted to enter a wait state, such processing is isolated as subtasks under the JES2 main task.

Figure 1-7 (Page 3 of 3). HASJES20 Source Modules and Task Structure

Source Module and Function‡	Subtask	Subtask Description
* Network Accounting Routines * Time Excess Processor HASPEVTL * Event Trace Log Processor		
HASPSPOL * Spool Manager Processor * Dynamic Spool Allocation Subtask	HOSPOOL	Dynamic spool allocation subtask (HOSPOOL) — initializes JES2's spool data sets (SYS1.HASPACE) used throughout JES2 processing.
HASPSTAM * Spool Offload Processor * Spool Offload I/O Manager Subtask	HASPOFF	Spool offload I/O manager subtask (HASPOFF) — performs allocation, open and close, and deallocation services for the spool offload data set.

‡ Unless otherwise indicated as a subtask, the functions itemized in this column are performed as part of the JES2 main task.

* The JES2 dispatcher schedules and dispatches various processors under the single TCB of the JES2 main task. Because the processing that runs under the JES main task is not permitted to enter a wait state, such processing is isolated as subtasks under the JES2 main task.

The JES2 dispatcher uses a queue of processor control elements (PCEs) to control the dispatching of processors. When a processor is eligible for dispatching, its PCE is on a dispatcher queue called the **$READY queue**. The dispatcher dispatches processors from this queue in FIFO order. The $READY queue can be in the following states, as illustrated in Figure 1-8:

1. **Active Processor PCE:** The currently active processor's PCE (PCE2) is addressed by the $CURPCE field of the HCT in the JES2 address space.

2. **Empty $READY Queue:** No PCEs are ready to be dispatched and the $READY queue is empty.

3. **Waiting Processing PCE:** When a processor is waiting on an event, it is ineligible for dispatching; its PCE is on one of several wait queues depending on what the processor is waiting for. If the processor is waiting for a resource, its PCE is chained to the designated resource wait queue; if the processor is waiting for a specific event, its PCE is queued to itself.

In addition to waiting for a JES2 event or resource, a processor might be waiting for an MVS post of an event control block (ECB). The address of this PCE and the address of the specific PCE are contained in the extended ECB (XECB). Posted XECBs are stacked off the $EXTECBQ queue. A processor is dispatched when the ECB is posted or when the event or resource is released.

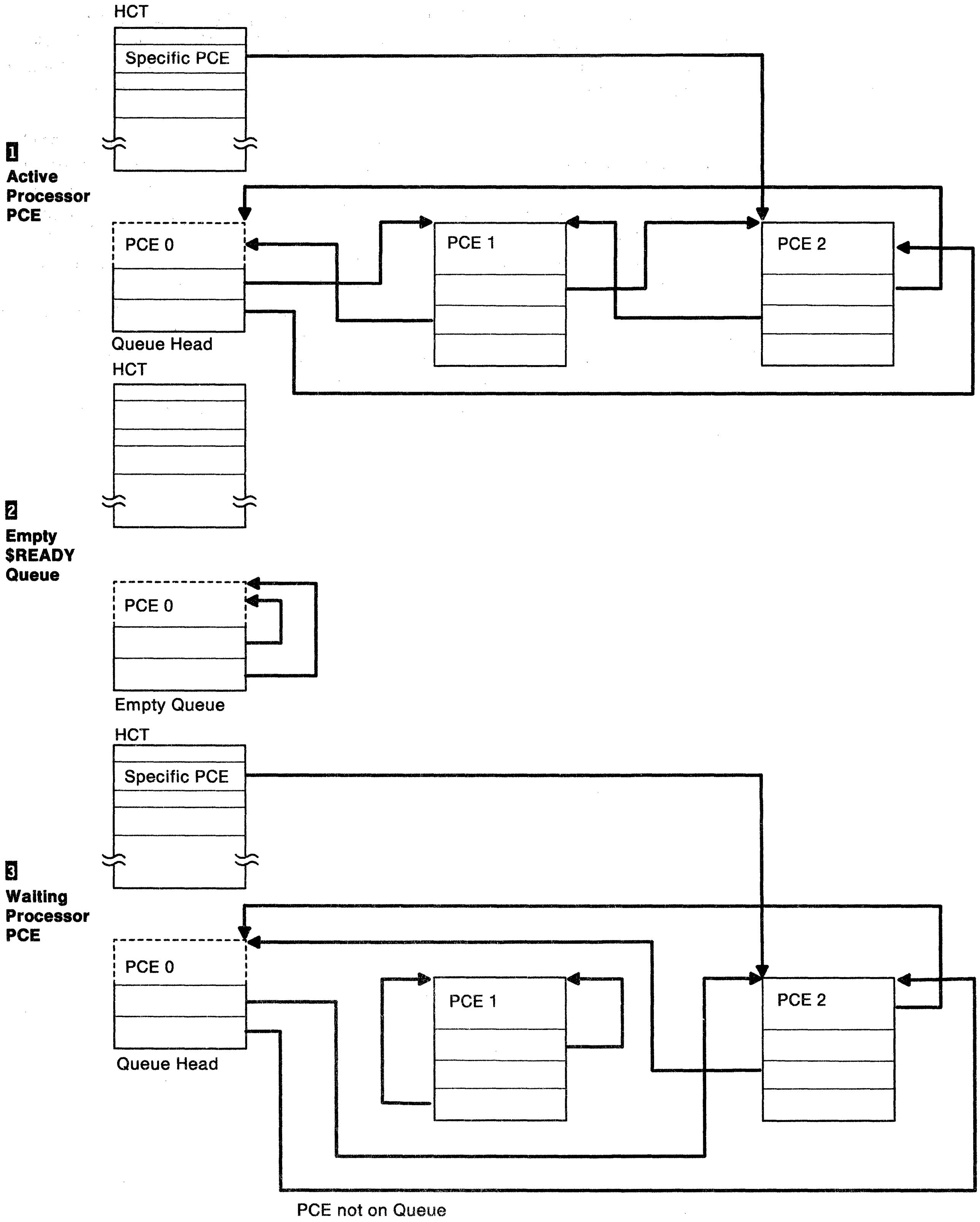


Figure 1-8. JES2 Processor Control Element Relationships

The dispatcher, running under the JES2 main task, is responsible for giving control to individual JES2 processors. When a JES2 processor gets control it runs, performing its function, until it relinquishes control via the $WAIT macro. The $WAIT macro causes control to be passed to the dispatcher, which gives control to the next available and ready JES2 processor.

Processors become ready when they are posted. This post can be a specific post, signaling the completion of an event, or a general resource post, indicating that a resource is available. Posting occurs within the JES2 main task (via $POST) or from a JES2 subtask or other address space (via $$POST). JES2 subtasks and programs (running in other user address spaces) use the $$POST macro to inform JES2 of particular events. The $$POST macro causes a HASPSSSM interface routine to cross memory post the JES2 main task. The $$POST promulgation routine (HASPSPEC), which receives control when the resource posting routine runs out of work, propagates event posts from the CCTECF field of the $HCCT (used by HASPSSSM interface routines) to the HCT master event control field ($HASPECF) used by HASJES20. The resource posting routine marks the corresponding HASJES20 processors eligible for dispatching. Figure 1-9 illustrates the HCT and the control information JES2 uses during its processing.

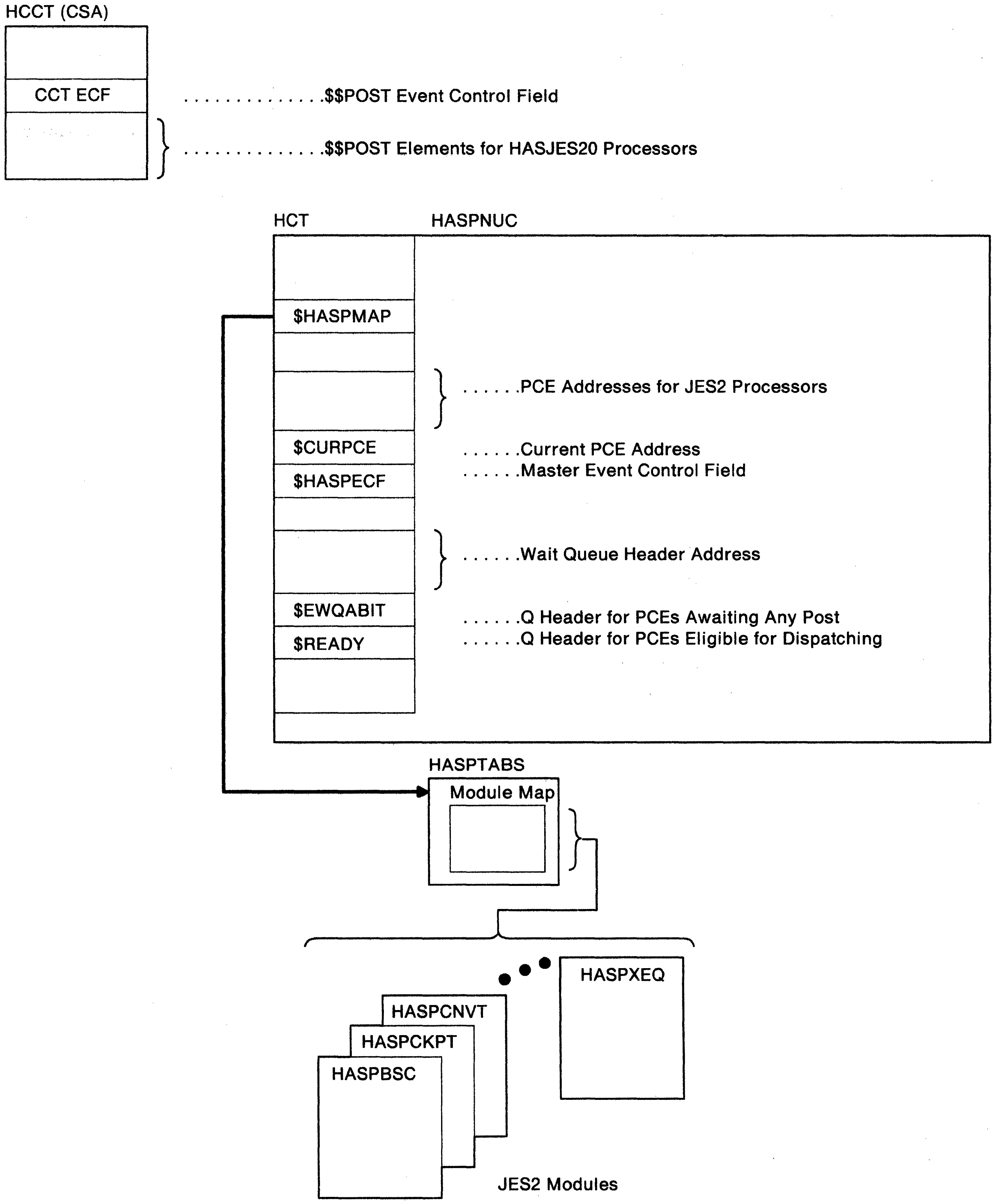

HCCT (CSA)
CCT ECF
..............$$POST Event Control Field
..............$$POST Elements for HASJES20 Processors
HCT
HASPNUC
$HASPMAP
......PCE Addresses for JES2 Processors
$CURPCE
......Current PCE Address
$HASPECF
......Master Event Control Field
......Wait Queue Header Address
$EWQABIT
......Q Header for PCEs Awaiting Any Post
$READY
......Q Header for PCEs Eligible for Dispatching
HASPTABS
Module Map
HASPXEQ
HASPCNVT
HASPCKPT
HASPBSC
JES2 Modules

Figure 1-9. JES2 Event Control and Module Directory

Subsystem Interface

The subsystem interface (SSI) and the specific control blocks used by the SSI support routines are shown in Figure 1-10.

1. The module using the SSI builds a subsystem options block (SSOB), requesting a particular function of JES2 and sets register 1 to point to a word that points to the SSOB.
2. The module issues the IEFSSREQ macro instruction. SSI support routines get control and validate the subsystem request, initialize a subsystem information block (SSIB), and index into the SSVT for the entry point in HASPSSSM that processes the requested function. The SSI support routines then pass control to HASPSSSM at this entry point.
3. HASPSSSM performs the requested function, if necessary communicating with HASJES20 via a cross memory post to complete the requested function. HASPSSSM then returns to the issuer of the IEFSSREQ macro.

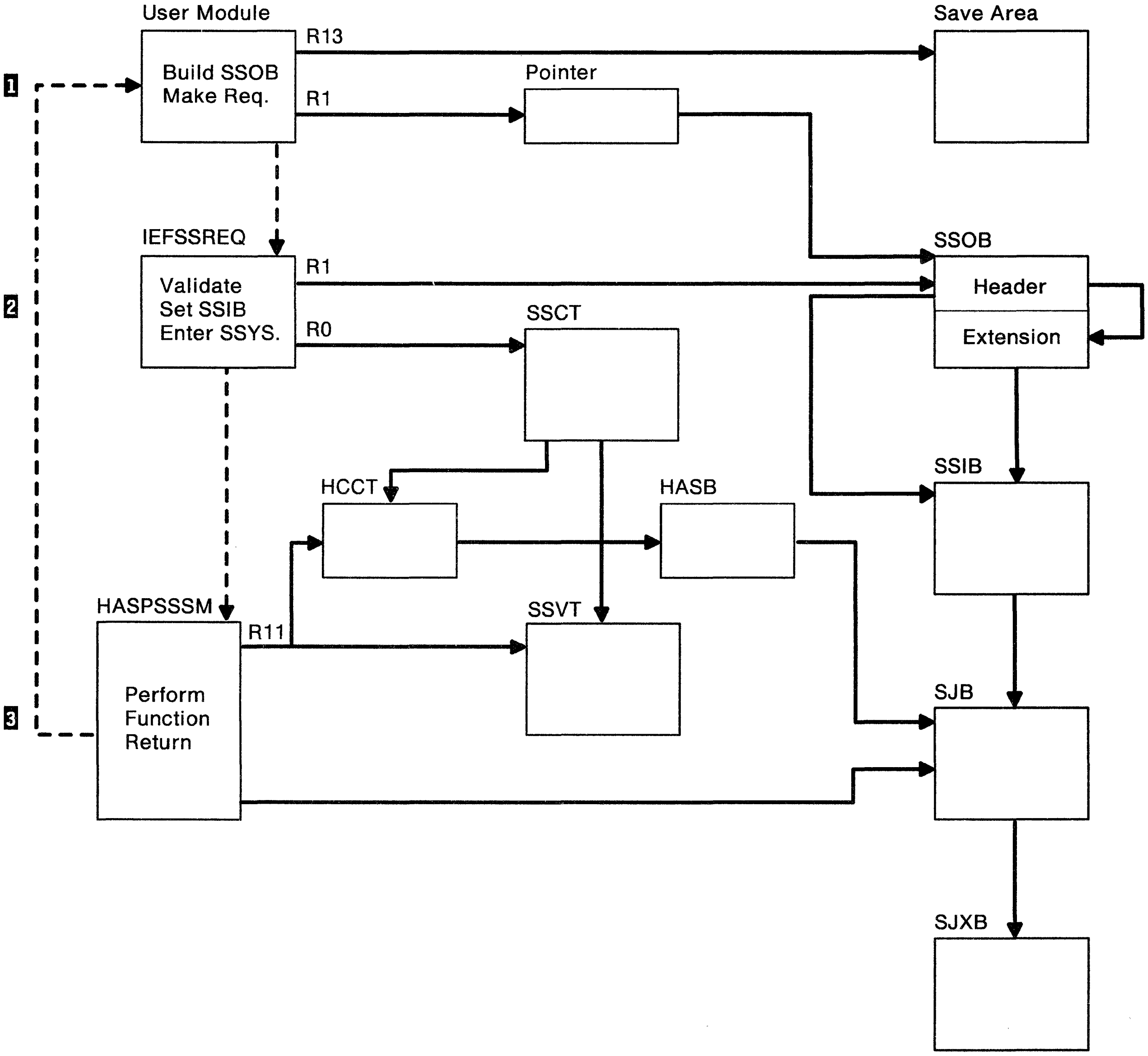


Figure 1-10. Subsystem Interface Control Blocks

JES2 subsystem support routines are invoked, using the SSI, for the following reasons:

- Scheduling and controlling jobs
  - Job selection
  - Job termination
  - Requeue job
  - Request job identification
  - Return job identification
  - End of address space
  - End of task
  - Checkpoint/Restart
- Accessing SYSIN/SYSOUT and process SYSOUT data sets
  - Allocation
  - Open
  - Restart
  - Close
  - Unallocation
- Communicating with interactive programs and external writers
  - Process SYSOUT
  - User id validity checks
  - CANCEL
  - STATUS
- Communicating with the operator
  - Command processing (SVC 34)
  - Write to operator (SVC 35)
- Communicating with functional subsystems
  - Connect and disconnect processing

Figure 1-11 illustrates a common example of SSI usage in processing a job. In this figure, various programs invoke the SSI to process the job.

1. MVS device allocation uses the SSI to get JES2 to allocate directly or dynamically the SYSIN and SYSOUT data sets specified by the JCL in the job.
2. After the job is attached as a task to the initiator, the job contends with other jobs in the system for a chance to execute. When the job is finally dispatched, the job runs, opens the input data set and output data set via the SSI and performs its I/O operations (reading input data from the input data set and writing output data to the output data set). The job eventually closes its SYSIN and SYSOUT data sets (via the SSI). HASPSSSM then either spins the SYSOUT data sets for immediate output processing or allows the SYSOUT data sets to be eventually deallocated when the job terminates. (Ultimately, JES2 makes the SYSOUT data sets available for output processing.)
3. The data sets allocated to the job for SYSIN and SYSOUT are deallocated. MVS deallocation uses the SSI to unallocate the job's SYSIN and SYSOUT data sets. Output processing for non-spun data sets are processed by JES2 after job termination.
4. When the job terminates, the MVS program that processes a terminating job uses the SSI to let JES2 terminate the job from its execution queue and place output from the job on the output queue.

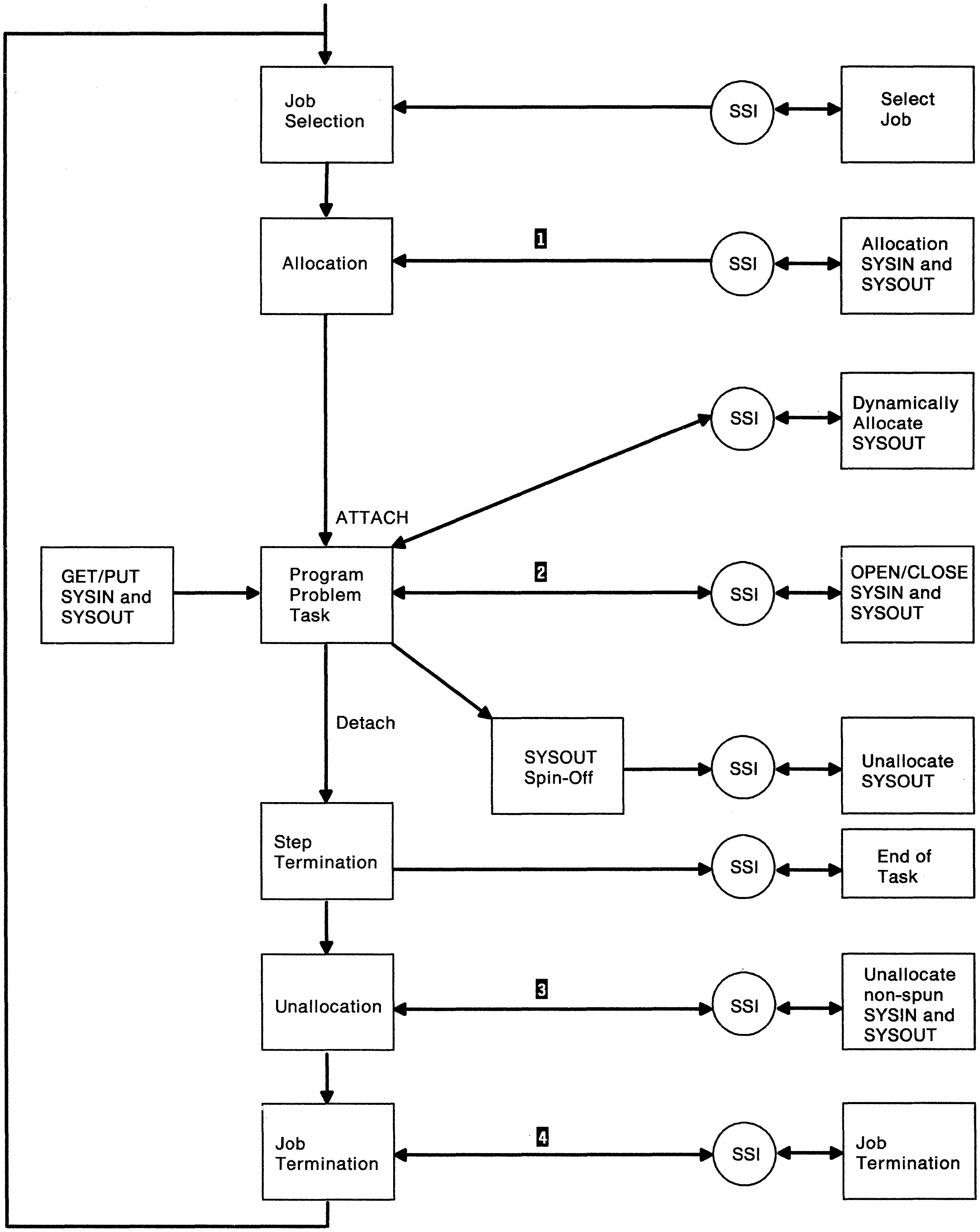


Figure 1-11. Using the Subsystem Interface

JES2 Initialization

Initialization is the means by which JES2 readies itself to process work. As illustrated in Figure 1-12, JES2 initialization is the process that takes place between the START JES2 command and the processing of jobs.

The initialization process obtains and processes JES2 initialization options and parameters. Using these parameters and options, the JES2 initialization process initializes various control blocks and data areas, defines the JES2 spool environment (for both SYS1.HASPACE and SYS1.HASPCKPT), and initializes the unit record and remote environments, as well as attaching the JES2 subtasks. Throughout the initialization process, JES2 records the progress of initialization and the communication that takes place with the operator.

When the START JES2 command is issued, JES2 receives control at the initial entry point at the symbol HASP in the source module, HASPNUC.

HASP ensures that JES2 is running in supervisor state and key 1, and switches to AMODE 31. It determines if HASPINIT is part of the HASJES20 load module; if it is not, HASP loads HASPINIT. HASP invokes the initialization routine administrator (HASPIRA) to perform the JES2 initialization. HASPIRA establishes the JES2 ESTAE to handle any JES2 main task abends that might occur. After HASPIRA has called all initialization routines or when an initialization routine indicates that JES2 is to terminate, HASPIRA returns to HASP.

When HASPIRA returns, HASP deletes HASPINIT (if HASP loaded it) and dispatches JES2 to process jobs provided that the initialization was successful. If HASPINIT indicates that an initialization error occurred, HASP terminates JES2.

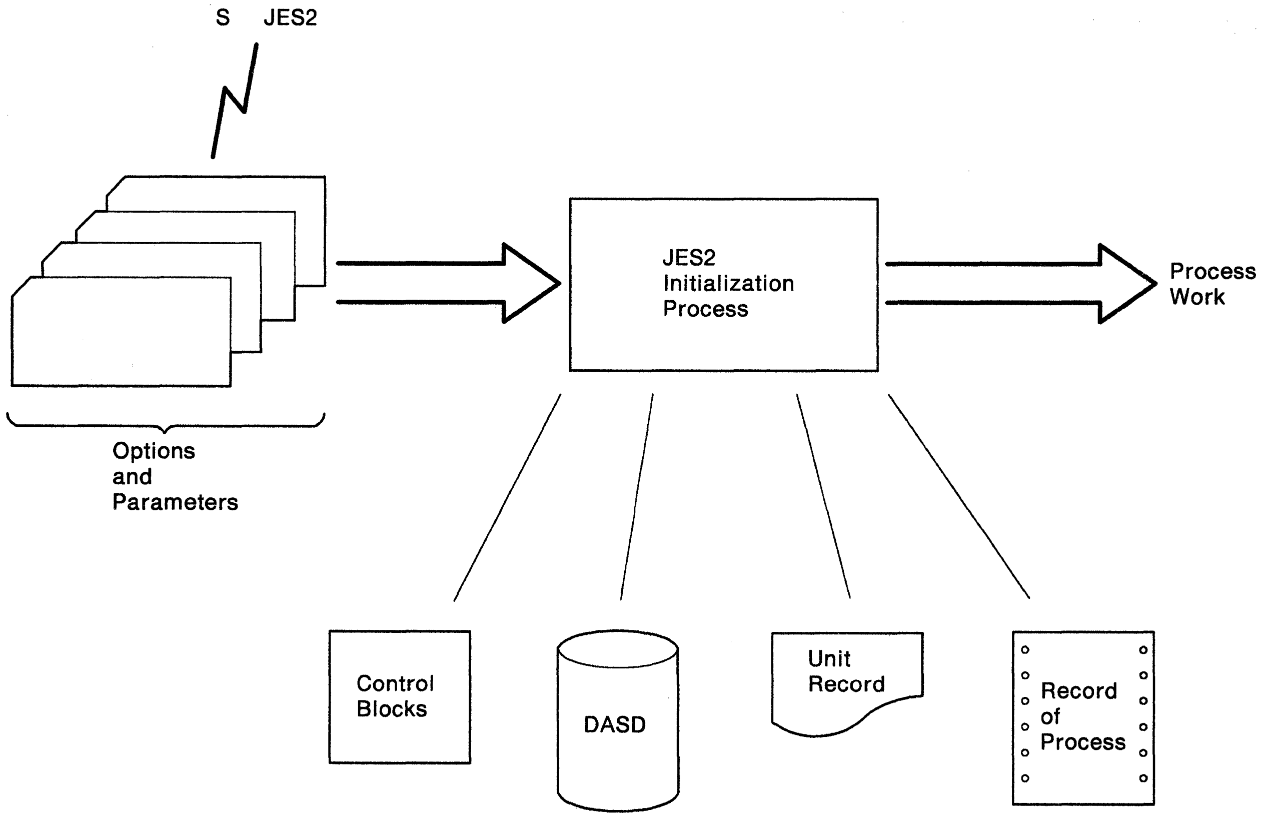


Figure 1-12. JES2 Initialization

Functional Subsystem Concepts

JES2 uses a functional subsystem (FSS) to provide certain functions that it normally provides in its own address space (for example, device-dependent functions related to printers that print page-mode data, such as the 3800 Printing Subsystem Model 3); these functions operate in their own address space.

As an extension to JES2 processing, functional subsystem (FSS) processing takes place in an individual address space separate from the JES2 address space. Specific functional processing (for example, device processing) that JES2 would normally perform within its own address space can be accomplished by the FSS or a number of FSSs; the FSS address space isolates this processing from JES2. JES2 communicates directly (using orders) with the functional subsystem (that is, the programs in the FSS address space) to perform the specific functional processing. JES2 is thus insulated from the specifics and the possible problems related to the processing associated with the functional subsystem (for example, the channel programs that drive a particular printer).

The functional subsystem relies on JES2 for its system image. That is:

- The installation specifies functional subsystem initialization as part of JES2 initialization.
- JES2 manages operator communication.
- JES2 controls its own resources, such as devices and the spool, even though they are used by the functional subsystem.
- JES2 controls input and output scheduling.
- JES2 coordinates the termination and restart of itself and the functional subsystem.

The implementation structure for the JES2 functional subsystem is illustrated in Figure 1-13.

1. A separate functional subsystem address space is associated with each FSS, and is under control of the MVS dispatching structure. More than one functional subsystem address space can be defined in the system, and JES2 communicates with each one of them.
2. JES2 and each functional subsystem communicate with each other through the functional subsystem interface (FSI). The FSI consists of a collection of service routines that JES2 and the functional subsystem employ. To effect operator control over the FSS, JES2 issues ***orders*** to the FSS for appropriate actions. To acquire or dispose of spool data sets for printing, the FSS issues get or release data set ***requests***, respectively to JES2. Both JES2 and the FSS use the FSIREQ macro to invoke particular FSI service routines. HASPFSSP in HASJES20 and HASPFSSM in the functional subsystem address space coordinate this communication.
3. Programs reside within the FSS that perform the processing for a single system entity (such as a device); this entity is managed by the FSS. The programs in the functional subsystem address space that perform the function-specific processing and communicate with JES2 are collectively called the FSA; the FSA communicates with JES2 via the FSI; there may be multiple FSAs per FSS.

To summarize, the FSS comprises the FSS address space and its contents, the functional subsystem application(s) (FSA or FSAs), and HASPFSSM. HASPFSSP is the JES2 module that communicates with the FSS.

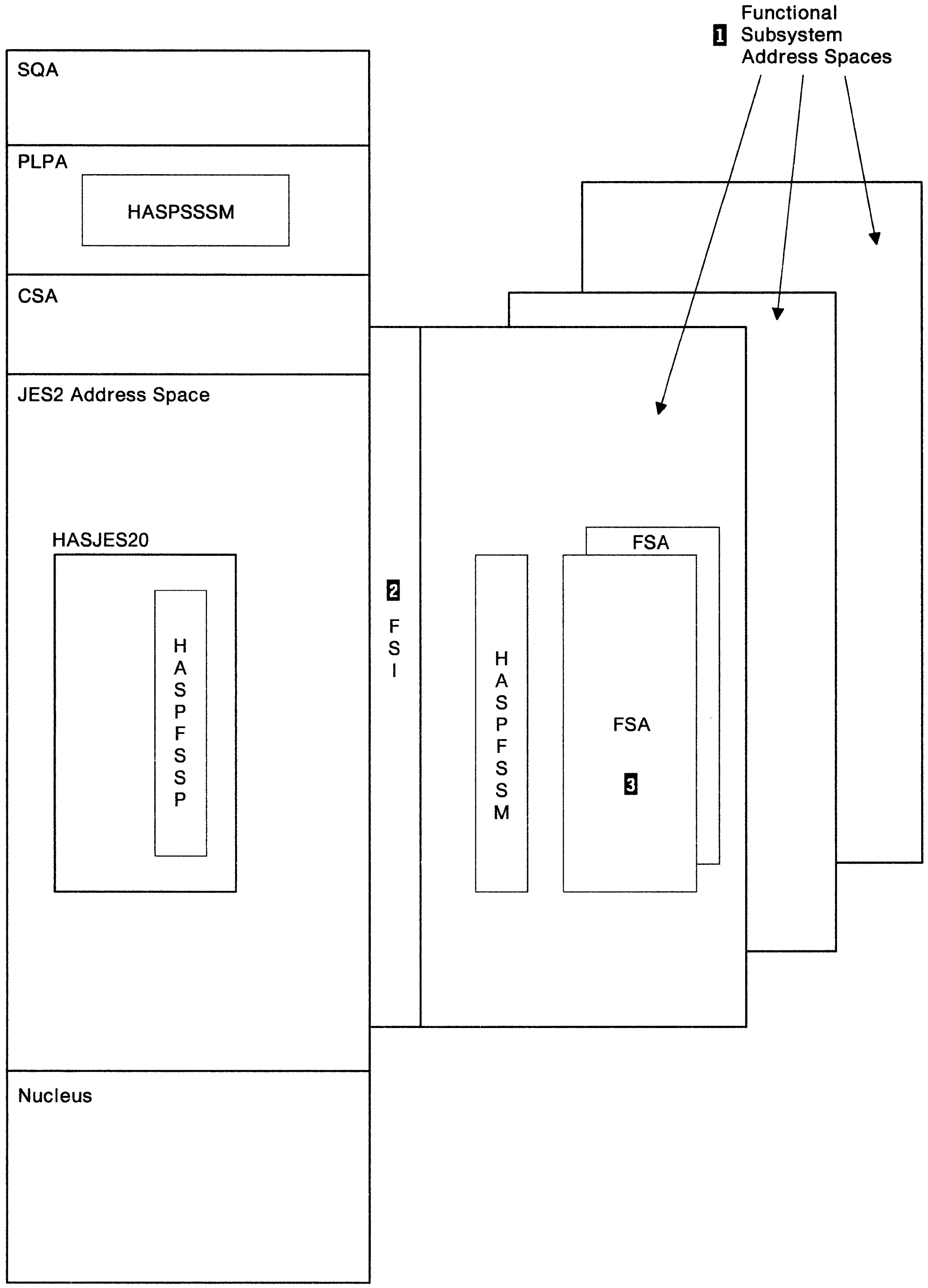


Figure 1-13. JES2 Functional Subsystems

Connect/Disconnect Services

For JES2 to communicate with the functional subsystem, JES2 must know that the functional subsystem exists and is active, that the functional subsystem application is ready to perform the functional-specific processing, and that the FSI is initialized for use. JES2 becomes aware of an active functional subsystem and an active FSA through the connect process.

The connect process is always initiated by the functional subsystem in response to a JES2-generated START command or order. The connect process occurs at two levels.

1. Functional subsystem connect

   This connect process informs JES2 that a functional subsystem has been initialized and is ready to be acknowledged as usable by JES2. This process occurs as a response to the JES2-generated START command.

2. FSA connect

   This connect process informs JES2 that the FSA itself is ready to process work. This process occurs in response to JES2 issuing a start FSA order.

The disconnect process is generally the reverse of each of the forms of the connect process. The FSA disconnect indicates to JES2 that the FSA is not ready to receive requests to process work. The FSS disconnect indicates to JES2 that the functional subsystem address space is terminating and that use of the FSI for that functional subsystem is not permitted.

The connect and disconnect services are part of a collection of FSI services.

FSI Services

JES2 and the functional subsystem communicate with each other through the functional subsystem interface (FSI), which consists of a collection of services that allow, for example, the FSA to access spool data sets and their contents. These FSI services consist of the following:

- Communication services (SEND, ORDER)

  JES2 uses the FSI ORDER communication service to communicate orders to the FSA, for example, when JES2 determines that an operator command is directed to a device under control of the FSA. The FSA in the functional subsystem performs the required processing to satisfy the operator command and uses the SEND communication service to communicate responses to JES2 concerning the outcome.

- Status services (QUERY)

  JES2 uses the QUERY status service to request status information concerning entities managed by the functional subsystem.

- Data Set Access services (GETDS, RELDS, GETREC, FREEREC, CHKPT)

  The FSA can obtain access to an existing JES2 SYSOUT data set and its characteristics (using GETDS). In this case, the FSA communicates the need to JES2 to process a SYSOUT data set, which resides on the spool; this is similar to allocating and opening the data set.

  The FSA can release a previously obtained data set (via RELDS). This is similar to closing and unallocating the data set.

The FSA, after obtaining access to a data set, can get and free records (using GETREC and FREEREC) of that data set; this is similar to performing I/O against the data set using an access method.

- Control services (POST)

  FSI services that require a large amount of processing may need to wait for extended periods of time. To prevent the wasting of resources during the period of waiting, JES2 signals the completion of these FSI services asynchronously, using the POST control service.

Functional Subsystem Initialization

The installation defines a functional subsystem to JES2 during JES2 initialization. The installation uses the FSSDEF initialization statement to define the functional subsystem and uses the PRTnnnn initialization statement to associate a device (for example, the 3800 Printing Subsystem Model 3) with the functional subsystem being defined in the FSSDEF statement. After JES2 is initialized and after the functional subsystem is started, the functional subsystem application (FSA) becomes associated with the device specified by the PRTnnnn statement. Figure 1-14 illustrates the specific processing that occurs when a functional subsystem is initialized to support a printer defined by the PRT2 initialization statement. The descriptions following Figure 1-14 summarize the specific steps in the process.

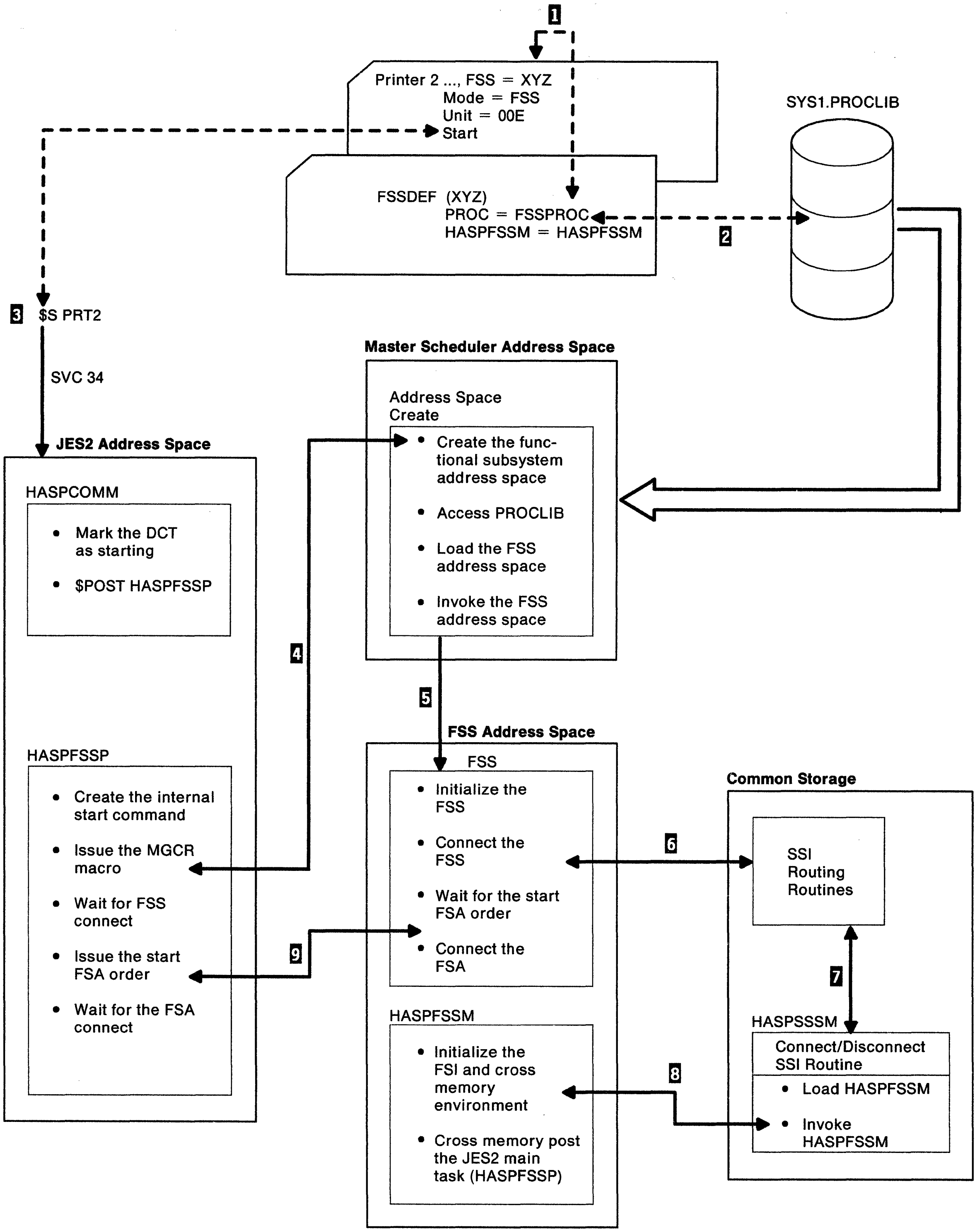


Figure 1-14. Functional Subsystem Initialization

1. The FSSDEF subscripted initialization statement and the PRT2 initialization statement keyword, FSS, associate a functional subsystem (FSS) with a device. During JES2 initialization, HASPIRMA allocates and initializes functional subsystem control blocks (FSSCBs) in CSA for each unique FSSDEF initialization statement. FSSCBs contain information about the functional subsystems that are defined in the system.

2. The PROC= keyword on the FSSDEF initialization statement identifies what "start" procedure in SYS1.PROCLIB is to be used to start the functional subsystem. This procedure identifies the program, the functional subsystem application (FSA), that is to be loaded into the functional subsystem address space and run as part of the FSS.

3. After JES2 initialization completes, the functional subsystem is either started automatically or started when the operator issues the $S PRT2 command.

   Automatic Start of the Functional Subsystem

   Specifying the PRT2 initialization statement with the START operand automatically starts the printer. JES2 places the PCE associated with the printer on the $READY queue. At the end of JES2 initialization, all JES2 processors on the $READY queue are dispatched to process work.

   Operator Start of the Functional Subsystem

   When the operator issues the $S PRT2 command to start a printer, a functional subsystem is initially started if it hasn't already been started by another printer defined to that address space. HASPCOMM in the JES2 address space receives control to process the $S command. HASPCOMM turns on the DCTSTART flag in the DCT (ensuring that the DCTDRAIN bit is off); then HASPCOMM $POSTs the functional subsystem service processor (HASPFSSP) so that HASPFSSP's PCE is placed on the $READY queue. After HASPCOMM returns to the JES2 dispatcher, HASPFSSP is eventually dispatched and, similar to the automatic start, HASPFSSP begins the start processing for the functional subsystem.

4. During the initial start processing for the functional subsystem, HASPFSSP creates an internal start command (START PROCNAME.FSSNAME,,,(SSNAME, FSID),SUB=(JES2NAME)) and issues the MGCR macro to invoke MVS address space creation; this results in an SVC 34. (JES2NAME is the name of the subsystem that issues the internal start command.) The master scheduler address space receives control and the address space creation routine executes.

   The address space creation routine creates a new address space for the functional subsystem and accesses SYS1.PROCLIB to obtain the start procedure for this new address space. From the start procedure, the address space creation routine locates the name of the program that is to be loaded into the new address space. This program is a functional subsystem application part of the functional subsystem. The address space creation routine then loads this program and invokes it. (For a printer device such as the 3800 model 3, the program is the Print Services Facility.)

   The master scheduler address space requests for SYSLOG to be started when JES comes up. Then, through the MVS WRITELOG command, the operator can request for it to be spun off and printed. The security token of the master scheduler address space is extracted during the SSI "request jobid" routine. This is the security token that is associated with the SYSLOG job and which will be passed to RACF on the early verify routine.

5. The functional subsystem (FSS) executes in its own address space, initializes itself, and issues a CONNECT request via the FSIREQ macro; this CONNECT request is at the FSS-level and indicates to JES2 that the functional subsystem is ready to communicate with JES2.

6. The FSS-level CONNECT request issued by the FSS causes the subsystem interface (SSI) to be invoked with a function code of 53 (decimal) in the SSOB.

7. The SSI service routines in common storage process the function code (53 decimal) and ultimately pass control to HASPSSSM at entry point SSIFSCNT.

   SSIFSCNT recognizes the FSS-level connect request, locates the module specified on the FSSDEF initialization statement for the HASPFSSM= keyword (the default is HASPFSSM and is used in Figure 1-14, loads HASPFSSM, and invokes it to complete the processing of the connect.

8. HASPFSSM completes the connect processing. HASPFSSM initializes itself, initializes a cross-memory environment between the JES2 address space and the functional subsystem address space, and initializes the functional subsystem interface (FSI).

   - **Initializing the FSI**

     HASPFSSM initializes the FSI by acquiring and initializing the functional subsystem vector table (FSVT), the functional subsystem control tables (FSCTs), and the functional subsystem extension block (FSSXB).

     An FSVT, anchored in the ASXB, exists for each connected functional subsystem. For each FSVT there exist two FSCTs: one for the functional subsystem and one for JES2. These FSCTs contain the addresses of the individual routines comprising the FSI functions that JES2 and the functional subsystem can invoke.

   - **Initializing the Cross-Memory Environment**

     HASPFSSM initializes a cross-memory environment between the functional subsystem and JES2. This initialization allows JES2 to PC to the FSS, the FSS to PR back to JES2, and the JES2 main task to SSAR to the FSS address space. This allows cross memory movement of FSI parameter list information to occur in access register mode.

   HASPFSSM acquires an extension to the FSSCB (FSSXB). The FSSXB contains the FSI order parameter list and the parameter list JES2 and the functional subsystem use when issuing and responding to FSS-level orders.

   HASPFSSM performs initialization of quick cell pools and associated free cell stacks, cross memory posts the JES2 main task (using HASPFSSP's XECB) and returns to HASPSSSM. When the JES2 main task gets dispatched, the JES2 dispatcher recognizes the XECB post from HASPFSSM by searching a chain of XECBs anchored from the the HCT; the JES2 dispatcher then $POSTs HASPFSSP so that HASPFSSP can continue FSS start up processing.

   Note: JES2 considers the FSS connect as a successful response to the START FSS command.

9. HASPFSSP, in preparation for the start of a functional subsystem application (FSA), acquires a functional subsystem access control block (FSACB) to represent the started FSA. The FSACB is chained from the functional subsystem control block (FSSCB). HASPFSSP issues the start FSA order to the functional subsystem to start the FSA. When the response to the start order (that is, a successful FSA connect) is received by HASPFSSP, HASPFSSP marks

the FSA as active by setting the FSACTIVE bit on in the FSAFLAG2 byte of the FSACB. HASPFSSP then issues a $POST FSS to indicate to itself and other functional subsystem service processors that requests to other FSAs under this FSS may be issued. HASPFSSP next issues the start device FSI order, waits for a response to the order, marks the device as started, and waits either for work ($WAIT on JOT) to be processed or for commands to be issued against the device.

JES2 - Functional Subsystem Processing

When the functional subsystem (FSS) and functional subsystem application (FSA) are connected and the FSA is started, JES2's functional subsystem service processor (HASPFSSP) handles all interactions with the functional subsystem.

Issuing Orders to the FSS/FSA

HASPFSSP can issue the following orders to the FSS:

- Start FSA
- Stop FSA
- Stop FSS

HASPFSSP can issue the following orders to the FSA:

- Start device
- Stop device
- Query device ($N)
- Set device ($T)
- Operator intervention
- Synch ($B, $F, $E, $C, $I, $Z, $T, ($CJ,P))

HASPSERV issues the query order ($DU).

HASPFSSP issues these orders to the FSA to affect how the FSA interacts with the device associated with it. Figure 1-15 summarizes the order processing that takes place for this device:

1. For each unique order, HASPFSSP has a routine that processes the order (Figure 1-16 lists the various orders that HASPFSSP can issue and identifies the routine within it that processes the order.) Each of these routines eventually issues a $CALL FSPORCMS to invoke the routine in HASPFSSP that handles the interaction between HASPFSSP (running in the JES2 address space) and HASPFSSM (running in the functional subsystem address space).
2. After FSPORCMS is invoked, FSPORCMS gets the PC number for the HASPFSSM order routine and issues a PC to that routine, which is the FSMORDER entry point in HASPFSSM.

   Note: When FSPORCMS regains control, it returns to its caller, which is the routine in HASPFSSP that initiated the order.
3. Entry point FSMORDER in HASPFSSM receives control and issues a PCLINK macro to save linkage information. The save area with the order parameter list FSMORDER uses is either in the functional subsystem application extension block (FSAXB) for a FSA order or the functional subsystem extension block (FSSXB) for a FSS order. FSMORDER then issues the FSIREQ REQUEST=FSIORDER macro to invoke the FSA to service the order.

4. When the FSA gets control, it services the order. For the 3800 model 3, where the FSA is the Print Services Facility, the Print Services Facility interacts with the device in response to the specific order. For example, the Print Services Facility stops the device (for a stop device order) or starts the device (for a start device order). The FSA then returns to HASPFSSM.

5. HASPFSSM restores the caller's linkage registers, reestablishes the cross memory environment and issues the PR instruction to return to HASPFSSP at the instruction immediately following the PC.

6. Processing of the order can be performed immediately by the FSA (for example, the query order) or can be performed asynchronously (for example, the synch order). For the asynchronous case, the response to the order is communicated via the FSI send function (FSIREQ TYPE=SEND, REQUEST=FSISEND).

7. The FSI send function, identified by label FSMSEND in HASPFSSM, cross memory posts the JES2 main task, running in the JES2 address space. This post activates the JES2 main task response processing function.

8. The JES2 main task response processing function is located at label FRSFORDR in HASPFSSP. FRSFORDR processes each response and initiates appropriate actions.

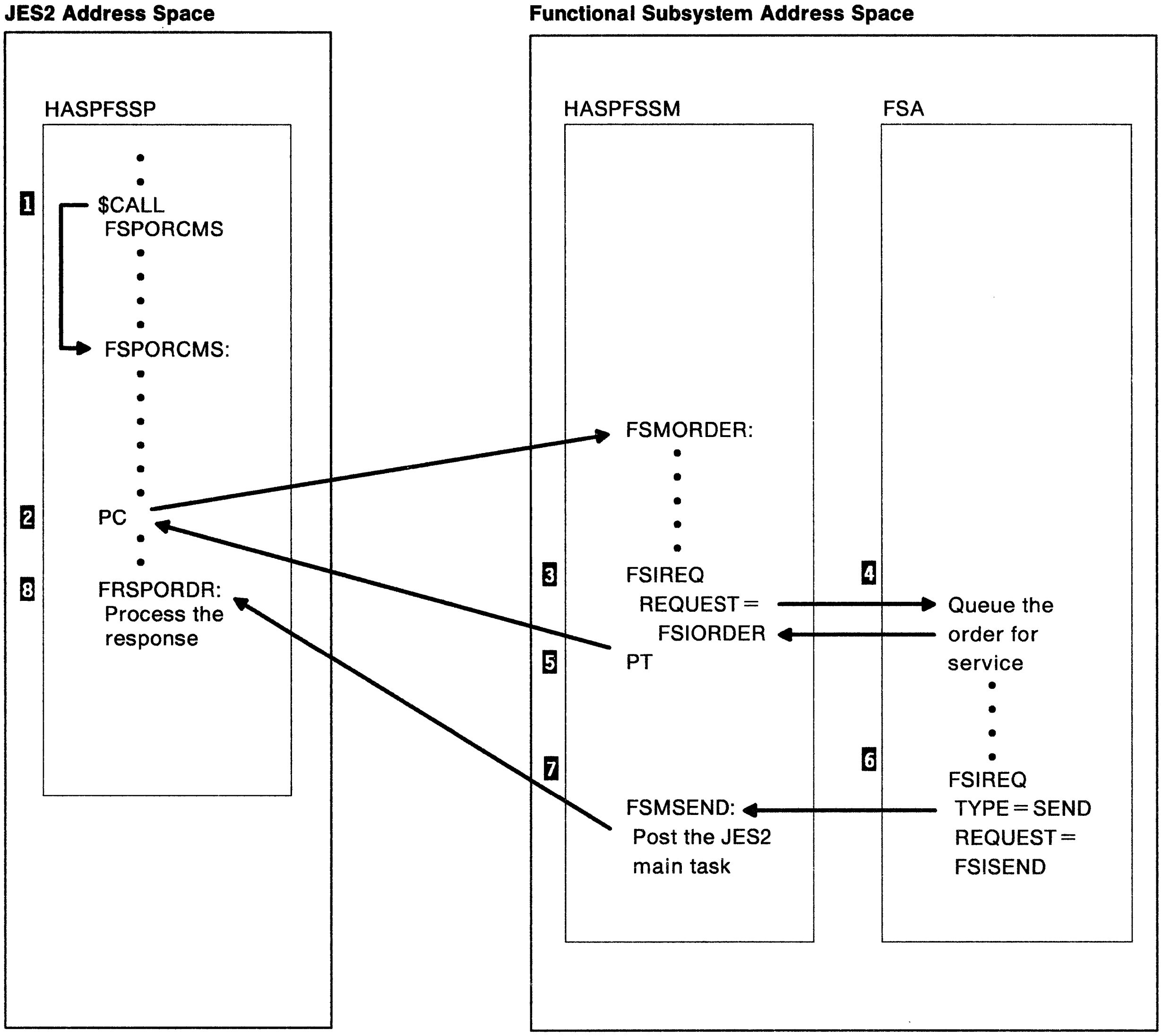

JES2 Address Space
HASPFSSP
1
$CALL FSPORCMS
FSPORCMS:
2
PC
8
FRSPORDR: Process the response
Functional Subsystem Address Space
HASPFSSM
FSMORDER:
3
FSIREQ REQUEST= FSIORDER
5
PT
7
FSMSEND: Post the JES2 main task
FSA
4
Queue the order for service
6
FSIREQ TYPE=SEND REQUEST= FSISEND

Figure 1-15. Order Processing

Figure 1-16. Order Summary

Order	Type	Activating Condition	Initiating Routine
Start FSA	FSS	Successful FSS connect and/or $S PRT command	FSPSTFSA
Stop FSA	FSS	Successful stop device order	FSPSPFSA
Stop FSS	FSS	$P JES2	FSPSPFSS
Start Device	FSA	Successful start FSA	FSPSTDVC
Stop Device	FSA	$P PRT or successful quiesce of data set processing	FSPSPDVC
Query ($N)	FSA	$N PRT	FSPRPTDV
Set	FSA	$T PRT NPRO=	FSPSETDV
Operator	FSA	Setup detected by HASPFSSM	FSPOIRDV FSACB
Synch		$B $F $E $C $I $Z $CJ,P $T	FSPBKFDS FSPRSTRD FSPCANCL FSPINTRP FSPHALTD FSPCANJB FSPCANJB

FSA Request Handling

HASPFSSP waits for requests from the FSA for work to process. Requests from the FSA are made through the functional subsystem interface (FSI) to JES2. These requests include the following:

- GETDS

  GETDS processing handles requests for SYSOUT data sets. These requests originate in the functional subsystem address space and are immediately processed by HASPFSSM (if possible). JES2 satisfies a GETDS request by selecting data sets (from a JOE) that have characteristics matching the current device setup.

- RELDS

  RELDS terminates the processing the FSA does for output data sets that were originally obtained via GETDS. The FSA issues the FSIREQ macro to request the RELDS, indicating one of the following actions:

  - JES2 can purge the data set because it has been successfully processed.
  - JES2 is to requeue the data set for processing, because the data set has been only partially processed or could not be processed due to an error condition.

- SEND

  SEND is invoked when the FSA issues a FSIREQ REQUEST=SEND macro. The FSA uses SEND to respond asynchronously to orders JES2 issues. After the FSA issues the FSIREQ macro, HASPFSSM gets control at FSMSEND and turns off the appropriate response outstanding bit. (This bit is associated with an order that is issued by JES2. FSSRSOUT applies to functional subsystem orders; FSARSOUT applies to FSA orders.) Depending on the settings of these bits and the order outstanding bits (FSSOROUT for FSS orders and FSAOROUT for FSA orders), JES2 through HASPFSSP completes the processing of the order; Figure 1-17 shows the relationships between the order and response outstanding bits.

  The FSA can also request termination with a SEND request. JES2 responds by stopping the device, then disconnecting the FSA.

- CHKPT

  The FSA uses CHKPT processing to periodically save on the spool restart information (the position of the last physical data processed for the data set) about the output data set that is currently being printed. The FSA issues the FSIREQ REQUEST=CHKPT macro to invoke this processing.

- POST

  When the FSA issues a GETDS request, the request may not be satisfied immediately; JES2 indicates to the FSA that the GETDS is to be satisfied asynchronously. The FSPPOST routine in HASPFSSP eventually gets control when a JOE is found with the correct characteristics to match the GETDS request. FSPPOST uses cross memory services to invoke FSMPOST in HASPFSSM which ultimately informs the FSA (via the FSIREQ FUNCTION=POST macro) that its GETDS request is satisfied.

		Order Outstanding Bit (FSSOROUT or FSAOROUT)	
Response		**ON**	**OFF**
Outstanding Bit	ON	Waiting for SEND from FSA	Not Possible
(FSSRSOUT or FSARSOUT)	OFF	Response Received. Action Required by JES2	No order is presently being processed

Figure 1-17. Relationship of Order and Response Outstanding Bits

Certain FSA requests do not cause JES2 to communicate with the FSA (that is, communication between HASPFSSM and HASPFSSP via the FSI does not occur); these requests are:

- GETREC

  The FSA issues the FSIREQ REQUEST=GETREC macro to invoke HASPFSSM to obtain records from the output data set it is processing. HASPFSSM uses the HASP access method (HAM) to obtain the records from this output data set.

- FREEREC

  The FSA issues the FSIREQ REQUEST=FREEREC macro to invoke HASPFSSM to release the storage occupied by the records that were obtained via GETREC requests.

JES2 - Functional Subsystem Control Block Structure

Figure 1-18 and Figure 1-19 illustrate the control block structure that is used to support the communication between JES2 and the functional subsystem. Two parts make up this control block structure:

- A part that represents a JES2-level of control (that is, those control blocks that both JES2 and HASPFSSM use to determine the status of requests or orders that are being processed).
- A part that specifically supports the use of the FSIREQ macro that the FSA and JES2 use in their communication. This support of the FSIREQ macro is part of the FSI.

JES2-Level Control Block Structure

Figure 1-18 illustrates the JES2-level control block structure used to control the communication between JES2 and HASPFSSM.

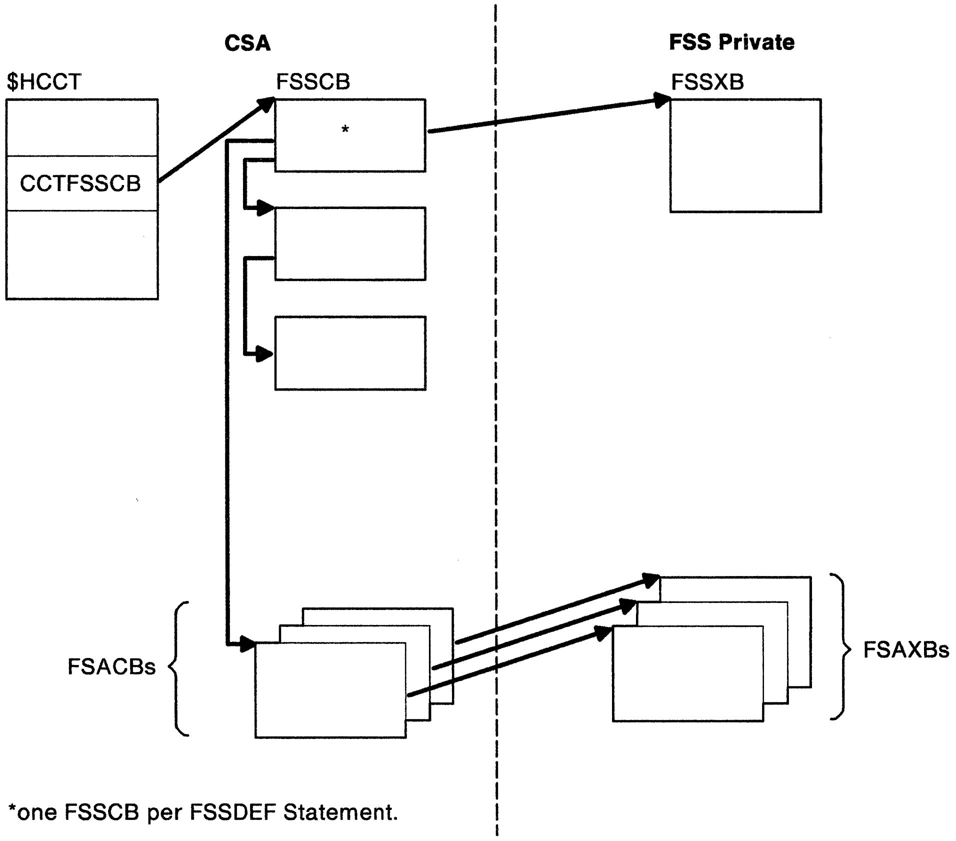


Figure 1-18. JES2-Level Control Block Structure

For each FSSDEF statement that JES2 processes during initialization, JES2 allocates a functional subsystem control block (FSSCB); the FSSCB describes a functional subsystem that is defined in the system. Functional subsystems can also be dynamically added with the $ADD command.

During the connect process for a functional subsystem, the functional subsystem extension block (FSSXB) is allocated in the functional subsystem address space private area and pointed to by the FSSCB. The FSSXB contains the order response area that JES2 uses to examine the status of orders (at the FSS level) that it has sent to the functional subsystem.

During the connect process for a functional subsystem application (FSA), the functional subsystem application control block (FSACB) and its extension (FSAXB) are allocated; the FSACB is in CSA; the FSAXB is in the functional subsystem address space private area; the FSACB points to the FSAXB. The FSAXB contains the order response area for orders and the parameter list that HASPFSSM uses to invoke the FSI ORDER and POST functions.

FSIREQ Control Block Structure

As a result of the connect process for a functional subsystem, a functional subsystem address space is created and the functional subsystem interface (FSI) is initialized. During the creation of the functional subsystem address space, an ASCB is created in SQA for use by the system. The ASCB points to the ASXB, which is initialized to point to the functional subsystem vector table (FSVT). The ASXB is located in the private area of the functional subsystem's address space. The FSVT (also in the FSS address space's private area) contains paired pointers to functional subsystem control tables (FSCTs).

Figure 1-19 illustrates the FSIREQ control block structure that supports the use of the functional subsystem interface (FSI), and the numbered steps of text that follow the figure give a corresponding description of this structure.

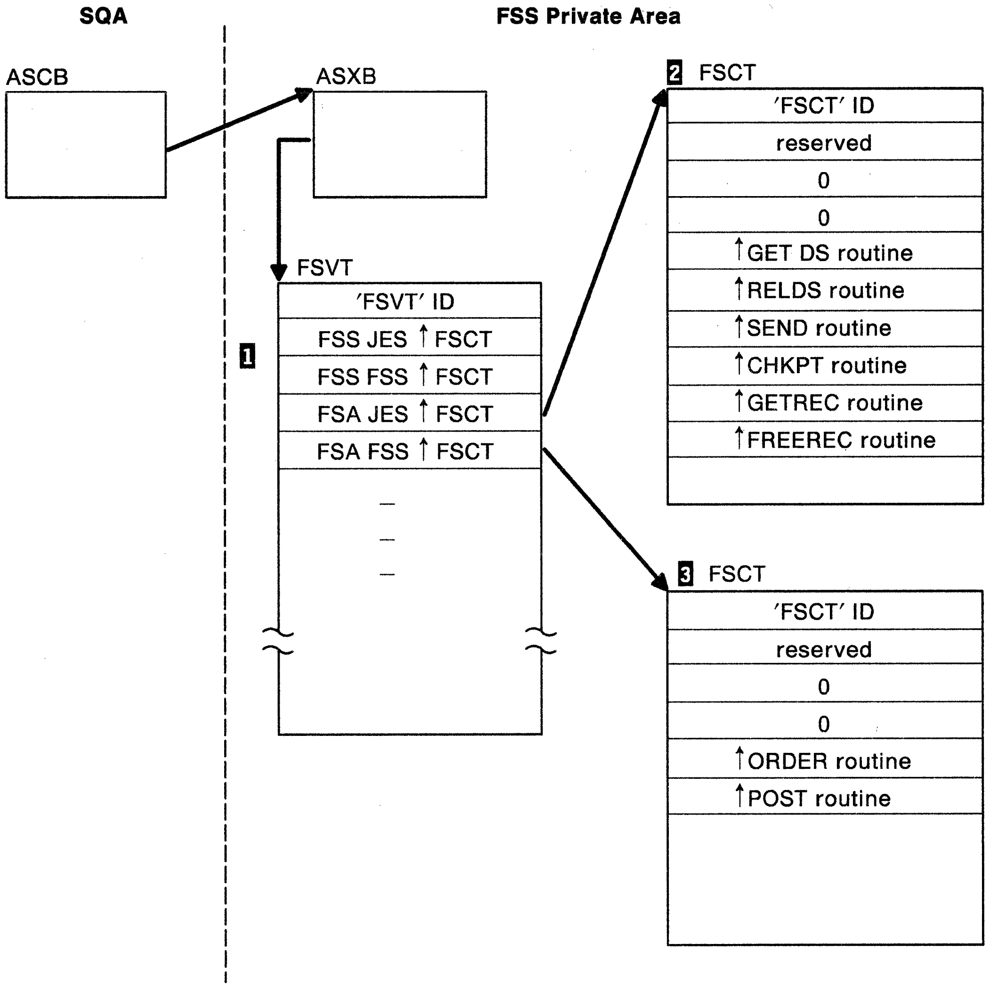


Figure 1-19. FSIREQ Control Block Structure

1. Functional subsystem vector table (FSVT)

   Each entry in the FSVT contains a pointer to a FSCT. The FSCT pointers in the FSVT are organized in pairs: one pointer is the address of the FSCT that supports FSI functions that are called when the FSIREQ TARGET=JES macro is issued; the other pointer is the address of the FSCT that supports FSI functions that are called when the FSIREQ TARGET=FSS macro is issued.

   Each pair of FSCT addresses also is defined either for FSS level functions or FSA level functions. Specifically, the first two pointers in the FSVT are for FSS-level functions and the rest of the pairs in the FSVT are for FSA-level functions; each FSA pair of pointers is associated with each FSA that is active in the functional subsystem.

2. FSCT for FSIREQ TARGET=JES

   This FSCT contains the addresses of routines in HASPFSSP that process requests (GETDS, RELDS, SEND, GETREC, FREEREC, and CHKPT) made by the FSA via the FSIREQ macro. When these requests are made, this FSCT is used to give control to the proper HASPFSSM routine that processes the request.

3. FSCT for FSIREQ TARGET=FSS

   This FSCT contains the address of the post routine that posts JES2 concerning the completion of FSA requests and the address of the order routine that invokes the FSA to process orders issued by JES2. When HASPFSSM issues either the FSIREQ FUNCTION=FSIORDER,TARGET=FSS macro or the FSIREQ FUNCTION=FSIPOST,TARGET=FSS macro, this FSCT is used to give control to the proper FSA routine that processes the request.

JES2 Job Processing

JES2 processes a job stream in an ordered way that comprises the following stages as illustrated in Figure 1-20:

1. Input
2. Conversion
3. Execution
4. Output
5. Print/Punch
6. Purge

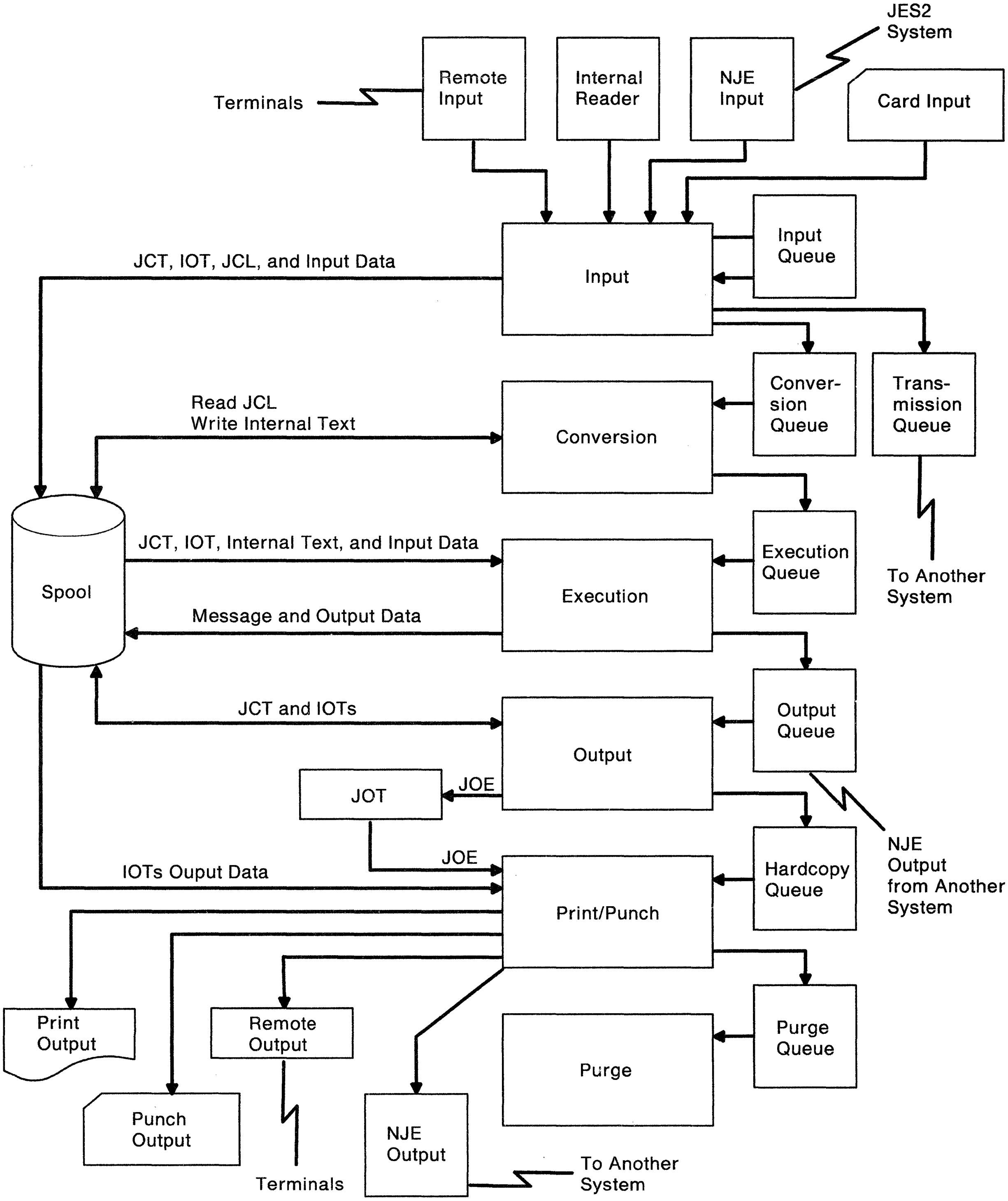


Figure 1-20. JES2 Job Processing

JES2 reads a job into the system (see Figure 1-20) from the incoming job stream that consists of jobs from a variety of sources: remote terminals connected directly to the JES2 system, the MVS internal reader (for a TSO LOGON, a TSO-submitted job, a system task, or a job presented to the internal reader from other sources), another JES2 system through an NJE line, another networking system, and the card reader. Each job that is read in is placed on the input queue and a SAF call is made to verify that the job can be submitted. If the job is to be processed on this JES2 system, JES2 places it on the conversion queue to await processing by the next stage. JES2 also writes the job's JCL and input data to the spool. If the job is not to be processed by this JES2 system, JES2 places the job on a job transmission queue for eventual transmission to another node for processing.

Conversion

JES2 invokes the MVS converter to scan each job's JCL for syntax errors (see Figure 1-20). The MVS converter converts the JCL to internal text, which JES2 writes to the spool. JES2 queues a job with correct JCL to the execution queue by its priority. For a job with JCL errors, JES2 bypasses the execution stage and queues the JCL together with appropriate diagnostic messages to the output queue for direct use by the output stage.

Execution

An eligible initiator uses the SSI to request a job from JES2 so that the job can be run. JES2 selects a job for execution from the execution queue (see Figure 1-20) by priority within its class. JES2 reads input from spool and writes output to it during the execution of the processing program. When the job completes execution, the job termination portion of MVS informs JES2 (via the SSI). JES2 then places the job in an output queue to await processing of its output.

Output

JES2 takes a particular job, after it completes execution, and processes all of its SYSOUT data sets by creating work elements and placing them in the job output table (JOT) (see Figure 1-20); JES2 builds these work elements based on the output characteristics of the SYSOUT data sets for the job.

Print/Punch

JES2 selects output work elements for processing from the JOT according to the work selection specified for the device (see Figure 1-20). A SAF call is made during selection time to verify that the data can print on, or be transmitted to, the selecting device. The selected output can be in a number of states; output from the job to be processed locally; output from the job to be processed at a remote location; the job output itself may be passing through this JES2 system and must be transmitted to another JES2 system in an NJE network; output from the job to be processed by a functional subsystem (FSS). (For FSS-selection, a SAF call is made to verify that the user requesting the printing has sufficient authority to select the data.) JES2 handles each of these situations in differing ways:

- **Job Output Transmission:** Job output passing through to another JES2 node resides on the transmission queue. JES2 selects a job's output from the JOT transmission queue for transmission to another node based upon the priority and the desirability of reaching the execution node over the available transmission line. When the job's output is transmitted, the receiving system requeues the output to its output processing stage or queues it for transmission to yet another networking system. When transmission of the job's output is completed, the transmitting JES2 node releases the resources that it used to represent the output, but only after the receiving node signals that it has accepted total responsibility for the output.

- **Local Output:** When output is to be processed at a local or remotely-attached output device, JES2 uses these local and remotely-attached output devices to produce the job's output. JES2 queues a job's print and punch data sets on the hardcopy queue for the local and remote output devices that are, when active, directly reachable either through a local attachment, via a remote job entry (RJE) connection, or through an FSS.

After processing the output for a particular job, JES2 puts the job on the purge queue.

Purge

When all processing for a job has been completed, JES2 performs purge processing (see Figure 1-20). For each job on the purge queue, JES2 releases the direct-access space acquired for the job, all control information associated with the job, and any other resources owned by the job.

JES2 Communications

JES2 subsystems can be connected, and thus can communicate with each other, in various ways:

- A multi-access spool configuration allows up to seven JES2 systems to be connected via a shared-spool device upon which common queue and checkpoint information resides. This allows individual participating JES2 systems (members) to select jobs for execution and output regardless of which system processed the jobs as input.

  When a multi-access spool configuration is part of a network job entry (NJE) configuration, the multi-access spool configuration is considered to be one node in the network. For further information about these capabilities, refer to *JES2 Initialization and Tuning*.

- A network job entry (NJE) configuration allows single JES2 systems and multi-access spool configurations to be connected together via network job entry lines and channel-to-channel adapters acting as communication lines. The NJE communication lines can be either binary synchronous communication (BSC) lines, or (in the case of SNA NJE) synchronous data link control (SDLC) communication lines using Systems Network Architecture (SNA) protocols communicating through ACF/VTAM. Note that BSC lines used for network job entry must be capable of transmitting and receiving the transparent-text mode. For further information about these capabilities, refer to *JES2 Initialization and Tuning*.

JES2 supports physical and logical communication lines and allows job entry nodes to be connected by one SDLC line and one or more BSC lines or CTC adapters. Both SDLC and BSC lines can be used concurrently, and the same SDLC line can be shared simultaneously for SNA communications between JES2 nodes as well as for other applications.

Use of the VTAM SNA application-to-application session feature in SNA NJE allows one JES2 node to directly communicate with another JES2 or other networking node, without incurring store-and-forward overhead on any intermediate nodes. Alternatively, two nodes can communicate with each other through an intermediate node by each establishing a session with the intermediate node and thereby incurring store-and-forward overhead at the intermediate node.

For ease of communication, the following terms have been defined to describe the various network job entry environments:

- Node: All of the JES2 systems sharing JES2 checkpoint information and spool space or any non-JES2 node capable of supporting the networking architecture.
- Single-system node: A node in which only one JES2 system has access to the JES2 checkpoint information and spool space.
- Multi-system node: A JES2 multi-access spool configuration in which more than one system shares the JES2 checkpoint information and spool space.
- All-systems cold start: A cold start on behalf of all of the JES2 systems comprising a node. All spool and checkpoint information is lost.
- All-systems warm start: A warm start on behalf of the only system in a JES2 complex or the first system in a MAS after an MVS IPL, or a clean shutdown of JES2. The shared spool data set and checkpoint data set and the track group map are rebuilt. All jobs in MAS are processed.
- Single-system warm start: A warm start on behalf of the processor that is not the first one in a MAS after a JES2 ABEND and an MVS IPL. Only the jobs active on this system are processed.
- Hot Start: A single-system warm start, or a warm start of a single system node, without an intervening IPL. Only jobs active on this system are processed.
- Quick Start: A warm start where the job queue and job output table (JOT) need not be updated. Use this type of start when you are adding a new member to a MAS after a clean shutdown of JES2 while other systems are still active.
- Restart: Occurs after the issue of the $E,SYS, sysid command. Only jobs active on this system are restarted.

For more information about the types of JES2 starts, see *JES2 Initialization and Tuning*.

Remote Job Entry Support

The job input and output services provided for local peripheral devices, along with a subset of the JES2 operator commands, are optionally extended to remote work stations, including both processor and non-processor terminals.

The remote terminal access method (RTAM) provides an interface between the JES2 processor and the remote work station. RTAM provides blocking/deblocking, compression/decompression, and synchronization in such a way that the processor need not be concerned with the characteristics of the remote work station with which it is communicating.

Multileaving

Multileaving, a computer-to-computer communication technique, enables synchronized, pseudo-simultaneous, and bidirectional transmission of a variable number of data streams between computers utilizing binary synchronous transmission facilities. This technique substantially increases the amount of data that can be transmitted over a BSC line in half-duplex mode. Multileaving is used by JES2 to communicate with all of the JES2 remote work stations and JES2 nodes not using SNA communications.

Work Stations

JES2 supports remote work stations running the remote terminal processing (RTP) work station packages when EML1102 is installed. Jobs may be submitted to be processed at a central computer. Print or punch output can be made available to a selected remote work station. Data is transmitted across telephone lines.

Work station RTP programs for System/360 Model 20 and higher (BSC only), System/370 and 1130 Computing System (BSC only) are generated as extensions to the central system and operate in the work station on a stand-alone basis. The JES2 RJE implementation for BSC processor work stations is based on HASP multileaving, which provides the capability for concurrent operations to support terminal job input, output, and console devices.

JES2 Features

JES2 has the following features:

- JES2 RJE supports up to 9999 remote work stations communicating over leased (point-to-point) or dial lines.
- JES2 BSC RJE provides for concurrent operations of lines assigned to unique communication line adapter addresses of the following types: SDA Type II on a 2701 Data Adapter Unit for BSC, synchronous base on a 2703 Transmission Control for BSC, and 3704 or 3705 Communications Controller providing 270X emulation.

  JES2 SNA RJE provides for the operation (concurrent with any of the above operations) of LUTYPE1 SNA remote work stations using synchronous data link control (SDLC) lines. JES2 uses a logical line interface to VTAM and the 370X network control program (NCP).
- Output routing control provides for print and punch output to be directed to the devices attached to the host system or remote work stations as designated by JES2 initialization parameters or remote networking systems, control statements submitted with the job, or an operator command.
- The remote operator control feature provides a subset of the JES2 operator commands for display of information and control of jobs and devices associated with the remote work station.
- The operator message output feature provides for the immediate transmission of messages and responses to remote operators with online multileaving and SNA work stations with consoles. The operator message output feature also provides for optional spooling of messages for all other remote work stations and offline work stations with consoles until they are online and have primary printers available.
- Terminal support on the host system provides for communication with: 2770 Data Transmission Terminal (BSC); 2780 Data Transmission Terminal (BSC); 3780 Data Communications Terminal (BSC); System/360 Models 20, 25, 30, 40, 50, 65, 65MP, 67 (in 65 mode), 75, 85, and 195 (multileaving); 1130 Computing System (multileaving); and System/32 (SDLC as a 3770). System/370s are supported as remote job entry work stations when operated in a dedicated basis control mode. With the HRTPB360 work station package, the 3777 Model 2 running the model 20 RTP multileaving package is also supported. None of the special System/370 features are used.

  In addition, the host system provides for communication with certain remote work stations which function as network addressable units in an SNA network. These work stations are the IBM 3770 and 3790 family of devices and System/32, in their SDLC versions. These work stations are referred to as type 1 logical units (LUTYPE1).
- JES2 support of BSC RJE includes use of the sign-on card and LINEnnnn initialization statement to represent a physical communication link.
- JES2 support of SNA RJE terminals includes the use of the LOGON command and the use of the LINEnnnn initialization parameter to represent a logical line.
- The sign-on feature provides for remote identification and offers optional line authorization capability.

- Remote characteristic support utilizes the unique features on each remote work station: full text transparency (required for object decks), text compression, print line width, buffer size, and blocking capabilities. Multi-point or multi-drop line features are prohibited for BSC; they may be defined to VTAM for SNA devices, but that definition must be transparent to JES2.
- Remote job priority adjustment provides for favoring or limiting the JES2 scheduling priority of jobs submitted from each remote work station.
- Line restart provides for warm starting of print output after remote work station or line failures.
- Line error recovery provides for a continuous retry until transmission is successful. (This feature does not apply to logical lines for which other recovery techniques -- such as VTAM error recovery procedures -- are provided.)
- JES2 support of the 3800 printer (and 3211) provides special allocation, despooling, and channel program management facilities. At the user's option, space may be allocated in terms of track cells (sets of blocked spool records allocated in definite order). With track cell allocation in effect, records optionally may be read from the spool volume (despooled) in track cell units through a single I/O operation. When printer data set records are written to the printer, program control interrupts are used each time a new channel program is constructed to append, if possible, the new channel program to one already in progress. This reduces the number of EXCPs required to write an output data set.
- JES2 support of devices includes use of functional subsystems (FSSs) where device support code exists in a functional subsystem address space.

Chapter 2. Method of Operation

The following HIPO (hierarchy plus input-process-output) diagrams illustrate the high-level, functional flow of logic through the JES2 components. These diagrams are intended to guide you to a particular area of the program listing or a more detailed discussion of the component within this manual.

Legend

The following symbols are used in the HIPO diagrams.

↑ NAME	contains pointer to NAME
──▶	pointer
- - - - - - - -▶	data reference
⇨	data movement
▨▶	input/output data flow
➡	control
①	action performed
◇1	conditional action performed
➡①	branch to step ①

Job Processing Overview

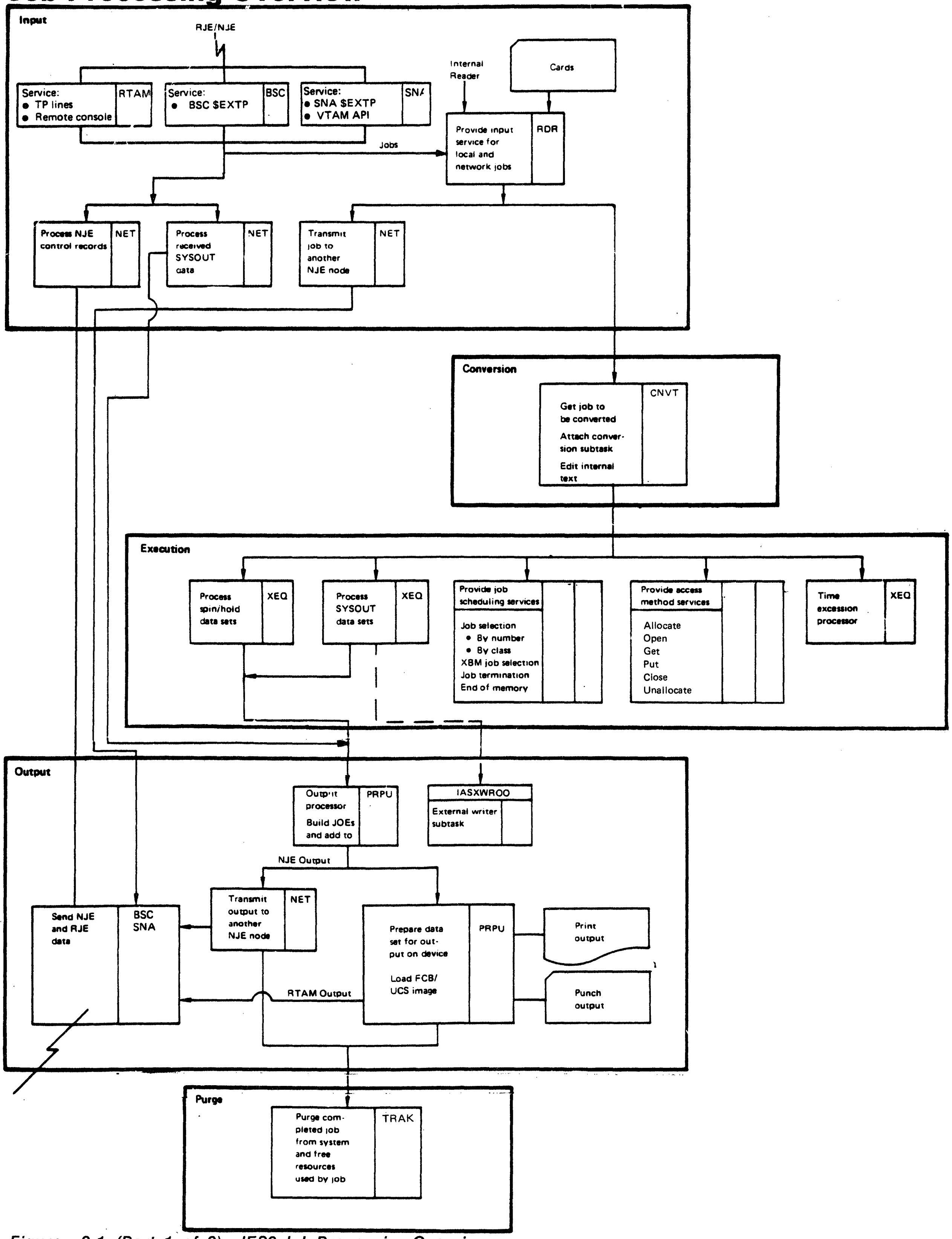


Figure 2-1 (Part 1 of 2). JES2 Job Processing Overview

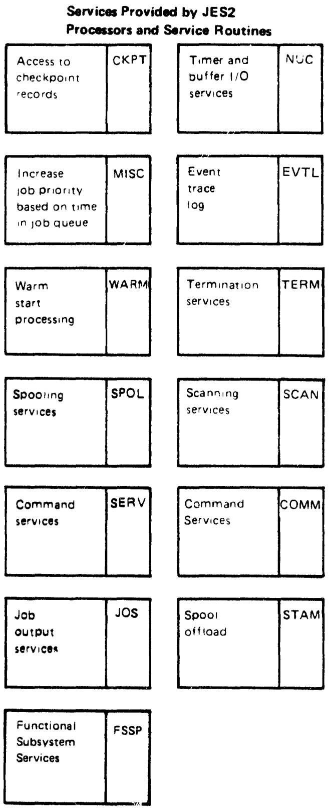
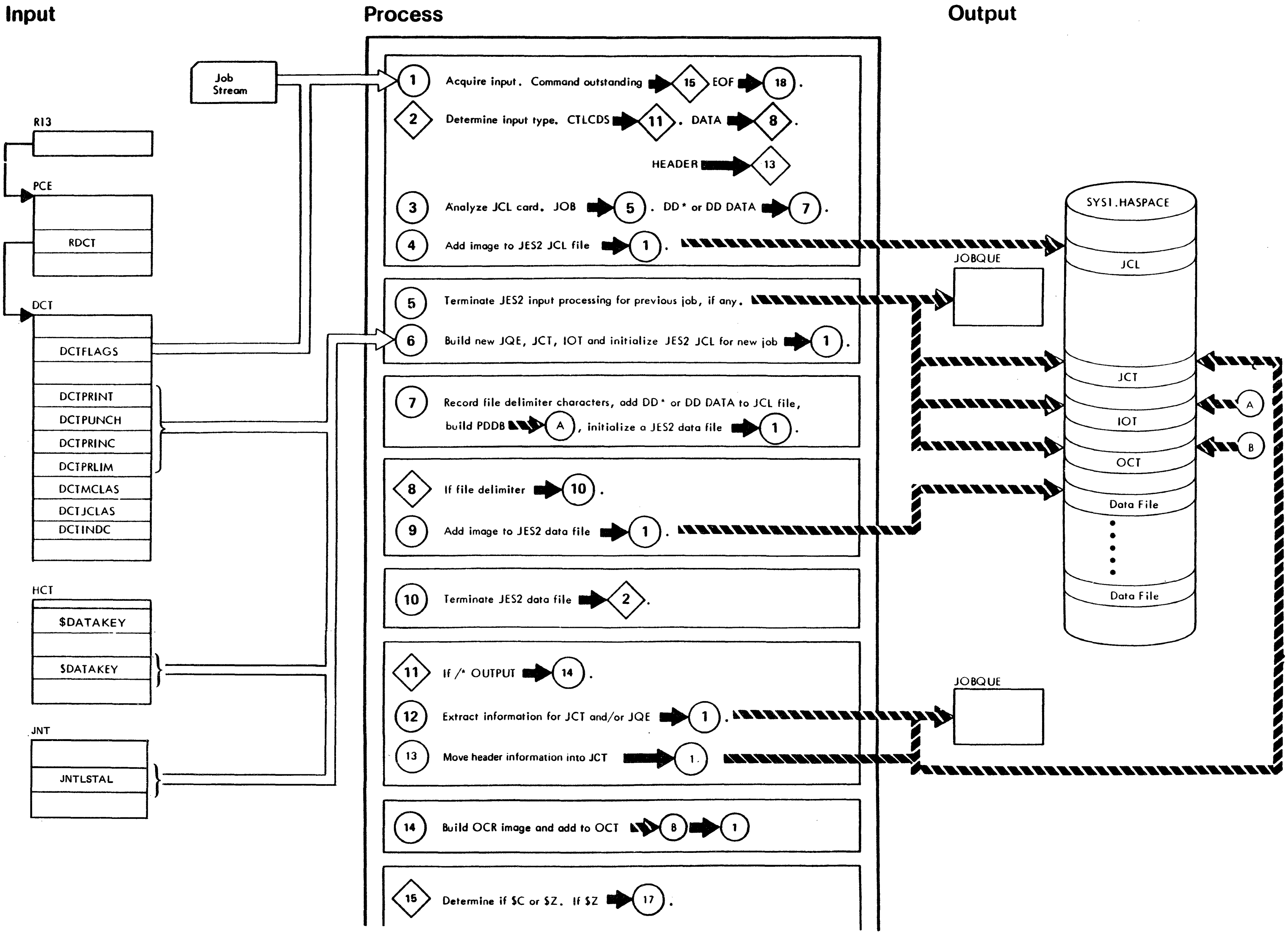


Figure 2-1 (Part 2 of 2). JES2 Job Processing Overview

JES2 Input Processor



1 Acquire input. Command outstanding ➡ 15 EOF ➡ 18.

2 Determine input type. CTLCDS ➡ 11. DATA ➡ 8.
HEADER ➡ 13

3 Analyze JCL card. JOB ➡ 5. DD* or DD DATA ➡ 7.

4 Add image to JES2 JCL file ➡ 1.

5 Terminate JES2 input processing for previous job, if any.

6 Build new JQE, JCT, IOT and initialize JES2 JCL for new job ➡ 1.

7 Record file delimiter characters, add DD* or DD DATA to JCL file, build PDDB ➡ A, initialize a JES2 data file ➡ 1.

8 If file delimiter ➡ 10.

9 Add image to JES2 data file ➡ 1.

10 Terminate JES2 data file ➡ 2.

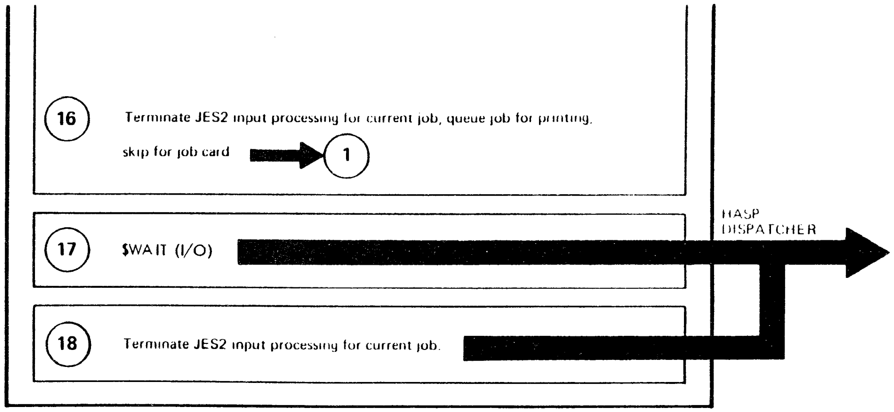
11 If /* OUTPUT ➡ 14.

12 Extract information for JCT and/or JQE ➡ 1.

13 Move header information into JCT ➡ 1.

14 Build OCR image and add to OCT ➡ B ➡ 1

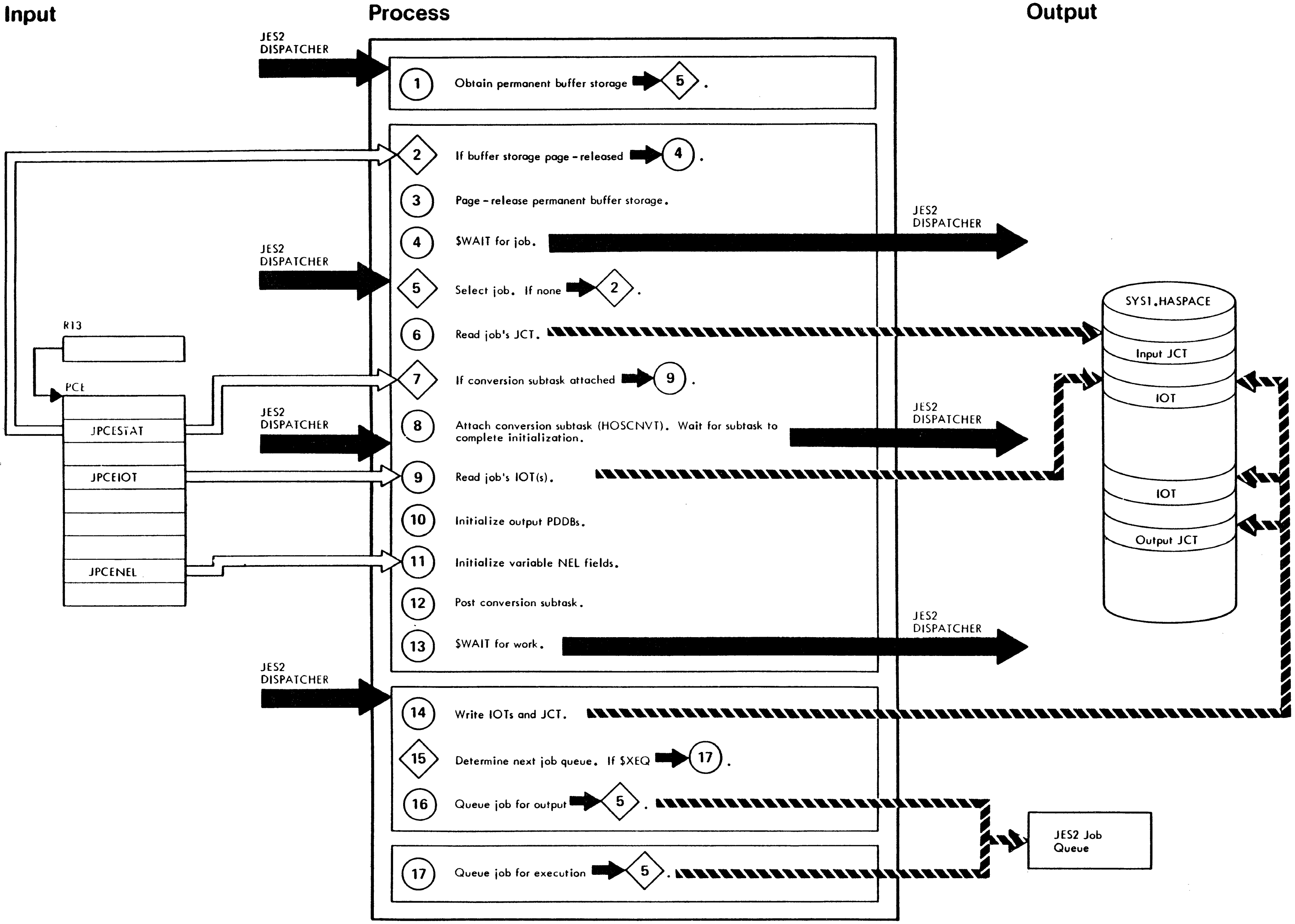
15 Determine if $C or $Z. If $Z ➡ 17.


16
Terminate JES2 input processing for current job, queue job for printing,
skip for job card
1
17
$WAIT (I/O)
HASP
DISPATCHER
18
Terminate JES2 input processing for current job.

ASSEMBLY LISTING NAME	ASSEMBLY LISTING LABEL	NARRATIVE
HASPRDR	RNXTCRD	1 Input is acquired from a local card reader, the internal reader, remote work station, or the network job receiver.
HASPRDR	HASPRCCS	2 The table of control words at RCCTAB is searched for the various JES2 control cards. Exit points RXITCCA, RXITCCB, RXITCCC (for exit 4) is used for a JECL card.
HASPRDR	RJCLCARD	3 Exit points , RXITJBCD and RXITJBCC, (for exit 2) is used for a JOB card. Exit point (for exit 3) is used for the RXITACC.
HASPRDR	HASPRJCS	5 The job queue management service routine SQADD is used to update the JES2 job queue.
HASPRDR	HASPRDDS	7 A PDDB is built to indicate the track address of where the data will be written.
HASPRDR	RFLTEST	9 If flush switch is not on put data record in JES2 data file.

ASSEMBLY LISTING NAME	ASSEMBLY LISTING LABEL	NARRATIVE
HASPRDR	RJBPCARD	12 Information is extracted from the control cards and placed in JCT and/or JQE.
HASPRDR	RNJEHDTR	13 The job header records received from the network are stored in the network header area of the JCT.
HASPRDR	ROTPCARD	14 The output control record is added to the output control table
HASPRDR	RDCKCOM	15 The DCTFLAGS bits are tested for $C or $Z.
HASPRDR	REXITA	16 Define exit point for exit 20.
HASPRDR	RWAIT	17
HASPRDR		18

JES2 JCL Conversion Processor

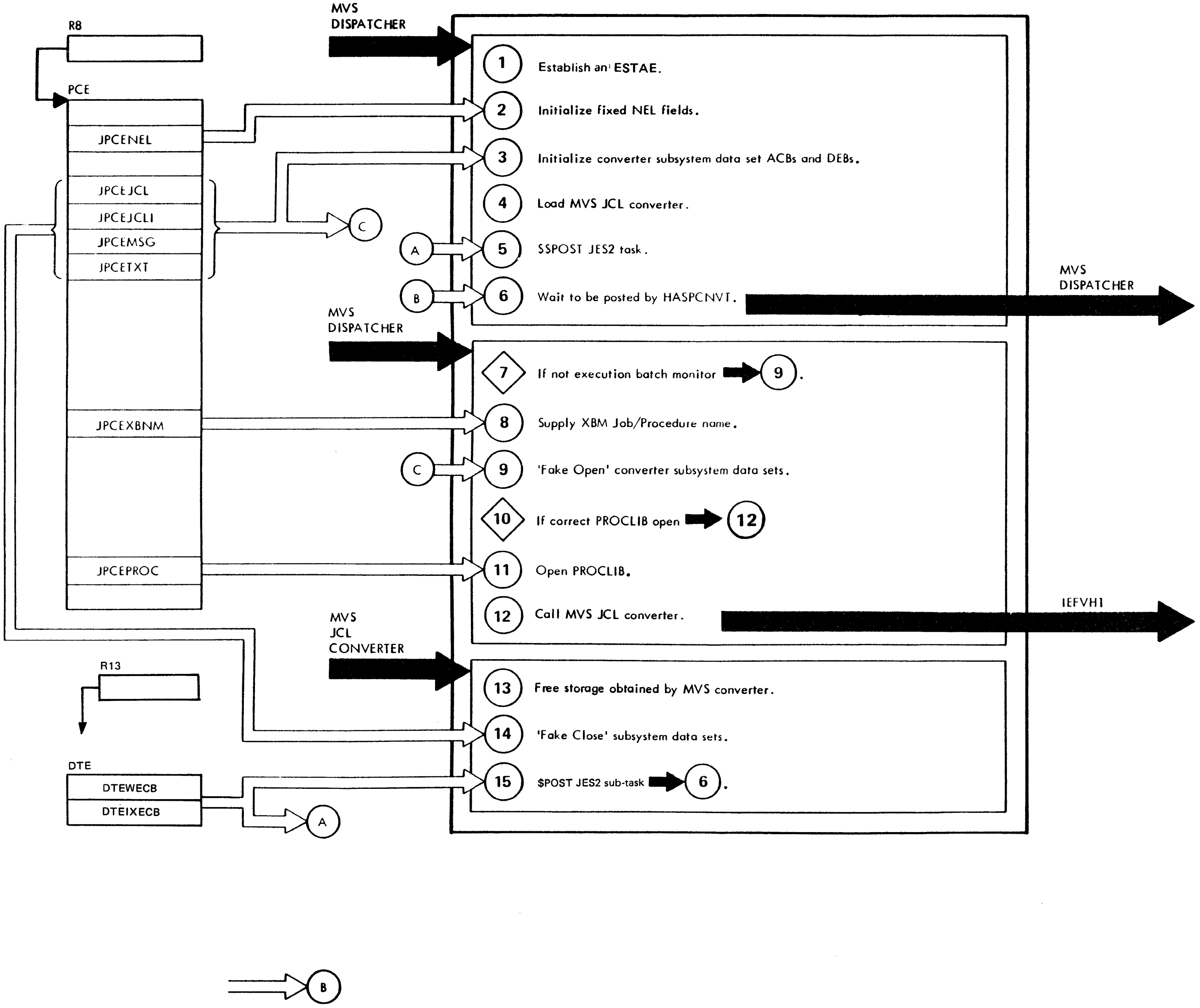


1. Obtain permanent buffer storage ➡ 5 .
2. If buffer storage page - released ➡ 4 .
3. Page - release permanent buffer storage.
4. $WAIT for job.
5. Select job. If none ➡ 2 .
6. Read job's JCT.
7. If conversion subtask attached ➡ 9 .
8. Attach conversion subtask (HOSCNVT). Wait for subtask to complete initialization.
9. Read job's IOT(s).
10. Initialize output PDDBs.
11. Initialize variable NEL fields.
12. Post conversion subtask.
13. $WAIT for work.
14. Write IOTs and JCT.
15. Determine next job queue. If $XEQ ➡ 17 .
16. Queue job for output ➡ 5 .
17. Queue job for execution ➡ 5 .

ASSEMBLY LISTING NAME	ASSEMBLY LISTING LABEL		NARRATIVE
HASPCNVT	HASPCNVT	1	Initial entry point. Obtain permanent storage, via GETMAIN, for use as buffers for JCT and IOT.
HASPCNVT	XCJGET	5	Attempt to get a job from JES2 job queue using $QGET.
HASPCNVT	XCPOST	12	Post JCL conversion subtask, HOSCNVT. Subtask is dispatched by MVS dispatcher.
HASPCNVT	none	14	Control resumes here following $WAIT. The $WAIT is satisfied when JES2 is $$POSTED by the conversion subtask at the conclusion of JCL conversion.
HASPCNVT	XCREQJOB	15	The queue in which the job is to be placed is determined according to job type, job run options and the results of JCL conversion.

JES2 JCL Conversion Processor Subtask

Input Process Output



1 Establish an ESTAE.

2 Initialize fixed NEL fields.

3 Initialize converter subsystem data set ACBs and DEBs.

4 Load MVS JCL converter.

5 $SPOST JES2 task.

6 Wait to be posted by HASPCNVT.

7 If not execution batch monitor → 9.

8 Supply XBM Job/Procedure name.

9 'Fake Open' converter subsystem data sets.

10 If correct PROCLIB open → 12

11 Open PROCLIB.

12 Call MVS JCL converter.

13 Free storage obtained by MVS converter.

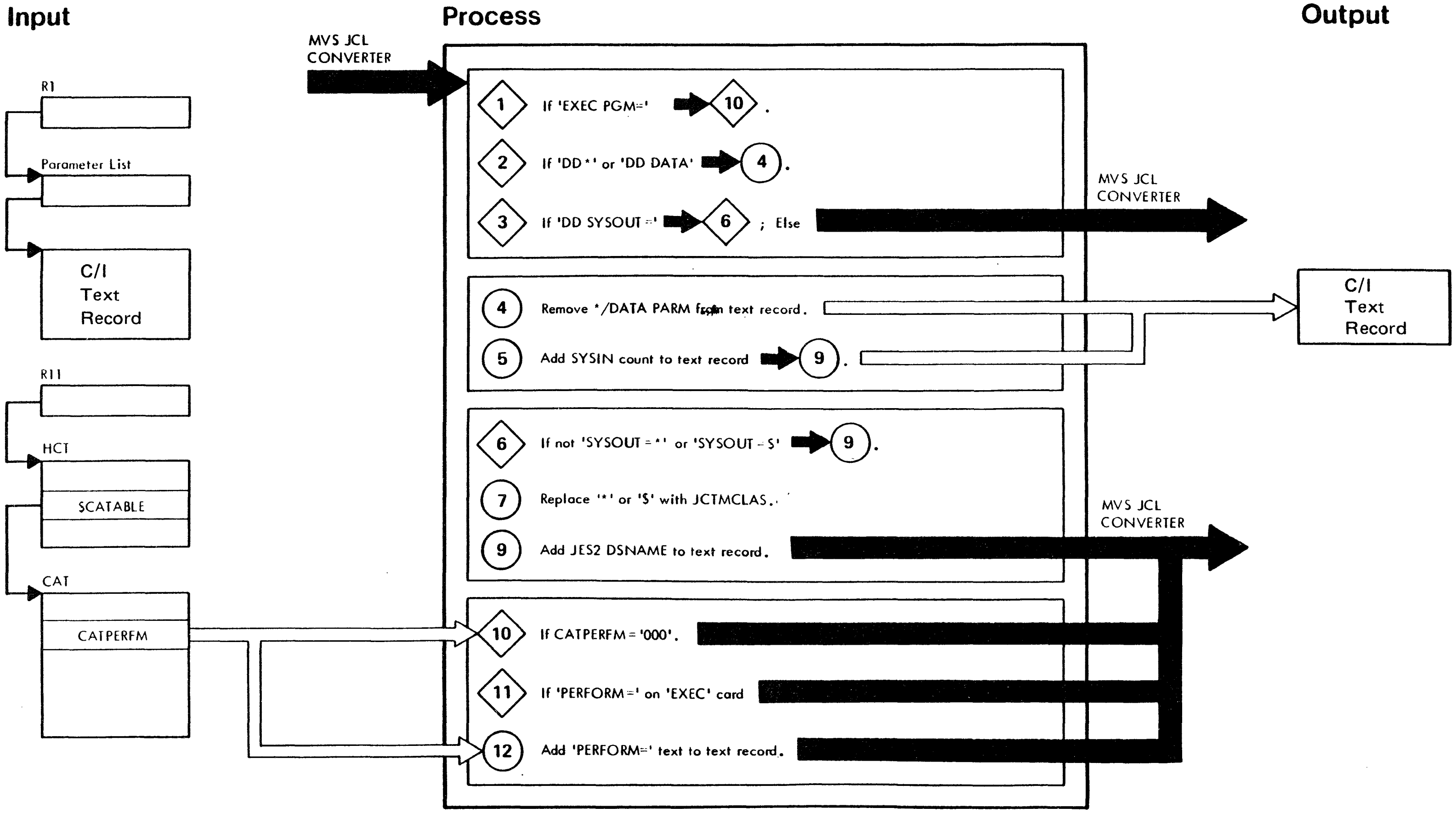
14 'Fake Close' subsystem data sets.

15 $POST JES2 sub-task → 6.

ASSEMBLY LISTING NAME	ASSEMBLY LISTING LABEL	NARRATIVE
HASPCNVT	HOSCNVT	1 Initial entry point to subtask. PCE address provided in R1 is saved in R8.
HASPCNVT		4 MVS JCL converter (entry point IEFVH1) loaded into storage. It will be deleted by STAE routine or JES2 shutdown.
HASPCNVT		6 Subtask will wait here until posted by the JES2 JCL conversion processor
HASPCNVT	XCNVFOPN	9 Control blocks associated with the internal text, JCL, JCL images, and system messages data sets are initialized by the 'fake open' routine in HASPSSSM.

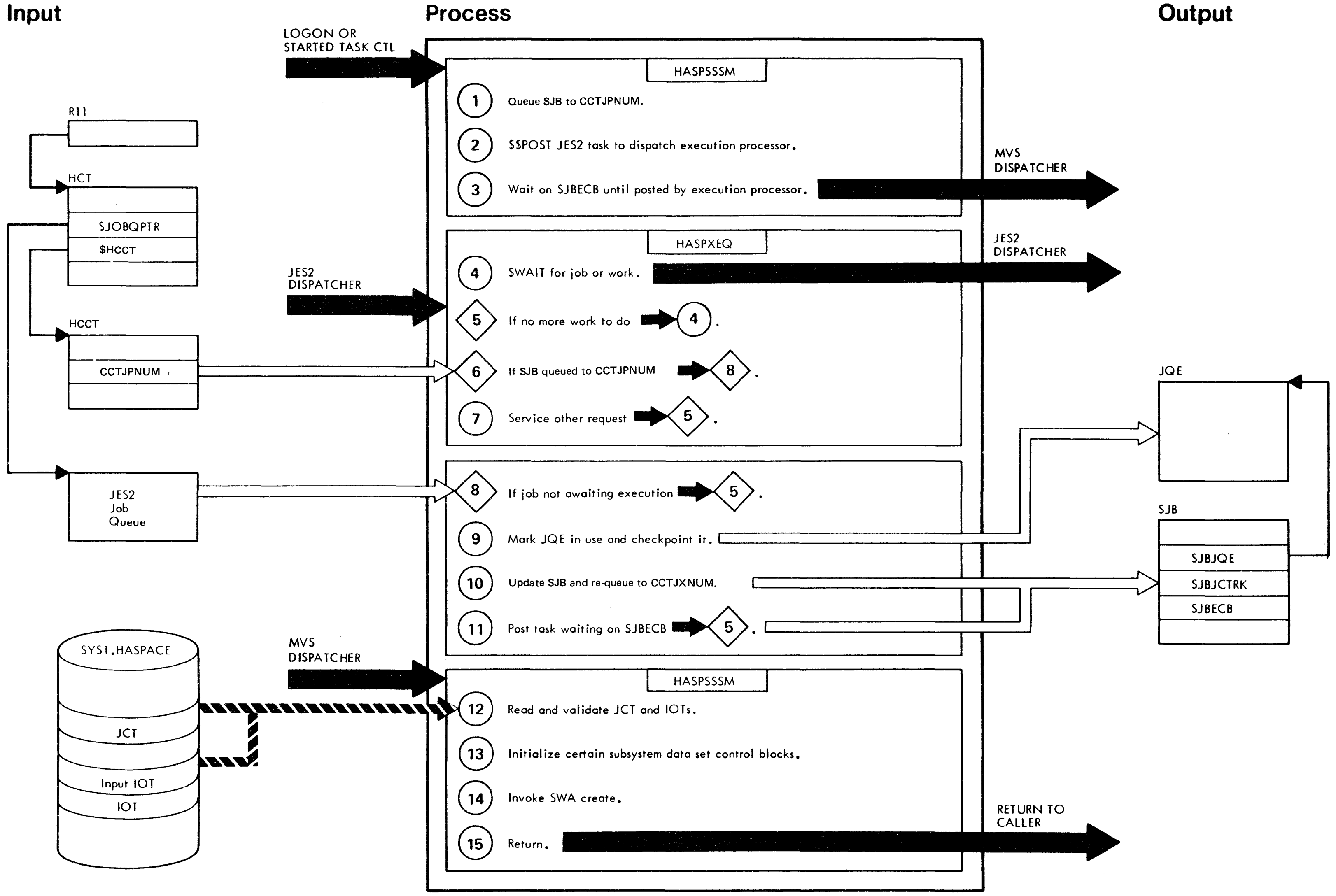
ASSEMBLY LISTING NAME	ASSEMBLY LISTING LABEL	NARRATIVE
HASPCNVT	none	10 The DDNAME of the PROCLIB required to process this job is checked against that of the currently opened PROCLIB.
HASPCNVT	XCNVCNV	12 Control is passed to the MVS JCL converter. The converter merges JCL from the input stream with JCL from PROCLIB, producing an internal text record for each resulting JCL record. The internal text records are processed as each is created by the routine beginning at XTXTEXIT in assembly HASPXEQ When all JCL has been processed, the converter returns. Exit point XCSTMUEE (for exit 6) is used after conversion is complete. Exit point XCSTCUET (for exit 6) is used after conversion is complete and JES2 has made its final changes.
HASPCNVT	XCNVPOST	15 The $$POST routine is used to post the converter subtask XECB.

JES2 Conversion Subtask Converter/Interpreter Text Edit


Input
Process
Output
MVS JCL CONVERTER
R1
Parameter List
C/I Text Record
R11
HCT
SCATABLE
CAT
CATPERFM
1 If 'EXEC PGM=' 10 .
2 If 'DD*' or 'DD DATA' 4 .
3 If 'DD SYSOUT=' 6 ; Else
MVS JCL CONVERTER
4 Remove */DATA PARM from text record.
5 Add SYSIN count to text record 9 .
C/I Text Record
6 If not 'SYSOUT=*' or 'SYSOUT=S' 9 .
7 Replace '*' or 'S' with JCTMCLAS.
9 Add JES2 DSNAME to text record.
MVS JCL CONVERTER
10 If CATPERFM='000'.
11 If 'PERFORM=' on 'EXEC' card
12 Add 'PERFORM=' text to text record.

ASSEMBLY LISTING NAME	ASSEMBLY LISTING LABEL	NARRATIVE
HASPCNVT	XDSKEY	(9) The DD number, which becomes part of the DSNAME, is obtained from the PDDB for SYSIN data sets. For SYSOUT data sets a number starting with 101 is incremented.

JES2 Execution Processor
(Job Selection by Job Number)



HASPSSSM

1 Queue SJB to CCTJPNUM.

2 $SPOST JES2 task to dispatch execution processor.

3 Wait on SJBECB until posted by execution processor.

HASPXEQ

4 $WAIT for job or work.

5 If no more work to do → 4.

6 If SJB queued to CCTJPNUM → 8.

7 Service other request → 5.

8 If job not awaiting execution → 5.

9 Mark JQE in use and checkpoint it.

10 Update SJB and re-queue to CCTJXNUM.

11 Post task waiting on SJBECB → 5.

HASPSSSM

12 Read and validate JCT and IOTs.

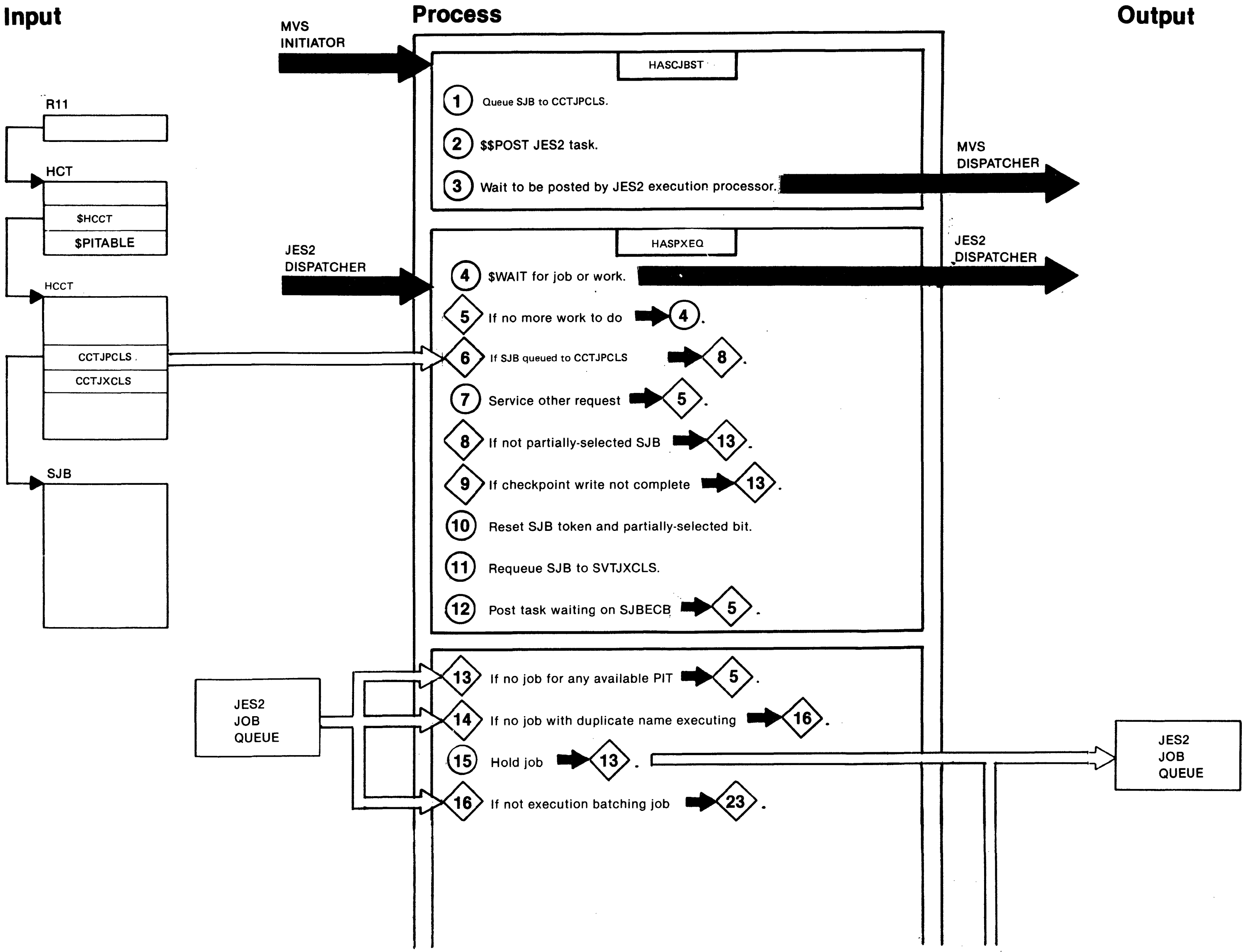
13 Initialize certain subsystem data set control blocks.

14 Invoke SWA create.

15 Return.

ASSEMBLY LISTING NAME	ASSEMBLY LISTING LABEL	NARRATIVE
HASCJBST	SSIJSEL	① The job number of the job being requested was provided to the caller by the subsystem earlier. For further details refer to JES2 input service processor.
HASPXEQ	XQSEARCH	◇5 The $$POST issued in HASCJBST causes the JES2 dispatcher to $POST the execution processor, which then searches all of its work queues beginning at XQSEARCH. HASPXEQ searches the cancel/status queues for a job and uses exit points ZTCSEXIT and YTCSEXIT (for exit 22) to aid in job selection.
HASPXEQ	XPOSTSJB	⑪ The task waiting in HASCJBST is posted to complete processing of the request for a specific job.

JES2 Execution Processor (Job Selection by Job Class)



1 Queue SJB to CCTJPCLS.

2 $$POST JES2 task.

3 Wait to be posted by JES2 execution processor.

4 $WAIT for job or work.

5 If no more work to do → 4.

6 If SJB queued to CCTJPCLS → 8.

7 Service other request → 5.

8 If not partially-selected SJB → 13.

9 If checkpoint write not complete → 13.

10 Reset SJB token and partially-selected bit.

11 Requeue SJB to SVTJXCLS.

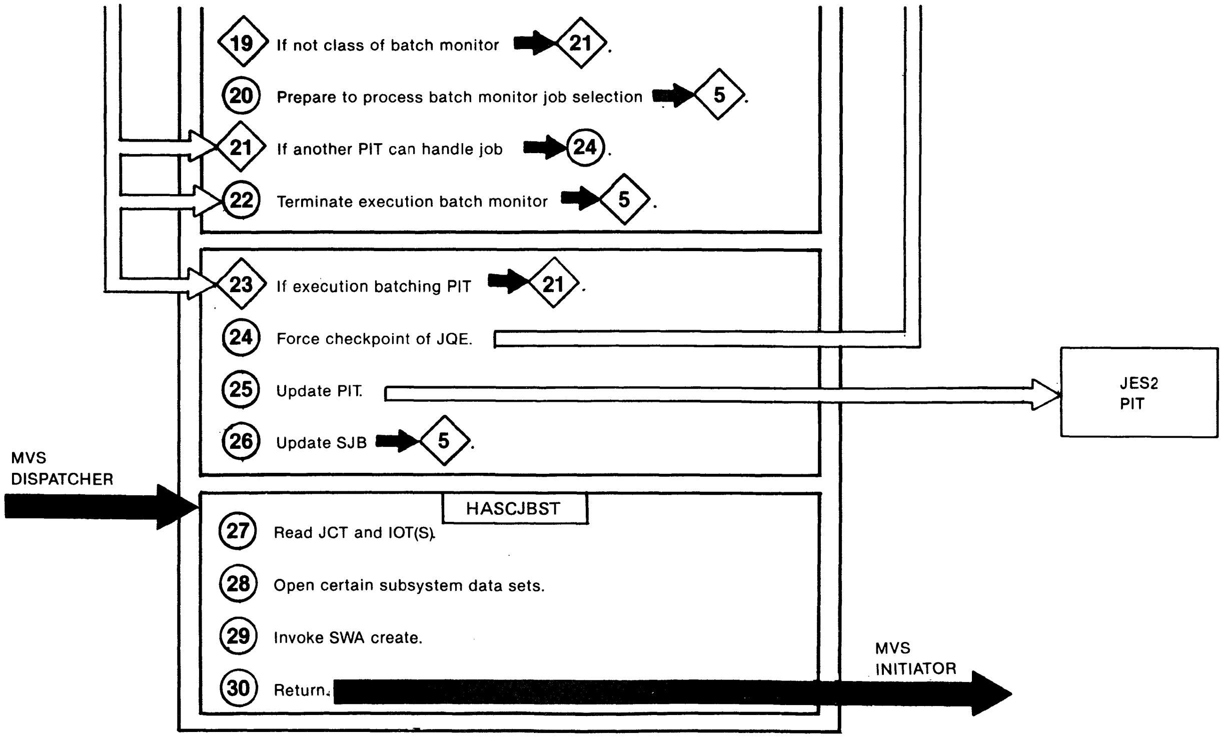
12 Post task waiting on SJBECB → 5.

13 If no job for any available PIT → 5.

14 If no job with duplicate name executing → 16.

15 Hold job → 13.

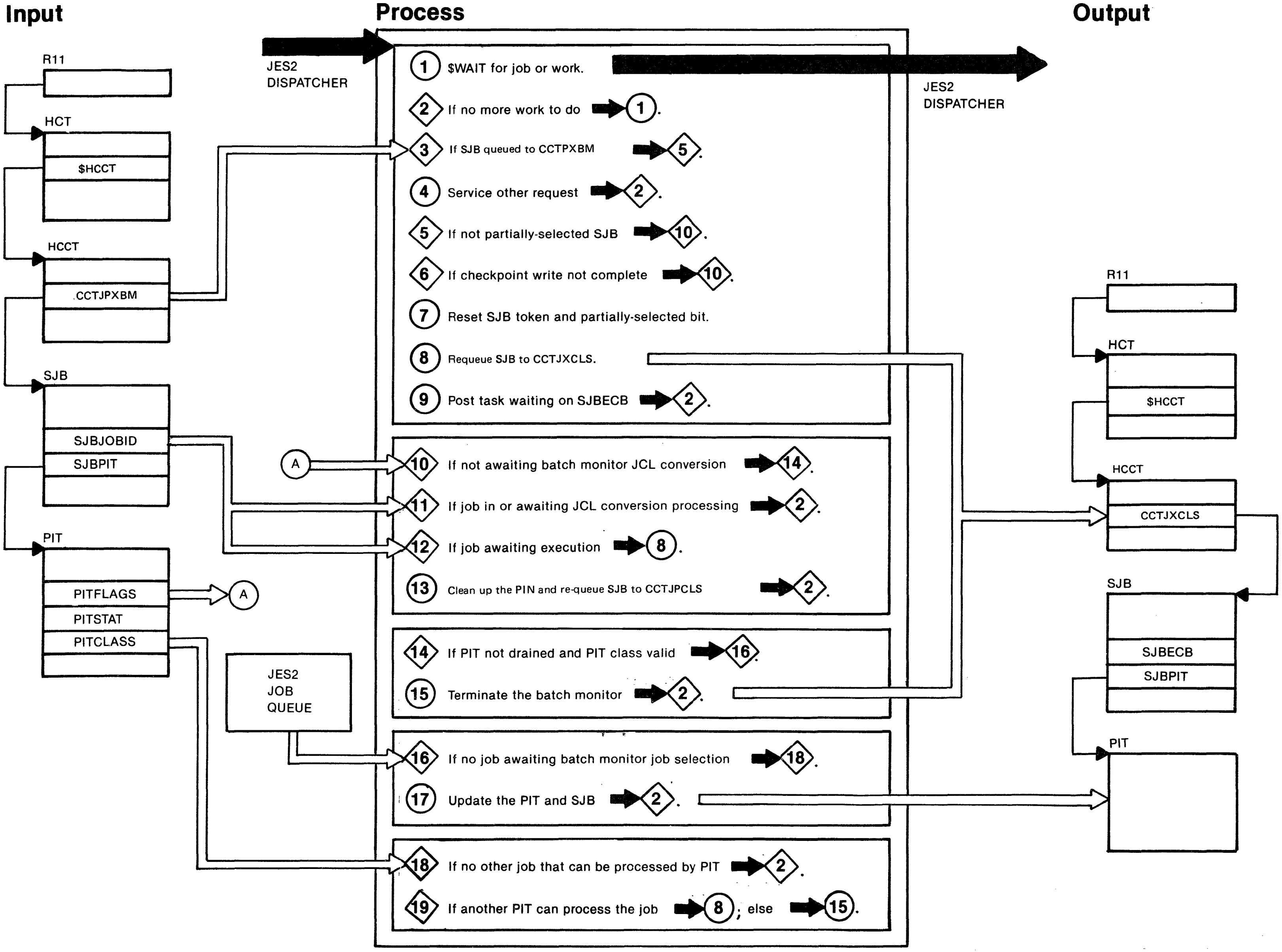
16 If not execution batching job → 23.


19 If not class of batch monitor 21.
20 Prepare to process batch monitor job selection 5.
21 If another PIT can handle job 24.
22 Terminate execution batch monitor 5.
23 If execution batching PIT 21.
24 Force checkpoint of JQE.
25 Update PIT.
26 Update SJB 5.
JES2 PIT
MVS DISPATCHER
HASCJBST
27 Read JCT and IOT(S).
28 Open certain subsystem data sets.
29 Invoke SWA create.
30 Return.
MVS INITIATOR

ASSEMBLY LISTING NAME	ASSEMBLY LISTING LABEL	NARRATIVE
HASCJBST	SSIJSEL	1 Request for job received from MVS initiator. An SJB is initialized to represent the request. It will later represent the job provided by HASPXEQ. The SJB is queued to CCTJPCLS.
HASPXEQ	XJBWAIT	4 The execution processor $WAITS here until $POSTed for job or work.
HASPXEQ	XQSEARCH	5 All of the queues serviced by the execution processor are searched beginning with the Spin/Hold queue and continuing until attempts have been made to satisfy all requests. HASPXEQ searches the cancel/status queues for a job and uses exit points ZTCSEXIT and YTCSEXIT (for exit 22) to aid in job selection.

ASSEMBLY LISTING NAME	ASSEMBLY LISTING LABEL	NARRATIVE
HASPXEQ	XPIT116	12 The MVS initiator was placed in the WAIT state (see Step 3) pending selection of a job. The job select routine in HASCJBST is now posted to complete the job select request processing.
HASPXEQ	XPBYCLS	13 The first available PIT containing a class for which a job awaiting execution is queued is selected. If no such PIT exists, the execution processor looks for other work to do.
HASPXEQ	XALTRNIT	21 In order to minimize the number of times an execution batch monitor is terminated, when a different class batch monitor job or a non-batch monitor job is selected for a batch monitor PIT, an attempt is made to find another PIT to process the job.
HASCJBST	JBFOUND	27 Using information provided by HASPXEQ, HASCJBST now reads and validates the job's JCT and IOT(S).

JES2 Execution Processor (Execution Batch Monitor Job Selection)



1 $WAIT for job or work.

2 If no more work to do → 1.

3 If SJB queued to CCTPXBM → 5.

4 Service other request → 2.

5 If not partially-selected SJB → 10.

6 If checkpoint write not complete → 10.

7 Reset SJB token and partially-selected bit.

8 Requeue SJB to CCTJXCLS.

9 Post task waiting on SJBECB → 2.

10 If not awaiting batch monitor JCL conversion → 14.

11 If job in or awaiting JCL conversion processing → 2.

12 If job awaiting execution → 8.

13 Clean up the PIN and re-queue SJB to CCTJPCLS → 2.

14 If PIT not drained and PIT class valid → 16.

15 Terminate the batch monitor → 2.

16 If no job awaiting batch monitor job selection → 18.

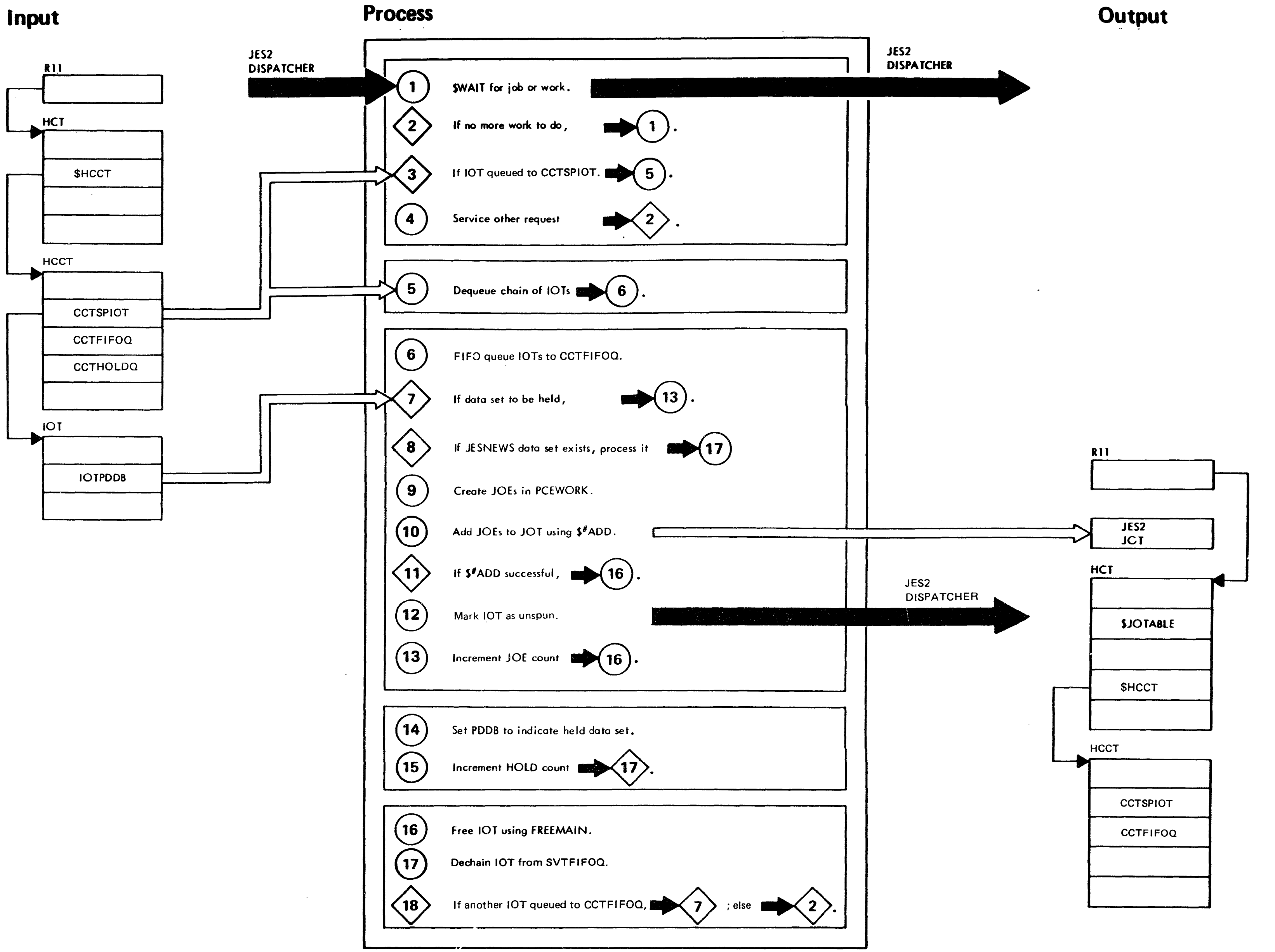
17 Update the PIT and SJB → 2.

18 If no other job that can be processed by PIT → 2.

19 If another PIT can process the job → 8; else → 15.

ASSEMBLY LISTING NAME	ASSEMBLY LISTING LABEL	NARRATIVE
HASPXEQ	XJQSRCH	◇3 The SJB for an execution batching job is re-queued to CCTJPXBM during termination processing of that job.
HASPXEQ	XPXBM	◇10 When an execution batching job is selected during job select by class processing and no batch monitor is available to process the job, the job is re-queued for JCL conversion processing.
HASPXEQ	XNOJCL	⑬ If a batch monitor job gets a non-zero return code from the MVS JCL converter it will not be re-queued for execution.
HASPXEQ	XKILLXBM	⑮ To KILL a batch monitor, the SJB is re-queued to CCTJXCLS leaving the SJB1XBWT bit on and the task waiting in HASPSSSM is posted.
HASPXEQ	XALTRNIT	◇19 A batch monitor can be automatically terminated if a job is selected for another class in its PITCLASS field. However, an attempt is first made to find another available PIT with the same class in its PITCLASS field.

JES2 Execution Processor (Spin Support)



1 $WAIT for job or work.

2 If no more work to do, ➡ 1.

3 If IOT queued to CCTSPIOT. ➡ 5.

4 Service other request ➡ 2.

5 Dequeue chain of IOTs ➡ 6.

6 FIFO queue IOTs to CCTFIFOQ.

7 If data set to be held, ➡ 13.

8 If JESNEWS data set exists, process it ➡ 17

9 Create JOEs in PCEWORK.

10 Add JOEs to JOT using $#ADD.

11 If $#ADD successful, ➡ 16.

12 Mark IOT as unspun.

13 Increment JOE count ➡ 16.

14 Set PDDB to indicate held data set.

15 Increment HOLD count ➡ 17.

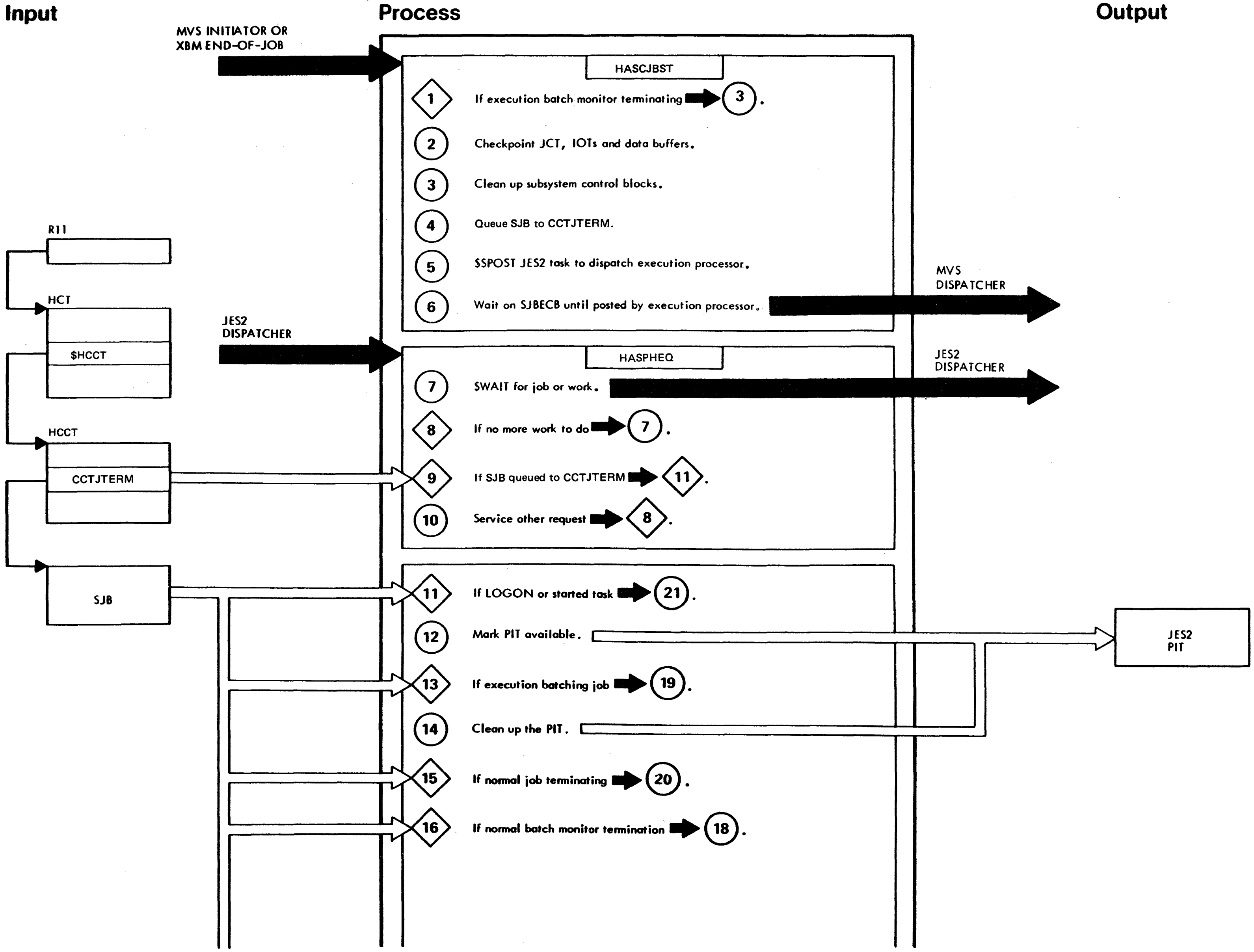
16 Free IOT using FREEMAIN.

17 Dechain IOT from SVTFIFOQ.

18 If another IOT queued to CCTFIFOQ, ➡ 7 ; else ➡ 2.

ASSEMBLY LISTING NAME	ASSEMBLY LISTING LABEL	NARRATIVE	
HASPXEQ	XJBWAIT	1	$WAIT for job to be $POSTed or for someone to $$POST the Execution Processor.
HASPXEQ	XQSEARCH	3	HASCDSAL queues an IOT to CCTSPIOT and $$POSTs the Execution Processor during data set unallocation if the data set is to be immediately spun.
HASPXEQ	XSPNHLD	6	The chain of IOTs that had been queued to CCTSPIOT is reordered in FIFO sequence and queued to CCTFIFOQ.
HASPXEQ		11	The IOT is written to SPOOL at the head of the queue pointed to by $UNSPUNQ using a compare and swap type logic.
HASPXEQ	XSPINBAD	12	HASPXEQ posts the output processor. The output processor writes the spin IOT to spool for processing later by the output processor.
HASPXEQ	XSPFRIOT	16	The IOT was originally obtained by HASCDSAL from common storage. Because HASCDSAL no longer needs the IOT and does not WAIT for the request to be satisfied, the IOT is now freed.

JES2 Execution Processor (Job Termination)



HASCJBST

1 If execution batch monitor terminating → 3.

2 Checkpoint JCT, IOTs and data buffers.

3 Clean up subsystem control blocks.

4 Queue SJB to CCTJTERM.

5 $SPOST JES2 task to dispatch execution processor.

6 Wait on SJBECB until posted by execution processor.

HASPHEQ

7 $WAIT for job or work.

8 If no more work to do → 7.

9 If SJB queued to CCTJTERM → 11.

10 Service other request → 8.

11 If LOGON or started task → 21.

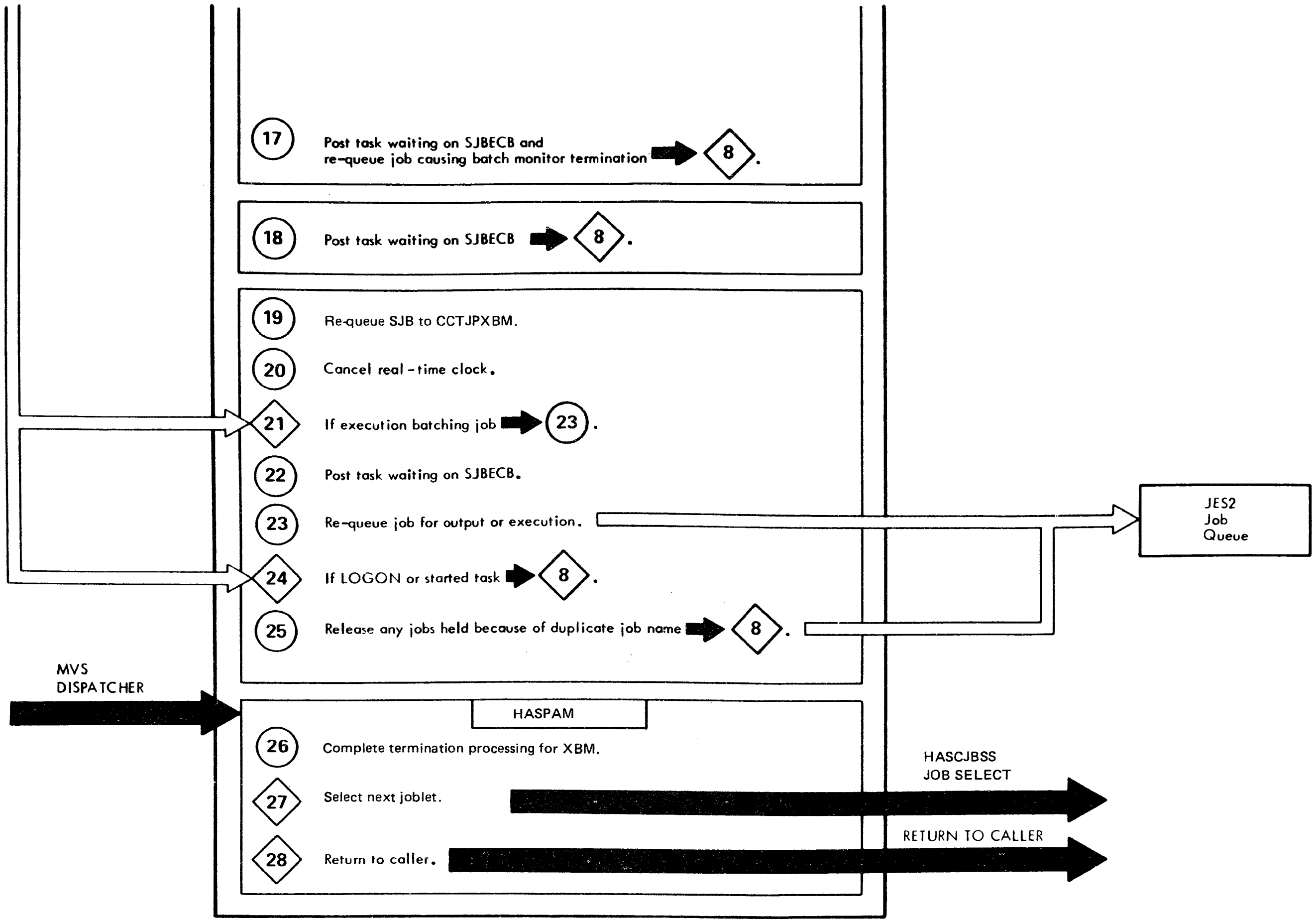
12 Mark PIT available.

13 If execution batching job → 19.

14 Clean up the PIT.

15 If normal job terminating → 20.

16 If normal batch monitor termination → 18.



17 Post task waiting on SJBECB and re-queue job causing batch monitor termination ➡ 8.

18 Post task waiting on SJBECB ➡ 8.

19 Re-queue SJB to CCTJPXBM.

20 Cancel real-time clock.

21 If execution batching job ➡ 23.

22 Post task waiting on SJBECB.

23 Re-queue job for output or execution.

24 If LOGON or started task ➡ 8.

25 Release any jobs held because of duplicate job name ➡ 8.

26 Complete termination processing for XBM.

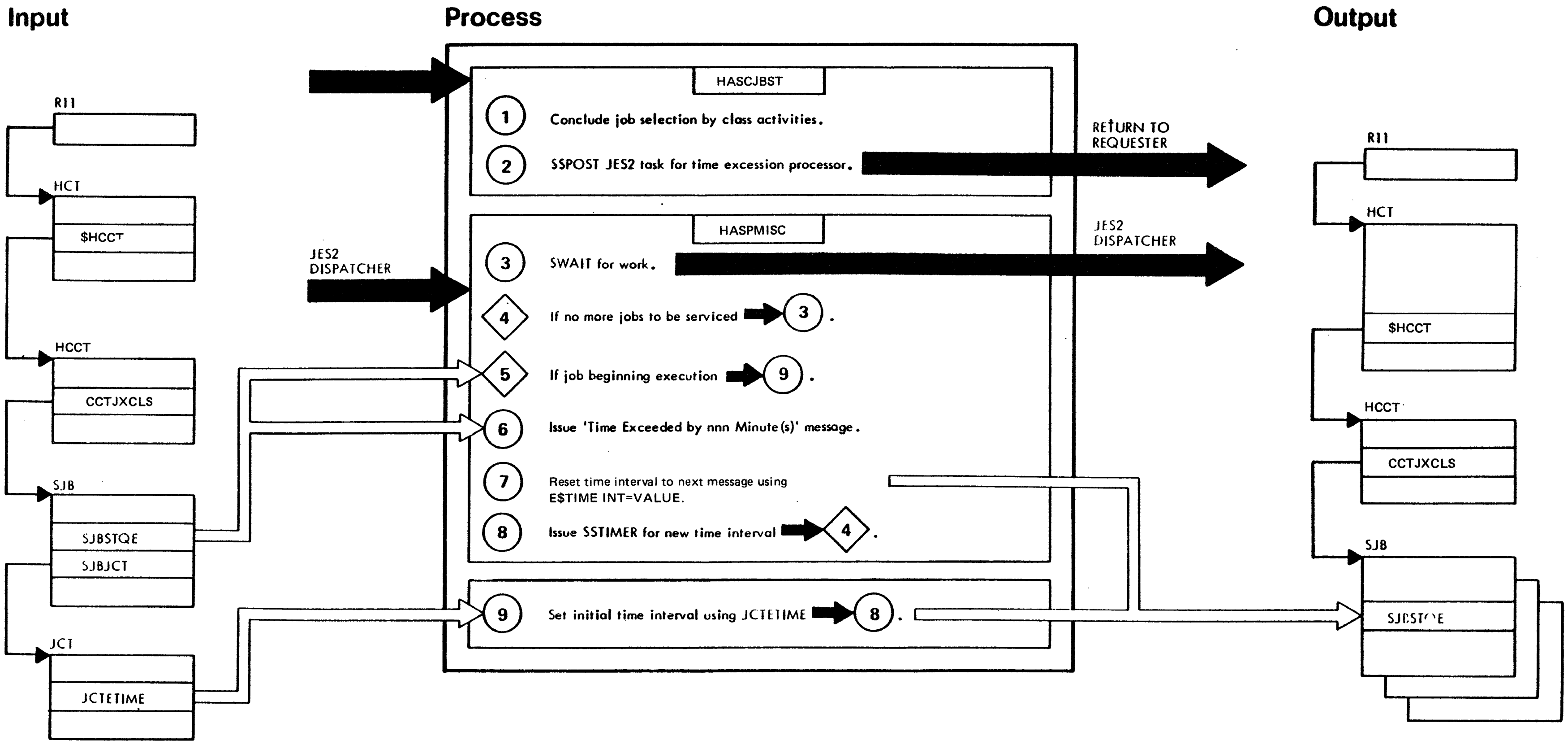
27 Select next joblet.

28 Return to caller.

ASSEMBLY LISTING NAME	ASSEMBLY LISTING LABEL	NARRATIVE
HASCJBST	SSIJTERM	1 HASCJBST termination processing may be entered from HASCJBST job selection to terminate an idle execution batch monitor. Otherwise termination is for a LOGON, started task, or normal batch job.
HASPXEQ	XKILLXBM	16 Execution processor select job by class processing may have caused a batch monitor to be terminated in order to run a normal job or another batch monitor.

ASSEMBLY LISTING NAME	ASSEMBLY LISTING LABEL	NARRATIVE
HASPXEQ	XREQSJB	19 Normal execution batching job termination is transformed into a request for another execution batching job.

JES2 Time Excess Processor



1 Conclude job selection by class activities.

2 SSPOST JES2 task for time excession processor.

3 SWAIT for work.

4 If no more jobs to be serviced → 3.

5 If job beginning execution → 9.

6 Issue 'Time Exceeded by nnn Minute(s)' message.

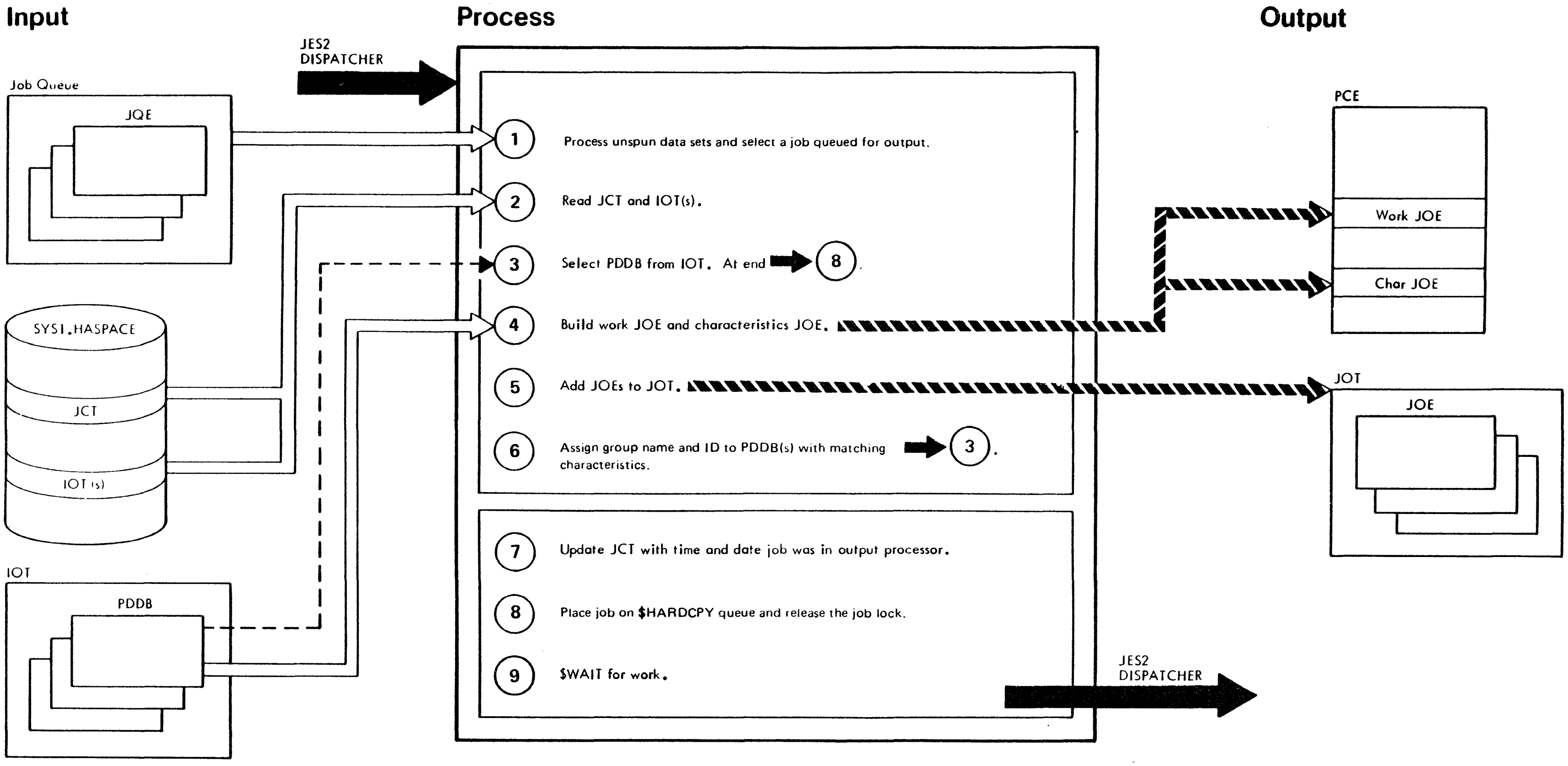
7 Reset time interval to next message using E$TIME INT=VALUE.

8 Issue SSTIMER for new time interval → 4.

9 Set initial time interval using JCTETIME → 8.

ASSEMBLY LISTING NAME	ASSEMBLY LISTING LABEL	NARRATIVE
HASCJBST	HJS570	① After the execution processor selects a job by class and posts the waiting task, HASCJBST concludes job select processing prior to returning to the originator of the job select request.
HASPMISC	HASPTIME	◇4 The queue of SJBS representing jobs in execution by class is searched for an SJB whose timer queue element has been posted or whose timer interval indicates a job beginning execution.

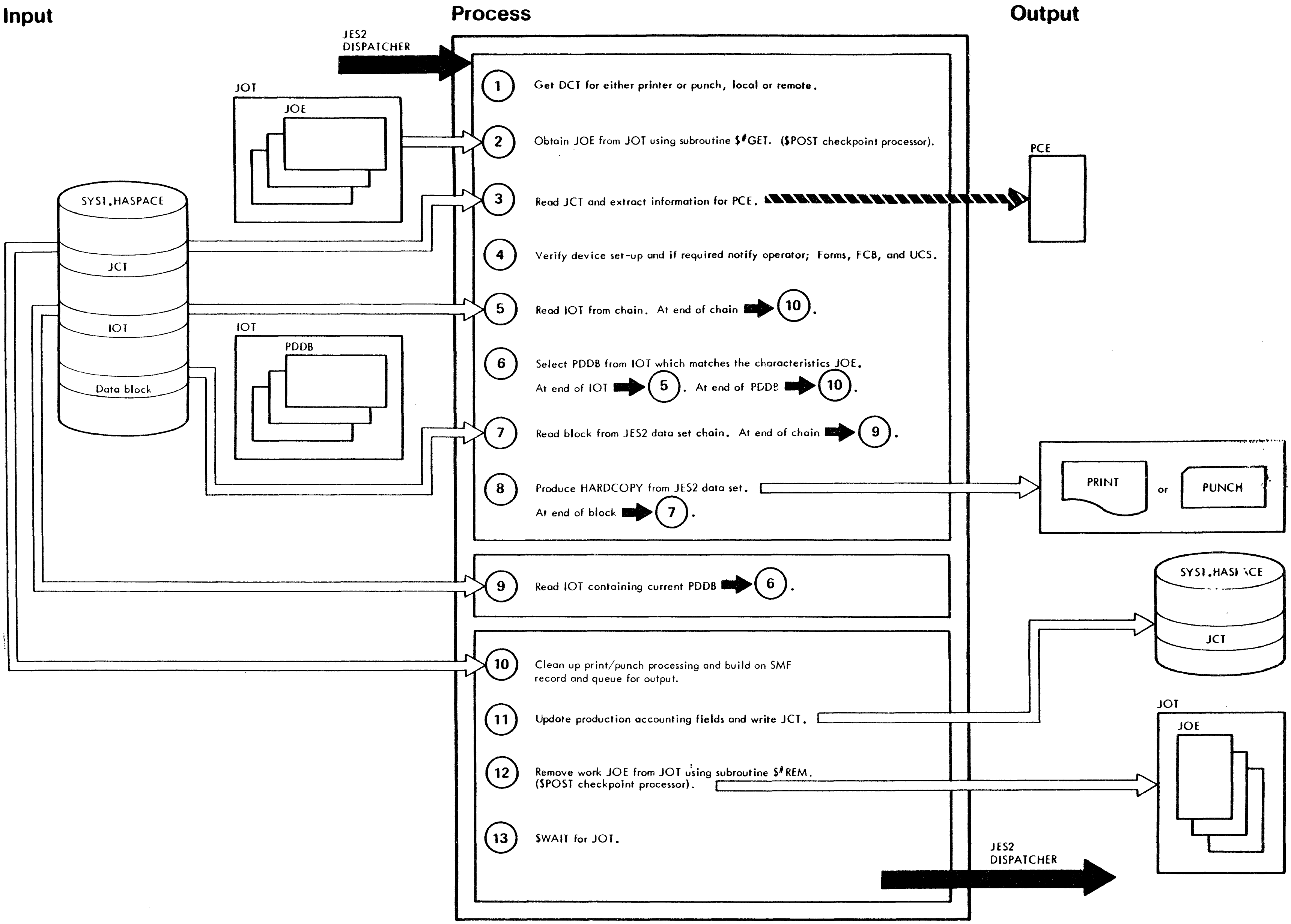
JES2 Output Processor



1. Process unspun data sets and select a job queued for output.
2. Read JCT and IOT(s).
3. Select PDDB from IOT. At end → 8.
4. Build work JOE and characteristics JOE.
5. Add JOEs to JOT.
6. Assign group name and ID to PDDB(s) with matching characteristics. → 3.
7. Update JCT with time and date job was in output processor.
8. Place job on $HARDCPY queue and release the job lock.
9. $WAIT for work.

ASSEMBLY LISTING NAME	ASSEMBLY LISTING LABEL	NARRATIVE
HASPPRPU	OPINIT OPQGET	1 Unspun data sets are processed before a job is selected. When it is time to select a job $QGET is used. If the $QGET routine cannot obtain a job queued for the output processor HASPPRPU issues a $WAIT.
HASPPRPU	OPJOB OPNEXIT	2 The JOB lock is obtained and the DASD address of the first IOT is obtained from the JCT and all IOT(s) are read into storage. The TSO notification message is built to notify the TSO user of output processing for the job. A JES2 installation exit is invoked at label OPNEXIT (for Exit 16) to allow installation modification of the notification message. When the exit routine returns, the notification message is sent to the TSO user (via $WTO).
HASPPRPU	OPPDBIOT	3 The first PDDB whose null bit is not on is selected.
HASPPRPU	OPPDBJOE	4 A Work-JOE and a Characteristics-JOE are built in the PCE using the information in the selected PDDB.
HASPPRPU	OPJOTADD	5 If insufficient space exists in the JOT to add the new JQE, the processor $WAITs for a free JOE. The processor tries again to add the new JOE to the JOT until it is successful. The number of job copies requested is the number of duplicate work JOEs placed in the JOT.
HASPPRPU	OPDBNEXT	6 Each PDDB is marked as HOLD=YES (if applicable); the processor sets the appropriate PRMODE and checks if the PDDB can be added to the JOE. If the PDDB can be added, the PDDB is updated with the group name and JOE id. If demand setup is not allowed, then only those PDDBs with matching SYSOUT class, destinations, PRMODE, group ids, writer names, FORMS, FCB, and UCS can be gathered into the same job.
HASPPRPU	OPQPUT	7 The sign-on and sign-off time and date fields are updated, and the JCT is written back to the SPOOL volume and the job lock is released.
HASPPRPU	OPNOJCT	8 If the job is marked for purging the job is placed on the $PURGE queue. The $HARDCPY queue is designed to hold jobs while their SYSOUT records are being converted to hardcopy.

JES2 Print/Punch Processor



1 Get DCT for either printer or punch, local or remote.

2 Obtain JOE from JOT using subroutine $#GET. ($POST checkpoint processor).

3 Read JCT and extract information for PCE.

4 Verify device set-up and if required notify operator; Forms, FCB, and UCS.

5 Read IOT from chain. At end of chain ➡ 10.

6 Select PDDB from IOT which matches the characteristics JOE. At end of IOT ➡ 5. At end of PDDB ➡ 10.

7 Read block from JES2 data set chain. At end of chain ➡ 9.

8 Produce HARDCOPY from JES2 data set. At end of block ➡ 7.

9 Read IOT containing current PDDB ➡ 6.

10 Clean up print/punch processing and build on SMF record and queue for output.

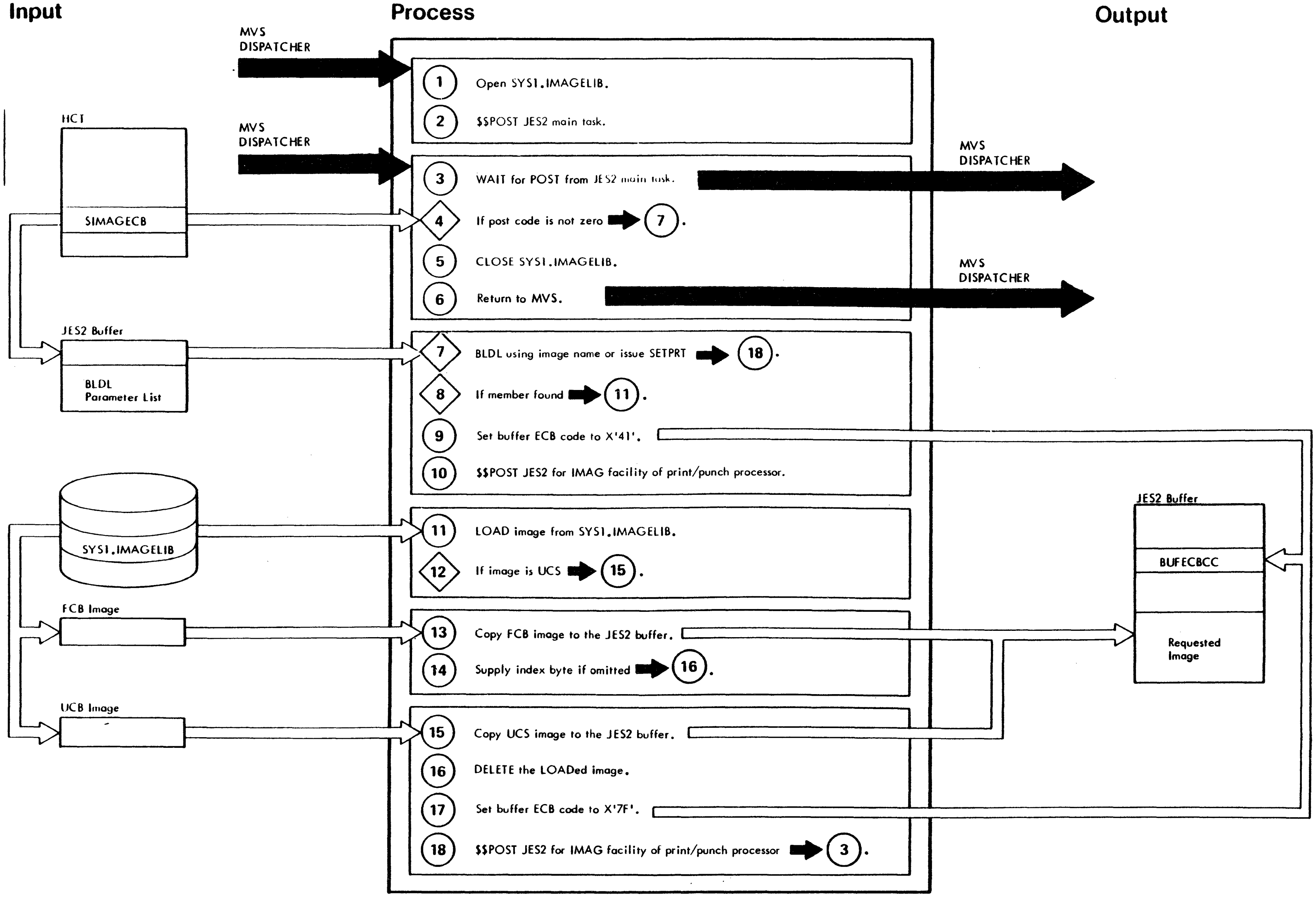
11 Update production accounting fields and write JCT.

12 Remove work JOE from JOT using subroutine $#REM. ($POST checkpoint processor).

13 $WAIT for JOT.

ASSEMBLY LISTING NAME	ASSEMBLY LISTING LABEL		NARRATIVE
HASPPRPU	HASPPI1 PGOTUNIT	1	If a requested DCT is not available the Print/Punch processor $WAITs. For a local printer or punch, the print/punch processor issues a $WAIT for UNIT. For a remote terminal, the print/punch processor issues a $WAIT for WORK (if the remote terminal is not signed on); otherwise, a $WAIT for JOT is issued. When the printer or punch becomes available, the print/punch processor prepares to use it.
HASPPRPU	PGETJOB	2	The $POST of the checkpoint processor is performed within the $#GET subroutine. If no work is available the print/punch processor $WAITs for JOT.
HASPPRPU	PJCTREAD	3	The $#JCT macro is used so that multiple processors may share the JCT for a job. The print/punch processor attaches the image loading subtask for non-impact printers and waits ($WAIT IMAG) for the subtask to initialize itself.
HASPPRPU	PCLDSTRT	4	If the device requires set up the processor informs the operator and $WAITs for I/O which will be $POSTed by the command processor when the operator responds with $START device.
HASPPRPU	PENDINIT	5	A job may have a chain of IOTs. Each IOT in the chain is processed until the chain pointer is zero.
HASPPRPU	PPPDB	6	Each PDDB represents a chain of HASP data records which contain related SYSOUT data. the field named PDBMTTR conveys the starting MTTR for each set. Use exit point PPEXIT (for exit 15) to optionally alter the number of copies of the data set that is to be printed. Use exit point PCEXIT (for exit 15) to optionally create a data set separator page.
HASPPRPU	PNXTCCW	8	A multi-command channel program is built in the PCE and executed to produce hardcopy. Use exit point PEXITSC (for exit 1) to create or modify the exiting header page. Use exit point PEXITTR (for exit 1) to create or modify the existing trailer page.
HASPPRPU	PPIOTCK	9	The IOT containing the PDDB just processed is read to restart the PDDB search/match sequence.
HASPPRPU	PPABORT	10	Save area clean up is performed, the recovery environment is canceled (via $ESTAE), and the type 6 SMF record is generated. A Type-6 SMF record is built representing the amount of work performed in behalf of the related job. For a 3800 printer, the SMF record is not written for the 3800 printer output until the output has completely reached the 38000 stacker.
HASPPRPU	PPJOEIO2	12	The job will be placed on the $PURGE queue if its number of work JQEs on the JOT equals zero. For a 3800 printer, the work JQE for the output is not removed until all output for the JQE has reached the 3800 stacker.
HASPPRPU	PNOJOB	13	If another work-JOE cannot be obtained from the JOT the processor $WAITs for JOT.

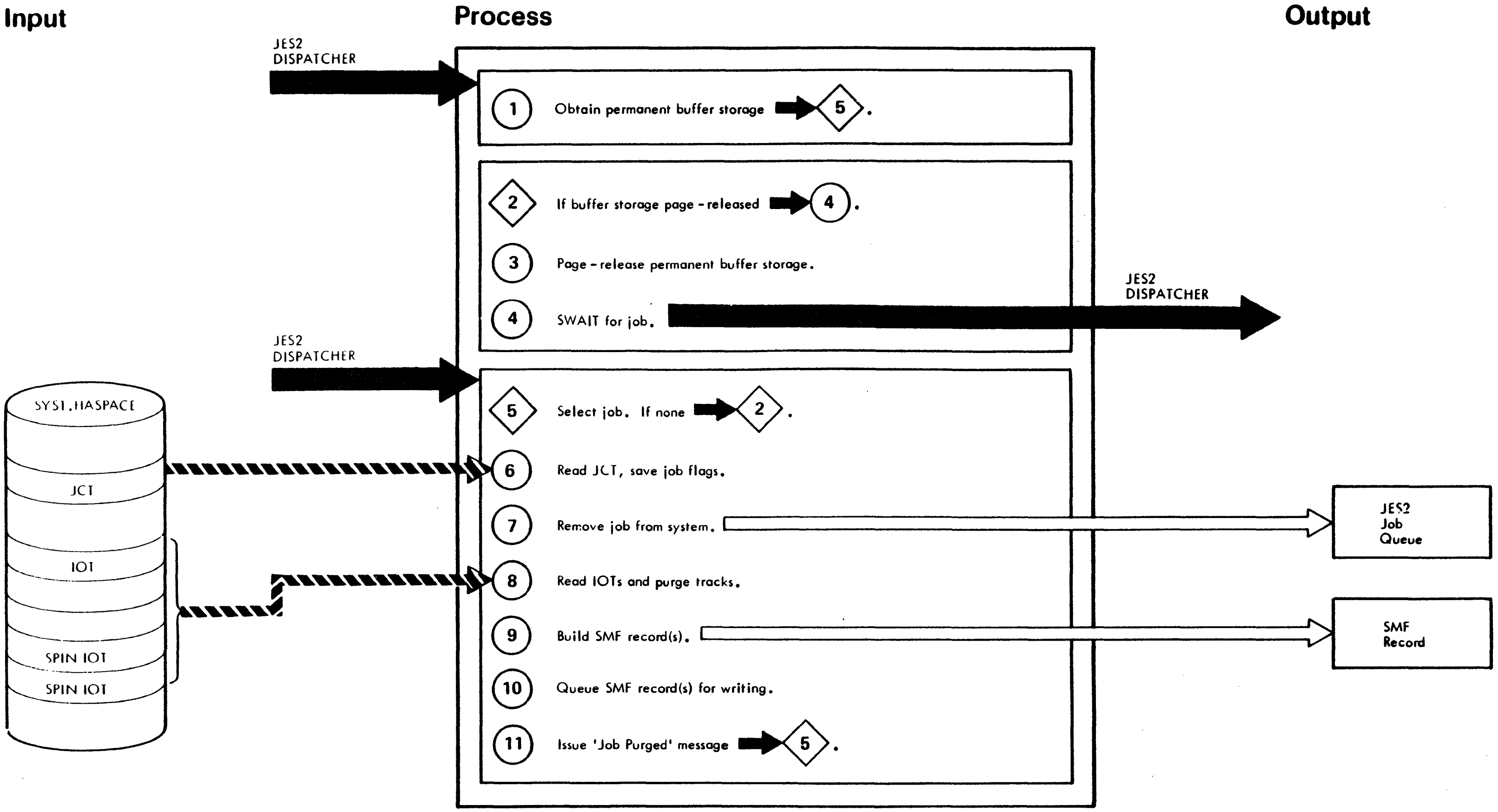
JES2 UCS/FCB Image Loader Subtask



1. Open SYS1.IMAGELIB.
2. $$POST JES2 main task.
3. WAIT for POST from JES2 main task.
4. If post code is not zero ➡ 7.
5. CLOSE SYS1.IMAGELIB.
6. Return to MVS.
7. BLDL using image name or issue SETPRT ➡ 18.
8. If member found ➡ 11.
9. Set buffer ECB code to X'41'.
10. $$POST JES2 for IMAG facility of print/punch processor.
11. LOAD image from SYS1.IMAGELIB.
12. If image is UCS ➡ 15.
13. Copy FCB image to the JES2 buffer.
14. Supply index byte if omitted ➡ 16.
15. Copy UCS image to the JES2 buffer.
16. DELETE the LOADed image.
17. Set buffer ECB code to X'7F'.
18. $$POST JES2 for IMAG facility of print/punch processor ➡ 3.

ASSEMBLY LISTING NAME	ASSEMBLY LISTING NAME		NARRATIVE
HASPPRPU	HASPIMAG	1	Initial entry point of subtask. The ESTAE macro is issued to establish a recovery environment for the image loader subtask. The image name is verified for the printer using SETPRT. The MVS macro IMGLIB is used to acquire a DCB address for SYS1.IMAGELIB.
		2	Using $$POST, signal the JES2 main task that the subtask is initialized and active.
HASPPRPU	IMGWAIT	3	WAIT to be posted by JES2 main task (post code is zero or a buffer address).
		5	The ESTAE macro is issued to cancel the recovery environment for the image loader subtask. MVS macro IMGLIB is used to close SYS1.IMAGELIB if post code is zero.
		6	Subtask terminates with a completion code of zero.
HASPPRPU	IMGBLDL	7	BLDL is issued to determine if the requested image is a valid member of SYS1.IMAGELIB. For a 3800 printer a SETPRT is issued (SVC 81) instead of a BLDL to initialize for the 3800. For the 3800, control goes to step 18; otherwise, control goes to step 8. to step 8. For the 4245 printer, a SETPRT is issued (SVC 81) instead of a BLDL to initialize. For this printer, control goes to step 18; otherwise, control goes to label RMGVCSVF to verify the image name.
		9	If the requested image is not found, an error post code is placed in the JES2 buffer.
		10	$$POST is used to activate the JES2 main task and signal the error.
HASPPRPU	IMGLOAD	11	The requested image member is LOADed from SYS1.IMAGELIB.
		12	Based on the first character of the image name, either FCB or UCS processing is selected.
HASPPRPU	IMGFCB	13	Image is copied from the LOADed member to a JES2 buffer.
	IMGUCSVF	14	If the first data byte of the image is not a print position index, a default of X'81' is supplied. The X'81' is not used if the device is a 3800 printer; for this printer, the 3800 FCB image is copied to the JES2 main task's buffer.
HASPPRPU	IMGUCSCP	15	Image is copied from the LOADed member to a JES2 buffer.
HASPPRPU	IMGDELET	16	DELETE the LOADed image member.
		17	The buffer post code is set to X'7F'.
		18	$$POST used to alert the JES2 task of successful image loading.

JES2 Purge Processor



1 Obtain permanent buffer storage ➡ 5.

2 If buffer storage page-released ➡ 4.

3 Page-release permanent buffer storage.

4 SWAIT for job.

5 Select job. If none ➡ 2.

6 Read JCT, save job flags.

7 Remove job from system.

8 Read IOTs and purge tracks.

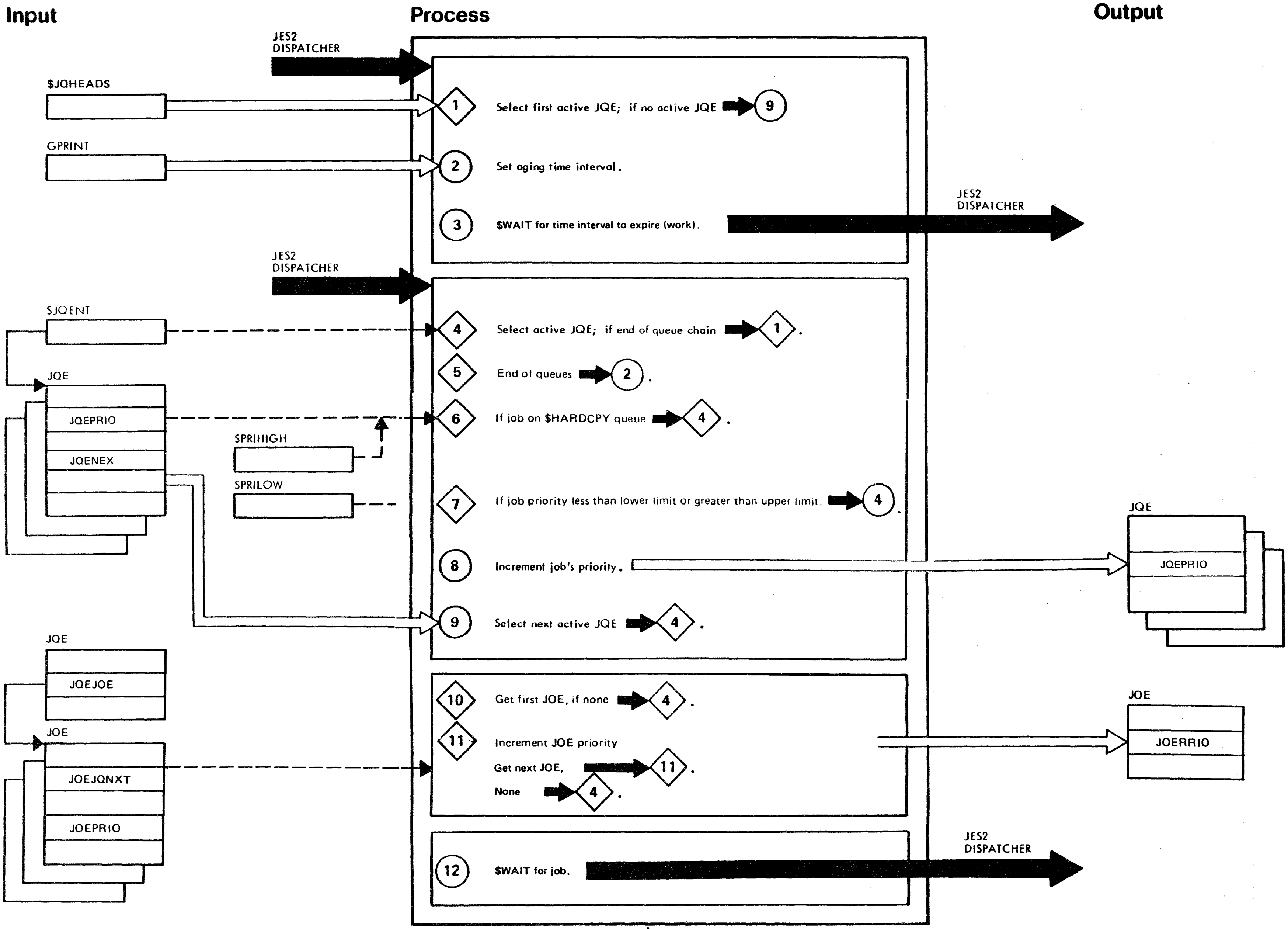
9 Build SMF record(s).

10 Queue SMF record(s) for writing.

11 Issue 'Job Purged' message ➡ 5.

ASSEMBLY LISTING NAME	ASSEMBLY LISTING LABEL	NARRATIVE
HASPTRAK	HASPVPRG	1 Use temporary GETMAINed storage for use as buffers for JCT and IOT. Storage is returned when processor goes inactive.
HASPTRAK	VGETJOB	5 Attempt to get a job from JES2 job queue using SQGET.
HASPTRAK	VREMJOB	7 Remove job from system using SQREM.
HASPTRAK	VIOTSPCL	8 The local subroutine VIOTPRG is used to read the job's SPIN, IOTs and purge, using PURGE, the tracks indicated in their track group maps.
HASPTRAK	VSMFPRG	9 If EXT = yes was specified as an SMF parameter to OS, two SMF buffers (Type 26 record and JMR) are build. Else just a Type 26 record is built.

JES2 Priority Aging Processor



1 Select first active JQE; if no active JQE → 9

2 Set aging time interval.

3 $WAIT for time interval to expire (work).

4 Select active JQE; if end of queue chain → 1.

5 End of queues → 2.

6 If job on $HARDCPY queue → 4.

7 If job priority less than lower limit or greater than upper limit. → 4.

8 Increment job's priority.

9 Select next active JQE → 4.

10 Get first JOE, if none → 4.

11 Increment JOE priority

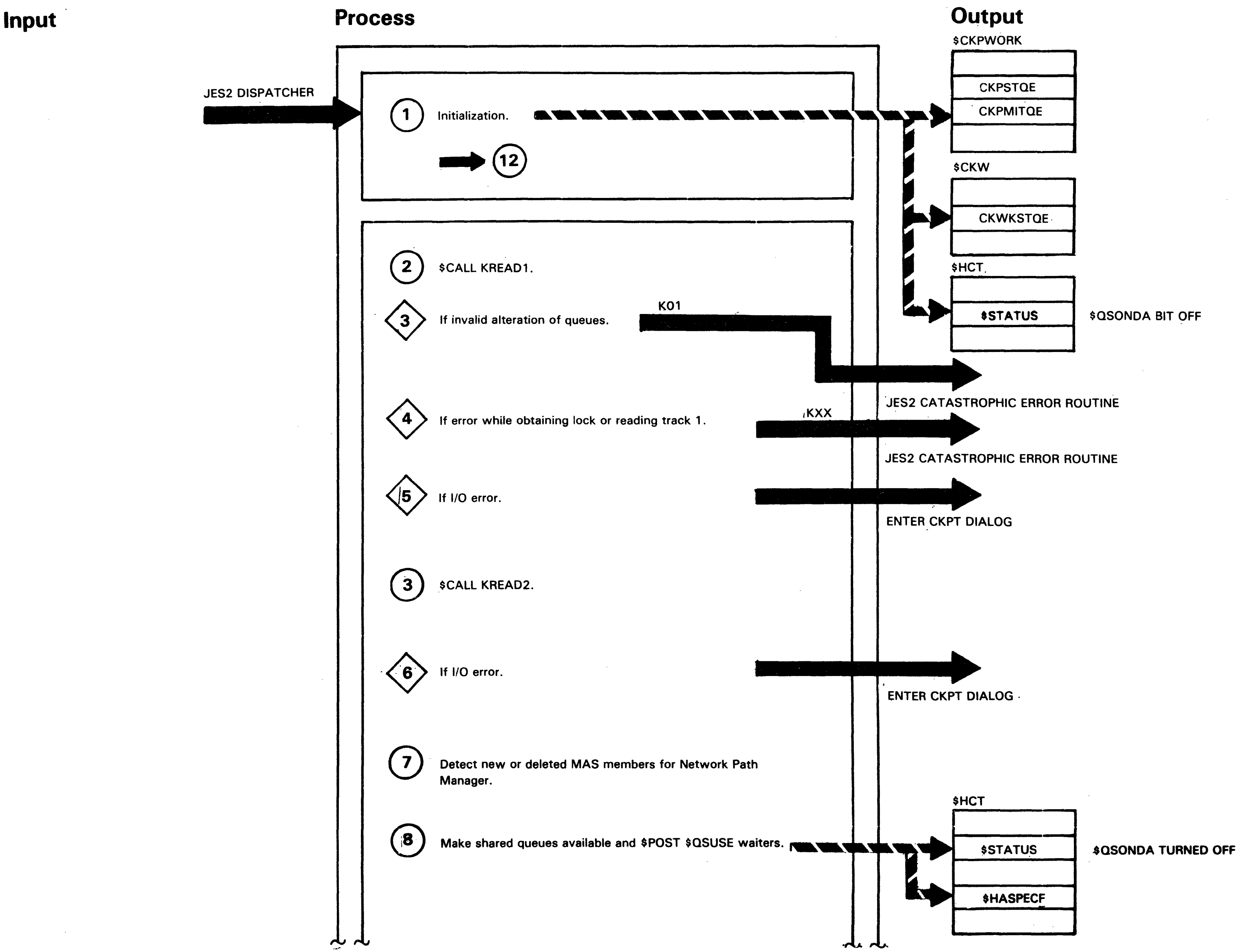
Get next JOE, → 11.

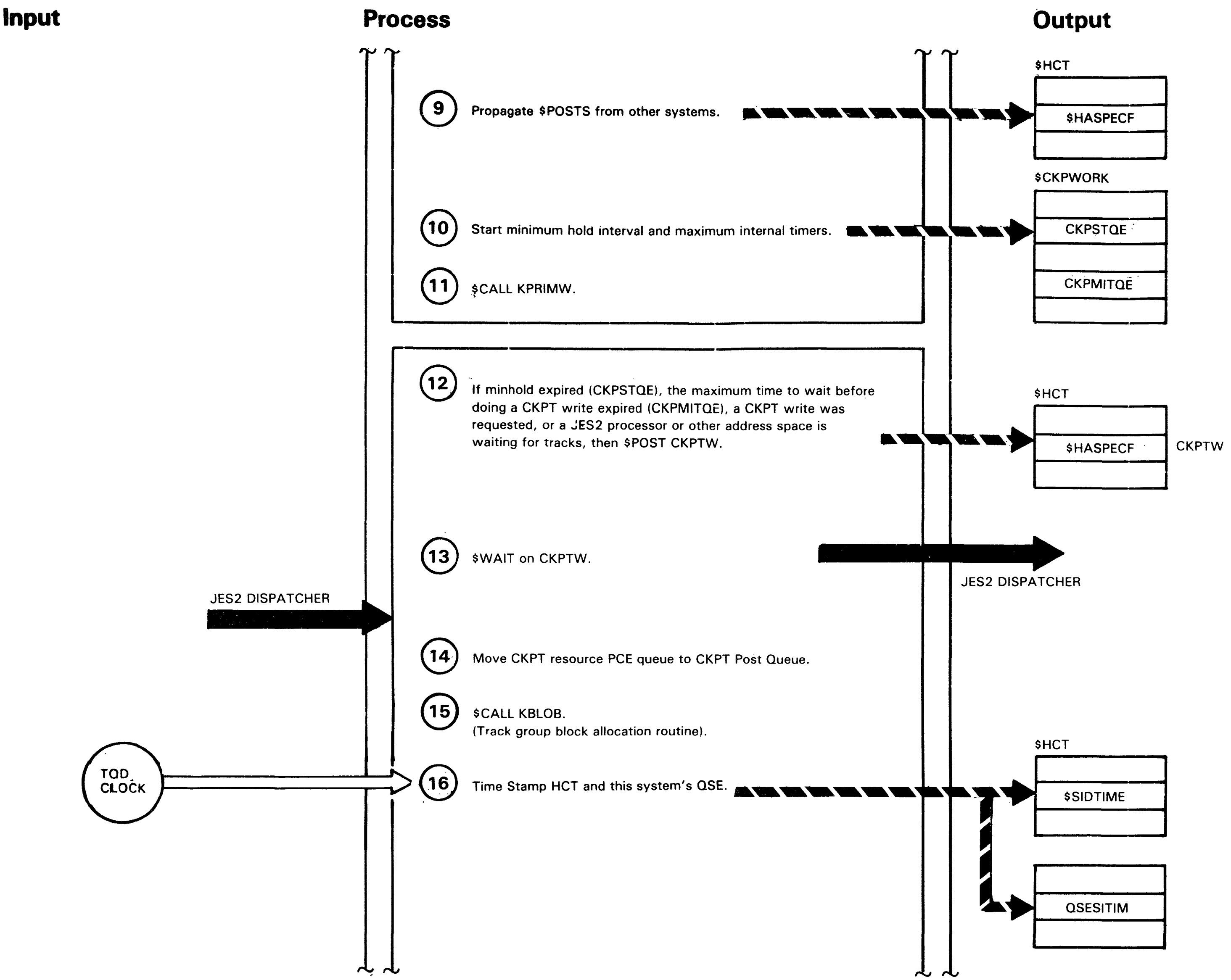
None → 4.

12 $WAIT for job.

ASSEMBLY LISTING NAME	ASSEMBLY LISTING LABEL	NARRATIVE
HASPMISC	GPRSTART	1 The priority aging processor is conditionally assembled depending on the value of PRTYRATE. If PRTYRATE is zero the priority aging processor is not included in HASPMISC.
HASPMISC	GRPTIME	2 The aging interval is determined by the JES2 generation variable PRTYRATE.
HASPMISC	GPRLCLOP	6 The highest priority to be considered for aging is determined by the JES2 initialization variable PRTYHIGH. The lowest priority to be considered for aging is determined by the JES2 initialization variable PRTYLOW.
HASPMISC	GPRJOEA	10 Priority age all JOES off the JQE.

JES2 Checkpoint Processor


Input
Process
Output
JES2 DISPATCHER
1 Initialization.
12
$CKPWORK
CKPSTQE
CKPMITQE
$CKW
CKWKSTQE
$HCT
$STATUS
$QSONDA BIT OFF
2 $CALL KREAD1.
3 If invalid alteration of queues.
K01
JES2 CATASTROPHIC ERROR ROUTINE
4 If error while obtaining lock or reading track 1.
KXX
JES2 CATASTROPHIC ERROR ROUTINE
5 If I/O error.
ENTER CKPT DIALOG
3 $CALL KREAD2.
6 If I/O error.
ENTER CKPT DIALOG
7 Detect new or deleted MAS members for Network Path Manager.
8 Make shared queues available and $POST $QSUSE waiters.
$HCT
$STATUS
$QSONDA TURNED OFF
$HASPECF



9 Propagate $POSTS from other systems.

10 Start minimum hold interval and maximum internal timers.

11 $CALL KPRIMW.

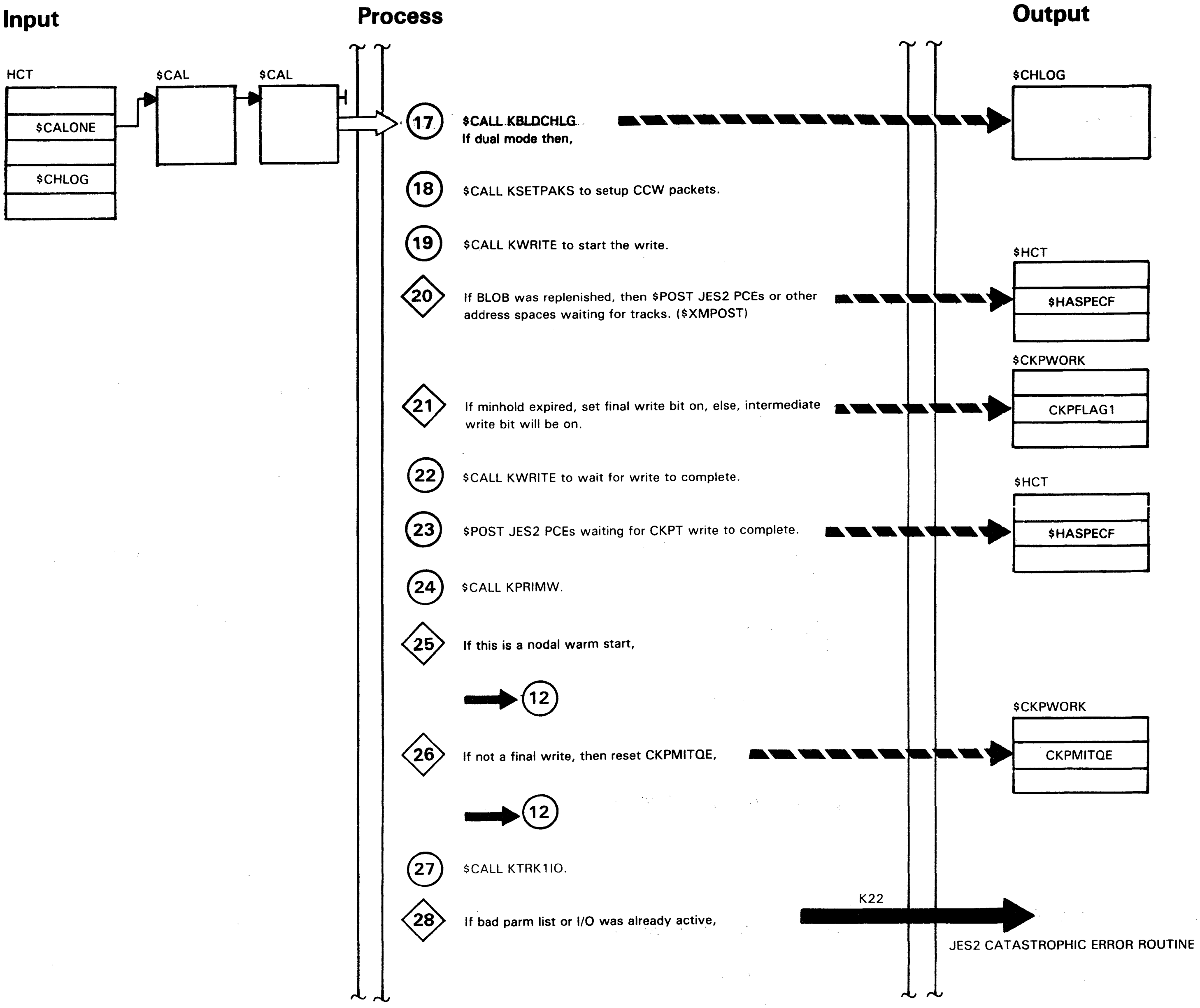
12 If minhold expired (CKPSTQE), the maximum time to wait before doing a CKPT write expired (CKPMITQE), a CKPT write was requested, or a JES2 processor or other address space is waiting for tracks, then $POST CKPTW.

13 $WAIT on CKPTW.

14 Move CKPT resource PCE queue to CKPT Post Queue.

15 $CALL KBLOB.
(Track group block allocation routine).

16 Time Stamp HCT and this system's QSE.



17 $CALL KBLDCHLG
If dual mode then,

18 $CALL KSETPAKS to setup CCW packets.

19 $CALL KWRITE to start the write.

20 If BLOB was replenished, then $POST JES2 PCEs or other address spaces waiting for tracks. ($XMPOST)

21 If minhold expired, set final write bit on, else, intermediate write bit will be on.

22 $CALL KWRITE to wait for write to complete.

23 $POST JES2 PCEs waiting for CKPT write to complete.

24 $CALL KPRIMW.

25 If this is a nodal warm start,

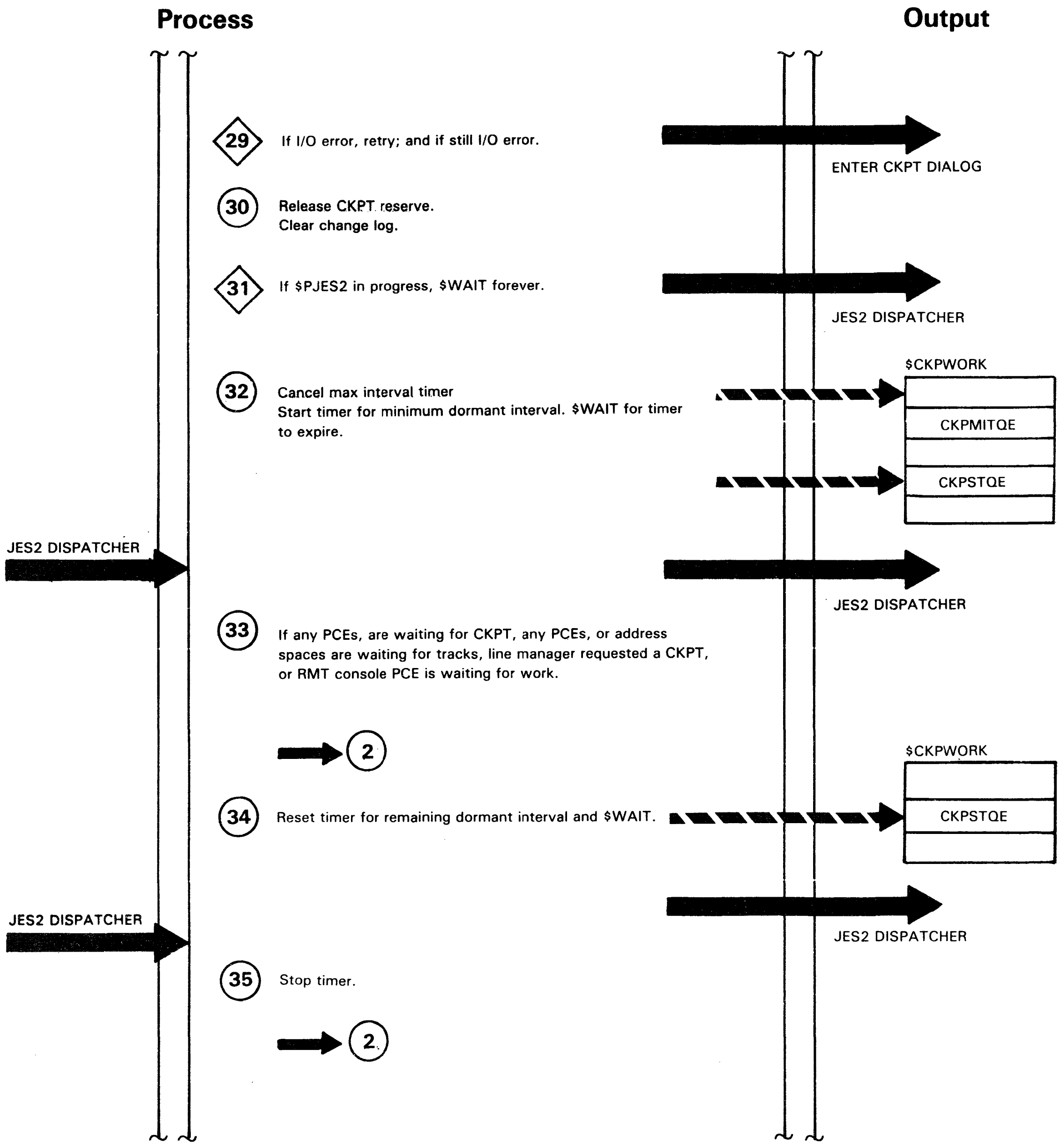
→ 12

26 If not a final write, then reset CKPMITQE,

→ 12

27 $CALL KTRK1IO.

28 If bad parm list or I/O was already active,



Process

29 If I/O error, retry; and if still I/O error.

30 Release CKPT reserve.
Clear change log.

31 If $PJES2 in progress, $WAIT forever.

32 Cancel max interval timer
Start timer for minimum dormant interval. $WAIT for timer to expire.

33 If any PCEs, are waiting for CKPT, any PCEs, or address spaces are waiting for tracks, line manager requested a CKPT, or RMT console PCE is waiting for work.

→ 2

34 Reset timer for remaining dormant interval and $WAIT.

35 Stop timer.

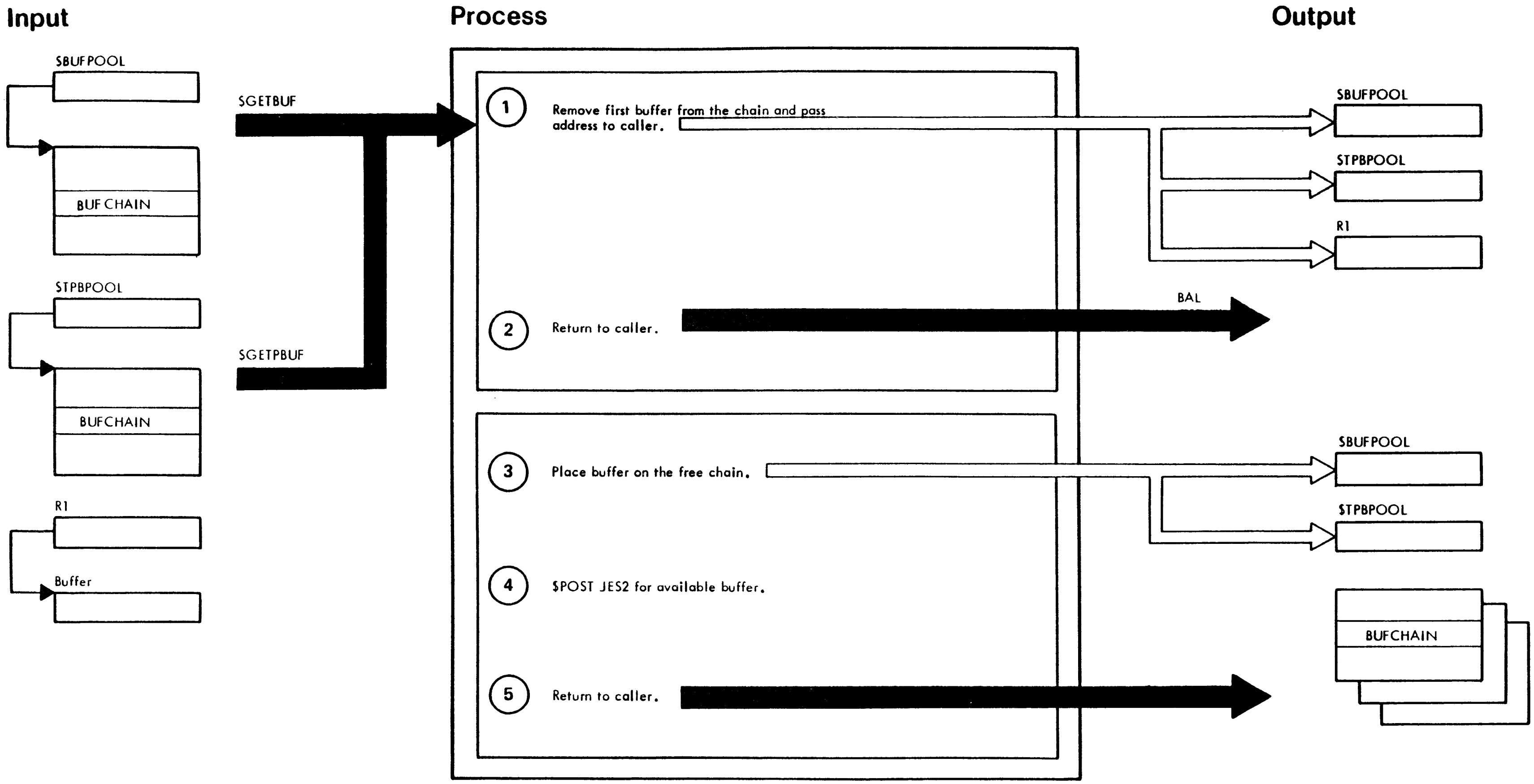
→ 2

Output

ASSEMBLY LISTING NAME	ASSEMBLY LISTING LABEL	NARRATIVE
HASPCKPT	HA$PCKPT	1 **Initialization executed only once.** ● Get addressability to $CKW (CKPT secondary work area) and to the $CKB of the CKPT data set to be written to. ● Copy 4-K records to I/O area. ● Initialize minhold timer (CKPSTQE), max interval timer (CKPMITQE), and timer in CKW (CKWKSTQE). ● $GETMAIN Page Pointer List. ● Initialize cross system $#POST Table. ● Start minhold timer (from CKPTDEF HOLD value). ● Indicate queues are in-storage ($QSONDA bit off).
	KREAD	2 Steps 3-5 occur within routine KREAD1.
		3 Step 6 occurs within routine KREAD2.
	KRDPOXEQ	9 The specific $POSTs from other systems include: 1) MAS resource $POSTs of XEQ 2) Spool offload for sysout transmitter 3) JOT Post 4) External writers
	KRDEND	11 The call to KPRIMW here is actually to do a primary write.
	KWCYCLE	12 The checks for minhold time left and tracks will not be done if it is a nodal warm start.
		19 This call to KWRITE is to initiate the write.
	KWRPWAI1	22 This call to KWRITE is to wait for the write.
	KDOWNLVL	23 This call to KPRIMW will only result in a primary write being done if 1) A RD2 was just done, 2) the system is shutting down, or 3) it is just after an intermediate write and the primary write count is down to zero.
	KRSETLOK	27 This call to KTRK1IO will reset the CKPT lock and DASD reserve.

Note: For a more detailed description of the individual $CALLed routines, see the documentation of HASPCKPT in Chapter 3.

JES2 – Buffer Services



1 Remove first buffer from the chain and pass address to caller.

2 Return to caller.

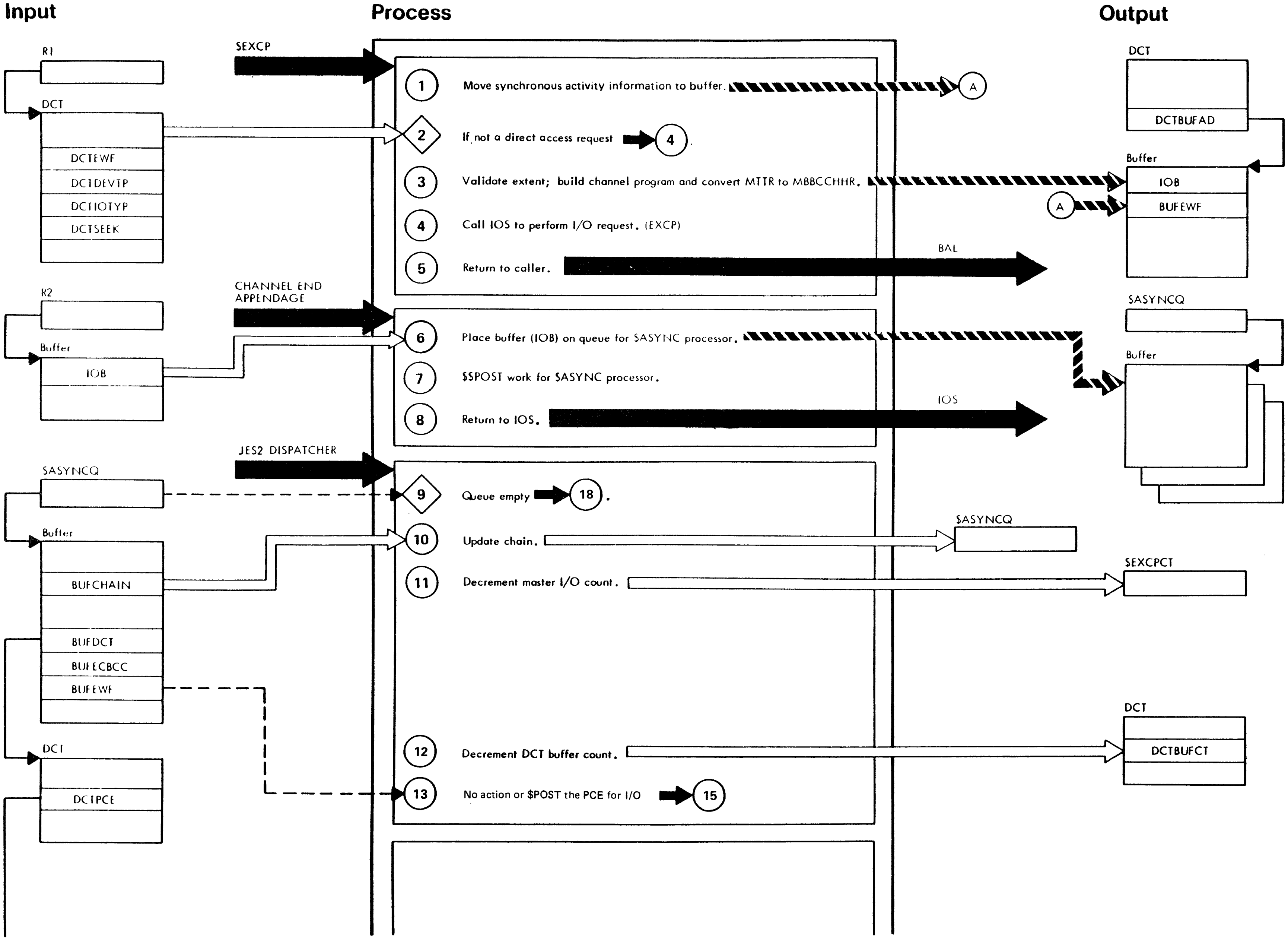
3 Place buffer on the free chain.

4 $POST JES2 for available buffer.

5 Return to caller.

ASSEMBLY LISTING NAME	ASSEMBLY LISTING LABEL	NARRATIVE
HASPNUC	$GETBUFR	① A buffer is taken either from $BUFPOOL or $TPBPOOL depending on the request and the chain is updated.
HASPNUC	$FREEBFR	② The appropriate buffer pool is used to chain the freed buffer. The buffers are placed in the queue in ascending order.
		③ JES2 is $POSTed for buffers.

JES2 – I/O Services



1 Move synchronous activity information to buffer.

2 If not a direct access request → 4.

3 Validate extent; build channel program and convert MTTR to MBBCCHHR.

4 Call IOS to perform I/O request. (EXCP)

5 Return to caller.

6 Place buffer (IOB) on queue for $ASYNC processor.

7 $$POST work for $ASYNC processor.

8 Return to IOS.

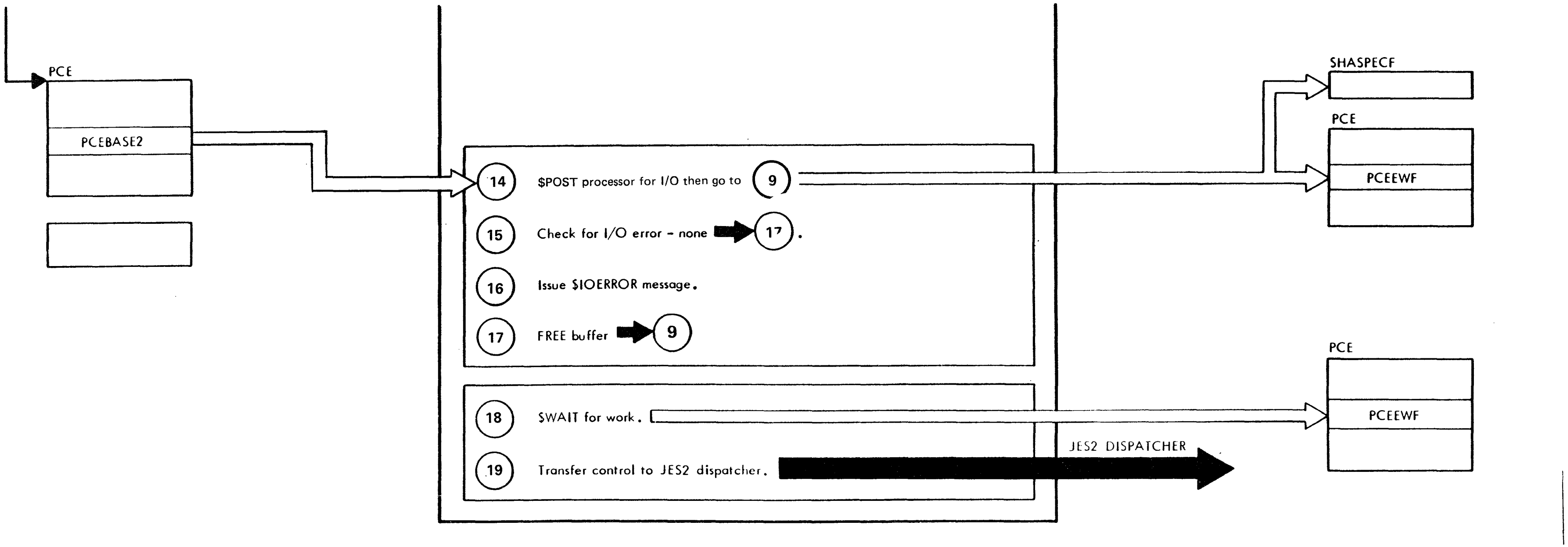
9 Queue empty → 18.

10 Update chain.

11 Decrement master I/O count.

12 Decrement DCT buffer count.

13 No action or $POST the PCE for I/O → 15



14 $POST processor for I/O then go to 9

15 Check for I/O error - none 17.

16 Issue $IOERROR message.

17 FREE buffer 9

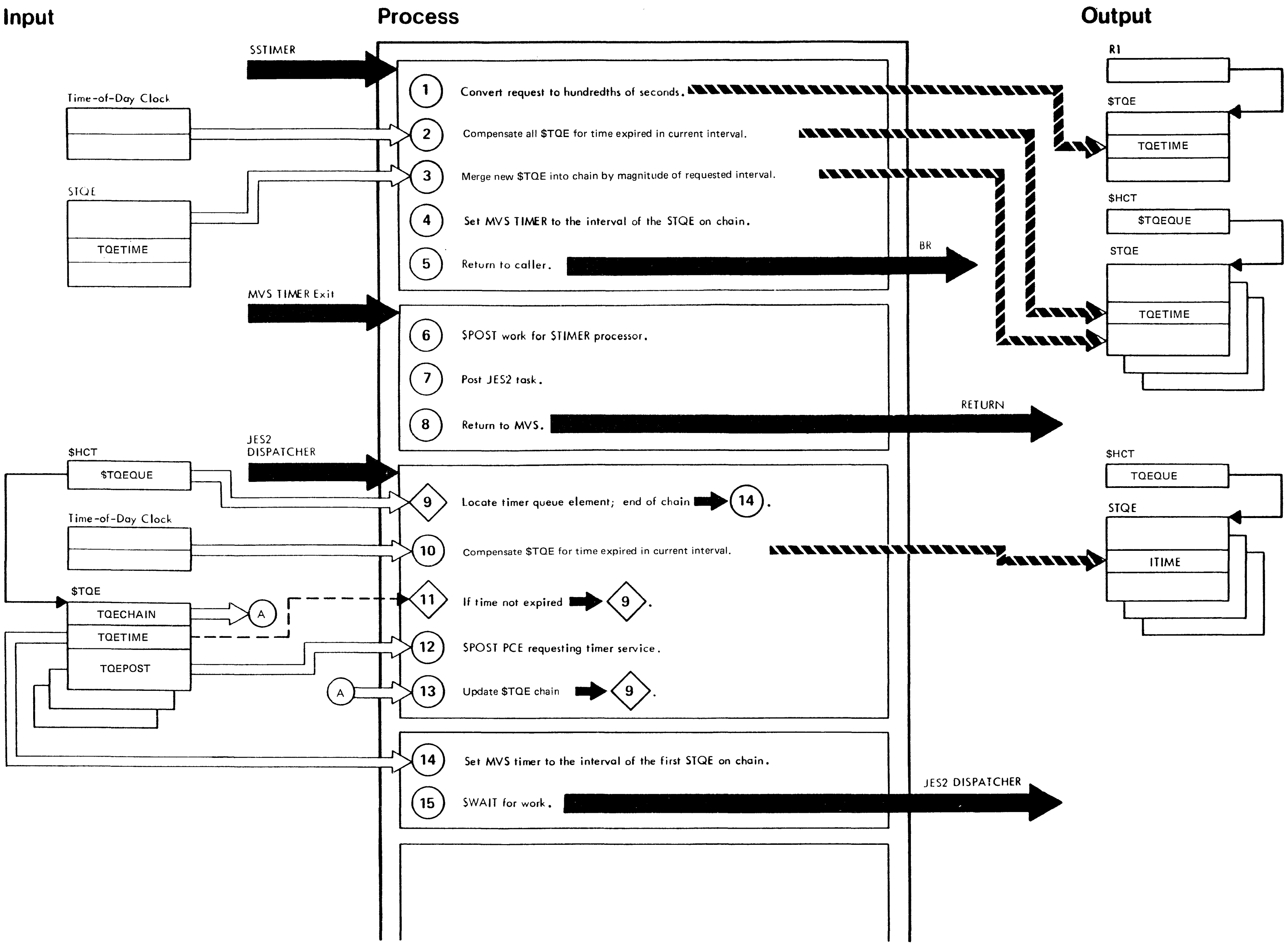
18 $WAIT for work.

19 Transfer control to JES2 dispatcher.

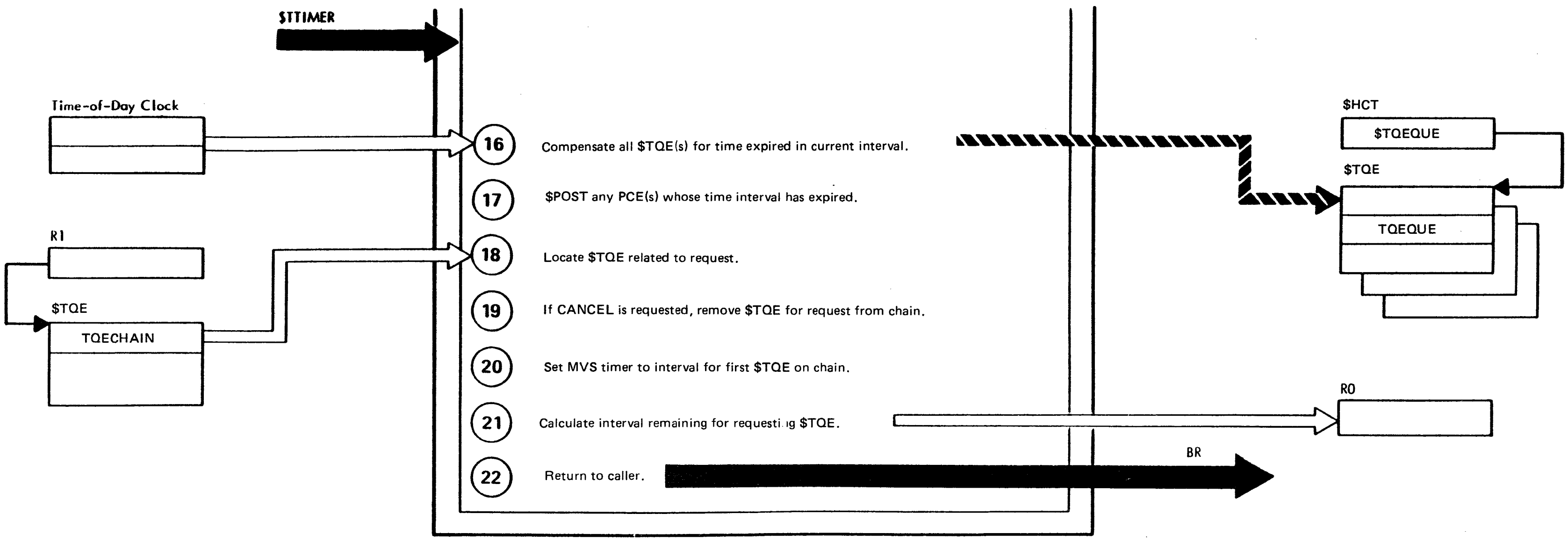
ASSEMBLY LISTING NAME	ASSEMBLY LISTING LABEL	NARRATIVE
HASPNUC	SEXCP	1 The synchronous activity to be performed at I/O completion is established.
HASPNUC	ESENDIT	4 The channel program is executed using a standard OS EXCP macro.
HASPNUC	ECHANEND	6 The ASYNC processor operates under control of the JES2 task and performs functions related to the completed I/O.
HASPNUC	$ASYNC	9 The queue of I/O completed buffers is tested for any elements. If none, $ASYNC processor $WAITs for work.
HASPNUC	ANOK	10 The chain is updated by use of the CS instruction to prevent interference from the channel end appendage.

ASSEMBLY LISTING NAME	ASSEMBLY LISTING LABEL	NARRATIVE
	AFREE	15 If an I/O error is detected, the JES2 macro $IOERROR is issued to log the condition on the operator console.
	AFREE1	17 The buffer is freed via $FREEBUF services.

JES2 Timer Services



1. Convert request to hundredths of seconds.
2. Compensate all $TQE for time expired in current interval.
3. Merge new $TQE into chain by magnitude of requested interval.
4. Set MVS TIMER to the interval of the STQE on chain.
5. Return to caller.
6. $POST work for $TIMER processor.
7. Post JES2 task.
8. Return to MVS.
9. Locate timer queue element; end of chain → 14.
10. Compensate $TQE for time expired in current interval.
11. If time not expired → 9.
12. $POST PCE requesting timer service.
13. Update $TQE chain → 9.
14. Set MVS timer to the interval of the first STQE on chain.
15. SWAIT for work.

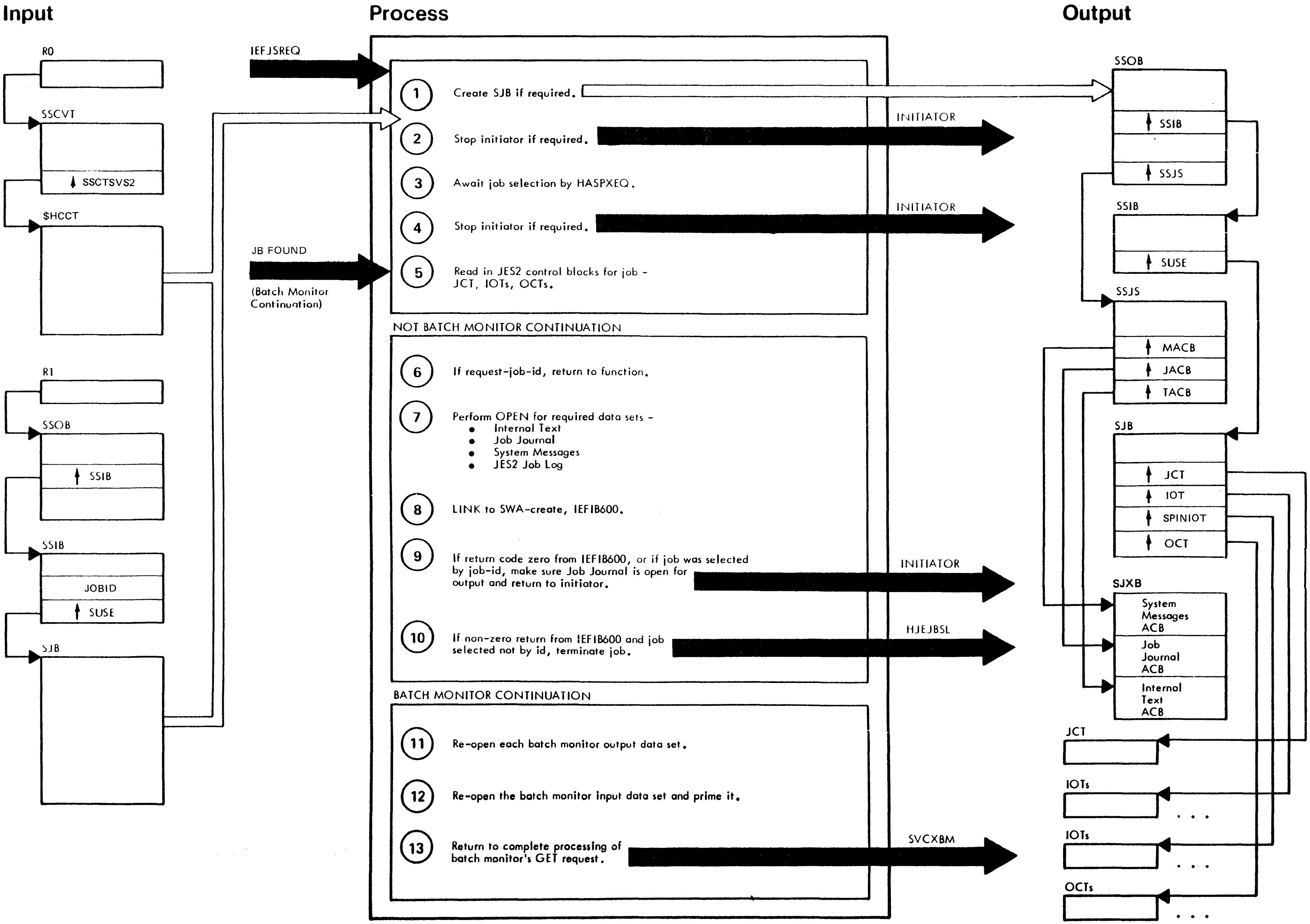

$TTIMER
Time-of-Day Clock
R1
$TQE
TQECHAIN
16 Compensate all $TQE(s) for time expired in current interval.
17 $POST any PCE(s) whose time interval has expired.
18 Locate $TQE related to request.
19 If CANCEL is requested, remove $TQE for request from chain.
20 Set MVS timer to interval for first $TQE on chain.
21 Calculate interval remaining for requesti.ıg $TQE.
22 Return to caller.
$HCT
$TQEQUE
$TQE
TQEQUE
R0
BR

ASSEMBLY LISTING NAME	ASSEMBLY LISTING LABEL	NARRATIVE
HASPNUC	$STIMER	1 The parameter value passed is converted to hundredths of seconds (see $STIMER macro description).
HASPNUC	IADJUST	2 The PCE for any $TQE whose time has expired is $POSTed for work and the $TQE chain is updated.
HASPNUC	INEXT	3
HASPNUC	ISETINT	4 The new interval is set using the MVS macro STIMER.
HASPNUC	IRETURN	5 The return is to the register link.

ASSEMBLY LISTING NAME	ASSEMBLY LISTING LABEL	NARRATIVE
HASPNUC	ITIMEUP	6 This is the asynchronous exit for the STIMER macro.
HASPNUC	$TIMER	9 The subroutine IADJUST is used to perform steps 9 through 13.
HASPNUC		14 The subroutine ISETINT is used to set the MVS time interval for the first $TQE by executing the macro STIMER.
HASPNUC	$TTIMER	16 Entry here is from the JES2 macro $TTIMER.
HASPNUC	IRET1	20 Same as 14 above.

JES2 – HASCJBST Job Select

MVS JOB SELECT REQUEST



1 Create SJB if required.

2 Stop initiator if required.

3 Await job selection by HASPXEQ.

4 Stop initiator if required.

5 Read in JES2 control blocks for job – JCT, IOTs, OCTs.

NOT BATCH MONITOR CONTINUATION

6 If request-job-id, return to function.

7 Perform OPEN for required data sets –
- Internal Text
- Job Journal
- System Messages
- JES2 Job Log

8 LINK to SWA-create, IEFIB600.

9 If return code zero from IEFIB600, or if job was selected by job-id, make sure Job Journal is open for output and return to initiator.

10 If non-zero return from IEFIB600 and job selected not by id, terminate job.

BATCH MONITOR CONTINUATION

11 Re-open each batch monitor output data set.

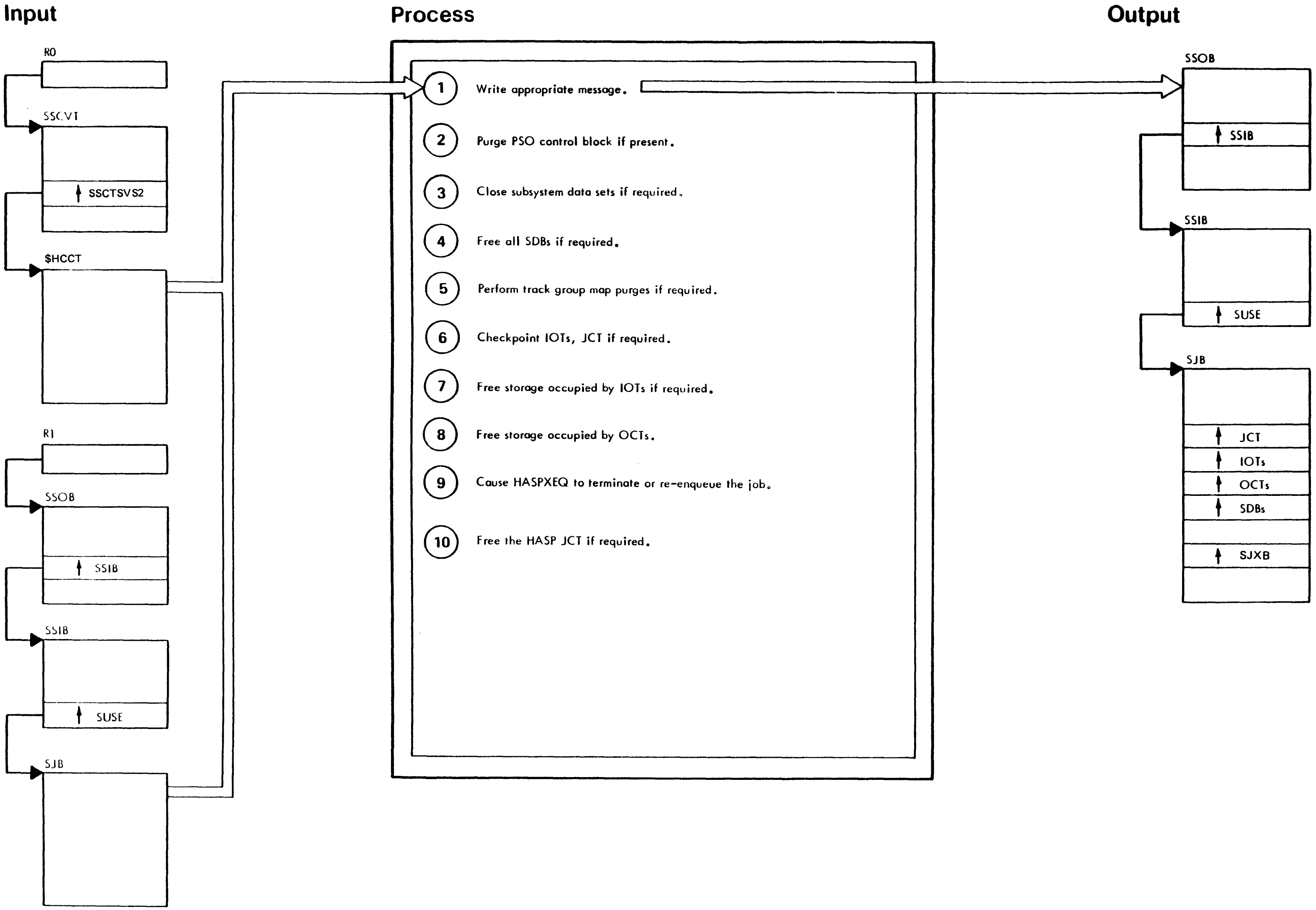
12 Re-open the batch monitor input data set and prime it.

13 Return to complete processing of batch monitor's GET request.

ASSEMBLY LISTING NAME	ASSEMBLY LISTING LABEL	NARRATIVE
HASCJBST	SSIJSEL	(1) A new SJB is needed if pointer SSIBSUSE is zero. This occurs for every selection by job id and for the first request from each initiator for a job to run.
	JSBLDSJB	(2) The initiator-stop flag, SJB2PNIT, may have been set later on in this routine if common storage resources are insufficient to run a job.
	HJS150	(3) The SJB is now on either of two job communication queues — CCTPCLS or $CCTPNUM. The pending-by-class queue means that HASPXEQ is free to select for this initiator, according to its own criteria, any job not read in through one of the two special internal readers. The pending-by-number queue means that HASPXEQ must select the job whose number, or job id, appears in field SJBJOBID. Such a job was read in through either STCINRDR (for started tasks) or TSOINRDR (for TSO logons) and received its JES2-assigned job id by issuing the ENDREQ operation to its internal reader.
	JSERCY	(4) The initiator-stop flag, SJB2PNIT, may have been set from the JES2 memory in response to the operator command $PIN.
	JBFOUND	(5) It should be noted that job selection by HASPXEQ is entirely on in-core process. From the end of conversion to tbe beginning of breakdown for output, the only reading and writing of HASP control blocks is done in HASPSSSM (except for spun and held data sets' IOTs). HASPXEQ stores in the SJB the MTTR of the job's JCT. The JCT in turn contains MTTRs that begin chains of input and output IOTs and OCTs.
	JBFXBMC	For execution batch monitor continuation (XBMC), IOTs must be read in differently, for the preserved in-core output IOTs of a batch monitor cannot be overlaid. Such IOTs contain PDDBs created by allocation of subsystem data sets when the batch monitor was first started.
	HJS320	(6) The request-job-id subsystem service sets up a job environment so that a system component can allocate and create subsystem data sets. The MVS system log uses this service.

ASSEMBLY LISTING NAME	ASSEMBLY LISTING LABEL	NARRATIVE
HASCTBST	HJS352	(7) Subroutines within HASCJBST are used, rather than SVC 19, to "fake-open" these data sets. The ACBs (and, for the JES2 job log, an RPL) are a part of the SJXB. The System Messages and JES2 job log data sets are here re-opened for continuation; they were originally opened for JCL conversion and continuation MTTRs were saved in the JCT.
	HJS582	(8) Call IEFIB600 to create the Scheduler Work Area. For a new job, it constructs in-core control blocks from the internal text data set. For an old job (warm-starting or restarting), it requires the Job Journal be present and open for input. Many JCL errors that were not detected by the converter are detected here in the interpreter function, the function which creates in-core scheduler, control blocks.
	HJS595	(9) If the Job Journal was open for input, SWA create will have issued POINT as its last operation to both the Journal and the System Messages data sets. Without disturbing the result of the POINT, code here converts the Journal to an output data set. If an error code is returned from IEFIB600 for a job rejected by job id, the job must nevertheless be passed to the initiator, our original caller, which was started specifically to run the job.
	HJS840	(10) The job-termination subsystem function serves in addition as a subroutine here of job select.
	HJS630	(11) Each batch monitor output data set retains intact its SDB and buffers. It must be assigned a new starting track from the output IOT whose track group map has been replaced (at 5, above) with the user job's track group map.
		(12) The previous user job running under this batch monitor was terminated by a GET at data to this data set.
		(13) Completion of this GET request starts the new user job in execution.

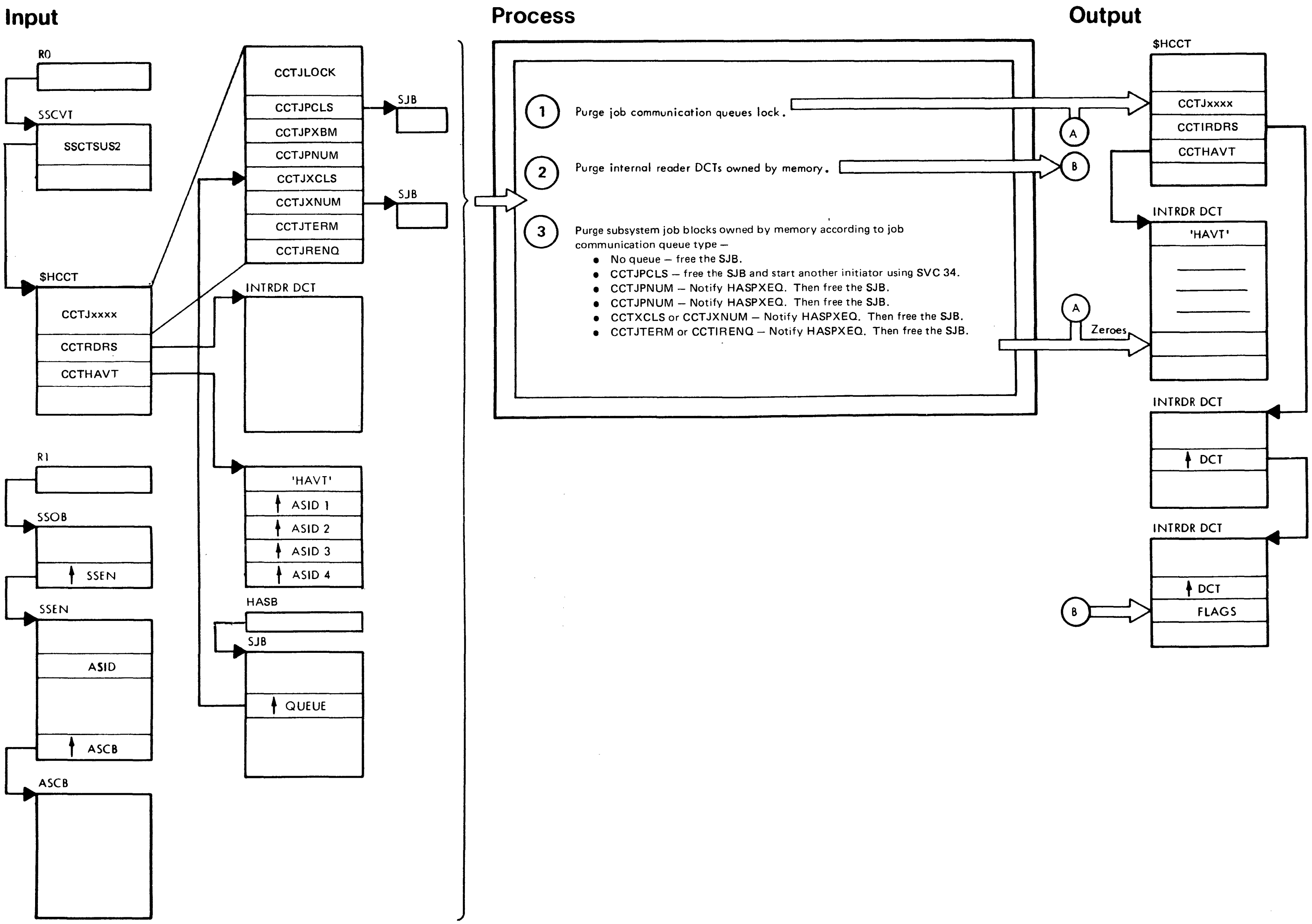
JES2 – HASPSSSM Job Terminate



1. Write appropriate message.
2. Purge PSO control block if present.
3. Close subsystem data sets if required.
4. Free all SDBs if required.
5. Perform track group map purges if required.
6. Checkpoint IOTs, JCT if required.
7. Free storage occupied by IOTs if required.
8. Free storage occupied by OCTs.
9. Cause HASPXEQ to terminate or re-enqueue the job.
10. Free the HASP JCT if required.

ASSEMBLY LISTING NAME	ASSEMBLY LISTING LABEL	NARRATIVE
HASCJBST	HJE000	① The appropriate message, as well as other required processing throughout this module, is determined from flags set by the HJE000 caller.
	HJC110	② Purge outstanding PSO requests, if necessary.
	HJEA00	③ No closes are performed for a batch monitor ending without a user job.
	HJE800	④ SDBs are preserved if entry is for XBM continuation.
	HJED00	⑤ If a job is terminating, all of its input track groups are purged here. If a job is to be rerun by JES2 (not an MVS restart), all of its output track groups are purged except those it had when it began execution. Otherwise, there is no purge.
	HJECKPT	⑥ No checkpointing is permitted for a batch monitor ending without a user job.
	HJEG00	⑦ IOTs must be preserved if entry is from HJEXBM — user job ending normally. Output IOTs contain PDDBs to be used when running the next user job.
HASCJBST	HJEG90	⑧ Output control table space is unconditionally freed.
	HJEXEQ	⑨ The SJB is queued on one of queued CCTJTERM and CCTJRENQ. When HASPXEQ completes processing, it cross-memory posts this processor, which continues.
	HJE100	⑩ For batch monitor continuation.

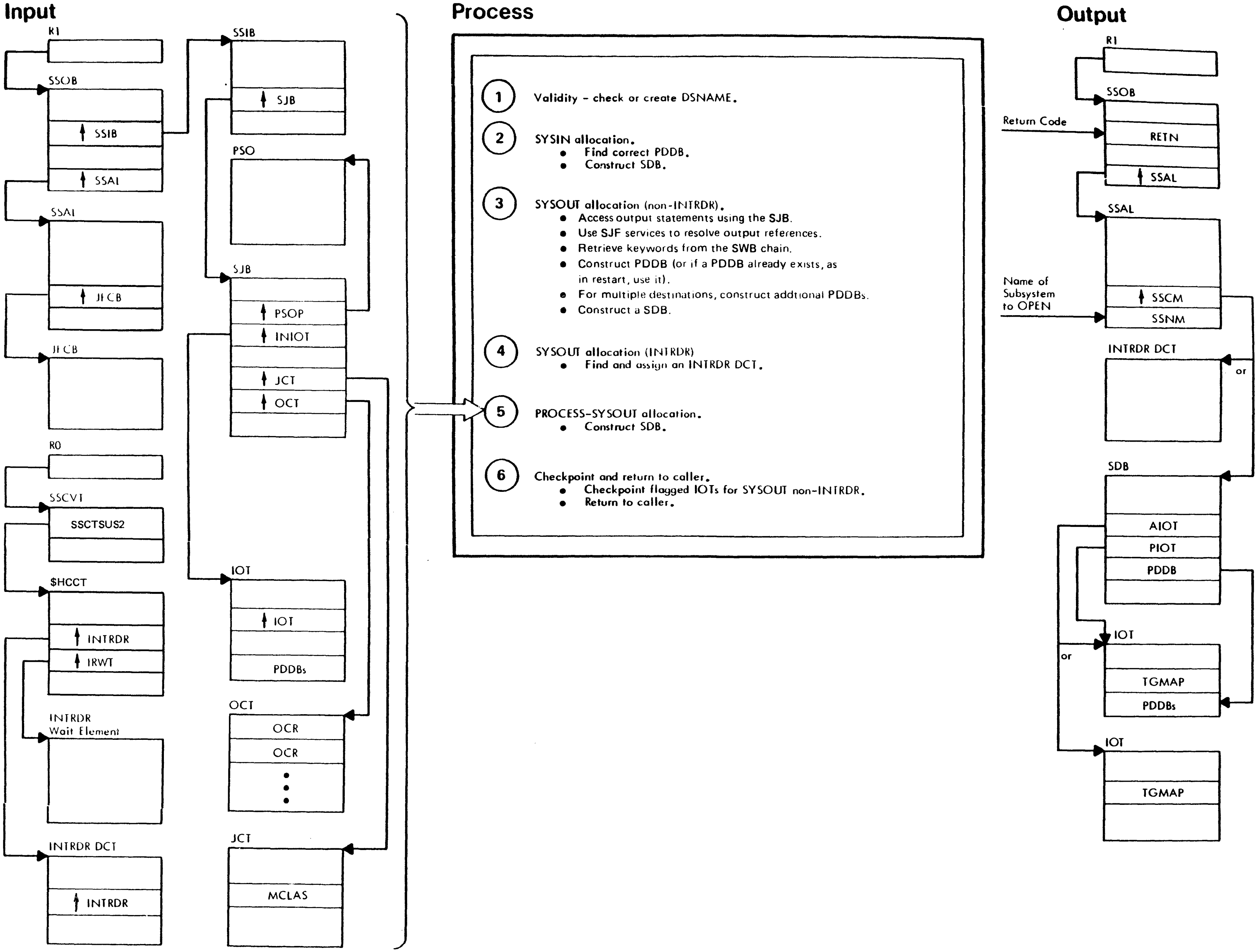
JES2 — HASPSSSM End-of-Memory



1. Purge job communication queues lock.
2. Purge internal reader DCTs owned by memory.
3. Purge subsystem job blocks owned by memory according to job communication queue type –
   - No queue – free the SJB.
   - CCTJPCLS – free the SJB and start another initiator using SVC 34.
   - CCTJPNUM – Notify HASPXEQ. Then free the SJB.
   - CCTJPNUM – Notify HASPXEQ. Then free the SJB.
   - CCTXCLS or CCTJXNUM – Notify HASPXEQ. Then free the SJB.
   - CCTJTERM or CCTIRENQ – Notify HASPXEQ. Then free the SJB.

ASSEMBLY LISTING NAME	ASSEMBLY LISTING LABEL	NARRATIVE
HASCJBTR	HEN100	(1) The job communication queues (JCQ) lock is a private, JES2 version of the MVS cross-memory-services (CMS) lock. In an end-of-memory situation, it is invalid for the JCQ lock to be held by the ending memory.
	HEN200	(2) Each allocated internal reader DCT contains the address of the ASCB it is associated with, for cross-memory posting purposes. If that address matches the terminating ASCBs address, the DCT is unallocated and any wait elements created by internal-reader allocation are cross-memory posted.
	HEN300	(3) The HASP address-space vector table (HAVT) is analogous to the MVSASVT. Each word of the HAVT contains a pointer to the $HASB control blocks associated with the particular address space. The $HASB, in turn, points to common storage control blocks, which are freed by end-of-memory processing.

JES2 — HASCDSAL Allocate



1. Validity – check or create DSNAME.

2. SYSIN allocation.
   - Find correct PDDB.
   - Construct SDB.

3. SYSOUT allocation (non-INTRDR).
   - Access output statements using the SJB.
   - Use SJF services to resolve output references.
   - Retrieve keywords from the SWB chain.
   - Construct PDDB (or if a PDDB already exists, as in restart, use it).
   - For multiple destinations, construct addtional PDDBs.
   - Construct a SDB.

4. SYSOUT allocation (INTRDR)
   - Find and assign an INTRDR DCT.

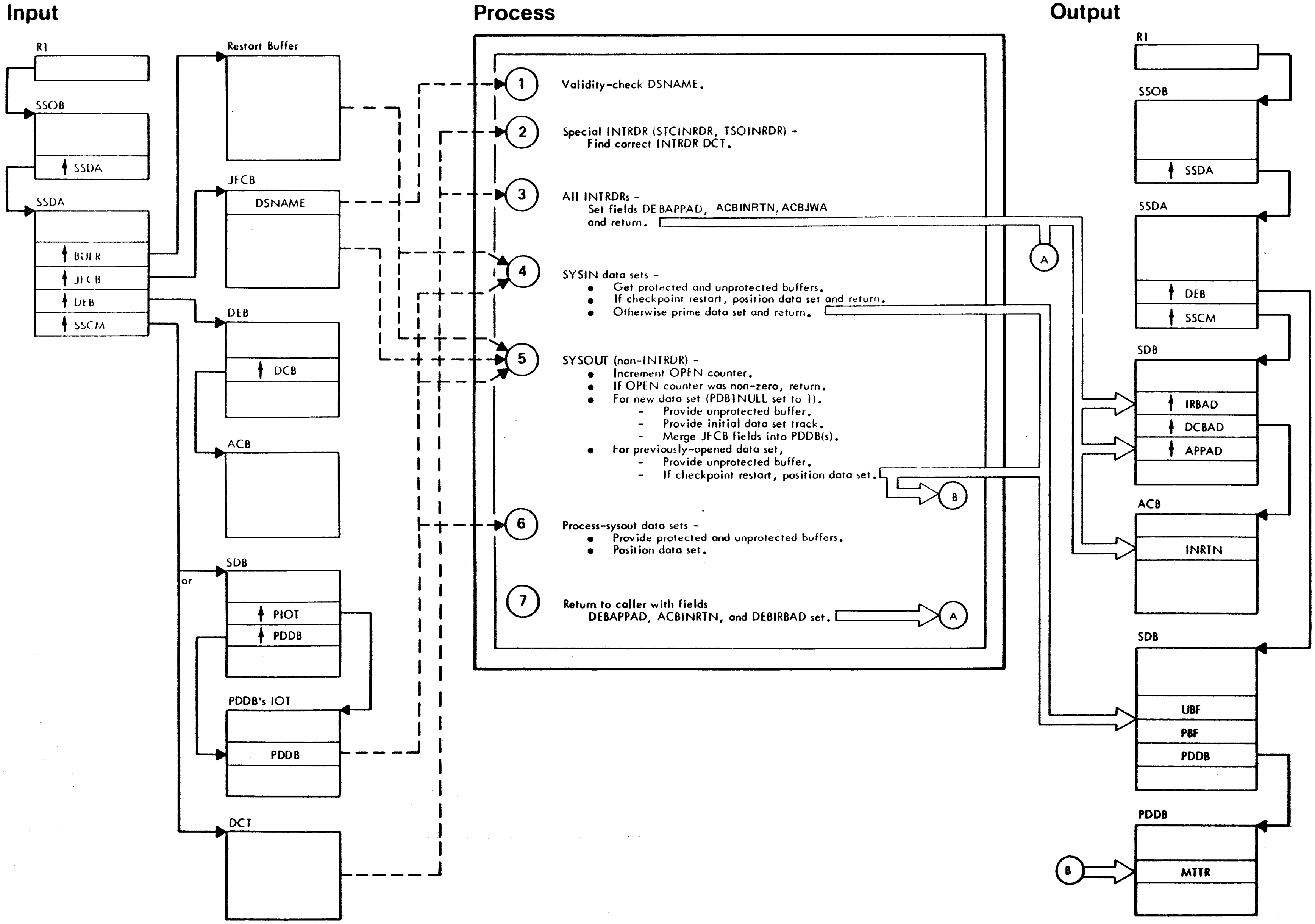
5. PROCESS-SYSOUT allocation.
   - Construct SDB.

6. Checkpoint and return to caller.
   - Checkpoint flagged IOTs for SYSOUT non-INTRDR.
   - Return to caller.

ASSEMBLY LISTING NAME	ASSEMBLY LISTING LABEL	NARRATIVE
HASCDSAL	SSIALOC	① Data set name is xxxx.JJJJJJJJ.TNNNNNNN where xxxx — subsystem name J — eight character jobid T — I for Sysin O for Sysout P for Process Sysout N — Dataset number If flag SSALASNM in SSALFLG1 is set, subroutine HALCRDSN creates a data set name using x x x x from CCTSSNM, JJJJJJJJ from SJBJOBID, and n n n n computed from JCTPDDBK.
	HALI	② Given the address of the first IOT, $PDBFIND scans to find a PDDB whose PDBDSKEY watches the binary equivalent of nnnnnnnn in the dsname in the JFCB.
	HAL015	③ After testing to see if the allocation IOT is pointed to by the SJB, check SSALASNM to see if the data set is dynamically allocated; if so, JES2 has already assigned a name to it; go build the SDB. If the data set is not dynamically allocated, get the next PDDB slot (using $PDBBLD) and use SJF services to resolve any references from the OUTPUT=parameter to OUTPUT statement. Use HASJFREQ to examine the SWB chain and retrieve the keyword associated with the OUTPUT = parameter. Merge into the JFCB the final characteristics of the PDDB and create additional PDDBs in the normal IOT for multiple destinations. Finally construct the SDB.

ASSEMBLY LISTING NAME	ASSEMBLY LISTING LABEL	NARRATIVE
HASCDSAL	HALR	④ Internal reader allocation is caused by use of the reserved user writer name 'INTRDR' with the SYSOUT keyword. The $HCCT points at CCTIRDRS to a chain of INTRDR DCTs built by HASPIRMA in common storage. This chain is scanned with the CMS lock held to find an available DCT. For control of internal-reader subsystem data sets, this DCT is used instead of an SDB.
	HALP	⑤ The PSO control block created during the previous process-sysout subsystem request, and pointed to by SJBPSOP, contains the dsname and PDDB for a PS-type request.
	HALDNORM	⑥ Subroutine HCBCK is used to checkpoint flagged direct-access-resident control blocks (JCT, regular IOTs) for the case of non-INTRDR SYSOUT allocation. On return to the caller, SSALSSNM contains a pointer to the name of the subsystem to be used to open this data set, and SSALSSCM contains a pointer to the SDB created (or to the DCT, for internal reader). Typically, the caller saves these fields in the DSAB at DSABSSNM and DSABSSCM for OPEN. The pointer stored at SSALSSCM reappears on an OPEN, subsystem request in the SSDA at SSDASSCM.

JES2 – HASCDSOC Open



1 Validity-check DSNAME.

2 Special INTRDR (STCINRDR, TSOINRDR) –
Find correct INTRDR DCT.

3 All INTRDRs –
Set fields DEBAPPAD, ACBINRTN, ACBJWA
and return.

4 SYSIN data sets –
- Get protected and unprotected buffers.
- If checkpoint restart, position data set and return.
- Otherwise prime data set and return.

5 SYSOUT (non-INTRDR) –
- Increment OPEN counter.
- If OPEN counter was non-zero, return.
- For new data set (PDB1NULL set to 1).
  - Provide unprotected buffer.
  - Provide initial data set track.
  - Merge JFCB fields into PDDB(s).
- For previously-opened data set,
  - Provide unprotected buffer.
  - If checkpoint restart, position data set.

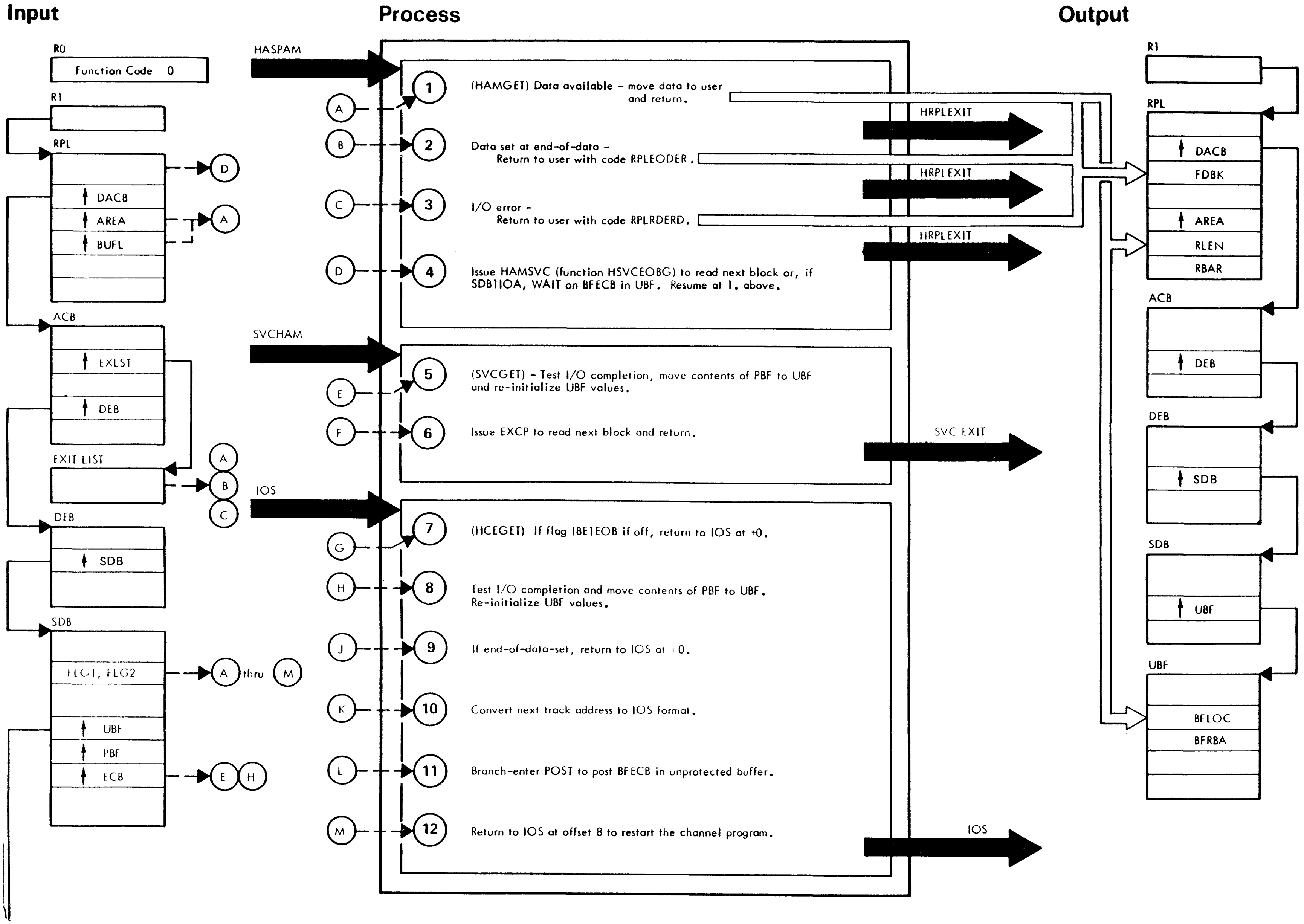
6 Process-sysout data sets –
- Provide protected and unprotected buffers.
- Position data set.

7 Return to caller with fields
DEBAPPAD, ACBINRTN, and DEBIRBAD set.

ASSEMBLY LISTING NAME	ASSEMBLY LISTING LABEL	NARRATIVE
HASCDSOC	SSIAOPN	① The data set type (IOP) is validated. Message HASP352 will eventually be issued if an error is detected.
	HOC560	② For performance reasons, the special internal readers are not allocated nor unallocated but merely opened and closed. IEFJSWT maintains serialization and is the only user of these data sets.
	DSO000	③ ⑦ DEBAPPAD points to HASPAM for subsystem data sets. It is used by SVC 111 to reference HASPAM+4, which contains the address of SVCHAM. ACBINRTN points to HASPAM and is used by the expansion of a VSAM request macro-instruction. The first byte of this full-word has bit X'04' set if the data set is an internal reader. DEBIRBAD points to the SDB if it is not an internal reader. Else ACBJWA points to the internal reader DCT.
	DSO110	④ Subroutine HOOLDINP provides buffers for an input data set. The RESTART subsystem request processor uses HOJOPEN to position a data set for an automatic or deferred checkpoint restart. The argument list (SSDA) is the same for RESTART as for OPEN (and CLOSE and CHECKPOINT), with flags SSDAAUTO and SSDADEFR to distinguish from normal OPEN. For automatic checkpoint restart the RBA saved in the checkpoint record by SSIDACKP is valid and positioning is rapid. For deferred checkpoint restart the RBA is invalid and a logical record counter, saved in the checkpoint record by SSIDACKP, must be used to count through the data set; positioning is slow.

ASSEMBLY LISTING NAME	ASSEMBLY LISTING LABEL	NARRATIVE
HASCDSOC	DSO200	⑤ An output data set may be open to more than one user at the same time. It is the user's responsibility to synchronize use of the data set. Subroutine HONEWOUT supplies an unprotected buffer and a starting track for a new output data set. On return, fields LRECL, RECFM, UCS, FCB, and FUNC are merged into the PDDB to cover the case of user JFCB modification before OPEN. Subroutine HOOLDOUT supplies an unprotected buffer for an old output data set. Positioning is necessary only for automatic checkpoint restart. Two cases arise — the data set was open at checkpoint or closed at checkpoint. For the former case, the checkpoint record contains a positioning RBA. For the latter case, the data set is positioned to file.
	DSO110	⑥ Subroutine HOOLDINP provides unprotected and protected buffers.

JES2 – HASPAM Get



1 (HAMGET) Data available – move data to user and return.

2 Data set at end-of-data – Return to user with code RPLEODER.

3 I/O error – Return to user with code RPLRDERD.

4 Issue HAMSVC (function HSVCEOBG) to read next block or, if SDB1IOA, WAIT on BFECB in UBF. Resume at 1. above.

5 (SVCGET) – Test I/O completion, move contents of PBF to UBF and re-initialize UBF values.

6 Issue EXCP to read next block and return.

7 (HCEGET) If flag IBE1EOB if off, return to IOS at +0.

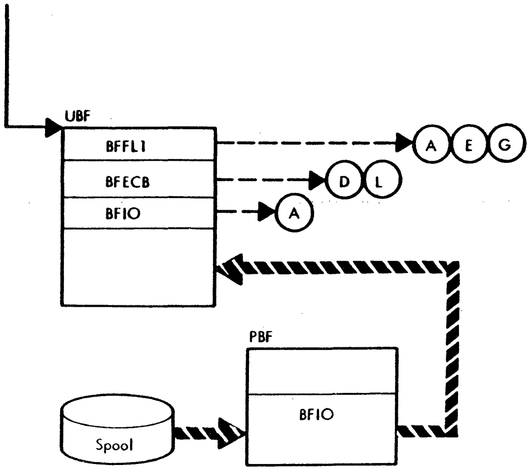
8 Test I/O completion and move contents of PBF to UBF. Re-initialize UBF values.

9 If end-of-data-set, return to IOS at +0.

10 Convert next track address to IOS format.

11 Branch-enter POST to post BFECB in unprotected buffer.

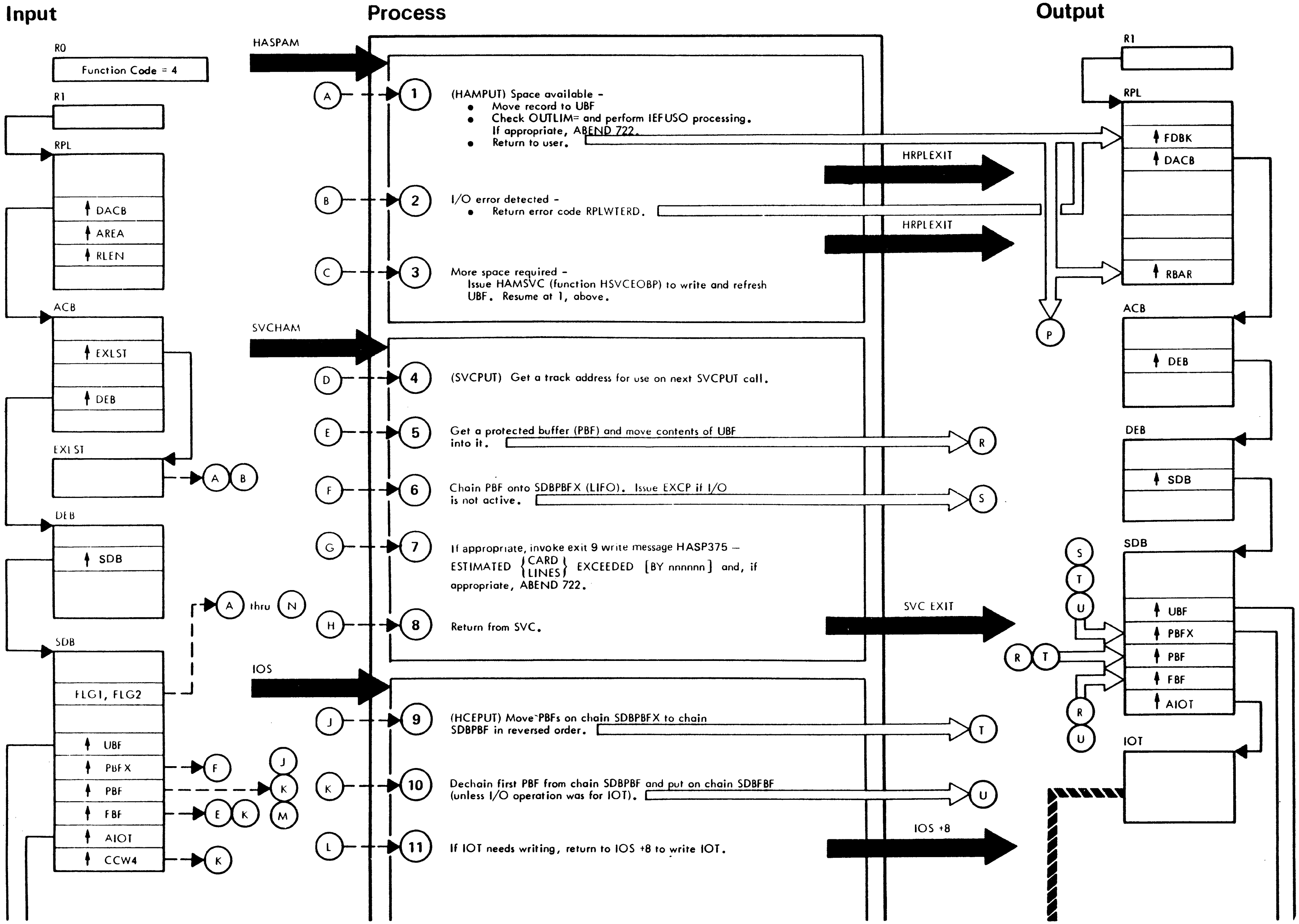
12 Return to IOS at offset 8 to restart the channel program.



ASSEMBLY LISTING NAME	ASSEMBLY LISTING LABEL	NARRATIVE
HASPAM	HAMGET	(1) Subroutine HGMOVE moves data to the user area (pointer from RPLAREA) unless end-of-buffer is found. On return, HGMOVE's return code dictates whether to move RBA to RPLRBAR and return or to continue at HGSPEC.
	HGS030	(2) (3) HGSPEC analyzes end-of-buffer conditions. These may be I/O error or normal end-of- data set. The user is notified, except that end-of-data on a batch input data set causes normal termination via SVCXBM of a batch job.
	HGSPEC	(4) (7) If the user is reading from this subsystem data set so slowly that the channel end appendage is entered for the next block before subroutine HGMOVE has set flag UBFIEOB, it is necessary to use SVCGET to move the next block of data to the unprotected buffer. But if the user is reading so fast that I/O is not yet complete, the UBF's BFECB is used for WAIT. Appendage HCEGET always posts this ECB after refilling the UBF.
	SVCGET HCEGET	(5) (8) SVCGET and HCEGET use common subroutine HENDREAD to validate the block just read (I/O complete without error, job and data set keys correct, next track address valid). They use common subroutine HMOVEPU to refill the unprotected buffer and to update UBF values and current and next track addresses and record count.

ASSEMBLY LISTING NAME	ASSEMBLY LISTING LABEL	NARRATIVE
HASPAM	HCRSPEC	(9) The end-of-data set flag, SDB2EOD, is set by subroutine HENDREAD above, as well as the I/O error flag, SDB2IOE.
	HCGU20	(10) The JES2 track address, contained in bytes 1-4 of the JES2 RBA, is of the format MTTR. For all subsystem data sets processed by HAM, subroutine HCNVFDAD converts this address, in word SDBMTTR, to the OS format, MBBCCHHR, placing the value directly in the SDB's IOB.
	HCRETURN	(11) Subroutine HPOSTECB is used to post BFECB. Since the appendage receives control in the user's memory, this subroutine attempts a quick (compare-and-swap) post before deciding to use MVS POST at branch entry IEAPT1.
		(12) Thus the channel-end appendage tries to increase performance by avoiding the overhead of an EXCP for each data block.

JES2 – HASPAM Put



1 (HAMPUT) Space available –
- Move record to UBF
- Check OUTLIM= and perform IEFUSO processing. If appropriate, ABEND 722.
- Return to user.

2 I/O error detected –
- Return error code RPLWTERD.

3 More space required –
Issue HAMSVC (function HSVCEOBP) to write and refresh UBF. Resume at 1, above.

4 (SVCPUT) Get a track address for use on next SVCPUT call.

5 Get a protected buffer (PBF) and move contents of UBF into it.

6 Chain PBF onto SDBPBFX (LIFO). Issue EXCP if I/O is not active.

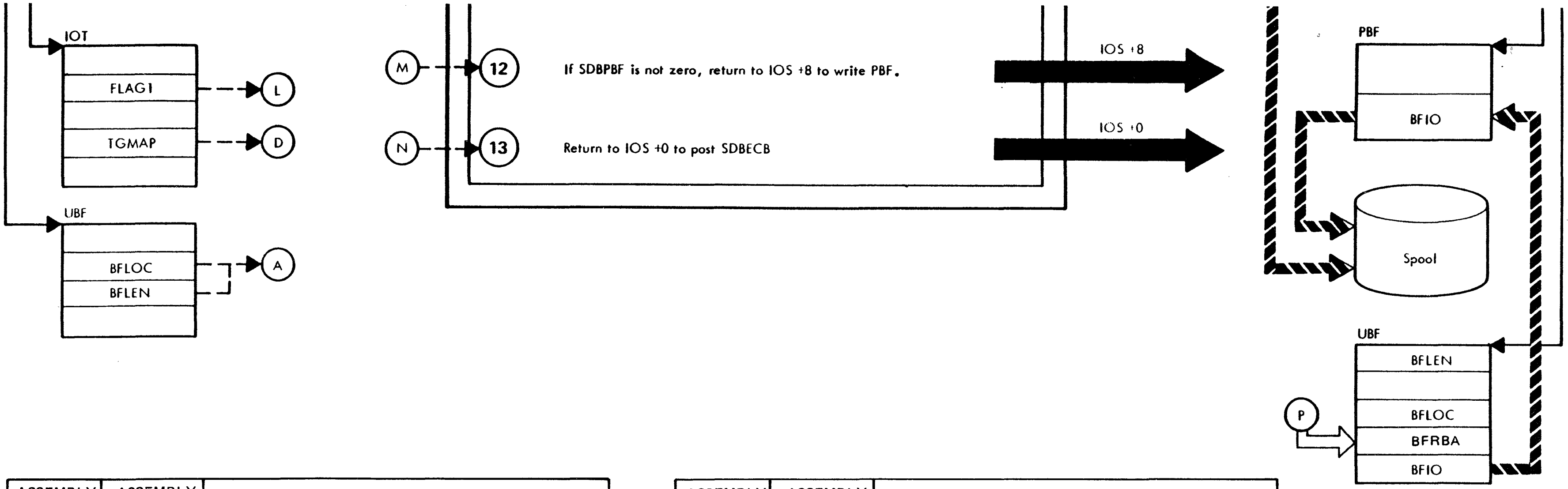
7 If appropriate, invoke exit 9 write message HASP375 – ESTIMATED {CARD / LINES} EXCEEDED [BY nnnnnn] and, if appropriate, ABEND 722.

8 Return from SVC.

9 (HCEPUT) Move PBFs on chain SDBPBFX to chain SDBPBF in reversed order.

10 Dechain first PBF from chain SDBPBF and put on chain SDBFBF (unless I/O operation was for IOT).

11 If IOT needs writing, return to IOS +8 to write IOT.

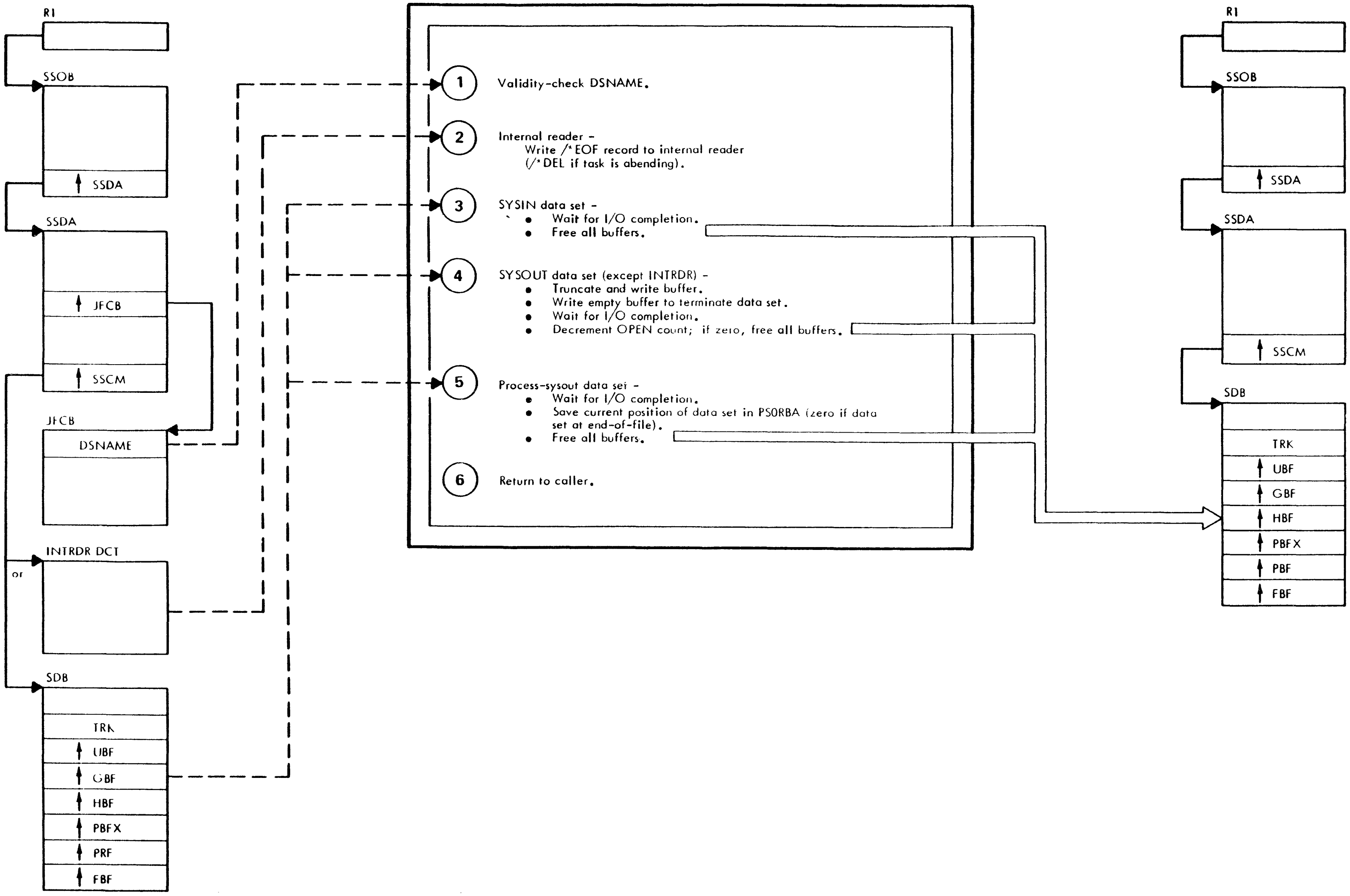

IOT
FLAG1
TGMAP
L
D
UBF
BFLOC
BFLEN
A
M
N
12
13
If SDBPBF is not zero, return to IOS +8 to write PBF.
Return to IOS +0 to post SDBECB
IOS +8
IOS +0
PBF
BFIO
Spool
UBF
BFLEN
BFLOC
BFRBA
BFIO
P

ASSEMBLY LISTING NAME	ASSEMBLY LISTING LABEL	NARRATIVE
HASPAM	HAMPUT	(1) Subroutine HPMOVE moves data from the user area to the data set's unprotected buffer (UBF) and the record's assigned RBA to the user's RPL. On return, HAMPUT updates field BFRBA for the next PUT and checks that the total record count is not greater than specified by the OUTLIM= keyword. If the output limit is exceeded, HAMPUT uses SVC function HSVCOUTL to call IEFUSO. IEFUSO decides whether to increase the limit or to abend the user's task with code 722 and a dump.
		(2) I/O error flag SDB21OE is set whenever the abnormal channel end appendage is entered.
		(3) More space may be required because the UBF had sufficient space to write a non-spanned record (a record less than 255 bytes long) or because HPMOVE successfully created the first or a middle segment of a spanned record.
	SVCPUT	(4) Unless flag SDB1CLOS is set, SVCPUT uses subroutine $STRAK to allocate a track address. This track address will be placed in field BFNXT of the PBF to be created on this call to SVCPUT. Data will not be written to this track address until the next call of SVCPUT. In its call to $STRAK, SVCPUT determines whether the track group has changed; if so, it sets flag IOTICKPT in the allocation IOT so that HCEPUT will write it.

ASSEMBLY LISTING NAME	ASSEMBLY LISTING LABEL	NARRATIVE
HASPAM	SVCP20	(5) One or more protected buffers may have been chained onto SDBFBF by HCEPUT. SVCPUT uses subroutine HSPFBFRE to free all but the most recently chained of these and re-uses that one. But if no buffer is found this way, SVCPUT calls GETMAIN if the count of protected buffers is not at a maximum. If the count is at a maximum or the conditional GETMAIN fails, SVCPUT WAITs on the first word of the SDB having set a flag to cause HCEPUT to POST this ECB when next it releases a buffer.
	SVCP60	(6) Of necessity the SDBPBFX chain must be LIFO because of its use by mutually asynchronous routines SVCPUT and HCEPUT.
	SVCP100	(7) Estimates of print lines and punched cards occur on the JES2 job card. Exit point SVCOUTX (for exit 9) is taken to allow an installation exit routine to process the excess output condition.
	HCEPUT	(9) HCEPUT dechains (using CS) the entire chain SDBPBFX and puts it on chain SBDPBF first-in-first-out.

JES2 — HASCDSOC Close

Input | Process | Output



1 Validity-check DSNAME.

2 Internal reader –
Write /* EOF record to internal reader
(/* DEL if task is abending).

3 SYSIN data set –
- Wait for I/O completion.
- Free all buffers.

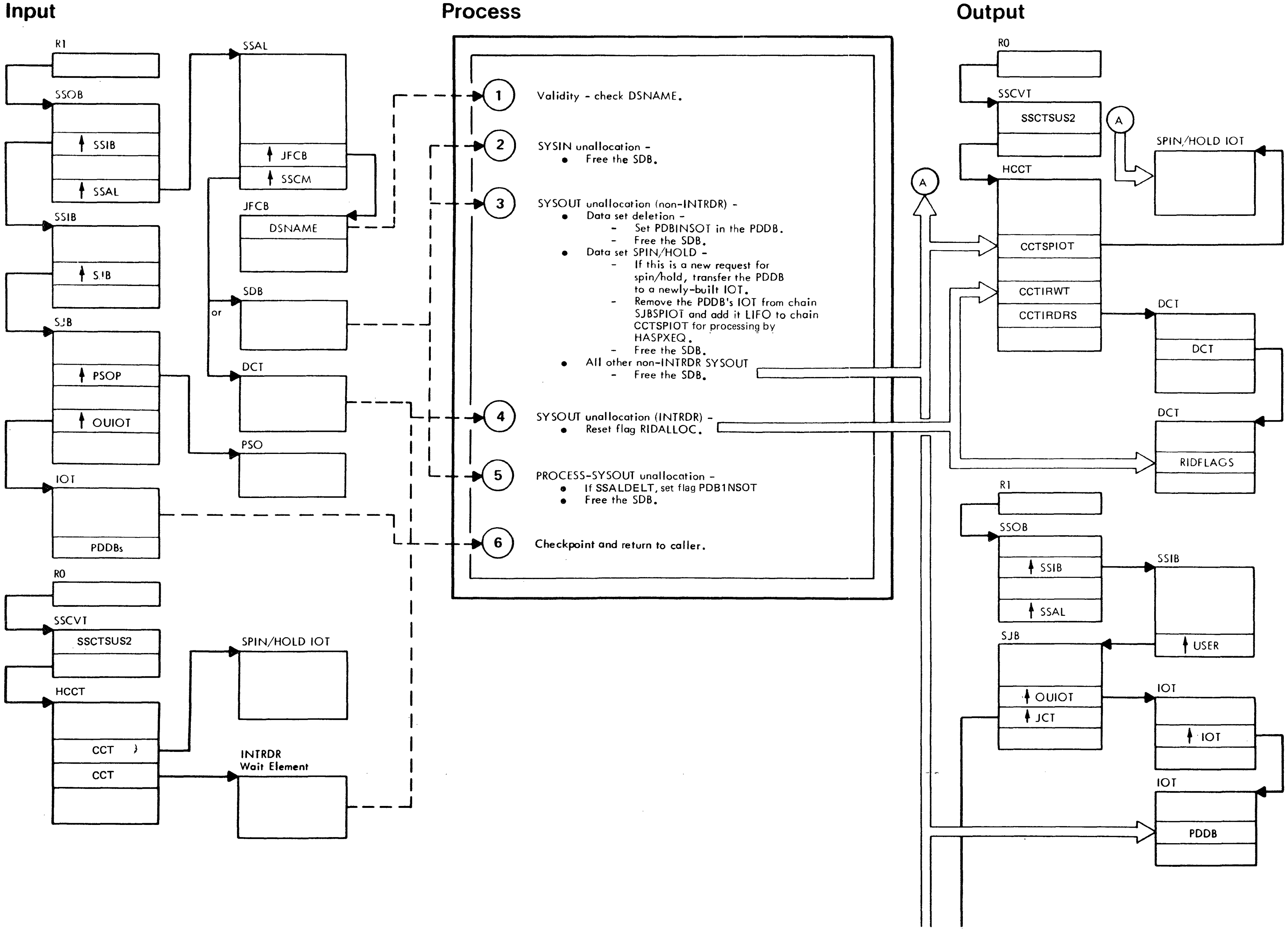
4 SYSOUT data set (except INTRDR) –
- Truncate and write buffer.
- Write empty buffer to terminate data set.
- Wait for I/O completion.
- Decrement OPEN count; if zero, free all buffers.

5 Process-sysout data sei –
- Wait for I/O completion.
- Save current position of data set in PSORBA (zero if data set at end-of-file).
- Free all buffers.

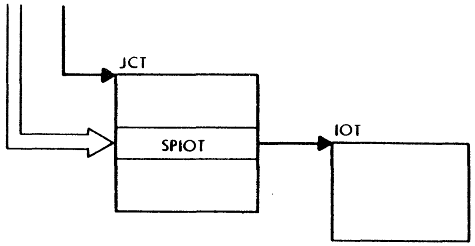
6 Return to caller.

ASSEMBLY LISTING NAME	ASSEMBLY LISTING LABEL	NARRATIVE
HASCDSOC	SSIDACLO HOCSETUP	① Only the subsystem-name portion of the dsname is checked for validity. Further checking is done as the data set type is examined (IOS); if the data set type is none of these, message HASP353 is written.
	DACIR10	② SSIDACLO uses HAM subroutine HINTRDR to write records to an internal reader. Register 2 contains the record's address, and register 0 contains the close flag BFD1ICLS and register 3 contains its length.
HASCDSOC	DACSO	④ The end-of-buffer marker LRCBFEND is put at the next available byte of the unprotected buffer and section SVCPUT of SVCHAM is invoked to transfer the data to a protected buffer and write it. The written buffer contains a non-zero chaining track. Then the now-empty unprotected buffer is marked end-of-buffer and SVCPUT is again invalid, this time with flag SDB1CLOS set to cause SVCPUT to write the final record with a zero chaining track. If the data set is again opened, this final record will be overwritten (at the next CLOSE or earlier) with a record containing a non-zero chaining track. More than one user can be OPEN at the same time to the same SYSOUT data set. If other users are still OPEN to the data set being CLOSEd, no buffers are freed.
	HC100 DACPOSI	⑤ Data set buffers are never freed for execution batch monitor OPENs and CLOSEs.

JES2 – HASCDSAL Unallocate



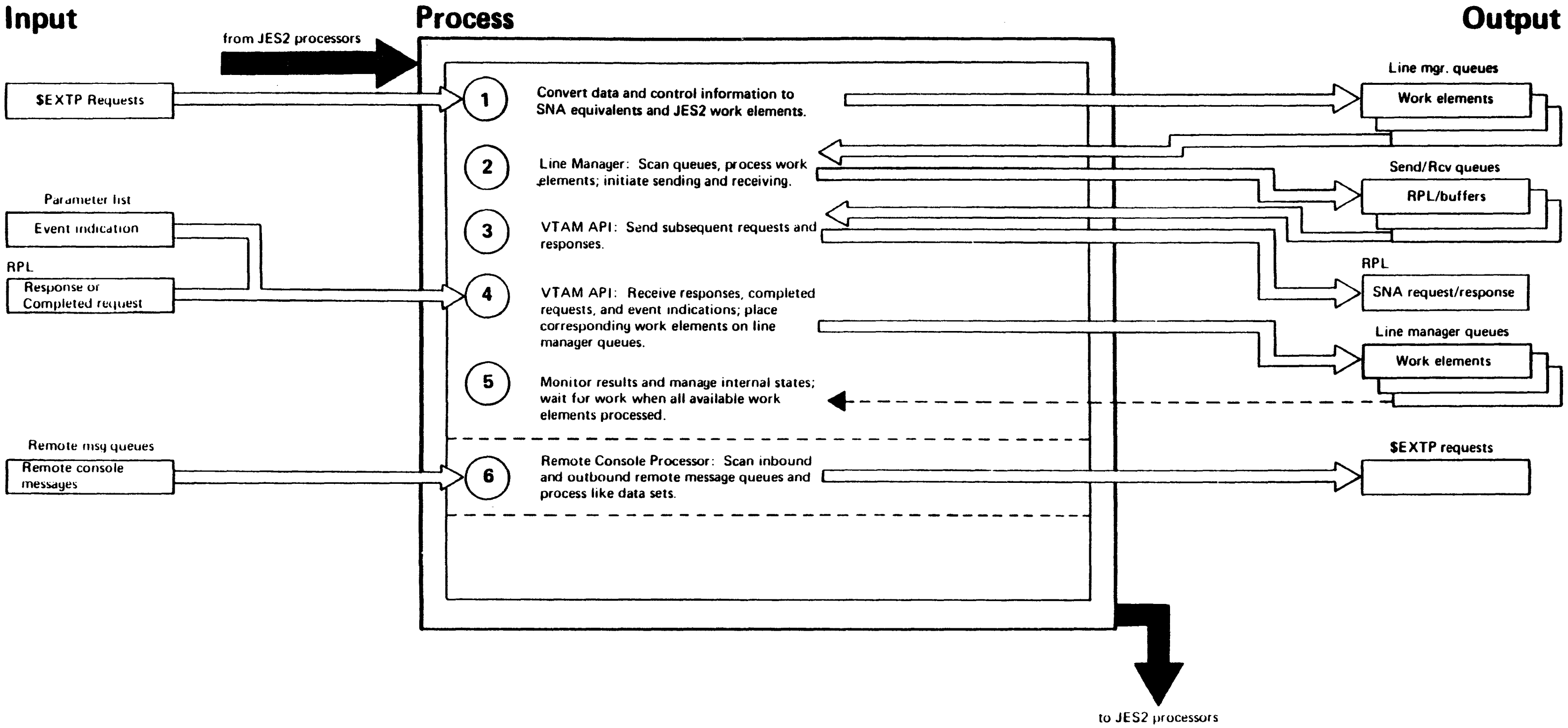
1. Validity - check DSNAME.
2. SYSIN unallocation -
   - Free the SDB.
3. SYSOUT unallocation (non-INTRDR) -
   - Data set deletion -
     - Set PDBINSOT in the PDDB.
     - Free the SDB.
   - Data set SPIN/HOLD -
     - If this is a new request for spin/hold, transfer the PDDB to a newly-built IOT.
     - Remove the PDDB's IOT from chain SJBSPIOT and add it LIFO to chain CCTSPIOT for processing by HASPXEQ.
     - Free the SDB.
   - All other non-INTRDR SYSOUT
     - Free the SDB.
4. SYSOUT unallocation (INTRDR) -
   - Reset flag RIDALLOC.
5. PROCESS-SYSOUT unallocation -
   - If SSALDELT, set flag PDB1NSOT
   - Free the SDB.
6. Checkpoint and return to caller.



ASSEMBLY LISTING NAME	ASSEMBLY LISTING LABEL	NARRATIVE
HASCDSAL	SSIALUNA	(2) Subroutine $SDBFREE cleans up and frees SDBs constructed by $SDBINIT, and dechains them from the SJB (header SJBSDB).
	HUAO	(3) Flags SSALDELT, SSALHOLD, and SSALSPIN, as well as the state upon allocation of the data set and its IOT (IOT1SPIN, PDB1SPIN, PDB1HOLD, PDB1PSO) determine processing. A data set which has the HOLD or SPIN characteristic becomes available for printing immediately on unallocation rather than at end-of-job. The IOT containing the single PDDB which represents this data set is given to JES2 (IOTs reside in common storage) for HASPXEQ to process and free.

ASSEMBLY LISTING NAME	ASSEMBLY LISTING LABEL	NARRATIVE
HASCDSAL	HUAR	(4) Flag RIDALLOC is reset with the CMS lock held to unallocate an internal reader. A chain of internal-reader-allocation wait elements may exist at CCTRWT; for each element, either the ECB is cross-memory posted (memory not ending) or the wait element is freed. The end-of-memory subsystem request processor performs similar actions.
	HUAP	(5) A process-sysout data set, like a sysin data set, is an input data set. But the user of the data set may request its deletion. If SSALDELT is on, PDBINSOT is set in the PDDB in the PSO control block.
	HUA900	(6) For non-INTRDR SYSOUT unallocation, all modified direct-access-resident control blocks are checkpointed.

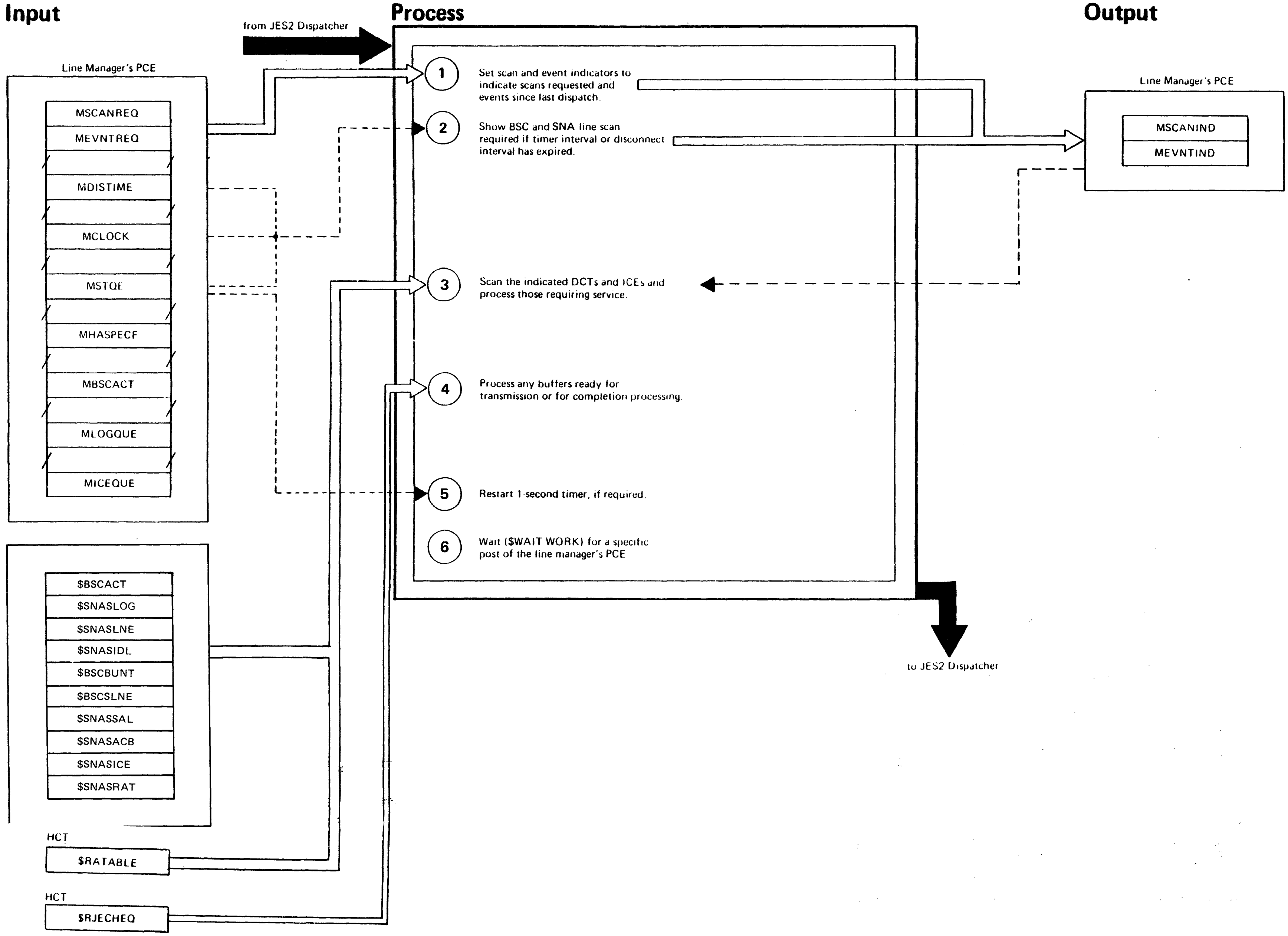
SNA RJE Overview



1. Convert data and control information to SNA equivalents and JES2 work elements.
2. Line Manager: Scan queues, process work elements; initiate sending and receiving.
3. VTAM API: Send subsequent requests and responses.
4. VTAM API: Receive responses, completed requests, and event indications; place corresponding work elements on line manager queues.
5. Monitor results and manage internal states; wait for work when all available work elements processed.
6. Remote Console Processor: Scan inbound and outbound remote message queues and process like data sets.

ASSEMBLY LISTING NAME	ASSEMBLY LISTING LABEL	NARRATIVE
HASPSNA	HASPSNAA	① Work elements include DCTs, representing the JES2/VTAM interface (logon DCT) or remote devices; ICEs, representing sessions; and VTAM RPLs, representing SNA request units to be sent or received. See $EXTP services.
HASPRTAM	HASPMLLM	② The line manager scans DCT, ICE, and buffer queues at each dispatch to determine the next action to be taken. See line manager.
HASPSNA	HASPSPRO	③ The request completion exit routine of JES2's VTAM application program interface invokes VTAM's ACB interface routine to cause actual transmission to continue, once begun by the line manager. See VTAM API Exit Routines.
HASPSNA		④ The VTAM API routines are entered asynchronously by VTAM when requests or responses have been completed by VTAM, when responses for JES2 are received, or when events such as a logon or the termination of VTAM occur. See VTAM API exit routines.
HASPSNA	VMLMPOST	⑤ Both the line manager and JES2 are posted when a VTAM API exit routine is entered by VTAM with new work. See line manager.

HASPRTAM Line Manager



1 Set scan and event indicators to indicate scans requested and events since last dispatch.

2 Show BSC and SNA line scan required if timer interval or disconnect interval has expired.

3 Scan the indicated DCTs and ICEs and process those requiring service.

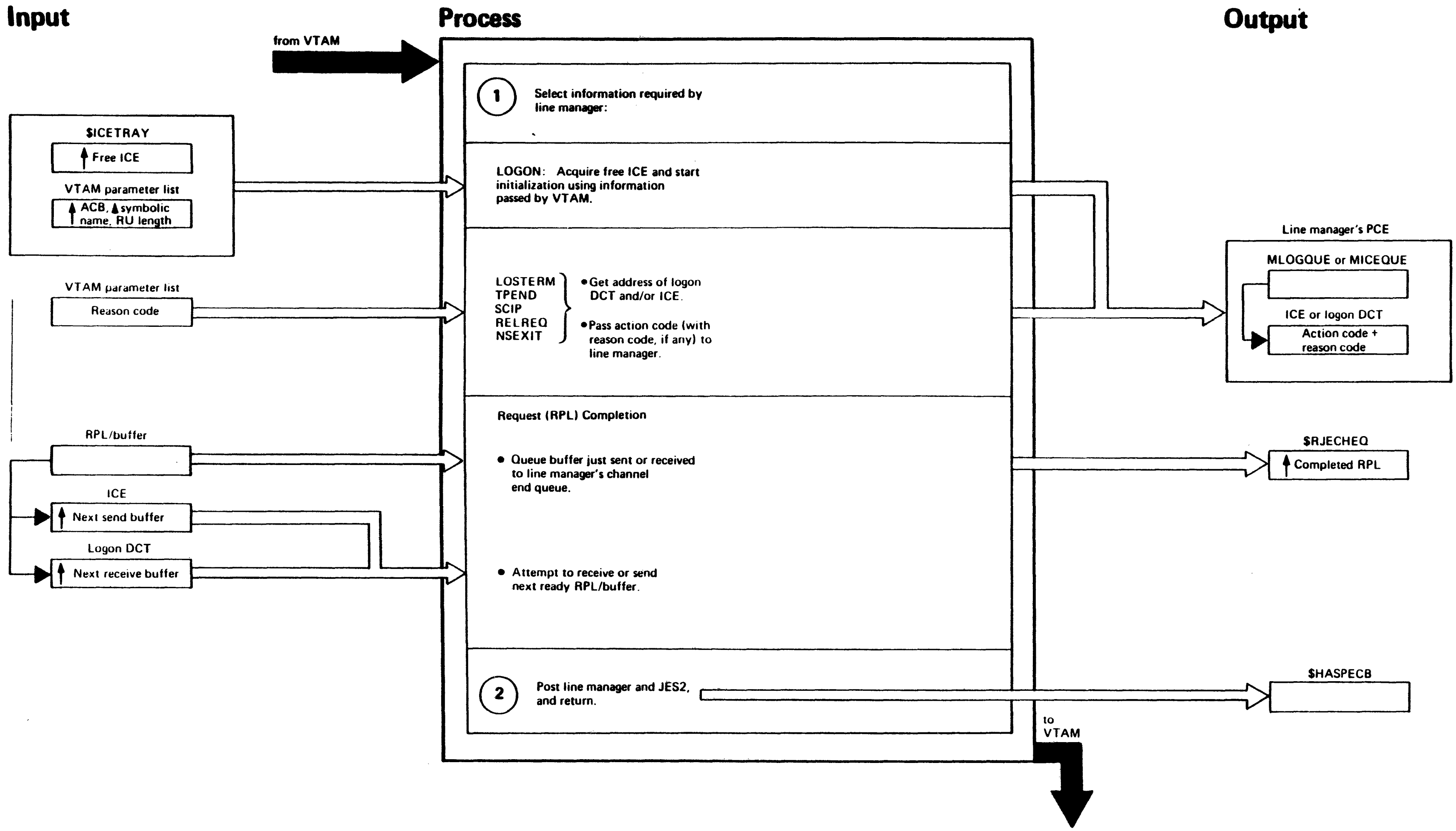
4 Process any buffers ready for transmission or for completion processing.

5 Restart 1-second timer, if required.

6 Wait ($WAIT WORK) for a specific post of the line manager's PCE

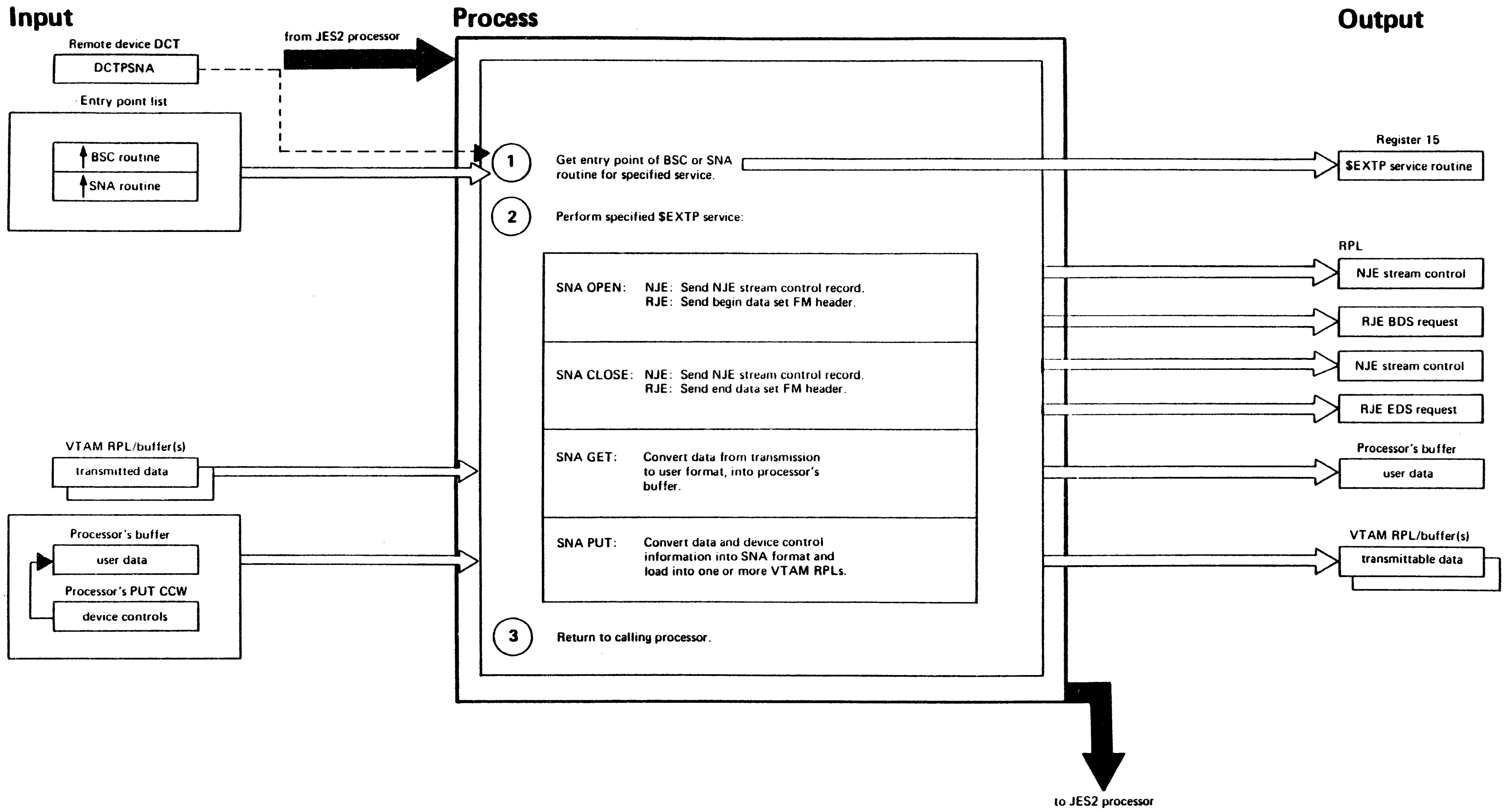
ASSEMBLY LISTING NAME	ASSEMBLY LISTING LABEL	NARRATIVE
HASPRTAM	HASPMLLM	The line manager is queue-driven. Queues and indicators processed by the line manager are updated asynchronously by the command processor, the VTAM API exit routines, the BSC channel end appendage, RTAM subroutines, the open/close ACB subtask, or the occurrence of a timer interrupt or $POST JOT.
HASPRTAM	MSEARCH	① At each dispatch, scan requests and event indications registered by other routines are moved into private line manager fields.
		② The 1-second timer is used to permit inbound jobs to be started; to check on disconnect interval expiration, and to trigger a scan for available devices (DCTs) for which output is available. The 1-second timer is the mechanism through which the delay requested through the $WAITIME initialization parameter is effected.
HASPRTAM HASPBSC HASPSNA HASPSNA HASPSNA HASPSNA HASPBSC HASPSNA HASPSNA HASPSNA	MSCNEXT HASPBACT HASPSLOG HASPSLNE HASPSIDL HASPSUNT HASPBUNT HASPSACB HASPSICE HASPSRAT	③ The order of scan is: ● Active BSC lines ● Active logons ● Active SNA lines ● Idle SNA lines ● Inactive SNA lines or logons ● Inactive BSC lines ● LOGON DCT and ACB completion scan ● ICE exit service scan ● Remote autologon scan
HASPRTAM	MBUFSRCH	④ Depending upon type, buffers are processed at MBSCPROC or MSNAPROC (see listing).
HASPRTAM	MTIMSRCH	⑤ If the timer had expired and timing is requested, $STIMER is issued to restart it.
HASPRTAM	MLLMWAIT	⑥ When a specific post occurs go to MSEARCH to continue processing.

HASPSNA VTAM API Exit Routine


Input
Process
Output
from VTAM
$ICETRAY
Free ICE
VTAM parameter list
ACB, symbolic name, RU length
VTAM parameter list
Reason code
RPL/buffer
ICE
Next send buffer
Logon DCT
Next receive buffer
1
Select information required by line manager:
LOGON: Acquire free ICE and start initialization using information passed by VTAM.
LOSTERM
TPEND
SCIP
RELREQ
NSEXIT
•Get address of logon DCT and/or ICE.
•Pass action code (with reason code, if any) to line manager.
Request (RPL) Completion
• Queue buffer just sent or received to line manager's channel end queue.
• Attempt to receive or send next ready RPL/buffer.
2
Post line manager and JES2, and return.
to VTAM
Line manager's PCE
MLOGQUE or MICEQUE
ICE or logon DCT
Action code + reason code
$RJECHEQ
Completed RPL
$HASPECB

ASSEMBLY LISTING NAME	ASSEMBLY LISTING LABEL	NARRATIVE
		1 The JES2 VTAM API routines are entered from VTAM when the corresponding event occurs. Each routine returns control to VTAM immediately if the ACB is no longer open.
HASPSNA	VEXITLGN	LOGON: The logon is rejected if the maximum number of sessions already exists. Otherwise, the ICE is placed on MICEQUE and session allocation is continued by the line manager at MSNALOGN.
HASPSNA	VEXITLST	LOSTERM: The reason code is placed in the ICE, which is placed on MICEQUE. The line manager (at MICELOST) closes or aborts the session.
		TPEND: The reason code is placed in the DCT, which is placed on MLOGQUE. The line manager (at MLOGTPND) drains, terminates, or aborts the JES2/VTAM interface.
HASPSNA	VEXITSCP	SCIP: Control returns to VTAM except for request recovery (RQR) requests. In that case, the ICE is placed on MICEQUE and the session is aborted (at MICEABRT) by the line manager.
HASPSNA	VEXITRLR	RELREQ: If the symbolic name specified cannot be matched with a logged-on terminal, control returns to VTAM. Otherwise, the ICE is placed on MICEQUE and the line manager (at MICETRAP) subsequently closes the session at the end of the then-current bracket.
		Request (RPL) Completion: This exit is entered as each send or receive request is processed by VTAM. The exit routine attempts to stay within the exit routine, cycling through the VTAM ACB interface routine, until all available requests have been completed.
HASPSNA	VEXITNS	NSEXIT: This routine is scheduled with a network services procedure RU (ignored by JES2) or a CLEANUP RU. For a CLEANUP RU, the ICE is placed on the MICEQUE queue, and the line manager (at MICENS) schedules the freeing of all resources used by the session.

HASPSNA $EXTP Services



1 Get entry point of BSC or SNA routine for specified service.

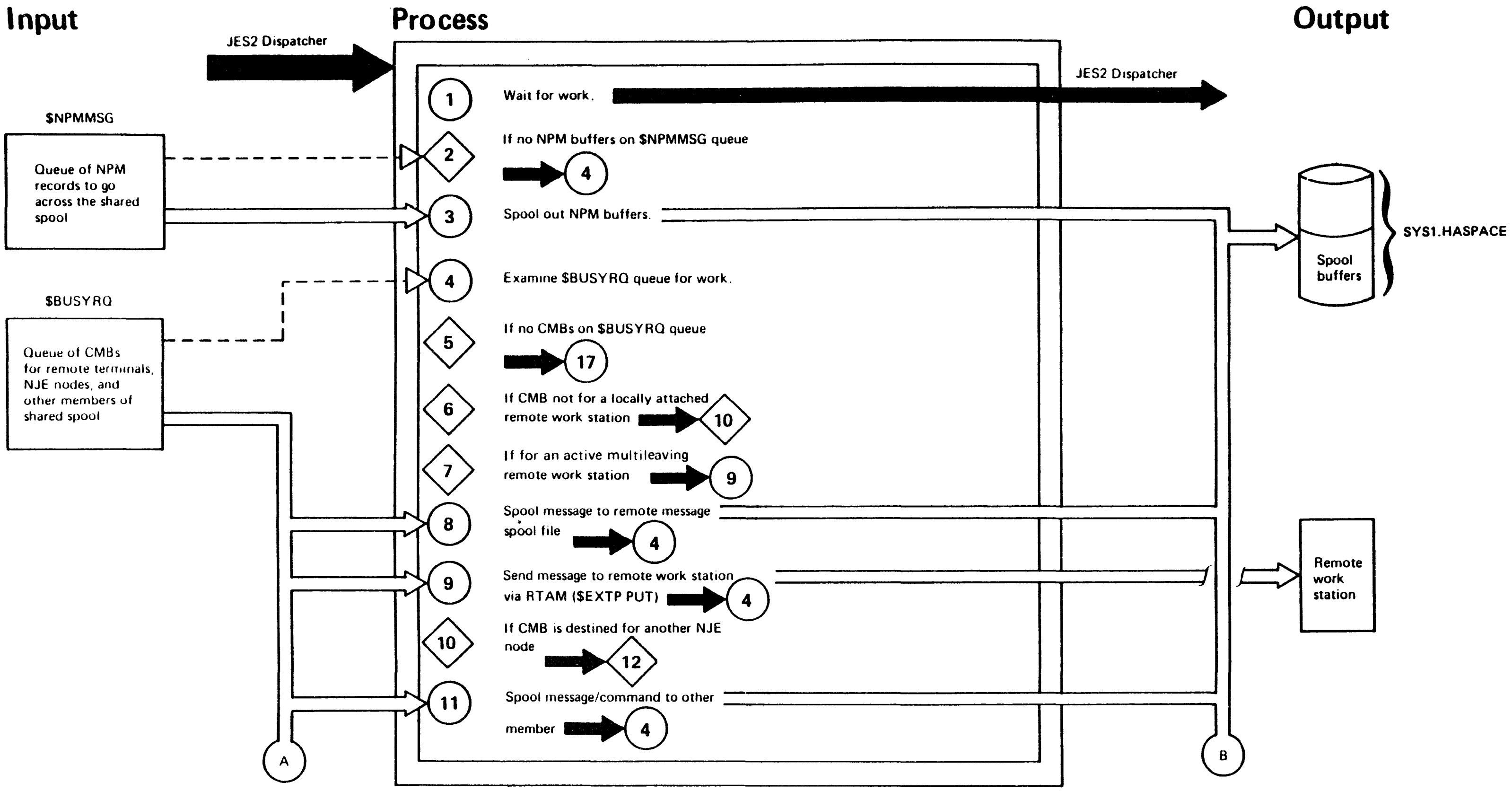
2 Perform specified $EXTP service:

SNA OPEN:	NJE: Send NJE stream control record. RJE: Send begin data set FM header.
SNA CLOSE:	NJE: Send NJE stream control record. RJE: Send end data set FM header.
SNA GET:	Convert data from transmission to user format, into processor's buffer.
SNA PUT:	Convert data and device control information into SNA format and load into one or more VTAM RPLs.

3 Return to calling processor.

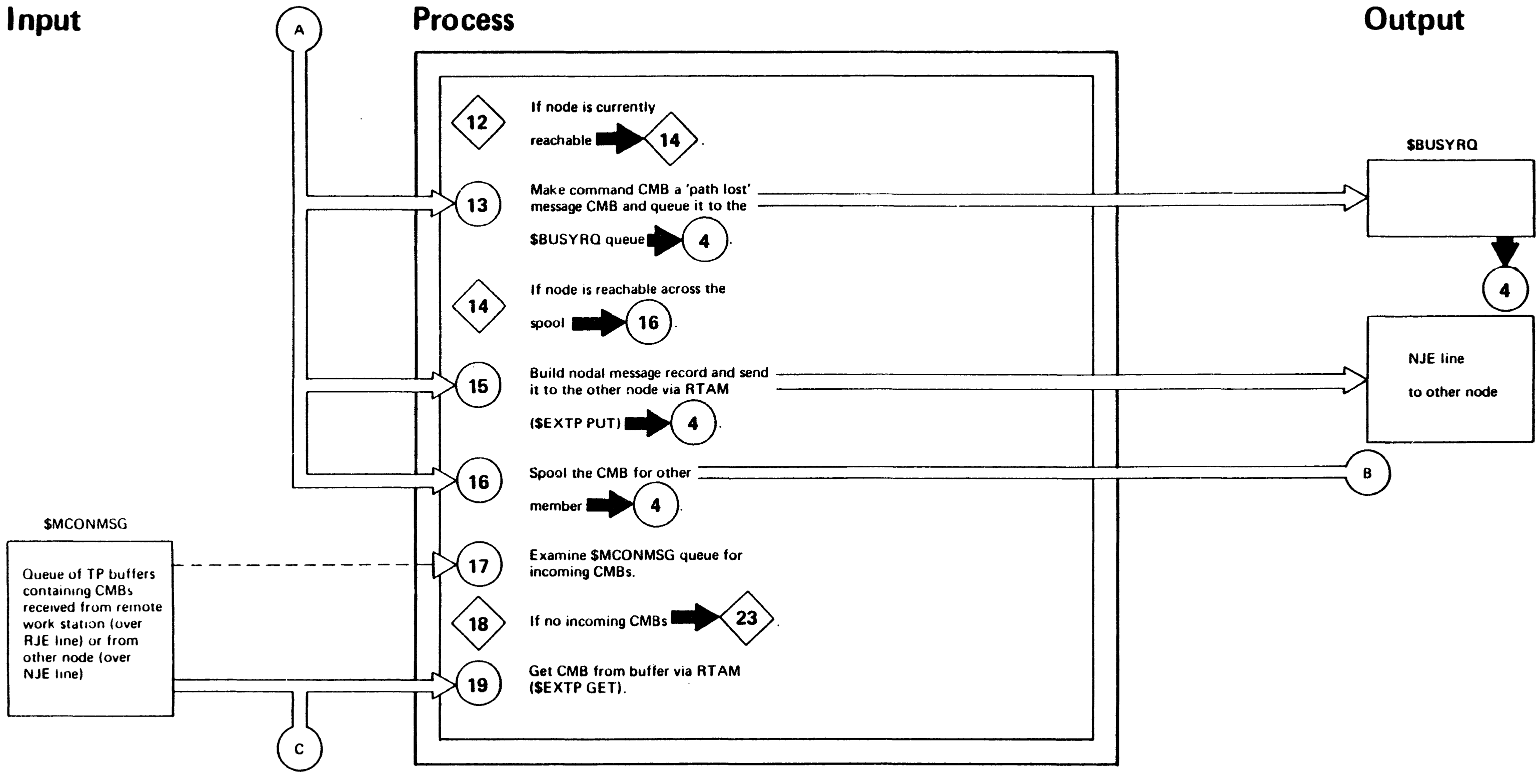
ASSEMBLY LISTING NAME	ASSEMBLY LISTING LABEL	NARRATIVE
HASPNUC	HASPEXTP	① Return to the caller is immediate if the specified device is aborting (DCTABORT on), or - except for $EXTP CLOSE the device is being flushed (DCTFLUSH on). Otherwise, registers are stored in the calling processor's PCE and are reinitialized for RTAM.
HASPSNA	SNAOPEN	② SNA OPEN for RJE: The BDS header is sent only for 'outbound open' for a print or punch processor. SNA OPEN for NJE: If called by a network receiver device, a 'permission to allocate granted' stream control record is sent. If called by a network transmitter device, a 'request permission to allocate' stream control record is sent.
HASPSNA	SNACLOSE	SNA CLOSE for RJE: The data set may be ended (EDS) or aborted (ADS), depending upon whether the remote device is being flushed. SNA CLOSE for NJE: If called by a network receiver device, an 'acknowledge end-of-transmission' stream control record is sent if a $EXTP CLOSE macro instruction was issued; a 'receiver cancel' stream control record is sent if a $EXTP NCLOSE macro instruction was issued. If called by a network transmitter device, and 'end-of-transmission' data record is sent if a $EXTP CLOSE macro instruction was issued; a 'transmitter cancel' data record is sent if a $EXTP NCLOSE macro instruction was issued.
HASPSNA	SNAGET	SNA GET: Data is decompressed, translated, or treated as transparent as applicable. The contents of successive request units (each contained in a VTAM RPL) are processed until the processor buffer is full.
HASPSNA	SNAPUT	SNA PUT: Data is compressed, translated, or treated as transparent as applicable. Device control information is converted into standard character string control sequences. Successive request units are filled and the corresponding VTAM RPLs are scheduled for transmission.

HASPRTAM Remote Console Processor (Part 1 of 3)


Input
Process
Output
JES2 Dispatcher
$NPMMSG
Queue of NPM records to go across the shared spool
$BUSYRQ
Queue of CMBs for remote terminals, NJE nodes, and other members of shared spool
1 Wait for work.
JES2 Dispatcher
2 If no NPM buffers on $NPMMSG queue
4
3 Spool out NPM buffers.
4 Examine $BUSYRQ queue for work.
5 If no CMBs on $BUSYRQ queue
17
6 If CMB not for a locally attached remote work station
10
7 If for an active multileaving remote work station
9
8 Spool message to remote message spool file
4
9 Send message to remote work station via RTAM ($EXTP PUT)
4
10 If CMB is destined for another NJE node
12
11 Spool message/command to other member
4
A
B
Spool buffers
SYS1.HASPACE
Remote work station

Assembly Listing Name	Assembly Listing Label	Narrative
HASPRTAM	MCSX	2 Check $NPMMSG queue for input (network path manager records going across the spool).
		3 Invoke MCSPUTX subroutine to write to spool.
HASPRTAM	MCS	4 Dequeue console message buffer (CMB) from $NPMMSG queue and invoke the appropriate RCP output function.
HASPRTAM	MCOSPOOL	8 If the CMB is for a remote work station which is non-multileaving or is currently inactive, it is spooled out to the message spool file where later it is picked up for printing by the remote print/punch processor.
HASPRTAM	MCSTRL	9 If the CMB is for an active multileaving remote work station, transmit the message to the remote work station via the $EXTP PUT interface.
HASPRTAM	MCSSO	11 If the CMB is for another member of a shared spool complex, invoke the spool output routine to output the CMB to the appropriate member.

HASPRTAM Remote Console Processor (Part 2 of 3)



Input
Process
Output
A
$MCONMSG
Queue of TP buffers containing CMBs received from remote work station (over RJE line) or from other node (over NJE line)
C
$BUSYRQ
4
NJE line
to other node
B

12 If node is currently reachable ➡ 14.

13 Make command CMB a 'path lost' message CMB and queue it to the $BUSYRQ queue ➡ 4.

14 If node is reachable across the spool ➡ 16.

15 Build nodal message record and send it to the other node via RTAM ($EXTP PUT) ➡ 4.

16 Spool the CMB for other member ➡ 4.

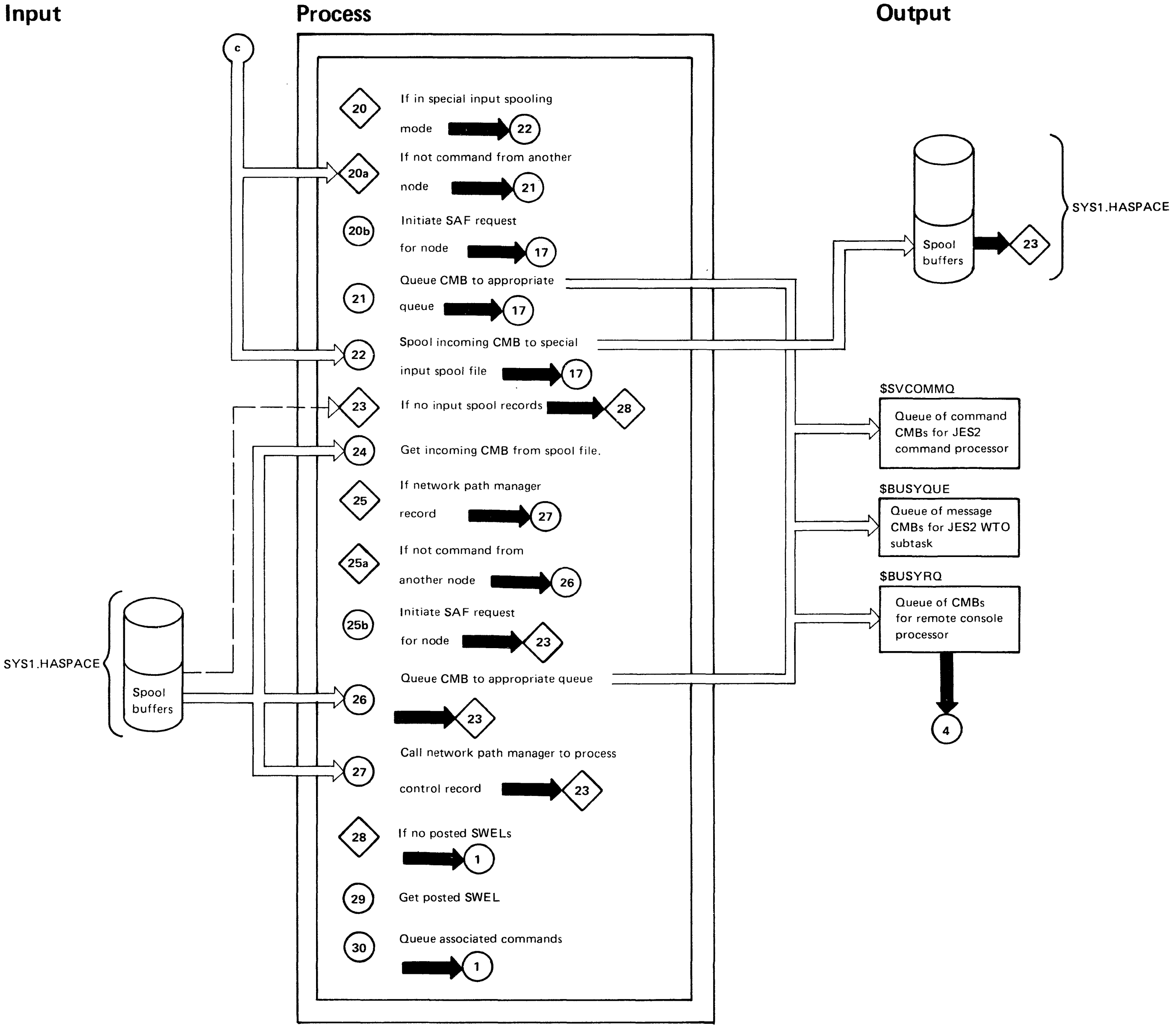
17 Examine $MCONMSG queue for incoming CMBs.

18 If no incoming CMBs ➡ 23.

19 Get CMB from buffer via RTAM ($EXTP GET).

Assembly Listing Name	Assembly Listing Label	Narrative
HASPRTAM	MCSNNTUP	13 If the CMB contains a command destined for another NJE node which is currently unreachable, the CMB is converted into a 'path lost' message destined for the originator of the command and requeued to the $BUSYRQ queue.
HASPRTAM	MCSNOX MCSSOA	15 If the CMB is destined for another NJE node which is reachable via another member of the shared spool, invoke the spool output routine to output the CMB to the appropriate member.
HASPRTAM	MCSNOP MCSNOPA	16 If the CMB contains a command or a message for another NJE node reachable via an NJE line, build a nodal message record (NMR) and transmit to the next node via the $EXTP PUT interface with RTAM.
HASPRTAM	MCINSI	17 Check for incoming TP buffers on the $MCONMSG queue. Buffers have been queued by the multileaving line manager after being received over an RJE or NJE line.
HASPRTAM	MCINGET MCINR	19 Get a CMB from the buffer via a $EXTP GET interface with RTAM.

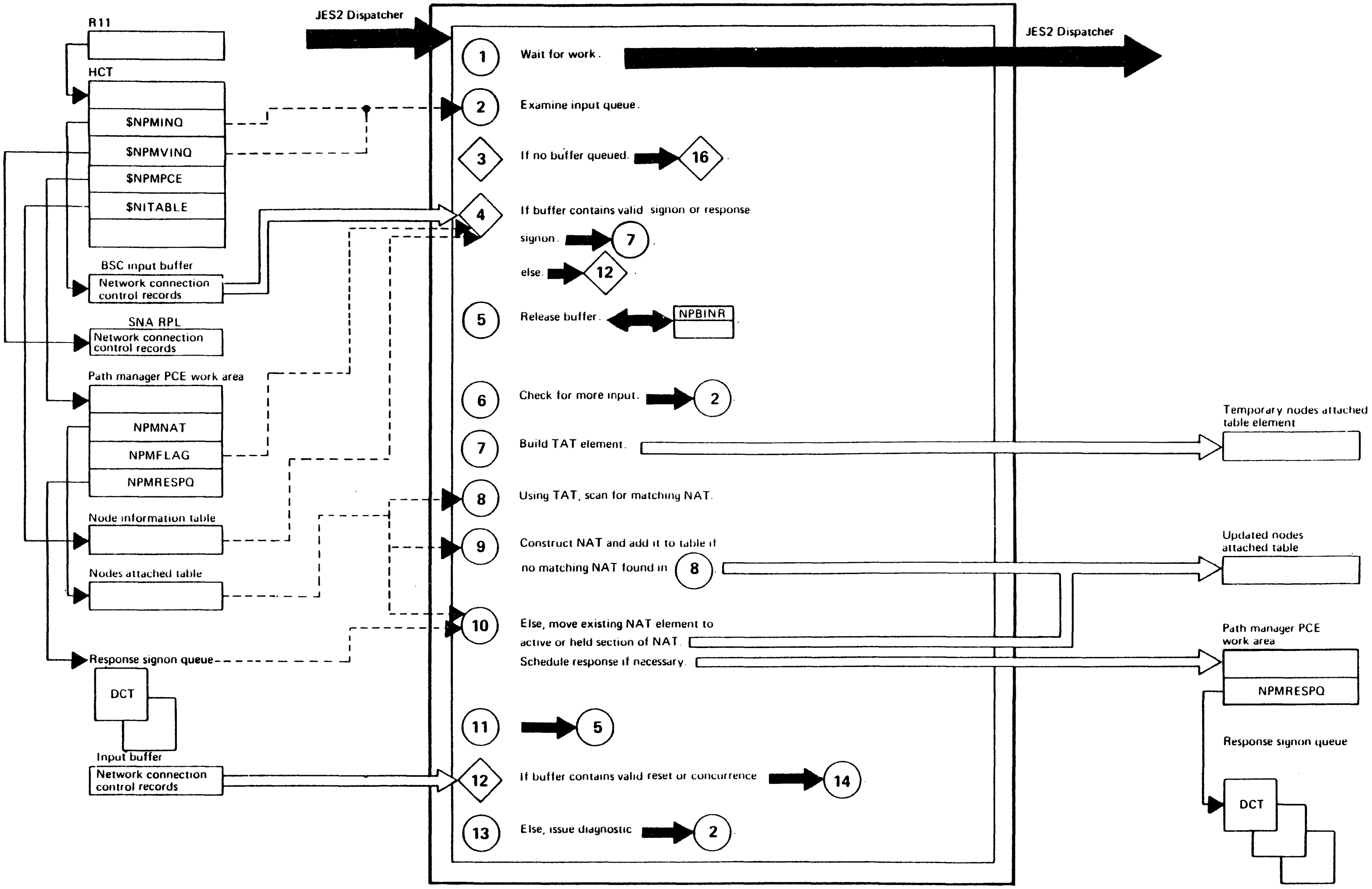
HASPRTAM Remote Console Processor (Part 3 of 3)


Input
Process
Output
c
20 If in special input spooling mode → 22
20a If not command from another node → 21
20b Initiate SAF request for node → 17
21 Queue CMB to appropriate queue → 17
22 Spool incoming CMB to special input spool file → 17
23 If no input spool records → 28
24 Get incoming CMB from spool file.
25 If network path manager record → 27
25a If not command from another node → 26
25b Initiate SAF request for node → 23
26 Queue CMB to appropriate queue → 23
27 Call network path manager to process control record → 23
28 If no posted SWELs → 1
29 Get posted SWEL
30 Queue associated commands → 1
SYS1.HASPACE
Spool buffers
SYS1.HASPACE
Spool buffers → 23
$SVCOMMQ
Queue of command CMBs for JES2 command processor
$BUSYQUE
Queue of message CMBs for JES2 WTO subtask
$BUSYRQ
Queue of CMBs for remote console processor
4

Assembly Listing Name	Assembly Listing Label	Narrative
HASPRTAM	MCINP	21 If not currently in special input spooling mode, queue the CMB to the appropriate CMB queue for further processing: ● If for a remote, another node, or another shared spool member, queue the CMB to $BUSYQUE for RCP output processing. ● If for the local system, queue the CMB to $BUSYQUE for a write-to operator and additionally to the $SVCOMMQ (if the CMB contains a command) for the command processor.
HASPRTAM	MCINSP	22 If in special input spooling mode (out of CMBs or currently receiving/spooling a multiple line write-to-operator), invoke the spool output function to spool the CMB to special input spool file.
HASPRTAM	MCISI	23 Check for incoming records on the spool (from other members or CMBs spooled while in special input spooling mode).
HASPRTAM	MCISIRA	24 Invoke spooling-in function to get a record from spool.
HASPRTAM	MCISIG	26 If CMB, perform CMB queuing logic as described in 21
HASPRTAM	MCISIOK	27 If NPM record, invoke the HASPNET routine (HASPNBUF) to process the record.

JES2 HASPNPM Network Path Manager Processor (Part 1 of 3)

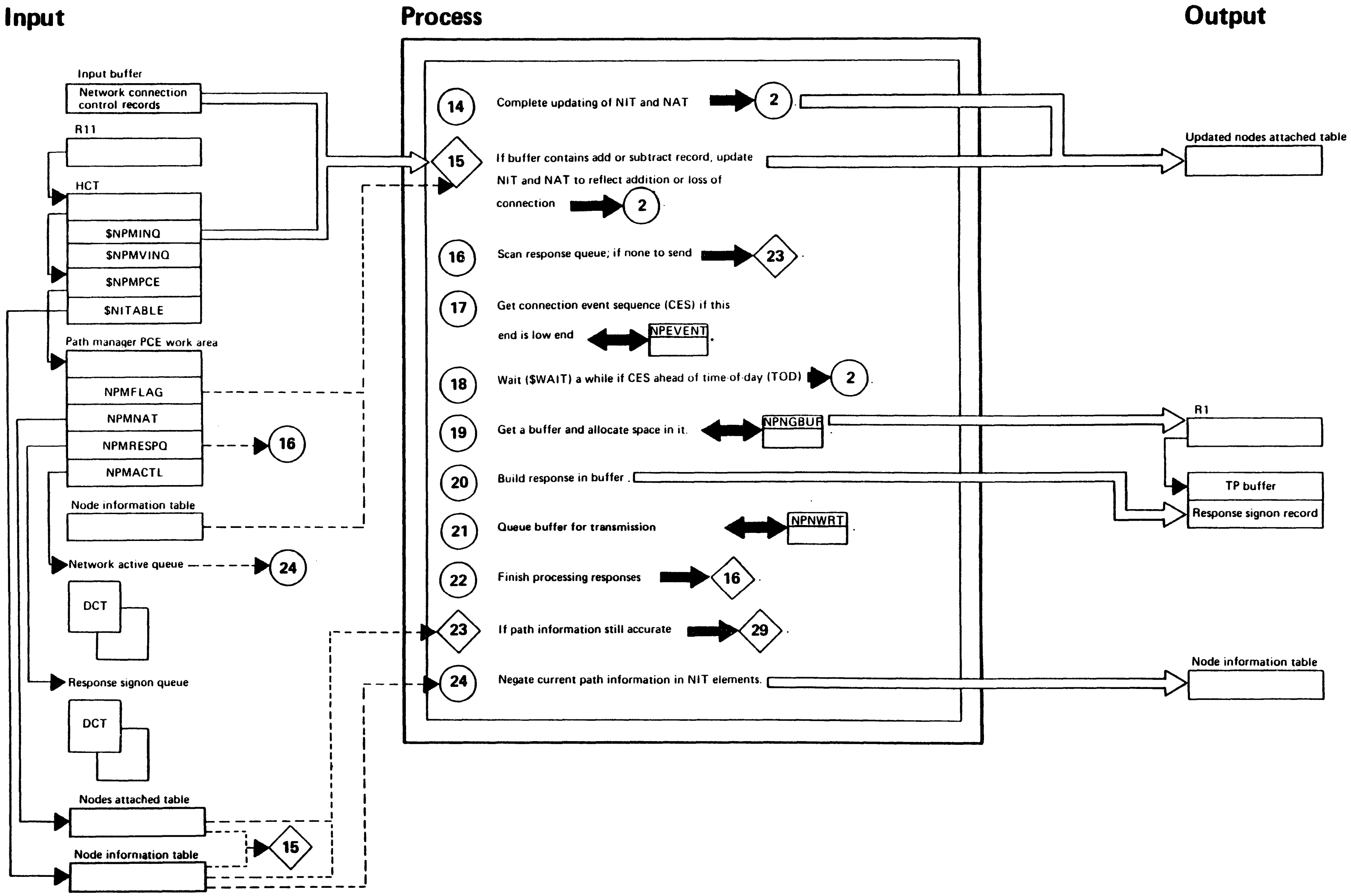
Input **Process** **Output**


R11
HCT
$NPMINQ
$NPMVINQ
$NPMPCE
$NITABLE
BSC input buffer
Network connection control records
SNA RPL
Network connection control records
Path manager PCE work area
NPMNAT
NPMFLAG
NPMRESPQ
Node information table
Nodes attached table
Response signon queue
DCT
Input buffer
Network connection control records
JES2 Dispatcher
1 Wait for work.
JES2 Dispatcher
2 Examine input queue.
3 If no buffer queued. 16
4 If buffer contains valid signon or response
signon. 7
else. 12
5 Release buffer. NPBINR
6 Check for more input. 2
7 Build TAT element.
8 Using TAT, scan for matching NAT.
9 Construct NAT and add it to table if no matching NAT found in 8
10 Else, move existing NAT element to active or held section of NAT.
Schedule response if necessary.
11 5
12 If buffer contains valid reset or concurrence 14
13 Else, issue diagnostic 2
Temporary nodes attached table element
Updated nodes attached table
Path manager PCE work area
NPMRESPQ
Response signon queue
DCT

ASSEMBLY LISTING NAME	ASSEMBLY LISTING LABEL	NARRATIVE
HASPNPM	NPL	2 Check $NPMINQ for input.
		4 Invoke service routine to handle network connection control record in input buffer as follows:
HASPNPM	NPBINR	5 For a SNA RPL, NPBINR branches to NPVBINR to dechain the RPL from $NPMVINQ.
HASPNPM	NPRI NPRR NPRE NPRC NPRA NPRS	12 15 ● Initial signon ● Response to signon ● Reset signon ● Concur signon ● Add connection ● Subtract connection
HASPNPM	NPRINDL	7 Information from the NCC record in the input buffer is used to build a temporary nodes attached table element (TAT).
HASPNPM	NPRIQD	8 The nodes attached table is scanned to determine whether an entry representing the connection already exists.

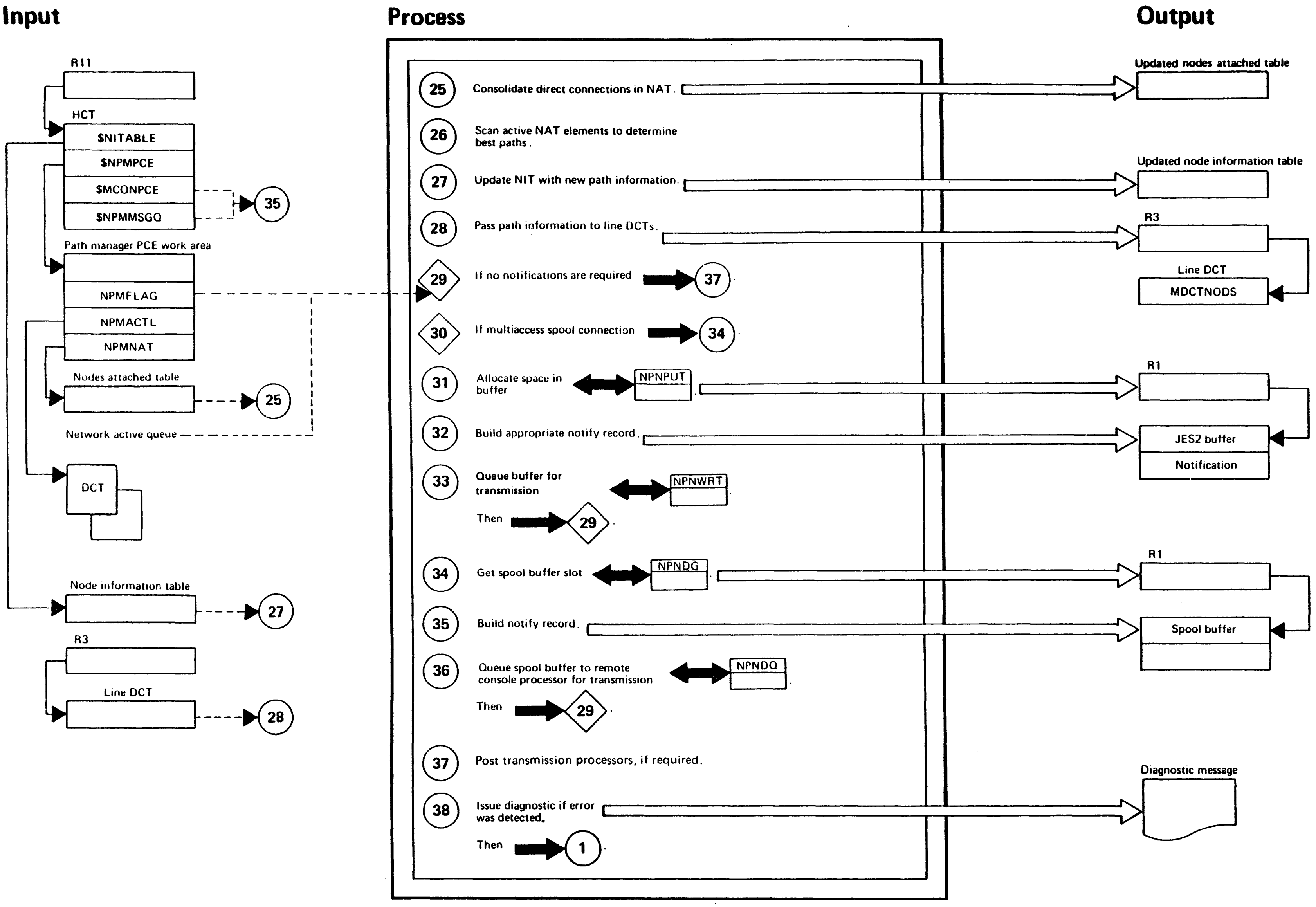
ASSEMBLY LISTING NAME	ASSEMBLY LISTING LABEL	NARRATIVE
	NPRINATE NPRINATA	9 Create NAT element from TAT element if none exists and insert it in held section of NAT.
HASPNPM	NPRINATF NPRIACT NPRIUNC	10 If the NAT is found in the active section of the table, this connection represents a parallel trunk and the DCT is placed on the multitrunk queue. If the NAT found is in the unconnected section of the table, it is placed in the held section pending further processing.
HASPNPM	NPDDMSG	13 The address of a message pointer and/or node name is supplied to NPDDMSG.

JES2 HASPNPM Network Path Manager Processor (Part 2 of 3)


Input
Process
Output
Input buffer
Network connection control records
R11
HCT
$NPMINQ
$NPMVINQ
$NPMPCE
$NITABLE
Path manager PCE work area
NPMFLAG
NPMNAT
NPMRESPQ
16
NPMACTL
Node information table
Network active queue
24
DCT
Response signon queue
DCT
Nodes attached table
Node information table
15
14 Complete updating of NIT and NAT 2
15 If buffer contains add or subtract record, update NIT and NAT to reflect addition or loss of connection 2
16 Scan response queue; if none to send 23
17 Get connection event sequence (CES) if this end is low end NPEVENT
18 Wait ($WAIT) a while if CES ahead of time-of-day (TOD) 2
19 Get a buffer and allocate space in it. NPNGBUF
20 Build response in buffer.
21 Queue buffer for transmission NPNWRT
22 Finish processing responses 16
23 If path information still accurate 29
24 Negate current path information in NIT elements.
Updated nodes attached table
R1
TP buffer
Response signon record
Node information table

ASSEMBLY LISTING NAME	ASSEMBLY LISTING LABEL	NARRATIVE
HASPNPM	NPRC NPRE NPRCPI NPRICONC	14 Verify the connection event sequence (CES). Place DCT on network active queue; if necessary, update NIT and NAT as required.
HASPNPM	NPRL	17 Low end is determined based on a comparison of EBCDIC node names.
HASPNPM	NPNGBF	19 For SNA, NPNGBUF branches to NPNGVBF to get and initialize an SNA RPL.
HASPNPM	NPRL	22 For predefined connections, the DCT is queued to NPMACTL and thus made active before more requests are processed.
HASPNPM	NPNR	23 The full-path routine is entered any time activity on the network has been such that information in the node information table (NIT) and nodes attached table (NAT) might no longer be valid.

JES2 HASPNPM Network Path Manager Processor (Part 3 of 3)



25 Consolidate direct connections in NAT.

26 Scan active NAT elements to determine best paths.

27 Update NIT with new path information.

28 Pass path information to line DCTs.

29 If no notifications are required → 37.

30 If multiaccess spool connection → 34.

31 Allocate space in buffer

32 Build appropriate notify record.

33 Queue buffer for transmission

Then → 29.

34 Get spool buffer slot

35 Build notify record.

36 Queue spool buffer to remote console processor for transmission

Then → 29.

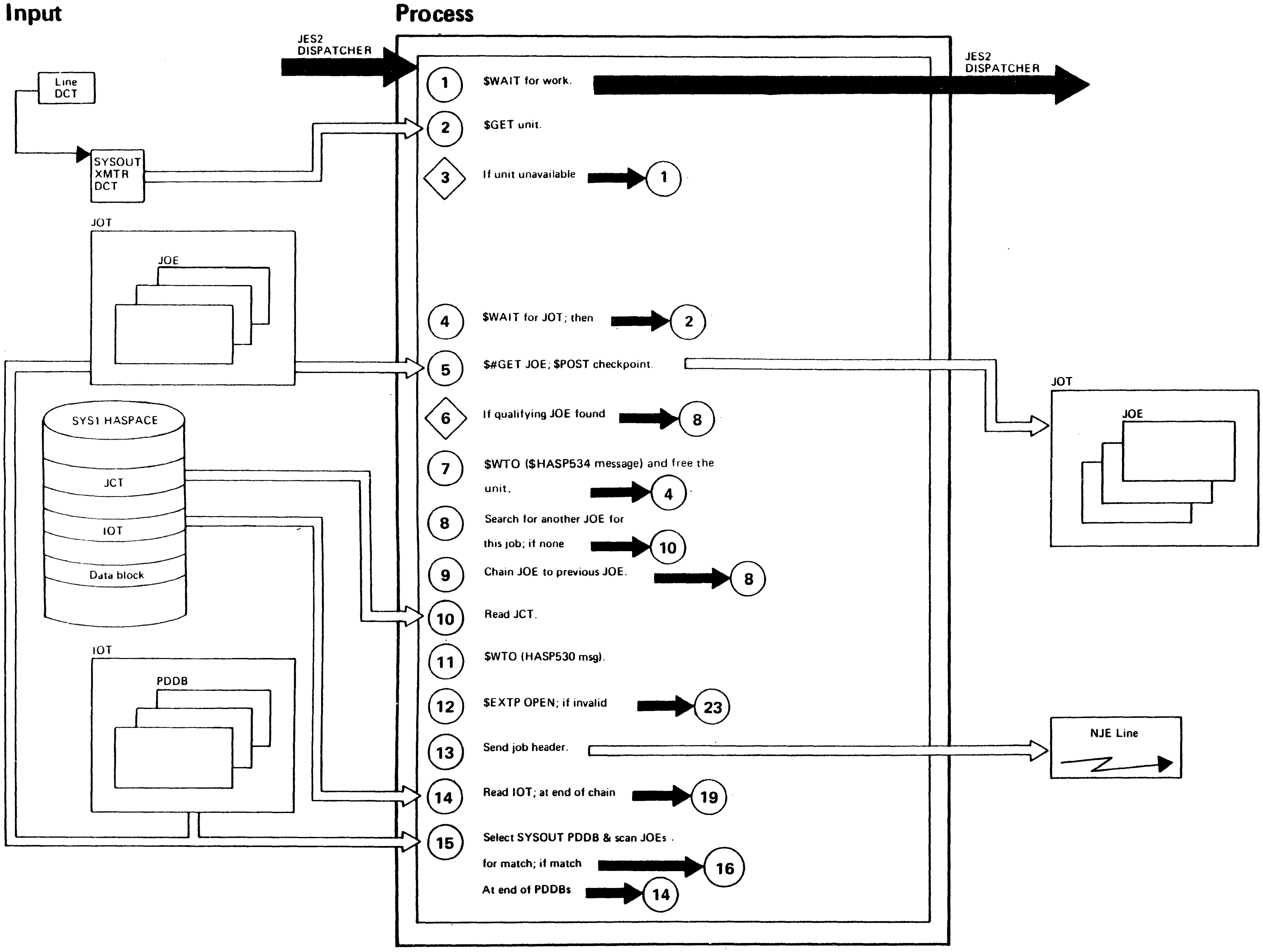
37 Post transmission processors, if required.

38 Issue diagnostic if error was detected.

Then → 1.

ASSEMBLY LISTING NAME	ASSEMBLY LISTING LABEL	NARRATIVE
HASPNPM	NPATHP	28 Whether or not transmissions are to be allowed over a given line is determined by the setting of a mask field (MDCTNODS).
HASPNPM	NPNPUT	31 For an SNA RPL, NPNPUT branches to NPNVPUT to allocate space in the SNA buffer.
HASPNPM	NPNDCTL NPNNE NPNOTL	32 Build reset record. Build concurrence record. Build add/subtract connection record.
HASPNPM	NPND	34 Build notify for attached shared spool member.
HASPNPM	NPNEND	37 Post JOB and SYSOUT transmitters if any new paths were made available.

JES2 HASPNET – SYSOUT Transmission Processor (Part 1 of 2)


Input
Process
JES2 DISPATCHER
Line DCT
SYSOUT XMTR DCT
JOT
JOE
SYS1 HASPACE
JCT
IOT
Data block
IOT
PDDB
1 $WAIT for work.
JES2 DISPATCHER
2 $GET unit.
3 If unit unavailable → 1
4 $WAIT for JOT; then → 2
5 $#GET JOE; $POST checkpoint.
JOT
JOE
6 If qualifying JOE found → 8
7 $WTO ($HASP534 message) and free the unit. → 4
8 Search for another JOE for this job; if none → 10
9 Chain JOE to previous JOE. → 8
10 Read JCT.
11 $WTO (HASP530 msg).
12 $EXTP OPEN; if invalid → 23
13 Send job header.
NJE Line
14 Read IOT; at end of chain → 19
15 Select SYSOUT PDDB & scan JOEs for match; if match → 16
At end of PDDBs → 14

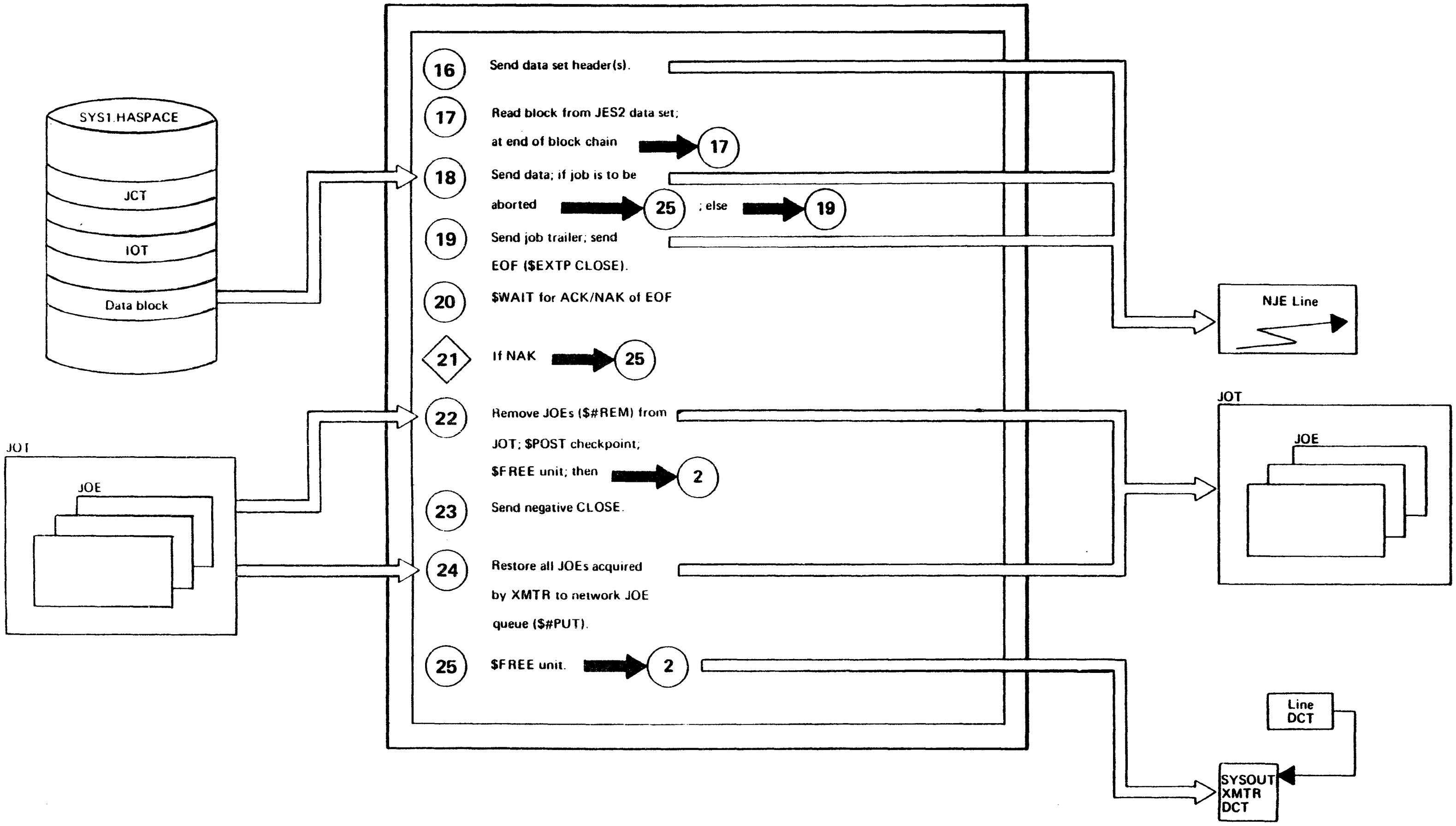
ASSEMBLY LISTING NAME	ASSEMBLY LISTING LABEL	NARRATIVE
HASPNET	NSTSWAIT	1 Wait for initial go ahead.
HASPNET	NSTSUNIT	3 Will be unable to get unit if SYSOUT XMTR has been drained.
HASPNET		5 The $POST of the checkpoint processor is performed within the $#GET subroutine.
HASPNET		6 The JOE qualifies if the destination node is one of the nodes specified in MDCTNODS.
HASPNET	NST$MSG	7 Issue the 'Ln. STn inactive' message.
HASPNET	NSTGOTJB	8 Take control of any other JOEs for the same job which can be transmitted on this NJE line.
HASPNET	NSTENDQ	10 The $#JCT macro is used so that multiple processors may share the JCT for a job
HASPNET		11 Issue message to identify job for which SYSOUT transmission is about to begin.
HASPNET		12 Request permission to transmit through RTAM; if granted, the receiving end is prepared to accept transmission.
HASPNET	NSTHXMIT	13 The NJE job header was stored in the JCT when the job entered the system.
HASPNET	NSTIOTRD	14 Read both regular and spin IOTs.
HASPNET	NSTNEXTJ	15 PDDB and JOE must match on DEST and CLASS, PDDB must also match characteristics - JOE.

JES2 HASPNET – SYSOUT Transmission Processor (Part 2 of 2)

Input

Process

Output



16 Send data set header(s).

17 Read block from JES2 data set; at end of block chain → 17

18 Send data; if job is to be aborted → 25 ; else → 19

19 Send job trailer; send EOF ($EXTP CLOSE).

20 $WAIT for ACK/NAK of EOF

21 If NAK → 25

22 Remove JOEs ($#REM) from JOT; $POST checkpoint; $FREE unit; then → 2

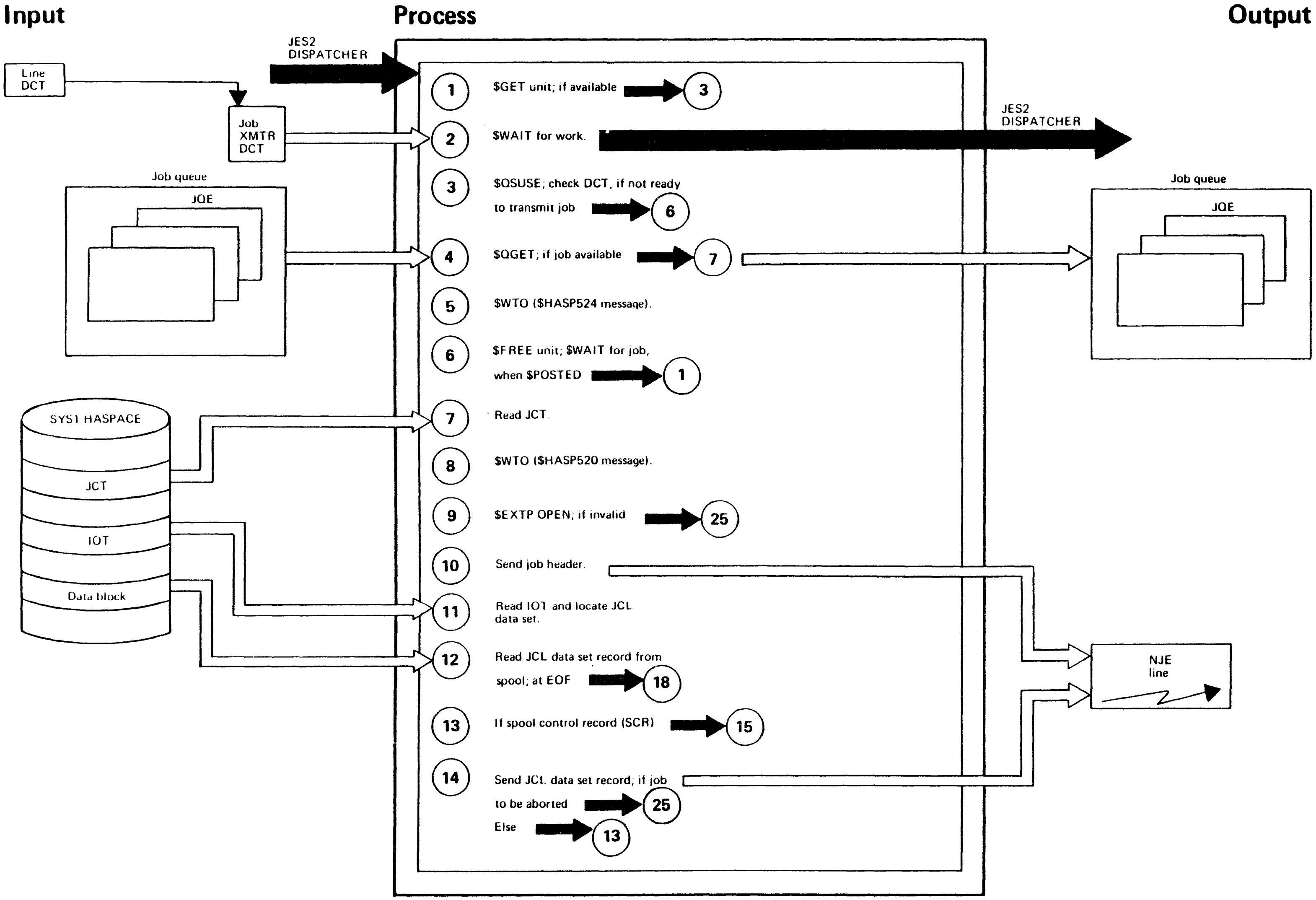
23 Send negative CLOSE.

24 Restore all JOEs acquired by XMTR to network JOE queue ($#PUT).

25 $FREE unit. → 2

ASSEMBLY LISTING NAME	ASSEMBLY LISTING LABEL	NARRATIVE
HASPNET	NSTDSHDR	16 Build data set header (or get from file if job did not execute locally) May have multiple headers if multiple matching PDDBs with different destinations.
HASPNET	NSTCKOPR	18 While reading data set and sending, continue to check transmitter DCT for $C and $E command entered by operator.
HASPNET	NSTENJOB	19 Job trailer was stored in the JCT when the job entered the system
		22 Free the JOEs acquired earlier; write network SYSOUT transmission SMF record; release resources acquired for JOB.
HASPNET	NSTNCLOS	23 Transmission error or job flushed by receiving node; call RTAM to perform "negative CLOSE."
HASPNET	NSTUNFLG	24 Put JOE(s) back in active queue ($#PUT), checkpoint JOE, issue JOB RESTARTED message ($HASP532)

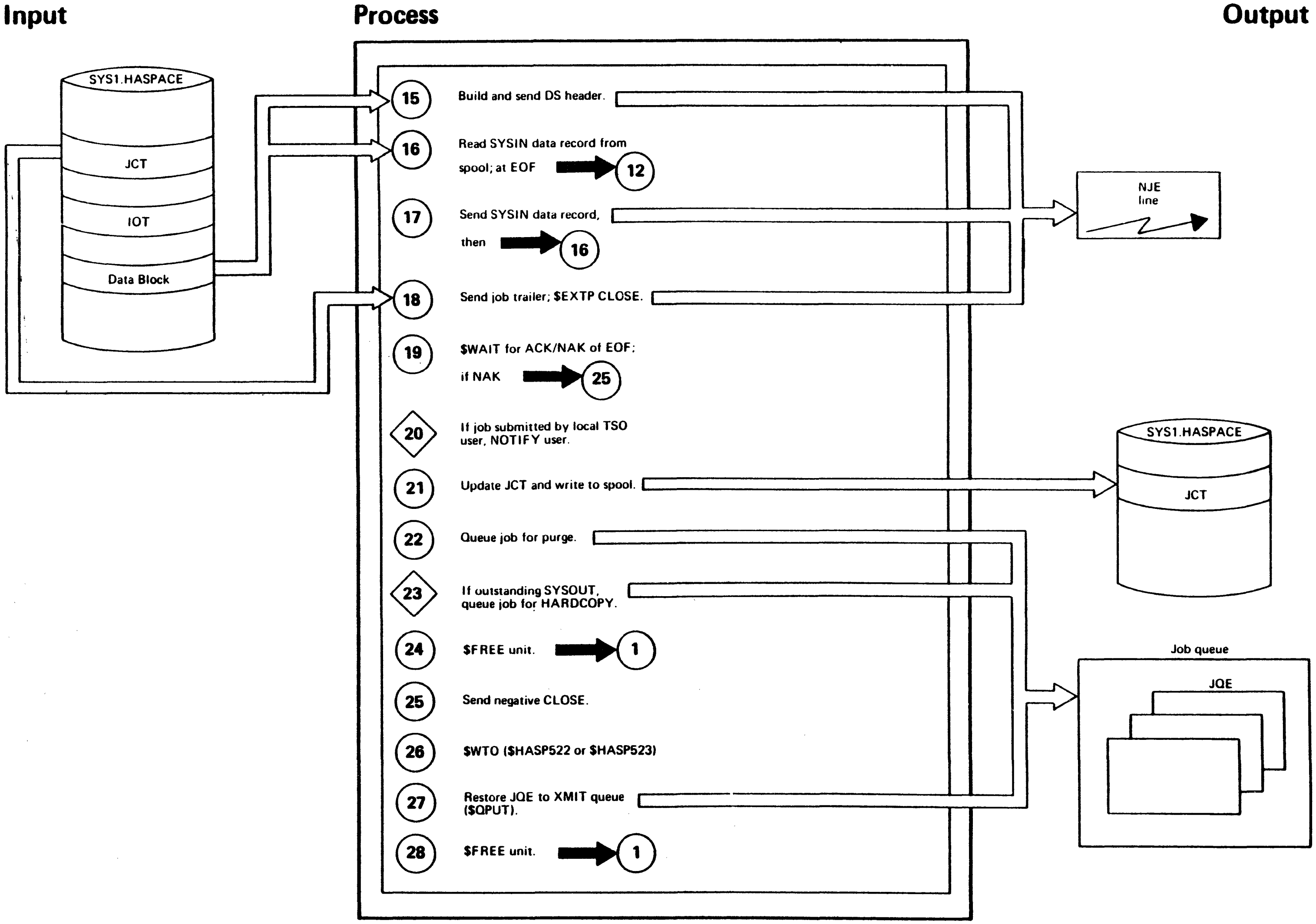
JES2 HASPNET – Job Transmission Processor (Part 1 of 2)



1 $GET unit; if available → 3
2 $WAIT for work.
3 $QSUSE; check DCT, if not ready to transmit job → 6
4 $QGET; if job available → 7
5 $WTO ($HASP524 message).
6 $FREE unit; $WAIT for job, when $POSTED → 1
7 Read JCT.
8 $WTO ($HASP520 message).
9 $EXTP OPEN; if invalid → 25
10 Send job header.
11 Read IOT and locate JCL data set.
12 Read JCL data set record from spool; at EOF → 18
13 If spool control record (SCR) → 15
14 Send JCL data set record; if job to be aborted → 25
Else → 13

ASSEMBLY LISTING NAME	ASSEMBLY LISTING LABEL		NARRATIVE
HASPNET	JXMTINIT	1	Will be unable to get unit if JOB XMTR has been drained.
HASPNET		2	Wait for device to be made available.
HASPNET	JGOTUNIT	3	Issue $QSUSE to get control of shared queues.
HASPNET		4	$QGET routine scans the $XMIT job queue for a job destined for a node flagged in MDCTNODS.
HASPNET		5	Issue "Ln.JTn inactive" message.
HASPNET	JXMTWAIT	6	Wait for a job to be added to the job queue.
HASPNET	JGOTJOB	7	If the read is unsuccessful, a disastrous error message is issued and the job is aborted.
HASPNET	JGOTJCT1	8	Issue message to identify job and line.
HASPNET		9	Request permission to transmit; if granted, the receiving end is prepared to accept transmission.
HASPNET	JXMTHDR	10	The NJE job header was stored in the JCT when the job entered the system.
HASPNET		11	A pointer to the job's JCL data set is contained in the first IOT.
HASPNET	JNEXTREC	13	The JCL stream contains a spool control record at each point where a SYSIN data set should be. The SCR points to the SYSIN data set on the spool.
HASPNET	JPUTREC	14	Send data over NJE line via $EXTP PUT. Continue to check flags in DCT in case job is to be aborted ($C or $E by operator).

JES2 HASPNET – Job Transmission Processor (Part 2 of 2)



15 Build and send DS header.

16 Read SYSIN data record from spool; at EOF → 12

17 Send SYSIN data record, then → 16

18 Send job trailer; $EXTP CLOSE.

19 $WAIT for ACK/NAK of EOF; if NAK → 25

20 If job submitted by local TSO user, NOTIFY user.

21 Update JCT and write to spool.

22 Queue job for purge.

23 If outstanding SYSOUT, queue job for HARDCOPY.

24 $FREE unit. → 1

25 Send negative CLOSE.

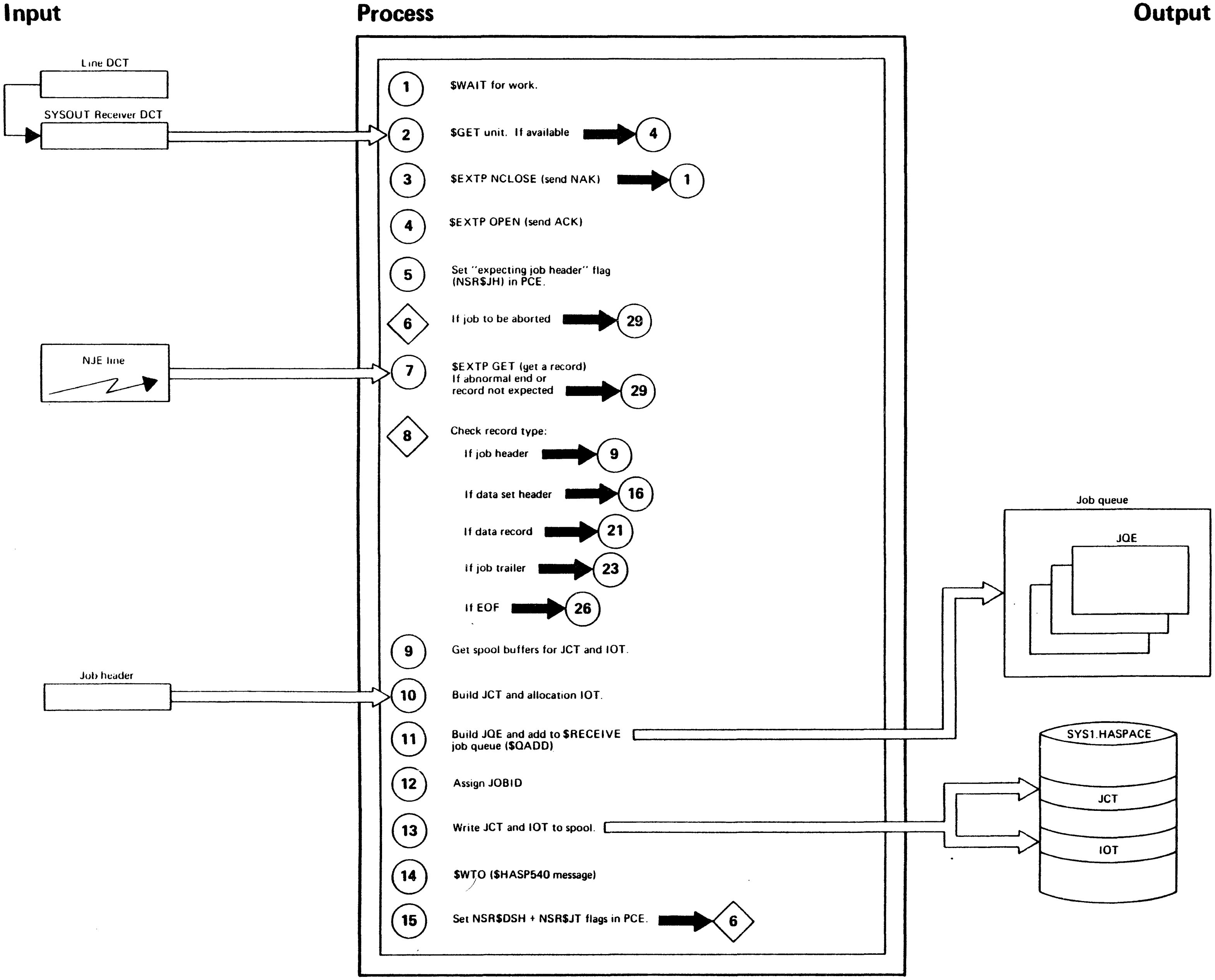
26 $WTO ($HASP522 or $HASP523)

27 Restore JQE to XMIT queue ($QPUT).

28 $FREE unit. → 1

ASSEMBLY LISTING NAME	ASSEMBLY LISTING LABEL	NARRATIVE
HASPNET	JCNTLREC	15 DS header is built from information in spool control record.
HASPNET	JXMTHDR	18 Job trailer was stored in the JCT when the job entered the system.
HASPNET	JNOTIFY	20 Send message to TSO user via $WTO to notify user that job is leaving this node.
HASPNET	JUPDTJCT	21 Update JCT with the job transmitter's ending time and accounting information.
HASPNET	JXMTQPUT	23 Job is queued for HARDCOPY if there is any outstanding SYSOUT data set activity (i.e., JQEJOECT or JQEHLDCT are non-zero). Otherwise it is queued for purge.
HASPNET	JABTERM	25 Transmission error or job flushed by receiving node; call RTAM to perform "negative CLOSE."
HASPNET		26 Issue JOB RESTARTED or JOB DELETED message identifying job and line
HASPNET		27 Put JQE back on XMIT queue of job is to be restarted

JES2 HASPNET – SYSOUT Receiver Processor



1 $WAIT for work.

2 $GET unit. If available → 4

3 $EXTP NCLOSE (send NAK) → 1

4 $EXTP OPEN (send ACK)

5 Set "expecting job header" flag (NSR$JH) in PCE.

6 If job to be aborted → 29

7 $EXTP GET (get a record)
If abnormal end or
record not expected → 29

8 Check record type:
- If job header → 9
- If data set header → 16
- If data record → 21
- If job trailer → 23
- If EOF → 26

9 Get spool buffers for JCT and IOT.

10 Build JCT and allocation IOT.

11 Build JQE and add to $RECEIVE job queue ($QADD)

12 Assign JOBID

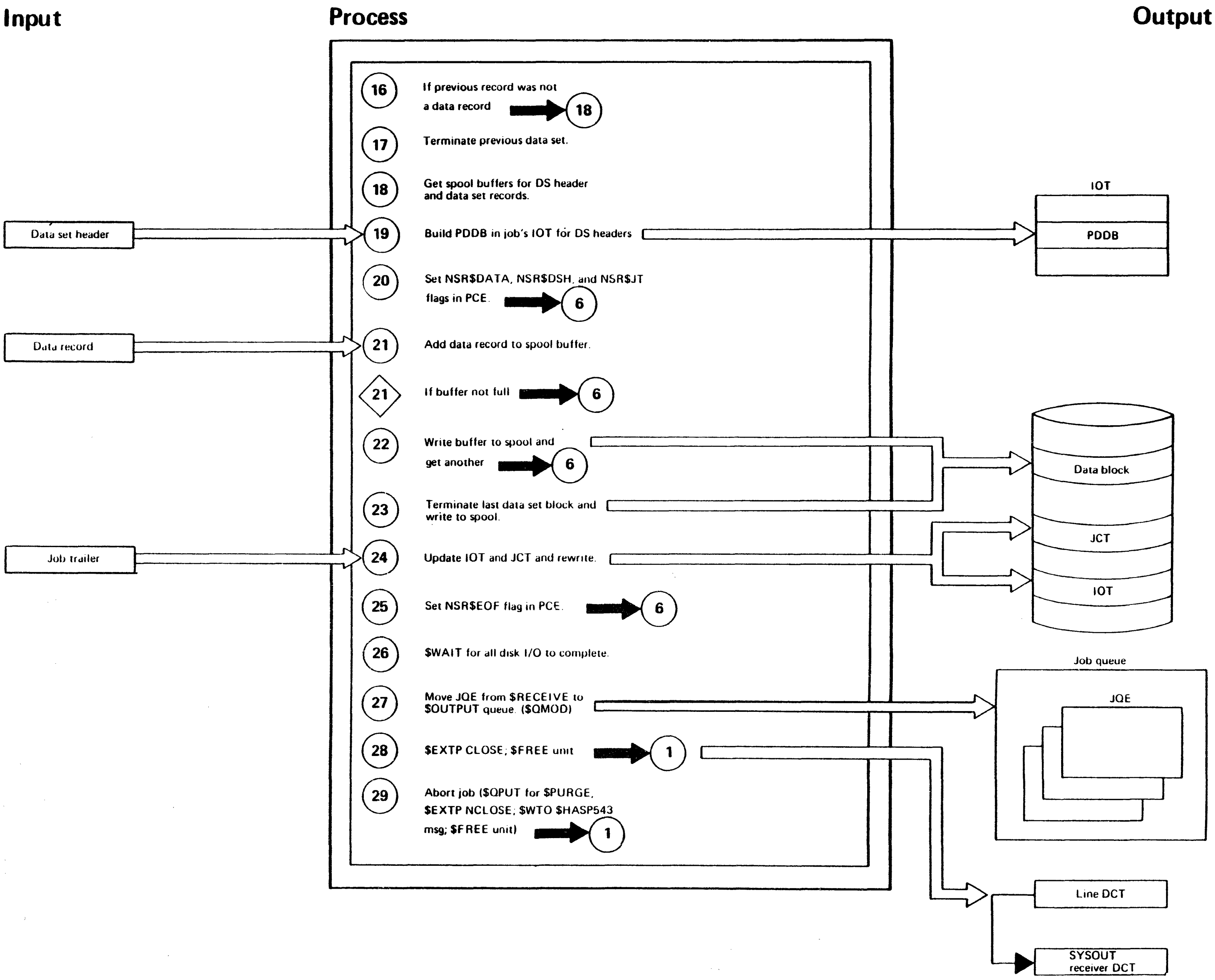
13 Write JCT and IOT to spool.

14 $WTO ($HASP540 message)

15 Set NSR$DSH + NSR$JT flags in PCE. → 6

ASSEMBLY LISTING NAME	ASSEMBLY LISTING LABEL	NARRATIVE
HASPNET	NSR$WAIT	1 $POSTed by HASPBSC or HASPSNA when SYSOUT transmitter on the other end of line requests permission to transmit.
HASPNET	NSR$UNIT	2 Will be unavailable if operator has drained the receiver.
HASPNET	NSR$NCL	3 Causes HASPBSC or HASPSNA to send negative acknowledgement to transmitter.
HASPNET	NSRSETH	4 Positive acknowledgement sent to transmitter.
HASPNET	NSR$GET	6 Check DCT and JQE flags to see if any operator commands ($E, $C, $P) have been issued against the receiver's device or job.
HASPNET	NSR$G	7 $EXTP GET macro is issued to get the next logical record. The returned record's sub-record control byte (SRCB) indicates the type of record received.
HASPNET	NSRJOBH	10 The JCT is built from information in the job header record.
HASPNET		12 Job Receiver attempts to assign same number originally assigned to job when it entered the NJE network.
HASPNET	NJQUPDC	13 Use $TRACK to obtain spool space for JCT and IOT
HASPNET		14 Issue message to identify job being received.
HASPNET		15 Indicate that the receiver is now expecting either a data set header or a job trailer.

JES2 HASPNET – SYSOUT Receiver Processor


Input
Process
Output
Data set header
Data record
Job trailer
16 If previous record was not a data record → 18
17 Terminate previous data set.
18 Get spool buffers for DS header and data set records.
19 Build PDDB in job's IOT for DS headers
20 Set NSR$DATA, NSR$DSH, and NSR$JT flags in PCE. → 6
21 Add data record to spool buffer.
21 If buffer not full → 6
22 Write buffer to spool and get another → 6
23 Terminate last data set block and write to spool.
24 Update IOT and JCT and rewrite.
25 Set NSR$EOF flag in PCE. → 6
26 $WAIT for all disk I/O to complete.
27 Move JQE from $RECEIVE to $OUTPUT queue. ($QMOD)
28 $EXTP CLOSE; $FREE unit → 1
29 Abort job ($QPUT for $PURGE, $EXTP NCLOSE; $WTO $HASP543 msg; $FREE unit) → 1
IOT
PDDB
Data block
JCT
IOT
Job queue
JQE
Line DCT
SYSOUT receiver DCT

ASSEMBLY LISTING NAME	ASSEMBLY LISTING LABEL	NARRATIVE
HASPNET	NSRLASTB	17 Write out the previous data set's last spool buffer with a zero chain field in HDBNXTRK and the EOB indicator set after the last record in the block.
HASPNET	NSRNIOT	19 When IOT full, obtain space for new IOT, chain to preceding one, and write out previous one.
HASPNET		20 Indicate that the receiver is now expecting data record, data set header, or job trailer.
HASPNET	NSRJOBT	24 The JCT is updated with information from the job trailer.
HASPNET	NSRCKFL	25 The receiver is now expecting EOF only.
HASPNET	NSRCLOSE	28 Use $EXTP CLOSE to send positive acknowledgement to job transmitter on other end of line.
HASPNET		29 Job is aborted by writing out its JCT and IOT, issuing $EXTP NCLOSE to indicate abnormal termination to the job transmitter, and issuing message to inform the operator.

Chapter 3. Program Organization

This section describes JES2 modules, their functions, basic methods of operation, and relationships to MVS and to other elements of JES2. The following table references the page on which each module description starts.

Figure 3-1 (Page 1 of 2). JES2 Modules

Module	Function Description	Page
HASPDOC	Describes the values and control blocks used throughout the executable JES2 modules.	3-3
HASPTABS	Contains various tables used by JES2.	3-3
HASPSTAB	Contains scan tables used by the $SCAN facility.	3-4
HASPMSG	Contains the scan tables and display routine for the JES2 message builder.	3-4
HASPINIT (load module)	Contains HASPIRA, HASPIRDA, HASPIRMA, HASPIRPL, and HASPIRRE, which initialize the subsystem and its interfaces.	3-4
HASPSCAN	Provides support for the $SCAN facility.	3-29
HASPSXIT	Contains all pre- and post-scan exits for the $SCAN facility.	3-33
HASPNUC	Provides central service subroutines to JES2.	3-36
HASPDYN	Dynamically builds PCEs, DCTs, and DTEs.	3-62
HASPSUBS	Initializes subtasks; removes and processes work requests from $STWORK; provides for generalized subtasking of calls to a specified routine.	3-65
HASPJOS	Provides job output services.	3-66
HASPTERM	Provides recovery and termination services.	3-77
HASPNPM	Manages network paths and provides network services.	3-86
HASPNET	Processes job and SYSOUT transmissions and SYSOUT receptions.	3-99
HASPSSSM (load module)	Provides the interface between JES2 and MVS; contains HASCDSAL, HASCDSOC, HASCJBST, HASCJBTR, HASCLINK, HASCSIRQ, HASCSRDS, HASCRIC, HASCSRJB, HASCDSS, and HASPAM.	3-116
HASPAM	Provides the access method for processing subsystem data sets.	3-130
HASPCON	Provides the console support services for JES2.	3-140
HASPTRAK	Provides track management and purge processing.	3-145
HASPCOMM	Receives and processes all JES2 commands from local and remote JES2 input sources.	3-150
HASPSERV	Provides command processing scanning services for use by HASPCOMM; provides work selection/modification services.	3-195
HASPSSRV	Provides various JES2 subtask services.	3-200
HASPRDR	Provides the functions of the input service processor.	3-205
HASPCNVT	Provides JCL conversion processing.	3-220
HASPCNVS	Provides a security environment for the jobs being converted and contains the entry to the converter subtask.	3-223
HASPXEQ	Provides execution services for jobs managed by JES2.	3-225

Figure 3-1 (Page 2 of 2). JES2 Modules

Module	**Function Description**	**Page**
HASPSTAC	Provides queue search and processing functions for the conversational job status and cancel requests.	3-230
HASPPSO	Process SYSOUT processor.	3-232
HASPPRPU	Determines the characteristics of a job's output data set and groups, performs the output operations, and loads the FCB and UCS images from SYS1.IMAGELIB.	3-237
HASPHOPE	Output processor.	3-257
HASPFSSP	Resides in the JES2 address space and provides the JES2 processor with routines to communicate with a functional subsystem.	3-260
HASPFSSM	Resides in the functional subsystem address space and supports the FSI functions used to interact between JES2 and a functional subsystem.	3-265
HASPRTAM	Provides major functions through which JES2 communicates with remote terminals and other nodes in a JES2 network.	3-277
HASPBSC	Provides remote terminal and network functions for BSC devices.	3-285
HASPSNA	Provides remote terminal and network functions for SNA devices.	3-303
HASPCKPT	Provides checkpoint processing services.	3-357
HASPCKDS	Provides checkpoint service routines and contains the checkpoint dialog routines.	3-378
HASPWARM	Provides warm start processing of the JES2 queues.	3-382
HASPMISC	Provides miscellaneous JES2 services: priority aging, SMF subtask support, and network accounting.	3-388
HASPEVTL	Provides for the tracing and logging of JES2 events.	3-392
HASPSPOL	Provides JES2 spool volume support.	3-399
HASPSTAM	Provides JES2 spool offload functions.	3-407
HASPRAS	Provides recovery routines, error, abend, and dump services.	3-416

HASPDOC: JES2 Control Areas Listing

HASPDOC is a non-executable module describing values and control blocks used throughout the executable JES2 modules. After assembly, HASPDOC contains the assembled DSECT of MVS and JES2 control blocks referenced during the assembly of JES2 source code. (Ordinarily, printing is suppressed at the point in each executable JES2 module where DSECTs are invoked; therefore, HASPDOC usually provides the only visual record of DSECT composition.)

In addition to depicting control blocks, HASPDOC includes equated values basic to JES2, such as absolute and symbolic register definitions, extended mnemonic equates, flag definitions, and default initialization limits.

The control block definitions contained in HASPDOC are available on microfiche (see the Preface of this manual for the microfiche order number). Since JES2 programming fixes distributed by IBM sometimes include updates to HASPDOC, a current listing of HASPDOC should be checked if there are questions about a value or field definition.

HASPTABS: JES2 Tables

HASPTABS contains JES2 tables that are used via the $GETABLE table-pair service or in conjunction with the $SCAN facility, table access routines, the master control table (MCT), the JES2 module map (MODMAP), and other JES2 tables.

The tables contained in HASPTABS are available on microfiche; see the Preface of this manual for the microfiche order number.

The HASP PCE table (HASPPCET) defines PCE names, descriptions, ids, lengths, and other processor characteristics. The PCE tables are accessed using the $GETABLE facility. Each entry in the table is created with the $PCETAB macro. The HASPPCER routine in this module is used to access the HASP PCE table. This routine is entered via the $GETABLE macro with TABLE=PCE specified.

The HASP event trace ID table (HASPTIDT) contains an entry for all HASP-defined trace ids. The tables are used during JES2 initialization and by the trace log processor (HASPEVTL) for formatting entries in the JES2 trace tables. The HASPTABR routine is used to access the trace id table. This routine is entered via the $GETABLE macro with TABLE = TID specified.

The initialization options table (HASPOPTT) is used to define the options specified on the OS parameter field or the $HASP426/$HASP427 WTOR replies. This table is accessed via the $SCAN facility.

The HASP parameter statement table (HASPMPST) contains an entry for each JES2 defined main parameter statement. Another table USERMPST, can be used to define user entries for main parameter statement processing. The MCT fields MCTMPSTH and MCTMPSTU point to these tables. The tables are accessed via the JES2 $SCAN facility.

HASPSTAB: $SCAN Tables

HASPSTAB contains tables that are used by the $SCAN facility. The $SCAN facility scans initialization statements, some commands, and parameters. For a list of initialization statements and the tables that contain them, see the HASPSTAB entry in Figure 4-1 on page 4-1.

HASPMSG: $SCAN Tables and Message Display Routines

This module contains entry points $MSGDISR and HASPMGST. $MSGDISR supports the message building function of $SCAN and is called from HASPSCAN. Its function is to construct the list forms (MF=L) and execute forms (MF=E) of the WTO, WTOR, or MLWTO macros. HASPMGST contains $SCANTAB entries for certain messages and their pre- and post-scan routines.

HASPINIT: JES2 Initialization Load Module

HASPINIT contains the source modules HASPIRA, HASPIRDA, HASPIRMA, HASPIRPL and HASPIRRE. HASPINIT initializes the subsystem and its interfaces and gives control to the JES2 $ASYNC processor.

HASP

The HASP routine is the first routine that gets control when JES2 is started. HASP loads HASPINIT, locates HASPINIT's MIT table (MITETBL), and passes control to HASPIRA. The HASP routine will use $MODLOAD to load HASPINIT if it is not link edited with HASJES20. HASP then locates MIT to pass control to HASPIRA. If JES2 initialization is successful, HASP deletes ($MODELETE) HASPINIT if it was loaded. If JES2 initialization is unsuccessful, $HASP428 is issued, meaning that a return code of 8 or 16 was returned by HASPINIT.

Finally, HASP exits to the ASYNCH processor; this allows control to eventually be given to the HASP warm start processor (HASPWARM), which concludes the preliminary initialization of JES2.

HASPIRA: Initialization Routine Administrator

The initialization routine administration is responsible for managing the initialization process. Upon initial entry, a work area is formatted like a processor control element. This work area is mapped by the $CIRWORK macro. Register 13 will point to the "initialization PCE" throughout the initialization process.

MODESET macro instructions are executed to ensure that JES2 is operating in supervisor state with the HASP protection key. The ESTAE is established for the JES2 main task.

At this point each initialization routine is called by the administrator. The initialization routines are called in the following order:

1. IRMODCHK - Checks that the SP release level of this JES2 is compatible with the release level of MVS installed on this system.
2. IROPTS - Processing of JES2 initialization options and the JES2 initialization exit point (for exit 0), producing initialized JES2 option flags.

3. IRSSI - Processing for JES2 SSI-related setup initialization, producing or validating a $HCCT and SSVT.
4. IRSETUP - Processing to allocate and initialize various temporary and permanent JES2 control blocks before initialization parameter processing.
5. IRPL - Processes the HASPPARM data sets.
6. IRPOSTPL - Post-parameter library processing.
7. IRDCTDCB - Allocates and formats most permanent DCTs.
8. IRDA - Allocates and initializes the checkpoint and spool DASD devices that JES2 uses during its processing.
9. IRCSA - Processing for JES2 control blocks - related allocations and initialization.
10. IRURDEV - Processing for JES2 unit record device initialization.
11. IRNJE - Performs NJE initialization.
12. IRRJE - Performs RJE initialization.
13. IRMVS - Processing for various JES2 functions that require using MVS system services and various MVS/JES2 interfaces.
14. IRDCTCP - completes all miscellaneous DCT initialization.
15. IRPCE - builds the non-dynamic PCEs.
16. IRFINAL - Processing to complete initialization before warm start processing.

On return from each initialization routine (IR) the return code (in register 15) is checked for an error. If an error occurs, HASPIRA returns to HASP in HASPNUC indicating an error situation.

If the return code from a given initialization routine is zero, HASPIRA continues looping and calling the next initialization routine until they all have been called. The HASPIRA takes the post initialization exit point (for exit 24) to allow installation exit routines to create their own installation-dependent control blocks or to alter JES2 control blocks. When the installation exit routine returns, HASPIRA continues initialization processing (return codes 0 or 4) or issues (via $WTO) the $HASP864 error message to the operator; then HASPIRA processes the error, returning to HASP in HASPNUC with an error indication.

If $TRACE and ID=6 are active, initialization options are traced. Initialization module map entries are zeroed only if HASPINIT was $MODLOADed, and HASPIRA returns to HASP in HASPNUC.

HASPIRA: Initialization Support Routines

The following routines are used by the various JES2 initialization routines throughout their processing. At IRERROR, HASPIRA processes any errors from initialization routines or from exit 24.

NBFBUILD: Buffer Pool Generation Routine

The NBFBUILD routine is used to generate buffer pools for JES2 data and control blocks and for general, print/punch, page, and teleprocessing buffers. A buffer pool occupies an integral number of pages. The first buffer in the pool contains a buffer pool map, defined by the $BPM macro expansion. This map contains a bit map, each bit of which relates to a buffer in the pool.

NSSSM: HASPSSSM Module Location Routine

NSSSM issues $MODLOAD to load and $MODCHK to validate HASPSSSM. If the load is unsuccessful, NSSSM sets R1 to point to diagnostic message, $HASP875.

If the $MODLOAD is successful, NSSSM initializes the MODMAP entries for the module, issues the DELETE macro to delete the CDE associated with the module and returns to the caller.

NQUERY: Operator WTOR Routine

NQUERY receives a zero or one in R1. Zero means that NQUERY issues the $HASP441 message to query the operator to see if initialization is to continue or termination is to continue. One in R1 means that NQUERY issues the $HASP872 message to query the operator to see if initialization is to continue, termination is to continue, or retry is to take place.

NRDCTINT: Remote Device DCT Initialization Routine

NRDCTINIT initializes remote device DCTs. On input, R0 contains the RWL entry and R1 contains remote device DCT address. NRDCTINIT uses the RWL to initialize the DCT with information that includes the device id, standard reader LRECL, execution class, message class, and work selection definitions. NRDCTINIT then returns to the caller.

HASPIRMA: HASP Miscellaneous Allocations and Initialization

HASPIRMA performs miscellaneous allocations and initialization on behalf of JES2. It:

- Checks the compatibility between the JES2 SP release level and the release level of MVS on this system
- Processes JES2 initialization options
- Verifies and/or allocates the SSVT and SSCT
- Processes PARMLIB setup
- Allocates and initializes various JES2-specific tables (such as the PIT, RAT, NIT, XIT, LMT, and the CPT)
- Builds the PCE and DCT
- Initializes the SWB $KEYLIST table
- Establishes the cross memory authorization index (AX)
- Attaches the image loader, allocation, and VTAM subtasks
- Completes the event trace CSA control blocks
- Processes the SMF subtask, exits, and subsystem initialization record (SMF type 43)
- Creates the console message buffers (CMBs)

IRMODCHK: Verify Compatibility

IRMODCHK ensures that the SP release level of this JES2 is compatible with the release level of MVS installed on this system. The system 370 version of JES2 is compatible with both the system 370 and the extended architecture versions of MVS, but the extended architecture version of JES2 is compatible only with the extended architecture of MVS. IRMODCHK also ensures that each JES2 load module and user exit routine have been assembled at the same level as the JES2 nucleus.

IROPTS: Process JES2 Options Routine

IROPTS scans the options specified on the EXEC card in the "Start JES2" procedure and uses $HASP426 (via WTOR) to request options from the operator if none are specified. IROPTS then prepares to establish exit 0. A temporary XIT is established in the CIR PCE work area and a temporary maximum length XRT is obtained for use by the $EXIT macro and exit effector. IROPTS attempts to load HASPXIT0. It issues the $MODLOAD macro to load and $MODCHK to verify the module. If the load fails, exit 0 is marked as disabled. If the load is successful, IROPTS loops through the MIT entry table (MITETBL) and builds XRT entries for the exit routines in HASPXIT).

After exit 0 and its routines are established, IROPTS copies the exec card parameters field to the CIR PCE work area. IROPTS then takes exit point NEXIT0 (for exit 0) passing control to the exit effector. The exit routines can return with a return code that requests that normal processing continue, the IROPTS processing is to end without scanning the JES2 options (assume options flags are set), that JES2 initialization is to terminate.

If normal IROPTS processing is to continue, IROPTS scans the JES2 options via the $SCAN macro. If the scan fails, NQUERY is called to request initialization parameters from the operator. If the scan is successful, the setup for exit is removed. IROPTS checks to see if a $PJES2 was issued. If it was IROPTS immediately returns to the system via SVC 3; otherwise XIT/XRT storage is freed and return is made to the IR administrator with an indication that IROPTS was successful.

IRSSI: Verify and/or Allocate the SSVT and SSCT Routine

IRSSI locates the subsystem CVT (SSCT). If it cannot be found, initialization is terminated; otherwise, the SSCT is used to locate the corresponding SSVT and $HCCT. The $HCCT is analyzed to see if JES2 can be initialized. If JES2 has terminated already, is already up and active, or is restarting with conflicting options, $HASP425 is issued.

If JES2 is not restarting and a hot start is not allowed, IRSSI issues $HASP490; if the hot start is allowed, but JES2 cannot be started yet, IRSSI issues $HASP425. Otherwise, JES2 can be initialized. IRSSI calls NSSSM to load HASPSSSM. If the subsystem has not yet been initialized, IRSSI obtains fixed global storage for the JES2 wait ECB, the MVS SSVT, and the $HCCT, and returns to the IR administrator.

IRSETUP: PARMLIB Setup Processing Routine

IRSETUP page fixes the HCT and other HASPNUC routines, obtains and initializes various fixed-length tables (such as, the priority table, the ACT, and the CAT). IRSETUP extracts the JES2 security token through a RACROUTE request. IRSETUP obtains and initializes storage for default remote and networking DCTs, initializes the temporary event trace id table, and obtains storage and initializes the PIT, RAT, CPT, NIT, and APW. Finally, IRSETUP obtains XITs and LMTs and initializes them, zeroes all exit routine addresses and the XRT (if this is a hot start), propagates the defined exit parts within HASJES20 to the XIT, allocates an LMT and places an entry in it for HASPXITO, allocates and initializes the PRMODE table and allocates and initializes the functional subsystem control blocks (FSSCBs), and returns to the IR administrator.

After IRSETUP, HASPIRMA obtains subpool 0 storage for fixed length permanent tables above the 16 megabyte address, the checkpoint work area (CKW) and the HASP file allocation map (HFAM). The tables are initialized.

IRDCTDCB: DCT Formatting Routine

This routine allocates and formats most DCTs. First it sets the default maximum CCW counts if they are not provided. It computes the variable DCT storage, and then obtains it. It sets the RTAM work area address. Then it formats the DCTs. The routine computes ACE and SMF buffers storage and constructs the automatic command element chain. Finally, it initializes the SMF buffer pool, gets storage for a direct access DCB and a maximum size DEB, and initializes the direct access DEB.

IRCSA: CSA Control Block Initialization

IRCSA completes the initialization of various CSA control blocks. First, the console message buffers are obtained out of subpool 231 and initialized. Parameter library commands are then queued for later command processing.

As part of IRSCA processing, the data space blocks for the internal reader protected buffers are established. On a hot start, JES2 first obtains access to the data space that contains the internal reader protected buffers.

The following routines are used to initialize the $HCCT:

IBLDHCCT

The IBLDHCCT routine initializes the JES2 $HCCT. If JES2 is being restarted, the $HCCT already exists and minimum initialization is required. The previous $HCCT is tested, however, to ensure that the options in effect during the previous JES2 initialization are compatible with the current options. If not, the $HASP430 error message is issued and JES2 is quiesced.

If JES2 is not being restarted, IBLDHCCT fills in various entries in the $HCCT from JES2 parameter defaults, initialization parameters, and system tables.

ISVTINIT

Unless JES2 is restarting, the final phase of initialization begins with the copying of the direct access DCBs and data extent block (DEB) into the $HCCT. The direct access storage control blocks (DAS) are copied into subpool 241 storage and their pointers in the $HCCT are initialized. The TGBE entries are initialized. Then $HCCT pointers are initialized to point to the first TGBE, the last TGBE to be used during checkpoint processing, and the first TGBE that routines $STRAK and $TRACK will use to begin their search for a TGB. Various other control parameters are moved to the $HCCT.

ICSAPITS

Storage for the partition information (PIT) tables is obtained and the PITs are initialized.

NTRC

NTRC determines valid trace ids and updates the $HCCT to reflect them.

IRURDEV: Unit-Record Device Allocation

IRURDEV locates and allocates eligible and specified unit-record devices and remote job entry lines. First, it locates the system attention table and stores the address of the JES2 attention appendage in the appropriate entry in this table.

IRURDEV then searches the unit control blocks (UCBs) using the $GETUCBS routine and identifies readers, printers, punches, and physical remote job entry lines. When a unit has been identified, the status of the unit is determined from flags in the UCB and from the response to an EXCP of a NOP channel command word (CCW).

IRURDEV searches the device control tables (DCTs) for a specific device request. If a parameter library statement has designated a specific unit address for a particular device, that unit address is in the DCTBUFAD field for that device's DCT. If a match is found between the DCTBUFAD field and the UCBNAME field, that device is allocated as described below and the DCT search continues for the other DCTs that may specify this unit. (Note that in DCTs representing logical lines, DCTBUFAD is set to ones by the DCT initialization routine. This ensures that no match between a UCBNAME field and a logical line DCT, and therefore no device allocation, can occur.) If the unit has been designated by at least one device, or if the device status shows that the device is not physically attached to the system, then when the end of the DCT chain is encountered, control is returned to examine the next UCB.

Otherwise, IRURDEV searches the DCTs again for an unspecified device of the type (reader, printer, punch, or line) that corresponds with the device type of the current UCB. If no available DCT is found and if the condition has not already occurred on a device of this type, the $HASP412 message is issued to the operator to indicate that at least one available device is not being assigned, and processing continues. If an available DCT is found, it is allocated as described below, and control is returned to examine the next UCB.

To allocate a device, IRURDEV stores the address of the UCB into the DCT (DCTUCB) to be allocated. If the device is an RJE line, the DCT is drained and certain fields are modified according to the adapter type. For other devices, the status of the DCT is examined (DCTSTAT) and if the device has been drained (using the parameter library), no further action is taken. A count of the number of IBM 3800 Printing Subsystems allocated by JES2 is kept for later use in buffer pool generation. For impact printers, if forms control buffer (FCB) and universal character set (UCS) were not specified on the PRTnnnn statement, the default values of FCB and/or UCS on the PRINTDEF statement are stored in the DCT. For non-impact printers (the IBM 3800), the default values stored are NIFCB and NIUCS on the PRINTDEF statement; the NIFLASH parameter of the PRINTDEF statement contains the default values for flash.

Note: The DEVFCB= parameter can be coded on the PRTnnnn and Rm.PRn initialization statements to indicate the device default FCB (DDFCB), which is the sole FCB value to be used in the event that the output element has not explicitly declared FCB= on the JCL. If DEVFCB= is not coded, or is set to null using the $T command, the default value is stored in the DCT as described above.

In addition, the content of the DCTFCB field, once initialized, are moved to field PRFCB in the print/punch processor control element (PCE) work area extension. If the device has not been drained, the status of the device itself is examined, and, if it is not ready, it is forced into a drain status. Otherwise, an attempt is made to allocate the device to JES2 using dynamic allocation. If the allocation is not successful, the device is drained.

Finally, IRURDEV scans all DCTs once more and drains all DCTs that have not had units assigned to them. Control is then returned to the IR administrator.

IRMVS: MVS Service Initialization

IRMVS identifies the JES2 POST exit routines to the operating system via branch entry post, using entry point IEAOPTOE. The post exits are in effect while the JES2 address space exists.

Next, IRMVS initializes the scheduler work block (SWB) $KEYLIST table in subpool 241. A check is first made to determine if the scheduler JCL facility (SJF) is initialized. If the SJF is initialized the SWB text unit keys are obtained via a SJFREQ REQUEST=JDTEXTRACT. If an SJF extract error occurs IRMVS issues message $HASP855, "JCL PROCESSING LIMITED DUE TO ERROR IN SYSTEM JOT". The operator is then queried as to whether JES2 initialization should continue. If the operator replies no (N), then JES2 initialization finishes and a $SDUMP a $SDUMP macro is issued. If the operator replies yes (Y), then all future SJF services are bypassed, the $KEYLIST table is freed and JES2 initialization continues. IRMVS issues SJFREQ REQUEST=TERMINATE to free SJF local storage and cancel the scheduler ESTAE.

The $KEYLIST table is used to hold the SWB text unit keys for the following characteristics; burst, destination, copies, class, chars, FCB, flash, forms, modify, UCS, writer name, and LINECT. This table is used later to build SWBs that can be sent to the functional subsystem. The functional subsystem application (FSA) uses the SWBs to print the SYSOUT data sets with the correct output characteristics.

IRMVS then initializes the authorization index for the JES2 address space and for the functional subsystem address spaces; these authorization indexes allow JES2 to set secondary address space (SSAR) to the functional subsystem address spaces. IRMVS then invokes the MVS assignable device initialization service specifying that all assignable devices should be processed. The command processor extended area is initialized. Then, SMF is initialized.

Any JES2 module that writes SMF records invokes $SMF. $SMF maps record types 52-58. $SMF calls IAFSMFR to map all other JES2 SMF records (types 6, 24, 26, 43, 45, 47, 48, 49).

The NWRAPUP routine obtains storage from the JES2 region and initializes the command processor's extended area. The location of the multiple console support (MCS) response target area verification module, IEE7603D, is determined through a LOAD macro instruction (with subsequent DELETE macro instruction), and its address is stored in this area. The write-to-operator subtask ($HASPWTO) is then identified, attached, and verified.

A loop, initiated with the $GETABLE macro, through all the DTETAB entries attaches HASPACCT (the SMF subtask), HASPIMAG (the image loader subtask), and HASPVTAM (the VTAM subtask) if they are defined to be automatically started, that is, specified as GEN=YES on the $DTETAB macro.

SMF processing is initialized with a LOAD macro instruction that finds the location of the IEFUSO and IEFUJP exits. If they are found, IRMVS replaces the default values in the $HCCT with the addresses of these routines. The first SMF record (type 43) is then generated using the standard SMF facilities in HASPNUC. If an application copy of the checkpoint is to be used, the HASPCKAP subtask is attached.

Next, IRMVS attaches the requested number of general purpose subtasks indicated on the GSUBNUM parameter of the SUBTDEF initialization statement. If all (or any) of the attaches fail, the $HASP475 message is issued to display the number of available subtasks. If no subtasks are available, required JES2 functions are performed with severe performance degradation.

If some attaches fail, but at least one subtask was successfully attached, some performance degradation may be apparent. (The $HASP872 message is issued for each attach failure.) The operator will then be prompted by the $HASP872 message to continue, terminate, or retry. If processing is to continue, IRMVS then exits to the IR administrator.

IRDCTCP: Miscellaneous DCT Initialization

This routine completes all miscellaneous DCT initialization. When this routine completes, the JES2 DCTs are complete and chained together. It does this by validating and initializing modify fields in the DCT for system affinity and route code for offload job receivers and modify fields for route code for SYSOUT receivers. It also verifies that remotes number less than $MAXROUT or is equal to X'FFFF'. It validates system affinity values specified on the OFFn.JT or OFFn.JR initialization parameters.

On a hot start, IRDCTCP re-initializes internal reader DCTs and reestablishes access to the internal reader protected buffer data space. If JES2 is cold or warm starting, this routine builds new internal reader DCTs in CSA and creates the internal reader protected buffer data space.

The routine then creates request-jobid DCTs (for the execution and trace log processors, among others) and properly connects the DCTs to FSACBs (hot start) or FSIDs and finalizes the FSSCBs in CSA.

IRPCE: PCE Generation

This routine generates the PCEs required for JES2 to execute and the PCEs for those devices that need to start immediately or cannot be added later. The routine builds the "primary" (non-device) PCEs required. The dynamic PCE service will mark the new PCEs as $WAITing for work or on hold; it will ready them immediately as required. The routine then builds the device-related PCEs. It will mark these PCEs the same way as the non-device related PCEs. The routine at this point also connects the DCTs and PCEs. It uses the $PCEDYN DCT-chain service routine, which allocates PCEs for DCTs that require them, connects DCTs to existing PCEs, or bypass DCTs that do not have related PCEs, or for which PCEs are later allocated. The routine also initializes the network path manager.

IRFINAL: Complete All Miscellaneous Initialization

IRFINAL initializes the internal reader device control tables (DCTs). The internal reader DCT address in the $HCCT is tested and if there are no DCTs from a previous start of JES2, storage is obtained from subpool 241 in the common storage area. The DCTs are constructed in this area from a temporary internal reader DCT. One DCT is built for each internal reader specified by the RDINUM parameter plus two for started task control (STC) and time-sharing user (TSU) processing. If the DCTs already exist, no new DCTs are built, but the in-use flag (DCTINUSE) is reset in each of the existing internal reader DCTs.

The appropriate processor control element (PCE) address is moved into the DCTPCE field of each DCT. The remote terminal DCTs, if any, are removed from the chain of DCTs to which $DCTPOOL points. The JES2 processor $WAIT queues are initialized, and the started task control (STC) and time-sharing user (TSU) task class attribute table (CAT) entries are updated.

Finally, all PCEs except for those for the asynchronous I/O, warm start, console, checkpoint, timer, and network job routing receiver and transmitter are placed in $WAIT state, and control is passed to the next section.

Buffer Pool Generation for Standard JES2 Buffers Routine

The default value buffer count is calculated if no value for BUFDEF has been provided. The value is calculated to limit the possibility of buffer lockout in a situation where the maximum number of devices is active. NBUFBLD calculates the value as the sum of the following:

20	For various concurrent subsystem processes
N1*N2*TGSIZE	The number of track-cell-despooling (TRKCELL=YES) local printers times their buffering option times the track cell size

where:

N1 is the number of (local) printers defined with the TRKCELL=YES keyword.

N2 is 2 if DBLBUFR(on PRINTDEF statement)=YES, 1 if NO
N3 is 2 if DBLBUFR(on PUNCHDEF statement)=YES, 1 if NO
N4 is 2 if RDBLBUFR(on PRINTDEF statement)=YES, 1 if NO
N5 is 2 if RDBLBUFR(on PUNCHDEF statement)=YES, 1 if NO

The upper limit on BUFNUM is 2000; if the calculated value exceeds this, 2000 is used.

Next, the buffer pool generation routine is invoked to construct the buffer pool in subpool 0 storage obtained with the $GETMAIN macro, and the returned address of the JES2 general buffer pool map is stored in the HASP communications table (HCT) at $BFRMAP. The actual number of buffers generated is stored in $NUMBUF; this may be greater than the requested number because the MVS GETMAIN service routine provides an integral number of storage pages and JES2 creates buffers to fill all of the pages obtained.

NBBLDPP: Buffer Pool Generation for Print/Punch Buffer Routine

JES2 printer/punch processor buffers are used in channel program creation for three processes: local printer output, local punch output, and input of track cells from a spool device.

For output operations, each print/punch buffer must be large enough to contain a JES2 input/output buffer (IOB), two channel command word (CCW) areas, a program-controlled interruption element (PCIE), and a checkpoint job output element (JOE). Each of the CCW areas must be large enough to contain the number of CCWs specified for the CCWNUM parameters on the PRINTDEF and PUNCHDEF statements.

For input operations, the print/punch buffer must be large enough to contain a JES2 IOB, a 9-byte entry for each record in a track cell, a set sector CCW, and the CCWs required to provide a search/TIC/read sequence for each record in a track cell.

Therefore, the print/punch buffer size is calculated as the size of a JES2 IOB plus 2*M, where M is the greatest of:

```
CCWNUM*8 + PCIESIZE + ((JOESIZE + 7)/8)*8
```

or

```
((TGSIZE*4) - 3)*4
```

The number of print/punch buffers required equals the number of local punches, plus the number of local printers defined with the the TRKCELL=YES keyword, plus the number of impact printers.

The NBBLDPP routine invokes the buffer pool generation subroutine (NBFBUILD) to construct the print/punch buffer pool in subpool 0. The address of the print/punch buffer pool map is then stored in the HCT at $PPBFMAP.

NBBLDPG: Buffer Pool Generation for JES2 Page Buffers Routine

A JES2 page buffer equals an MVS page (4096 bytes). Page buffers are used to construct output channel programs for non-impact printers, such as the IBM 3800 Printing Subsystem. Page buffers are broken into areas as described above for print/punch buffers. The number of buffers required equals the number of non-impact printers defined at initialization.

If at least one non-impact printer has been defined, this routine calls the buffer pool generation subroutine (NBFBUILD) to construct the page buffer pool in subpool 0. On return, the address of the page buffer pool map is stored in the HCT at $PGBFMAP.

End of Basic Initialization

The subsystem vector table (SSVT) and $HCCT are connected to the subsystem communication vector table (SSCT). Control is returned to the IR administrator.

HASPIRPL: JES2 Parameter Library Initialization

The label IRPL in HASPIRPL processes the SYS1.PARMLIB (HASPPARM) data set that contains the JES2 initialization statements and initializes JES2 in accordance with these initialization statements. IRPOSTPL in HASPIRPL post-processes the parameters that were initialized by HASPIRPL.

IRPL begins parameter processing by first opening the SYS1.PARMLIB (HASPPARM) data set using the DDNAME in the DCB pointed to by CIRHPDCB field of the CIR PCE work area. IRPL issues an OPEN macro instruction for this DDNAME. If the open is unsuccessful, IRPL issues a $$WTO for the $HASP450 message and calls the NQUERY routine to request operator response. If the error can't be bypassed, IRPL issues a CLOSE macro for the DDNAME, dynamically deallocates the SYS1.PARMLIB (HASPPARM) data set, frees any existing buffer pools, and returns to the caller.

If IRPL successfully opens SYS1.PARMLIB (HASPPARM) using the supplied DDNAME, initialization processing continues at label NPLSTART. At NPLSTART, IRPL issues a GETMAIN macro for 4K of storage that is to be used as the parameter statement work area. Then IRPL begins a loop at NPLLOOP to process each initialization statement. IRPL invokes NPLGET to obtain an initialization statement. If an error occurs in trying to obtain an initialization statement, IRPL propagates the return code to the IR administrator after first performing cleanup processing similar to that it performs for a bad open of the SYS1.PARMLIB (HASPPARM) data set. If the initialization statement is successfully obtained, exit point NPLEXIT (for exit 19) is taken to allow an installation exit routine to delete the initialization statement, change it, or terminate JES2 initialization. When the installation exit routine returns, IRPL issues a $STMTLOG macro instruction to log the initialization statement as changed or modified by the exit routine. If an initialization statement was inserted by the exit routine, IRPL issues another $STMTLOG macro instruction to log this inserted statement. The exit routine itself may issue $STMTLOG calls, placing commands, warnings, or diagnostics into the log with, if desired, their own message IDs.

Normal initialization processing continues if the return code from the exit routine is 0 or 4. If the return code is 8, IRPL bypasses processing this initialization statement and gets the next one at NPLLOOP. If the return code is 12, HASPIRPL issues a $STMTLOG macro instruction to log "TERMINATION REQUESTED BY THE USER EXIT", and issues a $$WTO message to the operator telling of the exit routine's action. Then IRPL cleans up resources and returns to the caller.

Processing of the initialization statement continues at NPLSCAN. Here IRPL uses the scanning facility ($SCAN) to scan the various parameters in the statement, initializing the scan work area with the information contained in the statement. If the initialization statement is a "display" statement, IRPL issues a $SCAN macro instruction to display the parameters on the statement. If this statement was not processed by the scanning facility, IRPL then processes the initialization statement if it has a PTENT entry, issues a warning message if needed, logs the statement, and returns to the caller.

Processing of initialization statements continues in this way, in a loop, until all initialization statements have been processed (that is, an end-of-file is reached on the SYS1.PARMLIB(HASPPARM) data set).

NPLCLOSE: Parameter Library Processing Termination Subroutine

NPLEND receives control when the end-of-data is detected on the parameter library input file. The list file is closed with the FREE option to cause deallocation of the devices or data sets. The input file is closed and the DYNALLOC macro instruction causes deallocation of the input devices or data sets. A FREEPOOL macro instruction is issued to free the buffers associated with parameter library processing. The CIRF1FER flag is then tested to determine whether an error has been detected on the parameter library. If so, the $HASP451 error message is issued, and the $HASP441 message is issued, allowing the operator to decide whether to allow initialization to proceed. If the reply is N, JES2 issues message $HASP428 and is terminated. Otherwise, control is returned to the IR administrator successfully.

IRPL Support Routines

IRPL provides the following routines to support the processing of initialization statements.

NPLDCBOE: HASPPARM/HASPLIST Exit Routine

When an end-of-file occurs while reading the SYS1.PARMLIB(HASPPARM) data set or when the open fails for this data set, NPLDCBOE gets control. For an open failure, NPLDCBOE sets the block size and the unlike data set concatenation in the DCB and returns to data management. For an end-of-file on SYS1.PARMLIB(HASPPARM), NPLDCBOE indicates an end-of-file in the initialization PCE work area and returns to data management.

NPLDISP: Display Subroutine

NPLDISP displays the WTO message (either through a DISPLAY id or a diagnostic message) that $SCAN has set up (via $$WTO) and returns to the caller. If this routine is called to issue a diagnostic (based on the return code passed by $SCAN), it checks to see if the keyword in error is a PTENT entry. If not, this routine issued the diagnostic message and returns to $SCAN. If the keyword is in a PTENT entry, the PTENT subroutine is called to process it, **not** issue the diagnostic. Then NPLDISP returns control to $SCAN.

NPLGET: Initialization Statement Get Routine

NPLGET initializes the CIR PCE work area with the address and length of the next initialization statement; the next statement is received from SYS1.PARMLIB(HASPPARM) or from the operator console. This routine will piece together the initialization statements (dependent on "," and comments).

NGETCON: DCB and Output Work Area Setup Subroutine

NGETCON loops, invoking NGETCARD for an input card and determining the starting address and length of the input. NGETCON accumulates, in the loop, a variable length parameter statement.

NGETCARD: Get Input Card Subroutine

NGETCARD issues a GET macro instruction to obtain an initialization statement from SYS1.PARMLIB(HASPPARM) and returns to the caller if the read is successful.

NGETLEN: Determine Input Length Subroutine

NGETLEN determines the input length of the parameter part of an initialization statement. The actual length is returned to the caller in R2.

NGETCNT: Determine Continuation Subroutine

NGETCNT determines if the input is continued.

NPLLOG: Log/List/WTO Diagnostic Subroutine

For each initialization statement, NPLLOG is called to either log the statement, list the statement to hard copy, or issue a diagnostic message (if the statement is in error).

A $$WTO macro instruction issues message HASP467, which displays the entire initialization statement that is in error. Diagnostic messages are logged to the console and listed in the HASPLIST data set.

NLOGHLST: Write HASPLIST Subroutine

If operator options indicate that the HASPLIST data set should be written, NLOGHLST is called. NLOGHLST issues the PUT macro instruction to write HASPLIST records that are printed later.

REP Facility Processing Subroutine

These routines process the BASE, the VERIFY (VER), the REPLACE (REP), the NAME, and the ENDZAP statements.

NPLCOMND: Command Processing Subroutine

The NPLCOMND routine records statements beginning with a dollar sign ($) (other than $$x statements). No processing is performed at this time. The command is saved in an area obtained via a $GETMAIN and placed at the end of the chain of commands pointed to by NCOMMTAB.

IRPOSTPL: Parameter Post Processing Routine

IRPOSTPL checks for a valid HASPSSSM name in LPA. If it is found, initialization continues; otherwise, $HASP869 is issued, a return code of 8 is set in R15, and control is returned to the caller.

IRPOSTPL continues initialization by supplying class conversion defaults, where needed, to the initialization parameters. IRPOSTPL initializes the class attribute table entries for started task control, logon TSU, and batch jobs.

IRPOSTPL then performs the following initialization:

NSETJWEL: The JOE/WRITER exclude table and the address space vector table are obtained and initialized. These tables are used to indicate when a device or address space access of a JOE has failed authority verification. This eliminates the performance overhead of additional RACROUTE calls each time the address space or device attempts to select a JOE.

NCKBSP: The console backspace character in $BSPACE is compared to the console command character in $CCOMCHR. If equal, $BSPACE is set to X'00', and the warning message $HASP448 is issued.

NBSPOK: The value of $OWNNODE is compared with NODENUM on the NJEDEF statement to ensure that this system node falls within the range of valid node numbers. If the test fails, the $HASP437 message is issued, and JES2 terminates.

NSMFTEST: If fewer than two SMF buffers are requested, the SMF buffer count is set to 0, and instructions are modified in the SMF buffer get and queue routines ($GETSMFB and $QUESMFB) in HASPNUC so that those subroutines return control to the caller immediately upon being entered.

NINRTEST: If JES2 is restarting, the number of internal readers indicated may require adjustment. The subsystem vector table (SSVT) is examined; if no internal readers exist, further testing is bypassed. If an internal reader device control table (DCT) exists but inspection shows that it is invalid, its pointer, SVTIRDRS in SSVT is set to 0, and no further testing is performed. If SVTIRDRS points to a valid internal reader DCT and a cold start is in progress, no change occurs; however, if a JES2 warm start is in progress, the number of internal readers ($NUMINRS) is reduced by two to account for an internal reader for started tasks and by one for time-sharing option (TSO) sessions.

NCMCTEST: This routine begins initialization for remote job entry. The maximum console message count $MAXCMCT (unless already set as a result of the RMTMSG parameter on a TPDEF initialization statement in the parameter library) is set to the value of CMBNUM on the CONDEF statement or 255, whichever is less.

NRJETEST: $NUMRJE and $NUMTPBF are examined. $NUMRJE is set to the highest RMT(nnnn) value found in the temporary remote attribute table (RAT). If $NUMLNES is zero, $NUMRJE is set to zero. $NUMTPBF is set to twice $NUMLNES, if $NUMTPBF is zero. NRJETEST then determines the number of lines specified as logical lines by scanning the chain of all line DCTs and increasing a counter for each table representing a logical line; the counter value is stored in NLOGLINE.

NPLXFINL: This routine performs final exit facility initialization. If JES2 is not being hot started, this routine computes the size of the final combined exit routines table (XRT) and then obtains the required storage from subpool 241. If a JES2 hot start is in progress, the XRT is not changed. This routine scans the exit information table (XIT) for each entry that has a temporary XRT associated with the user exit routine. NPLXFINL ensures that:

- The exit point is defined. If it is not defined message $HASP857 is issued.
- All specified exit routines were found. If not, message $HASP858 is issued.

If any errors have been detected when the scan of the XIT is completed, message $HASP441 is issued. This allows the operator to terminate or continue initialization. Finally, the LMT is obtained and all temporary storage is freed.

A call to routine TRGETTB initializes the trace tables if JES is not hot starting. TRGETTB is located in HASPEVTL. After completing this processing IRPOSTPL sets a return code and returns to the IR administrator.

HASPIRDA: DASD (Checkpoint/Spool) Initialization

IRDA in HASPIRDA allocates and initializes the checkpoint and spool DASD devices that JES2 uses during its processing. HASPIRDA processing includes the following functions:

- check and set the IOTPDDB offset by computing the number of track group allocation entries (TGAEs)
- set print/punch-related parameter defaults
- calculate the number of checkpoint pages
- acquire storage for the master checkpoint record and its I/O area
- clear and page-fix the master checkpoint record
- acquire and clear checkpoint pages
- fill in the KIT
- acquire a checkpoint buffer table
- locate spool volumes
- update spool volume allocation table
- if warm-starting, read in the checkpoint
- process forwarded data sets

- process changes to SID(s) and QSE(s)
- locate and flag the last-defined QSE.

The following routines support HASPIRDA processing.

Set IOTPDDB: Master Checkpoint Record Allocate/Format Routine

The size of the track group map is calculated based on $NUMTG and stored at $CYLMAPL in the HCT.

The size and number of fields in the IOT are then calculated. The IOT will contain the fixed area, at least 50 TGAEs, and as many PDDBs as will fit. The exact number of TGAEs is determined by first fitting as many PDDBs as possible with 50 TGAEs and then filling in any extra space with TGAEs. The number of TGAEs is stored at $TGAENUM in the HCT and the total size of the TGAE area in the IOT is stored at $TGAELEN in the HCT. The offset in the IOT of the first PDDB is stored at $IOTPDDB in the HCT.

The length of the spool volume prefix ($SPOOL in the HCT) is determined. The value is stored as length-1 (for execute instructions) at $SPLLEN in the HCT.

Create Volume Allocation Table Routine

The temporary volume allocation table (CIRVOLTB) is obtained during IRDA to contain the entries that represent each spool volume. The UCBs for all DASD volumes are scanned (using the $GETUCBS macro) for any volumes with the spool volume prefix. When a match is found, a check is made to ensure that there is only one UCB for the volume. If there is more than one, IRDA issues the $HASP422 message and exits with an error. Once it is known there is only one UCB, an entry is made in the volume allocation table for the volume using the data from the UCB.

Build the KITs from the CTENTs

After updating the spool volume allocation table, routine KBLDKIT is called to build temporary checkpoint information tables (KITs) based on the checkpoint information entries (CTENTs). There are NCTETENT number of CTENT entries in the checkpoint data set. KBLDKIT $GETMAINs NCTETENT number of KITs and fills each KIT with information obtained from the HCT and the CTENT for that table.

Build and Initialize Checkpoint Control Blocks

IRDA determines the number of checkpoint pages by looking at each KIT and adding up the size of each table. If the size of the change log was not specified, a default size is determined.

The size of the master record is determined and stored in the HCT ($MASTERL). Storage for the master checkpoint record and its I/O area is obtained by a $GETMAIN from subpool 0. The address of the master record area and its I/O area are stored in $MASTER and $MASTERI in the HCT. Pointers to the significant data areas in the master record and its I/O area are saved in the HCT. The temporary KITS are copied into the master record and the temporary KITs are $FREMAINed.

KBLDCB is called to build the remainder of the checkpoint control blocks. They are 4-K checkpoint pages (both in-storage and I/O copies), an extra copy of the control bytes (for the HASPCKAP subtask), a fixed list byte table for the 4-K pages (each byte in this table is used to determine if a 4-K page in the checkpoint is fixed or not), the initial CAL (used by $CKPT), and a page service list (PSL). One PSL entry is needed for each 4-K record to be serviced. The 4-K checkpoint entries and the 4-K change log entries both use the PSL, but not at the same time. Storage is calculated based on larger of the two 4-K record numbers. The routine then $GETMAINs a temporary track one table (TOT) and initializes the track one records (TOR). Finally, KBLDCB $GETMAINs the checkpoint trace work area (CTW), and returns to IRDA.

Verify Initialization Options

If RECONFIG=YES was specified on the CKPTDEF statement or if the RECONFIG option was specified, the RECONFIG bit is set in $OPTSTAT (this is the same bit set by the RECONFIG option). It is this field that is checked during initialization to determine if RECONFIG was specified.

If this is a COLD or a HOT start, then a check is made to see if CKPTn or RECONFIG was specified. If so, the $HASP487 message is issued and initialization is terminated. If RECONFIG and CKPTn were not specified and this is a hot start, the CKPT specifications in the HCCT replace those specified in the initialization deck.

The specifications for CKPTDEF are then saved in the INIT work area for later use.

Call Dialog Routine

A check is now made to see if RECONFIG was specified. If so the dialog is called to obtain the correct values for CKPTDEF. If the operator replies "CANCEL" to the dialog, initialization will terminate.

Verify CKPTn valid when RECONFIG specified

If the RECONFIG and CKPTn initialization options are both specified, a check is made to ensure that CKPT1 is INUSE=YES if PARM=CKPT1 was specified, and CKPT2 is INUSE=YES if PARM=CKPT2 is specified. If CKPTn is INUSE=NO, the $HASP481 message is issued and initialization is terminated.

Get CKPT Environment From Data Sets

The current mode this system is running in (DUAL/DUPLEX) is set to match the initialization deck. The change log length from the initialization stream is saved. If this is a cold start, the suppress I/O error indicator is set.

The GETCKENV routine is then called. This routine is passed the current checkpoint data set allocation and the specifications the installation wants this system to start with. It adjusts the allocation as needed, acquires (or bypasses) the lock, reads in track one of the data set, and determines if JES2 should come up. It handles all cases of data set forwarding, INUSE= parameter differences, and error on the data sets. The return code will indicate whether or not JES2 should come up.

If JES2 is to come up, I/O error messages are again enabled, and control is transferred to either cold start or warm start processing.

NGWARM: Warm Start Processing

First, the system parameter table is analyzed by the routine NGCIRCHK. If a problem is found, message $HASP435 is issued and JES2 terminates. The routine NGCQWARM is then called to add, change, or delete a system ID. Continuation or termination of JES2 depends on operator replies to the messages issued during this processing. The $HASP493 message is issued indicating the type of start in progress.

If a multi-access spool configuration is indicated, the system being initialized must be inactive and dormant. Otherwise, the operator is informed of the system's status via message $HASP470. The operator has the opportunity to continue by replying to message $HASP471; if the reply is N, processing terminates.

When the CKPTn or RECONFIG initialization option is specified and prior validations indicate that the system cannot perform a multi-access spool all-system warm start, the operator is notified with message $HASP487 and initialization is terminated.

NFSYSPRM: Checkpoint Record Verification Routine: NFSYSPRM compares the portion of the HASP communications table (HCT) containing information that must remain the same across a warm start with the copy contained in the first record of the checkpoint data set read (either the current data set or the data set specified in the CKPTn initialization option). If the information does not agree, message $HASP442 is issued, denying the warm start. Message $HASP496 is then issued for each initialization statement in error.

Read the Entire Checkpoint Data Set: Before reading in the data set, the size of the change log is adjusted to match that of the data set (by calling CKALADJ). The updates portion of the master record is saved and the check, master, and change log records are read from track 1 (by KTRK1IO). The HFAM is then updated from the check record (only the first half is copied if RECONFIG was specified). The control bytes are set to read all the records in the checkpoint and KREAD2 is called to read them in.

If there is an I/O error in either I/O, the dialog is entered (reason=INIT) to determine where the checkpoint is (it is assumed that this is the wrong data set). If the operator replies "CANCEL", initialization terminates. Otherwise control will be returned to early in IRDA, to set up and call GETCKENV.

If KREAD2 encounters an invalid change log entry, the installation is given the option of bypassing the change log (if this is an all-systems warm start). If the installation elects to do this, the length of the used portion of the change log is set to zero and KREAD2 is again called.

Once the reads have completed, the mode that the system will be running in is set. If RECONFIG was specified, MODE will match what the initialization stream said. If RECONFIG was not specified, MODE will be set to match that of the data set.

Allocation of Spool Volumes: The direct access spool control blocks (DAS) are used to determine the current spooling environment. The checkpoint data set contains SPOOLNUM DAS control blocks, one for each possible spool volume (SPOOLNUM can equal 1 to 253). Flags in each DAS control block define its current status. On a warm start, JES2 will attempt to return the spooling configuration to its previous status. JES2 will ensure that all volumes available during the last point of processing are now mounted on the proper devices and that each has the same extent limits as before.

If this restart is to make a previously shared spool volume non-shared, all work from the spool volume is purged, held or terminated -- unless SHARED=NOCHECK is coded on the MASDEF initialization parameter. If SHARED=NOCHECK is coded, this volume is automatically made active. **Note: It is accessible, however, only to the member to which it is attached**. All previously available spool volumes can be found on the $DASTRKQ.

The spool volumes characteristics will be obtained by attaching the HOSPOOL subtask in HASPSPOL. HASPSPOL will also allocate the spool volume.

If a previously mounted spool volume is determined not to be formatted, the $HASP421 message will be issued. If the extent limits do not match, the $HASP401 message will be issued. Both of these errors will cause JES2 to terminate.

No new spool volumes will be added during a warm start. The only time a volume can be removed during a restart, is during a configuration wide warm start. In that case, the $HASP424 message is issued. This message is followed by the $HASP853 message asking the operator what action should be taken. At this point three options are available:

- purge all work on the volume and continue processing
- hold all work on the volume and continue processing
- terminate initialization

If all of the volumes are valid, JES2 will continue processing and return the volume to its previous status. At this point, the bad track group map processing is performed. Control is then given to NGDONE.

NGCOLD: Cold and Format Start Processing Routine

If the initialization options specified a cold start and/or a format start, the NGCOLD routine receives control. First, the entire track group map is zeroed and the bad track group map is initialized with ones. The system parameter table is analyzed by the routine NGCIRCHK. If a problem is found, message $HASP435 is issued and JES2 terminates. The routine NGCQCOLD is then called to initialize the QSEs. A "COLD START IN PROGRESS" message is issued. NGCOLD sets the mode to match the initialization stream and initializes DAS extent numbers and the DASMASK. Then, each mounted spool volume is processed.

The volumes processed are those represented in the volume allocation table. For each volume in the table, the subtask HOSPOOL is attached to obtain SYS1.HASPACE characteristics, allocate and format the volume accordingly.

If the MASDEF parameter specifies SHARED=CHECK, NGCOLD ignores volumes that cannot be shared. If SHARED=NOCHECK is coded, however, HOSPOOL will automatically allocate all spool volumes. When all table entries have been processed, the subtask completions are waited for, one at a time, by waiting for the first subtask to complete. Any error detected by HOSPOOL results in the volume not being allocated.

As processing for each volume is completed, the $DAXTNTA routine in HASPSPOL will be called to initialize the DAS. The master track group map is updated via a call to the $DATGMA routine in HASPSPOL, making available all track groups in the first extent of SYS1.HASPACE on the volume.

If the maximum number of spool volumes is exceeded the $HASP411 message will be issued and the operator is given a choice with message $HASP441 whether or not to terminate JES2. If the extent of a spool volume is larger than the space remaining in the TGM, the operator is informed of the reduced size extent and then the operator is given three options:

1. Use this volume, but with smaller extent limits.
2. Terminate JES2.
3. Continue, but do not use this spool volume.

When processing of all volumes is completed, the cold start time stamp is saved in the HFAM, and the job queue and the job output table (JOT) are formatted. Control is then given to NGDONE.

NGDONE: Final Checkpoint Adjustments

At this point, one last check is made to ensure the system is allocated as it should be. If RECONFIG was specified, the change log size is adjusted to match that specified in the initialization stream. The temporary TOT is $FREMAINed.

The value of TGBPERVL is then verified. If not specified during initialization, it is set to the largest number of entries that will fit (up to 5). If it was set in initialization, a check is made to ensure the BLOB is large enough to accommodate the number of allocated spool times the TGBPERVL. If the BLOB is too small, TGBPERVL is adjusted to the maximum possible with this configuration and the $HASP834 message is issued.

The BADTRACK initialization statement information from the NBADTRAK table is used to set the appropriate bits off (0) in the permanent and temporary badtrack group maps. If a hot start is being performed, this routine goes to the NGBTGFRE routine and frees the storage in use by the NBADTRAK table via the $FREMAIN macro. If an all-systems warm or cold start is being performed, the storage required for the temporary bad track group map is obtained via the $GETMAIN macro and this map is initialized to ones.

At this point if no BADTRACK statements exist, NGDONE issues the $FREMAIN macro to release the NBADTRAK table storage and control is returned to the caller.

If there are BADTRACK statements, NGDONE continues processing. The DAS control blocks are searched to find the correct entry. If none is found, the $HASP861 message is issued. After NGDONE finds the proper DAS control block for the spool volume, the cylinder and head location (CCHH) for the first address is converted to a relative track address (MTTR). NGDONE uses the $TGMSET macro to mark the track bad (indicated by a bit value of zero) in both the permanent and temporary track group map. If a non-zero return code was returned from the first $TGMSET macro, NGDONE issues message $HASP861 to indicate a bad address, then processes the next statement.

If the first address is the same as the last address, the bit has already been set off and processing for the next statement is started. Otherwise, the same processing occurs for the last address as has just been completed for the first address. If the first address is valid and the second address is invalid, the first track group is still marked bad.

After the first and the last bits are marked, all the bits in between are then marked bad in both maps. This is done by first marking any remaining bits in the byte that contained the first address (stopping at the bit of the last address, if it's in that byte). Then, any entire bytes in the maps are marked bad. Finally, all bits until the one representing the last address are marked bad in the last byte. A test is then made to see if there are any more statements. If not, the NBADTRAK table is freed via the $FREMAIN macro, and control is returned to the caller.

NGQANAL: Job Queue Analysis Routine

A test is made to determine if a multi-access spool all-systems warm start is being performed. If not, processing continues with routine NGCKPT. Otherwise, processing continues with routine NGQANAL.

NGQANAL analyzes each element in the JES2 job queue and zeroes out the job queue index (JIX). Then, as each JQE is analyzed, the relative job offset (RJO) of the JQE is placed in the appropriate JIX entry. This analysis detects a JQE that is not on the correct chain, a JQE with an invalid queue type, and a chain that loops upon itself. If an error is detected, control passes to routine NGBLDJBQ. Otherwise, control passes to NGBLDJOT, which calls $#JOTCHK in HASPJOS.

NGBLDJBQ: Rebuild JES2 Job Queue Routine

The NGBLDJBQ routine first sets the $CKPERRQ flag in the checkpoint information byte $CKPTFLG. This byte, later copied to the checkpoint record, contains a record of special processing performed on behalf of JES2 checkpointing and the JES2 checkpoint data sets. This record is kept until the next JES2 cold start. NGBLDJBQ queries the operator with message $HASP483 to determine if a job queue rebuild should be performed. If the operator replies no (N), message $HASP441 is issued to ask the operator if JES2 initialization should continue. If the reply to $HASP441 is N, JES2 is terminated. Otherwise, processing continues at NGBLDJOT, where $#JOTCHK in HASPJOS is called.

If the reply to $HASP483 is yes (Y), the $CKPBLDQ bit is set in $CKPTFLG, the JES2 "REQ" initialization option is forced and the job queue is rebuilt.

To rebuild the job queue, the free queue and each type or class queue is rebuilt, one element at a time. The JQEs are analyzed in address sequence to eliminate dependencies on JQE chains.

Only validated JQEs are returned to the class or type queues. Free JQEs and any invalidated JQEs are put on the free queue.

The class and type queues are rebuilt in order of job number within priority sequence. This logic is different from that used to queue the JQEs originally. The reason for using the job numbers for queue ordering is to increase the likelihood that the JQEs will end up in the FIFO queue sequence that existed prior to the rebuild. The job queue index (JIX) entry for each JQE is then filled in.

When the job queue rebuild is complete, control passes to NGBLDJOT.

NGBLDJOT: Rebuild JES2 Job Output Table Routine

At label NGBLDJOT, the $#JOTCHK routine in HASPJOS is called to verify the JOT. Note that this verification (and rebuild, if necessary) is done only at initialization during a multi-access spool all-systems warm start. If a rebuild was necessary and failed, initialization terminates.

When the JOT rebuild is complete, control passes to NGCKPT.

NGCKPT: Final Initialization Checkpoint Routines

When NGCKPT is entered, checkpoint information in storage has already been built and/or validated. The systems bit and the duplex bit (if this system is duplexing) are reset in all the CTLBs. The to-be-written-to data set is set for HASPCKPT. The $QSE bit is set to indicate how many data sets this system is allocated to. KFORMAT is called for the down-level data set (if there are two data sets), and then the current data set. If there is an I/O error on either format, the dialog is called (reason=IOERR,K15). The DASD RESERVE and JES2 checkpoint locks will not be released by NGCKPT. Once the formats are done (or the dialog completes), IRDA page releases the master record I/O area and returns to the caller.

IRDA Support Routines

These subroutines are used in the allocation and initialization of direct-access device space. They use the $DTEDYN macro to attach and detach HOSPOOL to dynamically allocate and, optionally, format a spool or checkpoint volume.

GETCKENV: A check is made to ensure that at least one checkpoint data set is in use. If not, the $HASP472 message is issued, and dialog processing is entered. (Dialog processing in GETCKENV involves calling KDIALOG (reason=INIT). If the operator does not cancel the dialog, GETCKENV is called recursively. Otherwise, JES2 is terminated). CKALCADJ is then called to adjust the checkpoint data set allocations so that they match the specifications in the HFAM. If there is a problem allocating the data sets, dialog processing is entered. If either data set is locked, KTRK1IO is called to release the lock on that data set. If the reserve is held on a data set, KRELEASE is called to release that reserve.

The reserve is now obtained on CKPT1, if it is allocated (otherwise CKPT2), by calling KRESERVE and then calling KNOP. This prevents other systems in a multi-access spool configuration from updating the checkpoint records during initialization. If an I/O error is encountered attempting to get the reserve, KIOERROR is called and dialog processing is entered. GETCKENV attempts to acquire the checkpoint lock record by calling KTRK1IO. The parameter list passed to KTRK1IO specifies to attempt to lock a specific data set if that data set is allocated, and the data set has not had an I/O error on it, and if RECONFIG and CKPTn (where n is the other data set) was not specified. If KTRK1IO reports an I/O error on one of the data sets, a check is made to see if there are any data sets left that are eligible to have I/O done to them. If not, GETCKENV branches to issue the $HASP479 message. If there is a data set eligible for I/O, the data set that had the I/O error is made non-eligible for further processing.

If there is at least one eligible data set, a check is made to see if the lock was obtained. If the lock could not be obtained on any data set, a check is made to see who owns the lock. If this is a hot start and the data set is locked by this system, it is assumed that the lock was obtained. Whether or not the lock was obtained, a check is made to see if the data set read is a checkpoint data set. If not, an I/O error is simulated on the data set and GETCKENV branches to the code that handles an I/O error from KTRK1IO. If there are two eligible checkpoint data sets and this is not a cold start, a check is made to ensure both data sets are from the same cold start date. If not, the $HASP482 message is issued and initialization is terminated. CHCKSPEC is then called to ensure that the checkpoint specifications that are currently being used are allowed (based on the data set). CHCKSPEC will indicate the current specifications are correct, process a different specification, or encounter an error that causes initialization to terminate. If the current specifications are not correct, GETCKENV will return to its caller with an appropriate return code.

If the current specifications are correct, a check is made to see if any data set forwarding was done. If so, VERIFYCK is called to verify the current specifications. If the the operator indicated the current specifications are not correct, the dialog process is entered.

If the lock was not obtained, message $HASP488 is issued and the system attempts to retry the operation at 1-second intervals for up to 30 seconds. If attempts to retry the operation fail to acquire the checkpoint lock, or there was an I/O error on all eligible data sets, message $HASP479 is issued, allowing the operator the option of continuing initialization without the lock protection. Note that the locking operation is a backup mechanism intended to protect the checkpoint from failures of the RESERVE/RELEASE function. When functioning properly, the RESERVE mechanism protects the checkpoint from simultaneous updating.

If this is a cold start, the $HASP436 message is issued to confirm the checkpoint data sets and spool volumes being cold started.

The appropriate checkpoint data set (either the highest level data set, the locked data set, or the data set specified by the initialization options CKPTn) is read by calling KTRK1IO to determine the system environment. If the CKPTn initialization option was specified and the data set is not allocated, the $HASP481 message is issued and initialization is terminated. The $HASP478 message is issued to inform the operator of the data set read from. If an error is encountered on that data set and this is not a cold start, dialog processing is entered. The beginning of the master record and the write-check-value record are transferred to storage. GETCKENV then verifies the format of the master record. The value of the write-verification check record (CKBCKVAL) and the value in the master record $WCHECK are compared to ensure checkpoint integrity. If this comparison indicates checkpoint damage, message $HASP486 is issued and the operator is asked for permission to continue initialization with the damaged checkpoint data.

If the checkpoint data set can be read, GETCKENV attempts to determine the status of any other systems in the multi-access spool configuration. The current TOD is reduced by the number of seconds specified by the SYNCTOL on the MASDEF initialization statement. The resultant value is compared to the TOD recorded when each active system in the multi-access spool configuration last wrote a checkpoint record. If a system is found whose checkpoint data is more recent than the computed value, that system is assumed to be running. Therefore, the system being initialized is down and must perform a single-system warm start.

If one or more systems in the node are apparently active but not running, the operator must confirm the status of these systems before a cold, format, or all-systems warm start can be accomplished.

If a cold or format start has been requested and a multi-access spool configuration has been defined in at least one system is apparently active, the operator is informed of the status of the complex with message $HASP470. The operator then has the opportunity to continue by replying to message $HASP471; if the reply is N, initialization processing terminates.

CKALCADJ: This routine is passed 2 sets of HFAMs, one describing which data sets should be allocated and the other describing what is currently allocated. If they do not match, CKPTUNAL is called to unallocate any data sets that need to be unallocated (freeing any control blocks relating to those data sets) and CKPTALOC is called to allocate the data sets that need to be allocated. CKBINIT and KBLD4KP are called as needed to build the necessary control blocks for any data sets allocated. In addition, a new change log size can be specified when calling CKALCADJ. If the new size will fit in the data sets, the old control blocks relating to the data sets are freed and new ones are built reflecting the new change log size. If any errors are encountered allocating the data sets, an appropriate return code is passed back to the caller.

CHCKSPEC: This routine compares the in-storage HFAM to the HFAMs in any data sets that were read. If they indicate a different checkpoint configuration existed the last time this system came down and this is not a cold start nor was RECONFIG specified, the new specifications are processed. If the data set name or volume serial of a data set was changed (the data set was forwarded), the $HASP438 message is issued to ask if forwarding is to be done. If the reply is 'N', the dialog is entered. If the reply is 'Y', then the $HASP458 message is issued for each data set that was forwarded. Then, GETCKENV is called to process the new specifications. The $HASP438 message is issued every eighth time through CHCKSPEC.

If the data set's INUSE indicators are the only things that are different, GETCKENV is called (without any messages).

If the specifications changed and this is a cold start or if RECONFIG was specified, the $HASP438 message is issued, telling the operator that a forwarded data set was found, but no forwarding is done. If this is a warm start, NQUERY is called to confirm that the operator wishes to continue.

If the specifications all match, control is returned to the caller with a return code of zero.

VERIFYCK: This routine issues the $HASP416 and $HASP417 messages to request verification of the checkpoint data set specification. It will issue the long form of the $HASP416 message if the current checkpoint specifications are different from that specified in the initialization deck, and the short form if they are the same.

NGWAIT: This subroutine waits for the return from HOSPOOL. At that time, it checks for any errors detected by HOSPOOL and, if there are any, issues the $HASP443 message.

NGCIRCHK: This routine ensures that the system parameter table does not contain duplicate entries and ensures that this system's id is in the table. If this system's id is not present, NGCIRCHK puts it there. This routine also ensures that a duplicate system id does not exist.

NGCQWARM: This routine checks to see if a member is to be added, changed, or deleted from the MAS. The $HASP865 message is issued to inform the operator of this system's conclusions based on the status of the system parameter tables. Then, the $HASP870 message is issued to verify this system's conclusions. If this system determines that the requested action is not possible, the $HASP876 message is issued telling of the problem and asking whether initialization should continue.

This routine also locates this system's ID in the parameter table and initializes the $HCT field for this system. It determines if the remote console PCE is required; if so, it ensures the count for that PCE is set to one.

NGWTOR1: This subroutine calls the NQUERY routine in HASPIRA to obtain the operator response to a request.

HASPIRRE: NJE/RJE Initialization

HASPIRRE contains the IRNJE and IRRJE JES2 initialization routines. These routines perform the major portion of the initialization required for NJE and RJE post-parameters.

IRNJE: Network Job Entry Initialization Routine

IRNJE completes NJE initialization, including the following functions:

NEXTNIT1: The NEXTNIT1 routine scans all entries in the temporary node information table to ensure that no two nodes have the same name and that a default name of Nnnnn (where nnnn is the node number) is assigned if no name was supplied in the initialization parameter statements. If duplicate names are found, a $HASP445 message is issued, and JES2 terminates.

APWNIT: This routine verifies information in the APW against information in the node information table (NIT) examining each element. If NITFLAGV is set to 1 (indicating that SNA was specified on the Nnnnn statement), a branch is taken to APWAPSCN where APW elements are used to build the NJE/SNA application table (APT).

This routine performs various checks on each APW element. If an APW element is found for a BSC node, message $HASP473 is issued. If an APW element is found for a node outside the $NUMNODE range, message $HASP474 is issued. If no application is defined for the local node, message $HASP475 is issued. If duplicate applications are defined, message $HASP477 is issued. If any of the above errors are found, JES2 initialization terminates.

NNITMOVE: This routine determines the final value of NATNUM on the NJEDEF statement by forcing its value to be the greatest of the following:

- the value specified in the NATNUM parameter statement
- 1
- 1 less than NODENUM (on the NJEDEF statement).

The size of the permanent node information table (NIT), nodes attached table (NAT), and associated NAT stack area are calculated and storage is obtained from subpool 0. The size of the NIT is a function of the value of the NODENUM and PATH parameters, while the size of the NAT and stack are dependent upon the final value of NATNUM. The final values of $NITABLE (address of NIT) and $NATABLE (address of NAT) are set, and the permanent NIT is set using values from the temporary NIT. Path elements within the NIT are set to indicate that no nodes are connected at this time.

NNATL: The NAT is initialized to reflect the results of the predefined connections described by the CONNECT parameter statement. As each table element is created, the storage for the NAT temporary queue element (NTQ) representing the connection is freed. If the size of the NAT is exceeded during the creation of elements, message $HASP446 is issued and JES2 terminates.

NNATE: The NNATE routine initializes the network path manager processor control element (PCE) so that the path manager can take over management of the NIT and NAT elements.

NACC1160: The network account table initialization routine is given control. This routine builds the two NETACCT lookup tables from the chains built by NPLNACT. It computes the amount of virtual storage necessary to hold the completed tables obtains the necessary space by issuing the $GETMAIN macro. The amount of space required for each table is computed and divided by 4096; the remainders are then added together. If the sum is less than or equal to 4096, one $GETMAIN macro is issued for enough space to hold both tables, with alignment forced to a page boundary. If the sum of the remainders is more than 4096, a separate $GETMAIN macro (with alignment forced to a page boundary) is issued for each table. This process minimizes the number of virtual storage pages occupied by each table and, thus, reduces the paging caused by the lookup routines in HASPACCT.

After space for the tables is obtained, each table is built by moving information from the chains of entries built by NPLNACT into consecutive entries in the table. When both tables have been constructed, the entire virtual storage subpool occupied by the chains of NETACCT entries is deleted by $FREMAIN; this subpool contains only the NETACCT chains. Pointers to the start of each table are placed in fields in the HCT.

NJESBLD: NJESBLD initializes the header of the JES2-to-NET account table and moves entries into the table.

NNETBLD: NNETBLD initializes the header of the NET-to-JES2 account table and moves entries into the table.

IRRJE: Remote Job Entry Initialization

IRRJE completes final RJE initialization by storing remote DCT addresses and line DCTs into the RAT and initializes the RAT with RJE information, such as the number of printers, the number of readers, and the node id.

IRRJE then invokes NCRMTDCT three times for each remote terminal. NCRMTDCT invokes NRDCTINT, which initializes default values in the DCT; NCRMTDCT then completes the DCT initialization for remote readers and remote printers and punches. NCRMTDCT completes the initialization of the remote destination tables (RDT) and checks for conflicts between the node information table (NIT) and the RDT.

IRRJE then invokes the NBFBUILD subroutine to build the RJE buffers and obtains permanent storage for the CPT and initializes it. Permanent storage for the RAT and RRT is then obtained and the correct DCT RAT addresses are initialized. IRRJE also obtains and initializes storage for ICEs. IRRJE completes its processing by freeing temporary storage for remote device DCTs, RATs, PITs, CPTs, NITs, and APWs, then returns to the IR administrator.

HASPSCAN: $SCAN Facility

The $SCAN facility parses input character strings, initializes control block fields based on the input string, or optionally displays the initialized control block fields. JES2 uses this $SCAN facility to analyze JES2 initialization statements.

The input string must follow a set of syntax rules (pre-scan and post-scan exits exist in HASPSCAN to allow for processing of a non-standard syntax). Two tables, generated by the $SCANTAB macro, are used to define the actions HASPSCAN is to take in setting or displaying parameters, initialized control blocks, or the pre-scan or post-scan exit routines. For JES2 parameter processing, HASPSTAB contains the scan tables used to process the keywords in the input. HASPTABS contains the $MCT, which points to the correct $SCAN table in HASPSTAB.

The $SCAN facility can recursively call itself; this allows various levels of sub-scanning to occur. At each level of scan, HASPSCAN processes a $SCAN internal work area (SCWA), which it uses to hold the scanned information.

$SCAN is the entry point for $SCAN processing. The main loop consists of the following steps. First $SCAN establishes a $ESTAE. Then it finds the beginning of the keyword and calls PRKEYWRD to process the keyword. It loops to find the beginning of the next keyword. When no more keywords can be found, the routine cancels the $ESTAE and returns. If an error occurs, the routine issues a diagnostic message, restores the values that may have been changed, and returns to the caller.

PRKEYWRD calls FINDTAB to isolate the keyword and find the table entry. For each subscript found, PRKEYWRD calls PRTABENT to process the keyword based on table controls and issues a display if this is a display request.

The register conventions throughout $SCAN are:

R0 - parameter passing the registers

R1 - parameter passing and, on return, the diagnostic pointer

R2 - address of current location in the input

R6 - increment register (1)

R7 - address of last character of $SCAN input

R8 - address of the SCANTAB for the keyword $SCAN is currently processing

R10 - address of $SCAN's work area (SCWA)

R11 - address of the $HCT

R12 - base register

R13 - address of the PCE

R14 and R15 - linkage registers

HASPSCAN Support Subroutines

The following subroutines are used by HASPSCAN throughout the scanning process.

$SCANCOM: Find a Comment

This routine, which can be called from anywhere in the main task with the $SCANCOM macro, finds valid comments in input character strings and returns in register 1 a pointer to where the input string resumes. Return codes and their meanings are:

0 - a non-blank character and no comments were found; register 1 contains the address of the character

4 - a valid comment was encountered and a non-blank character was found; register 1 contains the address of the character

8 - an end-of-statement was encountered

12 - an invalid comment was encountered

FINDTAB: Find $SCANTAB Entry

FINDTAB analyzes the next segment of the parameter statement and locates the required $SCANTAB entry. FINDTAB isolates positional values, or vectors keyword values and bypasses blanks. After the specific parameter is isolated, FINDTAB searches the $SCAN tables for a match to the parameter. If a match is found, FINDTAB initializes the SCWA, sets a zero return code, and returns to the caller. If a match is not found, the following return codes indicate the reason:

RC=4	An obsolete parameter is used
RC=8	A subscript error exists in the parameter
RC=12	An unsupported parameter is used
RC=16	An invalid parameter is used
RC=20	A vector error exists

FINDTAB also recognizes whether or not a SET-DISPLAY request is being processed and issues a $SCANB call passing the STABNAME to be saved in the backup area. STABNAME is the keyword displayed in this case.

FINDTAB then returns to the caller.

FTABFIND: $SCANTAB Table Search Subroutine

FTABFIND searches a $SCANTAB table for an entry that matches the current parameter keyword. When the match is found, FTABFIND validates the caller's id.

FINDCB: Find Control Block Subroutine

FINDCB locates the address of the control block associated with a particular scan table entry. If indicated by the scan table entry, FINDCB allocates a temporary control block (using $GETMAIN). Once the control block is located, FINDCB performs any subscript indexing that may be required.

During initialization for scan 'set' requests, if the control block is a DCT, FINDCB calls $DCTDYN. If the DCT has already been built, the address is returned in register 1. If not, a DCT is constructed.

FINDVALE: Input Value Compare Subroutine

FINDVALE compares an input value against the set of value elements in a scan table entry to see if the value is valid. If the value is valid, a return code of 0 is returned to the caller. If the value is invalid a return code of 4 is returned to the caller.

DISPLAY: Display Current Values Subroutine

DISPLAY displays the current values for control block fields that are associated with the keyword that is indicated in the SCWASTAB field.

On entry, DISPLAY issues a $SCAND call to place the passed keyword and any subscript into the display SCWA. If a subscan is necessary, DISPLAY issues a $SCAN call. Then DISPLAY issues $SCAND to place the delimiter and the value to be displayed in the display SCWA. Values displayed are characters, flag setting keywords, text strings or numerical values. DISPLAY displays the value and returns a return code of 0 to the caller. If the value is not displayable, DISPLAY issues the following return codes:

RC=4	value not displayable
RC=8	not enough room to display the value

DISPLAY then returns to the caller.

DISPARSE: Segment Message Subroutine

DISPARSE gains control after the building of a message for a first level parameter statement request and after appropriate post-scan routines have processed the text. This routine parses the message into segments that fit within the specified message area and then calls the specified display routine to display the segments until the entire message is displayed.

$SCANB: Scan Backup Subroutine

$SCANB is used to backup the control block field that is to be changed by a scan setting. The $SCANB macro is used to call $SCANB. The backup is provided by using additional scan work areas by obtaining (using $GETMAIN) additional storage to hold the information of a parameter that is too large to fit in one of the pre-allocated storage areas. $SCANB copies the field to be backed up into the backup area, changes the SCWA to point to the backup area, sets a bit to indicate to RESTORE what it is looking at, and returns to the caller.

$SCAND

$SCAND is called with a $SCAND macro instruction issued by $SCAN, or a $SCAN pre- or post-scan routine, to place a specified text string into a display SCWA.

RESTORE: Restore Control Block Fields Subroutine

RESTORE restores control block fields that have already been set by previous scanning if an error was found. RESTORE distinguishes between normal backup areas, SET-DISPLAY, and diagnostic trace backup areas. RESTORE scans SCWAs and returns their storage.

RESTDISP: Search Backup Areas Subroutine

This is a $SCAN facility routine that searches the backup areas for keywords to display when a SET completes successfully. The routine is called only for a SETDISP (set-display) call to $SCAN to complete the display portion of the command. It searches the backup areas, starting at the top and working its way to the bottom, looking for keywords. When one is found, this routine issues a $SCAN call to process the display. In the process of looping through the SCWAs, these areas are returned.

SCANDIAG: Diagnostic Subroutine

If there is a keyword in error, this routine isolates the keyword in error, builds the diagnostic message and calls the $SCAN caller's message routine to issue the diagnostic. The routine scans the backup areas looking for diagnostic trace entries. These entries contain the keyword and information that describes at what level of $SCAN the keyword was. The last keyword for each level is displayed in the trace. The last keyword displayed in the trace is the keyword that was in error. For example, for the statement:

```
PRT1  CLASS=A,CKPTLINE=1,CKPTPAAGE=1
```

the SCWA would have:

Keyword	***Level***
PRT1	1 -- $SCAN, end of scan
CLASS	2 -- subscan
CKPTLINE	2 -- subscan
CKPTPAAGE	2 -- subscan, keyword in error

and the trace would be:

```
RC=(xx), PRT1 CHKPTPAAGE
```

Here is a second example:

If the statement is:

```
OFF1.JR  CLASS=A,MOD=(CLASSS=A)
```

the SCWA would have:

Keyword	***Level***
OFF1.JR	1 -- $SCAN, end of scan
CLASS	2 -- 1st subscan
MOD	2 -- 1st subscan, end of 1st subscan
CLASSS	3 -- 2nd subscan, keyword in error

and the trace would be:

```
RC=(xx), OFFn.JR MOD CLASSS
```

HASPSXIT: Pre-scan and Post-scan Exits

HASPSXIT contains all pre- and post-scan exits for the $SCAN facility. The following text describes some of the important exits. See Chapter 4, "Directory" for a list of all the exits and the statements they process.

$SCAN Facility Post-Scanning Exit Routines

These routines process application-specific parameter statements unique to the installation.

PSTAPPL: PSTAPPL validates and processes the APPL parameter statements. To do this, PSTAPPL verifies that the required operands are specified; the node number and application id must be specified. Then PSTAPPL updates APW slots and chains the new APW element in collating order. To do the chaining, PSTAPPL calls a chaining subroutine whose address is in the CIRAPPCH field of the CIR PCE work area.

PSTBADTR: When a BADTRACK statement is being processed, PSTBADTR validates the volume associated with the bad track and validates the specified address range. After this validation, PSTAPPL adds the bad track group (BTG) element to the initialization queue.

PSTCAT: PSTCAT post-scans the STCCLASS and TSUCLASS initialization statements. It ensures that the STC and TSU CAT entries indicate no journaling and no restart for these jobs, and that accounting information and programmer name are not required.

PSTCHARS: For the CHARS sub-operand of the COMPACT statement, PSTCHARS validates the amount of input supplied and processes the master and non-master character that is supplied.

PSTCKPT: This routine verifies that the parameters passed on the CKPTDEF statement are valid. If valid, and RECONFIG=YES was specified, the checkpoint processor is $POSTed.

PSTCKPTN: This routine ensures that, if a devname is present, this is also a volser, and, if inuse=YES, DSN and VOLSER are non-blank.

PSTCOMP: PSTCOMP processes the COMPACT statement.

PSTCONCT: PSTCONCT validates and processes the CONNECT parameter statements. PSTCONCT validates that the required operands (NODEA, MEMBA, NODEB, MEMBB) have been specified and queues a temporary nodes attached table (NAT) element (NTQ) for processing later.

PSTDEST: PSTDEST validates and processes the DESTID parameter statements.

PSTEXRTN: PSTEXRTN validates and processes the ROUTINE sub-operand of the EXITnnn parameter statement. PSTEXRTN invokes PSTEXLOC. PSTEXLOC locates the entry point for a given exit routine name by searching all module information tables (MITS) that are available via $REPTABL and $LMT. PSTEXLOC invokes PSTEXFND. PSTEXFND searches a given MIT for the entry point associated with a given exit routine name.

PSTFSSDF: PSTFSSDF validates the FSSDEF statement for the required operands. Default values are supplied as needed and these values are added to the functional subsystem control blocks (FSSCBs).

PSTHOLD: PSTHOLD processes the HOLD parameter on the MASDEF statement. When called because of operator command processing, the checkpoint processor is posted that the HOLD value changed. When called from initialization or command processing, PSTHOLD sets the maximum interval to wait before initiating a checkpoint write. This value is based on the HOLD parameter.

PSTJRNG: PSTJRNG validates the RANGE parameter on the JOBDEF initialization statement.

PSTLIMIT: PSTLIMIT verifies that the LIMIT and PLIM sub-operands of the output device initialization statements are valid.

PSTLINE: PSTLINE processes the UNIT=SNA initialization specification.

PSTLOAD: PSTLOAD loads an installation exit routine that is specified via the LOAD initialization statement. It calls MODCHECK to verify the modules.

PSTNODE: PSTNODE processes the Nnnnn (node) initialization statement. PSTNODE determines if a node name is specified, and if so, determines if the node name is valid and whether a temporary destination queue element (NDQ) has already been allocated for that node name. If the node name is not specified, PSTNODE returns to the caller with a return code of 0. If the node name is invalid, a return code of 4 is returned.

PSTNETAC: PSTNETAC validates and processes the NETACCT initialization statement. PSTNETAC checks for the required operands (JACCT, NACCT) and checks whether the JTHRU and NTHRU operands are specified and valid. PSTNETAC then queues the temporary NETACCT element on the JES2 network chains.

PSTNRT: PSTNRT validates the number of network receivers and transmitters that are set via the SRNUM, the JRNUM, and the STNUM parameters on the NJEDEF initialization statement.

PSTPRMD: PSTPRMD validates and processes the PRMODE sub-operand of the local and remote printer/punch initialization statements and the offload SYSOUT transmitter. PSTPRM updates the system PRMODE table and the DCT PRMODE indices, decreases by 1 the counts for each of the PRMODEs indicated in the old DCT index list, and processes each PRMODE by adding it to the system PRMODE table (if it's new) and increasing its use count by 1. An error occurs if a new PRMODE is specified and the system PRMODE table is full.

PSTRDR: PSTRDR validates and processes the RDR(nn) initialization statement.

PSTRMT: PSTRMT validates and processes the RMT(nnnn) initialization statement. It also sets the current time in the remote attributes table (RAT) for any remote being set to autologon mode.

PSTRC: PSTRC validates and processes the ROUTECDE sub-operand of the PRT(nnnn) and PUN(nn) initialization statements and the OFFn.JT, OFFn.ST, OFFn.JR, and OFFn.SR statements.

PSTSAF: PSTSAF post-scans the system-affinity sub-operand on the OFFn.JT and OFFn.JR statements. It moves the affinities into storage so that later during initialization they can be validated by a routine in HASPSERV. For commands, PSTAF also post-scans the system-affinity sub-operand. Then, if the system affinity is not valid, PSTAF issues an error message.

PSTSELCT: PSTSELCT validates and processes the SELECT sub-operand of the RnnnnPRn and RnnnnPUn initialization statements and the OFFn.JT, OFFn.ST, OFFn.JR, and OFFn.SR statements.

PSTSPL: PSTSPL processes the SPOOLNUM parameter on the SPOOLDEF statement. Whatever value the installation specifies for SPOOLNUM, $SCAN rounds it up to a multiple of 32. Although the limit is 253, for any specified value equal or greater than 225, $SCAN rounds it up to 256. This value ($SPOLNUM) is compared to $MAXDA and, if it is greater, is set to $MAXDA. This routine ensures that the size specified for TGBPERVL is big enough to accommodate all spool volumes. If not, the size will be altered by this routine. Flag bits will be set to indicate the BLOB needs to be totally replenished and a new ending address has to be set. If the routine is entered for a command set call, no further processing is done.

PSTTHRS: PSTTHRS calculates the threshold value for a given initialization parameter.

$SCAN Facility Pre-Scanning Exit Routines

These routines provides processing of initialization statements prior to scanning facility processing. See Chapter 4, "Directories" for a list of all the exits and the statements they process.

PRECKPT: PRECKPT ensures that this is not a command set call and RECONFIG processing is not active.

PRERNG: PRERNG pre-scans the RANGE sub-operand of the OFFn.JT, OFFn.ST, OFFn.JR, and OFFn.SR statements and the local and remote printer and punch statements.

PRETRCID: PRETRCID process the id sub-operand for the TRACE initialization statement. PRETRCID uses the PTIDCNVT subroutine to convert the id to a binary value, and PRETRCID uses the PTIDACTD subroutine to verify the id and activates or deactivates the id.

PREWS: PREWS pre-scans the WS sub-operand of the OFFn.JT, OFFn.ST, OFFn.JR, and OFFn.SR statements and the local and remote printer and punch statements.

HASPNUC: The JES2 Nucleus

HASPNUC provides central service subroutines to the JES2 system.

$MODLOAD

$MODLOAD is a module load routine called through the $MODLOAD macro interface. It performs the following three types of loads:

- Normal load - The search order for the normal load is:
  1. JOBLIB or STEPLIB
  2. link pack area (LPA)
  3. LINKLIB.
- Directed load - When the DCB parameter and the directory entry (DE) is coded on the MVS load instruction, the module is loaded from the specified library.
- Load of LPA - When the macro call is made with subpool=LPA and the module is also in the STEPLIB, this load call is made with the STEPLIB directory entry changed to look like the link list. In this case the load searches the LPA first.

A return code of zero indicates that the module was loaded properly. A return code of 4 indicates that the load was not done because the module was already loaded. If the return code is 8, the load failed. In this case, register 1 points to the diagnostic message.

$SUBDEST

This routine performs all processing necessary (setup and cleanup) to subtask a call to the $DESTCHK destination authorization routine. $SUBDEST issues a $GETWORK macro to obtain an SQD, then issues a $SUBIT to subtask the call. A return code of 0 indicates successful security authorization processing and a binary route code is returned to the caller in register 1. $SUBDEST sets a return code of 4 to indicate unsuccessful processing (an SQD could not be obtained, a subtask failure occurred, or authorization failure). In the event of a subtask failure, HASPNUC issues the $HASP077 message indicating an access request is denied.

$MODCHK

$MODCHK is a module verification routine called through the $MODCHK macro interface. It provides:

1. Validation for its callers, including tests for load module residency, version equivalence to that of the JES2 HASPNUC, equivalence of MIT and module names, and exit points establishment
2. Measures to ensure that the load module is large enough for a module information table (MIT)
3. That MITENTAD (a pointer in the MIT to the MIT entry table, which is an index to the names and addresses of the module's entry points) points to a field within the addressing range of the module
4. That the MITNAME is the same as the name of the CSECT.

$MODCHK gains control when a $MODCHK macro is issued (calls are possible from IRMODCHK in HASPIRMA, IROPTS in HASPIMA, NSSSM in HASPIRA, and the $SCAN post-scan exit routine for the LOADmod(jxxxxxxx) initialization statement) indicating the name of the module to be checked, which tests are to be performed, and whether or not to issue a HASP875 message for a failed test. It is able to verify that:

- The module resides below 16 megabytes in virtual storage by comparing the module's ending address to 2 raised to the 24th power.
- The module resides in common storage (CSA) by using equates for starting and ending addresses of common storage.
- The module is large enough for a MIT and $MITABLE points to an addressable field. To ensure that the data at the entry point is a MIT, the MIT contains a 4-byte identifier field. Addressability is checked by comparing the address of the MIT entry table to the starting and ending addresses of the load module and ensuring that it falls within that range.
- The version of the module in question is the same as that of the nucleus, including the level of the macro libraries being used, by checking the $VERSION field in the HCT against the MITVRSN field of the module's MIT, and the $MACVERS in the HCT against the MITMVRSN of the module's MIT (for the MACLIB check).
- The name of the module by comparing the MITNAME field in the MIT with the NAME= keyword used in the $MODCHK call.

The module also propagates the defined exit points bit map in the module's module information table (MIT) to the HASP exit information table (XIT) entries. It also resolves the addresses of exit routines in the XIT.

A return code of 0 indicates all tests were successful. A return code of 4 indicates a verification failure. If the return code is 8, register 1 points to a diagnostic message. If MESSAGE=YES is coded on the macro, this message is used as the reason text with the $HASP875 message.

$MODELET

$MODELET is a module deletion routine called through the $MODELET macro interface. It deletes the specified JES2 load module. If the module was directly loaded, the storage is freed.

$MODELET gains control when a $MODELET macro is issued. $MODELET is entered by issuing the $CALL macro.

A return code of zero indicates that the module was deleted properly. A return code of 4 indicates that the module was not found.

Dispatcher

The JES2 dispatcher allocates processor time to the JES2 main task processors. It performs its function by manipulating the JES2 processor control elements (PCE) on the dispatcher's ready and resource wait queues. A processor is considered able to use processor time if its PCE is on the $READY queue. Conversely, a processor with its PCE either queued to itself or on a resource wait queue is not able to use the CPU. The PCE of the currently active processor is on the $READY queue and is addressed by the $CURPCE field of the HASP communications table (HCT).

Queuing Structure

The dispatcher's queues are double-threaded; each PCE within the queue points to the next PCE as well as the previous PCE. The queue head points to the first and last PCE in the queue. Conversely, the first and last PCE in the queue point back to the queue head (offset so that the queue head appears to be a PCE itself). The queue head is referred to as PCE zero. If the queue is empty, the queue head points to PCE zero.

Event Wait and Control Fields

Each processor control element in JES2 contains an event wait field, PCEEWF. When the processor executes a $WAIT macro, a $POST inhibit flag is normally set in the PCEEWF field. If the wait is for a resource, the $EWFPOST flag is set on; otherwise, the flag representing the specific awaited event is set. For example, if a processor waits for I/O, the $EWFIO flag is set on. A flag set in the PCEEWF prevents the execution of a $POST macro instruction from actually scheduling the processor unless the $POST macro instruction specifies the corresponding inhibitor. A $POST directed to a specific PCE with a resource specification is ignored even though both $WAIT and $POST specify the same resource. If, however, the $WAIT macro instruction includes the INHIBIT=NO operand, the first byte of the PCEEWF field is set to 0, and any $POST macro instruction directed to the PCE schedules the processor on the $READY queue.

In the event a PCE is individually queued to the $READY queue, the $EWFPOST bit is set to prevent the overhead of requeuing the PCE on the issuance of succeeding $POSTs when not required.

There are three event control fields within JES2. Two of these, $HASPECF and MHASPECF, are contained in the HASP communications table (HCT). The $HASPECF field contains flags that represent whether a $POST for resource macro instruction has been executed and the requeuing process is still pending. If a flag bit is on, no action is required for the resource; conversely, if a flag bit is off, the corresponding resource is to be posted. The MHASPECF field contains the single flag indicator $EWFJOT and is used by the line manager to inform processors using the RTAM facilities about posted resources. The third field, CCTECF, is contained in the HCCT and represents whether a resource post has been requested by a source outside the synchronous routines of the JES2 main task. If the flag is on, the resource is to be posted; otherwise, the resource is not to be posted. The meaning of the flag bits in the CCTECF field is exactly the reverse of the meaning of the flags in the $HASPECF field.

$WAIT: Wait Routine

The dispatcher receives control when the currently active processor executes a $WAIT macro instruction. Control is received at entry $WAIT. The code at $WAIT can also be entered as a result of the $XECBSRV macro being invoked with FUNCTION=SETUP. In this case, the processing is the same as for a $WAIT with the XECB option except that after processing the XECB, control is returned to the macro caller.

If the wait is with the XECB option, a check is made to determine if the $XECB is on a dispatcher-maintained chain. If it is on the chain and has been initialized, control is transferred to normal $WAIT processing. If it has not been initialized, then it is removed from the chain. Once removed, if the ECB was posted, control is given to HASPGOP to dispatch the PCE. If it has not been posted, processing is the same as when it was not on the chain.

If the $XECB is found to be invaildy chained (non-zero chain pointer not on the chain) or if it is being used by multiple PCEs at one time (on the chain with a different PCE address), the a DP1 $ERROR is issued and the PCE that issued the $WAIT may become disabled.

If the $XECB is not on the $XECB chain (or if it was removed from the chain), it is placed on the front of the dispatcher $XECB chain. This chain is used later by the dispatcher to determine which $XECBs need to be converted to MVS extended ECBs. If the ECB has been posted, the $XECB is not placed on the chain; control is passed to HASPGOP to dispatch the PCE.

After $XECB processing is complete, a check is made to determine if the wait is for a specific event or a general resource. If the wait is for a specific event, the PCE is queued to itself. If the wait is for a general resource, the PCE is queued to the end of the designated resource wait queue. Control is then passed to the PCE dispatch routine.

HASPDISP: PCE Dispatch Routine

The dispatcher passes control to the first eligible PCE on the $READY queue. A PCE is eligible to be dispatched if either the non-dispatchability count is zero or it is exempt from non-dispatchability. On encountering the first dispatchable PCE, the dispatcher loads its registers and the processor corresponding to that PCE is given control. The processor maintains control of the JES2 main task CPU time until it executes a $WAIT macro instruction that gives control to the dispatcher's wait routine. When all eligible PCEs on the $READY queue have been dispatched, control is passed to the resource posting routine.

HASPLOOK: Resource Posting Routine

The dispatcher looks for any accumulated resource post requests as indicated by a 1 reset in one or more resource flag bits in the $HASPECF field. If resource posting is indicated, the PCEs, queued to each resource queue corresponding to a 1 flag bit, are moved from the resource wait queue and placed on the end of the $READY queue. A check is made to see if the line manager is waiting for any posted resources. If it is required, the line manager is then posted. Control returns to the PCE dispatch routine. If no resource post requests were accumulated, control is passed to the $$POST promulgation routine.

HASPSPEC: $$POST Promulgation Routine

The JES2 main task gets its work assignments from subtasks in the JES2 address space, tasks calling upon JES2 functions in other address spaces, or asynchronous exit routines. In the case of task requests and the timer asynchronous exit, the $$POST routine within the HASCSRIC module is executed to inform the JES2 dispatcher of the occurrence of an event; other exit routines perform the same basic logic in a manner consistent with the system facilities available to the exit routine. Because the exact sequence of events is extremely important for the correct operation of the system, the logic of both sides of the interface is described.

The $$POST function operates as follows:

1. If a resource is to be posted, the CCTECF field contained in the HCCT is combined with the designated resource flag using the logical OR instruction, and the result is set back into the CCTECF field using the compare and swap instruction.
2. The $$POST work indicator is set to all ones. (The $$POST work indicator is offset from the HASP post element, pointed to by field CCTHECBA in the HCCT, by the symbolic displacement CCTPOSTW.)
3. The event control block (ECB) pointed to by field CCTHECBA in the HCCT is posted, either through MVS POST facilities or directly, through a compare and swap instruction. The $$POST function is completed, and the routine continues with other functions.
4. If the $$POST routine is entered with a specific PCE post request, the high-order byte of the designated HCCT post element is set to ones and the actions described in steps 2 and 3 are performed. If a JES2 asynchronous routine such as an I/O supervisor appendage is *simulating* a $$POST for the JES2 ASYNCH processor, the high-order byte of the $PCEASYN field in the HCT is set to ones. This applies only for simulation. Normal $$POSTs for the ASYNCH processor use the procedure described above.

The promulgation function operates as follows:

1. The dispatcher sets the high-order byte of the ECB to 0; the address of the high-order byte is found at $HASPECB in the HCT.
2. If there is no work required, control is passed to the wait for work routine; otherwise, a check is made to determine if there are any XECBs on the XECB stack. If there are XECBs on the stack, each XECB is "popped" off the stack, and the PCE associated with each XECB is marked as being dispatchable. When the stack is empty, step 3 is performed.
3. The CCTECF field is copied and reset to 0 by a compare and swap double instruction. The copy of the flags is set into the $HASPECF field for promulgation by the resource posting routine.
4. Requests for the ASYNCH processor from the $$POST routine and JES2 asynchronous routines are combined; if the ASYNCH processor is posted, CCTASYNC is set to ones and $PCEPOST is reset. requesth individual PCE post element contained within the HCCT is examined. If the high-order byte of the element is X'FF', the associated PCE is posted using the $POST macro instruction for WORK, and the byte is set to X'80'. The X'80' setting indicates that the processor has not completed the work given to it. Associated processors are expected to reset this byte to 0 when appropriate.
5. The internal reader I/O complete post element is examined. If the high-order byte of the element is X'FF', all internal reader PCEs are posted for work using $POST macro instruction and this byte is set to X'00'.
6. Processors waiting for the ABIT resource are moved to the $READY queue, and control is passed to the dispatcher $XECB routine.

HPNEXT: $XECB Initiaization Routine

The $XECBs that are placed on the $XECB chain have not been initialized as MVS extended ECBs. This processing is done after all $$POSTs have been propagated and before JES2 is MVS WAITed. Normally, if HASPSPEC has found work to do, that work is done before the XECBs are initialized. However, because this could delay $XECB posts from being propagated to the PCEs, pacing code ensures that the $XECB initialization code gets control periodically.

The $XECB initialization code scans the chain of $XECBs looking for initialized MVS extended ECBs. If the XECB has been posted, the $XECB is removed from the chain and the appropriate PCE is placed on the ready queue. If the $XECB has not been posted, it is initialized as an MVS extended ECB. Once initialized, the JES2 posting exit (HASPPXIT) gets control when the ECB is posted.

If the $XECB is initialized, then the scan is stopped. Once the scan stops, if there is any work to do, control is passed to the PCE dispatch routine. Otherwise, control is passed to HNOSPEC.

HNOSPEC: Wait for Work Routine

If the dispatcher determines that no operator-controlled work is in process (except initiator activity) and the ALL AVAILABLE FUNCTIONS COMPLETE message has not been displayed since the last activity, the message is issued and control is returned to the resource posting routine. Otherwise the MVS WAIT service routine is called, and the dispatcher waits for new work as indicated by the $$POST function described previously. When the HASP ECB ($HASPECB) is posted, control is passed to the $$POST promulgation routine to let JES2 know what to post.

$POST and $POSTR: Post Routines

When the currently active processor executes a $POST macro instruction, the dispatcher receives control at entry point $POST, if a specific event has occurred, or at $POSTR, if a general resource has been posted as available. For a specific event $POST, the PCE specified by the issuer of the macro instruction is removed from the chain in which it is waiting and is placed at the end of the $READY queue. For a general resource $POST, the routine places all PCEs waiting for that resource at the end of the $READY queue.

HASPPXIT: Extended ECB POST Exit Routine

HASPPXIT receives control between the time when an extended ECB is marked as complete and the completed execution of the post routine. HASPPXIT processing provides a notification of additional work, and queues the work for further processing.

JES2 processors define an XECB and issue a $WAIT for the MVS posting of the ECB within the XECB. HASPPXIT queues the XECB, associated with a PCE, onto a stack of XECBs pointed to by the HCT. HASPPXIT then posts the JES2 main task; the JES2 dispatcher $POSTs the associated PCE. Only one PCE is dispatched per MVS POST of the ECB in the XECB.

Work Area Management

Work area management routines maintain pools of work areas of various sizes. All of the work areas in a given pool are the same size. The number of pools and size of work areas in each is determined by the information in the table at symbol GTWKTABL, which is used to initialize the work area table that has been obtained via the $GETMAIN macro.

Work area management routines support up to 255 pools. The maximum size work area supported is 2K bytes minus the size of a prefix area (less than 20 bytes) required by the work area management routines.

The $GETWORK and $RETWORK macros provide linkage to routines named $GETWORK and $RETWORK that acquire and release work areas, respectively. $GETWORK and $RETWORK keep track of each processor's current and outstanding work area requests.

$GETWORK: Get Work Area Routine

This routine provides the smallest available work area that satisfies the size requirements of the request. If the $GETWORK macro specifies the LOC=BELOW keyword, the $GETWORK routine selects storage from below 16 megabytes in virtual storage. If the LOC=ANY keyword is specified, storage can be obtained either from below or above 16 MB. If the requested storage size is within 10 percent of one of the available pool cells, it is allocated from that pool. Otherwise, $GETWORK obtains storage with a $GETMAIN, either above or below 16MB, depending on the LOC= specification. Requests for checkpoint buffers, because they involve I/O, will be made below 16 megabytes in virtual storage. $GETWORK sets the first 4 bytes of the area returned to the EBCDIC control block name specified via the USE= operand of the $GETWORK macro. The remainder of the area is set to hexadecimal zeros.

$GETWORK issues a catastrophic error $GW1 if the size of the current request is larger than the largest supported size (2K minus the prefix). $GETWORK issues catastrophic error $GW4 if a GETMAIN fails and the $GETWORK request is not conditional.

$GETWORK searches GTWKTABL for the pool of work areas that best fits the size requested (best fit means the closest to, without being less than, the requested size). A pool is used if its size does not exceed the request by more than 10 percent. If the pool contains one or more work areas, the first is used to satisfy the request. Otherwise, 4K bytes of storage is obtained with a $GETMAIN and divided into as many work areas of the appropriate size as possible. Any storage left over is allocated to pools of smaller work areas. If no pool of appropriate size for the request is found, the requested storage is obtained with a $GETMAIN.

Storage obtained when no appropriately sized pool is found, is released with a $FREMAIN by $RETWORK. All other storage obtained by $GETWORK is not released until JES2 terminates.

$RETWORK: Return Work Area Routine

This routine releases a work area obtained by $GETWORK. If the work area is from one of the valid pools, the storage is returned to that pool. If the work area is not from any pool (that is, the area was obtained using a $GETMAIN), the storage is released with a $FREMAIN. Catastrophic error $GW2 is issued if any of several validation checks fail:

- There is an invalid pool id in the prefix of the work area.
- The pool id is inconsistent with the size of the work area being returned.

Additionally, if $DEBUG=YES, and any part of the work area beyond that requested has been used, $RETWORK issues catastrophic error $GW3.

$EXCP: Input/Output Supervisor

The JES2 input/output supervisor ($EXCP) provides the $EXCP and $EXCPVR service routines necessary to interface between all JES2 main task input/output requests and the MVS input/output supervisor. Through the use of $EXCP, the JES2 processors can achieve a certain degree of device independence for direct access devices through the track and sector conversion functions contained in the $EXCP routine. $EXCP also provides all I/O appendages required by the IOS and provides for the posting of I/O completions to each processor.

The interface between the JES2 input/output service routine and the processors that use it is the device control table (DCT), which is passed to the routine using the $EXCP macro when I/O is requested. Upon entry to $EXCP, the address of the buffer to be used is obtained from the DCT, and the input/output buffer (IOB) (appended to the front of every buffer) is initialized. The user's event wait field address is moved from the DCT to the buffer, and a pointer to the DCT is placed in the buffer. If the DCT is a direct-access type, the coded track address from the DCT is used to compute MBBCCHHR.

Note: In processing direct-access requests, the $EXCP routine tests first the track extent information in the processor control element, (PCE), then the extent information in the corresponding direct access control block (DAS). If the specified extent is invalid, the routine passes information to the I/O supervisor that causes the request to be suppressed, with the completion code in the event control block (ECB) set to X'42': direct-access extent address violation.

With the IOB complete, the routine either issues the MVS EXCP or EXCPVR macro to schedule the I/O request for execution. If WAIT=NO was specified, the routine immediately returns control to the caller.

For a WAIT=YES request, the routine issues a $WAIT for I/O. If the operation completed successfully, the routine returns control to the caller. If completion was unsuccessful, the routine executes a $IOERROR macro, which logs the error and returns control to the caller. In any case, the condition code is set by a test-under-mask instruction with mask X'7F' compared with field BUFECBCC.

Each I/O request issued by JES2 specifies an I/O appendage list that causes the JES2 appendages to be entered at various stages of I/O processing. Abnormal and normal channel-end-appendages are provided to signal the end of the I/O operation.

A program-controlled interrupt appendage is provided to signal the progress of the I/O operation. Because these appendages are entered asynchronously with JES2 operation, the buffer associated with the completed I/O is scheduled for synchronous JES2 processing by the asynchronous input/output processor. The JES2 task is posted, and immediate return is made to IOS. (The post function may be performed by IOS via requests by the channel-end appendages.)

$EXTP Service Routine

HASPEXTP invokes terminal I/O services. It is entered in response to a $EXTP macro issued by one of the JES2 processors. If the $EXTP macro was issued by a network job routing receiver or transmitter, HASPEXTP passes control to the entry routine (HASPROUT) in HASPRTAM. If the device is aborting, HASPEXTP sets the condition code to zero and returns control to the caller. Otherwise, the DCTTYPE field is tested for either a BSC or SNA remote device. A branch is taken, depending on the test results, to the service routine entry point in either HASPBSC or HASPSNA.

Data Set Services

The JES2 main task uses data set services to write additional records to a specified data set.

$DSOPEN

$DSOPEN receives control when the JES2 main task issues the $DSOPEN macro. This routine "fake" opens the JES2 job log so additional records can be written to it via $DSPUT.

On entry, $DSOPEN obtains the address of the $DSSCB work area and then initializes it with the following information: the job key from the JCT, the data set key from the job log's PDDB, and the allocation IOT address. When the record is complete, $DSOPEN initializes two HASP buffers using $BFRBLD. $EXCP in $DSPUT uses these buffers to write new records to the spool. Control is then returned to the calling routine.

If $DSOPEN encounters an error, a flag is set causing any further calls to data set services to be rejected. In addition, any resources acquired by $DSOPEN are freed at the end of $DSOPEN processing.

$DSPUT

$DSPUT receives control when the JES2 main task issues the $DSPUT macro. If $DSOPEN completes successfully, $DSPUT writes additional records to the JES2 job log.

On entry, the $DSSCB work area is passed to $DSPUT. $DSPUT then moves the record stored in the work area to the active buffer.

When the active buffer is full, $DSPUT issues a $TRACK instruction to add a new buffer to the chain. The current full buffer is written to spool. $DSPUT fills the new active buffer with the next series of records. Control is then returned to the calling routine.

If $DSPUT encounters an error, a flag is set indicating the error. This flag prevents any further writing to the JES2 job log.

$DSCLOSE

$DSCLOSE receives control when the JES2 main task issues the $DSCLOSE macro. $DSCLOSE "fake" closes the JES2 job log, previously manipulated in $DSPUT and $DSOPEN.

On entry, $DSCLOSE chains the original buffer chain to the new buffer chain created in $DSPUT. The last $DSPUT buffer is then written to spool. $DSCLOSE also updates JES2 job log's PDDB with the new record and byte counts. The job log is then closed and control is returned to the calling routine.

If an error occurs during either $DSOPEN or $DSPUT, $DSCLOSE is notified of this error and exits to the calling routine with an error return code indicating that no changes were made.

Job Queue Manager

Jobs being processed or awaiting processing by a JES2 phase are represented in an ordered queue by job queue elements (JQEs). The job queue management routines are used by the JES2 processors to insert, alter, locate, and remove job queue elements. The queue elements are maintained in priority order at all times with the highest priority element at the top of the active chain. The queue element routines are called by issuing the following macros: $QADD, $QGET, $QPUT, $QREM, $QLOC, $QJIX, and $QMOD.

The JQEs are arranged in 50 chains. One chain contains all the queue elements which are not in use; 36 of the chains contain the jobs that are in or awaiting execution in the 36 supported job classes. Two chains contain the started task (STC) and LOGON time-sharing user (TSU) jobs in or awaiting execution. Eight chains contain the jobs in processing stages: $INPUT, $XMIT, $XEQ (JCL conversion), $DUMP, $RECEIVE, $OUTPUT, $HARDCPY, and $PURGE. The remaining two queue types, $SETUP and $DUMMY, are not currently supported.

All job queue management routines use the $QSUSE macro to ensure access to the checkpoint data, which includes the shared job queue.

$QADD Service Routine

The $QADD routine is entered whenever a queue element is to be added to an active queue. The $JQFREE queue is tested for the presence of a free job queue element (JQE). If there are none free, control is returned with the condition code and register 1 set to 0. (Return is through a segment of common code in the $QGET service routine.) Otherwise, a free JQE is removed from the queue, the count of free JQEs ($JQEFREC) is updated. The information provided by the caller is placed in the JQE and the JQE address is stored in the PCE. $QADD checks for various job attributes (such as a job having held output, a job waiting for spin data sets, a job cancelled, a job queued or requeued for transmission, and a job's output requeued for transmission) in order to determine the appropriate active queue to which to add the job. The appropriate active queue is scanned, and the new element is chained in front of the first JQE currently on the queue with lower priority or at the end of the chain. A checkpoint is scheduled for the newly created JQE, the previous JQE in the chain, and the queue head elements. Modified JQEs are scheduled for checkpointing, the appropriate processors waiting for a user-submitted job (JOB) are posted ($POST), and control is returned to the caller with register 1 pointing to the JQE and a non-zero condition code. $QADD issues catastrophic error $Q02 if the new queue does not exist.

$QJIX Service Routine

All updates to the JIX are made by this routine. It is invoked by the $QJIX macro, which specifies to allocate or to deallocate a job number, to swap job numbers, to format the JIX, or to verify job numbers.

To allocate a job number for the JQE, the routine first checks to see if the job has an original job number and attempts to reassign the original job number. If the number is in use, this routine then searches the JIX within the assignable job number range for a job number that has not yet been used. If one is found, the routine places the number in the JQEJOBNO field of the JQE, and the free count of job numbers is decremented. If no job number within the range is available and a $WAIT is requested by the caller, a $WAIT is issued. If the caller did not request a $WAIT, a return code of 4 is returned, indicating that no job number is available.

To deallocate, the routine frees the job number of the JQE, increments the free job number count if the number is within the assignable job range, and if any JES2 processors are waiting for a job number, posts that the job number is available.

To swap, the JIX entry for the new JQE is cleared. Then, the job number of the old JQE is moved to the new JQE and the related JIX entry for the job is updated.

Formatting of the JIX is done only at initialization. If an entry is made after initialization, a return code of 8 is returned. If the start is a cold start, the job number table (JNT) is initialized and the job number range is set from CIRWORK. (The JNT contains all of the information related to JES2 job numbers, and it is shared among all members of the MAS and restored on any JES2 warm start.) The total number of assignable numbers is calculated. On a single-system warm start, the job number range is set and the free count is adjusted. This is because the job number range can be changed in a single system start. On an all-systems warm start, the range is set and the JIX is cleared.

The verify routine ensures that the number is not used. If it is, a return code of 4 is set. If it is not, the JIX entry is set with the relative job offset, and the free count of job numbers is decremented. After the requested function is completed, the JIX and JNT are checkpointed as needed.

$QGET Service Routine

The $QGET routine is entered to acquire control of a job queue element (JQE) in the active queue.

It also calls work selection services to select JQEs to be processed by the offload job transmitter. If the JQE is found, control is returned with a zero return code. If not, a return code of 4 is returned. If the system is not draining, exit point QVALID (for exit 14) is taken to allow an installation exit routine to select a JQE. When the installation exit routine returns, normal queue scanning for a JQE continues or is bypassed because the installation exit routine found a JQE or because no JQEs were found. When no JQE is found, control is returned to the caller. If the installation exit found a JQE, normal queue scanning is bypassed and the returned JQE is processed. If normal queue scanning is to continue, the appropriate queue is scanned for an element that: is not held, is not already acquired by a previous request to the job queue service routines, has system affinity to the selecting system, has all it's spools online and has independent mode set in agreement with the current mode of the selecting system. If $XMIT is the requested queue, affinity is ignored, and the line device control table (DCT) is checked to ensure the execution node is reachable via the connection. A NODETBL address of zero indicates that a request has been made by the network job route transmitter for a /*XMIT job, which has been rerouted for local execution. If an entry is found, the element is flagged busy with the coded system ID of the selecting system, and control is returned to the caller with register 1 pointing to a parameter list containing the JQE and other information. If no element is found, control is returned with a return code of 4 and a condition code of 0. If the operator has entered a $P or $PJES2 command without a succeeding $S command, the scan will not find a JQE.

Note: The above description is true for every device issuing a QGET except offload transmitters.

$QPUT Service Routine

The $QPUT routine is entered to release control of an acquired job queue element (JQE). If the queue type is unchanged and the job is not being cancelled or purged, then the busy bits in the element are turned off, a checkpoint is scheduled for the JQE, the appropriate processors waiting for a user-submitted job are posted ($POST), and control is returned to the caller with register 1 pointing to the JQE. Otherwise, the JQE is removed from its current chain, and a checkpoint request is made for any modified element. Then the busy bits are turned off and the JQE is priority-chained to its new queue, after which processing continues as above. The removal and queuing process is performed if the queue type is $XMIT and the execution node is equal to the value of $OWNNODE. In this case, the queue type is forced to $XEQ.

If the JQE is not found on the proper queue, the routine exits to the catastrophic error routine with code $Q01. An error exit with code $Q02 is taken if the new queue type does not exist.

If the operator or time-sharing user has successfully flagged a JQE for cancellation, the job queue manager alters the queue type for the $QADD, $QPUT, and $QMOD functions as follows:

- If the job is being queued for execution or conversion, the queue type is altered to $OUTPUT.
- If the job is flagged for $PURGE, the queue type is altered to $HARDCPY if job output elements or held data sets exist, or to $PURGE if no job output elements or held data sets exist.

$QREM Service Routine

The $QREM routine is entered to remove a job queue element (JQE) from its active queue and place it on the free queue. $QREM sets to 0 the relative pointer in the job queue index (JIX) and schedules a checkpoint of the JIX entry. Then it scans the appropriate queue to verify that the JQE, pointed to by register 1, is on the active queue. If the JQE's queue type is invalid, $QREM issues a $Q02 catastrophic error code. If the JQE queue type is valid but the JQE was not on the queue, $QREM issues a $Q01 catastrophic error. The JQE is removed from the chain, modified, and is scheduled for checkpointing. The JQE is chained to the $JQFREE queue in order of ascending JQE addresses, and the free JQE count ($JQEFREC) is updated. A checkpoint is scheduled for the queue heads, the JQE, and the JQE behind which the element was queued. If the free queue is empty, all processors waiting for a user-submitted job (JOB) are posted ($POST). Control is returned to the caller.

$QLOC Service Routine

The $QLOC routine obtains the JQE address when the job number is known. If the JQE is found, control is returned with register 1 pointing to the JQE and with a non-zero condition code; otherwise, register 1 and the condition code are set to 0.

An entry is now located directly through the job queue index (JIX). This routine looks at the JIX entry for the job number passed and if the entry is zero the job number is not in use.

The JIX is used for accessing any specific JQE quickly by using its associated job number. The JIX is a table of 2-byte entries, each containing an offset that is used as an index into the job queue and other structures. This index relates a job's position to other jobs in the job queue (see Figure 3-2).

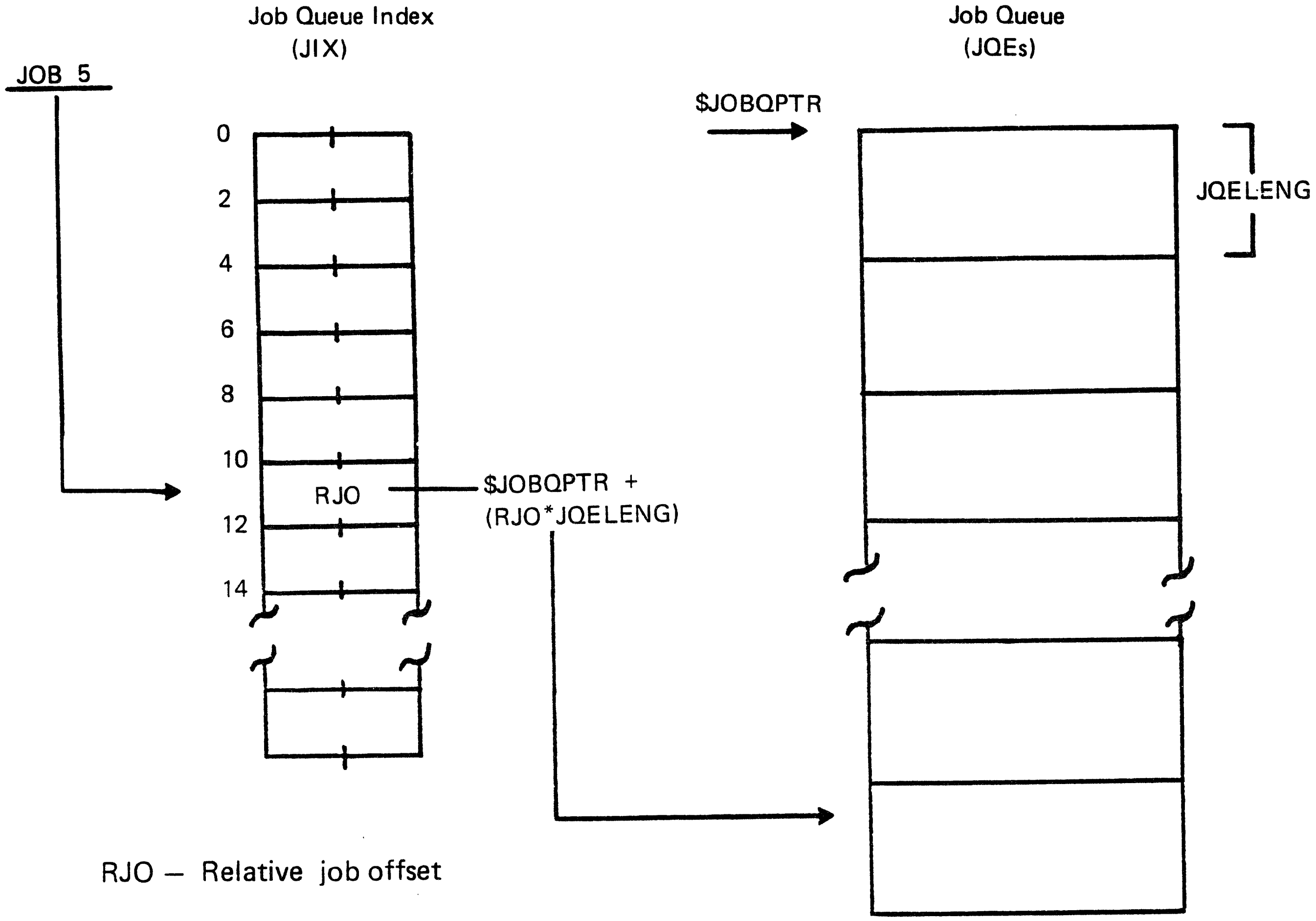


Figure 3-2. Relationships Between Job Queue Structures

$QMOD Service Routine

The $QMOD routine is entered to modify the position and/or priority of a job queue element (JQE) on the active queue. If access has not been reserved for checkpoint data on the primary spool volume or if the JQE is not on the active queue, the routine exits to the catastrophic error routine with code $Q03. (The caller of this routine must have issued a $QSUSE macro instruction and have had no intervening $WAIT before modifying the JQE and entering this routine.) The current queue is scanned to locate the JQE, the JQE is removed from the queue, and the JQE preceding the removed JQE is scheduled for checkpointing. If the new queue type is

$XEQ and the execution node does not equal the value of $OWNNODE, the queue type is set to $XMIT and the hold for duplicate job name is reset. The $QADD routine is entered to add the removed JQE back into the queue. The entry into $QADD scans the new queue, inserting the JQE and checkpointing the modified JQE entries or queue heads.

$QACT Service Routine

The $QACT routine is called to mark active and checkpoint a JQE. If the JQE is being scheduled for execution or conversion, an extension area is obtained to keep information that is referenced frequently during execution (such as the number of track groups a job is using). The extension area is returned by a subroutine call to $QDACT.

$CKPT Service Routine

The $CKPT macro, and this service routine, are used to notify the JES2 checkpoint processor of changes in control blocks in the JES2 checkpoint area. The POST operand of the macro specifies whether or not to post the checkpoint processor. If the processor is posted, a write is initiated. If there is no post, the write is not done until there is another request for checkpoint processing or the checkpoint ownership period is over. The default on the macro is not to request a post of the processor. This way, checkpoint requests can be processed in batches, rather than one by one.

To ensure that the checkpoint processor is posted enough times, this routine specifically posts the processor through a timer queue element (TQE) in the checkpoint PCE work area.

The $CKPT routine is entered when a processor has altered a field of a checkpoint element. A $QSUSE macro must be in effect when $CKPT is entered. The checkpoint of the field is scheduled, and control is returned to the caller.

A $QSUSE macro with TYPE=TEST specified is issued to ensure that the caller has control of the checkpoint data set. If the caller does not have control of the checkpoint data set, $CKPT issues a $Q03 catastrophic error.

The checkpoint information table (KIT) contains all information that is needed to schedule a queue element for checkpointing. $CKPT locates the KIT by using the offset passed in register 0.

$CKPT then checks the element address for validity. The element must be contained within the contiguous storage of the table. If the element does not meet these validity checks, $CKPT issues a $Q04 ($ERROR) catastrophic error. If an address of zero is passed, only the header record of the table is checkpointed.

If change log support is not in effect, this routine marks the control bytes (CTLBs and CTLBXs) for the page of the changed element. For change log support, this routine builds a list (CAL) containing the address, length, and CTENT ID of each element. If there is no space left in the $CAL, subsequent $CALs will be $GETMAINed. If a $GETMAIN fails, the control bytes for the elements are marked.

This routine returns to the caller with the token for the next checkpoint write in register 0.

$CHECK Service Routine

This service routine checks for completion of a checkpoint write using the checkpoint token parameter to determine whether to $WAIT for a checkpoint which has been scheduled but not yet completed, $WAIT for a checkpoint not yet scheduled, or return control immediately to the caller because the checkpoint is complete.

$PGSRVC Service Routine

This routine switches to key 0, and issues the correct form of the MVS PGSER macro based on flag bits set by the $PGSRVC macro. If the $PGSRVC macro was issued with a PSL specified instead of a length, the address passed points to a page service list (PSL); otherwise, the address passed is the address of the storage to be serviced.

$QSUSE Service Routine

The $QSUSE service routine is used by all of the job queue management routines (and by other routines and processors) to control access to the shared job queues. It is entered from a $QSUSE macro when the macro determines that the shared queues are not currently owned.

The routine issues $POST to post the checkpoint processor. Then specific $POSTs are inhibited for the processor control element (PCE) for which there is access to the job queue. The routine issues the $WAIT macro instruction to wait for the checkpoint resource. When that resource is available, the service routine returns control to the caller.

Unit Allocation and Deallocation Service Routines

The following service routines are used to allocate and deallocate devices.

$GETUNIT: Unit Allocation Service Routine

The unit allocation service routine tests the status field (DCTSTAT) of the device control table (DCT) representing the device to be allocated. If the table is marked as in use, or if a drain, hold, or pause is in effect for the device, the device is unavailable for allocation. The routine returns to the caller with the condition code set to 0, indicating that allocation did not occur. If a device is available, $GETUNIT issues a $DCBDYN macro to attach a DCB/DEB. If the return code from $DCBDYN is 4, a GETMAIN error has occurred and $DCBDYN has issued message $HASP184. $GETUNIT sets a zero condition code, indicating the device is not available. If there is no error, the routine marks the DCT as in use and returns to the caller with a non-zero condition code and the address of the DCT in register 1.

$FREUNIT: Unit Deallocation Service Routine

The unit deallocation service routine waits for buffers ($WAIT BUF) until the count of outstanding buffers associated with the specified DCT is 0; this ensures that all outstanding I/O has completed. The routine then turns off the DCTINUSE indicator and (for RJE and NJE devices only) ensures that the DCTEOF, DCTPOST, and DCTABORT indicators are also off. If the DCTDRAIN indicator is off, indicating that a stop device ($P) command is **not** being processed for the device, deallocation is completed and the routine returns to the caller.

If a stop command has been received for the device, the routine issues $DCBDYN macro to detach any DCB/DEBs. Then, if the device is not an FSS device, it issues a $ALLOC to unallocate the device. The routine calls the $DYN service routine, dynamically deallocating the unit control block (UCB) representing the device, and on return issues $WTO to schedule message $HASP097 (devname IS DRAINED) before returning to the caller.

Interval Timer Manager

The interval timer manager provides an interface between the JES2 main task processors and the standard MVS timer facilities. It notifies the processor of the completion of intervals and the time remaining with optional cancellation. It allows multiple processor intervals to be active concurrently while maintaining only one MVS timer element through the STIMER macro.

The user of the JES2 interval timer manager must provide a unique requestueue element (TQE) for each interval that is to be simultaneously active. To begin an interval, the user processor executes a $STIMER macro using the TQE as a parameter. During the interval, the TQE is chained to other active TQEs and the interval values are adjusted by the interval timer manager. When the interval expires, the TQE is removed from the active queue and the processor is posted ($POST WORK). When an active interval is to be terminated or the remaining time is desired, the user processor executes a $TTIMER macro. If the interval is terminated by $TTIMER, the TQE is removed from the active queue.

$STIMER: Set Interval Timer Routine

When a $STIMER macro is executed, the interval timer manager first ensures that the user's TQE is removed from the active chain. It then ensures that the user's TQE time interval is in units of 10 milliseconds. The IADJUST subroutine is called to adjust currently active TQEs for the change in time from the last adjustment, dequeuing expired TQEs and posting ($POST) associated processors for work. The new TQE is then inserted into the queue in front of TQEs with longer remaining intervals. The ISETINT subroutine is called to conditionally issue the MVS STIMER macro instruction. The STIMER macro is executed if the first TQE is not the TQE currently represented by the previously executed STIMER macro, or if there is no MVS timer currently active for the JES2 main task. Control is returned to the user.

ITIMEUP: Asynchronous Timer Exit Routine

When MVS recognizes the end of a JES2 main task time interval, control is given to the asynchronous timer exit routine. This routine records the fact that a timer interval is not active and executes the $$POST macro to activate the time processor.

When activated by the asynchronous timer exit routine, the timer processor calls the IADJUST subroutine to adjust currently active timer queue elements (TQEs) for the change in time from the last adjustment, dequeuing expired TQEs and posting ($$POST) associated processors for work. The ISETINT subroutine is executed to conditionally execute the MVS STIMER macro, and the processor waits for work.

$TTIMER: Test Interval Timer Routine

When a $TTIMER macro is executed, the interval timer manager uses the IADJUST subroutine to adjust currently active TQEs for the change in time from the last adjustment, dequeuing expired TQEs and posting ($POST) associated processors for work. The active queue is scanned to ensure that the TQE is on the queue and to locate the previous TQE. If the TQE is not found, control is returned to the user with register 0 equal to 0. The remaining time is recorded and if the request is to cancel the interval, the TQE is removed from the active queue. The ISETINT subroutine is called to conditionally issue the MVS STIMER macro.

The recorded time is converted and rounded to the nearest second, and control is returned to the user.

$TIMER Processor

This processor resets the MVS interval timer after a timer interruption has occurred. This processor calls the IADJUST and ISETINT subroutines in the $STIMER/$TTIMER interval timer supervisor, which causes the expired TQEs to be posted and the time interval specified in the first TQE in the TQE chain to be set in the MVS interval timer. The processor then waits for another timer interruption to occur. When the next timer interruption is processed, the asynchronous exit routine posts this processor, and the above procedure is repeated.

$STCK

$STCK services the $STCK macro instruction to obtain the TOD clock setting. If the correct TOD is not obtained, $STCK attempts to get the TOD with an MVS TIME macro instruction. If it cannot, $ERROR is called to issue catastrophic error message, "$C01 UNABLE TO OBTAIN VALID CLOCK TIME", and to terminate JES2.

DATECONV: HASP Day-to-date Conversion Routine

The MVS TIME macro returns the Julian date in packed decimal format in register 1. This routine takes a date in this form and converts it to numerical month, day, and year, and returns it in packed decimal format in register 1.

SMF Buffer Queue Manager

The JES2 SMF service routines obtain JES2 SMF buffers from a free queue, place allocated JES2 SMF buffers on a busy queue, and post the JES2 subtask.

$GETSMFB: Get SMF Buffer Service Routine

$GETSMFB is used to obtain a JES2 SMF buffer from the $SMFFREE cell in the HASP communications table (HCT). If no buffers are available and WAIT=NO was specified, the routine returns control to the caller. If no buffers are available and WAIT=YES was specified, the routine waits ($WAIT) for SMF and then loops back to try to obtain a buffer again. Once a buffer is obtained, the count of free SMF buffers is updated. The address of the newly-obtained SMF buffers is placed in register 1, the buffer contents is cleared to zero, and control is returned to the caller.

$QUESMFB: Queue SMF Buffer Service Routine

Prior to queueing the SMF buffer, exit point SMFEXIT (for exit 21) is taken to allow an installation exit routine to finalize the contents of the SMF buffer that is to be queued. When the exit routine returns, the buffer is queued or queueing is bypassed, depending on the return code from the exit routine. If queueing is bypassed, the buffer is freed. Control is then returned to the caller.

The $QUESMFB routine places a JES2 SMF buffer on the queue of busy JES2 SMF buffers. The $SMFBUSY cell in the HCT points to the busy queue. The JES2 subtask is then posted for work, and control is returned to the caller.

Save Area Management Routines

The save area management routines obtain save areas for the caller's registers, return save areas, restore registers and return the current save area, and convert symbolic destinations to binary route codes.

$GETSAVE: Get Save Area and Save Registers Service Routine

The $GETSAVE service routine, called through the $SAVE macro instruction, obtains a save area and saves in it the caller's registers. If the save area in the processor control element (PCE) is not in use, it is claimed as the save area. Otherwise, a save area is dequeued last-in-first-out (LIFO) from a pool of JES2-managed save areas headed by the HASP communications table (HCT) field $SAVAREA and added to the end of the PCE save area chain. If a free save area does not exist, a GETMAIN macro is issued to obtain a 4K block of storage in subpool 0. This storage block is cleared and formatted into save areas chained together via the save area ($PCE mapping macro) field PCESVNXT, and added to the queue of free save areas ($SAVAREA). A GETMAIN failure causes the JES2 catastrophic error code $S01 to be issued. Should such an abend occur, one or more of the JES2 buffer pools should be reduced in size to allow for additional GETMAIN storage in the JES2 address space.

$RETSAVE: Return JES2 Save Area Service Routine

The $RETSAVE service routine, called via the $RETSAVE macro instruction, returns the most recently obtained JES2 save area associated with the caller's PCE. If the most recent save area is located in the PCE, it is marked available. Otherwise, the save area is unchained from the PCE save area chain and is made available to the pool of free save areas headed by the HCT field $SAVAREA. Prior to freeing the save area, a $ESTAE macro with CANCEL specified is issued to free any PRE associated with it. If the save area chain is invalid, $RETSAVE issues a $S02 catastrophic error.

$RETURN: Return to Caller Service Routine

The $RETURN service routine, called via the $RETURN macro instruction, restores the registers in the current save area (that is, the most recently obtained save area for the caller's PCE), returns the current save area to the save area pool via $RETSAVE, and returns to the location indicated in restored register 14. The return address is augmented by the return code, if any, in register 15 on entry to this routine. Note that the condition code is preserved throughout the execution of this routine.

$PGRLSER/$PGFREER/$PGFIXR: Virtual Page Service Routines

The $PGRLSER, $PGFREER, and $PGFIXR routines provide the page-release, page-free, and page-fix services respectively. The service routines use a common page services routine (label PGRTN). The routine obtains the address of the MVS page services routine from the communications vector table (CVT) and uses a branch entry to that routine, using the MODESET and SETLOCK macro instructions to set and reset the protection key and local lock as required. $PGFIXR additionally ensures that the page has been fixed by actually referencing the page before returning to the $PGSRVC caller.

Buffer Pool Management

The buffer management routines allocate the JES2 dynamic storage area (buffer pool). Fixed-size buffers in this area are allocated and deallocated to JES2 processors and routines using the $GETBUF and $FREEBUF macros.

The $GETBUF macro permits the caller to specify one of five buffer types. According to the buffer type specified, a buffer is obtained from one of four subpools within the JES2 buffer pool, and its BUFTYPE field is initialized to indicate the buffer's characteristics. The structure of the BUFTYPE field is:

```
R F IPM D NN
```

where:

R is 1 if the buffer is to be used for the transfer of data between JES2 and a remote terminal, or 0 if the buffer is for local use

F is 1 if the buffer is to be page-fixed (FIX=YES specified)

IPM is 100 if the buffer prefix (which begins with the first byte of the buffer) is to be an input/output buffer, (IOB), or 010 if the prefix is to contain a request parameter list (RPL), or 001 if this buffer is part of a chain of buffers (MULTIPLE was specified in the $GETBUF macro); multiple buffers are chained through the BUFCHAIN field

D is 1 if the prefix is to contain a data event control block (DECB)

NN is an index value, used to access the buffer pool map with which this buffer is associated

At location $BPMTABL in the HASP communications table (HCT), there is a series of four pointers, one corresponding to each of the JES2 buffer types. (The six buffer types that can be specified require only four types of buffer pool map because both BSC and SNA buffers come from the common teleprocessing pool, and both SPXFR and PAGE buffers come from the PAGE pool.) The index value in the BUFTYPE field, when added to the origin address $BPMTABL, points to one of a series of four pointers in the HCT. The HCT pointer, in turn, points to the actual buffer pool map.

The correspondence between the buffer type specified by the issuer of $GETBUF and the BUFTYPE field contained in the buffer obtained (ignoring the page-fix and multiple-buffer specification) is:

TYPE=HASP The buffer is for local use and its prefix is an IOB. The buffer pool map is BPMTHASP.

TYPE=BSC The buffer is for remote use ad its prefix is an IOB. The buffer pool map is BPMTTP.

TYPE=VTAM The buffer is for remote use and its prefix is an RPL. The buffer pool map is BPMTTP.

TYPE=PAGE The buffer is for local use and its prefix is an IOB. The buffer pool map is BPMTPAGE.

TYPE=PP The buffer is for local use and its prefix is an IOB. The buffer pool map is BPMTPP.

TYPE=SPXFR The buffer is for local use and its prefix is a DECB. The buffer pool map is BPMTPAGE.

$GETBUFR: $GETBUF Service Routine

The $GETBUF service routine selects one or more buffers from the buffer pool, causes the buffers to be initialized, and allocates the buffers to the calling processor or routine. The buffer pool is described by a bit map that is subdivided into a separate map for each kind of buffer: regular JES2 buffers, TP buffers, page buffers, and print/punch buffers.

The service routine selects the appropriate map based on the buffer type specified by the caller, and determines whether enough buffers are available to satisfy the request. (If the buffer count, BPMBUFCT, indicates that a buffer is available but the bit map does not indicate availability of a specific buffer, $ERROR is issued with catastrophic error code $B02.) If buffers are available, the routine allocates the buffers to the caller. $GETBUFR updates BPMBUFCT depending on whether the request is to teleprocessing buffers or local buffers. $GETBUFR issues the $BFRBLD macro for each buffer to cause the buffer to be initialized; the bit in the bit map representing the buffer is turned off, to show that the buffer is allocated. If the caller requested that the buffers be page-fixed, the routine also issues the $PGSRVC macro to fix each buffer in main storage. When all required buffers have been allocated, initialized, and (if necessary) page-fixed, the routine sets a non-zero condition code and returns to the caller with the address of the first allocated buffer in register 1.

If enough buffers are not available and the caller has specified WAIT=YES, the service routine waits ($WAIT BUF, a wait for the availability of a buffer as general resource). At each subsequent buffer resource post, the routine regains control and branches to the beginning of the service routine; this process continues until the $GETBUF request can be satisfied.

If the buffer request cannot be satisfied immediately and the caller did not elect to wait, the routine sets the condition code to 0, sets register 1 to 0 indicating that no buffer was allocated, and returns to the caller.

$FREEBFR: $FREEBUF Service Routine

The $FREEBUF routine first ensures that the buffer address is valid. (If not, $ERROR is issued with catastrophic error code $B01.) The JES2 dispatcher's event control field is posted to show that a buffer is available. Then the corresponding bit in the appropriate buffer pool bit map is set on. If the buffer being freed was previously page-fixed, $PGSRVC is issued to free the buffer. If MULTIPLE was specified, the process is repeated (from the point of buffer address validity checking) until all buffers in the chain have been processed. The buffer count (BPMBUFCT) is updated for each buffer returned. Whether or not MULTIPLE is specified, if all buffers in the page in which the buffer exists are free, that page is released, and control is returned to the caller.

$BFRBLDR: Buffer Build Routine

The buffer build routine is invoked by the $BFRBLD macro instruction. Based upon buffer type specified by the caller, this routine builds an IOB, a request parameter list (RPL), or a data event control block (DECB) at the beginning of the buffer.

GETMAIN/FREEMAIN Services

To speed JES2 initialization, MVS GETMAIN and FREEMAIN macros are not used by the HASPINIT module. The $GFMAIN routine provides a branch-entry interface to GETMAIN/FREEMAIN services.

$GFMAIN: Branch Entry GETMAIN/FREEMAIN Service Routine

When a $GETMAIN macro and subsequently a $FREMAIN macro are issued, the $GFMAIN routine is entered. It processes GETMAIN/FREEMAIN requests in the same way that MVS does. The only difference is that corresponding SVCs need not be issued; $GFMAIN branches directly to the MVS service routine. The JES2 main task need not wait for an SVC.

The $GFMAIN routine obtains storage above or below 16 megabytes in virtual storage depending on whether the $GETMAIN macro specified LOC=BELOW or LOC=ANY. LOC=BELOW restricts obtaining storage to below 16 megabytes. LOC=ANY means storage can be obtained either above or below 16 megabytes in virtual storage.

$GETLOKR, $FRELOKR, $GETJLOK and $FREJLOK: Lock Control Services

Four subroutines located in HASPNUC provide an interface for JES2 main task processors to obtain and release the operating system's cross memory services (CMS) lock or the JES2 job lock. The $GETLOK and $FRELOK macros obtain access to these routines and perform the following functions:

- The $GETLOKR routine saves registers in the $CSAVREG area of the HASP communications table, (HCT), sets the program status word (PSW) key to 0, obtains the LOCAL and CMS locks, resets the PSW key to 1, restores registers, and returns.
- The $FRELOKR routine saves registers in the $CSAVREG area of the HCT, sets the PSW key to 0, releases the CMS and LOCAL locks, resets the PSW key to 1, restores registers, and returns.
- The $GETJLOK routine obtains access to the job JQE, marks the lock held, and then returns to the calling routine. The $GETJLOK routine issues a $WAIT macro for the lock.
- The $FREJLOK routine obtains access to the jobs JQE, marks the lock free and then returns to the calling routine. The $FREJLOK routine issues a $POST macro to indicate that the lock is available.

HASPATTN: Unsolicited Device End Interrupt Handler

HASPATTN runs as an SRB; the SRB is scheduled when an unsolicited device end interrupt occurs on a JES2-owned unit record device. HASPATTN resets the DCTHOLD and DCTPAUSE bits in the DCT associated with the unit record device and simulates a $$POST for the unit so that processing for the device can continue.

I/O Supervisor Appendages

Abnormal and channel-end appendages, as well as a program-controlled interruption (PCI) appendage, are provided to record the I/O interruption associated with a particular $EXCP operation and to trigger further processing in response to those interruptions. (I/O appendages for $EXTP requests are provided in HASPBSC and HASPSNA.) When an interruption occurs, the address of the associated JES2 I/O buffer is placed in one of two queues: $ASYNCQ for channel-end interruptions, or $ASYPCIQ for program-controlled interruptions. The asynchronous event processor $ASYNC, processes those queues, directing each queued buffer for further processing to the JES2 processor that initiated the corresponding $EXCP operation.

EABCHEND: Abnormal-end Appendage Routine

The EABCHEND routine first determines whether abnormal-end processing is required by testing the bypass appendage indicator in IOBFLAG1 and testing the event completion code in IOBECBCC for a non-error indication. If no error has occurred or if the bypass indicator is on, control is returned to the input/output supervisor (IOS). Otherwise, error indications in the input/output buffer (IOB) and data control block (DCB) are reset and control passes to the common channel-end processing routine, EABCONT, if the device is other than an IBM 3211 Printer or 3800 Printing Subsystem.

When a paper jam has occurred on a 3800, the fuser page ID from UCBPGID is set in DCTLDPID, and the DCTCKJAM indicator is set for later processing by the print/punch processor.

If the device is a 3800 and the operation completed with the unit-exception and cancel-key indicators set, a forward-space data set command is simulated in the DCT and control passes to EABCONT.

If the device is a 3800 printer and an intervention required condition has occurred, the resume CSW is modified so that the channel program will immediately terminate upon restart. This is to allow processing of any commands issued during the intervention required condition.

ECHANEND: Normal-End Appendage Routine

The ECHANEND routine determines whether the indicators in IOBFLAG1 are set for a bypass appendage or an error routine in control; it returns control to IOS if either is on. If the device is a local printer or punch, the PCIBUSY indicator in the print/punch processor's current program-controlled interruption element (PCIE) is reset to show that a break in the channel program has occurred.

EABCONT: Common Channel-End Processing Routine

The EABCONT routine is used by both the abnormal-end appendage and the normal-end appendage to post the JES2 main task to recognize the interruption. First, the I/O buffer is queued onto $ASYNCQ (using BUFCHAIN as the chain field) and a $$POST of the $ASYNC processor and the JES2 dispatcher is simulated by setting to X'FF' the 1-byte CCTPOSTW work indicator and setting the ASYNC post indicator. Next, if JES2 is in a wait state, JES2 is posted by IOS. If JES2 is not in a wait state, JES2 is notified through direct modification of the ECB via a compare and swap instruction.

EPCI: Program-Controlled Interruption Appendage

EPCI signals the progress of an ongoing I/O operation. The address of the program-controlled interruption element (PCIE) associated with the interruption is found at PPBPCIE in the I/O buffer. The PCIBUSY indicator in field PCISGNAL is reset to show that the channel command word (CCW) chain up to and including this PCIE has already executed and is no longer part of the executing channel program. However, if this indicator was already off at entry to EPCI (possibly because of multiple entries), control is returned immediately to IOS. Next, the PCIBUFAD field is set to the address of the I/O buffer, and the PCIE is queued to $ASYPCIQ (using the PCICHAIN field as the chain field). If the command-chaining indicator is set in the PCIE, indicating that the channel program is not yet complete, then IOBSTART is set to the address of the next section of the CCW chain (found at PPBCCWNX), and the PCI indication is reset in the new chain to prevent multiple entries to the EPCI because of CCW retries (such as intervention required). The channel program progress indicator in the unit control block (UCB) is reset to notify the MVS missing interrupt handler that the channel program has progressed despite the fact that no channel-end interruption has occurred. Finally, the JES2 main task is notified that the interruption has occurred, by a method similar to that used by the channel-end appendage routines, with the exception that, if necessary, a cross-memory post is used to post JES2.

$POSTEX: Spool Offload Buffer Completion Routine

The post exit ($POSTEX) routine is given control by MVS each time a spool offload I/O operation is marked complete.

$POSTEX queues the completed buffer to the $XFRBEND queue by using compare and swap logic. Finally, $POSTEX simulates a post of the spool transfer I/O manager.

$ASYNC: Asynchronous Input/Output Processor

Because the status of all JES2 I/O operations are signalled asynchronously with JES2 operation through the input/output supervisor (IOS) channel-end appendages and a program-controlled interruption (PCI) appendage, these interruptions must be queued by the appendage until all JES2 main task processors can be synchronized to receive the notification. The primary purpose of the asynchronous input/output processor is to notify all other processors of their I/O completions or program-controlled interruptions that were indicated by the MVS I/O supervisor when the interruption occurred. The buffer (and respective input/output buffers) associated with the interrupts are chained by the JES2 I/O appendages for later processing by $ASYNC. In addition to the post of the JES2 task by IOS and any I/O completion, the appendages also simulate posting ($$POST) of the asynchronous input/output processor to initiate its processing when the JES2 main task receives control.

The processor also provides a miscellaneous service that should be run under control of a JES2 main task processor but not be associated with any other processor. This service is:

- Dequeuing console message buffers (CMBs) from the $DOMQUEA queue and either queuing the CMB to the $DOMQUE or issuing an operating system delete operator message (DOM) SVC to delete the message display and free the CMB using the $FRECMB macro instruction.

If a CMB is on the $DOMQUEA queue, the $HASPWTO task routine has queued a CMB originally scheduled with the $DOMACT flag on; $ASYNC removes the CMB from the queue. If the $DOMACT flag is still on, the CMD is queued to the $DOMQUE in order of ascending DOM identification numbers which were set in the CMB by the $HASPWTO routine upon return from the WTO macro instruction execution. If the $DOMACT flag is not on, another processor has executed the $DOM macro instruction earlier than expected; therefore, $ASYNC frees the CMB and deletes the message using the $FRECMB and DOM macro instructions. Processing continues with the next function.

If a program-controlled interruption element (PCIE) is queued to $ASYPCIQ, it was placed there by the JES2 program-controlled interruption (PCI) appendage, routine EPCI in HASPNUC, as a result of an interrupt for the associated $EXCP operator. The buffer address is extracted from the PCIBUFAD field of the PCIE, and the processor associated with the interrupt (as indicated by BUFEWF) is posted ($POST IO).

If a buffer is queued to $ASYNCQ, it was placed there by the JES2 task channel-end appendage routine as a result of I/O completion for the associated $EXCP operator. The buffer is removed from the queue and the JES2 master I/O count is reduced by 1. If the count becomes negative, control is passed to the catastrophic error routine with code $E01. If RJE lines have been generated, the buffer could be a teleprocessing buffer; if it is, the buffer is placed on the $RJECHEQ queue, and the line manager is posted for work. For normal buffers, the associated device control table (DCT) active count is reduced by 1. If the count becomes negative, control is passed to the catastrophic error routine with code $A01. Depending on the value in the BUFEWF field, $ASYNC proceeds as follows:

- BUFEWF=0

  If the completion is in error, issue message using the $IOERROR macro instruction; free the buffer using the $FREEBUF macro instruction.

- BUFEWF>0

  Post ($POST) for I/O the processor addressed by the BUFEWF field.

- BUFEWF<0

  Control is given to the processor exit routine by a branch and link instruction. Registers are set are as follows:

R1	=	buffer address
R13	=	processor control element (PCE) address of the processor
R14	=	return to $ASYNC
R15	=	entry point address

The exit routine uses a $SAVE macro on entry and returns to $SASYNC via a $RETURN macro.

After taking the appropriate action, the processor goes to the beginning for more work.

If on an examination of the $ASYNCQ queue it is determined that there are no buffers to process, the processor waits ($WAIT WORK) for work and, when posted, goes to the beginning of the routine for work.

$JCTIOR: Job Control Table I/O Routine

This routine provides the JCT I/O installation exit point, JCTEXIT (for exit 7), after a successful read of a JCT from spool or before the write of a JCT to spool. This routine issues the $EXCP macro that performs the read or write of the JCT.

$DYN: Dynamic Device Allocation/Deallocation Routine

The $DYN routine uses SVC 99 to dynamically allocate a device to JES2 or to dynamically deallocate a JES2 device. The DCTUNAL indicator in field DCTSTAT is used by $DYN to determine whether the function is to allocate or deallocate. Upon entry to $DYN, register 1 points to a device control table (DCT). The DCTUNAL bit is changed if $DYN successfully allocates/deallocates a device, and the condition code is set to 1 for a normal return. If the dynamic allocation/deallocation is unsuccessful or if the DEBSUCBB field is 0 and the request is to deallocate, the DCTUNAL bit is not changed, and the condition code is set to 0 when control is returned to the caller. After allocating/deallocating a device successfully, $DYN checks the DCTATTN indicator to determine whether to set or reset the unit control block (UCB) attention index field and, if the device is an IBM 3211 Printer or IBM 3800 Printing Subsystem, determines whether to issue a GETMAIN/FREEMAIN request for a unit control block (UCB) log area before returning to the caller.

$JESEFF: JES2 Main Task Exit Effector

This is the exit effector for the JES2 main task. It is called when the $EXIT macro specifies (or defaults to) ENVIRON=JES2. $JESEFF passes control to the user exit routines based on the exit id.

This routine executes in 31-bit addressing mode. If the caller executes in 24-bit mode, the exit effector returns control in 24-bit mode.

Tracing of pre-exit calls produces a trace record made up of the exit number, exit name, system environment (that is: the indication that this trace is prior to the call of the first installation-exit routine), and calling register 0-15.

The exit information table (XIT) is found based on the exit id and the address of the installation-exit routine in the exit routine table (XRT).

The user exit routine is then called based upon information in the XIT. On return, the return code is saved and checked to see if either there are more installation-exit routines to call from this exit or if the processing is complete.

If there are more installation-exit routines, they are called until either an installation-exit routine returns a non-zero return code or until there are no more installation exit routines to call. Then the post-exit tracing is complete. Post-exit tracing produces a trace record made up of the exit number, exit name, system environment (that is, the indication that a trace is following return from the last routine), the calling registers 15, 0, and 1, and the name of last routine called.

A catastrophic error condition of $U01 results when the return code from the user exit routine is greater than the setting of MAXRC on the $EXIT macro.

UCB Services

The following routines process the UCB chain.

$GETUCBS: Obtain a UCB Address

The $GETUCBS routine uses the IOS UCB scan routine (whose address is in the CVT, CVTUCBSC) to obtain a UCB associated with an input device class. The UCB address is obtained in the UPLUCBAD field of the UCB parameter list supplied by the caller as input. A catastrophic error $UO3 is issued if $GETUCBS cannot acquire storage (via MVS GETMAIN) for the UCB parameter list.

$FREUCBS: Free Storage for the UCB Parameter List

The $FREUCBS routine issues a MVS FREEMAIN to free the storage used for the UCB parameter list (that was acquired by the $GETUCBS routine).

$SUBIT: General Purpose Subtask Work Queueing Service Routine

$SUBIT issues a $ESTAE to establish its recovery environment. If no general purpose subtasks were attached during initialization, processing continues only if the $SUBIT request was issued unconditionally. In this case, required parameters are set and the routine to be subtasked is $CALLed directly. When control returns, the contents of registers 0 and 1 are saved in the SQD. The fact that the call was made directly is also recorded in the SQD. $SUBIT marks the ECB as posted and returns to its caller.

If at least one general purpose subtask was attached during initialization, the SQDXECB field is cleared and the SQD is queued to one of the three priority queues (low, reg, high) based on the request. If a non-recoverable queue error is detected, $SUBIT issues the $ERROR GS1 macro ($HASP095 message) and JES2 terminates. If necessary, the XECB is $WAITed on.

Once the SQD is successfully queued, the first available subtask is removed from the subtask work queue and posted.

$SEAS: Security Authorization Service

The $SEAS routine performs the setup and cleanup necessary to subtask a $RACROUT call. An SQD is obtained and initialized, then a $SUBIT call is issued. If the caller did not want to $WAIT for the request to complete, the $SEAS routine returns to the caller. Otherwise, $SEAS $WAITs for a subtask post.

If a subtask failure occurs, $SEASMSG issues the $HASP077 message and the $SEAS routine retries. If no subtask failure occurs (or if there is a second failure), control returns to the caller.

Dynamic Allocation Subtask (HOSALLOC)

This subtask resides in HASPNUC with entry point, HOSALLOC. When this subtask is dispatched, it issues an ESTAE macro to establish recovery, initializes it and waits for work, after first posting the routine that attached it. When the HOSALLOC subtask is subsequently posted, it performs dynamic allocation or deallocation as requested.

HASPDYN: Dynamic Control Block Build

HASPDYN contains routines that dynamically build the major non-spool, non-checkpoint control blocks (PCE, DCT, DTE). Daughter task elements (DTEs) provide a centralized point of communication between subtasks and the main task.

$PCEDYN

This routine dynamically adds (ATTACH) and deletes (DETACH) PCEs. It uses the PCE tables that defines the PCE types and their attributes.

For an ATTACH, the routine allocates ($GETWORK) the storage required, formats the PCE, chains it, and either marks it ready to be dispatched or marks it as having to wait ($WAIT), as required. For a detach, it tests if it can be detached now, later, or never. This test indicates whether the PCE has any allocated outstanding save areas. If the PCE can be detached, this routine dechains the PCE and frees the storage ($RETWORK) it used after all resources are freed. On an attach, the routine connects the associated DCT for PCEs that have a one-to-one PCE/DCT relationship. Errors in the parameters passed or in the tables result in the $HASP095 catastrophic abend message with a system completion code of 02D and an appropriate abend code.

This routine contains exit 27, which is entered when a PCE has been attached or is about to be detached. It can be used to obtain and free resources associated with the PCE, or it it can be used to deny the attach of a PCE.

$PCEDYDC

This routine dynamically attaches and detaches PCEs for all the DCTs in a chain and connects or disconnects the DCTs from their associated PCE.

$DTEDYN

$DTEDYN is a generalized get/free DTE service routine that handles DTE management and subtask attaches and detaches for the JES2 main task. It is called by the $DTEDYN macro instruction.

$DTEDYNA

This routine obtains and initializes a daughter task element (DTE) for subtask management. DTEs provide a centralized point of communication between subtasks and the main task. $DTEDYNA uses the $GETABLE service to access the DTE tables (built by $DTETAB in HAPSTABS) that define DTEs and their subtask characteristics. $DTEDYNA uses MVS ATTACH to attach the subtask to the JES2 main task.

For an $DTEDYN ATTACH, the caller supplies the subtask identifier, DTESTID. Then $DTEDYNA issues an IDENTIFY macro for the entry point name specified in the $DTETAB entry for this subtask. $DTEDYN obtains storage, sets up DTE identification and chains the DTE onto the $DTEORG and $DTELAST queues, anchored in the HCT. $DTEDYNA then issues an MVS ATTACH for the subtask.

$DTEDYND

This routine is used to dechain and release a DTE. It issues an MVS DETACH. It is called with $DTEDYN, which has "DETACH" as its first operand.

For $DTEDYN DETACH, the caller supplies the address of the DTE to free. $DTEDYND then issues an MVS DETACH for the subtask, dechains the DTE from the queue and releases the DTE storage. The processor that requested the services of the subtask must ensure that it has terminated (DTETECB is posted) before calling $DTEDYN to free the task.

$DCTDYN

This routine services requests to find, chain, or build DCTs. It uses the DCT tables in HASPTABS (built by $DCTTAB) to define various attributes for the type of DCT, and to get the chaining and subchaining fields. During PARMLIB processing, when an initialization statement is encountered that requires a value to be set in a specific DCT, this routine is called to locate the DCT, and if one cannot be found, build and initialize it. It is entered with a $DCTDYN macro, which passes to this routine the name of the DCT and the subscript, and whether this is a "find" or "attach" request, that is, a request to locate the DCT or generate a new one. It scans the DCT tables, using $GETABLE services, until it finds a matching DCT entry. If the DCT is not located for a find request, the caller of this routine gets a return code of 8. If the DCT is not located for a chaining request, this routine builds a default DCT and insert it into the chain of DCTs and into the subchain if this DCT belongs within the subchain. It uses subroutines DDYNCVT, DDYNATT, DDYNWLST to do its work. DDYNCVT converts the subscript to EBCDIC, DDYNATT builds and chains new DCTs, and DDYNWLST sets the default work selection list in the DCT, calling SRVWSCAN in HASPSERV.

This routine supports only local devices. It is also used to locate just the DCT and return its address in response to a $T device command (but only those $T commands that use the $SCAN facility). HASPDYN also contains initialization exit routines for devices that require them, that is, the DCTs for local readers, local printers, local punches, the offload device, and its job transmitters/receivers, and SYSOUT transmitters/receivers, remote and network line DCTs and the SNA LOGON DCT.

$DCBDYN

This routine services $DCBDYN macro instructions to create or delete (attach or detach) a DCB and a DEB for a DCT. If, for an attach request, the DCT does not need a DCB or if a DCB is already attached, this routine returns immediately to its caller. If, for a detach request, the DCB is not attached or a BSAM DCB is required, this routine immediately returns to the caller.

When the request is for an attach of a BSAM DCB rather than an EXCP DCB, no initialization is done and a JES2-created DEB is not needed. For an EXCP DCB, however, storage is obtained for the DCB and DEB (if required), and chained off DCTDCB. If the GETMAIN for storage is unsuccessful, message HASP184 is issued and a return code of 4 is returned in register 15.

For a detach, DCTDCB is zeroed when the DCB and DEB are freed with a $FREMAIN.

$DESTDYN

This routine services $DESTDYN macro instructions to dynamically add JES2 destination IDs to a network. If the destination value that is passed is non-binary, it is converted to binary format. The destination name is verified (not a valid route value). If the destination name is a node name, the routine searches for the previous destination table (RDT) entry for that particular node. It then looks for an existing matching RDT entry. The RDTs are chained in alphabetical order. If a duplicate entry is found, it cannot be altered unless the destination value in the RDT already matches the destination value that was passed, but the RDT type flags are altered. If a duplicate entry is not found, one is allocated.

HASPSUBS: Generalized Subtask

HASPSUBS is the generalized subtask attached by the $DTEDYN routine in HASPDYN. It initializes a subtask and provides for generalized subtasking of calls to a specified routine by removing and processing work requests from the available work queue. HASPSUBS also contains the subtask ESTAE recovery routines.

A recovery environment is established during subtask initialization. If recovery is required before initialization completes, HASPSUBS sets the initialization error bit in the DTE and posts $DTEDYN. If a failure occurs after initialization has been completed, the recovery routine (SUBRTRY) is called from the $STABEND routine in HASPRAS when the subtask abends. The SUBRTRY routine attempts to retry at the point of failure. If unsuccessful, it sets an error return code in the SQD and posts the caller's ECB if the SQD address is available. HASPSUBS waits for work.

Note: SUBRTRY also detects recursive abends. If this occurs (or if the subtask abends three times while processing the same request), the subtask terminates -- unless it is the last one attached.

After establishing the recovery environment, HASPSUBS $GETMAINs a save area for use by the routine to be called and posts $DTEDYN when initialization is complete. Once attached, HASPSUBS waits for work on the LIFO subtask ready queue. When posted, it calls the routine specified in its parameter list, saves the routine's return codes in the SQD, and posts its caller.

Note: The $DTE and $SQD addresses are not available to the routine called from the subtask.

HASPSUBS then checks the available work queue using the $STWORK field in the HCT (which points to the $STWORK control block, which, itself, points to the three priority queues -- low, reg, high). If no work is to be done, HASPSUBS returns to the subtask ready queue and waits for work.

On exit, HASPSUBS places one of the following return codes in register 15:

- 0 - Processing successful
- 4 - The routine address was not valid; the routine was not called.
- 8 - An abend occurred in the called routine.
- 12 - A subtask error occurred before the routine could be called.
- 16 - A subtask error occurred after the routine was called.

HASPJOS: Job Output Services

Job Output Table Services

The job output table (JOT) is built by HASPIRDA in dynamically acquired storage, with its address stored at location $JOTABLE in the HASP communications table (HCT). The JOT contains several queues of job output elements (JOEs), which describe current system output requirements. The network queue and the 72 local and remote class queues that contain work JOEs, the queue of characteristics JOEs, and the queue of free JOEs, are at the beginning of the table with other maintenance data. The bulk of the JOT is then composed of JOEs that are in one of these queues and whose number is controlled by the JOENUM operand on the OUTDEF statement.

Each JOE in the job output table can serve one of the following functions:

- Work JOEs are chained in the network queue or class queues and are the primary representatives of output processing work. The JOE counter in the job queue element (JQE) counts the number of work JOEs for the job it represents.
- Characteristics JOEs are chained in the characteristics queue and are pointed to by a field in the work JOE. The setup of a device before it can process a given work JOE is described by a characteristics JOE. Because many work JOEs require the same setup, a characteristics JOE is used or pointed to by more than one work JOE.

JOEs not currently in use to represent output work are chained in the free queue.

Because the data in the JOT is necessary for a warm start of the JES2 subsystem, a checkpoint is requested each time a change is made. During warm start, a subroutine in the initialization processor ensures that all work JOE busy flags are reset.

JES2 provides internal macro instructions for access to job output services:

- $#BLD: Builds work and characteristics JOEs from peripheral data definition blocks (PDDBs). $#BLD also tests for whether the user has authority to specify the JOE priority.
- $#ADD: Adds a JOE to the class queues.
- $#EXIT: Common exit point that deletes all of a JOE's work elements when the JOE is modified.
- $#GET: Selects JOE(s) for processing by a print/punch processor or a network or offload SYSOUT transmitter. One JOE is selected for processing by a print/punch processor. One JOE or a chain of JOEs can be selected for processing by a network or offload SYSOUT transmitter.
- $#PUT: Releases a selected JOE for subsequent processing.
- $#MOD: Modifies a work JOE's routing of a SYSOUT class.
- $#REM: Removes a JOE from the class queues.
- $#CAN: Removes all unselected JOEs for a specific job from the class queues.
- $#PDBCAN: Marks all of a JOE's non-held data sets as non-printable.
- $#POST: Posts specific processors for output.

- $#CHK: Performs I/O for the output checkpoint control block.
- $#ALCHK: Allocates a spool record for use in output checkpoint processing.
- $#NEWS: Provides support of the JESNEWS data set active in output service.
- $#JOTBLD: Rebuilds the JOT on a cold start.
- $#JOTCHK: Verifies/corrects the JOT on an all-systems warm start.

To ensure the integrity of the job queue and the job output table, each subroutine issues the $QSUSE macro prior to any change or reference to either table. Subroutines that require allocation of JOEs from those available in the JOT determine that the free queue is large enough to ensure completion of the request. If the free queue is too small, a return code indicating no process is set, and return is made to the macro caller.

$#BLD Service Routine

The $#BLD routine builds work and characteristics JOEs, using data contained in a peripheral data definition block (PDDB).

Fields built in a characteristics JOE include:

- Forms ID
- Forms control buffer (FCB) id
- Universal character set (UCS) id
- External writer id
- Flash frame id
- Burster flag
- Process mode

Fields built in a work JOE include:

- JQE address
- JOE name
- JOE id
- Security id
- SYSOUT class
- Routing codes
- Total record count
- Track cell flag
- Demand-setup flag

$#ADD Service Routine

The $#ADD service routine first ensures that there are enough JOEs available to service this request. If there are not enough JOEs, the routine sets a return code of 4 and returns to the caller, indicating that the request could not be satisfied. If sufficient JOEs are available, the class of the prototype JOE (JOECURCL) is checked to see if it is valid. If not, a $J07 abend occurs indicating an invalid SYSOUT class. Otherwise, the routine scans the queue of characteristics JOEs to determine whether a JOE matching the prototype characteristics JOE being added already exists. If no match exists, a JOE is acquired from the free queue (through the GETJOE subroutine), and the setup data from the prototype characteristics JOE is copied into it. The position of the new JOE in the queue of characteristics JOEs is determined by the collating sequence of the setup data contained in the JOE, such as forms id and FCB id.

Having either found a match or built a new characteristics JOE, the routine adds 1 to the use count in the JOE to record the number of concurrent users of the block and issues the $CKPT macro instruction to checkpoint the JOE.

A free JOE is then acquired from the free queue, through the GETJOE subroutine, and the data from the prototype work JOE is copied into it. At this point a search of the work JOEs is made to ensure that the combination JOENAME and JOEID are unique within this job. The priority of the work JOE is computed, using the SVTXPRI table based upon the JOE record count, and stored in the JOE. This routine queues the work JOE to its characteristics JOE, JQE and class queue using the CHAINWRK, CHAINJQE and CHAINCHR subroutines. Next, at $#EXIT, all eligible processors are posted ($#POST) to alert them that work is available (provided the job lock is not held). Otherwise, the routine sets return code 0 to indicate successful completion and returns to the caller.

$#GET Service Routine

There are two types of calls to the $#GET routine: network and work selection calls (specified, respectively, as NET and WS on the macro). If the call indicates NET, this means that the caller is a network SYSOUT transmitter, and the network queue is scanned for eligible JOEs as follows:

1. The proper queue is obtained with a $#JOE
2. GTSCREEN is called to screen the JOE(s)
3. If GTSCREEN returns with a return code of 4, it means that a JOE is selectable. This JOE and all other JOEs for this job that meet the requirements of GTSCREEN are chained to the network transmitter chain. The address of the last JOE on the chain and the JQE address are returned to the caller with a return code of zero, which indicates that work is available.

   A check is made for each of the JOEs to be chained to the network transmitter chain to determine if they are authorized to be sent to the destination node. This checking ensures that all of the chained JOEs have the same USERID and SECLABEL field values.
4. When GTSCREEN returns a return code of 0, the next JOE on the network queue is screened. This process continues until GTSCREEN finds a selectable JOE, in which case the processing described in step 3 is performed, or until the end of the queue is reached, in which case $#GET sends a return code of 4 to the caller indicating that no more work is available.

Note that, for network SYSOUT transmitters, GTSCREEN does not use work selection to select JOEs and does not call work selection services, but checks the following output and job conditions:

- Is the JOE busy?
- Is the JOE not selectable?
- Are the job's spool volumes available?
- Is the job held?
- Is the node available?

For a WS call for a print/punch device, a work selection setup routine is called with the $WSSETUP macro to initialize the work selection work area. Then, the appropriate class queues are scanned for a JOE that best fits the work selection criteria specified in the device work selection list. If CLASS is in the work selection list, only those class queues defined for the device are scanned for JOEs. Otherwise, every class queue is scanned.

GTSCREEN is called for every JOE that is to be screened. GTSCREEN then calls the work selection service routine (WSSERV) to determine if this JOE is selectable by the device. A non-zero return code from WSSERV indicates that the JOE is selectable. A zero return code indicates that the JOE is to be rejected.

GTSCREEN also checks for an entry for the JOE in the JWEL table. If there is no entry, then the JOE is selectable. Otherwise, the JOE is rejected.

If the JOE is selectable, GTSCREEN returns a return code of 4 to $#GET. $#GET then determines if this JOE is the "best" JOE for the device (meaning that no other JOE can better match the work selection criteria). When a best JOE is found, a SAF call is made to see if the writer is authorized to access the data being selected. If permitted, the JOE address is returned to the caller along with the JQE address and a return code of zero (indicating work is available).

If it is not the best JOE, $#GET determines if the previous JOE is better than the current JOE screened by GTSCREEN. If this is the first JOE screened, it is saved as the previous JOE, and the other JOEs are scanned for comparison. If there is a previous JOE, the current JOE is compared to the previous. The better of the two is saved as the previous and other JOEs are scanned and compared, and the better of each comparison saved as the previous JOE. This continues until no more JOEs are available. Then the best JOE (the previous JOE) is returned to the caller with the JQE address and a return code of zero. If there is no best (previous) JOE, a return code of 4 is returned indicating that no work is available for the device.

Note that GTSCREEN will check the following conditions before calling work selection services:

- Is the JOE busy?
- Is the JOE selectable?
- Are there available spool volumes for the job?
- Is the job held?

For a WS call by an offload SYSOUT transmitter, the work selection setup routine is called as it is for print/punch devices. If the offload SYSOUT transmitter is set up to select held output (DS=HELD or ANY), the hardcopy queue is scanned for a JQE with held output. GTSCREEN is called to screen the JQE(s) with held output to determine if it meets the job-level criteria specified in the work selection list. Job-level criteria are HOLD, JOBNAME, RANGE, and VOLUME.

If no job is found and DS=HELD, a return code of 4 is returned to the caller to indicate that no work was found. If DS=ANY, the class queues are scanned for non-held output. This processing is the same as for print/punch processing. When a best JOE is found, however, $#GET continues to find all job-related JOEs (as is done for networking) that meet the work selection criteria, and chains them to the network transmitter queue. (Note that when a "best" JOE is found for a SYSOUT transmitter, the criteria after the slash are ignored when processing all other JOEs for the job.) The JQE address and the address of the last JOE on the chain are returned to the caller along with a return code of zero to indicate that only non-held output was found.

If a job with held output is found and DS=HELD, a return code of 8 is returned to the caller along with the JQE address. IF DS=ANY, $#GET determines if this job has any non-held output that matches the work selection criteria. If not, the JQE address is returned to the caller along with a return code of 8. If the job does have non-held output that matches the criteria, these JOEs are chained onto the network transmitter chain and the JQE address, the address of the last JOE on the chain, and a return code of 12 (indicating both held and non-held output was found) are returned to the caller.

If the offload SYSOUT transmitter is not set up to select held output (DS=NONHELD), class queues are scanned as described for a WS call by a print/punch device and the JOEs are placed on the transmitter chain.

JES2 orders remote queues by priority within destination on each of the 36 class queues. When searching the remote queues for output and $#GET detects that there is no more eligible work (when we have passed all work for any route codes the device is set up to select) on a queue it is searching, it advances to the next class queue.

$#PUT Service Routine

The $#PUT service routine decreases the active device use counter in the associated characteristics JOE; when that count becomes 0, it indicates that device processing involving that set of characteristics is not in progress. This count is used during $#GET processing to determine whether any devices are currently processing given setup data. After decreasing the count, the $#PUT routine issues the $CKPT macro instruction to cause a checkpoint of the altered JOE.

If the caller supplies an output checkpoint record (CHK), as occurs when a $I device command is received during print/punch processing, the call to the $#PUT routine can be with an indication as to whether the CHK is valid, invalid, or not to be updated. If the CHK is *valid*, $#PUT updates the line, page, and data set counts. If the CHK is *invalid*, then $#PUT resets the line, page, and data set counts with the original totals. If the CHK is not to be updated, then $#PUT does not change the counts. $#PUT writes the CHK to spool. If the address of the CHK is not supplied, the checkpoint valid flag ($JOECKV) in the work JOE is reset.

$#MOD Service Routine

The $#MOD service routine is called whenever the routing or SYSOUT class of a JOE has been changed. The designated work JOE is removed from whatever queue it is currently on and then placed on the proper queue, depending on routing class or offload status. If the JOE's new routing is for another node, the JOE is placed on the network queue. If the new routing is for the local node, the JOE is placed on a class queue as determined by its new SYSOUT class. If the JOE is flagged for off-loading, it is placed on the SYSOUT dump queue. $POST is used to invoke a checkpoint of the altered JOT. The subroutines REMCHR, REMWRK, and REMJQE are called to remove the work JOE from the characteristics JOE and JQE JOE chains. The work JOE is requeued using the CHAINWRK, CHAINJQE and CHAINCHR subroutines. $#MOD posts (via $#POST) waiting processors of work (if the job lock is not held) and returns to the caller. If the job lock is held $#MOD just returns to the caller without performing the $#POST.

$#REM Service Routine

The $#REM service routine first locates the address of the work JOE preceding the one to be removed by scanning the entire class queue associated with the JOE. When the JOE preceding the one to be removed is found, the routine calls subroutine REMWORK to remove the work JOE along with the corresponding checkpoint and characteristics JOEs from the class queue, characteristic JOE chain, and from the job JOE chain. If a checkpoint is required for the JOT, the $#REM routine passes control to REMWORK.

On return from REMWORK, $#REM checks to see if a post for JOE is required and if so issues a $POST before returning to the caller. This post wakes up processors waiting for JOEs. Otherwise $REM just returns to the caller.

If the JOE points to an unspun IOT, the JOE is removed. An unspun IOT occurs because of a previous JOT full condition, or because the execution processor marked the IOT unspun when it found a JOE busy.

If the JOE to be removed is a spin JOE, $#REM determines if multiple JOEs share the same allocation IOT. If so, only this one JOE is purged. If not, $#REM checks if PURGE=YES was specified on the macro. If so, it checks if it was passed the IOT. If not, the IOT is read in and passed to the $PURGER routine to free the spool space. If PURGE=NO was specified on the macro, the spool space is not returned. On return, $#REM removes the JOE.

$#POST Service Routine

$#POST posts specific processors that are waiting for output work so that they can select output work to process. The following types of posts are processed:

- Work JOE post
- JQE post
- SYSOUT transmitter post
- Spooled message post

$#POST uses PSTSUBJ, JOE post subroutine, to process this post request. $#POST checks to see if the JOE to be posted is associated with this node or a special local device. If so, $#POST scans the $WAIT JOT queue and posts ($POST) processors that are eligible to select this JOE. Otherwise, the JOE is destined for another node and $#POST posts the eligible SYSOUT transmitters.

When an output element is available to be processed, $#POST checks if there are any RJE devices in autologon mode. If so, it calls $GTSCREEN for each to determine whether they are eligible to select the work. If so, an indication is set in the remote attributes table (RAT), so that the line manager knows that work exists for this remote and can automatically log it on. An indicator is also set in the DCT, so that the line manager can post the device directly.

For a JQE post, all JOEs for this JQE are screened to determine if they can be selected by any processor waiting for work, that is, output. If no JOEs exists for this job, the following occurs:

- If there are any spool offload PCEs waiting for work, the JQE is checked to determine if there are any held data sets. If not, no processors will be posted. If there are held data sets for this job, and the spool offload device is set up to pick up held data sets (no device is posted if the spool offload device is not set up for held data sets) the job level criteria defined for the job (HOLD, VOLUME, RANGE, JOBNAME) are compared to the job-level criteria defined for the device. If the job can be selected by the spool offload device, the device will be posted. Otherwise, no device is posted.
- Any other PCE waiting for work will not be posted if no JOEs exist for the job.

For spool offload, if JOEs do exist for the job but none of them can be selected by the spool offload device, the JQE associated with this job will be checked to determine if held data sets exist, and the same processing as described above for held data sets will occur.

For other devices, if JOEs exist but cannot be selected by the devices, no device is posted.

$#POST calls the transmitter subroutine PSTSUBX to post networking SYSOUT transmitters. PSTSUBX posts (via $POST) all waiting ($WAIT) SYSOUT transmitters and all idle SYSOUT transmitters for this line.

$#POST calls the post spooled message subroutine, PSTMSG. PSTMSG loops through the line DCTs and flags the line DCT to indicate the output post.

$#POST issues an abend, catastrophic error $J06, when it is entered without the checkpoint being held. After successful processing, $#POST returns to the caller.

$#CHK Service Routine

This routine reads or writes output checkpoint records to spool. $#CHK doesn't check for I/O completion if reading is performed. If writing is performed with the WAIT=NO option $#CHK returns immediately to the caller with a return code of 0 even if the I/O doesn't complete. If writing is performed with the WAIT=YES option $#CHK waits for the I/O to complete before issuing the next $EXCP. If $#CHK experiences a permanent I/O error for either read or write, $#CHK issues a $DISTERR macro, checkpoints the work JOE, and exits to the caller with a return code of 4.

$#ALCHK Service Routine

This routine allocates a spool record so that job output can be checkpointed. $#ALCHK issues a $GETBUF to obtain JCT and/or IOT buffer space if the JCT and I/O buffers were not passed on input. The $#ALCHK obtains the job lock if required and reads into the JCT buffer the JCT (via $JCTIO) and into the IOT buffer the IOT (via $EXCP). Then $#ALCHK issues a $TRACK to obtain a checkpoint spool record, writes out to spool the IOT and JCT if requested by the caller, frees the job lock and the buffers and returns to the caller.

If a buffer had to be created for a secondary allocation IOT, $#ALCHK also frees that buffer. If an I/O error occurs while reading or writing, $#ALCHK issues a $DISTERR macro, checkpoints the work JOE and returns to the caller with a return code of 4. Otherwise, $#ALCHK returns to the caller with a return code of 0.

$#CAN Service Routine

The $#CAN service routine scans the network queue and each of the 36 class queues for work JOEs that belong to the job supplied in the parameter list. For each non-busy work JOE found in the scan, a call to REMWORK is made to return the work JOE to the free queue and to remove the JOE from the job JOE chain.

If a $POST of the JOE is required, on return from REMWORK a $#CAN issues a $POST before returning to the caller.

$#PDBCAN Service Routine

This routine marks all of a JOE's non-held data sets as non-printable so that, when data is released by the PSO processor, the data sets can be gathered together under the original JOE name. This service routine is entered via the $#PDBCAN macro.

$#JOTBLD

$#JOTBLD builds the JES2 job output table (JOT) during cold start initialization. It places all JOEs on the free JOE queue.

$#JOTCHK

$#JOTCHK verifies the JOT on an all-system warm start. JOTVERIF verifies, and if necessary, corrects errors found during verification. JOTVERIF validate the free JOE queue, the characteristics JOE queue, and the work JOE queues. It also validates the chains of work JOEs from the characteristics JOEs, and the chains of work JOEs from the JQEs.

Message $HASP440 is issued when JOT verification is complete. If an uncorrectable error is found during the verification process, message $HASP498 is issued to ask the operator if the JOT should be rebuilt or to terminate JES2. JOTREBLD rebuilds the JOT if the operator requests it. $HASP499 message is issued if all rebuilding of the JOT has failed. Initialization terminates.

To verify the free JOE queues, the routine passes through the JOT starting at the end of the JOT. For each free JOE found, its free JOE chain field will be verified or corrected. A flag is set in the JOT verification work area (JVWA) for each JOE when it has been successfully processed. The next free JOE on the chain will be verified as follows: (1) it is a valid offset in the JOT, (2) it has already been processed, (3) this free JOE was the last one processed. If there is an error, message $HASP497 is issued, indicating that error correction is in progress.

To verify the characteristics JOE queue, each JOE on the chain will be verified as follows: (1) a valid offset in the JOT, (2) it is sorted within the chain by setup characteristics, (3) the next characteristics JOE is a valid offset and it has not been processed. Message $HASP497 is issued if an error is found. The JOE in error is detached from the chain. A second pass through the work area (JVWA) picks up the unprocessed characteristics JOE and inserts it correctly into the chain.

Validating the work JOE queues is done by scanning the work JOE chain. Each JOE is verified as follows:

- Is it a valid offset in the JOT?
- Is it on the correct output queue?
- Is it sorted in the chain by priority order within destination?
- Is the next work JOE a valid offset of a work JOE yet to be processed?
- Is the SYSOUT class valid for this class queue?
- Is the offset of the JQE in this work JOE valid, and is the JQE not on the free or purge queue?
- Is the offset of the characteristics JOE valid?
- Is the backwards work JOE chain pointer correct?

The validation of the chains of work JOEs from the characteristics JOEs includes checking to see if the JOE is a valid work JOE and if it points back to the same characteristics JOE. The validation of the chains of work JOEs from the JQEs includes one pass through all the JQEs, and for each non-free JQE, the chain of work JOEs are verified to see that it is a valid work JOE and that the work JOE points back to the same JQE.

If an uncorrectable error is found, a $ERROR routine is issued with a reason code of $J08. A return code of 4 indicates an uncorrectable error was found during the verification/correction process. A return code of 8 indicates that an uncorrectable error was found during the total rebuild.

Job Output Table Services - Common Routines

The job output table (JOT) service routines described in the following paragraphs call the JOT services common routines, which provide common services to all JOT service routines without duplication of code.

REMWORK: Removing a Work JOE from the JOT

The REMWORK service routine entered with register 6 containing the address of the JOE to be purged. Next, the active device counter in the associated characteristics JOE is decreased if the work JOE is currently active on a local device. The count of elements associated with the characteristics JOE is decreased and if the count goes to 0, REMJOE is called to remove the characteristics JOE. Then REMCHR removes the work JOE from the characteristics JOE chain, and REMWRK removes the work JOE from the JOT class queue. REMWORK then frees the work JOE and unchains it from the work JOE chain. If the job has no more JOEs and JQEHLDCT is 0, the job is requeued for $PURGE.

GETJOE: Obtaining a JOE from the Free Queue

The GETJOE service routine decreases the number of free JOEs remaining in the JOT (JOTFREC) and compares this count with the JOE utilization threshold value ($JOETHRS). If JOTFREC has fallen to equal or go below $JOETHRS, GETJOE turns on the $JOEMSG bit in the $RESHORT flag to indicate a JOE shortage. GETJOE issues the $CKPT macro instruction to checkpoint both the new JOE and the table header to reflect the new head of the free queue (JOTFREQ). The subroutine returns the address of the new JOE in register 1.

REMWRK/REMJOE: Remove a JOE from the Current JOT Header Queue

To remove the work JOE, REMWRK gets the address of the JOT queue head from the JOE. REMWRK examines the JOEROUT field to determine if the JOE is on the network queue. If the JOE is not located on the network queue, JOECURCL is used to determine the proper JOT output queue. If the JOE is on the network queue, that queue is used.

After locating the JOE, the REMJOE routine removes the JOE from the SYSOUT queue. The JOE is removed by placing its chain field (JOENEXT) in the preceding JOE. This results in chaining the queue head if the JOE is the first in the queue. A $CKPT macro is issued to checkpoint the previous JOE.

REMCHR: Remove Work JOE from its Characteristics JOE Chain

This routine unchains a work JOE from its characteristics JOE and issues a $CKPT macro to schedule the checkpoint of the preceding JOE in the characteristics JOE chain.

REMJQE: Remove Work JOE from its JQE-JOE Chain

This routine unchains a work JOE from its JQE and issues a $CKPT macro to schedule a checkpoint of the preceding element in the JQE-JOE chain.

FREEJOE: Returning a JOE to the Free Queue

FREEJOE returns a JOE to the free queue. Register 2 contains the offset of the JOE to be added.

The free JOE count (JOTFREC) is increased and compared to the JOE utilization threshold value ($JOETHRS). If JOTPREC is not at or below $JOETHRS, FREEJOE turns off the $JOEMSG bit in the $RESHORT flag to indicate no shortage. Both the new JOE and the header are flagged for checkpoint. The new JOE is added to the free queue. Finally, the JOE preceding the JOE just returned is checkpointed and control returns to the caller. If the resulting free-JOE count indicates that there are processors waiting for a free JOE and that another scan of the JOT is required, FREEJOE sets register 4 to nonzero.

CHAINCHR: Chain Work JOE to Characteristics JOE

CHAINCHR is called to chain the work JOE from its characteristics JOE. The characteristics JOE is pointed to from the work JOE. The work JOE is chained by priority within the SYSOUT class.

CHAINJQE: Chain a Work JOE to its JQE

CHAINJOE is called to chain the work JOE from its job queue element (JQE). The JQE is pointed to from the work JOE. The work JOE is chained from the JQE by class.

CHAINWRK: Chain a Work JOE to SYSOUT Queue

CHAINWRK is called to chain the work JOE to its SYSOUT queue in the job output table (JOT) by priority within its destination. The JOE is chained off a SYSOUT class queue, network queue, or dump queue.

NEWSADJ: Adjust Responsibility Count

NEWSADJ adjusts the responsibility count in the JESNEWS job structure and clears work JOE fields.

$#NEWS: Create or Modify the JESNEWS Data Set

Called by the $#NEWS macro, this routine modifies or creates the JESNEWS data set. It is called when there is a JESNEWS spin IOT that a job created. The routine first marks the spin IOT as a non-allocation IOT (so the spool space will not be purged when the job that created it is purged) and is written to spool. A call is then made to HASPSSRV to create a JESNEWS job. The JESNEWS PDDB is then moved from the spin IOT to the job's IOT. Also, the spool space is consolidated in the job's IOT, if there is enough space for it. Otherwise, the spin IOT is made to look like a secondary allocation IOT to the job's IOT. Then the checkpoint fields of the JESNEWS job are updated, and the JESNEWS job is put on the hardcopy queue. Finally the old job is purged. If the JESNEWS data set in the spin IOT is null, the current JESNEWS data set will be deleted.

JESNEWS Security: To prevent the writing of data from a user with a high security classification to users with a lower security classification, JES2 carries with the data the security token of the user entering data for JESNEWS. When selecting the JESNEWS data for printing for a job, JES2 calls SAF to verify that the security classification for the job is equal to (or higher than) the classification of data in JESNEWS. If it isn't, JESNEWS will not be printed for the job. If the data set can be printed with the job, the security token of the job to be printed (not the token of the JESNEWS data set) is passed to the Print Services Facility or to the HASPPRPU module, as appropriate. This results in the JESNEWS data being printed with the same security labels as the other data sets being printed for the job.

XWTRPOST Service Routine

The XWTRPOST service routine is called from the $#POST routine to invoke an external writer that is waiting on JOT services. The $#WTR routine scans the subsystem job block (SJB) queue and uses the $XMPOST macro instruction to selectively post each waiting external writer that is waiting for this type of output.

HASPTERM: JES2 Termination Services

HASPTERM comprises the services that JES2 uses to process catastrophic errors and system abends, abend recovery, and both normal and abnormal termination.

The following functions are contained in HASPTERM:

HASPTRCA: the central termination/recovery control block, assembled into HASPTERM

$ABEND: the JES2 main task ESTAE routine

$RETRY: the routine that $ABEND indicates RTM should enter if JES2 recovery will be performed, and related routines in support of the $ESTAE facility

$HEXIT: the routine entered to process JES2 termination due to operator
(HEXTINIT) direction ($HEXIT), initialization failure (HEXTINIT),
(HEXIT) or abend (HEXIT)

HEXESTAE: the ESTAE routine established for $HEXIT to ensure an SVC dump is requested in JES2 termination abends

OTHERS: various subroutines required for the routines above

$ABEND: JES2 ESTAE Routine

The $ABEND routine provides standardized procedures for processing of MVS or JES2 errors. During initialization, JES2 issues an MVS ESTAE macro instruction so that MVS will give control to the $ABEND routine when an error occurs. $ABEND is entered as a result of an MVS detected error (for example, a program check or SVC detected error such as an S400) or a logic error detected by JES2 resulting in invocation of the $ERROR macro; the $ERROR macro issues the ABEND macro, which invokes the $ABEND routine.

The $ABEND routine also supports CALLRTMs issued by the SCV34 exit routine in HASCSIRQ and CALLRTMs issued by the subtask ESTAE routine, $STABEND.

When invoked, $ABEND informs the operator of the error, provides diagnostic logging, determines whether the environment in which the error occurred is appropriate for processor recovery and determines whether an eligible processor recovery environment exists.

If recovery can take place, $ABEND determines the rate at which errors are occurring. If that rate equals or exceeds an installation determined rate (specified via the RECVOPTS initialization parameter), the operator must authorize recovery. (The default error rate is 2 errors in a 24 hour period.) Finally, if recovery can occur, $ABEND uses the SETRP macro to establish $RETRY as the retry routine and to cause LOGREC recording. $ABEND then returns to the MVS recovery termination manager (RTM). RTM causes the JES2 main task execution to resume at $RETRY.

If recovery cannot occur, $ABEND branches to JES2 termination processing at label HEXIT.

An error recovery work area (ERA), built upon entry to $ABEND, and the central termination recovery control area (TRCA) provide all work areas and communication fields required by $ABEND, $RETRY, and the HASPTERM subroutines. The ERA is created by $ABEND and contains information relating to a specific abend. It is kept in storage by JES2 until all JES2 error and recovery processing for the abend is complete. The TRCA is assembled in HASPTERM at label HASPTRCA, and is mapped by the DSECT generated by the $TRCA macro.

$ABEND is entered in key 0 and in supervisor state running under on ESTAE SVRB. Immediately after establishing addressability, it changes to the JES2 protect key. $ABEND then establishes a PCE format save area (appropriate for $SAVE and $RETURN macros) and sets TRCAOREC to 1 in the TRCA so that recovery is authorized. If either of two recursion bits is on, $ABEND sets bit TRCATERM to 1 indicating that recovery is not possible. (The two recursion bits are TRCAABND and TRCARTRY indicating that either $ABEND or $RETRY was in control.) The TRCAABND bit is then set to 1 to avoid contention over multiple abend recoveries. Because $ABEND is not its own ESTAE routine, it cannot be entered on a recursive basis. If the TRCAABND bit is on when $ABEND is entered, it indicates that the error occurred after $ABEND returned to RTM (before $RETRY turned TRCAABND off) and is therefore an unrecoverable type of error.

Next, $ABEND sets up an error recovery area (ERA). If the GETMAIN macro that was issued to acquire the ERA fails, $ABEND uses an emergency ERA (in the TRCA). If the emergency ERA is already in use, it is overlayed and the TRCATERM bit is set on to cause JES2 termination.

The ERA and the control blocks to which it points represent the environment in which the error occurred. The ERA and the control blocks are also used by the processor recovery routines in communicating with $RETRY. $ABEND initializes the ERA fields required for its processing; if recovery is attempted, $RETRY will initialize the fields unique to recovery processing. If a system diagnostic work area (SDWA) was provided to $ABEND, the registers associated with the HASJES20 request block (from label SDWASRSV) are moved to the ERA. Then, if the error was a JES2 catastrophic error (signaled by the $ERROR macro), the values in registers 15, 0, and 1 are set to the values those registers contained immediately prior to execution of the $ERROR macro. Because $RETRY and other recovery processing can change these register fields in the ERA, a second copy of the registers is made in the ERA.

Next, if $ABEND was entered as a result of $ERROR macro invocation and if the nature of the error is such that a system IPL must be performed before JES2 can be successfully initialized (RIPL=YES specified as an operand of the $ERROR macro), then the value in counter TRCARIPL is increased. If recovery for this error succeeds, then the value in the counter is decreased. Should JES2 terminate for any reason while TRCARIPL is nonzero, bit CCTSTRPL in HCCT field CCTSTUS is set on to prevent subsequent JES2 initialization without an intervening system IPL. At this point $ABEND issues message $HASP095 to inform the operator of the error. $HASP095 is issued as an action message for which a DOM macro is issued to release the console message buffer and delete the MVS copy of the message when recovery of this error has succeeded. The DOM is issued in subroutine RTRYERAF.

$ABEND now calls ABNDERRL to analyze where the error occurred and save the results of the analysis in the ERA. ABNDERRL saves the failing instruction address and then determines whether the failing instruction address is in another address space or in JES2's address space. ABNDERRL uses the ERMODULE subroutine to search the job pack area (JPA), link pack area (LPA), and CDE extent lists for a module associated with the failing instruction address. When the module is found, ABNDERRL copies its name into the ERA. On return from ERMODULE, ABNDERRL copies the module map information into the ERA. If the failing address is in a JES2 load module, ABNDERRL then invokes the ERMODMAP subroutine to locate the entry in the JES2 module map for the address.

$ABEND now calls subroutine ABNDCKRP to determine whether a JES2 processor was in control at the time of the error. If a JES2 processor was not in control, bit TRCANOPC is turned on. ABNDCKRP also determines whether JES2 can recover from the error. If error recovery is not possible bit TRCATERM is turned on.

If a PCE was in control at the time of the error, bit TRCATERM is set to 1 unless an eligible recovery environment exists. This is represented by an inactive PRE with a nonzero recovery routine address.

If, at this point, recovery is possible (TRCATERM off) and the rate at which errors are occurring has reached or exceeded the installation's specified threshold (RECVOPTS initialization parameter), $ABEND issues message $HASP070 requiring the operator to authorize the recovery attempt and permit suppression of the automatic dump. (Error rate calculation is performed by subroutine ABNDRATE; $HASP070 operator communication is handled by subroutine ABNDH070.)

$ABEND then calls subroutine ABNDTVRA to set up SDWA serviceability information for LOGREC recording. ABNDTVRA formats the system diagnostic data area-variable recording area (SWDA-VRA) mapped by the MVS IHASDWA macro and JES2 macro $LGRR.

Unless suppressed by the operator (via a NODUMP response to $HASP070), $ABEND calls ABNDDUMP if recovery is to be attempted. ABNDDUMP invokes $SDUMP, using the default header. If the operator suppressed the dump or the dump was not successful, bit LGRP1ND in field LGRPFLG1 of the JES2 LOGREC record (LGRR) is turned on to indicate that there is no dump corresponding to this error.

If processor recovery cannot occur (TRCATERM is on) or if the operator denied authorization to attempt recovery (replied "TERMINATE" to message $HASP070), $ABEND provides an indicative dump and enters termination processing at label HEXIT. In this case, $ABEND issues a SETRP macro requesting RECORD=YES just prior to completion of termination processing.

If processor recovery can occur, $ABEND still issues an indicative snap dump to the hard copy log and then issues a SETRP macro to establish $RETRY at JES2's retry routine. The SETRP also indicates to **not** free the SDWA (it is freed when the associated ERA is freed in RTRYERAF) but to cause LOGREC recording of the error. $ABEND then returns to RTM and JES2 execution resumes at label $RETRY.

$RETRY: JES2 Retry Routine

RTM calls $RETRY when $ABEND (the JES2 ESTAE routine) determines that processor recovery should occur. $RETRY manages all JES2 processor recovery. $RETRY is divided into front-end processing and back-end processing. Front-end processing consists of passing control to a processor recovery routine via the $SETRP macro. Back-end processing receives control upon return from a processor recovery routine and effects resumption, termination, or percolation (as specified by the processor recovery routine) via the $SETRP macro.

$RETRY Front-end Processing

On receiving control from RTM (in the JES2 protect key), $RETRY establishes addressability and sets a recursion flag (TRCARTRY). When this flag is on, $ABEND knows that the error that caused $ABEND to be entered occurred in $RETRY processing. $RETRY then turns off the $ABEND recursion flag (TRCAABND), and sets up a PCE format save area to enable $RETRY to issue $SAVE and $RETURN macros when required.

If no other JES2 processor is in recovery, message $HASP072 is issued to inform the operator that recovery is in progress. The message is an action message, which will be deleted via a $DOM macro when all processors in recovery have successfully recovered.

When ERA initialization is completed, $RETRY sets the ERAPRECT field to zero. The value in this field is increased each time a processor recovery routine receives control to attempt recovery from an error. (The value of ERAPRECT exceeds 1 only in cases of percolation.) $RETRY then sets ERAPSVAD to indicate the current save area level.

If the error occurred during error recovery, the current ERA is displaced by the error and becomes the current error again when recovery for the latest error has succeeded. This occurs when a processor recovery routine issues a $ESTAE macro to provide a recovery routine for its processing. $RETRY does not receive control if an error occurs when the newest (or topmost) PRE is active.

If the processor was not already in error recovery, it is set to be temporarily exempt from non-dispatchability, the non-dispatchability counts of all PCEs are increased, and the number of processors in recovery ($RECVCNT) is increased.

At this point, the RTRYGORE recovery routine receives control. RTRYGORE is also invoked by percolation processing when a higher-level recovery environment exists and is eligible for use and inactive.

RTRYGORE marks the PRE active and associates the PRE and ERA with each other via pointers ERAPRE and PREERA. This association is necessary because the ERA contains the recovery routine attributes used by the PRE.

$RETRY modifies the register area in the ERA (ERAREGS), if necessary, so that register values are the same as when the recovery routine was established (that is when the $ESTAE routine is invoked and builds the PRE). If the error occurred at the same save area level at which the recovery environment was established (that is, no $SAVE macro was issued between the time the $ESTAE macro was issued and the error occurred) no modification of the register values already in the ERA is necessary. If a $SAVE macro was issued between the time the $ESTAE macro was issued and the error occurred, $RETRY changes the register area (ERAREGS) in the ERA to reflect the values in the registers prior to the issuing of the $SAVE macro. If no $SAVE was issued between the time of the error and the issuing of the $ESTAE macro, no register values need to be modified. Otherwise, the register area is initialized using the processor save area (PSV) created by the first $SAVE following the $ESTAE macro that provided the recovery routine.

Finally, the values of all registers except 1, 11, 13, 14 and 15, are set from the ERAREGS register area and the recovery routine is entered. (Register 1 is set to the address of the ERA; register 11 is set to the HCT address; register 13 is set to the PCE address; register 14 is set to the address of back-end processing

(RTRYBACK); and register 15 is set to the address of the recovery routine, which it gets from PRERECAD.)

$RETRY Back-end Processing

A processor recovery routine returns control to RTRYBACK upon completion of its processing (the processor recovery routine was passed the address of RTRYBACK in register 14).

RTRYBACK reestablishes addressability (ensuring that register 11 points to the HCT and register 10 points to the TRCA), turns on a recursion flag (TRCARTRY in TRCAFLAG) for inspection by $ABEND and performs validation of essential control blocks. Should any validation fail, RTRYBACK issues catastrophic error $ER5. A temporary PCE is established for use throughout most of the processing. The recursive flag is turned on to prevent looping through the recovery process if an error reoccurs.

If the processor recovery routine specified that termination processing should occur (the $SETRP macro specified TERMINATE), then message $HASP074 is issued near label RTRYI074 to inform the operator that the recovery attempt failed; termination processing is then entered at label HEXIT.

If the processor recovery routine specified that percolation should occur (the $SETRP macro specified PERCOLATE), the most recently created, eligible, and inactive PRE recovery routine is given control. If the recovery routine address is zero, which indicates no recovery routine is available, a branch is taken to label RTRYI074 where message $HASP074 is issued to inform the operator that the recovery attempt failed. Termination processing is then entered at label HEXIT. Similarly, if no eligible PRE is found, a branch is taken to RTRYI074, and the processing previously described takes place.

If the recovery routine's address is not zero, RTRYBACK removes each active PRE from the PCE's PRE stack and, for debugging purposes, stores its address in the PREPREPC field of the previous (next older) PRE. The PRE is removed altering the chaining. Note that an active PRE (other than the most recently created) exists on the PCE's PRE stack only when a processor recovery routine has issued an $ESTAE; that $ESTAE established recovery routine which in turn received control as the result of an error during processor recovery.

When the active PREs are being removed, RTRYBACK updates the PCEERA field. This makes it appear to the percolated-to recovery routine that it is processing the original error.

When an eligible and inactive PRE with a nonzero recovery routine address becomes the most recent PRE, $RETRY branches to RTRYGORE, to pass control to the PRE's recovery routine.

If the processor recovery routine specifies termination or percolation and (at RTRYCKRE resumption) the $SETRP RESUME= macro is not specified, a catastrophic error $ER5 is issued indicating that an invalid option was specified. If RESUME=YES is specified, all PSVs up to, but not including, the PSV associated with the active PPE are purged.

Each PRE on the active PRE's percolation stack (PREPREPC) is then freed.

At this point RTRYBACK marks the PRE inactive and moves resumption register values from the ERA to the TRCA. Any previous ERA is made the current ERA. The RTRYERAF subroutine is then called to perform any housekeeping functions for the ERA.

If there was no previous ERA, the processor is no longer in recovery and it temporarily can be dispatched. The PCEDSPXT flag is turned off and the non-dispatchability counts of all PCEs are reduced by one. At this point the number of processors in recovery (in field $RECVCNT) is reduced by one. If no other processors are in recovery ($RECVCNT is equal to zero), message $HASP072 ("RECOVERY IN PROGRESS") is deleted via a $DOM macro and $RETRY issues message $HASP073 RECOVERY SUCCESSFUL.

For every processor in recovery, register values for resuming are established, and the recovery routine turns off the recursion flag (TRCARTRY). This flag setting indicates that the next error did not occur while the recovery routine was in control, but instead is a new error. Finally a BR R15 is executed to branch to the specified resumption point in the processor.

$HEXIT: JES2 Common Exit Routine

HASPCOMM invokes $HEXIT when a $P JES2 command is accepted. HASPNUC enters HEXTINIT if JES2 initialization has determined that there has been an error and requires termination. $ABEND and $RETRY enter $HEXIT to invoke abnormal JES2 termination after abend or retry processing has failed. All three of these routines set a TRCA flag to indicate they were entered and then enter common code.

$HEXIT issues a ESTAE to provide at least a dump if $HEXIT should abend. Then $HEXIT checks for the presence of the $HCCT. If the $HCCT is present, $HEXIT frees authorization indexes for the JES2 address space; this is necessary for the orderly shutdown of JES2. If no $HCCT is present, the freeing of authorization indexes is not necessary. $HEXIT then stores the value $SYSEXIT in the $STATUS field of the HCT to indicate that JES2 is terminating and then proceeds with JES2 termination by halting JES2 devices and asking the operator for specific termination options.

Before $HEXIT scans the options table, user exit 26 can be invoked. This exit gives the installation the chance to free resources or to customize JES2 termination. If the reason for the termination is a $PJES2 command or an initialization exit, JES2 does not ask for termination options. If the reason for the termination is not a $PJES2 command or an initialization exit, the exit point is processed after the operator replies to message $HASP098.

When the operator responds, $HEXIT scans the response. The following are the abend options for JES2 termination:

DUMP	provides a system dump and request more termination options at HEXTDUMP
EXIT	provides a quick JES2 termination at HEXQUICK
PURG	provides some clean up then enter JES2 termination at HEXITGO
SNAP	provides a SNAP dump to the console and ask for more termination options at HEXTSNAP

For DUMP and SNAP, $HEXIT issues the dump and then asks again (via $$WTOR) for more termination options from the operator. When the operator supplies either EXIT or PURGE, $HEXIT continues JES2 termination.

If the primary subsystem is terminating, $HEXIT (at HEXITGO) deletes action messages $HASP050 and $HASP601 if they are outstanding and accesses the UCB lookup table through the communication vector table (CVT) and, in protection key 0 (via MODESET), scans each UCB. UCBs representing JES2 input and output devices contain attention indexes that cause the JES2 attention exit to receive control if an attention signal is received from the device. For each such UCB, the routine issues IOSGEN to reset the JES2 attention index.

When all UCBs have been processed, MODESET is issued to reset to the HASP protection key.

For non-3800 printers (for example, the 3211, 3203, 4245, and the 4248) the log area is freed. For IBM 3800 printers, if an MDR buffer exists, the log area is freed. For non-3800 printers, the log area pointer is cleared.

Next, $HEXIT scans:

- The status/cancel queue (pointed to by the HCTTSCS field in $HCCT)
- The process-SYSOUT queue (pointed to by the HCTPSOQ field in $HCCT)
- The STAC-pending queue (pointed to by the CCTPSPND field in $HCCT).

$HEXIT removes each subsystem job block (SJB) from the queue, and posts the SJB to indicate that the corresponding job abnormally terminated. $HEXIT (at HEXQUICK) frees the JES2 trace tables in CSA. If SMF is supported by the subsystem, a type 45 SMF record is written, indicating that JES2 is terminating. Finally, each active JES2 subtask is posted complete, then detached, if normal termination, and the JES2 CSA control blocks are freed.

ERRORRET: Termination Completion Routine

This routine is entered either after JES2 termination processing is complete or directly if termination processing was bypassed because the operator specified termination option EXIT in response to message $HASP098. The CCTSTUST indicator in HCCT field CCTSTUS is set to show that termination is complete. $HEXIT checks the TRCA to determine if it was entered from $ABEND as an ESTAE routine. If so, it returns to RTM after first issuing the JES2 termination message. If $HEXIT was not entered by $ABEND, it issues the JES2 termination message, zeroes register 15, and exits (via SVC 3).

$HEXIT ESTAE Routine

HEXESTAE is an ESTAE routine established on entry to $HEXIT. It's purpose is to ensure that, if JES2 termination fails, an SVC dump is issued. HEXESTAE issues message $HASP087 and uses the $SDUMP macro to request an SVC dump. The title used is the default $SDUMP title with "(HASP ESTAE ABENDED)" appended.

Recovery/Termination Subroutines

The ABNDDUMP subroutine invokes the JES2 $SDUMP macro to cause a system dump of JES2 to be written to a SYS1.DUMP data set. If no title was provided, one is constructed from the subsystem id (first 4 bytes of the job name field in the TIOT), module name (normally HASJES20), JES2 version, and error code. The $SDUMP macro is used to request the option indicating that the operator should be prompted if the dump should fail.

The ABNDSNAP subroutine produces the JES2 indicative dump (to the hardcopy log if register 1 equals 0 on entry, otherwise to the console). The indicative dump consists of the following:

- JES2 product version (FMID)
- Current level-set PTF applied
- Subsystem id (4 bytes of the job name field in the TIOT)
- Module name (normally HASJES20)
- JES2 version
- Current date and time
- Abend or catastrophic error code (with description if available)
- Address at which the error occurred
- CSECT in which the error occurred
- Offset within the CSECT at which the error occurred
- Subroutine calling sequence - as determined by the PSV chain (the subroutine must have used the $SAVE macro to be included in this trace)
- Failing instruction (if within the JES2 address space)
- PSW
- Instruction length code and interrupt code
- The cross memory environment (the home ASID, the primary ASID, and the secondary ASIDs)
- PCE name and address
- Related job name and job number, if any
- RB-level register contents at the time of error

The ERMODULE subroutine finds the name of the load module containing the address that it is passed and the offset within the load module of that address. It can be used in a subtask environment.

The ERMODMAP subroutine finds the name of the assembly module (if known to JES2) containing the address that it is passed and the offset, within the module, of that address. (ERMODMAP gets the address of the assembly module using the $MODMAP table in HASPTABS.) It can be used in a subtask environment.

The ERBASOFF subroutine formats the assembly module name, its address and offset within the module, at which the error occurred for ABNDSNAP.

The ABNDH070 subroutine handles the $HASP070 message (SPECIFY RECOVERY OPTIONS) dialogue with the operator. A reply of SNAP causes ABNDSNAP to produce an indicative dump followed by the reissue of $HASP070. A response of TERMINATE turns off bit TRCAOREC while a response of RECOVER does not alter bit TRCAOREC. If NODUMP was specified with either TERMINATE or RECOVER, bit TRCAODMP is turned off to indicate that no dump is to be issued. Any invalid reply to $HASP070 results in the operator being notified via the $HASP071 message, and message $HASP070 being reissued.

The ABNDTVRA subroutine formats SDWA fields and formats the SDWAVRA according to the JES2 $LGRR DSECT (with the exception of bit LGRP1ND). SDWAMODN is set to the name of the main JES2 load module (HASJES20). The ABVRESTA routine is an ESTAE that gets control if ABNDTVRA abends during the formatting of the VRA. ABVRESTA retries ABNDTVRA so that the LGRP1ABD bit remains set to 1.

The ABNDERRL subroutine determines details concerning the failing storage location, such as, the module that failed, the $MODMAP entry for the module, the failing address, and the failing address space (the failing address space might be a functional subsystem address space, which contains HASPFSSM). ABNDTWRA uses this information that ABNDERRL places in the ERA.

The ABNDT095 subroutine formats the $HASP095 message.

The ABNDCKRP subroutine determines whether a JES2 processor was actually in control at the time of the error and whether recovery is possible (recovery is not possible if a PCE was not actually in control).

If termination is occurring because of a $P JES2, then both bits TRCANOPC and TRCATERM are set to 1 to indicate that a PCE was not in control. If a PCE was in control, ABNDCKRP makes the following checks to determine if the error circumstances preclude recovery:

- Determining if another address space was in control at the time of the error.
- Determining whether the error is a machine check, restart key, translation error, paging I/O error, etc.

If these error circumstances exist, ABNDCKRP sets TRCATERM to 1 to cause termination.

The ABNDRATE subroutine monitors the rate at which JES2 main task errors are occurring and returns an indicator when the installation specified threshold is reached or exceeded. It can be used in a subtask environment

The RTRYERAF subroutine performs error recovery area (ERA) housekeeping and returns an indication of whether the ERA was freed. ERAPRECT (the number of PREs associated with the ERA) is decreased and if the result is not zero then the subroutine simply returns to its caller indicating that the ERA was not freed. (It is not freed because there are no outstanding PREs.)

If ERAPRECT reaches zero,

- All resources associated with the ERA are released.
- TRCARIPL is decreased if the error represented by this ERA caused it to be increased.
- The $HASP095 message, which was issued when the error occurred, is deleted via the DOM macro.
- The SDWA is freed if it exits.
- The ERA is freed if it was acquired via GETMAIN.

If the emergency ERA is being used (used when the GETMAIN for an ERA fails) bit TRCAEEIU is turned off to indicate that the emergency ERA is now available for use. The subroutine then returns to its caller indicating that the ERA was released.

HASPNPM: The Network Path Manager

The network path manager establishes direct connections and reacts to disconnections among the various members of a network job entry (NJE) network. Additionally, the network path manager ensures that all directly-connected members have a complete and consistent view of the network by distributing connection/disconnection information. The network path manager also provides other sub-components of JES2 with path information. The network path manager sets the address of desirable line device control tables (DCTs) (or dummy DCTs for paths across the multi-access spool connections), which may be used to reach a particular NJE member node, into the path information area of the node information table element used to represent the node within JES2. Along with this address the network path manager sets the path resistance, which is used to determine desirability of transmitting via the path. Correspondingly, each NJE line DCT contains a matrix that is used to indicate to transmission processors the desirability of reaching a given node via the line. Each node in the network is represented by an unique bit in the line matrix. If a node is to be reached via the line, the bit corresponding to the node is 1; otherwise, the bit is 0.

Network Path Manager Connection Control Protocols

The protocol for directly connecting two job entry subsystems depends upon the capabilities of the two network path managers in the subsystems, the number and relative speed of lines between the two systems, and the installation-supplied names of the nodes. Each protocol is described in the following paragraphs.

A unique connection in an NJE network has four basic parts:

1. The identification of the system with the low EBCDIC name.
2. The identification of the system with the high EBCDIC name.
3. The connection event sequence (CES).
4. The resistance of the connection.

The CES is a binary value that increases each time the low end system initiates or allows a connection. Because the value is ever increasing, network path managers can decide what information is the most recent and discard any old connection information. When CES values are assigned, the network path manager ensures that the sequence does not go above the current time-of-day (TOD) clock value; therefore, the value could possibly overflow in 143 years from the base TOD clock time (same as TOD clock overflow). Because the low end determines the CES, protocols vary depending upon which end initiates a connection. It should be noted that even though a line may be leased, no assumption is made that a particular node is at the other end until it identifies itself via the connection protocol.

The following transmissions are required when the low end initiates the connection. Note that this sequence will occur only for connections via a BSC line or via a channel-to-channel adapter supported as a BSC line. Connections via SNA sessions are always initiated by the high end. After RTAM at the low end sends SOH-ENQ and RTAM at the high end responds with ACK0, the following records are sent by the two path managers involved in the connection:

1. Low end sends initial sign-on.
2. High end sends response sign-on (CES 0).
3. Low end sends reset sign-on (CES set).
4. High end sends concurrence sign-on.

Note that the low end cannot concur with a primary connection (the connection with the least resistance); that is the responsibility of the high end.

The high end initiated connection permits a slightly abbreviated protocol. After an SNA session is established by RTAM through VTAM (for an NJE connection via an SDLC line) or after RTAM at the high end sends SOH-ENQ and RTAM at the low end responds with ACK0 (BSC connection), the following path manager records are sent:

1. High end sends initial sign-on.
2. Low end sends response sign-on (CES set).
3. High end sends concurrence sign-on.

A secondary trunk is a line directly connecting 2 systems already directly connected when the secondary connection is made; the new line resistance is not less than the original. Because this does not represent a new connection, no CES is assigned and no distinction is made between low or high end.

RTAM protocol for BSC:

1. Initiating end sends SOH-ENQ.
2. Response end sends ACK0.

For connection via an SDLC line, RTAM must establish an SNA application-to-application session through VTAM.

The network path manager records sent for a secondary trunk connection are:

1. Initiating end sends initial sign-on.
2. Response end sends response sign-on.
3. Initiating end sends concurrence sign-on.

If the low end of a connection determines that the primary trunk of a multi-trunk connection is no longer valid, a reset connection protocol is initiated. The trunk over which the reset control record is transmitted is usually the new primary trunk. The CES value is set to indicate primary or secondary. Other conditions may cause a reset to be initiated from either end; however, the high end must never require the low end to answer the reset.

The following lists the transmissions required to perform the reset protocol:

1. Low end sends reset sign-on.
2. High end sends concurrence sign-on.

Predefined connections allow a connection to be known only (private) to the systems connected unless the other members of the network also define the connection. If one of the two directly connected JES2 systems predefines the connection, the other must also.

RTAM protocol for BSC:

1. Initiating end sends SOH-ENQ.
2. Response end sends ACK0.

For connection via an SDLC line, RTAM must establish an SNA application-to-application session through VTAM.

The following lists the path manager transmissions required to complete a predefined protocol:

1. Initiating end sends initial sign-on.
2. Response end sends response sign-on (CES=X'FFFFFFFF').

Whenever a dynamic connection is agreed upon, each network path manager involved sends an add connection control record to systems not involved in the connection over all other NJE lines. The add connection control record is used by receiving network path managers to determine acceptable paths to nodes within the network. Each network path manager forwards the add connection control records. If a connection is already known (CES indicates the new control record received is not new), the record is ignored.

Disconnections are promulgated to the members of the network using a subtract connection control record. Disconnections may cause nodes formerly reachable via the disconnected line to be no longer available to the system. In this case, dependent connections are automatically determined by each system experiencing the disconnection or receiving the resulting subtraction control records.

Add and subtract connection control records may be blocked in the transmission buffer with reset and concurrence control records. This is quite common when a new trunk is established and complete pictures of the network are traded by the systems involved, or when a disconnect is received by a job entry subsystem and a reaffirm connection notification group is transmitted.

Nodes Attached Table

The nodes attached table (NAT) contains elements representing members of the network that are or were at one time attached directly via an NJE connection, that is, by the NAT elements. Address pointers to the various sections of the NAT discussed below are maintained in the processor control element (PCE) for the network path manager (NPMPCE). At the beginning of the table (located by NPMNAT address pointer), NAT elements representing currently attached members and predefined connections are represented (the last element is located by NPMNATA address pointer). Following the active elements are NAT elements representing unattached or unconnected members that were formerly connected (the last element is located by NPMNATU). Unconnected NAT elements are followed by NAT elements representing connections in process and are held pending full connection or disconnection. NAT elements have a primary (NATPRI) and secondary (NATSEC) member identifier which uniquely identifies the JES2 job entry subsystems involved in the connection. These identifiers, along with a connection event sequences (CES) supplied by the system involved in the connection with the lowest EBCDIC node name, uniquely represent the occurrence of a connection or disconnection, that is, each new connection will have a higher CES value than the previous connections, and corresponding disconnections will have the same CES value as the connection. One exception to this rule is predefined connections, which always have a CES of X'FFFFFFFF' and are never promulgated across the network. NAT elements also contain a resistance value that roughly represents the degree of difficulty the two subsystems have in transmitting NJE data sets.

Other fields within the NAT element are used to control the notification of other JES2 job entry subsystems about connections and disconnections and to locate the device control table (DCT) that should be used to communicate with a directly connected NJE member. NAT elements that represent direct connections differ from other

NATs in that the secondary member identification identifies the JES2 subsystem's own node and multi-access spool member, and the primary identifier identifies the member at the other end of the connection.

Network Path Manager Organization

The network path manager is made up of a JES2 processor controlled by the processor control element (PCE) dispatching techniques and several service subroutines called by other processor sub-components. The following paragraphs describe the routines used by the network path manager processor.

NPL: Examine Input from an NJE Line Routine

The network path manager processor begins processing by examining the $NPMINQ queue for a BSC buffer. Any such buffer has uncompressed images of network connection control (NCC) records. The current logical record is located, and the appropriate service routine to process the record is entered. If there are no BSC buffers in the $NPMINQ queue, the network path manager examines the $NPMVINQ queue at label NPLSNA for an SNA buffer. If none is found, a branch is taken to NPNI. If an SNA buffer is on the $NPMVINQ queue, a pointer to the line device control table (DCT) is obtained from the request parameter list (RPL) and a pointer to the data portion of the request unit (RU) is loaded into register 1. A branch is then taken to NPLNGRSP.

NPRL: Respond to Sign-on Routine

The response to sign-on routine begins by examining the NPMRESPQ queue head in the PCE to determine if responses are needed. (Line DCTs are queued to the NPMRESPQ when initial sign-on network connection control (NCC) records are received over the line and responses are required.) If the queue is empty, an exit is taken to the full path routine (NPNR).

If this JES2 is a member of a node with an EBCDIC name lower than the connected member's node name, the line is to be the primary trunk (preferred line), and the connection is not predefined, a new connection event sequence (CES) is obtained to represent the connection. The CES is an incremental value added to the high order word of a reference time-of-day (TOD) clock value, which is not allowed to go higher than the current clock value. If the CES attempts to go higher than the leftmost work of the TOD clock, the processor waits ($WAIT ABIT) and loops to the top of the processor; otherwise, the value is saved in the line DCT.

The NPNGBF subroutine is called to get a spare transmission buffer and allocate space for the response sign-on NCC record. The record is filled out using values from the subsystem's own node information table (NIT) element, line DCT, and system parameters. The buffer size to be used for transmission between the two NJE members is set in the NCC record. It is the size of this member's transmission buffer or the size of the connected subsystem's transmission buffer, whichever is smaller.

The line DCT is removed from the NPMRESPQ queue, and the NPNWRT subroutine is entered to queue the buffer for transmission. If the connection is not a prefixed connection, the sign-on routine is reentered at label NPRL to process more requests; otherwise, the DCT is queued to the end of the NPMACTL queue, indicating the line is an active NJE line and may receive all NCC records. Control is returned at label NPL, where processing continues.

NPNR: Full Path Routine

The full path routine begins processing by testing the full path request flag (NPMFLAGP) in the PCE. If the flag is off, no full path determinations are required, and the send notifications routine (NPNP) is entered. Otherwise, at label NPATHL, all NIT elements and line DCTs are set to reflect that no nodes are connected.

At label NPATHENX, the nodes attached table (NAT) control pointers are examined to determine if any connections are currently active. If not, the NPMFLAGP flag is reset, the rest of the full path process is skipped, and the send notifications routine is entered at the NPNP routine.

At label NPATHLL, all active NAT elements representing direct connections are collected at the beginning of the NAT table and corresponding DCT addresses and resistances are set into the primary path fields of the corresponding NIT entries. The last direct-connection a NAT element location is saved in NPMNATA so that path determination can be stopped when all lines have been serviced.

At label NPATHML, all direct-connection NAT elements beginning with the first are scanned to determine multiple-tree structures representing members of the NJE network that are connected by a desirable path via the line upon which the direct NJE connection is made.

At label NPATHNMA, the processing for a given direct-connection on NAT element begins by placing its address in a temporary stack (NPMSTACK). Using the NATPRI field as the base of the search for the next NAT element, attached NAT elements are scanned until a NAT element meeting both of the following criteria is found:

- The NATPRI or NATSEC identification in the NAT element matches the identification used as the base of the search.
- A pointer to the NAT element has not been placed in the temporary stack.

The path resistance is incremented by the value in the matching NAT element. If the resulting resistance value exceeds the value of RESTMAX (on the NJEDEF statement), the end of the path is assumed. Otherwise, an attempt is made to place the line address in the NIT element for the outboard node - the subsystem represented by the NAT element identifier not matching the base of the search.

At label NPATHN, the outboard subsystem's identifier is made the new search base and its NIT element is located. If the current resistance is less than the first path element resistance within the NIT element or not greater than the path resistance plus the tolerance specified by the value of RESTTOL, the line and resistance are inserted in the path elements in ascending order of resistance, unless insufficient path elements are provided in the NIT element. If the same line is encountered with a path resistance not greater than the current resistance during the attempt to insert the current line as a possible path, the attempt is aborted. If the same line is encountered with resistance greater than the current resistance, the old entry for the line is removed. If the line becomes the best path (has smallest resistance and is placed in the first path element of the NIT element), any path elements with a resistance above the allowed tolerance are removed. If no modification to the NIT element paths is made, the end of path is assumed because the connections have already been processed for another limb of the tree. If a modification is made, the address of the NAT element is placed in the temporary stack, and the search for more NAT elements continues.

If the NAT element represents a multi-access spool connection (that is, if the NATPRI and NATSEC identifiers represent members of the same node), the NIT element update is assumed, the NAT element address is stacked without an update attempt, and processing continues as though the update occurred.

At label NPATHNM during the stacking of NAT elements, the NAT element is moved to a position in the table adjacent to direct-connection NAT elements or elements representing previously verified attached connections. NAT elements with connection matches not used to update the NIT path elements are also moved to the verified section.

When an end of path is encountered at label NPATHNX, the resistance is reduced by the resistance value in the last encountered NAT element, and the subsystem identifier used to find the element is reinstated. The search continues with the NAT element following the rejected element. When the end of all attached NAT elements is encountered, the last stacked NAT element is unstacked. The search continues for the NAT element following the unstacked element, using the reinstated base of the previously stacked NAT element. In performing the unstack, the direct-connection NAT element is unstacked and processing for the line is complete. Succeeding direct-connection NAT elements are processed until all have been processed.

At label NPATHRL, (when all paths have been determined), any NAT elements remaining within the attached connection area of the NAT that have been verified and that are not predefined connections are removed from the attached portion of the table and placed in the unattached (or unconnected) portion. These removed NAT elements represent connections between network members, which may still be valid for the systems involved but, because other connections were broken, neither NJE member is now reachable. The CES values for these NAT elements are set to 0. When that portion of the network is again reachable, the notification records received over the network can be recorded. The notification of connection or disconnection is suppressed because the status of the connection is unknown.

At label NPATHP, the desirable path information in the NIT elements is promulgated to all affected line DCTs so that transmitters can select jobs and data sets for transmission. The NPMFLAGP flag is reset at label NPATHX.

NPNP: Send Notifications Routine

The send notifications routine begins processing by testing the NPMFLAG flag in the PCE. If the flag is off, no notifications are required, and the end of processing routine (NPNN) is entered. If the flag is set to one, the NPNP routine locates the line DCT zero, which is the location of the queue head minus the distance between the DCT origin and the active queue chain field.

If there are more active NJE line DCTs, the next DCT is made the current DCT for processing at label NPNDCTL; otherwise, the NPMFLAG flag is reset and the end of processing routine (NPNN) is entered.

If the current DCT is a dummy DCT, the connection represents a connection across the multi-access spool job queue, and the routine gives control to the processing at label NPND.

If the current DCT flags are set to indicate that a reset sign-on NCC record is to be transmitted over the line, the NPNPUT subroutine is used to obtain space in a teleprocessing buffer; a reset sign-on NCC record is created and the network flag in the DCT (MDCTNFL) is set to indicate that a concurrence record is expected.

If the current DCT flags are set to indicate a concurrence sign-on NCC record is to be transmitted over the line, the NPNNE routine calls the NPNPUT subroutine to obtain space in a teleprocessing buffer; a concurrence sign-on NCC record is created and MDCTNFL is reset to indicate that no further activity is required. Reset and concurrence sign-ons are not sent over the same line for the same pass through the NPNP routine.

At label NPNOT, all connected and unconnected nodes attached table (NAT) elements are scanned. For each element encountered, a test is made to determine if the line is the next line requiring notification of the connection or disconnection. If it is, the NPNPUT subroutine is used to obtain space in a teleprocessing buffer, an add or subtract connection NCC record is created, and the NAT element is updated to indicate that no notification has been transmitted over the line. When all NAT elements have been processed, the NPNWRT subroutine is called to cause queuing of the last partially filled buffer (if any), and control is passed to label NPNDCTL.

If the current DCT flags are set to indicate that a reset sign-on NCC record is to be transmitted over the multi-access spool connection, the processing at label NPND uses the NPNDG subroutine to obtain space in a spool buffer; an abbreviated internal form of the reset sign-on NCC record is created (actually a portion of the NAT element), and the flag (MDCTNFL) is set to indicate that no further activity is required.

All connected and unconnected NAT elements are scanned. For each element encountered, a test is made to determine if the multi-access spool connection is the next line requiring notification of the connection or disconnection. If it is, the NPNDG subroutine is used to obtain space in a spool buffer, and portions of the NAT element are moved into the buffer. If the buffer is filled for a given NAT element, the NPNDQ subroutine queues the buffer to the remote console processor for eventual spooling with the message data that indicates the appropriate member of the node. The NAT element is updated to indicate that the member has been notified.

When all NAT elements have been processed, a test is made to determine if a spool buffer is partially filled. If it is not, control is passed to the NPNN routine; otherwise, the NPNDQ subroutine is used to queue the buffer for writing, and control is passed to the NPNN routine.

The end of processing routine issues a $#POST TYPE=XMIT if any network path manager routine has determined that a new path has been made available for transmission of jobs or SYSOUT data sets to any node currently in the network. The processor waits ($WAIT) for work and when posted ($POST), control is passed to the beginning of the NPL routine.

Network Path Manager Processor Input Record Handlers

The following describes the routines used by the input record handlers of the network path manager processor.

NPRI/NPRR: Receive Initial and Response Sign-on Routine

When the network path manager processor receives an initial or response sign-on network connection control (NCC) record, the receive initial sign-on routine (NPRI) or receive response sign-on routine (NPRR) is entered. (NPRI and NPRR define the same location.) The addresses of the current line device control table (DCT) and buffer are given to the routine by selection processing performed at the beginning of the processor.

If NJE activity has already begun on the current line, the record is ignored, the NPBINR subroutine is used to release the input teleprocessing buffer, and control is passed to the NPL routine.

If certain error conditions are detected, sign-on is rejected, a diagnostic is displayed on the operator's console, flags are set to cause line restart, and the NPBINR subroutine is used to release the input teleprocessing buffer. Control is passed to the NPL routine. The error conditions and associated diagnostic messages that result in this execution sequence are:

- The current line is not transparent ($HASP500).
- No NJE function control DCTs are available ($HASP501).
- The required passwords do not match ($HASP502 or $HASP504 denoting invalid line and node passwords, respectively).
- The node names are incorrect ($HASP503).

At level NPRINDL, a temporary nodes attached table (TAT) element is created representing the connection, and various fields in the current DCT are set:

- TATSEC is set to this system's internal NJE identifier $SYSID.
- TATPRIN is set to the remote system's node number.
- TATPRIQ is set to the remote system's node qualifier (member number).
- TATREST is set to the total node-to-node resistance.
- TATALINE is set to address the current line.
- TATNLINE is set to indicate that the last modified line was the active line DCT zero, which is the address of the queue head minus the difference between the beginning of the DCT and the chain field.
- MDCTNCES is set to the connection event sequence number received.
- MDCTRAT is set to address the associated node information table (NIT) element.
- MDCTBFSZ is set to the maximum buffer space available in the teleprocessing buffer for normal compressed records, determined by the smaller of this system's buffer size or the remote system's buffer size.
- MDCTNFLL bit is set to indicate that this system is responsible for the connection if it is determined by a compare logical character (CLC) instruction that the EBCDIC name of this system is lower than the EBCDIC name of the remote system.

The NPBINR subroutine is used to release the buffer.

If the NCC record is a response sign-on, the current line DCT is placed on the NJE active queue, a concurrence is tentatively scheduled, and notifications are requested (NPMFLAGN is set).

If the NCC record is an initial sign-on, processing at label NPRISES places the current line DCT on the NPMRESPQ queue as input to the receive response sign-on routine, NPRR, and sets the MDCTNFLI bit.

At label NPRIQD, the nodes attached table (NAT) is scanned for an element that matches the TAT element. If there is no match, a new NAT element is added to the table, and the line is designated as the primary trunk of the connection. If the remote end has indicated a connection event sequence (CES) of X'FFFFFFFF' (predefined connection) and no NAT element is found, the systems are using incompatible connection protocols. A diagnostic message ($HASP506) is displayed, the current line is restarted, and control is passed to the beginning of the processor (NPL). If the NAT is not large enough to accommodate a new NAT element, a diagnostic message ($HASP507) is displayed, the current line is restarted, and control is passed to the beginning of the processor (NPL). If a NAT element is found by the NAT scan, control is passed to the processing at label NPRINATF.

If the concurrence is not required at this point in the processing, required processing of other routines is needed; therefore, control is passed to the beginning of the processor. Otherwise, the TATEVNT field is set to the current CES.

At label NPRIESET, the JES2 responsible for the connection forces the scheduling of a reset sign-on if the trunk is the primary one. If multiple trunks are involved and the current resistance is better than the previous primary line resistance, the new line is made primary. Control is passed to the beginning of the processor. The JES2 not responsible for the connection passes control to the beginning of the processor if the remote system indicates that the current line is not to be a primary trunk (CES 0 received in the NCC). Otherwise, the line is made the primary trunk.

At label NPRICONC, the NAT element representing the connection is replaced by the TAT element and moved to the actively attached portion of the NAT. Notifications and installation-submitted job (JOB) and JOT posting are requested.

At label NPRINOT, the primary path of the affected node information table (NIT) element is examined. If no paths are currently set in the NIT element, the element and current DCT are set to allow transmission of work across the line and the line is posted (via $#POST); otherwise, the full path process is requested (NPMFLAGP is set). Control is passed to the beginning of the processor.

Processing at label NPRINATF determines the status of the connection provided that a NAT element is found at label NPRIQD. If the NAT element is not in the active portion of the NAT and the system at the other end of the line indicated the connection is predefined, a diagnostic message ($HASP506) is issued, the line is restarted, and control is passed to the beginning of the processor. If the NAT element is in the unconnected portion of the NAT, it is moved to the held portion, and control is passed to the processing at label NPRINATA. Line DCTs with NATs that do not meet these criteria are placed on the multi-trunk queue as secondary trunks and the line is posted (via $#POST). If the connection is not predefined, control is passed to the processing at label NPRINATA; otherwise, the sign-on process is complete (the response sign-on NCC is currently scheduled for transmission). Control is passed to the beginning of the processor (NPL).

NPRC/NPRE: Receive Concurrence and Reset Sign-on Routines

When the path manager processor receives a concurrence or reset sign-on network connection control (NCC) record, the receive concurrence routine (NPRC) and reset sign-on routine (NPRE) are entered. The current line status is checked to ensure that acceptable initial or response sign-on records were received. If such records were not received, a diagnostic message ($HASP506) is issued, the line is restarted, and control is returned to the beginning of the processor (NPL routine).

At label NPLA, if the record is determined to be a concurrence sign-on with a connection event sequence (CES) mismatch, the record is ignored, and control is returned to the beginning of the processor (NPL routine).

For reset sign-on records, the line is restarted if the resistance totals are invalid; a diagnostic message ($HASP505) is issued in this case. The record is ignored when the CES is less than the previously recorded CES and a concurrence sign-on is requested.

At label NPRCPI, current line DCT and temporary nodes attached table (TAT) elements are initialized. If required, the current line DCT is placed on the active NJE line queue (if not already on the queue), the line is posted (via $#POST), and the initial and response sign-on routine is entered to schedule concurrence or reset sign-on responses.

NPRA/NPRS: Add and Subtract Connection Routines

When the path manager processor receives an add or subtract connection network connection control (NCC) record, the add and subtract connection routines (NPRA/NPRS) are entered. The connection record information is converted to internal form and set into a temporary nodes attached table (TAT) element. If an error occurs during the conversion, a diagnostic message ($HASP503) is issued and the record ignored.

The TAT element is used to locate a nodes attached table (NAT) element. If no NAT element is found, the NAT is expanded, and the TAT element is used to create the NAT element in the connected or unconnected portion of the NAT. If the new NAT element is placed in the unconnected portion of the NAT, a request for notification and full path is made. Control then passes to the beginning of the processor. If the new NAT element is placed in the active portion, either an attempt to update the outboard NIT element (at label NPRAQIK) is made, or a full path is requested. A full path will be requested when the outboard node is involved with predefined connections, when a full path was already requested, or when other paths to the node exist. Control then passes to the beginning of the processor.

Network Path Manager Processor Subroutines

The following describes the major subroutines used by the network path manager processor.

Send Network Connection Control (NCC) Record Subroutines: These are a collection of interrelated subroutines used to get teleprocessing buffers, obtain space for NCC records, and queue buffers to the line for transmission. Functions performed for BSC-type buffers are as follows:

- NPNPUT: This routine is used to obtain space for an NCC record in the buffer. If necessary, buffers are queued for transmission via the $EXTP WRITE macro, and buffers are obtained using the $GETBUF macro instruction.
- NPNWRT: This routine is used to queue a buffer for transmission; optionally it is used to get a new buffer and allocate space for an NCC record.
- NPNGBF: This routine is used to get a new buffer; optionally it is used to allocate space for an NCC record.

Functions performed for SNA-type buffers are as follows:

- NPNVPUT: This routine is invoked when the LINE DCT is specified by MDCTTYPE=SNA. If a request parameter list (RPL) is not available, a branch is taken to NPNGVBF to obtain a RPL. If a RPL is available, this routine picks up the current count and points to the data area. At label NPNVPUTA, the routine adjusts the count to include the length of the string control byte (SCB) and count requested by the caller. NPNVPUT branches to NPNWRT if the buffer is full; otherwise, it saves the adjusted count in RPLRLEN, updates the pointer to the next logical record, and stores the pointer in RPLAREA. This routine sets up an SCB in the first byte of the current record, sets the pointer to the current logical record plus one in register 1, and returns to the caller.
- NPNGVBF: This routine is called to get an RPL. It gets a VTAM-type buffer via $GETBUF; if no RPL is available, it waits ($WAIT) and tries to process more input via the NPL routine. When an RPL is available, it copies the RPL into register 4, initializes the RPL clear count register, branches to label NPNVPUTA, and continues processing.

Input Teleprocessing Buffer Handling Subroutines

These are a collection of interrelated subroutines used to step through an input buffer containing network connection control (NCC) records and release the buffer for more input. Functions performed are as follows:

NPGET Routine: This routine is used to step through a buffer of NCC records. If this is an SNA NJE buffer, this routine branches to NPVGET, which provides the comparable function for an SNA buffer; otherwise, if the end of the BSC buffer is reached, the NPBINR subroutine is entered to release the buffer.

NPBINR Routine: This routine is used to remove the BSC buffer from the $NPMINQ queue and release the BSC buffer via the $EXTP READ macro. If this is an SNA NJE buffer, this routine branches to the NPVBINR subroutine, which removes the SNA buffer from $NPMVINQ before issuing the $EXTP READ macro.

NPNDG Subroutine: NPNDG is used to obtain a spool buffer and initialize it for later queuing and transmission across the multi-access spool connection. If the buffer is not obtainable, the processor waits for the buffer, and control is passed to the beginning of the processor.

NPNDQ Subroutine: NPNDQ is used to queue a spool buffer to the remote console processor for transmission across the multi-access spool connection and to post ($POST) the remote console processor.

NPFPAN Subroutine: NPFPAN is used to scan the multi-trunk queue for a given NJE line device control table (DCT) and to locate the primary trunk DCT. In addition, NPFPAN locates the nodes attached table (NAT) element that represents the connection to the system at the opposite end of the line.

NPEVENT Subroutine: NPEVENT is used to obtain the next connection event sequence (CES). If the sequence gets ahead of the last time-of-day (TOD) clock value (leftmost word) recorded, a new TOD clock value is obtained. If the value is still ahead, an error exit is taken so that the caller can wait for the TOD clock to catch up. The initial value of the CES and the TOD clock is set by JES2 initialization. For the system to get ahead of the TOD clock, an operator would have to initiate connections faster than one per second for a considerable period of time.

Network Path Manager Services to Other Processors

The path manager provides service routines for other processors. These routines are discussed in the following paragraphs.

HASPNDCN: Line Disconnection Processing Routine

The HASPNDCN routine is entered when an NJE line has been disconnected. When entered, it clears the information concerning reachable nodes from the device control table (DCT) of the disconnecting line. If the DCT is in the path manager processor's NPMRESPQ queue, it is removed (label NPDISRL). If the DCT is on the active NJE line DCT queue, it is removed.

HASPNDCN scans the nodes attached table (NAT). If any NAT element indicates that the disconnecting line was the last line notified, the NAT element's notification reference is backed up to indicate the previous line on the active NJE line queue. If any element indicates that the disconnecting line is the line upon which notifications were received, the NAT element's active queue head address (DCT zero) is set into the line ownership field. New primary trunks are assigned if required.

At label NPDISC, the disconnecting DCT is removed from the multi-trunk queue, and the path manager processor is posted ($POST). Control is passed to the remote console processor line disconnect routine (HASPMDCN).

HASPNMUP: Locate DCT and Create NAT Routine

The HASPNMUP routine is entered when the checkpoint processor determines that another member of the node is now ready for NJE messages and control record communications. When entered, the routine locates the dummy NJE line DCT corresponding to the member indicated. If the DCT is already active, the request is ignored and control is returned to the caller.

The DCT is set up to resemble a primary trunk line DCT. A NAT element is created if required and placed in the held portion of the NAT. If the connection is to a higher numbered member of the configuration, the DCT is placed on line DCT queue and set to require a reset sign-on network connection control (NCC) record. Notifications are requested and the path manager processor is posted ($POST). Control is returned to the caller at normal return.

If the NAT is not large enough for a new NAT element, which may be required, control is returned to the checkpoint processor, which issues a diagnostic message ($HASP262).

HASPNMDN: Node Inactive Processing Routine

The HASPNMDN routine is entered when either the checkpoint processor or remote console processor determines that the designated member of the node is no longer active for NJE. When entered, the routine locates the dummy line DCT corresponding to the designated member. If the DCT is not active, the request is ignored and control is returned to the caller. The process of line disconnection is simulated (see HASPNDCN routine above) and control is returned to the caller.

HASPNBUF: Simulate Reading of Add/Subtract NCC Records

The HASPNBUF routine is entered when the remote console processor reads a spool buffer containing abbreviated network connection control (NCC) records from another member of the node. This routine simulates the reading of add and subtract connection NCC records. (See "NPRA/NPRS: Add and Subtract Connection Routine", earlier in this section.) A full path is always required when records are found that have meaning. The path manager processor is posted ($POST) and control is returned to the caller. If the NAT overflows during processing, an error return is taken. If the line is not flagged as up, the buffer is ignored. If the connection received is between sender and receiver, a NAT element is made active.

HASPNSNR: Examine Line and System Conditions

The HASPNSNR routine is entered by the command processor when the operator enters an acceptable $SN command to start NJE communications on an already active BSC line. For SNA sessions, this routine is entered from the MNJENPMC routine in HASPSNA after the SNA session has been successfully established. Also for SNA sessions, the call is always made at the node that has a node name higher, in the collating sequence, than that of its session partner.

When entered, HASPNSNR obtains a buffer and queues it for transmission via the $EXTP WRITE macro. If this is a BSC line, specific editing is performed, a resistance value is determined, a $GETBUF TYPE=BSC macro is issued to obtain a BSC buffer, and processing continues. If this is an SNA line, a $GETBUF TYPE=VTAM macro is issued to obtain a VTAM buffer. If the requested buffer type is not available, an exit is taken with a return offset of 12.

If the requested buffer type is available, an initial sign-on record is built. If a VTAM buffer is being processed, the request parameter list (RPL) information is set up and RTAM is called via $EXTP WRITE to transmit the buffer. If a BSC buffer is being processed, a BSC buffer is initialized and RTAM is called via the $EXTP WRITE macro to transmit the buffer. If RTAM returns with an error, a $FREEBUF macro is issued and the routine returns to the caller at offset +8.

The return codes upon exit from HASPNSNR are as follows:

Exit	Command
+0	Successful completion
+4	$S required
+8	Remote or node is already assigned to line
+12	No device control tables (DCTs) or buffers are available or the connection event sequence (CES) is ahead of the time-of-day (TOD) clock
+16	Line not transparent

HASPNET: The Networking Support Module

HASPNET handles network SYSOUT reception, SYSOUT transmission, job transmission, and the associated authority checking.

SYSOUT Reception Processor

The NJE SYSOUT receiver is responsible for receiving a job's generated system output (SYSOUT) data sets, writing it to the spool file, and preparing the received job for local output or further NJE transmission to other nodes. The SYSOUT receiver, through HASPRTAM, reads the records that are transmitted by the NJE SYSOUT transmitter.

Each line that is currently being used for NJE has associated with it one (or more, depending on the SRNUMR initialization parameter) SYSOUT receiver device control table (DCT). There is one SYSOUT processor control element (PCE) receiver for each of the device DCTs; thus each receiver PCE is associated with one and only one device DCT. For each SYSOUT receiver, there is a corresponding SYSOUT transmitter at the node on the other end of the NJE line.

HASPNSR (SYSOUT Receiver Main Entry Point): HASPNSR performs authorization checking during data set header processing and PDDB initialization. A token spool control record (SCR) is placed at the beginning of each data set buffer to contain data set header SCRs. The SCR is initialized and the token copied into it when a receivable data set header is found and again when a new buffer is needed.

Note: A token is copied into the SCR only if one was received.

When the security section is located, HASPNSR calls NSRAUTH to obtain a token for the data set, verify authority, and build a data set name. NSRAUTH passes a return code of 0, 4, or 8 back to HASPNSR, which processes as follows:

- 0 - continue data set initialization.
- 4 - call exit 39 to determine whether to continue initialization or delete the data set (default).
- 8 - call NSRTERM to purge the job (due to subtask failure). The receiver is shut down and the line drained.

If any data sets are to be purged (or received) locally, NSRNOTFY issues the $HASP546 message during EOF processing. At NSREOF, all of the data sets have been processed and the job trailer read. If any of the data set headers were selected, NSRAUTH is called to get a token to protect the job. (No data set build/create authorization is necessary.) If NSRAUTH returns 4 or 8, the job is purged.

NSR$WAIT: When the SYSOUT receiver is initially entered, it waits ($WAIT) for work. It will be posted ($POST) by HASPBSC or HASPSNA when the transmitter on the other end of its line requests permission to transmit.

NSR$UNIT: When this request occurs, the SYSOUT receiver issues the $GETUNIT macro to obtain control of its device DCT.

NSR$NCL: If the $GETUNIT is unsuccessful (for example, if the operator has drained the device), the SYSOUT receiver denies the transmitter's request. It does this by issuing a $EXTP NCLOSE (negative close) macro to cause a negative acknowledgment to be sent to the transmitter. The SYSOUT receiver then returns to wait ($WAIT) for work.

NSRSETH: If the $GETUNIT is successful, the SYSOUT receiver prepares to receive SYSOUT data for a job. It uses the $QADD macro to add a dummy (empty) job queue entry (JQE) to the $RECEIVE queue to ensure that a JQE is available for the job that is to be received. If no JQE can be added (the job queue is full), the operator is informed via message $HASP547, and a negative acknowledgment is sent to the transmitter via $EXTP NCLOSE. If the JQE is successfully added, a positive acknowledgment is sent to the transmitter via $EXTP OPEN, and the receiver branches to its main GET loop.

NSR$GET (Main GET Loop): The SYSOUT receiver first checks flags in the DCT and JQE to see if any operator commands have been issued against the receiver's device or job. If any have been issued, they are handled immediately. If no commands have been issued, the receiver issues a $EXTP GET macro to obtain the next logical record. If the GET ends abnormally, the receiver branches to abort its job.

If an invalid record is received in the middle of processing an input buffer for the spool offload facility, and a new job (header) is expected, a skip bit is set in XDCTSKIP). When NSR$GET issues the next $EXTP GET, the current buffer is discarded, and a new buffer is read. Buffers will be discarded until the next job is received. If the GET does not end abnormally, the GET completion code and the record's subrecord control byte (SRCB, returned by $EXTP GET) are checked to determine what type of record was received (job header, job trailer, data set header, data record, end-of-file). The received record type is checked against flag settings that indicate which types of record the SYSOUT receiver expected to receive. If the received record was not expected, the receiver branches to abort its job. Otherwise, the receiver uses a branch table (NSRBRTAB) to branch to a routine that handles the type of record that was received.

NSRJOBH: An NJE job header is only expected once per job and must be the first record to be transmitted for the job.

Verification checking is done here both when reading in the job header and at job termination with the job trailer. The time and date are compared with the time and date in the offload DCT (taken from the record descriptor). If there is a match, processing continues. If there is a difference, the job in error is purged, the load operation terminated, and the offload device drained. If a job or SYSOUT have been dumped from a previous release of JES2, there is no verification stamp. SYSOUT receivers detect this condition and pad the header with binary zeroes so that the header and the record descriptor match and processing continues. When a job header is received, the receiver prepares for receiving the job's data sets. Two spool buffers are obtained, one for the job's job control table (JCT) and one for the allocation input/output table (IOT). The JCT is constructed from information in the job header record, and the allocation IOT is initialized. The allocation IOT is used as a parameter for the $TRACK macro to obtain spool space for the JCT and IOT, and the two blocks are written to the spool file. The receiver fills in the job queue element (JQE) for the job (obtained earlier) and checkpoints the updated JQE.

The job header record contains the job ID (a halfword binary number) that was originally assigned to the job when it entered the NJE network. The SYSOUT receiver calls the $QJIX routine in HASPNUC to assign a job number and add the JQE to the JIX.

NSRFIND2: JES2 requires that each job's NJE job header contain a JES2 subsystem section so that the JCT can be properly initialized. If the incoming job does not have a JES2 section then JES2 will build a default JES2 section. The receiver stores the job header in the job's JCT for subsequent use by the SYSOUT transmitter.

To save system affinity information across spool offload operations, this information is moved between the JQE and the job header, depending on the operation.

When the receiver has completed initialization for the job, it issues message $HASP540 to inform the operator that the job is being received. It then sets flags to indicate that the next record to be read must be either a data set header or a job trailer and branches back to the main GET loop.

NSRDSH: When a data set header is received, the SYSOUT receiver performs the initialization required before receiving a data set. If the record received before the data set header was a data record, the receiver terminates the previous data set by writing out to spool its last spool buffer with a zero chain field in HDBNXTRK and sets the end-of-block indicator after the last record in the block.

The SYSOUT receiver then obtains spool buffers to hold the data set header and the data set records themselves. The SYSOUT transmitter may send more than one data set header before sending the data records. In this case, each data set header indicates a different destination for the same data set or copies of the data set are to be produced with different output processing requirements (JCL OUTPUT statement). The receiver stores all data set headers for the data set on the spool file for subsequent use by the SYSOUT transmitter. The data set headers are stored in spool blocks separate from the data set itself; the first record of the first block of the data set is a spool control record (SCR) which contains pointers to the spool blocks containing the data set headers. The SCR appears to be a print record that is marked both null on input and null on output. It is ignored if the data set is either printed/punched by HASPPRPU, by a printer under a functional subsystem, or by a installation-written writer or displayed by TSO.

After the data set header is read in, a call is made to work selection services if this is an offload SYSOUT receiver. If NSRDSH receives a return code of 4 from work selection services, a PDDB is created and the spool offload modification routine SRVMOD is HASPSERV is called for this dataset. If the return code is not 4, a PDDB is not created, and the next data set header is checked.

NSRNIOT: If the IOT becomes full, the receiver obtains space for a new IOT, chains it to the preceding one, and writes out the preceding IOT. The first (allocation) IOT is always kept in storage so that its track group map is available for the $TRACK routine. If more than one data set header is received for a data set, more than one PDDB is built in the IOT. Each PDDB points to the same data set on the spool.

Multiple data set headers, and hence multiple PDDBs, can be created if the output processing options for the data set are specified via the JCL OUTPUT card or the /*OUTPUT card. In this case, the output characteristics are passed through the network in the data set header. The data set header consists of a general section and a page-mode section. Optionally, the data set header can also contain a 3800 section as well as installation-generated user sections. The page-mode section contains all of the output characteristics for the data set that were specified on the JCL OUTPUT statement. In addition, if the data set being transmitted contains page-mode data, then the page-mode section contains the page counts for the data set. The 3800 section is created only when the data set represented by the data set header contains SYSOUT characteristics that are applicable to a 3800, such as CHARS or FLASH. A buffer is then obtained to move the output characteristics from the header to the spool. A control block called the SWB information table (SWBIT) is used to place the characteristics section on the spool and a call is made to NSRAUTH to perform authorization checking.

For each PDDB, the SYSOUT receiver on the target node checks for an incoming file that has been sent by the TSO TRANSMIT command. Such a file has an external writer name identical to the target TSO userid; if the target userid identifies a valid user, the SYSOUT receiver on the target node saves the target system id for later use, ensures that exit 13 is enabled, and, at exit point NSRNMXIT (exit 13) of HASPNSR, calls the exit routine. When the exit routine returns, its return code is checked. If a return code of 8 is received, a message is sent to the receiver of the data set. No message is sent to the receiver for a return code of 0 or 4; however a notify message is sent to the originator. Processing continues at label NSRNMEND.

Note: If data is re-routed, the new receiver must issue a TSO RECEIVE command and have the resulting authorization checks performed before being permitted to view the information.

The user exit routine returns a return code of 0, 4, or 8 in register 15 and can alter the PDB1NSOT bit in the PDBFLAG1 byte of the PDDB and the PDBWTRID field in the PDDB to affect subsequent JES2 processing. The PDBWTRID field identifies the userid that is to receive the incoming file from the Interactive Data Transmission Facility. The exit routine can change this id from the original one set by the sender to a different one at the receiving node. Also, the exit routine may set the PDB1NSOT bit to either retain or delete the incoming file. The following table summarizes the JES2 processing following the exit routine's return to the SYSOUT receiver:

Return	PDB1NSOT	Subsequent JES2 Processing Code Bit
0 or 4	off	SYSOUT receiver processing continues. The userid set in PDBWTRID receives the incoming file.
0, 4, or 8	on	SYSOUT receiver issues the $HASP548 message to the user identified by the userid in PDBWTRID. Although the incoming file is still spooled, the message text indicates to the user that the incoming file will eventually be deleted either because of an invalid userid or some other reason.
8	off	SYSOUT receiver issues the $HASP549 message to the user identified by the userid in PDBWTRID.

For all of the above cases, the SYSOUT receiver always issues the $HASP546 message to the sender of the file.

After all data set headers for a data set have been read and processed, the SYSOUT receiver releases the spool buffers it acquired for processing the headers and prepares to receive the data records themselves. The receiver then branches to NSRDATA to process the data set's first real data record.

NSRDATA: When a data record is received, the SYSOUT receiver checks to see if it is a regular data record or a spanned data record. NSRDATA handles regular records, and NSRSPAN handles spanned records. NSRDATA checks whether enough room remains in the current data set spool buffer to hold the received record. If not, the current block is terminated with an end-of-block flag, a new track address is obtained (via $TRACK) for the new data block, and the full block is written to spool with HDBNXTRK pointing to the new block. After the newly received data record is added and data pointers are updated, the receiver branches back to the main GET loop.

NSRSPAN: The spanned record receive routine has more processing to do. The effective record length (LRECL) of a spanned record can be up to 32,767 bytes. Thus, a spanned record not only can be longer than an RTAM logical record, it can be longer than an entire spool buffer. The spanned record is stored in segments, each segment containing some control bytes and being no longer than a spool buffer. The SYSOUT transmitter breaks each segment up into pieces of no more than 256 bytes and sends each piece as an RTAM logical record. The SYSOUT receiver reconstructs the record from the pieces without regard to the original segment boundaries on the transmitting system. The SYSOUT receiver makes maximum use of spool space by constructing spanned segments that fit the spool buffer size of the receiving system. A record that requires multiple spool buffers on the transmitting system may require only one on the receiving system if the receiving system's buffer size is large enough. If so, the received record would be stored as only one segment. The receiver processes each piece (RTAM logical record) of a spanned record, writes out and obtains spool buffers if necessary, sets flags and pointers to indicate that a spanned record has not been completely received, and branches back to the main GET loop to read the next part of the record. When the last part of the record has been processed, the flags and pointers are set to show that the spanned record processing is complete.

NSRJOBT: After all data sets have been transmitted, the SYSOUT transmitter sends the job trailer.

Verification checking is done here both when reading in the job header and at job termination with the job trailer. The time and date are compared with the time and date in the offload DCT (taken from the record descriptor). If there is a match, processing continues. If there is a difference, the job in error is purged, the load operation terminated, and the offload device drained. If a job or SYSOUT have been dumped from a previous release of JES2, there is no verification stamp. SYSOUT receivers detect this condition and pad the header with binary zeroes so that the header and the record descriptor match and processing continues. When the SYSOUT receiver receives the job trailer, it performs termination processing for the job. The last data set is terminated by setting the end-of-block flag after the last record is written to the spool. The IOT is updated and rewritten, and the JCT is updated with information from the job trailer and rewritten. The receiver sets flags to indicate that the next record to be received should be end-of-file and branches to the main GET loop.

NSREOF: When end-of-file is received, NSRAUTH is called to obtain a job token and NSRNOTFY issues the $HASP546 message for any data sets to be purged locally. The SYSOUT receiver waits ($WAIT) for all of its disk I/O to complete, then checks to see if any spin or held data sets were received.

NSRDIOT: If any spin or held data sets were received, the data set's IOT is read. For held data sets, space for a held queue record (HQR) is allocated in the job's held queue table (HQT), and the held IOT is updated and rewritten.

NSRSPIN: For both spin and held data sets, the IOT is passed to HASPXEQ for processing via the SVTSPIOT queue in the subsystem vector table (SSVT). A GETMAIN macro is issued for space in the common storage area, the IOT is copied into the space, and the moved IOT is placed on the SVTSPIOT queue via a compare and swap instruction. A simulated $$POST is then issued to notify HASPXEQ that a spin or hold data set requires processing.

NSRNOTIF: The SYSOUT receiver next checks to see if any of the data sets it has received are destined to be printed or punched on this node. If so, and if the job header contains a user ID, message $HASP546 is issued from the subroutine NSRNOTFY (as is the $HASP589 message) via an NJE $WTO directed to the specified user ID on the job's input node. The JCT is then written out for the last time, and the job is added to the output queue ($OUTPUT) via the $QMOD macro.

If the SYSOUT is loaded from an offload data set, message $HASP589 is issued. If this is not the input node or the job number is not the original job number, the message will include the name of the input node and the original job number.

NSRCLOSE: The SYSOUT receiver sends a positive acknowledgement to the SYSOUT transmitter by issuing an $EXTP CLOSE macro, releases control of its device DCT via the $FREUNIT macro, and branches back to its initial $WAIT to wait for work.

When the job is queued for output, it looks just as if it had completed execution on the receiver's system. The JES2 output processor (HASPHOPE) reads the job's IOTs and creates one or more job output elements (JOEs) to represent the job's data sets on the output queue. The data sets can then be available either for printing by HASPPRPU, XWTR, or for further transmission by the SYSOUT transmitter.

NSRTERM: If at any time the SYSOUT receiver receives an error return from HASPRTAM indicating that the communication line has broken or that the transmitter issued a negative close (NCLOSE), the receiver branches to abort its job. The job is also aborted if an error is detected in any of the received NJE headers or trailers, if records are received out of sequence (an unexpected record is received), if a normal data record is received when a continuation of a spanned record should have been received, or if the operator enters a cancel or restart command.

If the I/O error flag is set on in the receiver DCT, and the data flag indicates some valid data has been received, a message is added at the end of the data buffer indicating some data may have been lost. If the job trailer is not received before the I/O error occurred, this routine builds a default trailer. If no data has been received before the error, the job is purged. This routine issues message $HASP545 in either case.

The job is aborted by writing out its JCT and IOT, issuing a $QPUT macro to add the job to the queue of jobs to be purged ($PURGE), and issuing a $EXTP NCLOSE macro to indicate abnormal termination to the SYSOUT transmitter. In addition, message $HASP543 is issued to inform the operator. The SYSOUT receiver then releases control of its device DCT via the $FREUNIT macro and branches back to its initial $WAIT to wait for work.

SYSOUT Receiver Subroutines

The following subroutines support the SYSOUT reception processor.

NSRAUTH: SAF Subtasking Subroutine: NSRAUTH is called from HASPNSR to subtask a call to SYSOVFY for security authorization. NSRAUTH initializes the $SAFINFO parameter list and passes it to the SYSOVFY routine in HASPSSRV. NSRAUTH issues the $SUBIT macro to subtask the call and process the return codes. SYSOVFY gets a security token to represent the data set, builds the data set name, and issues the data set create call to SAF.

If subtask processing fails, $SEASMSG is called to issue the $HASP077 message and a return code of 8 is set. If authorization is denied, a return code of 4 is set. Successful processing sets a 0 return code. Control returns to HASPNSR.

NSRHRCV: NJE Header/Trailer Receive Subroutine: Because there is no restriction on the length of an NJE header or trailer, a header or trailer may require more than one RTAM logical record for transmission. This subroutine is called whenever the first (or only) piece of a header or trailer has been received. It moves the first piece to the desired location (in the job control table for job header or trailer, in a spool buffer for data set header), and checks to see if the header or trailer occupies more than one RTAM logical record. If so, it issues $EXTP GET for each of the subsequent pieces and reconstructs the entire original header or trailer. As each piece is received, the sequence number of the piece is verified. If an error of any type is encountered, condition code 0 is returned; otherwise, a nonzero condition code is returned.

NSREXPND: Header/Trailer Section Expansion Subroutine: This subroutine is called from the main line of the SYSOUT receiver when a desired section in a header or trailer has been located. The routine checks the indicated section against the required minimum length for this type of section. If the section is too short, the routine expands it to the required length by moving down any data following the section (via the $VFL MVC macro) and padding the section on the end with binary zeros. If the section is expanded, both the section length and the overall header or trailer length are updated.

NSRWHDB1: Data Set Header Buffer Manager Subroutine: This subroutine handles the buffering of received data set headers. As data set headers are received, they are moved to a spool buffer. When a buffer becomes full, this routine is called to write out the filled buffer and obtain a new one for additional data set headers.

NSRNEWB: Buffer End Subroutine: This subroutine is called when a data record has been received and not enough space remains in the current data buffer to hold the record. It is also called at the end of a data set when no more records remain to be received. The current buffer (if any) is terminated by placing an end-of-block indicator (X'FF') after the last data record. If a new block is desired (reception of this data set is not complete), the $TRACK macro is issued to obtain a disk address for the next block, and this address is placed in the HDBNXTRK field of the current block. The current block is then written to spool via the $EXCP macro. Then, if a new block is desired, the $GETBUF macro is issued to obtain a buffer for the new block, and the buffer is initialized.

NSREAD/NSRWRITE/NSRCHECK: Spool Read/Write Subroutine for Control Blocks: This subroutine has two entry points, one for reading a control block (such as an input/output table) and one for writing a control block. The routine sets PCEDEVTP to either read or write and issues $EXCP to perform the read or write. If a write was requested, the routine returns. If a read was requested, NSRCHECK waits ($WAIT) until the read is complete, and checks the I/O completion code. If a read error occurs the $IOERROR macro is issued and return is made with return code 4 (error). If no read error occurs, return is made with a return code of zero.

NSRQUIET: I/O Quiesce Subroutine: This subroutine is called when the receiver waits ($WAIT) for all its outstanding disk I/O to complete. It checks PCEBUFCT for active I/O and returns if no I/O is active. If PCEBUFCT is nonzero, the routine waits ($WAIT BUF). The processor is posted ($POST) when a buffer is freed, (that is, when one of the receiver's disk writes is completed and the written-out buffer is released by the $ASYNCH processor). After the $WAIT macro, the routine branches back to test PCEBUFCT.

NSRTRACK: Next MTTR Subroutine: This subroutine is called whenever the SYSOUT receiver wants to allocate a new track address. It uses the master track allocation block (master TAB) in the job's allocation input/output table (IOT) as a parameter list for the $TRACK macro. On return from $TRACK, the routine checks to see if the new MTTR required the allocation of a new track group; if not, the routine returns to its caller. If a new track group was allocated, the bit map in the allocation IOT has been updated. The routine will checkpoint the allocation IOT via $EXCP and then return to its caller.

NSRENCOD: Route Code Conversion Subroutine: This subroutine converts an 8-byte node name and an 8-byte remote name (present in a job header or data set header) to a 4-byte internal JES2 route code.

When supporting a spool offload processor, this routine also uses the address of an 8-byte field containing the special local routing attribute in EBCDIC form and uses it to properly build the 4-byte internal JES2 route code. This 8-byte EBCDIC field generally appears in either the JES2 section of the job header or the general section of the data set header. The default node number for the destination is found in the node information table (NIT). If the remote name is present in the NIT, the USERDEST routine in HASPSSSM is called to convert the name to a binary route code, using the node number from the first part as the default node number for the user destination. If the remote work station name is not recognizable as a JES2 remote work station, the 8-byte EBCDIC remote work station name is saved for retransmission.

NSRNOTFY: TSO Notify Subroutine: This subroutine formats and issues either the $HASP546 or $HASP589 message and sends it to the TSO user (see NSRNOTIF).

NSRSPCL: Purge Job Subroutine: This routine terminates a job that has been rejected by work selection services. It purges JCT, IOT, and data set header buffers and places the JQE on the purge queue.

NSRMFR: SMF Record Formatting Subroutine: This routine formats the type 24 SMF record.

SYSOUT Transmission Processor

When a job completes execution, it may have generated SYSOUT data sets that have to be printed or punched on another node (or nodes) in the NJE network. The SYSOUT transmitter is responsible for transmitting these data sets over the network lines to the destination nodes for printing or punching.

Each line that is currently being used for NJE has associated with it one (or more, depending on the STNUM initialization parameter) SYSOUT transmitter device control table (DCT). There is one SYSOUT transmitter processor control element (PCE) for each of the device DCTs; thus each transmitter PCE is associated with one and only one device DCT. For each SYSOUT transmitter, there is a corresponding SYSOUT receiver at the node on the other end of the NJE line.

When performing spool offload, the SYSOUT transmitter also writes a type 24 SMF record for each data set header transmitted.

HASPNST: SYSOUT Transmitter Main Entry Point

The following paragraphs describe the operation of HASPNST and associated functions performed within the code.

Authority Checking If the data set token was sent to this node, or the data set to be transmitted was created locally, the token and the data set name are moved into the header. NSTXDSN is called to extract the DSNAME field from PDBDSNAM.

The token SCR is skipped when scanning the data set header control record.

NST$WAIT: When the SYSOUT transmitter is initially entered, it waits ($WAIT) for work. It is posted ($POST) when its device control table (DCT) is available for use, that is, when NJE sign-on has been completed on the associated line.

NST$UNIT: After the $WAIT macro, the SYSOUT transmitter issues the $GETUNIT macro to obtain control of its unit (its device DCT).

For spool offload, no $GETUNIT is issued if the device has been halted. Instead the transmitter waits to be posted when the device is restarted with a $S OFFLOAD command. If the $GETUNIT is unsuccessful (that is, indicating that the device is drained), the transmitter returns to the $WAIT to wait for work. Otherwise, the transmitter issues the $QSUSE macro to obtain control of the shared job queue and checks to see if there is anything for it to send.

Work for the SYSOUT transmitters is indicated by job output elements (JOEs) on the network JOE queue in the job output table (JOT). The JOEs represent collections of data sets which are to be sent to other nodes in the NJE network. All JOEs for nodes other than the local node are queued on the single network queue; queuing is done so that subsequent selection will be by first-in-first-out by priority. A particular SYSOUT transmitter might be only eligible to send to a few of the nodes in the network; eligibility is determined by the network path manager (HASPNPM), which determines the best path to reach any given node. If the network path manager considers the line associated with a SYSOUT transmitter′s device DCT to be a desirable path to a particular node, it turns on a flag byte corresponding to that node in the line DCT (in the MDCTNODS flag). Thus, an NJE line DCT will have a series of flag bytes turned on in MDCTNODS indicating to which nodes the transmitter can send.

NST$MSG: If the $#GET indicates that there is no work for the transmitter, message $HASP534 is issued to inform the operator that the transmitter is inactive, the device DCT is released via the $FREUNIT macro, and the SYSOUT transmitter waits ($WAIT) for something to be added to the JOT. When the JOT is posted ($POST), and the SYSOUT transmitter is eligible to process the new work added to the JOT, the transmitter is $#POSTed and branches back to the initial $GETUNIT macro (NST$UNIT).

The transmitter issues the $#GET macro to obtain a JOE from the network JOE queue. The macro invokes the $#GET service routine. This service routine scans the network JOE queue for the oldest highest priority JOE that is routed to one of the nodes to which the SYSOUT transmitter can send (as indicated by MDCTNODS). If such a JOE is found, the $#GET service routine marks it busy, sets the transmitter's device id in the JOE, and returns the address of the JOE to the transmitter.

When $#GET has selected work for the spool offload function, this routine checks a return code to see if $#GET found held output. If it did, a bit indicating this is set in the PCE work area for this transmitter. The count of held data sets is also placed in the PCE. The transmitter then marks the JQE busy, checkpoints it, and sets the JQE offset in the LCK corresponding to the offload device dumping these data sets. If no JOE is found, $#GET sets a nonzero return code and returns to the transmitter.

NSTGOTJB: If the $#GET macro returns the address of a JOE, the SYSOUT transmitter prepares to transmit a collection of data sets for a job. Before relinquishing control of the shared queues, the transmitter scans the entire network JOE queue, searching for more JOEs for the same job. (JOEs that are routed to any of the nodes to which the transmitter is eligible to send). If any more are found, they are also marked busy and flagged with the SYSOUT transmitter's device id. Then they are chained to the original JOE via the JOENETCH field. At the end of the scan, the SYSOUT transmitter has collected as many JOEs as possible for transmission as a single unit.

NSTENDQ: After scanning the network JOE queue, the SYSOUT transmitter is ready to begin transmission. The job's job control table (JCT) is read and message $HASP530 is issued to inform the operator that SYSOUT transmission for the job is beginning. If the JCT read was in error, the transmitter branches to its abort job routine (NSTNCL01); otherwise, a request for permission to transmit is generated via a $EXTP OPEN macro.

If permission is granted (the receiving end is prepared to accept transmission), SYSOUT transmission continues. If the transmitter is a network transmitter, rather than a spool-offload or route transmitter, the authorization fields in the job header are cleared to zero. If the transmitter is a spool offload transmitter, the spool offload section of the NJE job header is initialized to contain job information from the JQE and the time and date stamp from the DCT. The SYSOUT transmitter then sends the job's NJE job header, which was stored in the JCT when the job entered the system. If permission is not granted, the transmitter branches to NSTNCLOS to abort the transmission.

NSTIOTRD: If any of the JOEs acquired by the transmitter were regular (that is, not spin) JOEs, the transmitter scans the job's chain of regular input/output tables (IOTs). Each SYSOUT peripheral data definition block (PDDB) in each IOT is checked against the transmitter's chain of work JOEs and the characteristics JOEs associated with them. If a match is found, the transmitter branches to NSTDSHDR to send the PDDB's data set header. After the data set header has been transmitted, the data set is sent. Once the data set has been sent, the PDDB scan is resumed.

Each PDDB is also checked for held status. If output is to be held, and held output is to be dumped, then the JOE chain will not be checked. And the data set and its header are immediately transmitted.

Note: Multiple data set headers can be sent for the same data set, created from the JECL /*OUTPUT statement or JCL OUTPUT statement.

NSTCKSPN: At the end of the regular PDDB scan, or if there were no regular JOEs, the transmitter checks to see if any of the acquired JOEs were for spin data sets. If so, the SYSOUT transmitter's JOE chain is scanned, and for each spin JOE, the transmitter reads the corresponding spin IOT and branches to send the data set header(s) and the data set.

NSTENJOB: After all possible data sets have been transmitted, the SYSOUT transmitter sends the job's NJE job trailer, which was stored with the job header in the JCT when the job entered the system. It then sends an end-of-file indication to the receiver via a $EXTP CLOSE macro. The transmitter then waits ($WAIT) for acknowledgment of the end-of-file. If positive acknowledgment is received, the responsibility for the transmitted data sets has been accepted by the receiver, and the transmitter removes all of its JOEs from the network JOE queue via the $#REM macro. If these JOEs represent the last work for the job, the job is queued for purging. The transmitter then releases control of its device DCT and tries to reacquire the device and look for more work.

NSTRJOES: For NJE or spool offload jobs specified as DISP=DELETE the JOEs for the job must be freed. For held datasets, the PDDBs must be freed. If the PDDBs for a job's held data sets are not deleted (even though the DISP=DELETE was specified), a message tells the operator that the held data set(s) for the job is/are not deleted. These data sets are processed as if DISP=KEEP had been specified.

NSTNCLOS: If the SYSOUT transmitter receives a negative acknowledgment to the end-of-file or to any preceding record, it aborts the job. This is done by sending a negative close via a $EXTP NCLOSE macro. All the JOEs acquired by the transmitter are restored to the network JOE queue via the $#PUT macro; the job is requeued for transmission. Any subsequent transmission will start over again from the beginning. No checkpoint/restart is attempted in the middle of a data set. After the negative close, the transmitter releases its device DCT and branches back to the start as for a normal end of transmission.

NSTDSHDR: When the transmitter determines that a data set is to be transmitted, it must first send one or more NJE data set headers before sending the data records themselves. If the job being transmitted did not execute on the local node, but was itself received from another node by the SYSOUT receiver, all the required data set headers have already been stored on the spool file. The first record of the data set is a spool control record (SCR), which contains disk addresses and offsets that point to one or more data set headers. If a data set has multiple destinations or the job had multiple JCL output statements, there will be one data set header for each destination or one data set header for each output card. When a data set is to be sent, the SYSOUT transmitter checks to see if the first record of the data set is an SCR; if so, the appropriate data set header is read from spool and transmitted. The accuracy of the data set header is ensured even if the operator modified the JOE.

NSTDSHDB: If the first record is not an SCR, then the job was executed on the local node and no data set header yet exists.

In this case, the transmitter builds a header from information in the work JOE, the characteristics JOE, and the PDDB; if the data set has output processing characteristics that are defined only for the 3800, then the transmitter sends the 3800 section of the NJE data set header. If the data set used the JCL OUTPUT statement to specify output characteristics, then scheduler work blocks (SWBs) are created. The SWBs are stored and sent as part of the page-mode section of the data set header. If the data set contained page-mode records, then this information is also carried in the page-mode section of the data set header.

NSTDSHDP: After sending the data set header, the SYSOUT transmitter checks the data set's PDDB for multiple destinations. If there are none, the data set itself is transmitted; if there are multiple destinations, the transmitter scans the rest of the data set's PDDBs for a match with any of the transmitter's JOEs. If a match is found, the appropriate data set header for this PDDB/JOE combination is retrieved or constructed and then sent immediately after the preceding header. Thus more than one data set header may be transmitted with no intervening data set records. The SYSOUT receiver recognizes this condition as indicating that the data set to follow has multiple destinations or the data set is to be printed with different output characters. The data set itself is transmitted as few times as possible.

NSTCKOPR: As the data set is sent, a check is made after each record is transmitted for operator commands ($C, $E). Any such commands are handled immediately. Otherwise, the records making up the data set are sent one by one via the $EXTP PUT macro.

NSTSPAN: Spanned data records are broken up and sent in pieces so that no single record passed to RTAM is more than 256 bytes long. The spanned records are reconstructed by the receiver. At the end of the data set (when HDBNXTRK becomes null), the PDDB scan (for regular data sets) or the JOE scan (for spin data sets) is resumed.

SYSOUT Transmitter Subroutines

The following subroutines support the SYSOUT transmission processor.

NSTHXMIT: Transmit Header or Trailer Record Subroutine: This subroutine is called with a pointer to a job header, a data set header, or a job trailer. The first 2 bytes of the header or trailer contain the total length of the header or trailer. The NSTHXMIT subroutine analyzes the total length and sends the header or trailer in pieces of no more than 256 bytes each, setting the sequence counter and continuation indicator in each piece that it sends. If any error is returned from $EXTP PUT, transmission of the header or trailer is terminated, and this error indication is passed back to the caller of NSTHXMIT.

NSTRDBUF: Direct-Access Read Subroutine: This subroutine is passed a buffer address in register SBUF (register 2) and an MTTR for PCESEEK; it issues a $EXCP macro to read a spool block into the buffer and waits ($WAIT) for completion. The success or failure of the read is reflected to the caller in the condition code: 3 if successful, 1 if failed.

NSTDECOD: Route Code Converter Subroutine: This subroutine converts a 4-byte internal JES2 route code, as it exists in a PDDB, JOE, or JQE into two 8-byte EBCDIC names.

When supporting a spool offload processor, this routine also converts the special local routing attribute found in the JES2 section of the job header and stores it in an 8-byte field provided by the caller. The first 2 bytes of the route code is the node number; this number is used as an index into the node information table (NIT) in order to extract the first 8-byte EBCDIC name - the name of the node. If the node number is 0 (indicating special local routing), $OWNNODE is used instead.

The last 2 bytes of the route code represents the remote number. If this number is 0, the second 8-byte EBCDIC name is left at binary zeros; this indicates local routing. If the remote number is nonzero, it is converted to decimal, preceded by the letter R and padded with trailing blanks to make up the second 8-byte EBCDIC name; thus, hex X'23' becomes R35.

NSTNEWS: Update Record Count: This subroutine updates the header record count to include the record count from the JESNEWS data set, if transmitted.

NSTAUTH: Security Authorization: A $SEAS call is issued to audit the fact that the SYSOUT is being transmitted.

Job Transmission Processor

When a job enters a JES2 system, it may be destined for execution on another node in the NJE network. The job transmitter is responsible for transmitting an image of the job's job control language (JCL) and its in-stream (SYSIN) data sets to the destination node for execution.

Each line currently being used for NJE has associated with it one (or more than one, depending on the JTNUM initialization parameter) job transmitter device control table (DCT). There is one job transmitter processor control element (PCE) for each of the device DCTs. For each job transmitter, there is a corresponding job receiver at the node on the other end of the NJE line.

Job transmission processor routines are also used by the network job route transmitter. The QUEXMIT flag in the JQE indicates that this job was entered using a /*XMIT statement or was received by a network job receiver and will not be executed at this node. If a /*XMIT job is rerouted for execution at this node ($OWNNODE), the network job route transmitter resubmits the job through normal JES2 input processing. A corresponding network job route receiver accepts the rerouted job. There is one job route receiver and one transmitter PCE. The job transmission processor writes a type 24 SMF record.

Writer class RACROUTE authority checks are performed for held spool offload facility data sets (only) by NSTSAFCK. NSTXDSN takes the DSNAME (last) qualifier of the data set name from the PDDB and copies it into the NDHGNAME field in the data set header.

HASPNJT: Job Transmitter Main Entry Point

The following describes the operation of HASPNJT and its associated functions.

JXMTINIT: When the job transmitter is initially entered, it issues the $GETUNIT macro to obtain control of its unit (its device DCT).

No $GETUNIT is issued if the device has been halted. Instead the transmitter waits to be posted when the device is restarted with a $S OFFLOAD command. If the $GETUNIT is unsuccessful (for example, the device is drained), the transmitter waits ($WAIT) for work. Otherwise, it issues the $QSUSE macro to obtain control of the shared job queue and checks the status of its DCT to see if transmission should be attempted. If transmission should not be attempted, it returns to its $WAIT to wait for work. It also returns to the $WAIT if the node information table (NIT) entry for its line indicates that job transmitters should be drained.

JGOTUNIT: If this is the network job route transmitter, control is passed to JNRTINIT. If transmission should be attempted, the transmitter issues the $QGET macro to get a job from the $XMIT queue. The $QGET service routine scans the $XMIT queue for a job that the job transmitter can send. A particular job transmitter may be eligible to send jobs to only a few of the nodes in the network; eligibility is determined by the network path manager (HASPNPM), which determines the best path to reach any given node. If the network path manager considers the line associated with a job transmitter's device DCT to be a desirable path to a particular node, it turns on a flag bit corresponding to that node in the line DCT (in the MDCTNODS flags). Thus an NJE line DCT will have a series of flag bytes turned on in MDCTNODS to indicate to which nodes the job transmitter can send. When $QGET scans the $XMIT job queue, it selects the oldest-highest priority job that is routed to one of the nodes to which the job transmitter can send.

JXMTWAIT: If the $QGET routine returns to the job transmitter without selecting a job, the transmitter issues $FREUNIT to release control of its device DCT and waits ($WAIT) for something to be added to the job queue. If message $HASP524 has not been issued, it is issued now to inform the operator that the job transmitter is inactive. When the transmitter is posted ($POST) for a job queue element (JQE), it branches back to reacquire its device DCT via $GETUNIT.

JNRTINIT: The network job route transmitter issues the $QGET macro instruction to obtain a job (from the $XMIT queue) that is destined for execution at this node ($OWNNODE). The transmitter waits if no jobs have been rerouted.

JGOTJOB: If a job was selected by $QGET, the job transmitter prepares for transmission. The $TIME macro is issued to obtain the job transmitter's starting time, a spool buffer is acquired, and the job's job control table (JCT) is read. If the read of the JCT is unsuccessful, processing continues at JCTERR1 where a disastrous error message is issued and the job is aborted. Otherwise, message $HASP520 is issued to inform the operator that job transmission is beginning. A request for permission to transmit is generated through HASPRTAM via a $EXTP OPEN macro.

If permission is granted (the receiving end is prepared to accept transmission), job transmission continues. If the transmitter is a network transmitter, rather than a spool-offload or route transmitter, the authorization fields in the job header are cleared to zero. If the transmitter is a spool offload transmitter, the spool offload section of the NJE job header is initialized to contain job information from the JQE and the time and date stamp from the DCT. Before the job transmitter sends the job's NJE job header (which was stored in the JCT when the job entered the system), checks are made to see if the passwords in the job header need to be encrypted and to determine if the target node can accept encrypted passwords. The passwords are encrypted only if the job originated at this node and the target node can accept encrypted passwords.

Before a 'held' job is transmitted, the 'hold' attribute is set with the time and date stamp in the spool offload section of the header. This allows a network held job to be dumped to the offload data set, and be reset as 'hold' in its JQE after being reloaded. The system affinity of the job is also stored in the spool offload section. If permission is not granted, the transmitter branches to JABTERMD to abort the transmission.

JGOTIOT: After sending the job header, the transmitter reads the job's first input/output table (IOT) and locates the JCL data set. The JCL data set is read from spool, and each card image is sent, one at a time, via $EXTP PUT.

JCNTLREC: At each point in the JCL stream where an in-stream (SYSIN) data set should be, HASPRDR inserted a spool control record (SCR) which contains the SYSIN data set's characteristics (RECFM, LRECL, BLKSIZE) and location on the spool file. When the job transmitter encounters an SCR, it builds and transmits a data set header, which gives the characteristics of the data set. The job transmitter then retrieves each SYSIN data record from the spool file and sends it via the $EXTP PUT macro.

JXMTEOB: After all of the data records for the SYSIN data set have been sent, the transmitter resumes sending the JCL data set immediately after the SCR. When the entire JCL data set has been sent, the transmitter reads the JCT again and sends the job trailer, that was stored when the job entered the system. In this way, the job transmitter sends an image of the job as it was originally submitted.

JGOTJCT2: After the job trailer has been sent, the job transmitter sends an end-of-file indication to the receiver by issuing the $EXTP CLOSE macro. A positive acknowledgment to the end-of-file indicates that the job receiver on the other node has accepted responsibility for the job. The job transmitter then queues the job for purging. A negative acknowledgment results in the job transmitter requeuing the job for transmission.

JNOTIFY: After a successful transmission, the job transmitter checks the job's JCT to see if the job contains a /*NOTIFY control card or if the submitter of the job specified NOTIFY=userid on the job statement. If either condition exists, and if the local node is the originating node for the job, message $HASP526 is sent (via $WTO) to the specified node and user ID in order to indicate that the job has been transmitted for execution and is no longer present on the local node.

If supporting a spool offload processor, message $HASP526 is not issued. Instead (if notification messages are not suppressed), message $HASP588 is issued.

JUPDTJCT: When transmission is complete, the job transmitter updates the job's JCT with the transmitter's starting and ending times and accounting information, rewrites the JCT, and releases the job's spool buffer. The job transmitter then issues $FREUNIT macro to release control of its device DCT and branches back to the initial $GETUNIT.

JXMTEND: If at any point the job transmitter determines that the job is to be aborted, it informs the receiver by issuing the $EXTP NCLOSE (negative close) macro. A $WTO is issued to inform the operator that the job has either been restarted (message $HASP522) or deleted (message $HASP523). The job is deleted if the operator has entered a cancel ($C) command or if an error was encountered while reading the job's JCT or IOT. Otherwise, the job is restarted (requeued for transmission). The job transmitter then proceeds as if for a normal termination.

Job Transmitter Subroutines

The following subroutines support the job transmitter.

JXMTIO: Disk I/O Subroutine: This subroutine stores the MTTR, passed on input, into the processor control element (PCE), and issues the $EXCP macro to read or write the desired spool block. It waits ($WAIT) for completion of the I/O and checks the resulting condition code. If an error occurs, it returns to the link address passed by the caller. If no error occurs, it returns to the link address plus 4.

JJCTCHEK: JCT Validation Subroutine: This subroutine ensures that the record that has just been read is a valid job control table (JCT) record. The JCT is considered valid if: it contains a JCT in the JCTID field, its job queue element (JQE) offset matches the JQE obtained by the transmitter, and its job name matches the job name in the transmitter's JQE. On return to the caller, this subroutine sets a condition code 0 to indicate a valid JCT; a nonzero condition code indicates an invalid JCT.

JXMTPUT: Network Output Subroutine: This subroutine issues all $EXTP PUT macros for the job transmitter. After each $EXTP PUT is issued, the subroutine checks for an error return and checks flags in the device control table (DCT) and JQE for any operator commands that might affect the job transmitter. If a $EXTP PUT fails, this subroutine branches to JABTERMR. If the DCTDELET or DCTRSTRT bits are set, this subroutine branches to JABTERM. If the operator has cancelled the job, this subroutine branches to JABTERMC. If none of the above has occurred, this subroutine returns to the caller.

JXMTHDR: Transmit Job Header or Job Trailer Subroutine: This subroutine is called when a job header or job trailer is to be transmitted. The first 2 bytes of the header or trailer contain the total length of the header or trailer. The JXMTHDR subroutine analyzes the total length and sends the header or trailer in pieces of no more than 256 bytes each, setting the sequence counter and continuation indicator in each piece that it sends. If any error is returned from the $EXTP PUT macro, transmission of the header or trailer is terminated, and the same error indication is passed back to the caller of JXMTHDR.

JDECODE: Route Code Converter Subroutine: This subroutine converts a 4-byte internal JES2 route code, as it exits in a PDDB, JOE, or JQE, into two 8-byte EBCDIC names.

When supporting a spool offload processor, it also converts the special local routing attribute found in the JES2 section of the job header and stores it in an 8-byte field provided by the caller. The first 2 bytes of the route code contain the node number. This number is used as an index into the node information table (NIT) to extract the first 8-byte EBCDIC name, which is the name of the node. If the node number is 0 (indicating special local routing), $OWNNODE is used instead.

The second 2 bytes of the route code represent the remote number. If this number is 0, the second 8-byte EBCDIC name is left at binary zeros, indicating that no remote was specified (that is, local or not defined with a RMT initialization statement). If the remote number is not zero, it is converted to decimal, preceded by the letter R and padded with trailing blanks to make up the second 8-byte EBCDIC name; thus, X'23' becomes R35.

HASPSSSM: Subsystem Support Modules

HASPSSSM, resident in either LPA or CSA, communicates directly with the operating system to provide job scheduling, data management (SYSIN and SYSOUT), operator communications, and other tasks associated with the operating system. The HASPSSSM load module contains function routines that are invoked through the use of addresses in the subsystem vector table (SSVT). The operating system uses these addresses to invoke functions defined by the IEFSSOB macro. Modules in HASJES20 (the private area of the JES2 address space) use addresses in the $CADDR to communicate with the HASPSSSM modules. HASPSSSM, therefore, serves as the communication vehicle between the user address space and the JES2 address space.

HASPSSSM contains the HASCDSS module, which provides data space services. When invoked by the $DSPSERV macro, it creates a data space in the current address space, the JES2 address space, or the master address space (if invoked to create a data space), or deletes an address space (if invoked to delete a data space).

HASPSSSM also includes the following source modules:

- HASCDSAL -- data set allocation/unallocation
- HASCDSOC -- data set open/close services
- HASCJBST -- job select/termination services
- HASCJBTR -- job end-of-task/end-of-memory services
- HASCLINK -- nucleus of the common storage routines
- HASCSRIC -- services and control subroutines
- HASCSIRQ -- miscellaneous SSI routines
- HASCSRDS -- subroutines related to data sets
- HASCSRJB -- subroutines related to jobs

Figure 3-3 on page 3-117 shows the relationship between these modules.

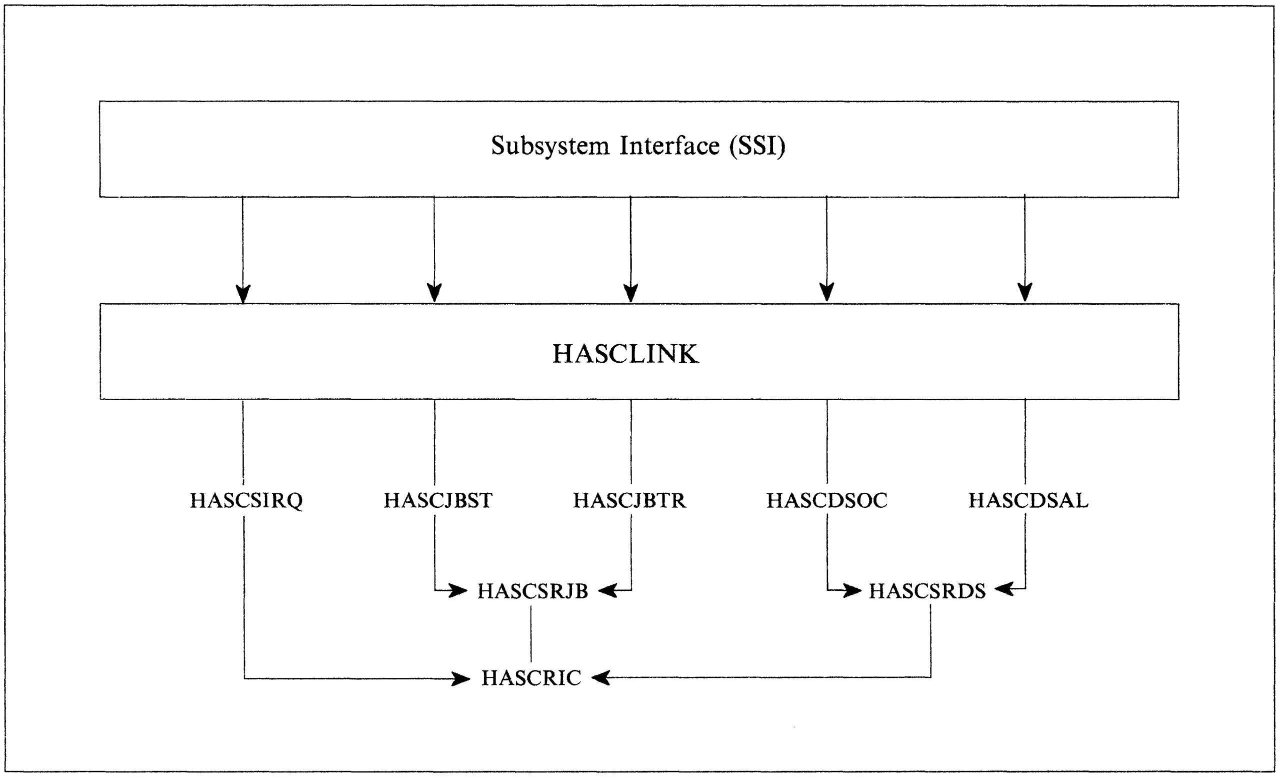


Figure 3-3. Relationship Between HASCLINK Modules

Figure 3-4 shows the major control blocks related to HASPSSSM processing. See chapter 5 for a table summarizing these control blocks.

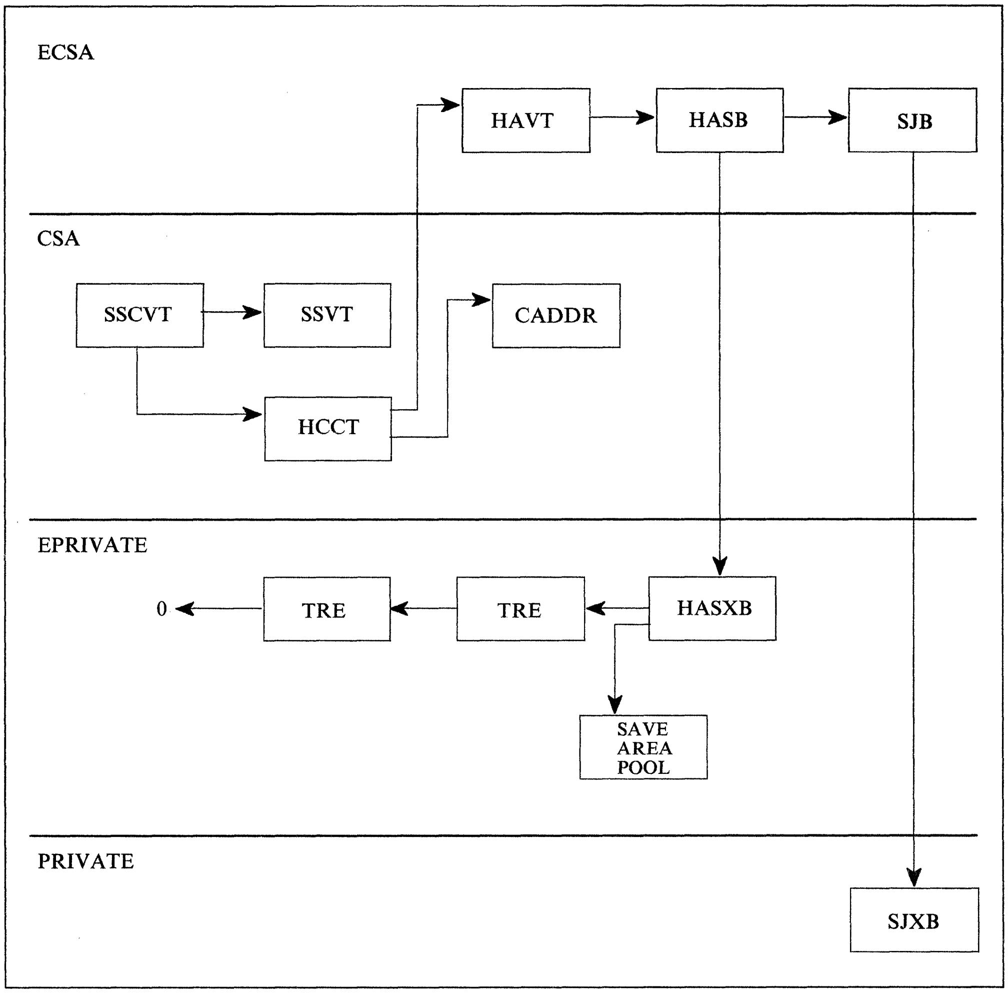


Figure 3-4. Control Blocks Structure Related to HASPSSSM Processing

HASCLINK

HASCLINK is the base service module for the LPA/CSA-resident JES2 modules. (These modules can be referred to as either the user environment or HASC* modules.) HASCLINK contains:

- The common storage routine address ($CADDR) table, which contains the addresses of routines used in conjunction with the $CALL macro to provide linkage within the HASC* modules.
- The $SENTRYS table, which is JES2's version of the MVS SSVT, and which is copied to the SSVT during JES2 initialization and provides addresses of SSI routines.

HASCLINK provides services mainly to the other HASC modules, but some of the routines are called out of all environments, whether the JES2 main task, the subtasks, user address spaces, or FSS address spaces. HASCLINK contains the following routines:

- HGFMAIN is invoked when a $GETMAIN or $FREEMAIN is issued in the user or FSS environment. It is also invoked when a user issues a $GETBUF or $FREEBUF. The routine protects the callers key and lock structure.
- $CPOOL obtains a cell of storage in the private area using the MVS CPOOL established via a $SSIBEGN call. The cell is taken from the TCB recovery element (TRE) pool (the cellpool ID is HXBTREID). This cellpool differs from $GETCEL services because those services manage CSA pools.
- $MLTBUF is called with a $FREEBUF TYPE=MULT in the user environment to free a chain of buffers.
- $CSAVE obtains a save area for a caller using CPOOL services. Register 13 returns the address of the save area. If tracing is active, $CSAVE issues a $TRACE for ID=11 (for a specific PCE/TRE) or ID=18 (for all $SAVES issued through this routine.)
- $CRETRN traces $RETURNS when tracing is active, and issues a call to $CRETSAV to free the current save area. $CRETSAV returns the save area to the save area pool if the FRETRE= operand was specified. It frees all save areas associated with the specified TRE and frees the TRE for reuse. $CRETRN then loads the registers from the previous save area and returns to the caller's caller.
- GETTRE gets a TCB recovery element (TRE) for the caller. Its address is returned in register 1. If there is no available TRE, a CPOOL is issued to get storage for one, and it is initialized.
- $GETCEL is used to obtain a CSA cell of the specified size. This routine scans the chain of cell control elements (CCEs) anchored from CCTCSACH in the HCCT looking for a cell with a length that matches the caller's request. If none is found, a cell of greater length is used. If none is found, a CCE with no associated cell is located and a $GETMAIN is issued. If no CCE is found, then 10 CCEs and a cell are $GETMAINed. The routine remembers the most eligible CCE, so that multiple scans are not necessary.
- $FRECEL frees a CSA cell when it is provided with a cell address, or frees any cells associated with a key1/key2. CCEs are not freed, only cells. $FREMAINs will be issued only if the cell utilization falls below 50% and the total CSA allocation for cells exceeds 8K bytes. The CCE of an unfreed cell will indicate that it can be reused.
- SSIFINE performs the first part of SSI environment termination. This routine frees the SJB lock if the SSI function did not do it and saves the return code it receives. If the return code is positive, it is placed in the SSOBRTRN field. If the return code is negative, it is made positive and left in register 15. The routine also issues a $TRACE ID=12 for tracing of specific PCE/TREs $RETURNs. If there is no tracing to be done for specific PCE/TREs, $TRACE ID=19 is issued to trace all $RETURNs.
- SSISETUP establishes an ESTAE routine called RECOVERY and establishes the fist available save area.

- $SSIBEGN services the $SSIBEGN macro issued in the user environment, and is responsible for ensuring that a HASB, HASXB, and two MVS CPOOL exists for any address space that issues a subsystem call to JES2. If no HASB/HASXB exists for the address space, they are $GETMAINed. The HASB storage is obtained in CSA. The HASXB storage is obtained in expanded private under the highest TCB in the address space, which means that MVS cannot free it until all TCBs have finished. There are three kinds of HASB/HASXBs:

  1. Permanent HASB/HASXBs are those created for address spaces started by JES2 itself, such as an FSS address space, which are expected to make many SSI calls.
  2. System HASB/HASXBs are those created for an address space that JES2 did not start but from which many SSI calls are nevertheless expected, such as SYSLOG.
  3. Temporary HASB/HASXB are those created for an address space JES2 did not start and from which few SSI calls are expected.

  The routine then creates two cell pools, one for save areas and one for TREs. The newly acquired control blocks are initialized and chained from the HASB. The HASB itself is chained from the HAVT. This routine then returns to the $SSIBEGN macro expansion with a negative return code indication that the SSI call should continue. A positive return call of 16 indicates that a disastrous subsystem error occurred. No other positive return code is possible.

- $SSIEND provides support for the $SSIEND macro in the user environment. It cleans up temporary HASB/HASXBs and the MVS CPOOLs. The $SSIEND macro is issued just after the SSIFINE routine has stored the SSI function's return code and cleaned up any resources the function did not clean up.

- RECOVERY is the SSI generalized recovery routine. If a $ERROR is issued in the user environment, an abend is generated and this routine gets control. This routine recovers from abends and attempts to retry at one of the addresses in the $SSIBEGN parameter list, which is built at assembly time and points to different locations within the $SSIEND macro expansion. These addresses point to cleanup routines. A successful retry result in a return to the SSI caller with an error indicator in register 15. The $SSIBEGN parameter list is accessed through the TCB recovery element (TRE) which is passed to this routine from RTM (ESTAE PARAM=TRE). RECOVERY may attempt to retry at the instruction after $ERROR, if that macro specified this in its parameters. RECOVERY is not invoked for user exits.

HASCDSAL

This module provides the support for subsystem data set allocation and deallocation. Routine SSIALOC is invoked to begin the process of allocation. It creates a data set name, if needed. It sets the subsystem name in the data set name and $CALLs the appropriate data set allocation routine for the kind of data set being allocated: HALI for SYSIN data sets, HALO for SYSOUT data sets, and HALP for process-SYSOUT data sets. User exit 31 allows the alteration of SYSOUT characteristics.

HALO calls HALR to allocate internal reader data sets. It also calls HALSSALP and HALOCRP for multiple-destination PDDBs to perform destination validity checks. HALSSALP merges values from the SSOB into a PDDB passed to it and verifies the destination (via $DESTCHK). HALOCRP merges OCR values into a passed PDDB and verifies access authority. If the verification is successful, the destination (including remote) is set in PDBDEST; otherwise the allocation is failed.

The other routines called by HASCDSAL include HALOPDBI, which completes the filling in of the PDDB for SYSOUT allocation; HALJMERG, which merges PDDB fields into the JFCB; HAOUTSCN scans the OUTPUT= references on a DD statement and updates the SJFREQ call parameter list with the PROC step label needed to retrieve the statement. The routine also updates the pointer to the next output reference. $PDBBLD builds a new PDDB and IOT, if necessary.

For unallocation, HASCDSAL contains SSIALUNA, which sets up the unallocation environment and then calls HALUNAL, which does the actual unallocation and performs some destination validity checking. Calling $SDBFREE unallocates SYSIN and PSO data sets by freeing the SDB. If a PSO data set is marked for deletion, the PDDB is marked as such for later JES2 purge processing. If the data set is to be deleted, the PDDB is marked, $SDBFREE is called, and records like the JCT and IOTs are written. If the data set is not to be deleted, its SDB is freed. Spin data sets are also processed. Routine HALDCIOT is called to dechain the spin IOT, and HIOTSPIN passes a spin IOT to HASPXEQ. For internal readers, the internal reader DCT whose address was passed in the SSOB extension is freed for reuse, and any wait elements waiting for an internal reader are posted. These wait elements represent tasks waiting in SSIALOC to allocate an internal reader. User exit 34 is the unallocation exit, and can be used to undo any processing (the alteration of SYSOUT characteristics with user exit 31) done at allocation.

Note: When a data set is deleted at purge time, JES2 issues a RACROUTE REQUEST=AUTH SAF call in the JES2 address space to determine if the requestor has alter authority to delete the data set. The security tokens are passed for comparison.

HASCDSOC

This module provides the basic support for opening and closing of subsystem data sets. Routine SSIAOPN accepts a problem program request to open a data set or an internal reader. It is entered through the subsystem interface.

When an internal reader is opened, the current task's security token is extracted, mapped with a user id and group, and pointed to by $DCT. This token is used as the submittor's token on later authority verification.

When a SYSIN data set is opened, JES2 issues a RACROUTE REQUEST=AUTH SAF call to verify authority to read the data set. (For SYSOUT data sets, the SAF call verifies update authority. For process-SYSOUT (PSO) data sets, JES2 adds the RECVR keyword to the RACROUTE request to ensure the receiver has sufficient authority to the data set.) This occurs in the caller's address space under the task that performs the OPEN.

The SSIAOPN routine issues a $SSIBEGN macro to invoke the $SSIBEGN routine in HASCLINK to initialize the subsystem function. Then it $CALLs the DSOPEN routine, which actually opens SYSOUT, SYSIN, PSO, or INTRDR data sets. On return, SSIAOPN issues the $SSIEND request to terminate the SSI function request. Exit 30 gains control after the data set has been opened (or restarted) and can be used to fail the open.

Routine SSIDACLO closes data sets and internal readers. It calls $SSIBEGN to initialize the subsystem interface request, and issues $SSIEND to end the subsystem request. User exit 33 gains control before the close. Routine SSIDARES restarts data sets. It uses the SSOB extension to get the address of the restart buffer and sets the SSOB to point to the SDB. Routine SSVOPNC is called from HASPCNVT to open data sets required for the converter. It calls HFOPSUB to obtain an SDB in the case of a JCL image data set, the job log data set, and internal text data set. SSVCLSC provides a fake close for converter data sets. Routine SSIDACKP is invoked when a $CKPT macro is issued for a subsystem data set. If the data set is SYSOUT, the current buffer is written out. If this is SYSIN data, the record number of the next record to be read (or an indication of end-of-data) is returned in the checkpoint buffer.

HASCJBST

This module provides the support for job selection and job termination. When a subsystem interface request for a job is issued, SSIJSEL either locates or creates an SSIB level SJB and gets its lock, calling $SJBFIND, SJBINIT, and $SJBLOCK. It then calls JBSELECT to select and process a job. It returns to the caller with the return code in SSOBRETN. The JBSELECT routine queues the SJB either onto the select-by-class or select-by-id queues, and posts HASPXEQ to select a job. When HASPXEQ has selected a job, JBFOUND is called before returning to the caller. JBFOUND completes the setup of information necessary for the job to run. JBFOUND processing includes reading in disk-resident control blocks, opening system data sets, linking to the scheduler's SWA create routine, and so forth. It does processing for batch jobs (select-by-class), STC/TSU jobs (select-by-id), and execution batch monitor jobs.

Exit 32 can be used in HASCJBST to allocate user control blocks, read or write spool resident control blocks or data, issue user-supplied start messages, or terminate jobs or initiators.

JSOPSSDS opens subsystem data sets for non-execution batch monitor continuation jobs. It opens the internal text data set, the job journal data set, the system messages data set, and the joblog data set. JSREOPEN does open processing for subsystem data sets (messages, JCL images, JOBLOG data sets) on a warm start, or if the job is to be rerun. If the data set is the JCL image data set, the routine is exited. For the other possible data sets, the routine repositions the data set. This means that the data set is opened for input, and the existing data set is read in. The routine then changes the input data set to output, and writes to the data set what was read in. The write is not done if the data set was empty or the job deleted or cancelled, or if an I/O error occurred while reading the data set.

A SAF call is made to extract the SYSLOG (request job id) token. SSIRRREQ provides a system-requested job with a valid JES2 job id. This is subsystem request SSIBRQST. SSIRRRET returns a JES2 job id. This is subsystem request SSOBRTRN. To requeue a job (subsystem request SSOBRENQ), SSIRQRNQ is invoked. Job termination is done by SSIJTERM (subsystem function number 12). SSIJTERM cleans up the job's associated control blocks when a batch job, an execution batch monitor job, a started task (STC), or a time sharing user (TSU) job is to be terminated. User exit 28 can be used to suppress messages issued to the job log. Other routines include MRGSWBS, which merges new output characteristics (retrieved from the JESDS keyword on the OUTPUT JCL statement) into the existing PDDB for the data set; HJMAKSL, which creates a slot in an IOT for a system PDDB so that the system PDDBs can remain contiguous for fake closes; HFJOBLOG, which places the header line and early messages from the JCT into the JOBLOG; and JCTSTATS, which updates JCTPAGES, JCTBYTES, JCTXOUT, and either JCTLINES or JCTPUNCH in the JCT to be the sum of the corresponding fields from the PDDB for the job log, JCL images, and system messages data sets.

HASCJBTR

This module terminates subsystem jobs, providing processing for end-of-task and end-of-memory subsystem requests. SSIETEOT processes end-of-task requests. If it is an FSS-related task, the FSS and its associated FSAs are disconnected (routine EOTFDCON issues a FSIREQ disconnect request). If it is not an FSS-related task, the SJB lock is obtained and all tasks in this address space are marked non-dispatchable. It frees all the cells associated with this SJB/TCB pair, and all TREs and save areas. Then, it restarts any pending I/O, waits for it to complete, resets other TCBs to dispatchable, releases the SJB lock, and returns to the caller of the subsystem interface. User exit 35 is given control just after the lock is obtained. It allows the freeing of CSA storage chained from the HASB or the SJBs.

The end-of-memory routine, SSIENEOM, is invoked by the subsystem interface as part of end-of-memory processing. If the JES2 address space itself is terminating, flags are set in the HCCT to indicate this, and the PSO queue is purged. If it is an FSS address space, this routine $$POSTs the resource queue of FSS and cleans up FSSCB pointers. For all address spaces, the job communications queue lock, and the STC and/or TSU locks are freed. Then, the internal readers are cleaned up and the SJB is cleaned up and requeued. User exit 29 gets control just before freeing the SJBs for an address space to allow the freeing of any CSA storage chained from the HASB or the SJBs.

HASCSRIC

This module contains locking routines. $SVJLOK and $SVJUNLK get and release the job communication queue locks; TSETLOCK and TSFRELOK get and release the local and CMS locks using the MVS SETLOCK macro. HASCRIC also contains the following routines:

- $TRACER and $TRAREL, which obtain and release a JES2 trace table entry.
- $$POST, which issues a cross-memory post to the JES2 main task indicating which processor or resource to $POST.
- $STRAK, which allocates spool in the user environment.
- $RACROUT, which issues calls to provide authority verification. The $RACROUT routine contains exits 36 and 37, which permit an installation to accept or fail a security request just prior to or immediately following the request, respectively.
- PRTAUTH, which is called by HASPPRPU or during FSS processing to verify that a job has sufficient authority to print a data set.
- Various other routines, which provide such services as freeing SJF storage and checking for successful I/O.

HASCSIRQ

This module provides the support for the following SSI functions: command processing (SSICMD, the SVC 34 exit routine), WTO processing (SSIWTA, the SVC 35 exit routine), process SYSOUT requests (SSISOUT), TSO cancel (SSICSCAN), TSO status (SSICSTAT), destination verification (SSIUSUSE), FSS connect/disconnect processing (SSIFSCNT). Additional service functions are: TSCNVJB, which converts a job id to a binary job number; TSQUEUE, which queues an SJB to the HASP work queue; USERDEST, which verifies a destination and converts it to a binary route code; USERSUB, which is the user/subtask exit effector; and $DESTCHK, which performs destination authority verification.

HASCSRDS

This module contains the following service routines related to data set processing:

- $SDBINIT creates and initializes subsystem data blocks (SDBs), which contain information about each allocated data set.
- $DSDBFREE is called to free an SDB.
- $IOTBLD creates and chains a secondary allocation, spin, or PDDB IOT.
- $FNDRRIOT is called to locate a reusable spin IOT, which is an IOT whose spin data set has been processed and purged and is ready to be reused.
- $PDBFIND searches a job's in-storage IOT chain starting with the passed IOT for a specified PDDB (indicated with a data set key).
- HONEWOUT does open processing for a new output data set. It gets an unprotected buffer, sets the starting track address, puts the PROC and step/job names in the PDDB, flags the IOT to be checkpointed, and indicates that the data set is open.
- HOOLDOUT handles open processing for old output data sets.
- HOOLDINP does open processing for an old input data set.
- HALCLASS checks the MSGCLASS for hold status by checking the SCAT.
- HALDEST is used to determine the ultimate destination of a held data set, whether it is this node (RC=0) or another node (RC=4).
- $CBIO provides an I/O interface to JES2 control blocks out of the user environment. Exit 8 gives control prior to a write and just after a read to verify the control block.
- The HCBCK routine using $CBIO to checkpoint the JCT and the chain of regular and spin IOTs pointed to be the SJB.
- HCBGM $GETMAINs storage for a HASP control block and returns the address of the buffer in register 1. A $GETBUF TYPE=CB invokes it.
- HCBFM is invoked by a $FREEBUF TYPE=CB and frees the storage of a HASP control block.
- The HJSRETAB routine rebuilds the SDB track allocation block, using the REINCR value in the HCCT.
- $VERIFY is a routine called to verify control blocks after they are read from spool.

HASCSRDS also contains MTTRVAL, which is used to validate track addresses, and HFCLSUB, which does fake closes for input and output data sets.

HASCSRJB

This module contains job-related services. It includes SJBINIT, which gets storage for and initializes an SJB, and SJXB, and and SJF work area. It is called from "job select" and "request job id" to obtain the SJB for a job, a started task, or an initiator. It is also called from HASCDSOC for fake opens of subsystem data sets by the converter. $SJBFIND searches for an SJB requested by the caller. The type of SJB requested can be a TYPE=LOJ, in which case the LOJ-SSIB SJB will be returned. It can also be TYPE=FIRST or TYPE=LAST, in which case either the first or the last SJB chained off the HASB for this address space will be returned.

If TYPE=SSIB, the SJB pointer is obtained from the caller's SSIB. Routine SJBFREE releases the SJB, the SJXB, and the SJF work area storage and removes the SJB from the HASB SJB chain. $SJBLOCK and $SJBUNLK get and release the SJB lock for a particular SJB. The $SJBRQ routine moves the SJB from a current job service queue to one specified by the caller. It places the new queue header address in field SJBQUEUE in the HCCT, and chains the SJB on the new queue. Other routines include TSHABDQ, which, if JES2 abends, dequeues the TSO queued

SJB. HETSOUT ensures that if a task currently processing SYSOUT or processing CANCEL/STATUS commands takes an interruption, it returns to the caller. Finally, $JBIDBLD generates an 8-character EBCDIC job id.

HASCDSS

The HASCDSS module provides data space services. When invoked by the $DSPSERV macro, it creates a data space in the current address space, the JES2 address space, or the master address space (if invoked to create a data space), or deletes an address space (if invoked to delete a data space). The DSPSERV routine is the common entry point for HASCDSS and routes a request to create or delete a data space to the appropriate routine.

HASCDSS issues the $HASP477 message with a return code if an error occurs during processing. It also supplies the following SDWAVRA debugging information in SYS1.LOGREC:

- Component description
- Data CSECT name and partial data area ($DSWA)
- The DSPSERV parameter list.

"Chapter 4. Directory" describes the HASCDSS entry points, the "Diagnosing Routines Without Source" section of chapter 5 provides additional diagnostic information, and *JES2 Messages* lists the HASCDSS return codes under the description for the $HASP477 message.

JES2 Event Trace Facility

The JES2 event trace facility is designed to provide a mechanism for tracing the flow of logic throughout JES2, with a minimum amount of overhead. By means of control tables, individual activities are traced under operator control. These activities can be altered at any time. Trace activity reports are produced at any time for a hardcopy log of the trace table entries (TTEs).

This facility is comprised of two major components:

- $TRACE routine (HASCRIC): Interfaces to individual trace points via the $TRACE macro instruction.
- Event Trace Log Processor (HASPEVTL): This processor formats TTEs to produce a output log of the trace activity and maintains the JES2 trace tables.

Each trace table consists of a prefix section and multiple TTEs, both mapped in the $TTE. JES2 trace tables are obtained via a $GETMAIN macro instruction from a fixed subpool (231) in the expanded common storage area so that tracing may be done from any routine in either the JES2 address space, FSS address space, or any other address space.

$TRACER: The Event Trace Service Routine

The $TRACER routine (in HASCSRIC) services the $TRACE macro instruction. The address of $TRACER is in the CADDR table. The routine can be called from all JES2 assembly environments.

The function of the $TRACER routine is to allocate a trace table entry from a JES2 trace table and to return its address to the caller. When the current trace table is filled, $TRACER locates the next empty trace table. If there is no empty trace table,the routine updates the trace discarding statistics. There are four types of calls:

1. When the caller is a TRACE ID=0 caller, the current table is truncated (that is, allow the trace log processor (HASPEVTL) to process the table so it can be spun at specified time intervals when trace activity is light). TYPE=TRUNC and TYPE=SPIN are the only trace calls allowed with TRACE ID=0.
2. When the caller passes information to be placed in the trace record (using the DATA= and LEN= parameters), $TRACER places the information in the trace table for the caller, and returns.
3. When the caller does not pass information to be placed in a trace table (LEN= is specified, but not DATA=), $TRACER obtains an area of the trace table to put the information of the given length into and returns to the caller holding an ENQ on the table. When the caller has finished putting the information in the table, the caller issues a DEQ to release the table.
4. If the caller is part of the JES2 main task, no ENQ/DEQ serialization is necessary. Because the trace log processor (HASPEVTL) is always managing the tables in the JES2 main task, other PCEs do not need $TRACER to ENQ on the current table to ensure serialization.

If $TRACER is unable to obtain a trace table entry, it means that discarding is in effect (there is no available table to put data into). In this case, discarding statistics are generated and a return code of 4 is passed back to the caller. If a table is available and data was passed, $TRACER issues an ENQ on the table, places the data into it, and then issues a DEQ on the table, and returns to the caller. If the caller did not pass data:

- $TRACER returns to the caller with an ENQ on the table (if the caller is not the main task)
- Register 1 contains the entry of the table in which to put the data
- Register 0 containing the address of the trace table page so that callers (other than out of the main task) can issue a $TRACE macro with the RELEASE option to call $TRAREL to release the table.

Trace tables are threaded in a circular queue. When data is not logged as fast as data is put into the tables, trace data is discarded. Pointers in the $HCCT keep track of table addresses. The CCTTRLGG pointer, or "log pointer", points to the trace table which is currently being logged. The CCTTRTBL pointer, or "curr pointer", points to the trace table which is available for data additions. The CCTTRPLG pointer, or "prvlog pointer", points to the table that precedes the "log" table. Only when JES2 is attempting to decrease the number of trace tables will this pointer be non-zero. If "log" has been set equal to "curr", it means that there are no tables available to put data into. (See the description of HASPEVTL for an explanation of table management and logging.)

If the data that is passed will not fit into the entry, the subroutine TRCKDATA is called, which issues the $HASP381 message indicating that this is an invalid trace entry.

Dynamic addition and deletion of trace tables can be done with operator commands.

USERSUB: JES2 User/Subtask Exit Effector

USERSUB is the exit effector routine used by JES2 to pass control to user exit routines that are called from a JES2 subtask, the user address space (the HASC* modules), or the FSS address space. USERSUB receives control when the ENVIRON= operand of the $EXIT macro specifies either USER, SUBTASK, or FSS.

If tracing was specified via the AUTOTR= operand of the $EXIT macro, USERSUB creates a pre-exit trace record containing:

- The exit number
- The exit name
- The trace environment description (as determined by the setting of the ENVIRON= parameter on the $EXIT macro)
- An indication that this trace precedes the user exit routine call
- The contents of registers 0 through 15

USERSUB uses the exit point number as an index into the exit information table (XIT) which, in turn, points to the exit routine's address in the exit routine table (XRT). Once the user exit routine address is found, USERSUB calls it.

When control returns to USERSUB from the user exit routine, USERSUB saves and checks the return code to see if additional user exit routines are to be called. If there are more to be called and the last exit routine set a return code of 0, USERSUB calls the next user exit routine. This process continues (assuming there are no errors) until all user exit routines have been called, or until a user exit routine sets a non-zero return code. When finished, USERSUB creates a trace record containing:

- The exit number
- The exit name
- The trace environment description (as determined by the setting of the ENVIRON= parameter on the $EXIT macro)
- An indication that this trace follows the execution of the user exit routines
- The contents of registers 15, 0, and 1
- The name of the last user exit routine called

If the return code from the user exit routine is greater than the value of MAXRC specified on the $EXIT macro for this exit, USERSUB issues a 02B abend using the ABEND macro instruction.

$STRAK Service Subroutine

The purpose of the $STRAK subroutine (contained in HASCRIC) is to return to the caller a track address (MTTR) for spool volume I/O. The caller supplies to $STRAK the address of a track allocation block (TAB). There are three types of TABs:

- A major TAB is the control block associated with a track-celled data set. It contains the address of an allocation input/output table (IOT), the MTTR of the last buffer allocated from the track cell, the number of buffers left in the track cell, and the major TAB indication, TABMAJOR.

- A minor TAB is the control block associated with a non-track-celled data set, such as a punch data set. The minor TAB consists simply of the address of an allocation IOT and the minor TAB indication, TABMINOR.

- The master TAB is a special case of the major TAB, identified by the TABMAJOR indicator. It is associated with all data sets in a job that belongs to a non-track-celled SYSOUT class and with the JES2 control blocks that reside on spool volumes in space allocated by track cell. Individual records are written and read and not just track cells.

The major and minor TABs are located in the subsystem data block (SDB) associated with the output data set. The master TAB is located in an allocation IOT, an IOT that contains a list of the track groups assigned to this job (if this is a job IOT) or dataset (if this is a spin IOT).

Each TAB contains a sub-permutation number P that is used in assigning the next buffer from a track group, the TTR of the last buffer in the currently active track group, and the number of buffers left in the cell. An algorithm is used to calculate the placement of buffers in the physical records of the track in such a way that the effect of rotational delay in spool device I/O is minimized. This allows the time required for rotation past interceding records to be used by the central processor for the processing associated with spool operations.

On entry, the TABFLAG field is examined. If the TABMINOR indicator is set to 1, the address of the master TAB is obtained from the allocation IOT pointed to by the TABAIOT field of the TAB. If the TABMAJOR or TABMASTR indicator is on, the address passed to $STRAK is used. In any case, the address of the track group map is obtained from the selected major or master TAB. Next, the TABUFCNT field is examined to determine whether there are any spool buffers left in this track cell. If so, the sub-permutation algorithm is used to calculate the MTTR of the next record to be used, and control is retuned to the caller with the new MTTR; the count of available buffers is decreased by 1.

If no buffers are left in this track cell, the routine determines whether there is room for another cell on this track. If there is, the new track cell is allocated to the caller's TAB, the MTTR is calculated, and control is returned with the MTTR of the first buffer in the newly allocated track cell. However, if the request was accompanied by a major TAB, indicating that the buffer is to be part of a track-celled data set, a check is made to determine whether the size of the new track cell is below the minimum size specified in the TTCM field of the track extent data table (DAS). If the remaining track cell is too small, the routine steps to the next track and attempts allocation.

If there are no tracks left in the current track group, $STRAK must allocate a new track group. At each checkpoint cycle, the KBLOB subroutine in the checkpoint processor allocates a fixed number of track groups. These track groups are represented in the track group blocks (TGBs), which are pointed to by a field in the $HCCT. Therefore, when $STRAK allocates a new track group, it must scan the TGBs for an unused track group that is on a spool volume that the job is eligible to use. If $STRAK finds no eligible track groups by checking the TGBs, it looks at the bit map in CCTMTSPL to find out if there are any track groups available on a volume that is eligible for the job's use. (Though the TBGs indicated that there were no eligible track groups, there may actually be some. If so, they would be indicated in CCTMTSPL.)

If the check of CCTMTSPL indicates that there are track groups available, $STRAK posts ($$POST) the checkpoint processor so that the checkpoint processor can try to allocate the track group after the next checkpoint cycle completes.

If there are no free track groups on any of the job's eligible volumes, $STRAK calls the $STRAK installation spool allocation JES2 exit (exit 12). If this JES2 exit point ($STRAKX) is disabled, has no installation exit routines associated with it or if the last installation exit routine that was invoked returned a 0 or 4 in register 15, $STRAK gets the first available track group from the BLOB, indicates that this spool volume may be allocated, and continues processing. JES2 accomplishes limited spool partitioning in this way.

If a return code of 8 is passed back, register 1 contains the address of a three-word parameter list containing the address of the IOT and the JCT (or 0, if a JCT is not available, which is true in the case of spooled remote messages, multi-access spool messages, and when acquiring a record for the JESNEWS data set), and the address of a 32-byte bit mask indicating the volumes from which the job can allocate space.

When an acceptable track group has been found, the MTT of that track group is placed in a track group allocation entry (TGAE) in the job's primary allocation IOT. If there is no space in the primary allocation IOT, the MTT will be placed in a secondary allocation IOT, if it exists. If there is no space in the secondary allocation IOT or if a secondary allocation IOT does not exist, a secondary allocation IOT is created. The very last TGAE in the most recently created IOT is saved and replaced with the MTT of the track group just obtained. If the last created IOT is a secondary allocation IOT, the IOT is written to spool. $STRAK will then clear the buffer for the secondary IOT, copy the IOT prefix from the primary allocation IOT to the secondary, place the saved TGAE as the first TGAE in this buffer, write the new secondary in the first buffer of the newly obtained track group. Finally, the primary allocation IOT is written to spool.

On exit from the $STRAK routine, the track allocation blocks associated with the request are updated to reflect the current status of the track cell. In the case of a major TAB, the buffer count field, TABUFCNT, indicates the number of buffers left in the current track cell. The TABMTTR field contains the track address of the just-allocated buffer, and the TABSPN and TABMAXR fields reflect the current track status. For a minor TAB, the same fields are updated; in addition, the associated master TAB is updated to reflect the current status of the shared track cell.

In addition to TAB updating, the track group map is always updated to reflect the current status of the track group and the track address (in TCMCELL) of the next available track cell.

HASPAM: HASP Access Method (HAM)

The HASP access method (HAM) is used by all subsystem data set users other than JES2 itself to create and read subsystem data sets.

HAM is VSAM-based; that is, it uses the access control block (ACB) and request parameter list (RPL) rather than the data control block (DCB) to access a data set. (For users of DCBs, a sequential access methods subsystem interface of SAMSI converts each DCB request to an ACB request.)

All GET and PUT requests to HASPAM are treated as requests for sequential access, synchronous processing, and move mode. Chained RPLs are not supported. For some types of subsystem data sets, POINT requests and update-form GET and PUT requests are supported. CHECK and ERASE requests are accepted and treated as no-ops. ENDREQ causes HASPAM buffer truncation on output data sets and no operation on input data sets.

HAM Basic Structure

HAM consists of three logically distinct sections: HASPAMI, SVCHAM, and HAMCEA.

HASPAMI, which operates in user key and state, is branch-entered by the ACB user. HASPAMI attempts to satisfy the user's requirements by moving data into or out of a HAM buffer unprotected from the user. If that buffer becomes full (or empty), HASPAMI issues SVC 111 to invoke SVCHAM.

SVCHAM moves data between the unprotected buffer (UBF) and a HAM buffer protected from the user (PBF) and can initiate I/O via the EXCP macro to write out the PBF (output) or to read into it the next physical record of data (input). SVCHAM works in coordination with HAMCEA.

HAMCEA, the channel-end appendage, does further processing in an attempt to return to EXCP with an indication to restart the channel program. This increases performance by reducing the number of actual EXCPs required to process a data set.

HAM Control Areas

In addition to the RPL and ACB associated with a user request, HAM uses the subsystem data block (SDB), which is the major control area associated with each data set. In addition to other information, the SDB contains:

- Data set status information
- HAM buffer pointers
- A pointer to the input/output table (IOT) which contains the track group map from which direct-access space is to be allocated for this data set
- An input/output buffer (IOB) for EXCP purposes

(The data extent block (DEB) and data control block (DCB) associated with the SDB and IOB are different from the user's; they reside in the subsystem job block and are common to all SDB IOBs).

HASPAMI Structure

HASPAMI is that portion of HASPAM that operates in user key and user state; HASPAMI's address is set in the ACBINRTN field at open time.

HASPAMI consists of three sections. The first section validates arguments and sets up registers; the second section processes user data; and the final section returns a completion code to the user. The final section also invokes exit LERAD and SYNAD, if appropriate. If control block validation fails, the final section of HASPAMI issues an ABEND macro instruction with a system code of 02A and the DUMP option.

On entry, HASPAMI saves registers in the standard user-provided save area. It validates the ACB and the RPL function code and stores the function code in field RPLREQ. HASPAMI continues by entering the appropriate HAM function routine via a vector table. Registers are set up as follows:

Register	**Usage**
6	Unprotected buffer (unless INTRDR)
7	RPL
8	ACB
10	Subsystem data block (SDB) (device control table (DCT) if INTRDR)
11	Subsystem vector table
12	HASPAMI base
13	User-supplied save area

On completion, the HAM function routine branches to label HRPLEXIT with a return code in register 0. HRPLEXIT, which is the final portion of HASPAMI, saves the code in the RPLFDBK field and simulates a post by setting to X'40' the first byte of the event control block (ECB) in (or pointed to by) the RPL. The return code is then analyzed. If either a logical or a physical error is indicated and the user has supplied a valid exit list containing an active entry for the indicated error, the exit routine is entered. (If it is not in the address space, it is loaded first, and the exit list entry modified appropriately.) On entry, registers 1 to 13 are as they were when the user called HASPAMI. The user may return on register 14 to HASPAMI, whereupon registers 14-12 are restored, and HASPAMI returns to the user on register 14.

SVCHAM Structure (SVC111)

SVCHAM is that portion of HAM that operates in supervisor state and key zero in support of a supervisor request by HASPAMI. The request is typically to perform an I/O operation, but it can perform other authorized functions, such as SYSOUT excession processing.

SVCHAM consists of three sections: the initial SVC entry point, link-edited with the nucleus and common to both JES2 and JES3; an initialization section, branched to from the initial entry point; and the main section.

At open time, DSOPEN stores the address of HASPAMI into DEBAPPAD. At an offset of 4 from this address is the address of SVCHAM. When the nucleus portion of SVC 111 gains control, register 1 contains either an RPL address or the complement of an ACB address. Using register 1, SVC 111 points to the data extent block (DEB) with register 2. If the SVC issuer was in problem state with a protection key of 8 or greater, the local lock is obtained and the MVS subroutine DEBCHK is used to validate the DEB. If the DEB is valid, SVC 111 releases the local lock and branches to SVCHAM; if the DEB is invalid, SVC 111 issues an ABEND macro instruction with system code 36F and the DUMP option.

SVCHAM is entered in AMODE 31, because IGC111, the SVC 111 interface routine, uses pointer defined linkage, that is, IGC111 sets the addressing mode of the routine to be called based on the state of the high order bit of the address of the subsystem routine. On entry to SVCHAM, an initialization portion sets up registers and (except for INTRDR data sets) uses register 0 to branch to the appropriate section of SVCHAM to process the request. Registers are set up as follows:

Register	**Usage**
2	DEB
3	ACB
7	RPL
10	Subsystem data block or IRDCT
11	Subsystem vector table
12	SVCHAM base

On completion, SVCHAM sets a completion code in register 15 and returns. To return, it points register 14 to the common SVC exit routine using pointer CVTEXPRO in the control vector table (CVT) and branches on register 14.

HAMCEA Structure

HAMCEA, a portion of HASPAM, operates in supervisor state and key zero under control of a service request block (SRB) provided by the input/ output supervisor (IOS). It processes channel ends resulting from EXCPs issued typically by SVCHAM.

An initialization section saves registers in the IOS-provided save area, points register 10 to the subsystem data block (SDB) and register 11 to the $HCCT, and enters the appropriate section of HAMCEA based on an index byte in SDBCHEND in the SDB.

When a HAMCEA concludes, it restores registers and returns to the IOS at an offset from the address contained in register 14. The offset is 0, when the IOS is to post the I/O operation complete; it is 8 when the IOS is to restart the I/O operation.

GET Processing

Open processing in SSIAOPN in HASCDSOC for an input data set provides an unprotected buffer (UBF) and a protected buffer (PBF) and initiates I/O. Therefore, at its first use, an input data set has a full UBF of data for the user and a full PBF of data to be moved into the UBF when that becomes empty.

When a user issues a GET macro instruction, HAMGET uses the HGMOVE subroutine to satisfy the request. That subroutine returns one of four condition codes:

- *0, 1, or 2*: The request is satisfied.
- *3*: The request could not be satisfied.

If HGMOVE could not satisfy the request, HAMGET calls subroutine HGSPEC, the special-case handler, which will find one of the following situations:

- The data set is at end-of-data. (HGSPEC returns logical error code RPLEODER.)
- The data set had a permanent I/O error.
- The UBF needs to be refilled.

HGSPEC returns logical error code RPLEODER for end-of-data except for an execution batch monitor operation, for which end-of-data means that the monitor's current user job is complete and should be ended.

To end an execution batch monitor user job, HGSPEC uses SVC 111 to invoke SVCXBM. SVCXBM locks the subsystem job block (SJB) pointed to by the batch input data set's subsystem data block (SDB); then it $CALLs HJE000. HJE000 closes the open subsystem data sets of the user job, writes HASP job control blocks, and invokes cross-memory HASPXEQ. After terminating the current user job, SVCXBM $CALLs JBFOUND to get the next joblet.

If the subsystem data set has an I/O error, HGSPEC returns to the user the physical error code RPLRDERD. The error can be an actual I/O error or a failure of a physical record's job key and data set key to match SDBKEY.

Flag BF1EOB indicates that the data set's UBF needs refilling. The channel-end appendage, operating asynchronously with user requests, automatically refills the buffer if BF1EOB is set; therefore, if I/O is active, there is no need to issue an SVC to refill a UBF. In this case, HGSPEC issues a WAIT macro on an event control block (ECB) in the UBF and, when posted by the channel-end appendage (HCEGET), retries the user GET request.

If I/O was inactive, HGSPEC uses SVC 111 to invoke SVCGET, which either sets an I/O error condition (in subroutine HENDREAD) or moves the PBF to the UBF (subroutine HMOVEPU) and initiates a read for the PBF (unless HENDREAD had set flag SDB2EOD). SVCGET then returns to HGSPEC to retry the user GET request.

Operating asynchronously from HAMGET and SVCGET, channel-end appendage HCEGET completes each I/O request and, if possible, starts another request. At channel-end time, if the UBF is not yet empty, HCEGET merely resets flag SDB2IOA and returns normally to the I/O supervisor (IOS). If flag BF1EOB is set, HCEGET uses HENDREAD to check completion of the I/O. It uses subroutine HMOVEPU to move the PBF to the UBF (unless HENDREAD set flag SDB2IOE) and either returns to IOS to start another read operation or (if HENDREAD set flag SDB2EOD) resets flag SDB2IOA and returns normally to IOS. If it refilled the UBF, HCEGET has reset flag BF1EOB (in subroutine HMOVEPU) and used subroutine HPOSTECB to post BFECB either by a compare and swap instruction or by branch entry to the POST macro.

If the condition code returned to HAMGET from HGMOVE was 0, 1, or 2, indicating that the user's request was satisfied, HAMGET returns in RPLRBAR the relative byte address (RBA) of the record it just gave the user, increases by one the contents of field BFRBA for the next user call, and returns normally to the user.

PUT Processing

Open processing in SSIDAOPN provides an unprotected buffer (UBF) for the data set. When a user issues a PUT macro, HAMPUT uses subroutine HPMOVE to truncate trailing blanks from the user data and move the remaining user data into the UBF. HPMOVE returns either of two condition codes:

- *0:* The request is satisfied.
- *2:* More space in the UBF is required to satisfy the request.

If HPMOVE satisfied the user request, HAMPUT increases by 1 the count of user requests against this UBF and adds the new count to the total count of requests against previous UBFs. This total count is compared against the user-specified output limit (specified on the DD statement with keyword OUTLIM=). If the output limit is exceeded, HAMPUT uses SVC 111 to invoke SVCUSO, which, in turn, calls the installation's SYSOUT excession routine, IEFUSO. Based on the return code from IEFUSO, SVCUSO either issues an abend with a system code of 722 and the DUMP option (for return code not equal to 4), or it increases by 1 the user limit in the returned contents of register 1 and returns to HAMPUT.

Before returning normally to the user, HAMPUT increases by 1 the value of BFRBA (its old value had been moved to RPLRBAR by subroutine HPMOVE) and calculates the new length remaining in the UBF.

If HPMOVE was unable to satisfy completely the user's request, HAMPUT uses SVC 111 to invoke SVCPUT. SVCPUT acquires a protected buffer (PBF), moves the UBF into it, initiates I/O, reinitializes the UBF, and returns to HAMPUT. HAMPUT again calls HPMOVE to continue the user request.

SVCPUT performs the following functions:

- Gets a chaining track
- Frees PBFs for which channel end has occurred
- Gets a PBF
- Waits for I/O to complete if no PBF is available
- Moves the UBF contents to the PBF
- Counts records, pages, and bytes contained in the buffer
- Initiates I/O
- Tests for output exceeding user estimates
- Reinitializes the UBF and returns

In every JES2 data set, each physical record contains the track address of the next sequential physical record belonging to the data set; the track address contained in the ending physical record is 0. SVCPUT gets a chaining track by invoking subroutine $STRAK, which assigns a new direct-access physical record address; but if the data set is closed (flag SDB1CLOS), SVCPUT merely uses zeros. The new record address, or 0 is saved in register 8 for later use.

Next, SVCPUT uses subroutine HSPFBFRE to free buffers chained by channel-end appendage HCEPUT on chain word SDBFBF. HSPFBFRE removes the chain of buffers from SDBFBF and frees all but one of them. If there were no buffers, it returns register 1 as 0; otherwise, register 1 points to the buffer it did not free.

If subroutine HSPFBFRE found no buffers, SVCPUT gets a buffer by branch entry to the main storage supervisor. As a safety mechanism, SVCPUT first checks the count of all buffers queued to SDBPBFX, SDBPBF, and SDBFBF; if that count has reached a maximum, or if the main storage supervisor could not get a buffer, SVCPUT waits until HCEPUT has removed a buffer from chain SDBPBF and put it on chain SDBFBF. When posted, SBCPUT starts over by calling HSPFBFRE again.

After obtaining a protected buffer (PBF), SVCPUT initializes (or reinitializes) it and moves into it the contents of the UBF. It uses the compare and swap instruction to chain the PBF last-in-first-out (LIFO) on header SDBPBFX, thus making it available to channel-end appendage HCEPUT for channel program restarting. If at this point I/O is active (SDB2IOA) for the data set, SVCPUT bypasses I/O initiation. But if I/O is inactive, HCEPUT may have found and processed the newly chained subsystem data block (SDB); if it has, it sets the PBF flag BF1IOC. If this flag is off, SVCPUT initiates I/O.

Exit 9 is invoked by subroutine HEXTCALL. Finally the UBF is reinitiated and control returns to HAMPUT, which again uses subroutine HPMOVE in an attempt to complete the user's request.

The PUT channel-end appendage, HCEPUT, operates asynchronously from HAMPUT and SVCPUT. It performs the following processing:

- Moves buffers from SDBPBFX to SDBPBF
- Dechains the first buffer (for which I/O has ended) from SDBPBF and chains it on SDBFBF
- Shows I/O complete by setting BF1IOC in the buffer
- Honors an early post request from SVCPUT
- Restarts I/O on the buffer pointed to by SDBPBF

The buffers queued on SDBPBFX by SVCPUT for I/O initiation by HCEPUT are queued last-in-first-out; that is, they are queued in the reverse of the order in which they should be written. Using a compare and swap instruction, HCEPUT removes the entire chain from SDBPBFX and chains each buffer, again in reverse order, onto the end of chain SDBPBF. (For output data sets, SDBPBF is used only by HCEPUT.) Now SDBPBF is a chain of one or more PBFs in correct writing order.

The first buffer now on chain SDBPBF is the one to which this channel-end condition applies. HCEPUT removes this buffer from SDBPBF and puts it on chain SDBFBF (LIFO, using a compare and swap instruction) to be freed when SVCPUT is again called. But SVCPUT must not free the buffer until HCEPUT has returned to the input/output supervisor and the supervisor (IOS) has freed the buffer; for this reason, subroutine HSPFBFRE frees all but the most recently queued buffer.

HCEPUT shows I/O complete on this PBF by setting flag BF1IOC. Also, if SVCPUT is waiting for HCEPUT to release a buffer, HCEPUT uses a branch entry to the post word, SDBSAVE, and resets the early post flag.

Having completed its cleanup phase, HCEPUT again checks SDBPBF. If that chain header is not 0, HCEPUT restarts the channel program. It does this by converting the JES2 track address MTTR to the MVS format (using subroutine HCNVFDAD) and returning to IOS at an offset of 8. But if SDBPBF is 0, it returns at an offset of 0 to cause IOS to post SDBECB. The common return point for all channel-end appendages resets the I/O active flag, SDB2IOA, if it is to return at an offset of 0.

HCEPUT performs the auxiliary function of checkpointing the output data set's allocation input/output table (IOT) if the data set (or any other data set using the same allocation IOT) was allocated a new track group. SVCPUT has detected a track group change upon return from direct-access allocation routine $STRAK and sets flag IOTICKPT. HCEPUT inspects this flag and writes the allocation IOT in preference to an output buffer.

GET-Update Processing

JES2 allows the MVS converter the user of GET-update and PUT-update when processing the internal text data set.

If HAMGET detects that flag RPLUPD is set in byte RPLOPT2, it branches to label HG100 instead of using the normal path for GET processing. HG100 attempts to satisfy the request with data currently in the data set's UBF, thus avoiding I/O. If the request cannot be satisfied, HG100 truncates and writes the UBF and then uses SVC 111 to invoke SVCGUP. SVCGUP attempts to read the physical record or records containing the logical record requested. It returns to HG100, which uses subroutine HGMOVE to move the record to the caller's area. On entry SVCGUP first ensures that I/O for the data set is complete; it frees buffers that HCEPUT has placed on the chain SDBFBF. Then it acquires a protected buffer, to be used for I/O, and an unprotected buffer, to be used upon return to HG100. The identifiers assigned these buffers are GBF and HBF respectively; a pointer to each is stored in SDBGBF and SDBHBF. Thus the buffers used for GET-update and PUT-update are independent from those used for GET and PUT processing.

The user passed to HAM in word RPLARG a pointer to an 8-byte relative byte address (RBA) previously returned in doubleword RPLRBAR by HAMPUT. SVCGUP moves these 8 bytes to SDBUPRBA, extracts the track address, and converts it to MVS format. Then the routine modifies the channel program to read data, modifies the channel-end appendage index to cause HCEGUP to be used, issues an EXCP macro, and issues a WAIT macro.

Processing initiated by SVCGUP is continued by HCEGUP. At channel-end time, HCEGUP uses SDBGBF to point to the correct protected buffer, validity-checks the job and data set keys of the data just read, moves the protected buffer to its unprotected buffer, and uses subroutine HFINDRBA to find the start of the requested record in the protected buffer. It saves the address of that record in the unprotected buffer at word BFLOC.

The converter is allowed to create very long records of internal text, and the record it updates may also be very long. If the record HCEGUP found is spanned and not complete in this physical record, it continues processing by getting and formatting another GBF and HBF and chaining them first-in-first-out on SDBGBF and SDBHBF. It then restarts the channel to the new GBF, converting and using as a track address the value in field BFNXT of the previous GBF, which has been moved into field BFTRK of the current GBF.

Finally, when all reading is complete, HCEGUP causes the input/output supervisor (IOS) to post the event control block (ECB) for which SVCGUP was waiting. SVCGUP regains control and restores the channel program and channel-end appendage index it had modified; it returns to HG100, which moves the requested record to the user as above.

PUT-Update Processing

JES2 allows the MVS converter the use of GET-update and PUT-update when processing the internal text data set.

If HAMPUT detects flag RPLUPD set in byte RPLOPT2, it branches to HP100 to process the request. As with GET-update, PUT-update first tries the current unprotected buffer (UBF). If the specified RBA is found there, HP100 avoids unnecessary I/O by using subroutine HPMOVE to move the updated record from the user's area into the UBF; then it returns to the user.

But if the UBF does not contain the correct RBA, the content of the user's area is moved into the HBF's area provided by GET-update, using subroutine HPMOVE. HP100 uses SVC 111 to invoke SVCPUP.

SVCPUP moves each unprotected buffer (HBF) to its corresponding protected buffer (GBF) and frees the HBF. Then it moves the entire chain of GBFs from header SDBGBF to SDBPBF and initiates I/O, letting HCEPUT, the normal output channel-end appendage, complete processing. When I/O is complete, SVCPUP restores the channel program and channel-end appendage index to be sure they are correct. It returns to HP100, which returns to the user.

POINT Processing

JES2 offers the capability of using the VSAM POINT macro instruction for the job journal as an input data set and the system messages data set. In addition, some subsystem interface routines take advantage of parts of the HAMPOINT service to reposition data sets.

The user issues POINT with RPLARG pointing to an 8-byte search argument. HAMPOINT upon entry tests that argument; if it is 0, HAMPOINT resets it to the RBA of the first record of the data set.

HAMPOINT uses SVC 111 to invoke SVCPNT, which ends the processing. SVCPNT finds the argument RBA in field BFRBA of the UBF. SVCPNT points correctly for both input and output data sets. On entry, if the data set is open for output, SVCPNT truncates and writes the contents of the unprotected buffer; in any case, it then waits until I/O is complete for the data set.

SVCPNT then sets up an output data set so that it looks like an input data set, getting a PBF, altering the channel program, and setting the channel-end appendage index so that HCEPNT is used.

SVCPNT initiates I/O starting with the track address contained in the RBA provided by the user. Then it waits for channel-end appendage HCEPNT to search to the desired logical record.

The HAM format of RBA is as follows:

Byte	**Usage**
Byte 0	Reserved
Byte 1	M (extent)
Bytes 2-3	TT (absolute track address)
Byte 4	R (physical record)
Bytes 5-7	LLL (logical record)

This format allows a given logical record to have more than one RBA. For example, the principal RBA of a logical record is the MTTR of the physical record within which it resides, followed by LLL with a value of 1, plus the number of logical records preceding it in that physical record. But for pointing purposes, another valid RBA would be the MTTR of the preceding physical record with an LLL value greater by the number of logical records in the preceding physical record.

HCEPNT gains control at the end of the I/O started by SVCPNT. Using subroutine HFINDRBA, it scans the PBF just read for the target record. If the record was not found by HFINDRBA, HCEPNT returns to the input/ output supervisor (IOS) at offset 8 to initiate a read for the data set's next physical record. But if the record was the last physical record, HCEPNT returns to IOS at offset 0, having stored at SDBSAVE a code which causes HAMPOINT to return logical error RPLEODER to the user.

If subroutine HFINDRBA found the desired logical record, HCEPNT moves the PBF to the UBF, sets the address of the found record and the remaining length in the UBF, and returns to IOS at offset 0 to post (POST) SDBECB. SVCPNT passes back to HAMPOINT the address of a completion code routine (or 0), and HAMPOINT returns to the user as described above.

Internal Reader Processing

Control structure and processing are different for internal reader data sets from those for normal subsystem data sets. Instead of a subsystem data block (SDB), an internal reader device control table (IRDCT) controls processing. (IRDCTs are constructed in the common storage area or CSA by module HASPIRMA.) Actual I/O is performed not in the user memory but by JES2 in its memory (module HASPRDR). Of the two buffers used during internal reader operation, the unprotected buffer is obtained and maintained by HAM, but the protected buffer resides in a JES2-managed data space.

HAM supports the PUT and ENDREQ operations to an internal reader. PUT is used to supply to an internal reader with a card image of a JCL stream, an input stream data set, or one of two special control statements (/*EOF, which requests immediate queuing of the preceding job, or /*DEL, which requests immediate deletion of the preceding job). ENDREQ is used to request that the preceding job be immediately queued and that the subsystem-assigned job identifier of 8 bytes be returned in the request parameter list (RPL) at RPLRBAR. ENDREQ is used by module IEFJSWT; the job identifier obtained by IEFJSWT is placed by MVS in the subsystem interface block (SSIB) at SSIBJBID. It is used by the job select subsystem interface (SSIJSEL) when called by a master-scheduler-started initiator to select a started task (for example, a MOUNT command or a TSO logon).

ENDREQ adds the card image of /*EOF to the internal reader data set, but so does a CLOSE request, unless ENDREQ had immediately preceded the CLOSE. The /*EOF and /*DEL statements are recognized and removed by HASPRDR.

HAM recognizes internal reader processing because a flag (bit 5) has been set in the high-order byte of ACBINRDR by SSIDAOPN HAM branches to HPIRDR or to HENDI for PUT or ENDREQ requests. Both of these routines, as well as SSIDACLO (the close subsystem interface), use the common subroutine HINTRDR to perform most of their work.

HPIRDR examines the card image for a /*EOF or /*DEL statement and sets appropriate flags in register 2 and sets the address and length of the statement in registers 2 and 3. HENDI sets in registers 2 and 3 the ENDREQ flag (HOSCLOS sets the CLOSE flag) and the address and length (5 bytes) of a /*EOF statement.

HINTRDR recognizes five processing cases:

- Normal PUT
- PUT of the /*EOF statement
- PUT of the /*DEL statement
- ENDREQ
- CLOSE

In all cases but CLOSE, if no unprotected buffer exists, HINTRDR uses SVC 111 to invoke SVCIRD to get a buffer and store its address at RIDUBF in the IRDCT. (For CLOSE, no action need be performed; HINTRDR returns.) If there was a buffer, or on successful return from SVCIRD, HINTRDR verifies that the UBF contains sufficient space for the card image. If there is insufficient space, it invokes SVCIRD to move the UBF contents to the PBF and to reinitialize the UBF; to SVCIRD the call appears to be for a normal PUT.

HINTRDR moves the new card image into the UBF, updates the UBF pointer and remaining length, and for a normal PUT returns to the caller, who returns normally to the issuer of PUT. But if the operation is not a normal PUT, HINTRDR invokes SVCIRD, specifying the control flags it received. NOTE that in these cases the last statement in the UBF is always /*EOF or /*DEL. Finally, upon return from SVCIRD, HINTRDR either returns to its caller, who returns normally to the issuer of PUT or supplies the job identifier from RIDJOBID and returns to the issuer of ENDREQ; or returns normally to the caller of the CLOSE subsystem interface.

SVCIRD, a portion of the internal reader logic within HASPSSSM, operates in zero protect key and supervisor state. SVCIRD performs either of two functions: initialization or normal processing. It uses the contents of RIDUBF to select between these functions.

If no unprotected buffer is present (RIDUBF=0), SVCIRD performs initialization and returns. To initialize an internal reader, SVCIRD obtains and initializes an unprotected buffer, stores the pointer to the buffer at RIDUBF, resets flags DCTHOLD, DCTDRAIN, and RIDEND in the IRDCT, and informs JES2 via a cross-memory post (&XMPOST) that an internal reader has become ready. The post starts JES2 processing in HASPRDR, which asynchronously performs its share of internal reader initialization and waits for data. Meanwhile, SVCIRD has returned to HINTRDR, which continues processing in user key and state.

If no entry RIDUBF is nonzero, SVCIRD performs normal processing. First, it waits until HASPRDR has posted to indicate that it is ready to receive data or that an error condition exists. Then, after checking for errors, SVCIRD moves the contents of the UBF and reinitializes the UBF; it posts ($XMPOST) HASPRDR, causing that routine to begin processing data asynchronously. Finally, for all PUT requests, SVCIRD returns normally to HINTRDR.

For ENDREQ and CLOSE requests, SVCIRD processes further before returning. It first waits until HASPRDR has finished its processing and stored the job identifier of the preceding job in RIDJOBID; then it performs conclusion processing. It frees the unprotected buffer, sets flag DCTHOLD for JES2 control service routine $GETUNIT, sets flag RIDEND to cause HASPRDR to perform its share of conclusion processing, does a final post ($XMPOST), and returns.

HASPCON: Console Support Services

Console support services consist of console support routines and a communication task routine. The functional descriptions of these routines follow. Console support routines are described first.

$WTOR: Write To Operator Routine

The $WTOR routine receives control when a JES2 processor executes the $WTO macro. The routine filters out console routines with a low importance level, and gets a console message buffer (CMB) from the CCTCMBFQ or $WCOMRES queue, copies the user's parameter list and message into the CMB (performing any required job information editing in the process), and queues the CMB to the $BUSYQUE for local unit control module ID (UCMID) and logical console routed messages. For messages directed to other systems and remote work stations, the routine time stamps the CMB and queues it to the $BUSYRQ rather than the $BUSYQUE.

Upon entry, the WSCREEN subroutine is called to expand short form $WTO macro calls to extended form and to determine which console routines are to receive the message.

The $WTOR routine attempts to get a CMB and if successful, adjusts the CCTCOMCT counter. The resource manager (HASPRESM) in HASPMISC issues message $HASP050 if the count of free CMBs falls below the specified threshold. If the request is for a multiline write-to-operator (MLWTO) response and the owner of the MLWTO is not the current processor control element (PCE), the $WTO request is rejected as though no CMBs were present. MLWTO ownership is obtained for the PCE (if required) and an attempt to get a CMB from the $WCOMRES or CCTCMBFQ queues is made. If both queues are empty, the attempt fails. Otherwise, processing continues with the MLWTO ownership being relinquished when the message is the end line for the MLWTO.

If the request is for other than an MLWTO response and there is a MLWTO owner, an attempt to get a CMB from the CCTCMBFQ queue is made; if there is no MLWTO owner, the attempt includes the $WCOMRES queue as above. If the request is for an action message, (CLASS=$DOMACT), CCTCOMCT is decreased by 1. If the attempt to get a CMB or if CCTCOMCT goes below a specified minimum value, the CMB (if any) is returned to the CCTCMBFQ and the attempt fails.

If the attempt to get a CMB fails, $WTOR either issues a $WAIT macro for the calling processor (WAIT=YES was specified in the $WTO macro) or returns with the condition code set to 0 (WAIT=NO was specified in the $WTO macro).

The WBLDCMB routine is used to format the fields within the CMB. Fields that are formatted are the time stamps for messages to remote work stations, control flags, console ID, and routing fields. In addition, the job name and job ID fields associated with the message are set, if required. If the job ID is not set, the CMBJOBID field is set to blanks.

The message ID is edited and inserted into the CMB and, if the job ID is contained within the caller's message, the CMBJOBID field is set from the caller's message area. The message text is inserted into the CMB along with the total length. If the message is a request for action, the register save area for register 1 is set to indicate the CMB address. The CMB is placed on the $BUSYQUE, and the

$HASPWTO routine is posted if remote routing is not specified. If remote routing is specified, the CMB is queued in first-in-first-out order within priority on the $BUSYRQ, and the remote console processor ($MCONPCE) is posted ($POST). Note that remote routing is associated with either an RJE work station or an NJE member of the network.

The routine returns control to the caller after restoring registers and setting a nonzero condition code.

$WTO JES2 Exit Point

The WTOEXIT JES2 exit point allows the installation (through installation exit routines) to alter, delete, or reroute a message. If the user deletes the message (indicated by a return code of 8), the HASPWQUE routine is bypassed and the CMB is freed via the $FRECMB macro.

This exit is entered before the CMB, containing the message, is put on the queue by routine HASPWQUE.

The return codes from this exit are:

0 - Continue normal processing
4 - Continue normal processing
8 - Free CMB and discard message

HASPWQUE: Special Purpose CMB Queueing Routine

The HASPWQUE routine is entered from console service processors or from $WTO service routine ($WTOR). The purpose of the HASPWQUE routine is to queue CMBs to either local destination queue ($BUSYQUE) or remote destination queue ($BUSYRQ).

CMBs for remote destinations are queued on $BUSYRQ in priority order, and the remote console processor is posted ($POST WORK). CMBs for local destinations are placed on $BUSYQUE last-in-first-out, and $WTOECB is posted. $WTOECB is posted either directly by using the compare and swap instruction or via an MVS POST macro. ($WTOECB is the event control block (ECB) through which the JES2 communications task, $HASPWTO, waits for work.) Because CMBs are removed from $BUSYQUE asynchronously by the JES2 communications task, they are placed on the queue using compare-and-swap logic to ensure synchronization when adding or deleting them.

$WTOCR: WTO With User-provided CMB

The $WTOCR routine receives control when a JES2 processor issues the $WTO macro with CMB=YES specified. The function of the routine is to filter out console routings of low importance and then enter the $WTOR service routine for functions common with $WTO service.

On entry, the routine uses the WSCREEN subroutine to expand the short form $WTO macro calls to extended format and determine the console routings of the message. If the request is for a multiline write-to-operator (MLWTO) response, an attempt is made to own the MLWTO function unless it is already controlled by the calling processor control element (PCE). If it is owned by another PCE, control is returned with the condition code set to 0. (The user may wait for the CMB and try again when posted or perform other functions). Control is passed to the WBLDCMB routine contained within the $WTOR routine.

$GETCMBR: $GETCMB Service Routine

The $GETCMBR routine receives control whenever a $GETCMB macro is executed by a JES2 processor. The routine reduces the contents of the CCTCOMCT field by a specified amount, dequeues a console message buffer (CMB) from the CCTCMBFQ queue, and returns to the user with register 1 pointing to the first of a queue of CMBs. Message $HASP050 (resource shortage) is issued if the count of free CMBs falls below the specified threshold. If all of the required CMBs could not be obtained, $GETCMBR returns to the caller if WAIT=NO was specified; otherwise, $GETCMBR waits ($WAIT) until sufficient CMBs are available.

JES2 processors use the $GETCMBR routine when a CMB is required to hold a command to be queued or a message to be displayed later by a $WTO CMB=YES macro instruction and when the processor is to honor the action without risk of later having to wait. If the content of the caller's register 0 is not 0, the absolute value in register 0 is subtracted from CCTCOMCT and compared against 0 (if the contents of register 0 are positive) or 2 (if the contents of register 0 are negative). If the result of the subtraction indicates that the new CMB count is low, and if WAIT=NO was specified, control is returned after setting register 1 and the condition code to 0; this means that no CMBs were obtained. If the result of the subtraction is positive (indicating that CMBs are available), the new CCTCOMCT is set, and an attempt is made to remove the required number of CMBs from the CCTCMBFQ queue. If $GETCMBR is unable to to get all of the CMBs, the ones that were obtained are returned, the CCTCOMCT is increased by the amount that it was originally reduced by, and, if WAIT=NO was specified, control is returned indicating no CMBs were obtained. If all CMBs were obtained, control is returned after setting condition codes to nonzero; register 1 points to the first of a chain of CMBs obtained for the caller.

$DOMR: $DOM Service Routine

The $DOMR service routine receives control when a JES2 processor executes the $DOM macro. The routine turns off the action flag in the console message buffer (CMB), scans the $DOMQUE for the CMB and, if the CMB is found to be on that queue, issues a DOM macro for the message and frees the CMB through a call to the WFREE subroutine.

The $DOMR routine is entered by a JES2 processor that has previously executed a $WTO CLASS=$DOMACT macro instruction. Register 1 contains the address of the CMB that contained the message returned to the processor by the $WTOR or $WTOCR routine that handled the $WTO. $DOMR sets to 0 the action indicator in the CMB, increases CCTCOMCT by 1, and posts processors for the CMB. An attempt is made to locate the CMB in the $DOMQUE. If the CMB is found, it is freed via the WFREE subroutine and a DOM macro is executed using the identification saved in the CMB by the $HASPWTO subtask. Control is then returned to the caller.

$FRECMBR: $FRECMB Service Routine

The $FRECMBR routine receives control when a JES2 processor executes the $FRECMB macro to release a console message buffer (CMB) that is no longer required. $FRECMBR calls the WFREE subroutine to free the CMB, updates the free count, posts the processor ($POST CMB), and returns control to the caller.

WFREE: Free CMB Subroutine

The WFREE subroutine is used by the $FRECMBR routine, the $DOMR routine, and the $HASPWTO subtask. If the $WCOMRES queue is empty, the console message buffer (CMB) is placed on that queue and control returns to the caller.

If the $WCOMRES queue is not empty but the SVTCMBRQ queue is empty, the CMB is placed on the SVTCMBRQ queue; otherwise, the CMB is placed on the CCTCMBFQ queue. (The SVTCMBRQ is used by the SVC 34 exit routine when attempting to process a JES2 command when the CCTCMBFQ queue is empty.) Control is then returned to the caller.

$HASPWTO: Communication Task Routine

The $HASPWTO routine processes operator messages queued by JES2 for display on operating system consoles. The routine removes the message from the queue, converts the message into the WTO parameter list format expected by the operating system, and displays the message. SVC 35 is used to display all messages other than operator commands to MVS. SVC 34 is used for MVS operator commands and messages to TSO users in order to pair the command or message to MVS. $HASPWTO is attached as a subtask to allow SVCs 34 and 35 to be issued without requiring the JES2 main task to wait until the requested service is complete.

All JES2 $WTO messages directed to operating system consoles are queued to $BUSYQUE and are serviced by the JES2 communications subtask. When a console message buffer (CMB) is placed in the $BUSYQUE, the main task posts ($POST) the subtask. The communications routine is entered and removes all CMBs from the $BUSYQUE, requeuing those CMBs to the $CONWKQ queue in priority order. The routine then examines $CONWKQ. (If no eligible CMBs are found on the queue, the routine waits for new CMBs and, on regaining control, repeats the $BUSYQUE scan.) If the routine has been entered to continue a multiline write-to-operator (MLWTO) response, all other requests are ignored until the last line of the MLWTO has been processed.

For normal WTO messages, the contents of the CMB are formatted into a list-form OS WTO. If the CMB includes a UCMID, indicating that the message is for a specific console, the WTO is altered accordingly. Out-of-line areas are detected and result in MLWTO formatting of the message.

The message text is copied from the CMB to the WTO field. Control fields are shifted to complete the WTO parameter list for normal logical console and for UCMID WTOs. If the message is for a TSO user, a TSO SEND command is created and sent via SVC 34.

If a UCMID request indicates that a command is present, the multiple console support (MCS) flags are set to 0 and SVC 34 is issued; if the UCMID indicates X'FF', the command contains a start initiator command, and on successful execution of the command, the routine sets the appropriate partition information table (PIT) flags to indicate that the start was accepted. The CMB is freed, and $HASPWTO returns to its initial entry point to repeat the $BUSYQUE scan. If the CMB contains a message for a logical console and the $DOMACT flag is on, the WTO parameter list is altered to indicate that immediate action is required. The WTO SVC is issued, and the DOM identification is saved in the CMB. The CMB is queued to the $DOMQUE queue, the asynchronous I/O processor is posted ($$POST), and the routine reenters its $BUSYQUE scan.

When all CMBs have been processed, the routine calls the WFREE subroutine to free each CMB and posts the main JES2 task ($$POST CMB).

Finally, the WTO macro is executed. If the request is for an MLWTO, the connect ID is saved and an MLWTO in process flag is set or reset, depending upon whether the endline indicator appears in the WTO after formatting. $HASPWTO then returns to its $BUSYQUE scan to await more CMBs to process.

HASPTRAK: Track Management

HASPTRAK manages spool track space. It consists of:

$TRACK
$TGMSET
$BLDTGB
$PURGER
HASPVPRG

Direct-Access Space Allocation

Direct-access (spool) space allocation is based on a technique of mapping bits in a table to groups of tracks on physical devices. A master map ($TGMAP) is defined that represents the available number of track groups across all spool volumes. Each bit in the map defines a contiguous set of tracks (and records) for a particular volume. If a bit is on (1), it indicates the availability of a particular track group. If a bit is off (0), it indicates that the particular track group has been allocated. The recording of the allocated track group is through a map (table) supplied by the user that is the inverse of the master map. A bit that is on indicates that the corresponding track group is allocated to the owner of the map. A bit that is off shows no allocation.

The master map is constructed at JES2 initialization. The size of the master track group bit map is determined from the TGNUM initialization parameter on the SPOOLDEF statement. The number of bits in the map, which represents a particular spool extent, will vary according to the size of the SYS1.HASPACE data set. Spool volume extents can be added to the master track group map until space in the map is exhausted. The characteristics of a spool volume are recorded in a direct access control blocks (DAS) (one for each volume), which is also constructed at JES2 initialization or at dynamic addition of a volume. Included in the DAS are: number of tracks per cylinder, data extent block (DEB) extent number, number of records per physical track, number of track groups, number of tracks per group, offset from the beginning of the master map to the first byte of the master map representing the extent defined by the DAS, number of bytes in the map for the extent, the relative track address of the beginning of the SYS1.HASPACE data set, and the address of the rotation position sensing (RPS) table for the mode of direct access.

Track groups are supplied to users through the track group block located in common storage as part of the subsystem vector table (SSVT). Each entry in the track group block (TGB) represents a track group as allocated from the master map. An entry consists of an MTTR (M = module or extent number, TT = relative track number, R = record number), offset of the byte in the master map containing the bit representing the track group, an offset to the JQE, and a byte that contains the bit representing the track group.

Removal (allocation) of a track group from the TGB is accomplished by two subroutines: $TRACK, described below, and $STRAK, in module HASCSRIC. The replenishment of the TGB is performed by the JES2 checkpoint processor, which calls a local subroutine (KBLOB), defined along with the checkpoint processor, in the HASPCKPT module of JES2.

To meet the performance requirements of the IBM 3800 Printing Subsystem, a second level of direct-access spool space allocation exists. Each spool track is logically divided into track cells, each containing two or more buffers. Track cells, rather than individual buffers, are allocated to data sets, and each output record that is produced is placed in a track cell allocated to the data set being created. When the data set is to be retrieved from the spool volume being written to the output device, channel programs read track cells rather than individual output records in a single I/O operation. (For additional information on the use of track cells, refer to the descriptions of the $STRAK subroutine in HASCSRIC, the $TRACK subroutine described below, and the discussion of HASPPRPU in this section.)

$TRACK Service Subroutine

This subroutine provides track group allocation to JES2 processors being executed in the JES2 address space. The general operation of $TRACK is identical to the HASCSRIC $STRAK subroutine, which provides track group allocation to users of JES2 services operating outside the JES2 address space. There are three major differences between $TRACK and $STRAK: (1) $TRACK operates on behalf of JES2 processors and uses $WAIT logic to force the caller into a JES2 wait when the track group block (TGB) needs replenishing. (2) When control is returned to the caller, the condition code is set to indicate the status of the TGB entry allocation. Condition code 0 is set if a new track group has been allocated for this call of $TRACK. and a track group allocation entry (TGAE) containing the MTT of the allocated track has been created. If no more available TGAEs exist in the allocation input/output table (IOT), $TRACK allocates a secondary allocation IOT. A non-zero condition code is set if an old track group has been used in the allocation of the return MTTR. (3) The mask for spools used, in the JQE, is updated if a new track group is obtained.

Like $STRAK, $TRACK incorporates a JES2 exit point. However, $TRACK uses exit points TBLOBANJ and $TRACKX (for exit 11) for installation spool partitioning purposes.

$TGMSET: Set Bit in a Track Group Map

This routine is passed a track group map address in register 0, and a MTTR value in register 1. A parameter list, which is pointed to by register 15, specifies whether the bit is to be turned on or off in the track group map. The parameter list also specifies whether or not $TGMSET should issue a $QSUSE macro.

This routine is also passed a parameter flag indicating that error checking is to be performed. If the bit to be set off is already off, or if the spool space to be marked does not belong to this job, error messages are issued. If the bit to be set is set on in the bad track group map, the bit will not be set off in the master track group map and the MTT is passed to $BLDTGB. $TGMSET exits with the address of the byte affected in the map in register 1 and the bit number in register 0.

$BLDTGB: Build a Track Group Block

This routine builds a track group block (TGB) for any bits that are set on in the provided track group map pointed to by register 1. The TGB is then queued to the $SPOOLQ, for later processing by the HASPSPOL processor. The $BLDTGB routine then posts ($POST) the HASPSPOL processor.

$PURGER: Direct Access Space Purge Routine

The $PURGER service routine is entered upon execution of a $PURGE macro instruction. Following a SAF call, the routine issues the $QSUSE macro to access the required checkpoint data, finds the master track group map, and frees that portion of the master that represents the track groups assigned to the user whose task is being purged.

The $PURGER service routine then marks any track groups in the bad TGM as being allocated in the master track group map (TGM). It then queues any new bad track groups to the HASPSPOL processor, via the $BLDTGB service routine and clears the users track group map. The routine then posts (via a $POST macro) the checkpoint processor, causing a JES2 checkpoint to be taken to reflect the new track group assignment.

During this purge processing, track groups that are allocated to the job or data set are normally returned to the master TGM. Any track groups that are defective will not be returned to the master TGM. However, the master TGM will indicate that these track groups are allocated; the only track group map that reflects that they are bad track groups is the bad TGM.

The bad track group map ($TGBAD) reflects the condition of the track groups in the master TGM. When an I/O error is encountered for a job or data set, the associated track group is immediately flagged "bad" in $TGBAD; the track group will remain allocated to the job.

When the job or data set is purged, $PURGER returns the track groups allocated to this job or data set by passing the $TGMSET routine the TGAE values for the track groups. $TGMSET then sets the bits in the master map to reflect the changes. If the bit to be changed is set on in the bad track group map, the bit is not turned off in the master track group map and the MTT is passed to $BLDTGB.

An attempt is made to recover bad track groups from the job or data set being purged. These track groups are determined by logically 'ORing' the IOT TGM with the bad TGM and then exclusively 'ORing' the results (which are in the IOT TGM) with the track group map.

If $PURGER finds a spin IOT, it marks it as reusable so that HASP allocation support can reuse the spool block for a new spin data set for the same job.

HASPVPRG: Purge Processor

HASPVPRG frees all tracks acquired for a job, queues an SMF type 26 purge record for output (and a JMR buffer if an exit is found), and notifies the operator that the job is purged. When a data set is deleted, a SAF call is issued to audit the deletion. (A SAF call is not made for the purging of any jobs marked non-selectable due to invalid work selection criteria.)

HASPVPRG: When the purge processor is first entered, it issues a $GETBUF macro instruction to acquire storage for two buffers: one to contain a job control table (JCT), and one to contain an input/output table (IOT).

VGETJOB: At this label, a $QGET macro instruction is issued to obtain a job to be purged from the $PURGE queue. If the queue is empty, VGETJOB releases its buffers with a $FREBUF, if not already released, and waits ($WAIT PURGE) for a job to be queued. When the wait is satisfied, VGETJOB is entered again to determine whether there is a job to be purged.

When the purge queue does contain a job to be purged, VGETJOB issues a $GETBUF for the JCT/IOT, if necessary, and then issues a $ACTIVE macro instruction to indicate that a processor is active. VGETJOB then prepares for I/O.

VGETJCT: VGETJCT prepares for JCT read processing by setting the track and buffer address for the eventual $EXCP. VGETJCT checks the JQEFLAG3 to see if the JQE is associated with a system data set. If the JQE is associated with a system data set, then no JCT exists for it; reading a JCT is bypassed. If the JQE is not associated with a system data set, VGETJCT checks JQEFLAG4 to see if a force purge of the job is required; if it is then if the volume on which the JCT resides is unavailable reading of the JCT is bypassed. Otherwise, VGETJCT invokes the $JCTIO routine to read the JCT. Calls are made to VIOTPRG to purge tracks.

After $JCTIO finishes processing or as a result of bypassing the read of JCT. VGETJCT issues a $WTO macro instruction to schedule message $HASP250, indicating to the operator that the job has been purged except if JQEFLAG4 indicates that this JQE is a copy of a moved job. After this $WTO, VGETJCT deletes the job from the purge queue ($QREM) and waits for the checkpoint to complete ($WAIT CKPT).

When the checkpoint is complete, a check is made for a JCT; if it doesn't exist, VIOTSPCL purges the IOT. If a JCT exists, a check is made if the JCT was read without an error; if it was, VNOERROR is invoked. If there was a read error, the $DISTERR macro instructions are issued to indicate the error and then a $DORMANT is issued to indicate that job processing is complete for this JES2 processor. VGETJCT a branch is made to the beginning of the main processing loop at VGETJOB to wait for the next job to be entered on the purge queue.

VNOERROR: If the correct JCT was read successfully, VNOERROR returns the job's tracks through calls to VIOTPRG. First, the spin IOTs, pointed to by JCTSPIOT, are read and the tracks represented in their track group maps are returned. Then the tracks represented in the track group map of the first regular IOT are returned and a branch is taken to VIOTPRG.

Before the tracks are purged, however, the allocation IOT is checked to see if the "in purge" bit is on. If it is, only the JQE is removed.

VIOSPCL: VIOSPCL purges the IOT belonging to a special system data set JQE (where JQETRAK = IOT track address) or handles the case when no JCT is available for a job. When purging the IOT, VIOSPCL invokes VIOTPRG to perform the I/O and purging. When no JCT is available for a job, VIOSPCL issues a $DORMANT macro instruction to indicate to the JES2 dispatcher that a JES2 processor has completed processing of a job and is going into a dormant state.

VIOTPRG: VIOTPRG performs the actual IOT purge requested by the main processing loop; it issues a $EXCP macro instruction to read the specified IOT, verifies that the correct IOT has been read, and determines that an allocation IOT has been read. (Other IOTs are ignored; VIOTPRG returns to its caller, unless an unrecoverable error occurs as the IOT is being read.) If a valid allocation IOT has been successfully read, VIOTPRG issues a $PURGE macro instruction to free the allocated tracks and returns to the caller.

If an I/O error occurs in reading the IOT, VIOTPRG issues a $DISTERR macro instruction to enter the disastrous error routine, indicating VDSTER as the label at which the error was detected.

VSMFPRG: When all of the job's direct-access tracks have been freed, a $GETSMFB macro instruction is issued to obtain a buffer; the macro is issued a second time if a user exit is to be taken. The common exit parameter area and the job management record (JMR) portion of the JCT are preserved in one buffer, and a type 26 SMF record is created and saved in a second buffer. A $QUESMFB macro instruction is issued to add the SMF buffer to the queue of busy SMF buffers and to post the SMF buffer management subtask. If the user exit is to be taken, the JMR and SMF buffers are chained together before the $QUESMFB macro instruction is issued. Otherwise, the buffers are not chained together, and the JMR buffer is ignored.

HASPCOMM: Command Processor

The JES2 command processor receives all JES2 commands entered from acceptable local or remote JES2 input sources. The processor is responsible for decoding each command and performing the processing necessary to cause appropriate action for the operator's request.

Command Authority Checking

SAF calls are made during JES2 command processing to perform the RACROUTE REQUEST=AUTH security function for all JES2 operator commands. JES2 issues the $SEAS macro which issues the SAF call. IEECB920 sends the following return codes back to JES2 indicating how to proceed with command processing:

- **Accept** - Process the command; bypass normal JES2 console authority checking.
- **Reject** - Do not process the command.
- **No decision** - Either the security package (RACF) or command authority checking was not active. No audit record is written by RACF and JES2 performs the default console authority checking. RACF returns a reason and return code indicating the reason for "no decision".

JES2 commands can be entered into the system from many different sources. Following is a list of ways and any special considerations regarding command auditing.

Commands from MCS consoles

JES2 receives its operator commands from MCS on the SVC 34 SSI call. This call includes the security token of the operator logged on to the console from which the command was entered. The token (passed in the SSOB extension) is included with the other command data passed to the JES2 address space for command processing.

Commands from readers

JES2 accepts commands imbedded in input streams from readers. If the commands are JES2 commands, JES2 processes them internally. In this case, JES2 passes to $SEAS the security token that was associated with the reader when it was last started. If the reader is started by an operator, the reader's security token contains some information from the operator's token. If the reader was started from initialization options, the reader's token contains the JES2 security information.

If the commands are MVS commands, JES2 does not process them. They will be recognized by the converter and passed to MCS command processing via a MGCR. JES2 will issue a RACROUTE REQUEST=VERIFYX ENVIR=CREATE call for the user who submitted the job before invoking the converter/interpreter (C/I). This allows the SAF call made by MGCR processing when it receives a command to check against the security level of the user who submitted the job containing the command. Upon return from the C/I, the RACROUTE call is issued again specifying ENVIR=DELETE.

Commands issued from SVC 34

A program (for example, SDSF) may send a command to JES2 by issuing an SVC 34. JES2 uses the security token passed across the SSI for these commands.

Commands from other nodes

The remote console processor (HASPRTAM) issues a SAF VERIFYX call to obtain a security token for the connected node and queues the command to HASPCOMM. Whenever a command is received from another node, HASPCOMM calls $SEAS passing it the security token associated with the transmitting node. If the $SEAS return code is 4, JES2 validates the source.

Commands from RJE devices

A RACROUTE call is issued for each RJE device in the system when the device signs on to create its security token. Whenever a command is received from the RJE device, HASPCOMM calls $SEAS passing it the security token associated with that RJE device. If the $SEAS return code is 4, JES2 validates the source.

Commands from internal readers

Commands from internal readers use the token associated with the user allocated to the internal reader.

Internally-generated commands

JES2 sometimes generates MVS or JES2 commands as part of its processing. These commands are not passed to an operator. They go through the normal command processing path, including the call to the MCS common command authorization routine.

However, the user security environment used for the check is the JES2 address space, not that of an operator. The installation must ensure that JES2 is running with a "trusted" security classification so that commands required by JES2 will not be rejected. If a command is rejected, a message is sent to the consoles receiving security route code messages and the command will not be processed. This will result in erroneous system operation because JES2 expected the command to complete.

Commands from the JES2 initialization stream

JES2 allows commands to be entered from within the initialization stream. The SAF call is made using the security token associated with the JES2 address space.

$VS commands

The JES2 $VS command, used to enter an MVS system command, is disallowed from an MCS console because the console operator can issue MVS commands directly without going through JES2.

The $VS command is permitted from readers, the initialization stream and from within automatic commands. When $VS is issued from a card reader, the security token of the reader is associated with the $VS command. When issued from the initialization stream, the token associated with the automatic command is JES2's security token. When issued from within an automatic command, the token associated with the automatic command is associated with the $VS command.

Display authority is the default command authority for card readers and internal readers. Therefore, the $VS command (in addition to other system commands) is not allowed from the devices unless the installation specifically allows it.

Automatic commands

Commands that manage automatic commands are audited like any other commands from a console. Commands entered as a result of automatic command initiation are also audited, but the security token used on these audits is the one for the last operator to enter a $T A command that changes the automatic command.

Multiple commands on an input line
JES2 separates multiple commands on an input line and processes them with separate calls to the MCS command authorization routine for each command.

Display and list commands
All JES2 display and list commands have read-level authority (except for $L SYS, which requires control-level authority).

Command text
JES2 calls the MCS command authorization routine during command processing to audit command text following exit 5 invocation, which allows modification of text finish processing.

Exit 36, invoked prior to command auditing, can provide a return code indicating that JES2 is to bypass the MCS command authorization routine and/or not process the command. In this case, it is the exit's responsibility to issue the SAF call or write an audit record.

Exit 37, invoked after command auditing, can provide a return code indicating that JES2 is to ignore authorization routine's return code.

HASPCOMM Processing

The command processor is initially entered at the beginning of control section HA$PCOMM. The initial entry activates the last phase of start initiator command processing. This phase schedules the queueing of console message buffers (CMBs) containing commands to the operating system to start desired initiators. Subsequent entries are returned from the various command sub-processors with optional requests to display the OK message or other messages contained in the command area (COMMAND) of the processor control element (PCE). After any requested replies have been displayed, the JES2 console message buffer queue $COMMQUE is examined for the presence of the next command to process. If $COMMQUE is empty, the CCTCOMMQ is examined, and any CMBs queued to that queue are removed and requeued to the $COMMQUE in reverse order. If CCTCOMMQ is empty, $COMMQTP is examined and any CMBs are handled in the same manner. If $COMMQTP is also empty, the automatic command queue is examined for an expired automatic command element (ACE). If an element exists, the commands contained in the ACE are placed in a CMB, the CMB is queued to the $COMMQUE, and the ACE is freed (if no interval exists for the element) or placed back into the active queue with a new expiration time (if an interval does exist). If attempts to obtain new commands are unsuccessful, the processor waits ($WAIT WORK) and repeats its dequeuing efforts when it is posted. When a CMB is successfully obtained, the command edit routine is entered.

HASPCOME: Command Edit Routine

The following text describes the function of the HASPCOME routine and its role in the overall command processor function.

Command Edit and Break Out

Selected control fields are moved from the console message buffer (CMB) into a list form of the $WTO macro contained within the processor control element (PCE) work area of the command processor. Command authority information is extracted, and the $WTO list is set up for responses to the command contained in the CMB.

The command area (COMMAND) of the command processor's PCE is cleared to blanks. If the CMB is flagged as containing a formatted global networking command (CMBTYPE=CMBTYPEF), command editing is bypassed, and HASPCOME issues a $FRECMB for this CMB and enters exit point, COMMEXIT, (for exit 5) where control is passed to the installation exit routine (if it exists and is enabled). The installation exit routine can preprocess the command before HASPCOMM does. When the installation exit routine returns, normal HASPCOME processing continues; an OK message is issued (via $CRET) for the command or an alternate message is issued (via $CRET) then HASPCOME returns, cancelling any existing ESTAE prior to returning. If command editing is not bypassed, the CMB buffer scan continues. The command is moved into that area from the CMB and edited as it is moved: single quotation marks replace double quotation marks, blanks and comments are eliminated, and all alphabetic characters are changed to uppercase. (Blank elimination and alphabetic character alteration are applied only to character strings not enclosed within quotation marks.)

As each comma outside paired quotation marks is encountered, an entry of the next available character position is made in the COMPNTER area of the PCE. (The first entry is the address of the character after the verb; the second is the address of the second operand, and so forth.) When the COMPNTER area is full, recording is discontinued. If a semicolon is encountered outside paired quotation marks, it is assumed to be a delimiter between multiple commands. If it is the first semicolon, the remaining portion of the CMB is moved to the beginning of the buffer, and the length field in the CMB is adjusted to reflect the new length of the content of the CMB.

The scan of the CMB is resumed. If another semicolon is found, a check is made to determine if it precedes the specification of a global L=cca operand. If not, that portion of the CMB that was moved to the PCE area COMMAND following the first command is cleared from the COMMAND area and scanning of the CMB is resumed. If a global L=cca operand is found, it is moved to the COMMAND area to become the operand for the current command (also contained in COMMAND).

If multiple commands are present, the CMB is requeued for subsequent scanning. If there are no multiple commands, the CMB is released, and the count in $COMMCT is increased by 1 (posting for CMB if necessary).

If the response to the command now contained in the COMMAND area is to be sent to an operating system console selected according to console identification, the text is examined for the presence of one of the following redirected response specifications:

Specification	**Meaning**
$command, L=cca	Individual
$command; L=cca	Global
$command, L=cca;L=cca	Both (individual overrides global)
$command, L=a;L=cca	Both (individual area overrides global area)

Although a semicolon is shown, a comma is inserted as the global separator by the edit routine. The specification, if entered correctly, is converted; the console identification in the COMUCM field is replaced by the cc specification, and the content of the COMUCMA field is replaced by the area specification (a). Appropriate flags are then set to indicate what fields were set by the L=cca specification. If a specification is found to be invalid, it is passed to the command sub-processors unless an invalid global specification is accompanied by an acceptable individual specification. The last operand pointer field plus 4 in the PCE is set to the address of the second character beyond the last solid character (null operand). Operand control registers are set as follows:

Register	Content
R3	Address of the first operand pointer in the COMPNTER field
R6	4
R7	Address of the last operand pointer in the COMPNTER field

Selecting the Command Sub-processor

While performing the command edit and breakout function, the HASPCOME routine examines the command to determine if it was an NJE send command; if so, the appropriate flag is set at that time. If the command has been so flagged (COMML≠0), and there is an available path to the desired command destination, a console message buffer (CMB) is obtained, and all data following the semicolon is copied into the message portion of the CMB. The CMB is updated with the destination information supplied by the NJE send command. Finally, the HASPWQUE routine in HASPCON is called to queue the CMB for transmission by the remote console processor. The edit routine then returns to the main processor with the OK message in the COMMAND area for display.

If it is not an NJE send command, two command selection tables are used to determine the sub-processor to be entered, as follows:

COMFASTR: This table consists of an entry for each letter of the alphabet that is valid as the first letter of a command. Each entry consists of the letter, followed by the address of the second-level table that is to be used in identifying the rest of the command. The command processor compares the first character of a command with the first character of each entry and, unless the command character is invalid and does not appear in this table, loads the address of the corresponding second-level table for further processing.

COMTAB: This is the label identifying the beginning of the second-level tables. A second-level entry contains (among other fields) a variable-length command verb with a maximum length of 7, and the address of the group processor needed to process the command. A complete second-level table contains one entry for each command verb beginning with the same first character. The command processor selects a group processor address from this table when the comparison for acceptable verb is successful.

If the end of the entries for that verb or the end of the table is encountered, the command is considered invalid, and the edit routine returns to the main processor with the "INVALID COMMAND" message in the COMMAND area so that the command will be displayed.

Validating the Source and Entering the Group Processor

HASPCOMM issues the $SEAS macro to validate the source of the command using RACF (or other installed security product). If the command source is determined to be invalid, the following COMTAB restrictions are bypassed. If no security product is installed, or if the $SEAS call returns a "no decision" return code, then the following (default) COMTAB restriction checking takes place.

Each entry of the second-level selection table beginning at COMTAB may have restriction indicators as follows:

- COMR=1: Reject remote sources and consoles which are restricted from entering network commands.
- COMS=1: Reject consoles that are restricted from entering system commands.
- COMD=1: Reject consoles that are restricted from entering device commands.
- COMJ=1: Reject consoles that are restricted from entering job commands.

The selection indicators correspond to the restriction indicators that appear in the COMAUTH field. The COMAUTH indicator is previously set from the CMBFLAG field of the JES2 console message buffer, which in turn is set by other JES2 processors as follows:

- CMBFLAG, when set by the remote console processor or remote reader processors, contains the remote indicator. This indicator corresponds to the COMR bit in the selection table.
- CMBFLAG, when set by the MVS console interface is the MVS authority indicator inverted with the exclusive OR immediate (XI) instruction.

The restriction indicators are used as the second operand of a test under mask (TM) instruction. If any restriction indicator in the COMAUTH field corresponds to any restriction indicator in the selection table entry, the command is rejected as invalid. If the operand L=cc has not been accepted and the source is an operating system console, the selection table entry is examined for the presence of an automatic redirection index. If the offset exists, the COMUCM field is adjusted with a new cc value, unless an L=cc specification has been previously accepted from the command text. The area ID portion of the COMCLASS field is also set with a new "a" value unless an L=cc or L=a specification has been previously accepted from the command text. If the resulting area specifies an out-of-line console area, the operating system console verification routine IEE7603D is called to verify the console and area. If IEE7603D does not accept the specifications, the area is set to binary zeros and the command is rejected; otherwise, register 1 is set with the value in the selection table entry COMTOFF field, and control is passed to the group processor indicated by the selection table entry element.

HA$PCOMM Command Preprocessing JES2 Exit Point

This exit point, COMMEXIT, allows the user (through user exit routines) to alter, delete, or reroute the response to a command.

The register values upon entry and exit are:

Register	Entry Value	Exit Value
R0-R4	N/A	Unchanged
R5	Address of current operand	Unchanged
R6	Increment value of 4	Unchanged
R7	Address of last operand	Unchanged
R8-R10	N/A	Unchanged
R11	HCT address	Unchanged
R12	N/A	Unchanged
R13	PCE address	Unchanged
R14	Return address	Unchanged
R15	Entry address	Return code

The valid return codes for this exit are:

0 - Continue normal processing.
4 - Continue normal processing.
8 - Take normal $CRET return (deletes command).
12 - Issue $CRET OK message.
16 - Issue $CRET message (message text must be in the COMMAND area and the message length must be in R0).

Note: For return codes of 0 and 4, a RACROUTE call is issued to perform security checking; for return codes 8, 12, and 16, no call is made.

COMMRCVR Command Processor Recovery Routine

COMMRCVR is called by the $RETRY routine to perform error recovery for the HASP command processor. $RETRY invokes COMMRCVR based on the setting of the RECADDR= parameter in the $ESTAE macro that was issued prior to a system abend condition or catastrophic error; COMMRCVR attempts recovery only from program check errors.

Upon entry, COMMRCVR issues a $SAVE macro to save the caller's registers. Then it sets addressability to the ERA, PRE, and SDWA using registers R5, R3, and R4 respectively. Next, it checks the SDWAERRA field in the SDWA. If the flag SDWAPCHK is on, then the abend was caused by a program check. If it is off, COMMRCVR branches to issue a $SETRP macro with the PERCOLATE option, and returns to $RETRY.

To recover from an abend caused by a program check, COMMRCVR discards the command that was being processed when the program check occurred. This action is based upon the probability that the command was the cause of the program check. This is done by setting the resumption address via the $SETRP macro using the RESUME=CONEXT parameter, and then returning control to the $RETRY routine via $RETURN. The operator is informed of the recovery action prior to returning ($HASP691).

Before the command is discarded, COMMRCVR checks for a multiple-line WTO (MLWTO) in progress by testing two fields in the PCE, COMFLAG and COMUCMA. If COMFLAGU is on in the COMFLAG field and COMUCMA is not equal to zero, then a MLWTO is in progress. To terminate the MLWTO, COMMRCVR issues a null line

WTO and zeros the COMUCMA field to prevent the message being issued by COMMRCVR from being considered part of the MLWTO. Then COMMRCVR moves the $HASP691 message into the command area of the PCE along with the partial command in the PRE (PRETRACK) and issues the message to the operator. COMMRCVR then uses the $SETRP macro to set the resumption address and the $RETURN macro to restore the caller's registers and return to the $RETRY routine. Note that the $ESTAE is in effect only while the command is being processed by its group processor.

If COMMRCVR is entered for any reason other than a program check, there is no recovery; the PERCOLATE option of the $SETRP macro is set and control is returned to $RETRY. The command terminated message ($HASP691) is sent to the master console as well as to the console issuing the command that abnormally terminated. If the master console issued the command, that console receives the message twice.

The following are the register settings upon entry to COMMRCVR:

Register	Entry Value
R0	Same as at time of abend
R1	Pointer to ERA
R2-R10	Same as at time of abend
R11	HCT address
R12	Same as at time of abend
R13	PCE address
R14	Return address
R15	Entry address

Command Group Processor Control Sections

On entry, the checkpoint data (including the job queue and job output table) has been stabilized (that is, no checkpoint read is in progress). The $CWTO routine, if forced to wait ($WAIT) for a console message buffer (CMB), reestablishes stability. It is the sub-processor's responsibility, in a multi-access spool environment, to reestablish stability following any other $WAIT.

If applicable, the entry routine of each command group processor control section uses the offset value in register 1 (set by the edit routine) to determine the entry point for the designated sub-processor. Normally, the sub-processor is entered directly by a branch on register 1. However, some control routines preprocess the operands of the command prior to entering the sub-processor. Each sub-processor performs the desired functions and returns to the main command processor for the next command.

Command Processor Organization

The JES2 command processor is created by a single assembly with two control sections (CSECTs). The main CSECT, HA$PCOMM, contains the main entry point, console message buffer (CMB) queueing, and R12 service routines. The other CSECT, HASPCOMA, contains the edit processor (symbol HASPCOME), the command processor recovery routine (symbol COMMRCVR), and all of the command sub-processors. HASPCOMA is not addressable; however, the edit and each command sub-processor use the R8 service routine to provide addressability for their respective functions. The edit routine loads R8 with the address of the command sub-processors as well as initializes register 1 with the offset entry to the verb processor within a command sub-processor group. The logical organization of the command sub-processors follows:

1. Scan command request sub-processor (HASPCSCN)
2. Job queue commands (HASPCJB1, HASPCJ1A)
3. Job list commands (HASPCJB2)
4. Miscellaneous job commands (HASPCJB3, HASPCJ3A, HAPSCJB4)
5. Device list commands (HASPCDV1)
6. $T device commands (HASPCDV2)
7. Spool data set commands (HASPCDV4)
8. PCE commands (HASPCPCE)
9. System commands (HASPCSY1)
10. Console commands (HASPCSY3)
11. Automatic operator commands (HASPCAOC)
12. Miscellaneous display commands (HASPCMS1)
13. Network job entry commands (HASPCNT1)
14. Global networking commands (HASPCSSI, HASPCFCP)
15. Exit commands (HASPCXIT)
16. Remote job entry commands (HASPCRM1)

Command Processor Work Area

The processor control element (PCE) work area of the JES2 command processor is the primary work area for the processor. Its fields are described in the following paragraphs.

Standard fields are set by the command edit routine HASPCOME and are used to locate the beginning of each of the specified operands in the command currently being processed. Operand pointers (1 through n) are left-adjusted in COMPNTER. A dummy operand pointer locating the second character after the last operand is placed in the first unused COMPNTER (or COMNULOP) field. Command sub-processors use these areas for additional work space after the operand pointers are no longer needed. Examples of other uses are:

- Job queue commands $D N and $D Q place queue scanning control elements in the COMPNTER area.
- Job list commands place the job range number (j-jj) in the corresponding operand pointer element area.
- Exit commands $TEXIT and $DEXIT place the exit range number (nnn-mmm) in the corresponding operand pointer element area.

COMFLAG to COMOUT: These fields contain a list form of the $WTO macro. The $WTO is referenced by the single-execution form of the $WTO (located within the HA$PCOMM CSECT of the command processor), which is used for operator messages generated by routines within the processor. The control fields of the console message buffer (CMB) for each command are used to construct the list form of the $WTO and provide correct route codes for replies.

COMSQD: This field contains the address of the SQD to be used for reroute processing that was obtained via the $GETWORK macro during HASPCOMM initialization.

COMINCON: This field is used to record the input console identifier.

COMAUTH: This field contains a copy of the console authority restriction flags contained in the input console message buffer CMBFLAG field. The edit routine as well as sub-processors refer to this field to determine acceptability of commands and operands entered through the input console.

COMACEID: This field is used to identify to the operator the automatic command element (ACE) when an automatic command response location is rejected by the operating system console verification routine IEE7603D.

COMJROUT: This field is used to identify the input source of the command and the destination of the command response. In the case of a networking command entered locally, it identifies the destination of the command itself.

COMEWORK: This field is used as a work area by function routines as follows:

Routine	**Contents Upon Exit from Routine**
COFCVE	Last character is blank
COFDCTL	First 4 characters of requested device name
COFJDCT	Address of job queue element for requested job
COFJMSG	Same as COFCVE
COFRTC	Same as register 1

COMDWORK: This field is used as a doubleword work area for convert to binary (CVB) and convert to decimal (CVD) instructions. It is also used to save the contents of registers. This field, aligned on a doubleword boundary, is used by function routines identified by the macros as follows:

Routine	**Contents Upon Exit from Routine**
COFCVE	5-character number in EBCDIC with leading blanks
COFDCTL	Last 4-characters of requested device name
COFJMSG	Same as $CFCVE
COFRTRC	Numerical conversion

COMWREGS: This field is 2 doublewords and is used as a register save area by several of the command sub-processors ($L, $C, and $T).

COMFWORK: This field is a three-byte area used for addressing by the command processor as well as by the command sub-processors $TEXIT and $DEXIT.

COMLCCA: This field is a halfword save area for the global L=cca operand used in the JES2 command edit routine.

COMCONNO: This field is a halfword save area that contains the number of MVS consoles; it is set in the HASPCOMM PCE by HASPINIT.

COMEXTEN: This field contains a pointer to a dynamically obtained work area that contains routing and area specifications for local consoles. This information is used in order to redirect command responses as specified by the $T M command.

COMMID: This field is a 2-byte area used for the JES2 message identifying number.

COMMAND: This field is a 410-byte area containing the compressed form of the operator command. The command is overlaid by responses to the command built by the individual command sub-processors. Some command sub-processors use the area as a scratch area, and in some cases, sub-processors use the right end of storage to hold critical information while message replies are generated in the left end of the area.

COMVERB: This field is a symbol used to point to the second character of the compressed command (that is, COMMAND+1). This is the verb of the command.

COMOPRND: This field is a symbol used to point to the third character of the compressed command (that is, COMMAND+2). This is the first character of the first possible operand.

COMJNAME: This field is an 8-byte area used primarily to hold the job name for the job name commands. It is also used as a work area by some of the command sub-processors.

COMPNTER: This field is a 20-word area that contains the address of the operands of a command. The last used word may be used instead of COMNULOP as the null operand pointer.

COMNULOP: This field is a fullword area that contains a pointer to a null location after the last operand of command when insufficient space is available in COMPNTER. It is used as a stop value for the command sub-processors when scanning operands.

COMPINDX: 20-byte field used for COMPNTER and CDUTABLE index bytes.

COMREGSV: This field is a 60-word area used to save registers and hold an argument list for HASPLIST ($L command) as well as scratch space for other sub-processors.

COMRWORK: This is a 10-byte field used as an input area for the $DEST interface to the USERDEST destination verification/conversion routine.

COMJQHDS: This field is used as a save area for job queue offsets.

COMFCMDA: This is a 36-byte area (with its origin at COMREGSV) which is used by HASPCSSI as an area in which to build a global formatted command.

COSIWORK: This is a 54-byte work area (with its origin at COMPNTER) which is used by HASPCFCP and contains a 36-byte subarea (COSICMDA) used to copy the global command from the console message buffer (CMB).

Command Processor Coding Conventions

The symbols within the command processor conform to the following conventions:

- All main processor, edit routine, and processor control element (PCE) work area symbols start with the characters CO.
- All function macro-generated symbols start with COF.
- All command sub-processors have entry point symbols of the following form:

Form	Example	Command	Comments
Cvvvvvvv	CDN	$D N	v = the verb of the command (maximum length of 7)
Cvxx	CD7D	$D 'jobname'	Apostrophe is hexadecimal 7D

- All symbols created for the support of the command start with characters that identify the entry point (CDNxxxx identifies a location originally written for the $D N command). Commands with no unique operand character symbol have the character X as the third character. (CBXxxxx identifies a location originally written for the $B device command.) These conventions may be altered in cases where the command identification characters are redefined after original development.
- The main processor CSECT is HA$PCOMM. The secondary CSECT, HASPCOMA, contains the command edit routine, command recovery routine, and the command sub-processors. Each group of sub-processors is identified by the symbol field of the $COMGRUP macro specified, starting with the characters HASPC.

Command Processor Register Conventions

The command edit routine passes control to the appropriate command sub-processor by a branch instruction. When the command group entry routine receives control, the registers contain the following:

Register	Contents
R0	Unpredictable
R1	Entry of individual command sub-processor
R2	Unpredictable
R3	Unpredictable
R4	Unpredictable
R5	First operand pointer (0 if no operand)
R6	4
R7	Last operand pointer
R8	Base for command sub-processor groups
R9	Never used (unpredictable)
R10	Unpredictable
R11	HASP communications table (HCT) address
R12	Beginning of main command processor (HA$PCOMM)
R13	Processor control element (PCE) address
R14	Unpredictable
R15	Unpredictable

If more than one command sub-processor appears within the group, register 1 is set by the edit routine so that an unconditional branch on register 1 enters the command sub-processor.

Command Authority Checking

JES2 command authority checking exists on two levels:

1. Command/console level (default).
2. Security product (RACF) level (supersedes default).

The $COMTAB macro (described in the "Organizational Macros" section) defines command elements. Keywords on the $COMTAB macro define the authority required to issue a command.

Command/Console Level

JES2 command authority is based on the console from which it is issued. Additionally, each JES2 command has an associated authority. A command is allowed to be issued if the authority level of the console is higher or matches the JES2 authority associated with the command. Figure 3-5 shows the correlation between JES2 command authority and console authorities:

Figure 3-5. JES2 Command/Console Authorization

Command Level	**Console Level**
System	Master, Console
Job, Device	SYS, I/O
Display	Info

This command/console authority checking is performed if no security product is active or if the security product has insufficient information to perform authority checking.

Security Product (RACF) Level

During the command edit routine's processing, a SAF call is issued to determine if the command is permitted. The following information is passed to the security product to make the determination:

- A security token representing the issuer of the command.
- The resource name of the command.
- The access level required to issue the command.

A security token is created to represent each command source. Consoles and card readers receive a token associated with the operator starting the device; internal readers receive a token associated with the TSO user or job; the JES2 token represents the INIT deck and card readers started from the deck.

Each command has an associated resource name of the format **subsystem.verb.qualifier** where:

subsystem
is the subsystem name where the command is being processed.

verb
is the JES2 command verb that describes the command action (such as Start or Display).

qualifier
is the general name for the receiver of the action (such as devices, jobs, or SYSOUT).

The JES2 command authorities are associated with security access levels, as shown in Figure 3-6 on page 3-163.

Figure 3-6. JES2 Security Authorization

Command Level	Security Access Level
System	Control
Job, Device	Update
Display	Read

The return codes from the SAF call indicate that the issuer of the command is permitted, not permitted, or that the default command/console checking is to be performed instead.

Command Processor Macro Instructions

To provide for flexibility in the development and possible modification of the command processor, a macro package is included in the assembler source deck. This supplements the JES2 command processor source listings obtained from the JES2 generation and assembler process and aids the user in understanding the generated code used in JES2.

Each JES2 command processor macro may depend on the definitions contained in the command processor source code as well as other members of the JES2 source library. These macros are categorized as follows:

- Organizational: Macros that provide basic definitions and are closely associated with the organization of the processor.
- R12-Based Services: Macros that call upon the main command processor to perform a service (display a reply).
- General Service Functions: Macros that perform the function inline or link to a routine that performs the desired function.
- Data Conversion Service Routines: Routines used for conversion of data from one form to another.
- Miscellaneous Service Routines: Routines used for performing miscellaneous functions.

The following conventions are used in specifying parameter requirements:

Convention	Requirement
parameter=**	The keyword parameter is required.
parameter=text	If this parameter is not specified, a default value indicated by "text" is used.
parameter	The parameter is an optional or a required positional parameter.

Command Processor Macro Summary

Operation Code	Definition
Organizational:	
$COMGRUP	Define group of command sub-processors
$COMTAB	Define command table element
R12-based Services:	
$CRET	Return to main command processor
JES2 macro	(macro not in HASPCOMM) - code in HASPCOMM
General Service Functions:	
$CFCVB	Convert to binary
JES2 macro	(macro not in HASPCOMM) - code in HASPCOMM
JES2 macro	code in HASPSERV
$CFDCTL	Device control table locate
$CFINVC	Reply invalid command
$CFINVO	Reply invalid operand
$CFJDCT	Find job's device control table (DCT) identifier
$CFJMSG	Display job information message (conditional)
$CFJSCAN	Assist job queue scanning
$CFSEL	Select routine based on character
$CFVQE	Verify console control over job

Organizational Macros

The following is a description of macros that provide basic definitions and are closely associated with the organization of the processor.

$COMGRUP: Define Group of Command Sub-processors Macro

The $COMGRUP macro provides an entry point for the group of command sub-processors. It also allows definition of a common processing routine for all commands of that group.

n positionals: Each positional specifies the command identification characters for the corresponding command sub-processor located within the group. For example:

Specification	Command	Sub-processor Entry Point Name
AA	$A A	CAA
DA	$D A	CDA
B	$B device	CB
C	$C device	CC
P40	$P	CP40
S40	$S	CS40
D7D	$D 'jobname'	CD7D

DELAY=NO: The verb sub-processor is entered by an unconditional branch on register 1. If YES is specified, register 1 contains the entry address for the selected verb processor, and control is given to the statement following the macro.

$COMTAB: Define Command Table Element Macro

The $COMTAB macro defines an element in the command selection table, which is used by the command edit routine HASPCOME for identifying legal commands, eliminating unauthorized input sources, and entering the correct command group CSECT.

Verb: A verb is required to identify the command. No two $COMTAB macro statements may specify the same verb character string. All macro statements creating entries for command verbs with the same first character appear in consecutive statements with the statement that specifies a single identification character verb last. Within each $COMTAB macro, the command verbs must be listed in descending character length within alphabetical order. The alphabetical order requirement takes only the first character of the verb into account.

Group: A group is required to identify the exact characters used in the specification in the symbol field of the appropriate $COMGRUP macro statement.

LABEL: The LABEL= parameter is optional. The parameter specifies the name of the sub-processor that is to get control after the command group processor. If no name is specified, $COMTAB generates a sub-processor name according to command processor coding conventions.

REJECT: The REJECT parameter is the command source rejection mask. One or more of the following symbols may be specified:

Symbol	Meaning
COMR	Reject the command if entered from a remote work station or NJE system unauthorized for networking control.
COMS	Reject the command if entered from a console not authorized for system control.
COMD	Reject the command if entered from a console not authorized for device control.
COMJ	Reject the command if entered from a console not authorized for job control.

Rejection of either a remote work station or a console not authorized for system control appears as follows:

```
REJECT=COMR+COMS
```

The following keywords may also be specified on the $COMTAB macro:

CALLER: CALLER=0 indicates that HASPCOMM will process the command. CALLER not equal to 0 indicates that the command is to be processed by the scan facility.

COMENT: COMENT=*qualifier(s)* specifies the second and/or third qualifiers of the resource name.

DELAY: DELAY=NO indicates that authorization checking should be done by the COMEDIT routine. DELAY=YES indicates that authorization checking should be delayed until the sub-processor gains control.

REDIR: REDIR=0 indicates that responses to commands entered through the operating system console are not to be automatically redirected. If the entry console is an operating system console and values 1-15 are specified, the appropriate entry in the redirect response table for the apparent source console is used to redirect responses.

Selection Table Element

The following shows the organization of the selection table element.

Symbol	Offset	Length	Description
COMTFL	0	1	Redirection index and restriction flags
		0000 xxxx	No automatic redirection response
		nnnn	xxxxn=1-15 index in the redirect response table for the console of apparent entry
		xxxx 1...	COMR; reject command if from remote work station or if source is not authorized for network control
		xxxx.1..	COMJ; reject command unless source authorized for job control
		xxxx..1.	COMD; reject command unless source authorized for device control
		xxxx...1	COMS; reject command unless source authorized for system control
COMTGRP	1	4	Address of group processor
COMTOFF	5	2	Offset for sub-processor
COMTVBLN	7	1	Length of the command verb minus one
COMTVB	8	7	Command verb
COMTFLGS	15	1	Flags for future use

Note: Although the edit routine allows entry to the command sub-processor, each command sub-processor may reject the command due to restricted operands.

R12-based Services

The following is a description of macros used to call upon the main processor to perform a service.

$CRET: Return to Main Command Processor Macro

The $CRET macro causes a return to the main command processor. $CRET issue $WTO and $WAIT as directed by caller.

Registers Used:

R0 = length of message if response requested
R1 = message address - (COMMAND)
R15 = Return code
CORTNORM - no message.
CORTOK - OK
CORTMSG - general message.

MSG=: This operand can be set equal to:

- The address of the message to be moved to the processor control element (PCE) area COMMAND for display (L=operand if a non-register form is required)
- A literal surrounded by quotation marks of the message to be moved to the COMMAND area for display (L=operand not used)

- The characters OK to indicate that the response OK will be moved into the COMMAND area and displayed (L=operand not used)

L=: This value represents the length of the message that is to be moved or has already been moved.

INFO=NO: skips block comment describing $CRET
YES: displays block comment describing $CRET

MSGID=: Specifies the message identifier to be assigned to the message issued.

JOB=YES/NO: Specifies whether or not a job number and job name should be issued with the message.

$CWTO: Write to Operator Macro

The $CWTO macro causes a write to operator and then returns control to the code issuing the $CWTO.

Registers Used: R0,R1,R14,R15

R0 = length of the message
R1 = message address
R14 = return address
R15 = routine address:
CWTO - if TRUNC=NO
CWTOT - if TRUNC=YES

MSG=: This is identical to MSG as used in $CRET processing, except that MSG=OK should not be used (use MSG='OK').

L=: This is identical to L=as used in $CRET processing except that register format is also accepted.

TRUNC=NO: The multiple line WTO is not truncated. If it is set to YES, the multiple line WTO is truncated, and additional $CWTO or $CRET macro executions specifying messages result in issuance of an SVC 34 which uses the message text as the command to the operating system.

MSGID=: Specifies the message identifier to be assigned to the message issued.

JOB=YES/NO: Specifies whether or not a job number and job name should be issued with the message.

General Service Functions

The following macros perform the indicated function inline or link to a routine that performs the desired function.

$CFCVB: Convert to Binary Macros

The $CFCVB macro converts the numeric portion of a command operand to one or two numeric values. It allows job numbers up to 32,767.

Registers Used: R0,R1,R14,R15

R0 =	last number converted
R1 =	next-to-last number converted (last number if the only one or if the last is smaller than the previous)
R14 =	link register
R15 =	upon return, address of the character that stopped the conversion

TYPE = CALL: generates a calling sequence for COFCVB
RES: generates the source code for the branch and link entry

Note:

- TYPE = RES should only be coded once to generate the source of the macro.
- TYPE = CALL is the default and should be used to invoke the macro for actual use.

Pointer = (R1): This is the address of the COMPNTER field that addresses the operand containing one or more numerical values separated by a hyphen(-).

NUM = 2: a set of numbers is to be converted (i.e. J2-200)
≠2: only one number is to be converted

INFO: YES/NO displays/skips block comment describing $CFCVB

NOK = **: This is the address of the error exit routine if the operand does not contain a number or if the number is larger than 9999.

MAX = : 9999 (default) maximum number to be converted.

$CFCVE: Convert to EBCDIC Macro

The $CFCVE macro converts the number in register 0 to printable EBCDIC and sets the resulting digits in the first 5 characters of the PCE area COMDWORK.

Registers Used: R0, R14

Work Areas Used: COMEWORK and COMDWORK are used to convert numbers to EBCDIC values.

Value = (R0): This is the positive binary halfword value used to convert to EBCDIC. If the register form is not used, the value is contained in the addressed halfword.

$CFDCTD: Display Device Control Table Macro

The $CFDCTD macro displays the device name as known to JES2, the unit address, and the activity status of the device control table (DCT) requested. Optionally, an extended display may be requested that details the complete status of the DCT (format varies with the device). For example, an extended display of a printer DCT includes the name, unit address, and activity status plus such information as the current forms, carriage, train, job name and number (if active), separator option, and pausing option.

Registers Used: R0, R1, LINK, R15

Work Areas Used: COMMAND is used as an output area, and COMJNAME is used as an output area extension.

DCT=(R1): This is the address of the DCT to be displayed.

EXT=NO: This indicates whether extended display of the DCT is desired. When set to YES, extended status is displayed.

$CFDCTL: Locate Device Control Table Macro

The $CFDCTL macro information converts the abbreviated or long form of the device name to the device control table (DCT) address. If the device is remote, the DCT address is searched in the remote attribute table (RAT) chain. If the device is an NJE device, the DCT address is searched in the $LNEDCT chain. If the device is a local unit, the DCT address is searched in the $DCTPOOL chain. If the requested device is an internal reader, the address of the first internal reader DCT is always returned.

Registers Used: R0, R1, R14, R15

Work Areas Used: COMEWORK and COMDWORK are used to hold the long form of the device name and also serve as intermediate work areas. On return, register 1 contains one of the following:

- The address of the DCT
- A 0 indicating the requested device's DCT was not found
- A negative value indicating that the address is the two's complement of the RAT entry for the remote work station.

POINTER=R1: This is the address of the COMPNTER field which addresses the operand containing the device name.

TYPE=CALL: generates a calling sequence for COFDCTL **RES:** generates the source code for a branch and link entry

INFO=NO: Skips block comment describing $CFDCTL **YES:** displays block comment describing $CFDCTL

Note:

TYPE=RES: should only be coded once to generate the source of the macro.

TYPE=CALL: is the default and should be used to invoke the macro for actual use.

$CFINVC: Reply Invalid Command Macro

The $CFINVC macro returns to the main command processor and causes the display of the INVALID COMMAND message.

TYPE=CALL: generates a calling sequence for COFDCTL

RES: generates the source code for a branch and link entry

INFO=NO: skips block comment describing $CFDCTL

YES: displays block comment describing $CFDCTL

Note:

TYPE=RES: should only be coded once to generate the source of the macro.

TYPE=CALL: is the default and should be used to invoke the macro for actual use.

$CFINVO: Reply Invalid Operand Macro

The $CFINVO macro moves 9 characters, starting with the first character of the current operand, to the COMMAND area and returns to the main command processor, causing the display of the operand INVALID OPERAND message.

OPERAND=(R1): This is the address of the operand to display.

TYPE=CALL: generates a calling sequence for COFDCTL **RES:** generates the source code for a branch and link entry

INFO=NO: Skips block comment describing $CFDCTL **YES:** displays block comment describing $CFDCTL

Note:

TYPE=RES: should only be coded once to generate the source of the macro.

TYPE=CALL: is the default and should be used to invoke the macro for actual use.

$CFJDCT: Find Job's Device Control Table Identifier Macro

The $CFJDCT macro finds JES2-controlled device identifiers on which the requested job is active. This macro searches the active job output elements (JOEs) looking for a JOE belonging to the requested job. If the JOE is found, the JOE contains a halfword field which designates the unit record device on which the job is busy. If the requested job is on a reader, SYSOUT receiver queue, or a job transmit queue, the job's job queue element (JQE) contains the device descriptor field. These fields are moved to a work area and the address of the first field is returned in register 1. If the requested job is not active on a JES2-controlled device, control is returned to the instruction immediately following the macro call; otherwise, control is returned to the above location plus 4.

Registers Used: R1, R14, R15

Work Area Used: COMREGSV is used to save halfword device descriptor fields for a job active on a device.

JOBQE=(R1): JQE address

$CFJMSG: Display Job Information Message Macro

The $CFJMSG macro moves a display of the requested job into the processor control element (PCE) COMMAND area and displays it. Because a job may be active in more than one JES2 phase simultaneously, more than one message may appear for the requested job. Return after the macro call is to the instruction plus 4 if the job is displayed; otherwise, return is to the next sequential instruction following the macro call.

Work Area Used: COMDWORK is used as a scratch area, and COMMAND is used as a display area and flags area.

TYPE=CALL: Generates the calling sequence for COFJMSG.

TYPE=RES: Generates the source code of the macro for the branch and link entry.

JOBQE=(R1): This is the address of the JES2 job queue element for the desired job.

AFF=: This field is used to indicate the active system for which the display is requested. (COFAFF field) Default: X'7F' - all systems.

OPT=COFU: This flag is used to indicate which jobs are to be displayed and must be set to one of the following symbols:

Symbol	Meaning
COFN	Display batch jobs
COFS	Display system control tasks
COFT	Display time-sharing users
COFJ	Display all of the above (COFN+COFS+COFT)
COFX	Display all jobs in execution
COFD	Display all jobs on devices
COFA	Display all active jobs (COFJ+COFX+COFD)
COFI	Display pre-execution queued jobs
COFO	Display jobs queued for output processing
COFP	Display jobs queued for print/punch processing
COFQ	Display all queued jobs (COFJ+COFI+COFO+COFP)
COFU	Display jobs unconditionally (COFJ+COFA+COFQ)
SET	The OPT= parameter was set prior to this macro call

OPT2: (sets COFOPT2). Default: X'00' - no setting. Indicates if the display should include a list of the spools used by the jobs (COFSPLAG). The default indicates that the spools should not be included in the message.

VOL=$ALLFFS (sets COMSPMSK field). Default: Job display does not depend on which volume job resides on.

If COMSPMSK is other than $ALLFFS - only those jobs that reside on the volumes represented will be displayed. Volumes are represented by a bit mask corresponding to DASMASK.

$CFJSCAN: Job Queue Scan Macro

The $CFJSCAN macro assists in scanning the job queue. As each entry is located, the user's PROCESS routine is entered. The user examines the entry, performs the function desired on the entry, and returns to the symbol specified by the NEXT=operand. When the end of the queue is encountered, control is given to the instruction following the macro. An optional feature of the macro is to allow the PROCESS routine an IGNORE entry to the generated code to indicate that the current job entry is not acceptable to the PROCESS routine. If the IGNORE=option is specified, the corresponding EMPTY=option is required. Register 1 is the scan register and is assumed to be unaltered by the user's PROCESS routine.

Registers Used: R1, R12

R1 =	Scan registers
R12 =	Found/not found switch (in addition to processor base)

PROCESS=: This is the address of the caller's job queue element processing routine. Register form is prohibited.

EMPTY=: This is the name of the caller's exit routine to be entered when the job queue is found empty of jobs of the desired type. Register form is prohibited.

IGNORE=: This symbol is used to define the entry where the scan continues when the current job entry is not of the desired type.

NEXT=:** This symbol is used to define the entry where scan continues when the current job entry is of the desired type.

QUEUE=:** This symbol specifies the queue to be scanned. If not specified, all queues will be scanned.

$CFSEL: Select Routine Based on Character Macro

The $CFSEL macro generates a table of sub-parameter arguments and then passes control to the COFSEL service routine. The positional character sub-parameter can be up to 256 characters long.

Registers Used: R1, R14, R15

R15 = upon return, the length of the keyword sub-parameter

n Positionals of Form (character, address): Each positional character sub-parameter specifies an argument. The corresponding address is the address of the routine to be entered, if the input character matches the character argument. Register form is prohibited.

OPERAND=(R1): This is the address of the designated input character to be examined.

$CFVQE: Verify Console Control Over Job Macro

The $CFVQE macro tests the COMAUTH field of the processor control element (PCE) to determine if the input source is a remote work station or an NJE node without network authority. If the source is a remote work station, the NOT OK routine is entered unless either the print or punch route codes for the indicated job match the route code contained in the corresponding remote attribute table (RAT) element. If the source is another node, the NOT OK routine is entered unless either the print or punch route codes for the job match the COMJROUT field (if entered from a remote work station), or unless the node in either the print or punch route code or originating node of the job match the node specified in the COMTONOD field (if entered at an OS console). Otherwise, the OK routine is entered.

Registers Used: R0, R1, R14, R15

JOBQE=(R1): This is the address of the job queue element for the desired job.

OK=(,B): This is the address of the routine to be entered if the console has control over the job. The address may be a symbolic register specified as OK=(register,B).

NOK=(,B): This is the address of the routine to be entered if the console does not have control over the job. The address may be a symbolic specified as NOK=(register,B).

Note: Either OK= or NOK= parameters must be specified.

TYPE=CALL/RES: This operand indicates whether or not an EJECT is generated before the block comment. TYPE=RES does not generate an EJECT before the informational block comment prior to generating the macro: TYPE=CALL, which is the default, generates the $CDFJMSG macro call. If INFO=NO is specified, TYPE= has no meaning for the EJECT. TYPE=RES generates the source code for the branch and link entry.

INFO=NO: Skips block comment describing $CFVQE. **YES:** Displays block comment describing $CFVQE.

Data Conversion Service Routines

The following routines are used for conversion of data from one form to another.

COFCVB: Convert Value to Binary

COFCVB converts a numeric to a binary value.

COFCVE: Convert Value to EBCDIC

COFCVE converts a halfword binary number to EBCDIC.

COFEDTR: Convert Value to EBCDIC

COFEDTR converts a fullword value in R0 to EBCDIC and places the result in R2.

COFLIM: Converts Limits to Binary

COFLIM converts, to binary, a set of limits. The limits are specified in the form:

```
LIM=m-n|m-*
```

where m is the lower limit, n is the upper limit and * represents the JES2 default value of X'FFFFFFFF'. COFLIM checks the values for validity. If a validity error occurs, such as M>N, a message ($HASP650) is generated which indicates that the operand is invalid.

COFRTC: Convert Route Code to EBCDIC for Display Subroutine

COFRTC converts a given 4-byte internal remote code into EBCDIC form for display. The output text is up to 10 characters, starting in byte 2 of a 12-byte work area. Unfilled bytes of the area are set to blanks.

COFRTD: Convert to Default Route Range Routine

COFRTD examines the source console information in field COMJROUT of the processor control element (PCE). If the source is a remote work station, registers 0 and 1 are set to the route code assigned to that station; otherwise, register 0 is set to the system default routing (0-$MAXROUT), and register 1 is set to LOCAL routing.

COFRTR and COFRTRA: Convert Destination Ranges to Route Ranges

The COFRTR subroutine uses USERDEST to convert each destination into a binary route code. This routine scans the current operand of a command for either a single destination or a destination range separated by a dash (-). Entering the routine at the COFRTR entry point causes the destination range to be converted directly to a binary route code; entry at the COFRTRA entry point causes the first 2 operand bytes to be skipped over and the remainder to be converted to a binary route code. (The destination code is assumed to be of the form "R=...".) The routine sets the condition code on return to indicate the validity of the input.

Acceptable input to this routine is as follows:

NnnnnRnnnn (-Rmmmm), Nnnnn.Rnnnn (-Rmmmm), or Name.Rnnnn (-Rmmmm)	-	Range of remotes at a given node ("Name" is an 8-byte EBCDIC representation for a node from a DESTID or Nnnnn card)
Nnnnn (-Nmmmm)	-	Range of all remotes/units at given nodes
Unnnn (-Ummmm)	-	Range of units at the local node only
Rnnnn (-Rmmmm)	-	Range of remotes at the local node only
Name1 (-Name2)	-	Range of names that are symbolic representations of the above-mentioned ranges (from the DESTID or Nnnnn initialization parameters). (The names must be in ascending order based upon the internal JES2 representation of the name.)
LOCAL	-	All local destination

COFR$DSC: Parse Given Destination to a Binary Code

COFR$DSC parses the given destination and returns a binary route code.

Miscellaneous Service Routines

COFDCTL: Device Control Table Locate

COFDCTL locates the DCT of the device pointed to by register 1.

COFINVC: Reply Invalid Command

COFINVC generates a $HASP649 message.

COFINVO: Reply Invalid Operand

COFINVO generates a $HASP650 message.

COFJDCT: Find Active Devices

COFJDCT finds devices that are active for a specific job.

COFSEL: Select a Routine Based on Key Input Character

COFSEL matches the designated input characters against the list of arguments provided by the $CFSEL macro. When a match is found, control is passed to the routine designated by the corresponding address found with the matching character. If no match is found, processing continues with the next sequential instruction.

COFVQE: Verify Console Control

COFVQE tests for a restricted console.

COFJID: Job ID Message Information

COFJID inserts standard job information into a message area.

COFJMSG: Generate Job Information Output

COFJMSG generates job information output.

CFJOEDS: JOE Display Routine

CFJOEDS finishes formatting the $HASP688 message.

CFJOESET: OUTGRP Set Routine

CFJOESET gathers JOE information for the $C, $TO, $L OUTGRP command.

CFJOEMOD: Set JOE Modification Parameters

CFJOEMOD sets the JOE modification parameters for the $TO command.

COFM630: Issue $HASP630 Message

COFM630 uses $CWTO to issue the $HASP630 message.

COFM646: Issue $HASP646 Message

COFM646 uses $CFCVE to convert information to printable text and uses $CWTO to issue the $HASP646 message.

Command Processing

The following identifies the group processors for each set of command sub-processors (for example, job queue commands, job list commands, job name commands) and describes the commands processed by each command sub-processor.

HASPCSCN: Process DISPLAY and SET Requests

This command sub-processor processes display and set requests that will be processed by the $SCAN facility. This routine makes the $SCAN call and then displays any error messages returned by $SCAN.

HASPCJB1: Job Queue Commands Group Processor

The following is a description of the individual job queue commands processed by the HASPCJB1 group processor.

$A A

This command releases all jobs in the job queue that were previously held by a $H A command. The resident job queues are scanned for jobs that were held by the $H A command. If there are any, the jobs' system affinity is compared with the requested system affinity. The jobs are released if the requested system affinity is included in the jobs' system affinity. The jobs are released by turning off the QUEHOLDA bit and posting JES2 for a job.

$A Q

This command releases an execution queue. The CAT (class attribute table) is searched for the classes specified; the appropriate bit indicating that the classes were held is turned off. JES2 is posted for a job.

$D A

This command is used to display active jobs. The resident job queue elements (JQEs) are scanned for any jobs, and if a job is found, it is displayed via the $CFJMSG macro instruction. The job's status is displayed by $CFJMSG, depending on the position of the job in the queue and the operand used for the command.

This routine is also aware that a JQE marked busy on the hardcopy queue, with a JQE offset in a LCK element, represents a job being processed by a SYSOUT transmitter dumping held data sets. Appropriate messages are issued.

$D N

This command is used to display information on queued jobs. The operands are examined and internal flags and addresses are set for the specified options. The job queue is scanned, and when a job is found meeting the specified criteria, it is displayed via the $CFJMSG macro instruction.

$D Q

This command is used to display the count of queued jobs. The processing is combined with $D N. The number of jobs for a particular queue are counted and displayed, instead of displaying information on each particular job as is done with $D N.

$H A

This command is used to hold all jobs currently in the job queue. Jobs can only be released by a subsequent $A A command. The job queues are scanned and each job whose system affinity includes the specified operand is held by turning on the QUEHOLDA bit in the job queue element.

$H Q

This command is used to hold an execution queue. This processor shares the logic with $A Q, described previously, except that the class or classes are held instead of released.

HASPCJ1A: Additional Job Queue Commands Group Processor

The following is a description of job queue commands processed by the HASPCJ1A group processor.

$D F

This command is used to display the forms output queue. After the operands are inspected and converted to appropriate values, the job output elements (JOEs) are scanned. A characteristic JOE is found and saved, then the work JOEs are scanned by class for a JOE that meets the operand criteria. When all such work JOEs of each class are accounted for, a response is displayed showing DEST, forms, FCB, UCS, writer, flash, burst, class, and PRMODE.

The process is repeated for each remaining characteristic JOE.

$O Q, $O J, $O S, $O T

This command is used to release or cancel held data sets. The operands are extracted and saved for later use. The job queues are scanned, and when a job queue element (JQE) indicates that there are held data sets (JQEHLDCT is nonzero), a process-SYSOUT element is built and command processing issues a $GETWORK for a job disposition request (JDR) element. The command operands are mapped into the JDR and it is queued to the $JDRQUE. The JDR processor is then $POSTed for work.

$P Q

This command is used to cancel output data sets. The operands are extracted and saved for later use. The job output element (JOE) queue is examined for any JOEs of the type specified. If any are found, they are removed from the JOE queue using the $#REM macro facilities. The process is continued, always scanning from the top of the JOE queue, until all requested JOEs have been deleted.

HASPCJB2: Job List Commands Group Processor

This group processor is declared with the DELAY=YES keyword of the $COMGRUP macro instruction. This is done to allow the job numbers or range of job numbers to be extracted. A loop is then set up in which the job queue is scanned for the first desired job. When found, the particular verb processor is entered (see $COMGRUP macro). If no jobs are found in the specified ranges, a NO JOB(s) FOUND message is issued.

$A J, $A S, $A T

These commands are used to release user-submitted jobs (JOB), started-task control (STC), or time-sharing user (TSU), jobs previously held by the corresponding $H J, $H S, $H T, or $H A command. A test is made to see if the jobs are really held. If so, the bits QUEHOLDA and QUEHOLD1 in the job queue element (JQE) are turned off, and JES2 is posted ($POST) for jobs.

$C J, $C S, $C T

These commands are used to cancel JOB, STC, or TSU jobs.

If the request is to cancel a TSU or STC, a check is made to see if it is prior to execution, in execution, or on a reader. If so, the TSU or STC is displayed by branching to the $D S or $D T routine. If the request is to cancel a JOB, STC, or TSU job and the PURGE operand has been used, the spin and held data sets as well as the job output elements (JOEs) are cancelled. If the job is in the output or print/punch phase and the PURGE operand has been omitted, the JOB, STC, or TSU job is only displayed by branching to the $D J, $D S, or $D T routine.

If the request is to cancel a batch job that is in execution, the partition information tables (PITs) are searched for the requested job. If the job cannot be found, it is executing on another system and the command must be entered from that system.

If the job is found, and this system is an MVS/370 system, the command scheduling control block (CSCB) in the job's subsystem job block (SJB) is posted with either an X'122' (if the job was cancelled with a dump) or X'222' completion code.

When this MVS is MVS/XA, control passes to label CCJCACT, where an MVS cancel is built and passed to MVS with a type=34 $WTO. This allows reviews of the command by other subsystems, such as IMS, to clean up before the cancel is executed. If the JOB, STC, or TSU job is on a device and the PURGE operand has been supplied, the QUEOPCAN and QUEPURGE bits in QUEFLAGS (in the job queue element) are set so that the current data set on the device will be cancelled by the device processor.

$D J, $D S, $D T

These commands are used to display jobs, started tasks, or TSO sessions. The item is displayed unconditionally via the $CFJMSG services. This routine is also aware that a JQE marked busy on the hardcopy queue, with a JQE offset in a LCK element, represents a job being processed by a SYSOUT transmitter dumping held data sets. Appropriate messages are issued.

$H J, $H S, $H T

These commands are used to hold a JOB, STC, or TSU job. The QUEHOLD1 bit is turned on for the job and displayed via a branch to the $D J, $D S, or $D T routine.

$P J, $P S, $P T

These commands are used to stop a JOB, STC, or TSU job when the current activity has completed, or cancel JOB, STC, or TSU non-busy output data sets (JOEs).

This routine shares the same logic with the $C J, $C S, and $C T routines. The main difference is that the $P J, $P S, and $P T logic allows the job to finish its current activity before purging it.

- If the job is a TSU or STC and is in or awaiting execution, the STC or TSU is displayed by branching to the respective $D S or $D T routine.
- If the batch job, STC, or TSU is in execution, it is flagged for purge and displayed as explained above.
- If the JOB, STC, or TSU is currently active on an output device, that activity is allowed to complete, but all other JOEs are cancelled.
- If the Q = operand is supplied, the job deletion functions are omitted and the output data sets (JOEs) for that job are cancelled using the $#REM macro facilities.

$E J

This command is used to restart a job currently in execution. This command is not supported for STC or TSU jobs. The job's job queue element (JQE) is tested to see if the job is in execution. If it is not in execution, a diagnostic message is issued stating that the job is not restartable. If the job is eligible for restart, the partition information tables (PITs) are scanned for active jobs. If a PIT has an active job, the subsystem job block (SJB) using that PIT is tested to see if it belongs to the requested restartable job. If so, the appropriate flag is set in the SJB to indicate restart, and the job is displayed via a branch entry into the $D J routine.

$TOJ, $TOS, $TOT

This command sets or changes the characteristics of the specified output group identified by its output group name and, optionally, its JOE identifier.

Two paths exist in $TO processing, depending upon whether the characteristics JOE is to be modified or not. If not, the JOE is modified directly in the JOT. Otherwise, the characteristics and work JOEs are copied to a work area and modified there. This is done because other work JOEs may be using the characteristics JOE.

In both paths, if a new DEST is specified, a $REROUTE call is made to validate authority to acces the new destination. $REROUTE may change fields in the PDDBs, IOTs, JCT, and JQE if the output is rerouted to the local node and local SAF information (a security token) is needed.

If the $REROUTE call fails, the DEST (and any other fields) are left unchanged. If the $REROUTE is successful (or if not performed), the output group is either $#MODed or $#REMed/$#ADDed, depending on whether the changes were made directly in the JOT or to copies of the JOEs in the work area.

$L J, $L S, $L T

These commands list the output for a job (JOB), started-task control (STC), or a time-sharing user (TSU) job. If the HOLD operand is not used, the job output element (JOE) work queue is examined for JOEs belonging to the specified job. If any non-held JOEs belonging to the job are found, they are listed or counted, depending upon the operands specified. If the request is to list the held data sets belonging to the job (HOLD operand was used), an argument list is constructed, and HASPLIST is entered via $LINK services. HASPLIST returns with a count of held data sets, which are then listed.

HASPCJB3: Job Name Commands Group Processor

The purpose of this group processor is to accept job commands using the job name format instead of the job number format. If the specified job is located, control is transferred to the corresponding job list sub-processor.

$A, $C, $D, $E, $H, $L, $O, $P, $T, $TO

These commands accept job commands using the job name format and transfer control to the corresponding job list sub-processor. All have a common entry point in the group processor.

The job name is extracted from the command area. The resident job queue elements (JQEs) are searched for a job with that name. If one is found, the search is continued, looking for multiple jobs with the same name. If the request is to display the requested job name, it is done via the $CFJMSG macro instruction. If the request is for any of the other services in the commands and there are multiple jobs of the same name, a diagnostic message is formatted and sent to the operator. If there is only one job of the requested name, control is transferred to the particular sub-processor for the command.

HASPCJ3A: Set Base Numbers Group Processor

This group processor is used to set submitted job (JOB), started-task control (STC), or time-sharing user (TSU) job base numbers or to change the JOB, STC, or TSU class, priority, or system affinity.

$T J, $T S, $T T

This sub-processor changes the JOB, STC, or TSU class, priority, or system affinity. If the command contains operands other than the JOB, STC, or TSU number, they are extracted and validated. The job queues are then searched, and when a job is found that is in the range of the command, it is changed in accordance with the operands. The following changes are valid:

- Class Change: Only to jobs awaiting execution.
- Priority Change: Only if the job, started task, or TSO session is not currently active in some JES2 phase.
- System Affinity Change: Only for STCs and TSUs that are not awaiting conversion or execution. All jobs can have their system affinity changed at any time.

Note: Changing the system affinity of any job that is queued for print/punch processing is meaningless because all printers and punches controlled by JES2 are able to select work independent of system affinity.

$T NUM

This command sets the base job number to the new base number indicated in the command. If the number indicated in the command is being used, the base job number is set to the next highest available number.

HASPCJB4: Release/Cancel Held Data Sets Group Processor

This group processor releases or cancels held data sets belonging to the specified JOB, STC, or TSU job.

$O J, $O S, $O T

The one command verb processor in this sub-processor releases or cancels the buffer queue elements (BQEs) of the specified job, STC, or TSU. The operands are extracted and saved for later references. A check is made to see if the designated job has any held data sets (JQEHLDCT is nonzero). If so, a process-SYSOUT element is constructed for the job and queued to $OQUEUE. Subsequently, $PSOPCE, the process-SYSOUT processor control element (PCE), is posted for work.

HASPCDV1: Device List Commands Group Processor

This group processor is generated with the DELAY=YES keyword of the $COMGRUP macro instruction. In the routine executed before each individual verb processor is entered, the device control table (DCT) is located via the $CFDCTL macro instruction. If it is not found, a diagnostic is sent to the operator. If it is found, a test is made to see if it is a remote console DCT. If so, the group processor HASPCSY3 is entered for changes to a remote console. If the operand is a valid DCT, a test is made to see that the entering console is the owner of the requested device or is allowed to modify the requested device. If so, the verb processor is entered; otherwise, an error diagnostic is sent to the requesting console.

$B

This command backspaces print or punch devices by the specified number of pages (cards). The requested device is tested for the correct class (that is, print or punch). If it is the correct class, the parameters are scanned. The C parameter and a number of pages may be specified separately or in conjunction. If the D parameter is specified, no other parameters may be specified. The backspace count is converted to binary and saved in the requested device's processor control element (PCE). The device's PCE is then posted ($POST) for I/O.

$C

This command is used to cancel the current activity on a device. The class of the device is tested for a valid specification (that is, a reader, printer, or punch). If the class is valid, the appropriate bit is turned on in the device's DCT, and the device's PCE is posted ($POST) for I/O.

$E

This command is used to restart the current function on the device. The class of the device is tested for validity. Only printers, punches, and RJE lines may be restarted. If this test is passed, the restart bit is turned on in the device's DCT, and the device's PCE is posted ($POST) for I/O.

$F

This command is used to forward-space a print or punch device. This command verb processor shares logic with $B (backspace); the only difference is that the value saved in the device's PCE indicates a forward-spacing operation, as opposed to a backspace.

$I

This command is used to interrupt the activity of a device. A test is made to see if the device is a printer or a punch. If it is neither, an error message is sent. Otherwise, the appropriate flags are turned on in the DCT to indicate an interrupt, and the device's PCE is posted ($POST) for I/O.

$N

This command is used to repeat the current function (activity) on the requested device. A test made to see if the requested device is a printer or punch. If it is not, an error message is issued; otherwise, the repeat flag is turned on in the device's DCT, and the device's PCE is posted ($POST) for I/O.

$P

This command is used to drain (deallocate) the requested device. The device is tested to see that it is not an internal reader. If it is, an error message is issued. Otherwise, the appropriate drain flag is turned on in the device's DCT. If the device is not active, the unit control block (UCB) for the device is deallocated via the $FREUNIT macro instruction. If the device is active, the deallocation will be done at that device's processor termination. If the device is under control of a functional subsystem, HASPFSSP is eventually used to complete the deallocation; otherwise HASPPRPU is posted to complete the deallocation.

$S

This command is used to start the requested device. If the device requested is an internal reader, an error message is issued to the requesting console. If the device is a local (that is, not a line or a remote) device, it is allocated to a unit control block (UCB) via the $ALLOC macro instruction. If the device is remote or line, the appropriate bits are turned off in the device's DCT, and HASPRTAM is posted ($POST) for work. If the device is a dump or load device, the appropriate bits are turned off in the device's DCT, and HASPRTAM is posted ($POST) for work.

If the device is an offload device and the device is available to start, the spool offload I/O manager is $POSTed to start the device, and its associated transmitter or receiver. For all devices, other bits are turned off in the device's DCT, SYSOUT transmitters are $#POSTed, and JES2 is posted ($POST) for unit, job, and the job output table (JOT).

$Z

This command is used to halt a device. A check is made of the device's class. If the request is valid, the DCTSTOP bit in the DCT is turned on.

HASPCDV2: $T Device Commands Group Processor

$T

This command is used to set a characteristic for a device. A check is made for any operands following the device name. If none are found, an error message is issued to the operator. If this test is passed, the device class is determined, and an appropriate routine is entered for each device class. Starting with the first operand, each operand is examined for validity, and the device control table (DCT) is updated to reflect the change. If the DCT indicates that the device is being controlled by a functional subsystem, the DCT is updated so that HASPFSSP issues FSI orders to make the appropriate changes indicated by the operand. This process is continued for each successive operand in order until all operand settings have been honored or an error is encountered.

HASPCDV4: Spool Volume Command Group Processor

This group processor handles the spool volume commands $D SPOOL, $P SPOOL, $S SPOOL, and $Z SPOOL. This group processor is generated using the DELAY=YES keyword of the $COMGRUP macro instruction.

$D SPOOL

This command displays the status of the spool data sets. Operands are checked for correctness using $CFSEL.

$P SPOOL

This command causes the designated spool data set to be drained. Operands are checked for correctness using $CFSEL.

A $QSUSE is issued to acquire access to the job queues, the operands are selected for processing, and a $SVOLOC is issued for the designated volser of the spool data set to obtain its DAS.

$P SPOOL processing according to spool volume state -- CANCEL *not specified*:

- Active - The spool data set will be deallocated and drained. The DASFLAG2 flag is set to indicate $PSPL and HASPOOL is $POSTed.

- Starting - Message $HASP650 is issued because a STARTING volume cannot be drained.
- Halting - Message $HASP650 is issued because a HALTING volume cannot be drained.
- Draining - The spool data set is checked to see if it can drain. If it can, draining continues. If all necessary spool volumes are not all available to drain the volser, the operator is notified via $HASP825. Because the volume is already in a draining state, draining continues and the operator is notified via $HASP630.
- Inactive - Message $HASP650 is issued because an INACTIVE volume cannot be drained.

$P SPOOL processing according to spool volume state -- CANCEL *specified*:

- Active - The spool data set will be deallocated, drained, and all jobs cancelled. The spool data set is checked to see if it can drain. If it can, draining continues. If spool volumes are not all available to drain, the operator is notified via $HASP825.
- Starting - Message $HASP699 is issued because of conflicting parameters. A STARTING volume cannot be cancelled.
- Halting - Message $HASP699 is issued because of conflicting parameters. A HALTING volume cannot be cancelled.
- Draining - Message $HASP699 is issued because of conflicting parameters. A DRAINING volume cannot be cancelled.
- Inactive - A forced cancel of all jobs on the spool volume is initiated.

$S SPOOL

This command causes the designated spool data set to be started. Operands are checked for correctness, using $CFSEL. Processing is dependent on the current state of the spool volume (active, starting, halting, draining, inactive, or a new volume) and how the command was specified. $SVOLOC is issued to obtain the DAS. For a new volume, a return code is saved so that a DAS can be assigned to the new volser. The following table indicates the subsequent processing performed.

Volume Status SPPL Operands	Active	Starting	Halting	Draining	Inactive	New Volume
V=Volser, Format	Issue $HASP699	Issue $HASP699	Issue $HASP699	Issue $HASP699	Issue $HASP699	Process at SSPVNEW
V=Volser, P,Cancel	Issue $HASP699	Process at SSPSTART	Process at SSPSTART	Process at SSPSTART	Process at SSPSTART	Process at $HASP699
V=Volser, P,	Issue $HASP699	Process at SSPSTART	Process at SSPSTART	Process at SSPSTART	Process at SSPSTART	Process at $HASP699
V=Volser, Z	Issue $HASP699	Process at SSPSTART	Issue $HASP630	Process at SSPSTART	Issue $HASP630	Process at SSPVNEW
V=Volser	Issue $HASP630	Process at SSPSTART	Process at SSPSTART	Process at SSPSTART	Process at SSPSTART	Process at SSPVNEW

SSPVNEW: Start a New Volume

An available DAS is obtained for the new volume and $HASP411 MAXIMUM OF nnn SPOOL VOLUMES EXCEEDED is issued if a DAS is not available. Once a DAS is found, it is initialized and control is passed to SSPSTART.

SSPSTART: Start the Volume

The DASFLAG2 is updated to reflect the specific options specified on the $S SPOOL command and HASPOOL is $POSTed to start the volume.

$Z SPOOL

This command halts a spool data set from being used any further in JES2 processing. The operand is checked for correctness. A $QSUSE is issued to acquire access to the job queues, $CFSEL is invoked to select the operand, and a $SVOLOC is issued for the designated volser of the spool volume in code to obtain its DAS. The command is processed to handle the following spool data set states:

- Active, Starting, Draining

  The DASFLAG2 is set appropriately to allow deallocation of the spool volume, then HASPOOL is $POSTed to halt the volume.

- Halting, Inactive

  Message $HASP630 is issued to display the volumes status.

CVLCANCL Routine

The CVLCANCL routine purges all jobs associated with a spool volume. The routine is called as a result of the CANCEL operand on a $P SPOOL or $S SPOOL command. Jobs can be cancelled on an available or inactive volume depending on the command.

Upon entry, the callers registers are saved (via $SAVE). If the jobs should go through normal or forced purge processing and the volume is inactive, forced purge processing is invoked; some track groups may be lost until the next all-systems warm start.

A normal purge cancels a job if any part of the job resides on the volume being cancelled and all other volumes shared by the job are available. Any job that should be moved (QUEMVRQ) will be ignored. Jobs in input processing or STC/TSU type jobs are not cancellable and are also ignored. The $JCAN service routine performs the cancels.

A forced purge is invoked if CANCEL is specified for an inactive volume. If any part of the job is resident on the volume, it is cancelled. The hold count is zeroed, all hold flags are reset, and the unspun flag is turned off. $#CAN is issued to purge the job's output, and $QMOD issued to put the job on the purge queue, indicating the job should be forced off (QUE4CAN).

CVLDRAIN Routine

CVLDRAIN informs the operator of the volumes that are needed in order to drain the volume requested on the $P SPOOL or $S SPOOL command. The draining process cannot complete unless all necessary volumes are available. Necessary volumes are those used by any job residing on the volume being drained. Message $HASP825 is issued containing all the necessary volumes.

Upon entry, the callers registers are saved (via $SAVE) and a "volumes needed" mask is initialized. The job queue is searched and the spools used by each job on the draining volume are ORed into the "volume needed" mask. Then a mask is created that contâins the volumes that cannot have work selected, but are needed by jobs on the draining volume (i.e. inactive and halting volumes). This mask is created by exclusive ORing $SPLEXST with $SPLSLCT and then ANDing the result with the "volumes needed" mask. If the result is zero, no message is issued. A non-zero mask results in message $HASP825 VOLUME volser REQUIRES volid,volid,... TO PROCESS.

HASPCPCE: PCE Commands Group Processor

The following is a description of the individual PCE commands processed by the HASPCPCE group processor.

$T PCE, $D PCE

These commands set and display the tracing status for special processors. They both have a common entry point in HASPCPCE.

The specified processor's PCE is located via the special PCE table (PCETAB) and the $GETABLE macro instruction. If the address for the PCE is zero, there is no PCE for the processor and the $HASP698 message is issued and processing continues with any remaining operands. If the address is nonzero and a display was requested, the $HASP653 message is issued to display the processor's current tracing status; any remaining operands are then processed. If a display was not requested, the PCE's trace flag (PCETRACE) is turned on or off according to the setting of the trace operand (TR=P - processor trace, TR=N - no trace). After the trace flag has been set, the $HASP653 message is issued displaying the processor's new tracing status.

If no operands are specified on the $D PCE command, the $HASP653 message is displayed for each special processor. (A special processor is one that does no I/O. Because they do no I/O, special processors have only one PCE associated with them. Table NPCETBL in HASPINIT contains a list of special processors.)

HASPCSY1: System-oriented Commands Group Processor

This group processor is generated using the DELAY=YES keyword of the $COMGRUP macro instruction. In this routine, executed before control is transferred to the verb processor, a check is made to see if the command is an initiator command. If not, the verb processor is entered via a branch on register 1. If the command is for initiators, the initiator identifier or range of identifiers is extracted for later use. Control is then passed to the specified initiator command.

$D I, $P I, $S I, $T I, $Z I

These commands are used to display, drain, start, halt, or set new execution classes for an initiator, a range of initiators, or all initiators.

Upon entry to the initiator routines, an index value is set based upon the request (command verb type). This is later used for entering the appropriate service routine. The processor determines whether the request was for all initiators or for a single initiator. If the request was for all initiators, a subroutine is entered in which the specified action is done to all initiators.

Note that when an initiator is stopped (drained) or halted using the ALL operand (that is, no identifiers were supplied), the appropriate ALL bits in the partition information table (PIT) are turned on. When an initiator is stopped (drained) or halted using an identifier operand, the appropriate single bit is turned on. Although a start initiator identifier turns off both sets of bits, a "start all initiators" only turns off the ALL bits.

If the request is to start an initiator, a check is made to see if the corresponding PIT has a system initiator. If not, an SVC 34 to start the initiator is issued. In either case, JES2 is posted for a job.

If the request is to set new execution classes for the initiator, the class string is validity-checked. If valid, the string is moved into the PIT. JES2 is then posted ($POST) for a job.

$P

This command is used to stop (drain) the system. The $DRAINED bit in $STATUS (located in the HASP communications table) is turned on, and a response is issued.

$S

This command is used to start the system. The drain flags in $STATUS (located in the HASP communications table) are turned off, and each PCE is posted ($POST). JES2 is posted ($POST) for the job output table.

$P JES2

This command is used to withdraw JES2 from the system. This command verb processor is not entered from the sub-processor group to which it belongs. The entering console is tested to see if it is authorized to withdraw JES2. If so, the system drain flag is turned on. The status of the system is then tested, and if it is currently inactive, a status bit is set in the subsystem vector table (SSVT), and the exit routine is entered via a branch instruction. If there are active functional subsystems (FSSs), the FSSDRAIN flag for each functional subsystem is set to indicate that the functional subsystem application (FSA) in each functional subsystem address space is to withdraw processing support for the FSS.

$V S

This command is used to enter an MVS system command in a job stream or as an automatic JES2 command. The current multiple line write-to-operator (MLWTO) is truncated in preparation for an SVC 34. The operand is checked for the proper format (that is, surrounded by quotes), and the command is then sent to MVS after removing the quotes.

$E SYS

This command is used to restart a system's work that is not currently active. The system, whose system identifier is the operand, is verified as being inactive. It if is inactive, the address of the system queue elements (QSEs) is stored in $ESYSQSE and $WARMPCE is posted ($POST) for work.

$L SYS

This command is used to list status of each system in the JES2 complex. The system queue elements (QSEs) are scanned, displaying a message that signifies the status of each system in the JES2 complex.

$T ALL

This command is used to change the system affinity or independent mode status of all jobs belonging to the designated system. The operands are extracted and saved for later reference. The job queues are scanned and all eligible jobs belonging to the designated system are changed in accordance with the other operand. If any jobs' system affinity is changed, JES2 is posted ($POST) for the job.

$T SYS

This command is used to place a system in or remove a system from the independent mode of operation. If the system is to be placed in independent mode, the $INDMODE bit in $STATUS is turned on and JES2 is posted ($POST) for a job. If the system is to be removed from independent mode, the $INDMODE bit is turned off, and JES2 is posted ($POST) for a job.

HASPCSY3: Console Command Group Processor

This sub-processor group processor is used to process console commands.

$T M

This command is used to set the message routing of a console. The cca operand is extracted and examined for a valid value. If the cca operand is not present, the current message routings for the entering console are displayed. If the cca value is valid, the next operand is examined. If no more exist, the current message routings for the entering console are displayed. If a list of display groups have been supplied, they are extracted by matching the input value with an internal table. The table value is then inserted into the entering console's redirected response area.

The areas are chained to the HASPCOMM processor control element (PCE) work area and are dynamically created during initialization of JES2. One area is built for each console unit control module (UCM) found in the MVS system. Finally, the entering console's revised message routing is displayed.

Note: If the L=cca operand is included with this command, the apparent entry console may be altered, thus causing the named console default display area to be altered.

HASPCAOC: Automatic Commands Group Processor

This sub-processor group processor processes automatic commands.

$C A

This command is used to cancel any automatic commands that have been previously set. The current timer value for any automatically generated commands is cancelled, and all active automatic command elements (ACEs) are freed.

$S A

This command is used to start automatic commands that were previously halted. The flag in the automatic command element (ACE) table that indicated halted automatic commands is turned off. Then, the HALTED ACE queue is examined for any entries. All halted ACEs are chained into the active queue.

$T A

The $TA command is used to:

- Display the automatic commands currently in effect.
- Specify a new command or series of commands for processing. The maximum length of an automatic command entry is 80 characters.
- Modify an existing command entry.

In processing the command, the ID for the ACE is extracted. If there is no ACE, and the request is to create a new ACE, a generated ID is prepared. If the request is to change a previous ACE, the ID must be supplied.

If a new ACE is required, one is taken off the free queue and prepared for addition to the chain. If the request is to change a previous ACE, that ACE is removed from the queue (either active or halted queue). The operands are then extracted from the command. If any operands are invalid or missing, an 'INVALID OPERAND' message is issued and processing for the command is terminated. If the operands are valid, appropriate fields of the ACE are set, and the ACE is added to the active or halted queue. Finally, the ACE is displayed and the command processor is exited.

$Z A

This command is used to halt automatic commands. The halted flag in the automatic command element (ACE) table is turned on, and each active ACE is rechained to the halted queue.

HASPCMS1: Miscellaneous Display Commands Sub-processor

This sub-processor group processes miscellaneous display commands not covered by the other sub-processors.

$D U

This command is used to display the status of units (JES2-controlled devices). All operands are scanned before any display is initiated, and an internal table, built from the operand pointer area at label COMPNTER, is initialized with an entry for each operand. There are four forms of table entry:

- For each specific device operand (RDR1, for example), the table entry contains the address of the device control table (DCT) representing that device.
- For each class name operand (such as PUNS or LNES), the table entry contains the address of the first DCT in the group of DCTs representing the class.

- For each remote work station (RMTn) or range of remote work stations (RMTn-nn), the table entry contains the numerical value of the beginning and end of the range; if a specific work station was indicated, the beginning and end values are the same (as though RMT2-2, for example, had been specified).
- For any unrecognizable operand, the table entry is set to 0.

In addition, an index byte is maintained in the COMREGSV work area for each operand processed. The index bytes in COMREGSV correspond one-to-one with the above entries in the internal table. The index byte contains the offset into a second table (CDUTABLE). CDUTABLE contains instructions on how to process the corresponding operand.

Each display routine returns control to the main display unit loop, which processes each entry in the table that was constructed when the $D U operands were scanned. When all operands have been processed, the display unit routine returns control to the HASPCOMM main entry point, indicating that the INVALID OPERAND(S) DETECTED message is to be issued, if at least one invalid operand was found.

HASPCNT1: Network Job Entry Commands Group Processor

This group processor processes network job entry commands.

$S N

This command is used to start network communication on a line. If this is establishing a BSC connection, an attempt is made (via HASPNSNR) to sign on to whatever system is at the other end of the specified line. If this is establishing an SNA connection, the application name is validated, and control is passed to the $S N exit routine (HASPSNET) in HASPSNA to initiate the session.

$T NODE

This command is used to set and display nodes. The appropriate network information table (NIT) entry is located and updated (if any update parameters are supplied). A display of the NIT entry is output.

$Nn[Mn], $Mn, $N

This command is used to send a command to another member in a MAS configuration or to a specific node off the target member. If the command was entered locally and the destination is valid and reachable, the command text following the semicolon is sent via the remote console processor (HASPMCON) to the indicated node or node member. Command editing and text compression are bypassed.

HASPCSSI: Global Networking Commands Group Processor

This group processor processes global commands entered on the system where this group processor resides. The group processor is generated using the DELAY=YES keyword of the $COMGRUP macro instruction. This initializes register 15 to 1 before branching to the appropriate entry point (via a branch to register 1). This results in register 15 being updated by the number necessary to produce the command option code identifying the command operand in the $G command (the character following the G).

If the command was not entered at this node, an immediate error exit is taken. Otherwise, the receiving node operand (not required of the $G D command) is validated and, if necessary, converted to an internal node number. If this is a $G R command, a check is made for the OUT or XEQ option, one of which must be present. Next, the job name operand (a required operand) is validated and moved to the formatted command area being built in the PCE (COMFCMDA).

The remaining operands (if any) are examined one at a time, using the operand pointer area (COMPNTER) to access each operand and branching to the appropriate subroutine to validate that operand and update the formatted command area.

If the command successfully passes all operand validation tests, a console message buffer (CMB) is acquired and initialized, the formatted command area is moved to the CMB, the CMB is flagged as containing a global networking command (CMBTYPE=CMBTYPEF), and HASPWQUE is called to queue the CMB for console services processing where it will be transmitted to the receiving node. Note that if the command is a global locate ($G D command with no receiving node operand), a CMB is acquired, built, and queued for each mode currently marked as reachable in the node information table (NIT).

The following is an explanation of the individual command requirements for the commands handled by this group processor.

$G D

This command is used to display job information for a job at another node. The operands are validated, and the formatted command area is constructed (COMFCMDA). If no validation errors are encountered, a CMB is acquired, the formatted command is moved to the CMB, and the CMB is queued for transmission to the receiving node. (Note that if the receiving node operand was not specified, a CMB is queued for transmission to each reachable node.)

$G C

This command is used to cancel a job at another node. The operands are validated and the formatted command area is constructed (COMFCMDA). If the D or P operand is present (not both), the appropriate flag is set in COMFFLG. If no validation errors are encountered, a console message buffer (CMB) is acquired, the formatted command is moved to the CMB, and the CMB is queued for transmission to the receiving node.

$G H

This command is used to hold a job at another node. The operands are validated, and the formatted command area is constructed (COMFCMDA). If no validation errors are encountered, a console message buffer (CMB) is acquired, the formatted command is moved to the CMB, and the CMB is queued for transmission to the receiving node.

$G A

This command is used to release a job held at another node. The operands are validated, and the formatted command area is constructed (COMFCMDA). If no validation errors are encountered, a console message buffer (CMB) is acquired, the formatted command is moved to the CMB, and the CMB is queued for transmission to the receiving node.

$G R

This command is used to request another node to reroute job output or reroute a job for execution. The operands are validated and the formatted command area is constructed (COMFCMDA). If no validation errors are encountered, a console message buffer (CMB) is acquired, the formatted command is moved to the CMB, and the CMB is queued for transmission to the receiving node.

HASPCFCP: Received Global Formatted Commands Group Processor

This group processor processes global commands sent by another node and received at the node on which this group processor is resident. This group processor is branched to (before any editing) by the command edit routine (HASPCOME) when it discovers that the console message buffer (CMB) contains a global formatted command (CMBTYPE=CMBTYPEF). This group processor is reached via a special COMTAB entry, COMTBLFC. Since the CMB contains a formatted command that was built by the job entry subsystem at the sending node, command editing is bypassed. Instead, the formatted command area is copied into the processor control element (PCE) work area (COSICMDA), and HASPCFCP is called to process the command.

The formatted command area contains the following fields:

- Command option code (COSIOP): Identifies the command type (delete, cancel, release, hold, or route).
- Flag byte (COSIFLG): Set to indicate the presence of certain optional operands on the cancel ($G C) or route ($G R) command.
- Job name (COSIJNAM): EBCDIC name of the job the command is to act upon.
- Job id (COSIJID): Binary job number assigned to the job by the originating node (optional).
- Originating node name (COSIORGN): The 8-byte EBCDIC name of the node which originated the job.
- Destination node name (COSID): The 8-byte EBCDIC destination name which should equate at the system of entry to a node number (and possibly a remote number); this field is meaningful only for a route ($G R) command.
- Destination remote name (COSIR): The 8-byte EBCDIC destination name which should equate at the system of entry to a remote number (optional); this field is meaningful only for route ($G R) command with the OUT option.

Upon receiving control, HASPCFCP searches the node information table (NIT) for an entry which matches the originating node name in the formatted command area (COSIORGN). If none is found, an immediate error exit is taken. Otherwise, the binary node number is extracted from the NIT and saved for the job queue search.

The job queues are then searched, via the $CFJSCAN macro, for a job queue element (JQE) with a job name, originating node number, and originating job ID matching those from the formatted command. (Job IDs are not compared if none was supplied with the formatted command; that is, COSIJID=0.) If no match is found, an error exit is taken. If a match is found, the resulting processing differs, depending on whether this is a display ($G D) command.

If this is a display command, the $CFJMSG macro is invoked to output job information for display. If an originating job ID was supplied with the formatted command, this is considered to be a unique occurrence, and a return is made to the main command processor (HA$PCOMM). If an originating job ID was not supplied, the job scan is completed, searching for and displaying all jobs which have a job name and originating node matching those in the formatted command. After each display is complete, $CFJSCAN is used in an attempt to retrace the job queue path back to the same JQE. If it is not found or if it is now on another job queue, a return is made to HA$PCOMM, and the LIST INCOMPLETE message is displayed.

If this is not a display command an originating job ID was not specified, the job scan is continued, searching for a job with a matching job name and originating node number. If one is found, an error message (MULTIPLE JOBS FOUND) is output via the $CWTO macro, the formatted command type is changed to display ($G D), and the job scan is restarted and results in a display of job information for all of the matching jobs. If an originating job ID was supplied, or if no duplicates were found during the job scan, a branch is made to the appropriate entry point in the job list group processor (HASPCJB2) if this is a cancel ($G C), release ($G A), or hold ($G H) command, or to the remote job entry group processor (HASPCRM1) if this is a route ($G R) command.

HASPCXIT: Exit Commands Group Processor

HASPCXIT processes the $T EXIT command, which is used to set or display the status of exit points regardless of whether they are supplied by JES2 or by the installation. The exit range(s) are extracted from the command area of the PCE. Then, each exit's corresponding entry in the exit information table (XIT) is located.

HASPCRM1: Remote Job Entry Commands Group Processor

This group processor processes remote job entry commands.

$D M

This command is used to display a message at another node, at another member of a MAS configuration, at a remote work station (attached to this node or another node), or on the JES2 job log on an executing job.

If the first operand begins with R, N, M, or D, the destination is determined, and the message is shipped for display via the $CWTO macro. If a job ID or job name is supplied, the subsystem job block (SJB) for the executing job is located, and the message is queued to the SJB in the same manner as are operator responses to WTORs.

$R

This command is used to route output of a job or device to another device (local or remote) or to another node, or to route a job at another node for execution.

The first operand is extracted and saved in the COMNULOP byte. The second and third operands are extracted, and the appropriate route code is saved. Finally, the optional fourth operand is examined and saved if present.

If the J=operand is specified, the job is located, and if found, a check is made to see if the entering console is the owner of the job. If so, a check is made for a Q=operand. If this operand is present, all job output elements (JOEs) of the specified classes are changed to special routing. Otherwise, the job queue element (JQE) punch and/or print routing is changed to the new routing, and if the type operand specifies ALL, any specially routed JOEs are rerouted to the new destination.

If the R= operand specifies a destination change for a job or output, HASPCRM1 processing issues a $REROUTE call to perform destination authority checking. The job lock is obtained prior to the call; if the job is not available, the $HASP695 JOB IS BUSY - RETRY LATER message is issued.

The JOE will be locked (if it can be processed) in the case of rerouted SYSOUT. A check for local output routing is performed after a $REROUTE call to determine the whether a $#MOD or $#POST is needed. If the request is successful for the first PDDB in the JOE (except for requests to reroute locally, which must be successful for all PDDBs), processing continues. Local requests are requeued. For network-bound output, the JOE will be $CKPTed and a $#POST issued. If the reroute fails, the routing is not changed and processing continues.

If the request is to change execution routing, a TYPE=JOB $REROUTE call is made. On successful calls, processing continues (as described in the following paragraph), the new JQEXEQND is set, the JQE checkpointed, and the job requeued if necessary. Otherwise, the next job will be processed.

If the R= operand is specified, the job queue is examined for any JQEs that are owned by the R= destination. If the request is made with the Q= operand, any special JOEs with the R= destination and of the requested classes are rerouted to the D= destination. Otherwise, any JQEs that have print and/or punch routing for the R= destination will have their routing changed to the D= destination and if the type was ALL, all specifically routed JOEs with the R= destination will be routed to the D= destination.

$JCANR: Cancel Job Service Routine

The $JCANR routine is entered by JES2 main task processors to schedule a job for cancellation. Upon entry, register 1 contains the address of the job queue element (JQE) for the job, and register 0 contains request information as described following.

CCJRDUMP: If this symbolic flag is included and the job is cancelled out of execution, a dump results that is compatible with the operating system command C jobname, DUMP.

CCJRPURG: If this symbolic flag is included, the CCJRDUMP flag may be included, and the routine attempts to schedule the job for purge, deleting its current activity.

CCJRSTOP: If this symbolic flag is included, other request flags are not included; the routine attempts to schedule the job for purge but does not delete its current activity.

If none of the above flags are included, or if only the CCJRDUMP flag is included, the routine attempts to schedule a job for output, deleting its current execution or pre-execution activity.

If the request is for a started task control (STC) or time-sharing user (TSU) job that has not passed the execution phase, the request is rejected, and control is returned to the NOT JOB exit of the $JCAN macro instruction.

The operating system command scheduling control block (CSCB) is interrogated to determine whether the job is cancellable. If not, the NOT JOB exit is given control.

If neither the CCJRPURG nor the CCJRSTOP flag is on, the request is rejected if the job has passed the execution phase, and control is returned to the NOP exit of the $JCAN macro instruction.

If either the CCJRPURG or the CCJRSTOP flag is on, the JQE is released from any hold status it may have, and flagged for cancellation and purging.

If a cancelled job indicates held data sets, this routine obtains a job disposition request (JDR) element using $GETWORK and queues it to the $JDRQUE. Then, $JCANR $POSTs the JDR processor for work. If the job has any job output elements, the $#CAN macro instruction is executed to cancel all job output elements (JOEs) for the job which are not currently busy.

If the request is to cancel a job on a reader device, the appropriate reader device control table (DCT) is flagged for deletion. DCT stop flags are reset, and the owning processor is posted ($POST). If the request is to purge a job and there are JOEs which were not freed by the $#CAN macro instruction, the output device DCTs which are busy are located, and the activity is deleted as described above for readers.

If the JQE is for a system dataset STC, it will not be cancelled if the JQENAME does not begin with "RMT", or if the JQE is busy.

If the JQE for the job is not owned, a $QPUT macro instruction for $OUTPUT is executed. ($QPUT assigns final queue type based upon the resulting JQE flags and queue counts.) If the JQE for the job is owned, a checkpoint write is scheduled for the JQE.

If the JQE type indicates that the job is in execution and the request is not CCJRSTOP, an MVS cancel command is used to cancel the job either with or without a dump, depending upon the CCJRDUMP request flag setting. Control is returned to the OK exit of the $JCAN macro instruction.

Upon exit, register 15 is set to appropriate exit offsets which match the offsets generated by the $JCAN macro expansion.

$IOTPUR: IOT Purge Service Routine

The $IOTPUR service routine is entered from the $PJ command sub-processor to deallocate track space for purged spin data sets.

On entry, register 1 contains the address of a job output element (JOE) for a spin data set to be purged. The $JOEBUSY flag is turned on to ensure that no other processor gains control of the JOE. An input/output table (IOT) buffer is acquired ($GETBUF), and the spin data set IOT is read in ($EXCP). If the IOT is an allocation IOT, the spin data set tracks are purged ($PURGE IOTTGMAP), the PDB1PSO flag is turned off in the PDDB (to make this data set ineligible for warm start), and the updated IOT is written back to disk.

HASPSERV: Command Services

Command services consists of routines that support processing of JES2 initialization and JES2 command processing. The functional description of these support routines follows and includes any macro interface descriptions that apply.

SUBRRT: $REROUTE Subtasking Routine

The SUBRRT routine subtasks the $REROUTE routine, which verifies authority to reroute jobs or output to a specified destination. SUBRRT is called from HASPCJB2 and HASPCRM1 in HASPCOMM and is passed a JOE address, if destination checking is to be performed for SYSOUT; otherwise, HASPCOMM is passed a JQE address.

SUBRRT acquires a reroute work area, $RRTWA, and initializes it with the JOE or JQE address, PCE address, and reroute flags. A SAFINFO parameter list is also initialized and $SUBIT is issued to set up the subtasking for the $REROUTE routine in HASPSSRV.

If $REROUTE passes back new USERIDs and SECLABELs for the job or output, these are set in the JQE and/or JOE (after regaining control of the queues).

A return code is passed back to the caller in register 15. If subtasking is successful and the destination is authorized, the return code is 0. If the subtasking fails, the $HASP077 message ACCESS REQUEST DENIED DURING REROUTE PROCESSING is issued and a return code of 4. A return code of 4 is also set if the destination is not valid or if access is not permitted.

A $WAIT can occur during processing. This routine also requires access to the necessary queues on entry and just prior to returning to the caller. When processing a $TO command, the JQETMOD and JOETMOD flags must be set. JQE1BUSY must be set for all $R requests and JOE1BUSY must be set for $R ALL, $R PRT, and $R PUN. (Setting JOE1BUSY requires the job lock also.)

WSSERV: Work Selection Service Routine

WSSERV manages work selection of both jobs and SYSOUT. It obtains the WSA, which is the work selection work area that is set up before each call to WSSERV, for information such as comparison and device control block addresses, the work selection table address, and the address of the work selection list. It first determines which criteria were specified for specific devices in the work selection list defined by the device and finds the entry corresponding to each of these criteria in the work selection table. Then the routines needed to process these criteria are called. These routines are specified in the work selection table. Refer to *JES2 Customization* for information on the work selection tables ($WSTAB) and *JES2 Initialization and Tuning* for information about the WS operand on the printer/punch, OFFn.ST, OFFn.SR, OFFn.JR, OFFn.JT statements.

Display Device Control Table (DCT)

The $CFDCTD macro is issued to cause the DCT to be displayed. Expansion of the macro creates code that invokes the COFDCTD routine, which actually performs the display.

The $CFDCTD macro displays the device name as known to JES2, the unit address, and the activity status of the device control table (DCT) requested. Optionally, an extended display may be requested that details the complete status of the DCT (format varies with the device). For example, an extended display of a printer DCT includes the name, unit address, and activity status plus such information as the current forms, carriage, train, job name and number (if active), separator option, and pausing option.

Registers Used: R0, R1, R14, R15

Work Areas Used: COMMAND is used as an output area and COMJNAME is used as an output area extension.

DCT=(R1): This is the address of the DCT to be displayed.

EXT=NO: This indicates whether extended display of the DCT is desired. When set to YES, extended status is displayed.

COFDCTD: Device Control Table Display Routine

HASPCOMM calls COFDCTD to create a status message ($HASP628) in an area labeled, COMMAND, and initiates an operator response. For the device associated with the input DCT, COFDCTD fills the COMMAND area with information that includes the following items:

- Device address
- Job number and job name if the device is active
- Output-related information if the device is a printer or punch (such as forms, record, and page counts, FCB, UCS, page limits, output classes, and process mode).

  Note: For a printer that prints page-mode data, such as the 3800 model 3, this output-related information can include the name of an associated functional subsystem and related time, line, and page limits that were in effect when checkpoints were taken.
- For a reader device, job and message classes, system affinities, default routings
- RJE line disconnection state
- DCT logging and password status
- Remote attribute table (RAT) data

COFDCTD provides for an extended DCT display and issues a $CWTO to write the final message before returning to the caller.

If the device associated with the DCT display is a printer that prints page-mode data, such as the 3800 model 3, operating as a functional subsystem, COFDCTD invokes the internal subroutine SRVFSS to obtain information from the functional subsystem about the device. If the printer is halted by operator intervention, SRVFSS issues a cross memory move (MVCP) to get the address of the JIB that is currently being processed. The JIB is then checked for a CHK record. The CHK record contains current page and record counts. If there is no CHK record, the total counts from the copy of the JOE in the JIB are used to give page and record counts. If the printer is not halted, SRVFSS enters cross-memory mode and issues a query order to the FSA in the functional subsystem address space to supply the record and page counts in a response area. SRVFSS then exits from cross-memory mode and returns to COFDCTD.

SRVM630: $HASP630 Message Formatting Routine

HASPCOMM calls SRVM630 to format the $HASP630 message; HASPCOMM supplies as input the DAS address and the address of the area where the $HASP630 message is to be built.

After first initializing the message area with the spool name, SRVM630 checks the status of the spool volume by examining the DAS, stores its status (active, starting, halting, or draining) in the message area, and returns to the caller.

$SVOLOC: Scan the Direct Access Spool Control Blocks

The $SVOLOC macro is used to scan the direct access spool (DAS) control blocks.

Format Description

```
[symbol]  $SVOLOC  [VOLSER=(R1)|addrx]
                   ,ACTIVE=addrx
                   ,START=addrx
                   ,HALT=addrx
                   ,DRAIN=addrx
                   ,INACT=addrx
                   ,NOEXIST=addrx
                   ,INVALID=addrx
```

VOLSER=
Specifies the address of the EBCDIC volume serial number for the direct access spool.

ACTIVE=
Specifies the branch address to SRVOLOC when the volser is active.

START=
Specifies the branch address to SRVOLOC when the volser is starting.

HALT=
Specifies the branch address to SRVOLOC when the volser is halting.

DRAIN=
Specifies the branch address to SRVOLOC when the volser is draining.

INACT=
Specifies the branch address to SRVOLOC when the volser is inactive.

NOEXIST=
Specifies the branch address to SRVOLOC when the volser does not exit.

INVALID=
Specifies the branch address to SRVOLOC when the volser is invalid.

SRVOLOC: DAS Scan Routine

Using the volser passed on input, SRVOLOC finds the direct access control block (DAS) associated with the volser and passes back its address in R1. If the input volser does not exist, R1 is set to zero and a return code of 20 is returned to the caller. If the input volser is null, too long, or does not match $SPOOL, R1 is set to zero and a return code of 24 is returned to the caller. Along with the DAS address, SRVOLOC returns a return code indicating the status of the spool volume (active, starting, halting, draining or inactive).

$WSSCAN: Work Selection Scan Macro

This macro generates code that calls the work selection scan routine SRVWSCAN; $WSSCAN loads R1 with the address of the WS parameter and R0 with the output area in which SRVWSCAN puts the completed work selection list.

Format Description

```
$WSSCAN   LIST=addrx|(R0),
          WS  =addrx|(R1)
          WSTADDR=label or register
          WSTLISTL=L'    '
```

LIST

Specifies the symbolic name of the area where the work selection list is to be built.

WS

Specifies the operand of the WS keyword either from the $T command or the output device initialization statement.

WSTADDR

Specifies the address of the device work selection table pair. This address will be loaded into register 15.

WSTLISTL

Specifies the length of the work selection list. It is passed to the routine via an inline parameter list.

SRVWSCAN: Work Selection Scan Routine

SRVWSCAN scans the WS operand supplied from the $WSSCAN macro and builds a work selection list in the area, whose address is also supplied on input.

After the work selection list is built in the input area, SRVWSCAN returns to the caller with a return code that indicates one of the following results:

- RC=0 - The WS operand is valid. R0 contains the address of the end of the WS operand.
- RC=4 - The WS operand contained an illegal character, an unknown criterion, or contained information not valid for the specific device.

SRVSETUP: Work Selection Setup Routine

This routine is called by any device (with the $WSSETUP macro) requesting work selection before the call to work selection services. It clears the work selection work area fields and creates the device volumes mask if a volume was specified in the work selection list.

WSROUT: Work Selection Route Code Routine

This routine determines if the output element's route code matches any of the device's route codes. If the output element's route code exceeds all of those of the device, the routine sets an indication in the $WS flag, so that $#GET can terminate the queue scan and advance to the next queue.

$CFPRSCN: PRMODE Parameter Scan Macro

JES2 initialization and HASPCOMM use this macro to generate a calling sequence to SRVPRSCN, supplying as input the address of the DCT/RWT index list and the address of the PRMODE operand that was supplied by the operator on the $T DEV command or supplied in a JES2 initialization statement.

Format Description

```
{symbol} $CFPRSCN LIST=addrxl(R0),
                  PR  =addrxl(R1)
```

LIST
Specifies the address of the DCT/RWT index list.

PR Specifies the address of the PRMODE operand.

SRVPRSCN: PRMODE Operand Scan Routine

JES2 initialization and HASPCOMM use the $CFPRSCN macro to call SRVPRSCN to determine the indices in the DCT that are represented by the values specified on the PRMODE operand. SRVPRSCN scans the PRMODE operand that was specified from either an initialization statement or in the $TDEVN command, and then builds a temporary PRMODE table and a temporary PRMODE index list in $GENWORK of the $HCT work area.

If no errors occur during processing, SRVPRSCN updates the PRMODE table, returns in the output area (received on input) the updated DCT, and returns a zero return code. If errors occur during processing, SRVPRSCN makes no changes to the DCT table and PRMODE table and returns to the caller with an error return code.

SRVROUT: Route Code Sub-parameter Conversion Routine

This routine is used to verify the route code sub-operand on the local and remote printer/punch statements (initialization only) and the OFFn.ST, OFFn.JT, OFFn.SR, and OFFn.JR (initialization and commands). It converts a character route code to binary. The routine is called by post-scan routines in HASPSCAN before the DEST tables have been initialized.

SRVSASCN: Verify System Affinity Routine

$CFSAF calls SRVSASCN to verify one affinity or a list of system affinities. SRVSASCN receives the address of the affinity list to be validated and the device affinity mask from the calling routine via $CFSAF. If they are valid, this routine sets the mask to indicate the affinities specified on the initialization statements or $T command.

SRVMODIF: Modify Routine

This routine uses the information in $WSTAB to modify the characteristics of a job or data set when being loaded from an offload data set. For a job, CLASS, ROUTECDE, HOLD, and SYSAFF can be modified. For a data set, BURST, DSHOLD, FCB, FLASH, FORMS, HOLD, PRMODE, QUEUE, ROUTECDE, UCS, and WRITER can be modified.

HASPSSRV: JES2 Subtask Services

This module contains various JES2 subtask services; subsequently, it is not invoked directly by the JES2 main task because MVS WAITs can occur during processing. HASPSSRV contains the following entry points:

- DSFOPEN - Data set fake open routine.
- DSFCLOSE - Closes the fake opened data set.
- $LOGMSG - Puts message(s) into a job's SYSMSG data set and, optionally, performs a WTO.
- PSAFSCAN - PDDB scan and SAF call subroutine.
- NEWSCRE - Completes initialization of the JESNEWS PDDB and requests authority to create the JESNEWS data set.

HASPSSRV also contains the following routines, which perform NJE authorization and destination validity checking:

- JOBVALM - Performs job validation.
- RPDBSEC - Performs system PDDB verification.
- SYSOVFY - Performs SYSOUT validation.
- $REROUTE- Authorizes the final destination for a job or data set reroute requested by a $R or $TO command.

DSFOPEN: Data Set Fake Open Routine

DSFOPEN performs a fake open of the system messages data set, allowing WRITEs to the data set. Upon entry, the routine validates the JCT and IOT and ensures that they have spool MTTR addresses. If either of these input parameters is incorrect, DSFOPEN sets a return code of 4 in register 15 and returns.

Otherwise, DSFOPEN GETMAINs an SJB, and initializes its prefix area. (A GETMAIN failure terminates processing with a return code of 8.) SJBINIT is called to obtain an SJXB and initialize fields in both the SJB and SJXB. DSFOPEN then obtains the RPL address and performs the fake open for the job-related data set. It copies fields from the PDDB to the SDB and opens the data set for output.

If the data set has been opened before, it is repositioned to the last spool record. DSFOPEN then sets a return code (0 for successful open, 4 for an internal error, 8 for a storage problem, or 12 for an I/O error), frees control blocks if an error occurred, and returns to the caller.

DSFCLOSE: Data Set Close Routine

This routine closes a data set opened by DSFOPEN. It calculates the SJXB address (using the passed RPL address), obtains the addresses of other needed control blocks, fake closes the data set, and frees the SDB. DSFCLOSE calls $CBIO to write the primary allocation IOT, then frees the SJB and related control blocks, and returns.

$LOGMSG: SYSMSG/WTO Routine

$LOGMSG places a job-related message into a job's SYSMSG data set and, optionally, issues a WTO to the operator's console. It is passed an inline parameter list defined in member $LG of $PARMLIST; on exit, register 15 contains a return code of 0 if there are no errors, or 4 if $LOGMSG is unable to open the data set requested. WTO and FREEMAIN processing for messages is still performed (if applicable) if the open for data set fails.

$LOGMSG calls DSFOPEN to open the data set. If successful, $LOGMSG loops through the messages, performing a PUT to the job's SYSMSG data set for each message and WTO/FREEMAIN processing, as indicated. When all messages have been processed, $LOGMSG $CALLs DSFCLOSE to close the data set if the open was successful, sets the return code, and returns.

PSAFSCAN: PDDB Scan and SAF Call Subroutine

PSAFSCAN scans each PDDB in an IOT to obtain the data set's name and token, which are needed to issue the MODIFY form of the RACROUTE macro. Following the RACROUTE invocation, PSAFSCAN issues the $SEAS macro to make the SAF call for each data set (if the CREATE call made previously for the data set was successful).

Upon entry, PSAFSCAN is passed a parameter list containing:

- A JCT address (if deletes are to be performed in behalf of a job) or the address of $JESTOKA (if deletes are for JES2).
- An IOT address and the PDDB offset into the IOT.
- The WAVE address.

PSAFSCAN uses these addresses and those of the first and last PDDBs to determine if a deletion is to be performed on behalf of the job or JES2. If for a job, PSAFSCAN sets the job exit mask from the JCT for use with exits 36 and 37 and obtains the security token that represents the job. Otherwise, PSAFSCAN obtains the JES2 security token. Next, the SAF parameter list located in the WAVE is initialized and a RACROUTE request is issued using the appropriate token. PSAFSCAN then issues the $SEAS macro to make a SAF call for deleting the data set.

Processing continues (SAF parameter list initialization and subsequent RACROUTE and $SEAS invocation) for each PDDB until all have been processed, then PSAFSCAN returns control to its caller.

NEWSCRE: JESNEWS Create Routine

NEWSCRE completes the initialization of the PDDB for a new JESNEWS data set. The routine builds the token for the new JESNEWS data set and ensures the requestor has sufficient authority to delete the old data set and create the new data set. If either authorization fails, the job can not create the JESNEWS data set.

NJE Authorization/Validation Routines

The following describes the JOBVALM, RPDBSEC, SYSOVFY, and $REROUTE routines, which perform destination verification and authority checking of jobs and SYSOUT routed through a network.

JOBVALM: Job Validation Routine

JOBVALM performs processing to validate the destination and check the authority level of jobs transmitted through a network. It is subtasked by the RJOBVFY routine in HASPRDR (via $SUBIT). RJOBVFY initializes the SAFINFO parameter list with available information about the job and its associated security token, which contains destination and authorization data to be verified. The SFITOKEN field in the parameter list contains the token address and the SFI2STKN flag, which indicates whether the token represents the job submittor or the job itself.

JOBVALM obtains the parameters needed to issue macros that actually perform the authority and destination checking from the SAFINFO parameter list. It also sets fields in the SAFINFO parameter list for use by the RJOBVFY routine when JOBVALM has completed its processing.

JOBVALM issues a RACROUTE VERIFY request and sets a return code accordingly. If a default token is to be used for authority checking, then JOBVALM issues the $SEAS macro following the RACROUTE request. (A default token is used for network-bound jobs submitted from a source other than an internal reader.)

If RACROUTE processing is successful, JOBVALM issues a $DESTCHK call to check the JCTXEQND, JCTPROUT, and JCTPUOUT fields. If an error is detected in any one of these fields, JOBVALM sets the SFIRESCD field in the SAFINFO parameter list to indicate to RJOBVFY that the job is to failed with a JCL error and a message written to the JCLIN data set. Based on the RACROUTE return code, JOBVALM also sets the JCTTOKEN field in the SAFINFO parameter list, indicating that the security token returned to RJOBVFY represents the job submittor, or sets the PDBTOKEN field, indicating that the token represents the job itself.

JOBVALM sets one of the following return codes in register 15 and and performs further processing based on the result of the RACROUTE request:

0 Processing was successful. (The destination is valid and the job has sufficient authority or RACROUTE had insufficient information to verify destination/authority.)

4 RACROUTE determined that the job has insufficient authority or the destination is invalid. Any messages provided are issued by the $LOGMSG routine; RJOBVFY is to continue processing.

8 A severe parameter error occurred. (The PCE, DCT, WAVE, or JCT address was unavailable.) RJOBVFY is to fail the job.

RPDBSEC: System PDDB Initialization Routine

RPDBSEC initializes system PDDBs with a token, security label, and system data set name. It is called via $SUBIT from RJOBVFY in HASPRDR and passed the SAFINFO parameter list. RPDBSEC invokes RACROUT and $SEAS to obtain security authorization information and passes back one of the following return codes:

- 0 - Processing was successful. All required PDDBs have been initialized.
- 4 - An authorization failure occurred.
- 8 - The JESMSGLG data set could not be created. The job cannot be printed and will be purged.

SYSOVFY: SYSOUT Validation Routine

SYSOVFY is subtasked by the NSRAUTH routine in HASPNET (via $SUBIT) and $CALLed by the $REROUTE routine in HASPSSRV to validate destination and authority for SYSOUT while on this JES2 node. It issues RACROUTE calls to obtain security tokens to represent SYSOUT, builds data set names, and issues SAF data set create authorization calls. It also issues a RACROUTE VERIFYX call to obtain a token to represent SYSOUT from a local job.

The SYSOVFY routine obtains the addresses of the PDDB, JCT, PCE, and WAVE from the SAFINFO parameter list. If the JCT, PCE, or WAVE address is not available, SYSOVFY sets a return code of 8 (disastrous error) in register 15 and returns.

Otherwise, SYSOVFY issues a RACROUTE REQUEST=VERIFYX to obtain a security token. If the PDDB address (supplied to SYSOVFY in the SAFINFO parameter list) is zero, SYSOVFY places the token in the JCT to protect the job locally and then returns to its invoker. If non-zero, SYSOVFY builds a data set name and saves it (along with information extracted from the NJE header and security token) in the PDDB.

SYSOVFY then issues a RACROUTE REQUEST=AUTH to determine if the job to which the SYSOUT belongs has sufficient authority to create a SYSOUT data set, sets a return code in register 15, and passes control back to its invoker. A return code of 0 is set (if the CREATE call, itself, returned less than 8) to indicate successful processing and access authority is permitted. A return code of 4 indicates that a valid token was not returned by the RACROUTE VERIFYX request.

Note: If the create fails after the VERIFYX was successful, SYSOVFY sets an 8 return code (disastrous error); this should never occur.

$REROUTE: Destination Authorization Routine

The $REROUTE routine determines whether or not a job or SYSOUT data set rerouted by a $R or $TO command has sufficient authority to be transmitted to its final destination (including locally). When a new destination is specified, $REROUTE is subtasked by the SUBRRT routine in HASPSERV and passed a single parameter, the $REROUTE work area ($RRTWORK), which contains a JQE address (to process a job) or a JOE address (to reroute SYSOUT).

During its processing, $REROUTE sets fields in the SAFINFO parameter list (passed to other routines for authorization checking) that contain:

- The origin and execution user ids and nodes.
- The PDDB address and JCT address.

If this is the executing node and $REROUTE determines that the job/SYSOUT is permitted to be transmitted to the new destination, it replaces any necessary security information (only if this is a store-and-forward node and the job/SYSOUT is being forwarded to a local node). Upon completion, $REROUTE sets a return code in register 15 of 0 (the destination is valid and the job/SYSOUT may be transmitted) or 4 (the destination is not valid or authority is insufficient; transmission denied).

Regardless of where SYSOUT is being rerouted, $DESTCHK is called only once. It is assumed that other PDDBs would succeed or fail based on the success/failure of the first PDDB.

For rerouting to a non-local node (regardless of current routing), the checking ends here because a valid local or store-and-forward token has already been obtained. If the rerouting is to the local node and there is no local token (as indicated in the JOE), then SYSOVFY is called for each PDDB to obtain a local token for each. In addition, if there is no local token for a job, SYSOVFY is called once more to obtain a job token.

The tokens used as input to SYSOVFY are the received data set tokens (if available) for the PDDBs and the received job token (if available) for the job.

The following further describes $REROUTE processing for the four possible cases of job or SYSOUT rerouting to a new local or non-local destination.

Job Rerouted to New Non-local Destination:

For this case, the $REROUTE routine accesses the JCT to obtain the job's security token and passes the token (and a binary route code) to the $DESTCHK routine in HASCSIRQ to determine whether or not the job can be sent to the requested destination. If not, $REROUTE sets a return code of 4 in register 15 to indicate that the command should be failed. Control returns to $REROUTE's caller.

Job Rerouted to New Local Destination:

In the case where the new destination is local (and the old destination was NOT local), $REROUTE accesses the JCT to obtain the $DEST default node (in the JCTROUTE field) and token. The job is processed normally; RJOBVFY and JOBVALM replace the store-and-forward token in the JCT with a valid local token when the job is later sent to the reroute job transmitter/receiver.

SYSOUT Rerouted to New Non-local Destination:

For this case, all of the PDDBs in which the destination must be changed must be authorized for rerouting. The $REROUTE routine calls the $DESTCHK routine for each of these PDDBs, passing the information in the first PDDB's PDBTOKEN. If the $DESTCHK fails, the JOEROUT/JOEDEST field will not be changed. Because all of the PDDBs in the JOE have have the same user id and security label (and the same USERID and SECLABEL values in the token), only one check is necessary; it is assumed that subsequent checks would also fail.

SYSOUT Rerouted to New Local Destination:

In the case where the new destination is local (and the old destination was NOT local), $REROUTE accesses the token SCR and calls SYSOVFY to obtain a valid local security token. This new information is stored into the PDDB.

HASPRDR: Input Service Processor

The functions of the input service processor are to:

- Read card images from an input device or from another node in the job entry network.
- Detect and scan JOB statements, extracting parameters for job accounting, job control, authorization processing, and print and punch identification.
- Detect and process other JES2 control statements such as COMMAND, NETACCT, PRIORITY, ROUTE, SETUP, MESSAGE, JOBPARM, OUTPUT, XEQ, DD*, and DD DATA.
- Assign a unique JES2 identifier (job id) to each job.
- Log jobs into the JES2 subsystem.
- Assign job priority based upon the PRIORITY statement, JOB statement, or JOBPARM statement parameters.
- Generate, from statements read, a JCL file and input data files, and record these files on direct-access storage device(s) for later use by the job execution states (spooling function).
- Generate JES2 job control tables, input/output tables, job queue elements, and other JES2 control blocks required for later job processing.
- Queue jobs for processing by the JCL conversion processor, for processing by the output processor, or for transmission to another node in the network.
- Validate job information and perform security authorization processing.

Each input source defined to JES2 — that is, each local card reader, internal reader, and remote reader — is represented by a device control table (DCT). Associated with each reader DCT is a processor control element (PCE), specifying what input service processor is to service the input device. When JES2 is started, the reader PCEs (among others) are queued on the $READY queue. As the JES2 dispatcher dispatches the ready PCEs, the input service processor receives control as each reader PCE is dispatched. (The input service processor is reentrant: all information specific to a particular PCE dispatch is maintained within the PCE. This permits the input service processor to process jobs concurrently for a number of input devices that have different characteristics.)

Initial Entry Point: The processor receives control at entry point HA$PRDR, moves the command character specified at JES2 initialization into the control statement routing table, and proceeds to the processor initialization phase, $READ.

$READ: Initialization Phase

The input service processor initialization phase, label $READ, receives control (via entry point HASPRDR) upon the initial dispatch of each reader processor control element (PCE). During initialization, the RDW4JVFY, RDW4STKN, and RDW4SREQ flags are reset to perform security authorization checking for incoming jobs.

RGETUNIT: The initialization routine attempts to acquire the input device associated with this PCE: a local card reader, internal reader, remote reader, or network job reader. If the device is not available, the processor waits ($WAIT UNIT) for the device. For a local device or internal reader, the routine waits for a post of the resource. For a remote device, the routine waits for a specific post indicating that the device is available. When the wait is satisfied, the routine branches to RGETUNIT to repeat the GETUNIT request; this cycle is repeated until the required device becomes available.

RGOTUNIT: When the unit has been obtained, a $ESTAE macro is issued, which establishes the RDRRCV0 routine as the recovery routine to receive control in case of an abend condition in the HASPRDR processor. RGOTUNIT issues the $ACTIVE macro instruction to indicate that a processor is active, then RGOTUNIT determines the type of device that has been acquired. If the input device is an internal reader and a buffer is not already available, main storage is obtained from subpool 231 (fetch-protected storage in the common storage area) for the internal reader device, and RINRSTRT branches to RRETURN.

RNOTINR1: If the input device is a network job receiver (NJR), fields in the job queue entry (JQE) are initialized. An attempt is made to reserve a JQE for the input processor and, if successful, a request (via ROPEN) is made to open the remote terminal. If the open attempt is unsuccessful, $HASP127 message ("******* INPUT NOT ACCEPTED -- JOB QUEUE FULL") is issued; the device is released (via $FREUNIT); the active count is decreased by one (via $DORMANT); and a branch is taken to RNJWAIT.

RNJWAIT: RJNWAIT branches to RJEWAIT to wait for work if the processor is a route receiver or the device is an offloaded spool device. Otherwise, RNJWAIT places the device in a hold status, issues a negative close ($EXTP NCLOSE), and waits (via $WAIT) for work.

ROPEN: If the input device is a remote terminal or a network job receiver (NJR), transmission is initiated by calling RTAM to open the remote terminal device control table ($EXTP OPEN). The device is placed in hold status to indicate that it is unavailable for use by another processor. RDRSW2 reader switches are zeroed, and the JCT pointer is cleared.

RINBGET: If the device is a local reader, RINBGET issues the $GETBUF macro instruction to obtain an input buffer. A chain of channel command words (CCWs) is constructed in the input buffer, to be used in reading input records into the rest of the buffer; the CCWs are constructed such that successive input records are read into adjacent areas in the rest of the buffer until the buffer is full. With the CCW chain complete, the routine enters RRETURN.

RRETURN: The RJFLUSH bit is set to 1 in RDRSW and RXITFLAG is set to zero. If the input device is not a network job reader, control is passed to the main processor to process the next input card. If the input device is a network job reader, the first statement is read into storage. If that statement is not a job header, control is transferred to SKIP50 to issue error message HASP121 ('ERROR READING JOB HEADER OR TRAILER'). If the input card is the job header, information received in the job header is converted and stored in the device control table (DCT) and the processor control element (PCE) work area. Control is then transferred to the main processor.

Main Processor Phase

The main processor phase reads statements from the input device, scans each statement to detect JES2 control statements, and for all JES2 control statements and JCL statements (except JOB statements), invokes exit point 4. Exit 4 allows user exit routines to scan the control statements. Depending upon the return code from the user exit routine(s), either the job is aborted, control statement scanning is skipped, or control statements are processed as follows:

- /*control statement: The control statement processing routine (HASPRCCS) is called to process the control statement and take any appropriate action.
- JOB statement: The JOB statement scan routine (HASPRJCS) is called to terminate the previous job (if any), to scan the JOB statement, and to initiate the processing of the following job. If job submission is using an internal reader, HASPRJCS also propagates the authorization information from the job statement to the job header in the JCT.
- DD * or DD DATA statement: The DD */ and DD DATA statement scan routine (HASPRDDS) is called to scan the DD statement for a DLM parameter and to construct a peripheral data definition block (PDDB) reflecting the track address and physical characteristics of the input data set.

If the device is a network job reader (NJR) and the current job is not destined for this node, then JES2 serves as a store-and-forward node. In this case, the entire job is written to spool without any JCL statement or JES2 control statement syntax file. If the delimiter specified on the /*XMIT statement is encountered, the job is terminated.

When an end of file is detected on the input device, control is given to the termination phase. In the termination phase, exit point REXITA (for exit 20) is taken. Exit 20 allows installation exit routines to modify a job's priority, execution node, and system affinity. Finally, the $ESTAE macro is used to cancel the recovery environment established for this processor.

HASPRTRM: Termination Phase

The input service processor termination phase, entry point HASPRTRM, receives control from the main processing phase when an end of file is detected on the input device that is being processed. HASPRTRM calls the RJOBEND subroutine to terminate the last job, if any, then determines the type of input device that was being serviced.

If the device is an NJR, termination ensures that the current record is a job trailer and moves the trailer into the job control table (JCT). If no job trailer is found or if the job had been restarted by the receiver, an error is sent to the job transmitter by issuing a negative close (NCLOSE) to RTAM.

If the device is an internal reader, the routine clears the buffer pointers in the device's device control table (DCT) and for this device frees only any existing fetch-protected buffer before exiting to the common exit routine RUNFREE.

RNOTINR2: If the device is a remote or network job reader, RNOTINR2 issues the $EXTP CLOSE macro instruction for the device and enters RUNFREE.

If this is a spool offload device and the job has been rejected by work selection, the routine places the JQE on the purge queue.

RNOTRJE2: If the device is a local reader, RNOTRJE2 issues the $FREEBUF macro instruction to free the input buffer, sets the DCTHOLD indicator to mark the device unavailable, and enters RUNFREE.

RUNFREE: The common exit routine issues the $FREUNIT macro instruction to release the input device; issues the $DORMANT macro instruction to decrease the active processor count in the HASP communications table (HCT); and returns to the main processor at the label $READ. The processor attempts to acquire (via $GETUNIT) the next input device requiring service or waits until an input device does require service.

HASPRCCS: Control Statement Processing Routines

HASPRCCS is called whenever the main processor phase reads an apparent JES2 control statement. HASPRCCS uses a table lookup method to select the appropriate processing routine.

RCOMCARD: JES2 Command Statement Processing Routine

The RCOMCARD routine determines whether a job is being read and rejects the command (by branching to the illegal statement subroutine, RILLCCRD) if a JES2 control statement is encountered within a job. Otherwise, RCOMCARD gets a console message buffer, waiting until a buffer is successfully obtained if necessary, then logs the command and displays it at the operator's console. Finally, the command is added to the head of the queue of commands to be processed by the command processor, the command processor is posted ($POST WORK), and the routine exits to the main processor. Any commands received from a network job reader (NJR) are ignored.

RPRICARD: PRIORITY Statement Processing Routine

If the device is not an NJR, the previous job (if any) is terminated, the specified priority is converted to binary and saved, and control is returned to the main processor. If a JOB statement does not follow, the message "DEVICE SKIPPING FOR JOB CARD" is displayed at the operator's console, the effect of the PRIORITY statement is nullified, and the input stream is scanned for another PRIORITY or a JOB statement. A PRIORITY statement received from an NJR is ignored.

ROUTCARD: ROUTE Statement Processing Routine

The appropriate routing fields are set to the values associated with the indicated destination. If an invalid field is encountered, the $HASP111 message "INVALID /* ROUTE CARD" is placed in the output file and is displayed at the operator's console. Further processing for the job with that ROUTE statement is bypassed.

RSETCARD: SETUP Statement Processing Routine

If the job is currently routed to be run at this node, the volume to be mounted is listed on the operator's console, and the job is placed in hold status. Otherwise, the SETUP statement is ignored.

RMSGCARD: MESSAGE Statement Processing Routine

If the job is currently routed to be run at this node, leading and trailing blanks are removed from the message contained on the MESSAGE statement, and the message is routed to operator's console. Otherwise, the MESSAGE statement is ignored.

RJBPCARD: JOBPARM Statement Processing Routine

If the job is currently routed to be run at this node, the RKEYSCAN subroutine is called to scan the parameters coded on the JOBPARM statement and set the appropriate values in the job control table (JCT). If the line count has been specified, it is merged into the job header. A line count of 0 is put into the header as X'FF', while no specification of line count is given a 0 default value. If the restart option has been specified, flags are set in the JCT to allow or disallow job restart as required. If system affinity has been specified, the indicated system names are converted to a binary value for insertion into the job queue element (JQE). If parameters on the JOBPARM statement are found to be in error, RJBPCARD issues the $HASP112 message ("--INVALID /*JOBPARM card"). The JOBPARM statement is ignored if the job is not currently routed to be run at this node.

ROPTCARD: OUTPUT Statement Processing Routine

If the /*OUTPUT statement is encountered before a JOB statement, control is immediately returned to the main processor with an indication that the RNOTCCRD routine should be executed. If a JCL file is not being read, the /*OUTPUT statement is considered to be the beginning of such a file; the JCL file indicator is set, and ROPTCARD again calls RPUT to terminate the output file that was being created. If the job is currently routed to be run at this node, ROPSCAN scans the /*OUTPUT statement for the forms code. A forms code of an asterisk (*) indicates a continuation of the preceding /*OUTPUT statement. The contents of the previous output control record (OCR) are read into the scan work area and the RKEYSCAN routine is called; values specified on the current /*OUTPUT statement replace (or, in the case of destination, continue) those specified or omitted on the previous /*OUTPUT statement. If a forms code of an asterisk is specified on the first output statement, the continuation is rejected and RILLOUPT is called to issue the HASP113 message, "INVALID /*OUTPUT CARD". The job is then terminated and the output stream flushed. If the forms code is not an asterisk, the scan work area is cleared before RKEYSCAN is called to scan the remaining /*OUTPUT statement parameters.

Provided that no specification errors are found by the scanning subroutine, ROPBURST sets the burst indicator to the appropriate value if BURST=Y or BURST=N was specified. Next, any specified right or left indexing value is converted to the appropriate binary code and moved into the job control table (JCT). Destination codes are scanned, converted to binary routing codes, and stored in the the JCT. Finally, ROPTCARD attempts to move the contents of the scan work area to the output control table (OCT) as an output control record. If not enough space to accommodate this OCR remains in the OCT, $EXCP is issued to write the OCT to a spool volume, $GETBUF WAIT=YES is issued to obtain another OCT buffer, and the current OCR is placed in the new OCT.

When input processing for the current job is complete, RJOBTERM is entered. This routine issues the $EXCP macro instruction to write any current OCT to the spool volume.

The /*OUTPUT statement is ignored if the job is not currently routed to be run at this node.

RXEQCARD: XEQ Statement Processing Routine

If the device is not a network job reader, the execution routing byte is set to the node number associated with the destination specified. If the destination is invalid, an error message is issued to both the operator and the programmer, and further job processing is bypassed. The XEQ statement is ignored if this device is a network job reader.

RXMTCARD: /*XMIT Control Statement Processing Routine

RXMTCARD processes the /*XMIT JES2 control statement. First RXMTCARD sets the QUEXMIT flag on in the JQE to indicate that this is a /*XMIT job, and schedules a checkpoint. Then RXMTCARD sets the execution routing bit to 1 in the JCT according to the destination specified on the /*XMIT statement. RXMTCARD then calls HASPRDDS to process the delimiter specification (DLM=). If an invalid delimiter was specified, HASPRDDS sets the delimiter to /*. The /*XMIT statement is invalid if it is not immediately after a valid JOB statement or if the destination is OWNNODE. Only comment statements or /*NETACCT statements are permitted between the JOB and /*XMIT statements. For an invalid /*XMIT statement, RXMTCARD issues the message $HASP117, "--INVALID */XMIT CARD.".

RNETACRD: NETACCT Control Statement Processing Routine

The RNETACRD routine processes /*NETACCT control statements specified in a job that is read from a local device (local card reader, remote reader, or internal reader). /*NETACCT statements are ignored by the NJE job receiver; the job header specifies the network account number to be used for jobs read by the job receiver.

The network account number specified on the /*NETACCT statement is saved in the job header. At the end of job processing, HASPRDR checks both the JES2 account number in the job control table (JCT) and the network account number in the job header. If both are present or both are absent, processing continues. If only one is present, the conversion routines in HASPACCT are called to convert the supplied account number (JES2 or network) to the type of account number that was not supplied. Thus, only one account number need be specified on a job, assuming that the NETACCT conversion tables are set up properly. For an invalid /*NETACCT statement, RNETACRD issues message $HASP115, "--INVALID /*NETACCT CARD."

RNFYCARD: NOTIFY Control Statement Processing

The user ID specified on the /*NOTIFY statement is placed in the JCT and in the NJE job header just as if the user ID had been specified on the job statement (NOTIFY=user ID).

If the /*NOTIFY statement specified both a node name and a user ID, the specified node name is placed in the NJE job header as the origin node name.

For an invalid /*NOTIFY statement, RNFYCARD issues message $HASP116, "--INVALID /*NOTIFY CARD."

RROUTCNV: Route Code Conversion Routine

The RROUTCNV routine accepts a node remote number and a pointer to an output area that contains the remote name and number. If the remote name is not zero, RROUTCNV converts it to EBCDIC and stores it in the remote name portion of the output area. If the remote name is zero processing continues. Using the node number as an index into the node information table (NIT), RROUTCNV moves the node name for this node into the output area and returns to the caller. (If the node number is zero, $OWNNODE is stored in the node name portion of the output area.)

HASPRDR Subroutines

The following describes the subroutines used in HASPRDR.

RDESTSCN: Convert Destination to Route Code Subroutine

This subroutine is used by the ROUTE, OUTPUT, XEQ and the NOTIFY statement processing routines to prepare an EBCDIC destination to be converted to an equivalent binary code. Input destinations can be of the following forms:

- LOCAL: General local device
- RMTnnn,RMnnn,Rnnn: General remote work station at the local node
- NnRn: Specific node and remote work station (*)
- Un: Special local device (n is from 1 to 255)
- Nn: Specific nodes general local device (*)
- xxxxxxxx: Defined name of device as an 8-byte EBCDIC character
- xxxxxxxx.yyyyyyyy: Two-part specification of both node and user ID.remote ID

(*) - n is from 1 to 1000

If the destination is not one of those listed above or if no match is found in the remote destination table (RDT), control is returned to the caller with a zero condition code. If a match is found (the destination is valid), control is returned with a non-zero condition code.

RILLCCRD: Invalid Control Statement Placement Exit Subroutine

RILLCCRD loads register 6 with the address of the RNOTCCRD routine and returns to the main processor, which always branches to the exit specified by register 6; this allows the statement to be a non-JECL statement.

RILLCCRD is entered because a control statement is unrecognizable or is placed incorrectly with respect to the JOB statement. The subroutine is entered from the following routines under these circumstances:

- HASPRCCS: Unrecognizable control statement that does not match an entry in the routing table
- RCOMCARD: A JES2 COMMAND statement found within a job
- RPRICARD: CONTROL statement is within a network job
- ROUTCARD: ROUTE statement not within a job
- RSETCARD: SETUP statement not within a job
- RJBPCARD: JOBPARM statement not within a job
- ROTPCARD: /*OUTPUT statement not within a job
- RXEQCARD: XEQ statement not within a job
- RXMTCARD: XMIT statement not with a job
- RNETACRD: NETACCT statement not within a job
- RNFYCARD: NOTIFY statement not within a job

HASPRDDS: DD * and DD DATA Statement Scan Subroutine

The HASPRDDS subroutine, is called whenever the main processor phase encounters a DD * or DD DATA statement. HASPRDDS is also called by RXMTCARD to process the DLM parameter on the /*XMIT statement. The statement is first scanned for a DLM parameter, and, if found, the value specified is saved in RDRDLM for later use by the main processor phase. If an error is encountered, the parameter is ignored, setting the delimiter to slash asterisk (/*). The error will be detected later by the MVS converter and appropriate error messages will be written to the system message file. When the entire DD statement has been scanned and written to the JCL file, a control record is generated containing the track address, record format, and logical record length; this control record is also added to the JCL file in such a way that it is not passed to the converter, but is available to the job transmitter for reference in reestablishing input data set characteristics.

Next, the current input/output table (IOT) is located and a peripheral data definition block (PDDB) is built reflecting the location and characteristics of the input data set. If space is not available in the current IOT, a new IOT is created and added to the IOT chain. Finally, the input data set buffer is initialized and control is returned to the main processor phase.

HASPRJCS: JOB Statement Scan Subroutine

The HASPRJCS subroutine is called whenever the main processor phase encounters a JOB statement or a job header record. The previous job (if any) is terminated by calling the RJOBEND subroutine.

During the scan of the JOB statement, the scan routine examines the job's CLASS parameter. If the CLASS associated with the job is defined as an execution batch job (XBM= on JOBCLASS initialization statement), a SYSIN data set is opened and the JOB statement is placed into the SYSIN data set. Then, the procedure named on the XBM= parameter is found and placed in the SYSIN data set followed by the input stream that originally followed the JOB statement.

Following each call to RJOBEND, HASPRJCS resets the RDW4JVFY, RDW4STKN, and RDW4SREQ flags. When an invalid JOB card is found, the RDW4SREQ flag is set and propagated to the $SAFINFO parameter list (in RJOBVFY) so that a submitter security token will be returned from JOBVALM. The JCTNUPAS field is cleared prior to calling RJOBVFY to prevent a password change; the RJOBVFY return code is not checked in this case.

Following the RJOBEND call, a buffer is obtained for the job control table (JCT). Subroutine RNJEHDTR is called to fill in the job header section of the JCT with the actual job header, but only for a network job header; otherwise RNJEHDTR places default information in the job header section. Another buffer is obtained for the I/O table (IOT). A job queue element is then created for this job and added to the JES2 job queue in active input status.

This subroutine then tests the bits in the header indicating whether a job was held prior to dumping to an offload data set and sets corresponding bits in the JQE if 'hold' was specified. A check is also made for special local routing fields in the JES2 section of the header and, if they exist, to convert them to route codes that are also moved into the JQE. The JES2 job id is determined by calling $QJIX. Both the IOT and the beginning of the JCT are initialized and a pointer is set to the first free TGAE. If the job is not destined for this node, the routine returns immediately. Otherwise, exit point RXITJBCD (exit 2) is invoked allowing user exit routine(s) to scan the JOB statement initially. Depending on the return code from the user exit routine(s), either the job is aborted or the JOB statement is scanned normally and the first JCL block is initialized. Normal JOB statement scanning includes authorization checking and propagation, accounting field scanning, and keyword analysis.

If this is an offload job, verification checking is done here both when reading in the job header and at job termination with the job trailer (routine HASPRTRM). The time and date are compared with the time and date in the offload DCT (taken from the record descriptor). If there is a match, processing continues. If there is a difference, the job in error is purged, the load operation terminated, and the offload device drained. If a job or SYSOUT have been dumped from a previous release of JES2, there is no verification stamp. SYSOUT receivers detect this condition and pad the header with binary zeroes so that the header and the record descriptor match and processing continues.

For a user-submitted job where the JES2 initialization parameter &RJOBOPT indicates that an accounting field scan is to be performed and the JOB statement has been successfully scanned, exit point RXITACC (exit 3) is invoked allowing user exit routine(s) to scan the JOB statement accounting field and/or alter the JCT fields following the JOB statement scan. Depending on the return code from the user exit routine(s), the job is either aborted and HASPRJCS returns immediately, or control is passed to HASPRSCN for accounting field interpretation. If ACCTFLD on the JOBDEF statement indicates that the accounting field is not to be scanned, or the job is a started-task control (STC) or time-sharing user (TSU) job, control is returned to the main processor phase.

If the JOB statement and its continuations require more than one buffer, the additional buffer must be obtained when the first buffer becomes full. Because JOB statement processing is not yet complete, the mask of eligible spool volumes might not yet be set. (When the mask is set depends on when the user exit routine sets it.) Therefore, the full buffer might not be written to the spool to which the user wants it written. The full buffer would only be written to the correct spool if the user exit routine set the eligible spool volumes mask prior to the full buffer being written to spool.

After HASPRJCS completes the JOB statement processing, it calls RWRTJOB to obtain track addresses for the JCT, IOT, and first JCL block. RWRTJOB then writes the JCT and IOT to the spool. These actions are delayed until after JOB statement scanning so that the mask of eligible spool volumes can be set by a user exit routine at the JOB statement scan exit point 2 after it examines the JOB statement.

RNXTSLT: Subroutine to Get Next Available PDDB Slot

The RNXTSLT subroutine is used by HASPRJCS during JOB statement scan to acquire a new PDDB slot to be associated with the IOT initialized as a result of the JOB statement scan. After acquiring a PDDB slot and using subroutine RFMTIOT to get the next IOT, RNXTSLT returns to its caller.

RJSCAN: Subroutine to Extract Fields from JOB Statement

The RJSCAN subroutine scans the JOB statement and extracts fields for further processing. The fields may be split between several statements (in accordance with MVS JCL standards) and may be enclosed in either parentheses or quotation marks. This routine is used to scan the accounting field and the programmer name from the JOB statements, which are processed by the input service processor. Exit point RXITJBCD (exit 2) is invoked for each JOB continuation statement detected.

HASPRSCN: Accounting Field Subroutine

The HASPRSCN subroutine is called whenever a user-submitted JOB statement is successfully processed to interpret any variables present in the JOB statement accounting field and to set the appropriate fields in JES2 control blocks representing those variables. The contents of the accounting field are scanned from the JOB statement by the HASPRJCS subroutine and left in the job control table (field JCTWORK) as input to this subroutine. This routine is not called for JOB statements associated with started task control (STC) or time-sharing user (TSU) jobs. Depending on the value of the JES2 initialization parameter ACCTFLD, HASPRSCN may or may not be called and may or may not enforce certain JES2 and/or MVS JOB statement standards during the scan. Upon completion of the scan, control is returned to the main processor phase via the $RETURN macro instruction.

RCONTNUE: Subroutine to Read and Validate Continuation Statements

The RCONTNUE subroutine reads and validates JCL continuation statements by ensuring that columns 1 and 2 have slashes and that column 3 is blank. RCONTNUE contains exit point RXITCCC (exit 4), which is invoked for each JCL continuation statement detected. The start of the continuation statement is located, and control is returned to the calling routine. If an invalid continuation statement is discovered, control is returned to the location specified by register 15. If no error is encountered, control is returned to 4 bytes beyond the location specified by register 15. RCONTNUE contains exit point RXITJBCC (for exit 2), which is used for each JOB statement continuation that is detected.

RNJEHDTR: Move Header Information into JCL Subroutine

The RNJEHDTR subroutine is called by the HASPRJCS subroutine and by the termination phase HASPRDRT. These callers are responsible for storing job header and job trailer records, respectively, in the job control table (JCT). RNJEHDTR determines what information needs to be in this area of the JCT and moves the information accordingly.

If the job is being received from the network or offload processing, the header and trailer records are read in with the job and put in the JCT. Because a job received from a local or remote reader does not include a header or trailer, default records are stored in the JCT.

RJOBEND: Subroutine to Complete Job Input Processing

The RJOBEND subroutine tests whether the input processor is currently processing a job and, if it is not, returns control immediately.

If a job is being processed, RJOBEND calls RJCTTERM to complete processing of the job control table (JCT). If RJCTTERM provides a non-zero return code, the job is queued for either output or purge processing, depending on the contents of the RDW1ABRT flag set by RJOBVFY. Otherwise, processing continues for non-reload jobs with exit 20 and queueing for the next job phase. Because RJCTTERM does not call RJOBVFY for selective reload jobs, processing for these types of jobs continues as follows.

RJOBEND calls $CFMOD to determine whether the job's execution node has been changed. RJOBEND then passes this information on a call to RJOBVFY (which passes it to JOBVALM). If RJOBVFY returns a value of 4 in register 15 (indicating the job is either uninitialized or unauthorized), the JQE will **not** be marked non-selectable, if determined non-selectable during subsequent WSSERV processing, to preserve auditability during purge processing.

RJOBEND then invokes work selection (WSSERV) processing, which determines whether or not the JQE is selectable; if non-selectable, the JCT buffer is released. Otherwise, a sign-on message is issued, any new route codes are stored in the JCT and JQE, an SMF type 24 record is formatted, new information from the SPOF section of the header is obtained, and the JQE is checkpointed.

RJOBEND then invokes exit 20. If the JQE cannot be selected by the offload receiver, it is marked non-selectable and checkpointed. ("Non-selectable" status prevents any further processing; no SAF call is made for non-selectable jobs during purge processing.) The output buffer, OCT, JCT, and IOT are written and control is returned to the RJOBEND caller.

RGET: Subroutine to Get Next Record

The RGET subroutine returns the address of the next record to be processed by the input service processor in register R4. RGET also returns the length and characteristics of that record in the RCARDLRL and RINFLAG1 fields of the processor control element (PCE) work area of the input processor.

If a user exit routine indicated (via flag byte RXITFLAG in the PCE) that it is supplying the next card image to be read as part of the input stream, the card image contained in the exit work area of the JCT (JCTXWRK) is moved to the input buffer overlaying the previous card image. RGET then returns.

If the input device is a card reader and if there are unprocessed cards in the input buffer, the input register (R4) is set to the address of the next card and processing continues. If the input buffer is empty or if all the cards in the input buffer have been processed, a read ($EXCP) is issued to the input device to refill the input buffer, and the subroutine places the processor in a wait state ($WAIT) until the input buffer has been filled. The input register is then set to the address of the first card in the buffer, and processing continues. If a permanent error is detected on the card reader, no action is taken until after the last card has been processed. Then, the job currently being processed is deleted with an appropriate message to the operator. Processing then continues by scanning the input stream for the next JOB statement.

If the input device is an internal reader and there are unprocessed records in the internal reader buffer, the next record is expanded and processing continues. If all the records in the input buffer have been processed and the end-of-file indication has been set, control is returned to the caller. If the internal reader buffer is empty and the address space that has the internal reader allocated is not in address space termination, RGET issues a $XMPOST to HASPAM. RINRW1 places the processor in a wait state ($WAIT) until the internal reader input buffer has been refilled. When the input buffer has been refilled, JES2 is posted, the input service processor is posted, and the first record in the buffer is expanded.

If the input device is a remote terminal or network job reader, the remote terminal access method (RTAM) is invoked (via $EXTPGET) to obtain the next card image. An unsuccessful return code from RTAM causes a network job to be purged and a negative close error indication to be sent to the originating node. For network jobs only where a successful non-end-of-file return occurs, RJEGNOCC determines if the record received is a data set header. Information from a data set header is stored in the current peripheral data definition block (PDDB) before the next record is retrieved.

The RGET subroutine also processes the operator commands $Z input device and $C input device by entering the wait state ($WAIT) for the stop command and calling the subroutine RJOBKILL to delete the job if the cancel command is received. If a network device is to be cancelled, the subroutine HASPRDRT places the job in the output queue and issues a normal close ($EXTP CLOSE) against the job.

RPUT: Subroutine to Add Card to Output Buffer

The RPUT subroutine accepts card images, removes trailing blanks, and blocks card images into standard HASP data blocks (HDBs). The length of the input card image is defined by the value in the RCARDLRL field of the processor control element (PCE) work area of the input processor. If the RINFLAG1 or RCARDLRL bytes indicate carriage control, the carriage control is added from the RDRCCTL field. If the current output buffer is full, it is truncated and scheduled for output. A new buffer is acquired and used as the next output buffer. If no output buffer exists upon entry, the processor is skipping for a JOB statement, and the subroutine returns without taking any action.

RWRTJOB: Subroutine to Write the JCT and IOT to Spool

The RWRTJOB subroutine is called by HASPRJCS to write the JCT and IOT to spool. RWRTJOB first obtains spool buffers for the JCT, IOT, and first JCL block. RWRTJOB then sets the JCT to point to the IOT, and the IOT to point to the first JCL block. Finally, RWRTJOB writes the JCT and IOT to spool.

RKEYSCAN: Subroutine to Scan and Process Keyword Values

The RKEYSCAN subroutine accepts a pointer to a parameter field and an address of a control statement scan table (keyword table). RKEYSCAN returns converted values in the caller-provided output table. If an error is encountered, control is returned to the location specified by register 15. If no error is encountered, control is returned to 4 bytes beyond the location specified in register 15. The control statement scan table specifies such characteristics as keyword name, keyword abbreviation, whether the value should be converted to binary, whether the value should be left-or-right adjusted in the field, whether the value should be filled with blanks or zeros, and the maximum number of values permitted. The RDESTSCN subroutine is called to interpret the destination specifications. RDESTSCN is used by both the JOBPARM and /*OUTPUT control statement routines to perform their respective scans.

RJOBKILL: Subroutine to Delete Current Job

The RJOBKILL subroutine tests whether the input processor is currently processing a job and, if it is not, returns control immediately. If a job is being processed, the operator is notified that the job is being deleted; the RJCTTERM and RJOBTERM subroutines are called to terminate the input processing of the job.

If a job has not been verified yet, then RJCTTERM invokes RJOBVFY, which, in turn, invokes JOBVALM. If JOBVALM returns an 8 return code (indicating a severe error), RJOBVFY sets the RDW1ABRT flag to indicate to RJOBKILL, following the call to RJCTTERM, that the job is to be queued for purge processing.

If the job is a network job and it is being deleted because of a line error, the job is placed in the purge queue; otherwise, the job being deleted is placed in the output queue. Control is returned to the calling routine.

RJCTTERM: Subroutine to Terminate Job Control Table

The RJCTTERM subroutine performs the final update of the job control (JCT) table. The time estimate is converted from minutes to seconds; the estimated line count is converted from thousands of lines to actual lines, estimated byte count from thousands of bytes to actual and the total output estimate is computed and set. RJCTTERM then moves in NJE fields for the SMF origin node and execution node. Once the security token, if any, is obtained from the job header, RJCTTERM calls RJOBVFY for non-spool offload devices (for reload devices, RJOBVFY is called after selective reload). RJOBVFY obtains a token (or verifies the one that was passed) to be used to protect jobs that will execute locally or that will be transmitted.

RJOBVFY is not called if a job has already been validated locally. RJCTTERM checks the RDW4JVFY flag, set at the end of RJOBVFY processing, to determine this condition. Before returning to the calling routine, RJCTTERM sets the RDW4STKN flag (based on NJHTF0JB) to indicate whether the token obtained represents the job itself or the job's submittor.

RJOBTERM: Subroutine to Terminate Job

The RJOBTERM subroutine terminates the last input stream data set if it was not already terminated. A message, 'JOB DELETED BY JES2 OR CANCELLED BY THE OPERATOR BEFORE EXECUTION', is added to the end of the JCL file only if the job is cancelled before it is processed by the JCL conversion processor. The JCL file is then terminated, and the last buffer is scheduled for output.

The peripheral data definition block (PDDB) for each of the four system data sets with a potential for output — JCL, JCL images, JOBLOG, and system messages — is updated with information such as job forms and SYSOUT classes. If the SYSOUT class is one that was specified with the TRKCEL attribute during JES2 initialization, the PDB2TCEL indicator in the PDDB is set for later processing by the JES2 access methods.

If any /*OUTPUT statements were encountered during the processing of the job, the output control table (OCT) is written to disk. The job queue element (JQE) is updated to reflect any changes in routing that might have occurred during the processing of the job.

For network jobs, the job control table (JCT) and the job header are merged and placed in the job header. A mapped version of the job's security token is also placed into the header for jobs originating or executing locally. The RDW4STKN flag (set by RJOBVFY) is checked to determine if the token represents the job itself or the job's submittor. If it is the job's token, RJOBTERM sets the NJHTF0JB flag in the header as well.

If TYPRUN=JCLHOLD was specified, the HOLD immediate indication is set. The JCT is updated to reflect end-of-file status. If the job is a TSO session and is transmitting data using the TSO TRANSMIT command, the JCT is updated with the transmitter's job name.

If the job is an NJE job, message $HASP122 is issued. If the job is a spool offload job, message $HASP588 is issued. The JCT and all input/output tables (IOTs) are written to disk. Control is then returned to the calling program.

RJOBVFY: Job Verification Subtasking Routine

RJOBVFY sets up the SAFINFO parameter list and calls JOBVALM under a subtask (via $SUBIT) to perform job validation. It then processes the job based on the information returned from the SAF call and calls RPDBSEC to initialize the system PDDBs.

RJOBVFY is called by:

- HASPRJCS for request jobid and error processing
- RJCTTERM and RJOBEND for spool reload processing
- RJCTTERM for locally originating/executing jobs and network jobs.

On exit, register 15 contains a return code of 0 (successful processing; a valid token was obtained) or 4 (a severe parameter error occurred; the token is undefined).

If subtask processing fails, RJOBVFY issues the $HASP077 message and issues an unsubtasked RACROUTE SAF call requesting an undefined user token, which will allow the job to be sent to another node or printed locally. RJOBVFY then subtasks RPDBSEC to initialize the JESMSGLG PDDB. If this subtasking fails or if a PDDB initialization error occurs, RJOBVFY sets the RDW1ABRT flag (queue the job for purge processing), clears the RDW2PURG flag (don't queue the job for output processing), and sets a return code of 4 to indicate to its caller that the job should be failed.

If JOBVALM subtasking is successful, RJOBVFY continues processing according to JOBVALM return code. Return values of 0 (successful) or 4 (validation error) allow the job to continue and RJOBVFY sets its own return code to 0. A return code of 8 (severe parameter error) causes RJOBVFY to bypass the remainder of job initialization and issue a $ERROR IP2 to terminate in the main task.

Notes:

1. For the 0 and 4 return codes from JOBVALM, RJOBVFY subtasks RPDBSEC, as previously described.
2. For return code 4, RJOBVFY checks the accompanying reason code (SFIRESCD field) as well. If the reason code is 4 or 8, the job is flushed with a JCL error and the appropriate message is placed in the JESJCL data set:
   - 4 - $HASP111 INVALID /*ROUTE CARD
   - 8 - $HASP114 INVALID EXECUTION NODE

When the job is authorized (return code of 0 from JOBVALM), the USERID, GROUPID, and SECLABEL values are extracted from the token and saved in the JQE and JCT; the verified USERID and GROUPID are copied into the NJE header fields if the job has been verified (RJOBVFY checks the setting of the JCT2AVDP field). RJOBVFY sets a field to indicate the token type (submittor's or job's) for later setting in the NJHTF0JB field of the NJE header. The password will also be changed if a new password has been specified. RJOBVFY checkpoints the JQE and subtasks the call to RPDBSEC to initialize the system PDDBs. The return codes from RPDBSEC are processed as previously described.

RJOBVFY sets its return code according to the higher of the return codes set by JOBVALM and RPDBSEC. Just prior to returning to its caller, RJOBVFY sets the RDW4JVFY field to indicate that validation processing has been performed.

RTRACK: Subroutine to Acquire a Track Address

The RTRACK subroutine acquires a JES2 track address and returns with the track address in register 1. The $TRACK routine is called, passing the address of the master track allocation block (TAB) in the job's allocation input/output table (IOT) as input. If the acquisition of the track address causes a new track group to be allocated (indicated by the condition code returned by the $TRACK routine), then the allocation IOT is written to disk in order to allow the track group to be recovered in the case of a subsequent system failure.

RFMTIOT: Subroutine to Format Secondary IOTs

The RFMTIOT subroutine acquires buffers (via $GETBUF) for a new IOT and initializes the new IOT using the old IOT, the last IOT on the chain. RFMTIOT then ensures that the new IOT is the first one pointed to by the PDDB. RFMTIOT then returns to the caller.

RTRTIME: Convert S/370 TOD Subroutine

RTRTIME converts System/370 time-of-day (TOD), obtained from the STCK instruction, to TOD in 0.01 second units with the date. RTRTIME then returns to the caller.

RGETBUF: Subroutine to Initialize Output Buffers

The RGETBUF subroutine acquires a JES2 buffer for an output buffer, initializes this buffer with a chain track address, the job key, and the data set key, then returns with the address of the buffer in register 1. If no buffer is available, the processor is placed in a wait state ($WAIT). When a buffer becomes available, the processor is posted, and another attempt is made to acquire a buffer. This process continues until a buffer is available, whereupon it is initialized as described above.

RDRRCV0: Processor Recovery Subroutine

The RDRRCV0 routine performs the error recovery processing for the HASPRDR module. The routine is invoked via the $ESTAE error recovery mechanism in the event of a system abend condition occurring in HASPRDR. The routine first checks the SDWA for the type of error that occurred. If the error is not a program check, recovery is ended (RDRRCV0 issues the $SETRP PERCOLATE macro). If a program check error occurred, RDRRCVO attempts to retry the processor by deleting the job currently being read in and continuing with the next job in the input stream. RDRRCV0 sets the DCTDELET flag on in the reader DCT and resumes processing in the RGET subroutine via the $SETRP macro. The $HASP128 message is issued to the operator indicating the recovery action.

HASPCNVT: JCL Conversion Processor

When first dispatched after JES2 has been started, HASPCNVT performs some basic initialization, establishes an $ESTAE routine, and issues a $GETMAIN macro instruction to obtain permanent storage for two buffers: one to hold a job control table (JCT) and the other to hold an input/output table (IOT). After having obtained the required storage and stored the addresses of the two buffers in its processor control element (PCE), HASPCNVT branches to the XCJGET routine to process the first available job.

The PCE work area resides above the 16 megabytes in virtual storage and is accessed by the conversion processor in 31-bit addressing mode. Information needed by routines executing in 24-bit mode is contained in the DTE work area, which resides below 16 megabytes in virtual storage.

If the $GETMAIN macro is unsuccessful, the HASPCNVT issues a $ERROR for a catastrophic error message $X03; this results in an abend.

Multiple converters are supported. Up to 10 conversion processors can be initialized.

XCJWAIT: Main Job Wait Routine

When HASPCNVT has finished processing and either the current job or no jobs remain to be processed, HASPCNVT waits ($WAIT CNVT) at XCJWAIT.

XCJGET: Start Job Processing Routine

This portion of the converter is entered at the completion of processor initialization, or whenever HASPCNVT is waiting and a job is posted to the queue. The $QGET macro instruction is issued to obtain a job that is ready for JCL conversion. If no such job is available, XCJGET issues the $PGSRVC macro instruction to release (but not free) the storage that had been set aside for the expected job's JCT and the IOT; XCJGET then indicates that the storage has been released and again waits for a job. (The page release step is bypassed if the storage has never been used.)

If a job is available, HASPCNVT issues the $ACTIVE macro instruction and determines whether the storage required for job processing has been released. If so, XCJGET issues the $BFRBLD macro instruction twice, to create an input/output block (IOB) within the JCT buffer and an IOB within the IOT buffer.

XCRDJCT: JCT Read Routine

After IOBs have been created, if the buffers had not been released previously, the XCRDJCT routine initializes the necessary fields, then calls the $JCTIOR routine, which reads the job's JCT from a spool volume. If the JCT does not point back to the job queue element (JQE) through which it was obtained, XCRDJCT branches to the disastrous error routine, XCDISTR, to abort the job. Normally, however, XCRDJCT proceeds to update the job management record in the JCT. The TIME macro instruction is issued to obtain the current time and date, which are stored in the JCT along with the system identification.

HASPCNVT then determines if the conversion subtask (the HOSCNVT routine in HASPCNVS) is attached. If not, it is first detached (if necessary) and then attached. HOSCNVT establishes a security environment for the job being converted and performs a link and call to the OS JCL converter. HASPCNVS also contains the Converter/Interpreter text.

Then, HASPCNVT waits (because of the WAIT parameter on the $DTEDYN issued in this routine) for the subtask to finish its initialization. That is, the subtask posts JES2 using the $$POST macro instruction, which causes the JES2 dispatcher to post ($POST) the conversion processor.

XCRDIOT: IOT Read/Create Routine

Once HOSCNVT has been attached, HASPCNVT reads the first IOT into the buffer obtained during processor initialization. Any additional IOTs are then read into buffers obtained from the JES2 buffer pool.

Next, new spool space for the IOT is allocated out of its track group map. Then, an interpreter entry list (NEL), partially created during HOSCNVT initialization, is completed using information in the JCT and job class-related information from the class attribute table (CAT). This completes pre-conversion preparations by HASPCNVT.

XCPOST: During these preparations, the subtask HOSCNVT has been waiting on the event control block (ECB) located at $CNVECB in the HASP communications table (HCT). HASPCNVT now posts $CNVECB and then waits on OPER and the work ECB in the DTE (DTEWECB) When HOSCNVT has completed JCL conversion, it posts the JES2 task, causing HASPCNVT to be dispatched again. HOSCNVT then waits again on $CNVECB.

If HOSCNVT is not successful in its initialization, it issues a $HASP305 message to alert the operator that it is awaiting storage. Then HASPCNVT sets a time interval (via $STIMER) and waits for the time interval to elapse. When the time interval is up, HASPCNVT again posts HOSCNVT to initialize itself for processing. This cycle of message $HASP305, set timer ($STIMER), and wait continues till HOSCNVT is initialized.

XCCHKPT: Checkpoint Processing Routine

When control returns to HASPCNVT after JES2 converts the JCL, HASPCNVT writes every IOT that is marked as needing checkpointing to spool. Using $TRACK, HASPCNVT acquires a new track for each changed IOT and stores the track address in each IOT. After all IOTs are processed, HASPCNVT gets a new track address on the spool for the output JCT.

HASPCNVT then writes all the changed IOTs and the JCT to spool, waiting (via $WAIT) for the I/O to complete. When the I/O is complete HASPCNVT validates that the JCT and IOT writes were successful; if they were unsuccessful, HASPCNVT issues a $DISTERR macro to indicate a control block error, stores the error code in the JCT, frees the IOT buffers, and queues the job again according to the results of the conversion and the information in the JCT.

If the converted job represents a time-sharing user (TSU) logon, a test is made to determine whether another logon job with the same job name is already queued for or in execution anywhere among the systems sharing the JES2 job queue. If the logon job name is not unique, the JCT is rewritten with this indication of not being unique. HASPCNVT then frees any IOT buffers obtained from the JES2 buffer pool and requeues the job according to the results of the conversion and the information in the JCT.

If the job specified TYPRUN=SCAN, then it is queued, at highest priority, for output processing. If the job is a started task (STC) or a time-sharing user (TSU) logon, then the job is queued for execution processing, regardless of the results of JCL conversion. Otherwise, jobs that did not complete JCL conversion processing without error are queued, at highest priority, for output processing. JES2 assumes the job encountered an error in conversion processing if either the converter set a non-zero return code or a user exit routine associated with JES2 exit point 6 set a return code greater than 4. If TYPRUN=HOLD was specified or implied (via operator commands or initialization parameters) for the job, and the job converted normally, then the job is queued for execution and held, and message $HASP101 is issued.

Once the job has been requeued, HASPCNVT tries to get another job. If it does not get another, the permanent buffers obtained during processor initialization are page-released. They are reformatted when another job is obtained.

CVESTAE: JCL Conversion $ESTAE Routine

CVESTAE receives control when HASPCNVT abends. It analyzes if recovery is possible and, if so, causes JES2 to recover the main task conversion processor. If the processor is processing a job and if a resumption point is defined, CVESTAE issues a $SETRP macro to resume at that point. Otherwise, the $SETRP macro is issued indicating percolation.

HASPCNVS: JCL Conversion Processor Subtask

HASPCNVS contains a single entry point, HOSCNVT, which is subtasked by HASPCNVT to perform JCL conversion processing. HASPCNVS establishes a security environment for the job being converted and issues a link and call to the OS JCL converter. The module is reentrant and has RMODE 24, and AMODE 31 attributes.

HOSCNVT: JCL Conversion Processor Subtask

When it is attached by HASPCNVT, HOSCNVT establishes $STABNDA as its ESTAE routine and initializes an interpreter entry list (NEL) that is located in the daughter task element (DTE).

To begin JCL conversion, HOSCNVT completes the initialization processing by loading the MVS JCL converter (entry point IEFVH1), posting ($$POST) the JES2 task, and waiting on $CNVECB. This is the WAIT macro instruction, located at XCNVWT, that HOSCNVT issues after each job's JCL has been converted.

After being posted by HASPCNVT, HOSCNVT checks to see if JES2 is being terminated. If it is, HOSCNVT closes the SYS1.PROCLIB, deletes (via DELETE) the MVS JCL converter, and exits (via SVC 3) to the system. If JES2 is not terminating, HOSCNVT gets the address of the JCT, the first IOT, and the PDDB for use in subsequent processing.

The SSVOPNC routine in the subsystem support module is used to perform a pseudo open of the subsystem data sets required by the MVS JCL converter. The access method control blocks (ACBs) and data extent blocks (DEBs) for these data sets are located in the DTE. SSVOPNC creates the appropriate subsystem data blocks (SDBs), obtains buffers for the data blocks, and reads in the first JCL data block. Then, it returns to HOSCNVT.

After the fake opening, HOSCNVT checks the DDNAME of the procedure library required for the job. If the DDNAME is not the same as for the procedure library currently open, HOSCNVT closes the current library and opens the one required for this job. If the opening is unsuccessful and the DDNAME was other than the default, PROC00, the open is tried again using PROC00. If the procedure library was opened successfully but its DCBLRECL field was not 80 or its DCBBLKSI field was not a multiple of 80, then a message is issued to the operator and a copy of the message is sent to the job's system message data set.

Failure to open a procedure library with valid DCBLRECL and DCBBLKSI fields causes HOSCNVT to zero the address in the NEL pointing to the device control block (DCB) for the procedure library.

HOSCNVT now establishes the security environment for the job currently converting so that, if any MVS commands are passed from the job stream to the converter, the correct authority will be associated with each. If the authority checking (performed by RACROUTE processing) fails, further conversion processing is bypassed and the job is failed. Otherwise, HOSCNVT calls the MVS converter.

The MVS converter merges JCL statements from the JCL data set with JCL statements from the procedure library. The resulting statements are converted into internal text. As each internal text record is created, the converter calls the routine in HASPCNVS, located at XTXTEXIT. This exit routine modifies the internal text for subsystem data set DD statements and for certain EXEC statements. This exit also allows messages to be passed to the converter/interpreter from the first point of invocation only.

For SYSIN and SYSOUT statements, a dsname keyword sequence is added. If the dsname keyword exists, it is appended to the dsname keyword sequence. Otherwise, a "?" is put at the end. (For SYSIN statements, a SAF call is made to audit the fact that the job owner is creating the SYSIN data set.)

If the SYSOUT class is asterisk (*), the class is changed to the class used for system messages (JCTMCLAS). For DD * or DATA statements, the * or DATA sequence is removed, and keyword sequences are added for dsname and card count. For EXEC PGM= statements, if the class attribute table (CAT) entry for this job indicates a performance group of other than zero and if no performance group was specified on the EXEC statement, the keyword sequence for PERFORM is added using the value in the CAT.

After modification of the internal text has taken place, XTXTEXIT invokes any enabled installation exit routines associated with JES2 exit point XCSTCUET (for exit 6). If no installation exit routines are to be invoked (as determined by a $EXIT macro with TYPE=TEST specified), XTXTEXIT returns to the converter at label XCSTCEND. If any installation exit routines are to be invoked, then XTXTEXT produces the exit point trace records (id 13) with internal text rather than accepting the default trace records produced by the exit effector. The trace records produced are identical to those produced by the exit effectors except that they include the internal text. The post invocation record will not contain a valid last-routine-called field.

On entry to an installation exit routine, register 1 contains the address of a 4-word parameter list. The first word is the address of a 4-word work area (DCNVUWA) in HOSCNVT's DTE); the second word is the address of the internal text; the third word is the DTE address; the fourth word is the JCT address. Register 0 contains a value of 0 to distinguish this instance of exit point 6 from that which occurs at the conclusion of conversion.

On return from the last installation exit routine, the return code in register 15 is saved in field DCNVERC1 of the DTE if it is higher than any previously-returned return code. The return code does not otherwise affect subsequent processing, but is saved for later inspection by HASPCNVT at label XNOHOLD, where a value greater than 4 will cause the job to be queued for output instead of execution. At label XCSTCEND, XTXTEXIT returns to the converter.

HOSCNVT invokes RACROUTE services again to delete the security environment created for the job. Should an error occur in RACROUTE processing, HASPCNVS issues the $HASP313 message and fails the job. Otherwise, processing continues at exit point XCSTMUEE, where exit 6 is invoked once again.

This instance of exit 6 is distinguished from the internal text variation by a value of 4 in register 0. Register 1 points to a two-word parameter list; the first word is the address of the same work area passed to internal text exit routines; the second word is the address of a full word containing the converter return code; the third word is the address of the DTE; the fourth word is the offset of the JCT address. The return code from the last in exit routine is saved in field DCNVERC2 in the DTE for later inspection by HASPCNVT at label XNOHOLD, where a value greater than 4 will cause the job to be queued for output instead of execution.

HASPXEQ: Execution Services

The following describes the execution services provided by JES2.

HASPEXEC: Execution Processor Description

The execution processor, HASPEXEC, provides job queuing and a variety of services related to the execution of a job. It is a queue-driven processor; that is, the presence of an element on a queue is the indicator to the processor that it has a function to perform. The following queues are serviced by the HASPEXEC:

- Spin queue
- Job-pending-execution-by-class queue
- Jobs-pending-execution-by-number queue
- Jobs-pending-requeue queue
- Jobs-pending-termination queue

Note: The HASPSTAC module services the cancel/status queue.

The headers of the queues serviced by HASPEXEC are located in the $HCCT. When all the queues have been emptied or when all attempts to empty them have been made, HASPEXEC waits ($WAIT) at location XJBWAIT. HASPEXEC is dispatched again to service the queues when another element is added to one of the queues, when a change is made to any entry in the JES2 job queue, or when the operator modifies a JES2 logical initiator.

Spin Queue Support

Spin queue support, beginning at location XSPNHLD in HASPEXEC, is entered when an element is added to the queue headed by CCTSPIOT. Each element in the queue is an input/output table (IOT) subsystem control block that resides in common storage. The IOT is put on the queue by HASCDSAL during subsystem data set unallocation. HASCDSAL, by this time, has finished its use of the IOT, including writing it, and does not wait for HASPEXEC to complete its spin support. Spinning a data set makes it immediately available for output processing; holding a data set makes it available to conversational terminal system users.

The spin support first dequeues all of the IOTs queued to CCTSPIOT (that is, all pending spin requests) and requeues the IOTs in first-in-first-out (FIFO) sequence to CCTFIFOQ, a queue used exclusively by HASPEXEC's spin support. This reordering is done to facilitate processing in the proper sequence of any hold requests on the queue. CCTFIFOQ is then processed one IOT at a time.

After requesting access to checkpoint data, HASPEXEC at XSPRQST processes the first spin request -- that is, the first IOT chained off of CCTFIFOQ. After checkpointing the JQE associated with the IOT and obtaining the job lock via $GETLOK, HASPEXEC processes the single peripheral data definition block (PDDB) contained in the IOT. If the PDB1HOLD bit in the PDBFLAG1 field is 0, the data set is to be spun, provided that it contains information to be printed (PDB1NSOT in PDBFLAG1 is 0). If it contains information that isn't to be printed, HASPEXEC at XSPHPUR deletes the data set and frees its tracks.

If the data set is to be spun and is not JESNEWS, HASPEXEC at XSPINJOE removes any JOEs pointing to the data set's spin IOT. (These JOEs would exist only in certain error conditions involving an ABEND and hot start of JES2.) They are not removed if marked "busy" and the job will be marked "unspun.". The output processor will remove the JOE later. HASPEXEC then creates work and characteristic JOEs, adds then to the job output table (JOT), allocates a spool record for the JOEs, writes the JOEs, and checkpoints them. Then, HASPEXEC frees the job lock and IOT and gets the next request at XSPRQST. If there is no space in the JOT for the JOE, HASPEXEC at XSPINBAD turns on the unspun flags in the JQE and HCT and posts (via $POST) the output processor that work is available.

If the data set is to be held, indicated by the PDB1HOLD flag set to 1 in the PDBFLAG1 field, HASPEXEC sets the creation date and status byte, initializes the remaining PDDB fields for the held data set, rewrites the IOT, frees the job lock and IOT buffer, and gets the next request at XSPRQST.

General Description of Jobs-Pending Queues Support

The routines described next support the jobs-pending queues. The queue element for each of these queues is the subsystem job block (SJB). An SJB is the primary control block used by HASPSSSM, the subsystem support load module residing in the pageable link pack area (PLPA) or CSA. HASPSSSM, executing under a subsystem service requester's task control block (TCB), queues and dequeues SJBs from the various jobs-pending queues. HASPEXEC, executing under the JES2 TCB, also queues and dequeues SJBs from these queues. To ensure queue and SJB integrity, a locking mechanism is used incorporating the compare double and swap (CDS) instruction.

The SJB-queues lock protects critical SJB fields and all of the jobs-pending and the jobs-in-execution queues as a group. After the lock is obtained, the queues may be modified. The macro instruction $DEQSJB in HASPXEQ is used to obtain the lock and remove an element (SJB) from one of the queues through the local subroutines XSJBLOCK and XDEQSJB. The local subroutines XQUESJB and XSJBUNLK are used to requeue an SJB and release the lock.

To ensure that use of the SJB-queues lock does not significantly degrade system performance, a rule has been established that no SVC be issued, directly or indirectly, by a routine holding the lock. Hence, in processing the queues of SJBs representing requests for jobs pending execution, an attempt is first made (often involving waits on I/O) to satisfy the request. Only when the request is satisfied is the lock obtained, the SJB that is dequeued is updated and requeued, the waiting task (HASPSSSM) posted, and finally the lock is released. Similarly, routines that support job termination and requeue run the preceding sequence using the information extracted from the SJB while the lock was held; they complete the job processing without holding the SJB lock.

Jobs-Pending-Execution-by-Class Queue Support

When an initiator is ready to process a new job, it makes a subsystem request for it using the macro instruction IEFSREQ. This request is first processed by HASCJBST. HASCJBST queues the SJB last-in-first-out (LIFO) off CCTJPCLS in the $HCCT removes the lock, causes the JES2 task to be posted, and then waits for the subsystem to provide a job. The routine used to post JES2, $$POST, also ensures that the execution processor is posted ($POST), causing the processor to be dispatched. The main functions and data areas used in job selection by class are documented in the HIPO "JES2 Execution Processor Job Selection by Job Class" in chapter 2.

An SJB queued to CCTJPCLS causes the routine in HASPXEQ beginning at label XPBYCLS to be entered. Although activated by an SJB representing a job request by an initiator, the flow of this routine is governed by the status of the partition information table (PIT), pointed to by the HCT field, $PITABLE.

The PIT is composed of elements that represent JES2 logical initiators. The system operator controls logical initiators when controlling the scheduling of jobs for execution. Normally there is an MVS initiator active for each non-drained PIT, even though an initiator is not associated with a particular PIT except when a job is being executed.

Beginning at XPBYCLS, the CCTJPCLS queue of SJBs is scanned. Any partially-selected SJBs that were waiting for a checkpoint write that has already completed are dequeued and the waiting task is cross-memory posted. Also any SJB that represents a first-time job select request will be matched-up with an available PIT entry. The PIT is then searched for an available entry, one that is not busy, stopped, or drained (an initiator exists for a stopped PIT, but not for a drained one). When an available PIT is found, the JES2 job queue is searched for a job that is awaiting execution and has a job class matching one of those in the PIT. This search is performed at label XGOTPIT using the $QGET macro instruction. If a job is found and no user job with the same job name is currently executing, the SJB is marked "partially-selected", and the PIT and SJB are updated. If no job is found for the PIT, the operator is sent an idle message (if one has not been already issued), and the scan of the PIT is resumed.

Jobs-Pending-Execution-by-Number Queue Support

START commands, issued by the system or by the system operator, and time-sharing user logons result in JCL being submitted to the subsystem via internal readers. Job selection by number is illustrated in the HIPO "JES2 Execution Processor Job Selection by Job Number" in Chapter 2.

When the input processor (HASPRDR) has created an entry for the job in the JES2 job queue, it returns the JES2 job id of the job to the submitter of the JCL. The submitter of the JCL then makes a subsystem request for this job by its number (job id).

This request is first processed by HASCJBST, which handles the request similarly to the way it handles initiator requests for jobs (see "Jobs-Pending-Execution-by-Class Queue Support" preceding). It differs primarily in that HASCJBST queues the SJB to CCTJPNUM. Another way to queue an SJB to CCTJPNUM is through the request job ID subsystem function, which creates an entry in the JES2 job queue along with sufficient control blocks to represent an executing job. The request job ID function allows a MVS system component to dynamically allocate a SYSOUT data set and, through dynamic deallocation, to spin off that data set for processing by the JES2 output processor.

The presence of an SJB queued to CCTJPNUM causes the routine in HASPXEQ beginning at XPBYNUM to be entered. Control is passed to the routine beginning at XREQID if this is a request for job ID support. If the SJB is not queued to CCTJPNUM, the JES2 queue is searched for the particular job whose number was provided in the SJB. If the job requested is awaiting execution, the SJB is requeued to CCTJXNUM and the waiting task is posted. If the job is in or awaiting JCL conversion, then the execution processor looks for other work to do until JCL conversion is complete and the job is requeued for execution.

Request job ID support invokes the HASPRDR job creation routine (HASPRJCS) to create an entry in the JES2 job queue. It then writes to disk the job control table (JCT) and input/output table (IOT), frees the buffers for these control blocks, requeues the job, requeues the SJB, and posts the waiting task.

Jobs-Pending-Requeue Queue Support

If a job that has terminated abnormally has been specified as restartable from a step or a checkpoint, then the SJB for that job is queued by HASCJBST to CCTJRENQ in the $HCCT. On recognizing this SJB, the execution processor enters the routine in HASPXEQ located at XPRENQ. After extracting pertinent data from the SJB, the waiting task is posted. Then the job is requeued in the JES2 job queue for execution. Note that the SJB was not requeued. Any jobs withheld from execution because of duplicate job names are then released.

Jobs-Pending-Termination Queue Support

Jobs that terminate normally or are not restartable result in a subsystem request for job termination. This request is first handled by HASCJBST, which requeues the job's SJB to CCTJTERM and then waits for the subsystem to complete its termination processing. Job termination processing is illustrated in the HIPO "JES2 Execution Processor Job Termination" in chapter 2.

The execution processor job termination support begins at XPTERM. After obtaining the SJB queue lock and dequeuing the SJB, the waiting task is posted without first requeuing the SJB, and the lock is then removed. A 'X04' catastrophic error can occur if there is no SJB to dequeue and process. The job is then requeued either for execution (if specified by the operator) or for output processing. In either case, if the job had been executing by class (user job), any jobs withheld from execution because of duplicate job names are then released.

Execution Processor Subroutines

The following subroutines support the execution processor.

XQSUSE: Subroutine to Request Access to Checkpoint Data

The XQSUSE subroutine issues a $QSUSE macro instruction to obtain access to the JES2 checkpoint data.

XIO: Control Block Read/Write Subroutine

The XIO subroutine initiates an I/O operation, using the $EXCP macro instruction, and then waits for the I/O operation to complete. It performs no error checking on the I/O operation.

XSJBLOCK: Subroutine to Obtain SJB Queues Lock

The XSJBLOCK subroutine uses a compare double and swap (CDS) instruction to obtain a lock on certain SJB queues. If necessary, the subroutine waits for any current lock holder to release the lock. Return is made only when the subroutine has obtained the lock.

XSJBUNLK: Subroutine to Release SJB Queues Lock

The XSJBUNLK subroutine removes the SJB queues lock and posts the waiting lock requester.

XDEQSJB: Dequeue on SJB

The XDEQSJB subroutine is invoked by the local macro instruction, $DEQSJB (which is unique to XDEQSJB), after first obtaining the SJB queues lock. The appropriate SJB queue is searched for the given SJB. If it is found, the return is made with condition code 0; otherwise, the SJB queues lock is released, and a non-zero condition code is returned.

XQUESJB: Return SJB to SJB Queue

This subroutine adds passed SJBs to the head of the specified SJB queue.

XPITMSG: Issues PIT Status Message

The XPITMSG subroutine issues a message to the operator indicating that the PIT is inactive, halted, or drained.

X$PHASP: Subroutine to Withdraw JES2 from the System

After processing all the SJB queues, HASPXEQ checks to see if the system is being stopped (a $P JES2 command has been entered). If it is, and no FSSs are active, X$PHASP is invoked to ensure that all PITs are drained and inactive, and that the initiators and request-jobid tasks are stopped. If there is any system activity, this routine sets a flag to prevent a quickstart when JES2 is restarted. It then dequeues all starting tasks in the system, and finally communicates to HASPCOMM that its processing is complete and puts HASPXEQ into a permanent wait state.

HASPSTAC: Cancel/Status Processor

The HASPSTAC processor removes requests (SJBs) from the cancel/status queue. The job queues are then scanned for any jobs to be processed. The success or failure of status/cancel processing is returned in the SJB, along with any additional information (for status) required by the caller. HASPSTAC is invoked by a $$POST of the STAC resource by the TSCAN routine in HASCSIRQ and the HETSOUT routine in HASCSRJB.

The cancel/status support routine, beginning at entry point HA$PSTAC, is entered when a subsystem job block (SJB) is queued, via its SJBTCHN chain field, to the CCTTSCS queue in the $HCCT by the cancel/status support routines in HASCSIRQ and HASCSRJB. The routines wait for completion of processing, which proceeds in accordance with the requested function.

If there is a cancel/status request HASPSTAC dequeues the SJB from the CCTTSCS queue. HASPSTAC issues a $GETLOK, sets CCTTSLOK in the $HCCT to 0 and then does a $FRELOK and branches to SRCPTCS.

At STCSTART, HASPSTAC requests access to the checkpoint data set and checks for a cancel request in the SJB. If a cancel request is indicated, exit point STCZEXIT (for exit 22) is taken, if enabled, so that an installation-defined exit routine can change/modify the selection criteria.

Note: Because this exit allows installations to perform their own job queue searches as well as determine who has authority for TSO CANCEL and STATUS commands, it is the responsibility of this exit routine to ensure that, if any additional privilege is given (for example, to cancel another user's job), it performs the proper authorization and auditing.

HASPSTAC uses the criteria to select a job to cancel. If a status request is indicated, HASPSTAC issues a $GETCEL to acquire cell storage for the status information. If storage is unavailable, a cross-memory post is issued (via $XMPOST) against SJBECBP to activate the caller's cancel/status support routine and HASPSTAC looks for more work at STCQSRCH. However, if storage is obtained, HASPSTAC takes the same exit point STCZEXIT (for exit 22).

If the request is to cancel a job, HASPSTAC obtains a security token for the job and calls $JOBPROF to obtain a job profile name. It then issues the $SEAS macro to subtask a call to the RACROUTE authorization service. RACROUTE processing determines if the caller has authority to cancel the job and processing continues accordingly.

The job queue is scanned to locate the one and only job queue element (JQE) that matches the requested job name and optional job number (job id). If a JQE is found, the $JCAN macro instruction is executed to cancel the job. $JCAN either cancels the job or returns to an error location. If the cancellation is rejected by $JCAN, the job was not uniquely identified, or the job was not found; an error return code is set into the SJBTRETR field. Otherwise, an acceptance return code is set. The SJBECBP ECB is posted via $XMPOST and the processor looks for more work.

If the request is for status, a work area is obtained via the $GETCEL macro instruction, and the job queue is scanned in one of three modes:

1. A search for job queue elements with job names equal to the user id plus one character
2. A search for job queue elements with job names that exactly match the requested job name
3. A search for the single job queue element with a job name and job id that matches the requested job

If the scan results indicate that no job queue elements are found or that too many found to fit the response information in the work area, a diagnostic return code is set in the SJBTRETR field, the SJBECBP ECB is posted, (via $XMPOST) and the processor returns the count of the number of jobs found and looks for more work. Otherwise, for each job found, a response element is constructed in the work area indicating the status of the job. During the job queue scan for a multiple status request, exit point STCSEXIT (for exit 22) is optionally taken to adjust the scan criteria. Also, exit point STCYEXIT (for exit 22) is taken when the work area is too small to hold the status information.

When the installation exit routine returns, the job queue scan continues or ends depending on the return code. A return code of 8 causes the job queue search to end; a return code of 0 or 4 causes the job queue search to continue until the end of the job queue is encountered. In either case, the SJBRETR code is set, the ECB is posted, and the processor looks for more work. If the work area is not large enough, the required size is calculated and returned. If the required size becomes greater than 32K or if the $GETCEL macro instruction is unable to obtain the storage required by the requester, the register 15 return code field in the subsystem job block (SJB) is left non-zero (indicating a logic error).

Return codes in SJBTRETR for cancel are:

Code	Meaning
SSCSNOJB (X'04')	No job found
SSCSBADI (X'08')	Job name found but no matching job id
SSCSOUTP (X'14')	Job on $OUTPUT queue (user did not request output cancellation)
SSCSRTOK (X'00')	Job set for cancel
SSCSICAN (X'1C')	Invalid request. User attempted to cancel execution of an STC or TSU job or user attempted to cancel individual output

Return codes in SJBRETR for status are:

Code	Meaning
SSCSNOJB (X'04')	No job found
SSCSBADI (X'08')	Job name found but no matching job id
SSCSMALL (X'10')	Area requested too small
SSCSRTOK (X'00')	Area contains useful response

HASPPSO: Process-SYSOUT Processor

This routine provides data set names and characteristics of subsystem data sets to conversational terminal systems and to external writers. It also provides the interface between the HASP command processor and the process-SYSOUT processor(s) for disposition processing of a job's held output.

When the process-SYSOUT processor is first entered, it obtains permanent buffer storage (via a GETMAIN macro instruction) for an input/output table (IOT) and job control table (JCT). It issues a $GETBUF for the IOT/JCT input/output buffer and the OUTPUT CHK input/output buffer. (The buffers are released when there is no work on the SYSOUT queue.) HASPPSO abends with an error code of T01 if the buffer storage is not obtained. Otherwise, a pointer is established for the IOT/JCT buffer and the $BFRBLD macro is issued to build a buffer for checkpoint I/O. Because the processor never needs more than one buffer at a time, this buffer is always used, ensuring that the processor never needs to wait for a buffer. When the processor must wait for more work, the buffer will, if necessary, be page-released using the $PGSRVC macro instruction.

Requests for process-SYSOUT support are first processed by HASCSIRQ. The information passed to HASCSIRQ in the subsystem options block (SSOB) is transferred to a subsystem control block, located in common storage, called a process-SYSOUT work area (PSO). HASCSIRQ stores the address of the PSO in the requesting user's subsystem job block (SJB). The SJB is then chained, using the chaining field SJBTCHN, to the field CCTPSOQ in the $HCCT.

HASPPSO checks the PSOUFLG field during group request data set disposition processing. If the flag indicates a data set routing change to a new destination, the $DESTCHK routine in HASCSIRQ will determine if the SYSOUT can be sent to the specified destination. If so, HASPPSO stores the PSOROUTE field in the PDBDEST and sets the route change flag (PDB2PSOR). Otherwise, the PDDB is skipped and the next one processed.

After posting JES2, HASCSIRQ waits for the request to be processed by the JES2 process-SYSOUT processor (HASPPSO). Because requests are queued to the CCTPSOQ in LIFO order, HASPPSO requeues these requests in FIFO order to the CCTPSOFF queue. If JES2 is down, these PSO requests are queued to the CCTPRGQ.

When dispatched, HASPPSO at label TQSEARCH first looks for any requests on the CCTPRGQ. If there are none, it next searches for any operator-initiated requests on the $OQUEUE. If that queue is empty, HASPPSO processes requests from the CCTPSOFF queue, then from the CCTPSOQ.

When processing the CCTPSOQ, the entire chain is requeued in FIFO order as the CCTPSOFF queue. HASPPSO is then ready to process the normal process-SYSOUT request.

Operator commands can also result in requests for process-SYSOUT support. If held data sets are to be deleted or released for output processing because of $C or $O commands, the JES2 command processor creates process-SYSOUT requests in the form of PSOs obtained from subpool 0 via GETMAIN.

These requests are queued to the HASP communications table (HCT) field, $OQUEUE. They appear to the process-SYSOUT processor as group requests affecting specific jobs. PSO requests are rejected for jobs whose spool volumes are not online.

HASPPSO, when it is not processing a request, waits ($WAIT) at location TJBWAIT. When dispatched, it first processes any operator-initiated requests (that is, those queued to $OQUEUE). It is then ready to process the normal process-SYSOUT request queue, CCTPSOQ.

To ensure the integrity of the process-SYSOUT request queue, a locking mechanism based upon the compare double and swap (CDS) instruction is employed. First, an SJB is dequeued from CCTPSOQ using CDS. Then the lock byte, CCTTSLOK, is tested. If CCTTSLOK is 0, the process-SYSOUT processor may proceed to use the SJB just dequeued. If it is non-zero, then the possibility exists that another task in HASCSIRQ is currently modifying the dequeued SJB. To ensure the integrity of the use of the SJB, HASPPSO must then issue the $GETLOK macro instruction. When control is returned, CCTTSLOK is zeroed and the $FRELOK macro instruction is issued.

When the lock is available, and there is a request to process held data sets, this routine ensures that they are not dumped.

Conversational Process-SYSOUT Requests

Conversational requests (for example, those originating for TSO or for the external writer) seek access to the queue of held SYSOUT data sets. The job queue element (JQE) for a job contains a count of the data sets currently held for a job in the field JQEHDSCT.

Each held data set in a job's hold queue is represented by a peripheral data definition block (PDDB) in a spool-resident IOT. The spool address of the first IOT is contained in the job control table (JCT) field, JCTIOT. Each contains a copy of the peripheral data definition block (PDDB), which describes the data set represented.

Conversational requests allow for altering the characteristics of a data set and for checkpointing the processing of a data set. These functions force modifications to the PDDB.

Having dequeued an SJB representing a conversational request, HASPPSO at TGOTSJB determines where to process the request. If so, information in the PSO (pointed to by the SJB) is used to update the PDDB representing the data set. If the PDDB is modified, then the IOT to which it belongs is rewritten to spool.

If the data set alteration is not specifically requested, the PSYAUTH routine is called and issues a RACROUTE REQUEST=AUTH call indicating that only READ access to the data is required. Otherwise, UPDATE access is requested. PSYAUTH is *not* called during operator command processing.

If a data set is to be cancelled, its PDDB is marked not held and the job's hold count is decreased. If the data set is to be released, its PDDB is marked not held and the process-SYSOUT (PSO) processor attempts to gather the held data set into the JOE assigned by the output processor (HASPHOPE). This is done at output processing according to the following rules:

- Spun data sets are never recombined.
- Released data sets cannot be combined into output groups for which a print request has been partially processed; this includes requests that are currently being printed or punched.
- A released data set cannot combine with an output group for which there are multiple job output elements (JOEs).
- A released data set can only combine with an output group it would normally have been a part of had it not been held.

If the released data set is unable to fit into an existing pair of work and characteristics JOEs, JOEs are created in the PSO processor's processor control element (PCE) work area. An attempt is then made to add these JOEs to the job output table (JOT). If it is successful, the job's held data set count JQEHDLCT is decreased. HASPPSO waits ($WAIT) for an available JOE.

An initial conversational request is indicated by the first byte of PSODSN being set to hexadecimal 0. In this case, the request is one of two types: an initial data set request, or a group request. In the latter case, all held data sets that satisfy the given search criteria are modified and disposed of one at a time.

An initial data set request normally causes the hold queue to be searched (starting at the top) for the first data set that matches the search argument.

The external writer initial request, recognized by its unique writer name in the parameter list, requires that only data sets with a data set identifier (DSID) be returned.

External Writer Process-SYSOUT Requests

External writer requests are assumed to be on behalf of a program writing JES2 SYSOUT data sets to high-speed devices. For this reason they are allowed to compete for JOEs on an equal basis with JES2 print/punch processors.

Unlike the print/punch processors, HASPWTR allows any combination of the following search criteria: job name, job ID, SYSOUT class, destination, writer name, and forms number. A branch is taken to the JOE screen routine in HASPJOS, (pointed to by $JOESCRN in the HCT,) to check the CPU id and spool availability for the JOE. If the JOE fails this check, another JOE is obtained. Otherwise, HASPWTR continues to check the JOE against the selection criteria. If there are no JOEs available, HASPWTR XMPOSTs the writer to wait on the JOT.

When processing an initial external writer data set request or when all of the data sets corresponding to a previously selected JOE have been processed, HASPPSO scans the entire JOT looking for a JOE that satisfies the specified search criteria. If more than one JOE satisfies the search criteria, the highest priority JOE is selected. A RACROUTE call is then issued to verify that the external writer device has authority to process the JOE. (An entry is placed in the JWEL table if authorization is denied.)

Once a JOE has been selected and authority verified, the JOT is checkpointed, and system management facilities (SMF) data in the JCT is transferred to the PSO. Then the first regular IOT is read and its PDDBs are scanned (the PDDB routine must match that in the print/punch processor) for the first PDDB whose characteristics match those of the selected work and characteristics JOEs. Information from this PDDB is used to complete the PSO, and the waiting task is posted.

When the PDDBs in the current IOT have been exhausted, the next IOT (if any) is read, and the scan is continued. When all IOTs have been searched, the JOEs are removed from the JOT using the $#REM macro instruction. Another JOE, which satisfies the PSO search criteria, is searched for.

If no JOE is found, the external writer is made to wait until a change is made to the JOT. This is done by providing an event control block (ECB) address in PSOWTRC and return code 4 in PSORETN. When work becomes available, $#POST posts any external writers that are waiting for that type of work, causing the writers to repeat their process-SYSOUT requests.

Job Disposition Processor

The job disposition processor communicates between the HASPCOMM processor and the PSO processor(s) to handle held data sets. Processor work requests are indicated by JDR elements queued to the $JDRQUE anchored in the HCT. $O operator requests are mapped into JDR requests by the command processor.

If no work is queued, the processor issues a $WAIT on work.

If work is obtained, the processor increments the active count with $ACTIVE and establishes a $ESTAE environment.

When the job disposition processor processes JDR elements, the job queue is searched for JQEs with held data sets that meet the criteria specified in the JDR element. The PSO element (contained within the processor PCE work area, $JDRWORK) is initialized from information in the JDR and the selected JQE, then queued to the $OQUEUE, and the PSO processor is $POSTed. For purge requests, each JQE represented in the JDR element is located and tested for the presence of the QUEPURGE flag. For each JQE that indicates QUEPURGE and held data sets, the PSO request is queued to the $OQUEUE to delete the held data sets, and the PSO processor is #POSTed. On completion, the job disposition processor issues operator messages based on information returned by the PSO element. When all jobs represented in the JDR have processed, the JDR is freed with a $RETWORK, the $ESTAE is cancelled, and the active count is decremented using $DORMANT.

This routine also deletes data sets for cancelled jobs processed by the $JCAN service routine.

Process-SYSOUT Processor Subroutines

The following subroutines are used in support of the process-SYSOUT processor.

TCHKBUF: Subroutine to Get IOT/JCT Buffer and Checkpoint Buffer

The TCHKBUF subroutine is called to get a IOT/JCT buffer and a checkpoint buffer for use in processing requests for held data sets. It issues a $GETBUF macro for the buffers, if they do not exist.

TIOTWR: Subroutine to Write an IOT/JCT Buffer

This subroutine issues a $EXCP macro to initiate the I/O event. It then waits until the I/O is complete and then returns without altering or inspecting the return code.

TCBREAD: Subroutine to Read and Check an IOT/JCT buffer

The TCBREAD subroutine issues a &EXCP macro to initiate an I/O event. It then waits until the I/O is complete and returns if the IOT contains a valid identifier key. Otherwise, it branches to an error routine.

TEXCP: Subroutine to Read or Write an IOT Buffer

The TEXCP subroutine issues a $EXCP macro to initiate the I/O event. It waits until the I/O is complete and then returns without altering or inspecting the return code.

PSYAUTH: Process SYSOUT Authorization Routine

PSYAUTH performs RACROUTE authorization requests for access to PSO data sets. It is called by HASPPSO and HASPWTR to handle requests:

- For the TSO RECEIVE command
- To read a PSO data data set
- To alter a PSO data set.

PSYAUTH places one of the following return codes in register 15:

- 0 - Processing was successful. Access to the data set is permitted.
- 4 - Access is not permitted for a TSO user trying to receive data. Access may be possible if the user issues a RECEIVE command when logged on with another SECLABEL.
- 8 - Access to the data is not permitted.

HASPPRPU: Data Output Functions

Module HASPPRPU provides two groups of routines:

- The print/punch processor, entry point HASPPPI1, which performs the output operations that cause a data set to be written on the appropriate device
- The HASP image loader, entry point HASPIMAG, which loads forms control buffer (FCB) and universal character set (UCS) images from SYS1.IMAGELIB.

Print/Punch Processor

The print/punch processor, a re-enterable program, concurrently supports all of the local and remote printers and punches that have been identified to JES2. In addition, when the $SPOLMSG initialization parameter indicates that undeliverable remote console messages may be spooled, the print/punch processor transmits such messages to a remote printer. (Console messages directed to a remote work station may be undeliverable because the work station was not available when the message arose, or because the work station does not include a console.)

At JES2 initialization, a processor control element (PCE) and a device control table (DCT) are constructed for each printer and punch in the system. Each execution of the print/punch processor represents the transmittal of one or more output data sets to a specific device. The PCE contains control flags and work space associated with the current use of the processor; the DCT describes the specific device.

For the 3800 printer, JES2 initialization also acquires storage for the 3800 pending page queue. The 3800 pending page queue comprises a page queue header (PQH) followed by page queue entries (PQEs).

The PQH contains pending page queue status information. Each PQE contains a 3800 physical page id that links each PQE with the physical 3800 output page that it represents. (Once the 3800 stacks a physical output page, JES2 deletes its associated PQE.) Therefore, the pending page queue serves as a map of all data sent to the 3800 but not yet stacked. JES2 uses the pending page queue for checkpoint processing, operator command processing, paper jam processing and SMF processing.

The print/punch processor error recovery consists of a $ESTAE environment created in the JES2 main print/punch loop. The start of the $ESTAE environment is at PGOTPDDB and the end is at PDPDSEND. PRPURCV0 attempts to recover only from program check errors; it simulates the $I command to interrupt the I/O at the point of the error and to requeue the output to recover from the error.

Message $HASP185 is issued if an error occurs during print processing of a data set. The possible errors are: a data set is not closed, an invalid track address, an I/O read error, and invalid job key, an invalid data set key, or an obsolete JOE.

HASPPPI1: Print/Punch Processor Initialization

The following subroutines provide print/punch processor initialization logic.

PRPUINIT: When initially dispatched, the processor sets up its base register and clears the fields in its work area.

PGETUNIT: When initially dispatched, the processor issues the $GETUNIT macro instruction to attempt to acquire the output device on behalf of which the dispatch occurred. If the device is unavailable, the initialization routine waits, then reissues the $GETUNIT request, repeating this cycle until the unit does become available. For the 3800 printer, if the device is unavailable, HASPPRPU calls the PPQMGR routine to process unstacked output not waiting on the 3800. For a local device, the routine waits for the device as a resource ($WAIT UNIT). For a remote printer or punch, the routine waits until the corresponding PCE is specifically posted ($WAIT WORK). Ordinarily, local output devices are available unless drained through an operator command or initialization parameter. Remote printers and punches are ordinarily *not* available when the processor is initially dispatched on their behalf. Remote devices are marked as unavailable until work is available in the JOT for them.

PGOTUNIT: When the $GETUNIT request is successful, the routine marks the device's DCT in-use and issues the $ACTIVE macro instruction to alert the JES2 dispatcher that the system is not dormant. If the device is a remote printer (Rn.PRn), and a remote message data set exists for this remote device, then the existing remote messages are despooled and transmitted to the remote printer. The address of the remote message data set JQE created by the remote console processor is obtained from the RAT and locked via $GETLOK. A buffer is acquired for despooling the remote message data set IOT and message records, and the remote message data set IOT is read. The track address of the first remote message record is obtained from the IOT and read. Various control values are set in the HASPPRPU PCE work area in order for the remote messages to be despooled and transmitted by main HASPPRPU processing code; that code is then entered at P1STBLK. The main print loop is used as a closed subroutine until all messages routed to the devices are printed. Finally, blank lines are sent to the remote printer, resources obtained previously are freed, the remote message JQE is removed from the JOB queue, track groups for the message records purged, the $DORMANT macro instruction is issued, and control returns to PRPUINIT.

PGETJOB: When the $GETUNIT request has been successful for a device other than a remote printer, the initialization routine issues the $#GET macro instruction to scan the job output table (JOT) for work for the device. Unless the device's setup has been altered through operator commands or JES2 initialization options, the device (if local) is assumed to be in automatic mode with standard forms mounted; preference is thus given to jobs that require matching setups.

If no work is available for this device, PGETJOB sets the DCTHOLD indicator, issues the $FREUNIT macro instruction to release the device, and issues the $DORMANT macro instruction to cancel its earlier active status. For the 3800 printer, HASPPRPU issues a CLEARPRINT CCW to force printing of any output in the 3800 page buffer and to process any outstanding operator commands. If a remote device is being processed, control returns to PRPUINIT, which waits ($WAIT WORK) until work is available. Unless the local device has not been active since the message was last issued, the routine issues message $HASP160 to indicate that the device is inactive. After issuing the message, or if no message was required, the routine waits ($WAIT JOT) for work to be added to the JOT. On regaining control, PGETJOB waits again if a command is in progress; otherwise PGETJOB releases the DCT and checks for a 3800 device. If the device is not a 3800, PGETJOB returns to PRPUNIT; otherwise it checks to see if the 3800 is draining. It it's draining, PGETJOB calls the PALLOC subroutine to acquire buffers for the draining 3800 and then branches to PDRAINMC to finish the draining process (via an eventual $FREUNIT), PGETJOB then issues a $DORMANT and returns to PRPUNIT.

PGOTJOB: If work is available, the $#GET macro returns the addresses of the work job output element (JOE), the characteristics JOE, and the job queue element (JQE). The initialization routine stores these addresses in the PCE, together with the time of day (obtained through the TIME macro), then proceeds, depending upon whether a local or remote device is being serviced.

For a remote device, the routine issues a $EXTP OPEN macro instruction for the device, then uses the $GETBUF macro instruction to obtain one or two buffers, depending upon the buffering option selected at JES2 initialization. After acquiring the required number of buffers, the routine branches to the common job control table (JCT) read routine, PJCTRD.

For a local device the initialization routine determines whether the single buffer method or the track-cell method of despooling is to be used. Track cell despooling is the process by which a local print processor can input several spool records of a data set in a single EXCP, thereby reducing the number of accesses to the spool volume. If the print processor is eligible for this despooling method, and if the data set is of a SYSOUT class that was ordered into track cells when it was created (both determined by JES2 initialization options), then enough buffers are obtained to read all the spool records in a track cell. For a 3800, PGOTJOB then calls the PALLOC routine to acquire all necessary buffer resources.

The JCT is read into into main storage; the JCT is found through the track address in the JQE. If an error is detected, the routine branches to PRPUEXIT to terminate initialization. If no error is detected, PGOTJOB attaches the HASPIMAG subtask, and waits for it to initialize. If the subtask is already active or the device is a punch, then no attach is necessary. PGOTJOB at PCLDSTRT then processes the print/punch header and the checkpoint data, depending on whether a cold or warm start took place.

PCLDSTRT: PCLDSTRT calls the PHEADER subroutine to setup the printer and print an output separator, if necessary. On return from PHEADER, if the PPNEWS flag is set, control is passed to the main print loop at entry PNEWSGO.

PCKPTNIT: The last part of processor initialization consists of initializing the print/punch processor work area checkpoint information. For cold starts, the checkpoint data is cleared to zeros, the MTTR of the input/output table (IOT) is set from the JCT (or, for spin data sets, from the WORKJOE), and the initial peripheral data definition block (PDDB) displacement into the IOT is set. For warm starts, the checkpoint data is set from the CHK read from the SPOOL. Finally, for both warm and cold starts, the physical page size for printers is set from the JCT. For punches, the physical page size and the logical page size are the same, and this value is set by the PGOTPDDB routine from the value of CKPTLINE used for the current data set.

Data Set Selection

The following subroutines provide data set selection logic.

PENDINIT: With initialization complete, the data set selection routine reads the first IOT associated with the current PCE. If the IOT is not read successfully from the spool volume, the routine issues $DISTERR to indicate that an error has been detected at label PIOTPRE in control section HASPRPU, and PPDONE is entered to abort the job. (The original error indication, an event control block completion code indicating an I/O error, is detected in PRDCHK, the I/O check subroutine.) The routine also checks to see if this is a spin IOT. If so, it ensures that it is still valid for this JOE. The spin IOT may be a reused IOT, and the JOE may be (because of an intervening JES2 abend) the JOE associated with the IOT before it was reused. If it is an old JOE, a $DISTERR is issued and message $HASP185 is issued. The routine then exits to PPDONE to abort the job.

PDDBNEXT: If the initial IOT is read successfully, PDDBNEXT scans each PDDB associated with the IOT, and on reaching the end of each PDDB chain moves into the PCE the pointer to the next IOT associated with this job. Then PDDBNEXT returns to PENDINIT. This cycle is repeated until all PDDBs have been examined. PDDBs are scanned to find those which match the work and characteristics JOEs. PDDBs that do not match the applicable JOEs are ignored.

PGOTPDDB: For each selected PDDB, PGOTPDDB extracts and saves the device setup information. The device setup information consists of such items as forms, UCS, FCB, forms flash, writer id, and processing mode (PRMODE). PGOTPDDB initializes the characteristics JOE with this information. A SAF call is then issued to determine if the device is authorized to print the data. If so, PGOTPDDB invokes exit 15 at exit point PPEXIT. The installation exit routine can change the number of copies of this data set, up to a maximum of 255, or bypass this PDDB and request another PDDB.

If the device is a 3800 printer, additional setup information is extracted; if a data set partially printed to a 3800 is being resumed, copy group and forms flashing counts are adjusted.

All 3800 data sets and copies are offset-stacked if BURST=YES was specified by the programmer; this causes the 3800 printer to physically separate all data sets in the burster/trimmer sheet stacker.

PSMFTST: The characteristics of the new PDDB are compared with those of the previously processed PDDB, and if they are found to be different, a type 6 SMF record is produced (through a call to the PPSMF6 routine) for all data sets printed or punched since the last type 6 record was written. Additionally, such events as buffer read errors, operator overrides of carriage control, or a JES2 warm start cause a new type 6 SMF record to be generated.

PDDBFCHK: The device setup verification subroutine, PRPUDSV, is entered. On return, processing continues based on whether a data set warm start is in progress. (A data set warm start is the resumption of printing or punching for a data set for which output has been partially completed.) The PDDB for such a data set is the first one to be processed. Processing of the first data set (not warm start) causes the output checkpoint record (CHK) to be initialized and written to spool. The checkpoint-valid indicator in the work JOE ($JOECKV) is set, and the work JOE is checkpointed.

PNXTCPY: This subroutine is the restart point for making copies of data sets. After establishing recovery (via $ESTAE) PNXTCPY establishes forced spacing between each copy of a data set and then at exit point PCEXIT (for exit 15) invokes an installation exit routine (if existing and enabled). The installation exit routine can place its own unique separator between the copies of the data set. When the installation exit routine returns control to PNXTCPY, a checkpoint area is initialized, if this is the first data set, and PNXTCPY writes the checkpoint area to spool.

PFASTRT: The data set selection routine completes preparation for output by turning off the warm-start bit in the processor work area and, if necessary, initializing the backspace table. The data set selection routine calls the PQEDINIT subroutine to acquire and initialize a data set page queue element (PQE).

Main Processor (Main Print/Punch Loop)

The following subroutines provide the main print/punch loop logic.

PBSFSGO: This subroutine reads from a spool volume the output records that are to be printed or punched. PBSFSGO is the entry to main loop initialization for new or warm start data sets. For impact printers, it is the restart point for backspace ($B) or forward space ($F) processing.

PBSFSGO calls the PRDTCEL subroutine to read the first data block or track cell from a spool volume. On regaining control, PBSFSGO tests to see if there has been an operator command for the data set or the output device requiring that processing for the data set should be suspended or terminated. If so, PBSFSGO branches to the data set termination subroutine, PPDSEND. Otherwise, PBSFSGO calls the input I/O check routine, regaining control when input (despooling) I/O is complete.

P1STBLK: The main print/punch loop, which begins at this label, steps from one record control block to the next throughout each buffer belonging to the data set, building a CCW for each output record. As each data buffer belonging to the selected data set is read, it is checked for validity. A failure of this validity check results in termination of that data set and selection of the next data set. Each CCW for a local output device is added to a printer or punch chain through a call to the PPPUT subroutine. CCWs for data sets intended for remote work stations are passed to the remote terminal access method (RTAM) by means of the $EXTP macro.

PNXTCCW: This section of the print/punch loop is entered whenever a CCW is created that will cause an output line to be written. Before proceeding with preparation of the next output line, the routine determines whether another input area is available and, if so, calls the PRDTCNXT subroutine to read the next input block; this ensures that despooling will proceed, if possible, concurrently with output processing.

Preparation of the next output CCW continues. For spanned records, only the first 254 bytes are printed. For data sets using control characters, American National Standard codes are converted to machine codes and a single CCW is created for each record, the control information being merged with the previous CCW.

PRINT: Printer CCWs are completed and tested for validity. Automatic page overflow is provided for data sets when the lines-per-page parameter on the JOB statement is not zero. This does not, however, prevent printing over the page perforation; because, the line counter is reset whenever a skip to any channel is encountered in the data set.

If the end of the physical page is reached, either because the page is full or a channel skip is indicated, the data set page count is updated, a new entry is added to the backspace table (if applicable), and the PPCHKPT routine is entered.

If not at the end of the physical page, PRINT branches to PPCKLNS to determine if the data set is at the end of a logical page.

PPCHKPT: PPCHKPT is entered at the end of each physical page: at every channel skip issued to a printer, or after the number of cards specified by CKPTLINE have been written to a punch. The SMF page count is updated and a branch is taken to the PPCKPGS routine.

PPCKLNS: PPCKLINE is enter from PGOTPDDB to determine the end of a logical page for printers. If the CKPTLINE value used for the current data set is zero, the logical page is determined solely by channel skips and line count (that is, logical page size = physical page size). If this is the case, PPCKLNS branches to PRNOVFL.

If CKPTLINE is not equal to zero, the CKPTLINE counter is decreased by the space count of the current CCW. If the adjusted CKPTLINE counter is less than or equal to zero, a logical page boundary has been reached and the PPCKPGS routine is entered. Otherwise, a branch is taken to the PRNOVFL routine.

PPCKPGS: For printers, PPCKPGS is entered at the end for each logical page, at every channel skip, when the physical page is full, or when CKPTLINE lines have been written. For punches, PPCKPGS is entered when the number of cards specified by CKPTLINE have been punched.

A check is made to determine if the number of pages specified by CKPTPAGE have been generated. If not, a branch to PRNOVFL is taken; otherwise, processing continues as follows.

For remote devices, the checkpoint-required flag is set to cause a checkpoint after the current record has been printed or punched.

For local devices, the checkpoint will be taken when the number of pages specified by CKPTPAGE is reached, but not necessarily each time it is reached.

For the 3800 printer, PPCKPGS calls the PQECINIT and PPGIDIO subroutines to acquire and initialize a checkpoint PQE. The checkpoint will be taken when the page associated with the checkpoint PQE reaches the 3800 stacker.

PRNOVFL: If requested, each print line not directed to a 3211, 3800, 1403, or 3203 printer is translated to remove unprintable bit patterns, and to translate lowercase characters to uppercase. The translation table is compatible with PN and QN print chains; it should not be specified if TN or other special print chains are to be used.

For a 3800 printer, if OPTCD=J was specified, and if a table reference character different from the last used is specified, the routine creates a special CCW to select the required translation table and calls the PPPUT subroutine to add the CCW to the chain.

PRCHAIN: PRCHAIN calls the PPPUT subroutine to add the new print or punch command to the CCW chain. A check is made to determine if a checkpoint is required after the current CCW has been added to the chain. A branch is taken to PPPFUNCI if a checkpoint is not required. Otherwise, processing continues as follows.

For remote devices, a buffer truncate command (CCW op code of X'FF') is sent to the remote terminal access method (RTAM) via a call to PPPUT. The truncate command causes RTAM to send all output for this device to this device and does not return until the remote indicates it has received the output successfully. Once

RTAM returns to PPPUT, PPPCKPT is entered to take the checkpoint. Note that the checkpoint is not taken until the processor knows the output was successfully received by the remote. Finally, control returns to PRCHAIN, which in turn branches to PPPFUNCI.

For local devices, PPWRITE is called to close out the current CCW area and schedule if for execution. Processing continues at PPPFUNCI.

PPPFUNCI: PPPFUNCI determines whether special 3525 card punch processing is required. Special support for the 3525 punch is provided to supply interpreting on lines 1 and 3 of the punched card image if a print feature is installed and the DCB parameter FUNC=I was coded in the JCL. Using machine control characters, the user can mix print and punch operations to write any text on the punched card as desired, if the appropriate punch feature is installed. The subroutine adds one or (if the print line exceeds 64 characters) two CCWs to accommodate the print line.

For print records, the current CCW is set to write, no space, and an indicator is set if the record contains an American National Standard carriage control character. The routine then returns to the main loop to continue control character processing.

PRNOPRNT: If the device is a 3800 printer where the printer is being repositioned, PRNOPRNT calls the PMAPFCB subroutine to perform FCB mapping against the current CCW.

PNXTRCB: The address of the next record control byte (RCB) is calculated. Unless the end of the data buffer has been reached, PNXTRCB reenters the main loop at PNXTCCW to process the record described by the new RCB.

PCPEND: At the end of the input data buffer, if the end of the input data set has not been reached, PCPEND obtains the next buffer. If input data has been despooled by track cell and the next buffer is already in main storage, PCPEND branches to the PNXTBLK subroutine to process the new buffer. If despooling is by single buffer, or if all buffers of the last-despooled track cell have been processed, control passes to the PENDTCEL subroutine. For 3800 printers, if an end-of-data is reached prematurely while processing a $F command, PCPEND calls the PRECOMP subroutine to adjust the PQE page IDs.

PENDTCEL: PENDTCEL is entered when CCWs have been built for all RCBs in the current single buffer or the last buffer of a track cell. If the processor is currently skipping buffers in the course of processing a forward-space ($F) command, the available buffer count is increased by 1 and the next read operation is scheduled.

PSETFINL: An indicator is set in the new CCW area to indicate that the data buffer will become available when the CCWs have completed execution. (Subroutine PPCHECK subsequently will update the buffer count upon detecting this indicator.) A call to the PPWRITE subroutine is then made to schedule the new CCW area for execution.

PCHREAD: A test is made to determine whether a despooling operation has already been started; if so, PCHREAD branches to the PCHKNBLK routine at the beginning of the main processor to check the I/O and to process the new data.

If a despooling operation is not in progress, a call to the PRDTCNXT subroutine is made to start one, using an available buffer. If no buffer is available, the PPCHECK subroutine is called (using the PCIWAIT entry point) until a buffer is made available.

Finally, the main processor branches to the PCHKNBLK subroutine to wait for the despooling operation to complete and to process the new data.

PPDSEND: Data Set Termination

The PPDSEND routine is entered when the end of the current input data set is reached because of an end of file, buffer read error, or operator detection or suspension. If the device is a 3800 printer, PPDSEND calls the PQECINIT and PPGIDIO subroutines to create the last page PQE for the data set. PPDSEND forces execution of the channel commands in the CCW area currently being filled and reads the current IOT to prepare for selection of the next data set PDDB scan. If the processor is printing spooled remote messages, control returns to PRSMBEOB to process the remaining message buffers. Otherwise, message $HASP185 is issued if data set processing was terminated because of an I/O error or validity error. PPDSEND issues the $DISTERR macro instruction and aborts the job if an I/O error occurs when the IOT is read or if the IOT read does not match that for the current job.

For impact printers for data sets suspended to permit a forward-space ($F) or backspace ($B) of the device or data set, PPDSEND performs the required spacing operation and uses message $HASP170 to indicate that the action is complete. Control then returns to the main processor at label PBSFSGO to continue printing.

For the 3800 printer, the PPDSEND routine calls the P3800CMD routine to process any repositioning commands. On return, depending on the command, PPDSEND may:

- Pass control to the PPDONE routine to terminate the job
- Pass control to the PENDINIT routine to resume printing at a new location
- Continue through the PPDSEND routine.

If the data set just completed is the JES2 job log, PPDSEND prints the JES2 job statistics block, using data collected from the JCT. If the PDDB for the current job indicates that more than one copy of the output data set is to be produced, the same PDDB is selected until the required number of copies has been produced. For the 3800 printer, copy group counts and the count of the number of remaining copies for which forms flashing is required are also updated, and if the 3800 is bursting the current data set, an end-of-data-set CCW is issued to cause the data set to be offset-stacked in the burster for easier separation from the next copy. Printing of data set copies commences at label PNXTCPY in data set selection.

Print/Punch Processor Service Routines

The following text describes the print/punch processor service routines.

PRPUT: Print/Punch Separator Service Routine

PRPUT is entered via a call from any $PRPUT macro instruction coded in the user's exit routine that is called from the JES2 separator exit point. PRPUT constructs a CCW for each data record to be output. It calls PPUT to put the CCW into an output buffer. If the output device is a SNA remote device with spooling capabilities that are being used, the user exit routine must create and send a PDIR (peripheral data information record). To do this, the user exit routine can use a $SEPPDIR macro instruction to call the SEPPDIR remote setup header routine.

PPPUT: Channel Command Processing Subroutine

The PPPUT subroutine is called by the main print/punch loop to build a channel program by adding a CCW to the chain of CCWs in a CCW area. If the channel program in the specified area is not already active and room for an additional CCW remains, PPPUT translates the data address field of each CCW passed to it from a virtual to a real address and appends the CCW to the channel program.

The subroutine calls the PPWRITE subroutine to schedule the channel program for execution if the current CCW area is full. (The size of the CCW area is determined by the CCWNUM parameter on the PRINTDEF and PUNCHDEF initialization statements.)

For remote devices, PPPUT issues a $EXTP PUT macro instruction to pass the output request CCW to HASPRTAM. PPPUT then calls to the PPCHECK subroutine to check for operator commands. If the $EXTP PUT was not successful, an indicator is set to prevent any more checkpoints of this data set. CCWs requesting an immediate skip to channel 1 are ignored if the printer is already positioned at channel 1. For local devices where an immediate control request is received and for which the preceding request was for a write no-space operation, PPPUT merges the two requests into a CCW requesting write with control.

PPWRITE: Channel Program Processing

To reduce the number of EXCPs required to write an output data set to a local printer or punch, JES2 uses program-controlled interrupts as an aid in scheduling channel programs for execution.

Channel programs are constructed in a CCW area by the PPPUT subroutine The last two doublewords in each CCW area are a program-controlled interruption element (PCIE). The PCIE is a JES2 control block consisting of an NOP CCW followed by a transfer in channel (TIC) CCW; portions of each CCW, ignored by the channel, are used by JES2 to contain control information.

PPWRITE: PPWRITE checks to determine whether the print/punch processor is operating on behalf of a remote device, or if the CCW area is empty. In either case, PPWRITE returns to its caller with no further action. Otherwise, PPWRITE initializes the PCIE at the end of the specified CCW area by turning on the PCIBUSY and PCIACTIV indicators in the PCISGNAL byte and by turning off the command chaining flag in the NOP CCW. The PCI flag is set in the first CCW in the new area.

If the CCW area contains a channel program but is not full, PPWRITE skips over unused space by appending a TIC CCW to the channel program, with the real address of the PCIE as the transfer-in channel target. After adding the TIC CCW, or immediately if the CCW area was already full (in which case the last data CCW was already contiguous with the PCIE), PPWRITE reverses the pointers to the primary align and secondary CCW areas in the processor control element (PCE) and tests to determine whether a write operation is already in progress. If it is not, PPWRITE enters the PWEXCPVR subroutine to initiate a channel program.

If a write operation is already being executed, PPWRITE attempts to chain the new channel program in the current CCW area to the one being executed. The NOP CCW in the current area's PCIE is converted to an NOP with the command chaining flag on. A compare and swap (CS) instruction is used to attempt to set the command chaining flag on in the NOP CCW of the channel program being executed. If that attempt is successful, the TIC is executed when reached in that channel program, causing channel program execution to continue with the first CCW in the new CCW area.

The attempt to set the command chaining flag on is successful if:

- The PCIE has not been executed by the channel. This is indicated by the PCIBUSY flag being on.
- The channel program has not been aborted to process a 3800 repositioning command. This is indicated by the PCIABORT flag being off.

When the first CCW in the CCW chain to which the TIC transfers is executed, the PCI appendage (routine EPIC in HASPNUC) resets the PCIBUSY bit in the PCIE.

If the attempt to set the chaining flag on is unsuccessful, the current channel program cannot be chained to the one already in execution and must be executed through a new EXCP. PPWRITE links to the PPCHECK subroutine to await completion of the active channel program, then enters the PWEXCPVR subroutine to have the current channel program initiated.

PWEXCPVR: PWEXCPVR is entered only when it is necessary to initiate execution of a channel program. In addition to initializing the input/output buffer (IOB) as required by the input/output supervisor, PWEXCPVR establishes the link between the IOB and the applicable PCIE that permits the JES2 PCI appendage in HASPNUC to communicate with the PPWRITE subroutine in HASPPRPU. The address of the channel program's PCIE is placed in the IOB. When the program-controlled interruption occurs, the PCI appendage uses the PCIE address in the IOB to access the correct PCISGNAL flag, turning the PCIBUSY flag off for inspection by the PPWRITE subroutine as described above. PWEXCPVR resets the PCI bit in the first CCW, issues a $EXCP (TYPE=VR) macro instruction to initiate channel program execution, turns on an indicator that a channel program has been scheduled, and returns to the caller.

PPCHECK: Error Detection and Correction Subroutine

The following describes print/punch error detection and correction logic.

PCIWAIT: This entry point in PPCHECK is used if it is necessary to wait for a program-controlled interruption to occur. A branch to PPCHECK is taken if no write operation has been scheduled or if the specified PCI has already occurred.

PPPWAIT: Upon entry at this label, directly or from PCIWAIT, PPPWAIT waits ($WAIT IO) for an I/O operation to complete before entering PPCHECK. The wait is satisfied when an I/O event occurs, such as a PCI or a channel end, or as a result of an operator command to interrupt or delete output.

PPCHECK: The PPCHECK subroutine communicates with the JES2 command processor, that takes action in response to operator commands; the JES2 I/O supervisor appendages that recognize such conditions as printer or punch error recovery and program-controlled interruptions; and the JES2 checkpoint processor that maintains the current status of the processor for warm start situations. Support for the 3211 printer CANCEL key is also provided that simulates a forward-space data set and effectively cancelling the data set in progress.

PPCHECK initiates 3800 command processing on a channel-end basis. Channel ends can occur for an intervention required condition, which causes the abnormal-end appendage to abort the channel program. if a command is detected and no channel end has occurred, PPCHECK waits until the channel end is received before processing the command. PPCHECK calls PLOCATE to determine if the command is processable. If it is, a branch to PPDSEND is taken to process the command. If it is not, and an intervention required occurred, PPCHECK restarts the channel program at the CCW where the intervention occurred.

Whenever a PCIE is executed for a 3800 printer, PQECOMP is called to complete any PQE assignments and to process any PQEs that have reached the stacker.

Track Cell Read Routine

The following paragraphs describe the track cell read routines.

PRDTCEL - Track Cell Read Routine: If track cell despooling is in effect for the current data set, the track address (MTTR) of each input buffer image in the track cell on the spool volume is determined, and the track addresses are sorted into ascending order. A channel program is then generated to read the entire track cell in a single rotation of the spool device, and a $EXCP macro instruction is issued. (A set sector CCW is included in the channel program if the spool device supports rotational position sensing.) Control then returns to the caller.

PRDTCHK - Track Cell Read Analysis: PRDTCHK waits for the completion, and analyzes the success, of each track cell despooling operation. If a read error occurs when a buffer image is being read from the spool volume, the corresponding main storage buffer is flagged as being invalid, and the channel program is restarted at the next buffer image in the track cell. When all buffers in the track cell have been read, control returns to the caller, who processes the buffers in buffer order as opposed to track order, up until the error buffer, if any.

Single Buffer Read Routine

The following paragraphs describe the single buffer read routines.

PRDBUF - Single Buffer Read Routine: If single buffer despooling is in effect for the current data set, PRDBUF initializes a common channel program to access the required buffer image on the spool volume, then issues a $EXCP macro instruction to read the buffer image into main storage. The routine reduces by one the number of input buffers available to be filled and returns to the caller.

PRDCHK - Single Buffer Read Analysis Subroutine: PRDCHK swaps the primary and secondary buffer pointers, then tests the current event control block (ECB) to determine whether the I/O operation has completed successfully. If not, the subroutine waits ($WAIT IO) for the operation to complete and then returns to the caller. If a read error occurs, the routine issues a $IOERROR macro instruction to log the error before returning to the caller.

Print/Punch Processor Termination Routines

The following routines close and deallocate data sets, free resources used by the print/punch processor, provide device setup support, and provide separator page support.

PPDONE: Print/Punch Processor Termination Routine

If the termination is abnormal, messages are written both to the operator and to the output device (if a printer), giving the reason for termination. If system management facilities (SMF) data recording has been requested, a type 6 SMF record is built in an SMF buffer and the $GETSMFB and $QUESMFB macro instructions are issued to schedule the buffer to be written.

For 3800 printers, PPDONE delays the $QUESMFB so that PPQMGR does not write the SMF record until all output associated with the SMF record has reached the 3800 stacker.

Punch processors punch a blank card to clear the last valid data set card record. Using data from the characteristics JOE as a setup descriptor, all processors call the device setup verification subroutine (PRPUDSV) in preparation for the trailer page on printers. Punch processors also follow this path to ensure correct device setup for the next $#GET request. Prior to the test for a separator page, the separator exit point allows the user to add to or replace the standard separator page via user exit routines. If the user exit routine returns a return code of 0 or 4, JES2 creates a separator page based on the setting of the flags in field DCTPPSWS. If the return code is 8, JES2 does not create a separator page. If the return code is 12, JES2 unconditionally creates a separator page.

For 3800 printers, a mark-forms CCW is optionally issued to mark the perforations of the trailer page. (The mark forms CCW is issued if the DCTNIMRK bit of the DCTPPSW2 flag byte is set on via the PRTnnnn initialization parameter or the $T operator command.) An end-of-transmission CCW is issued by PPDONE causing offset stacking. The-end-of-JOE is indicated in the 3800 pending page queue.

PRPUEXIT: PRPUEXIT issues a $EXTP CLOSE macro instruction for remote devices under normal circumstances. A $EXTP NCLOSE macro instruction (negative close) is issued in abort situations, such as when $C or $E commands are issued. If the $EXTP routines detect abnormal situations, such as a line drop or a remote device disconnected, they set the interrupt flag ($I) in the device control table (DCT). This prevents the print processor from discarding the work and checkpoint JOEs.

For all devices, the JCT is released from the processor. For impact printers if a repeat ($N) command has been issued but could not be honored while printing or punching the data set because of a lack of JOEs, a $#PUT macro instruction is issued to place the work JOE back into the JOT for the requested copy. For impact printers, if an interrupt command ($I) has been issued (or simulated by HASPRTAM), a $#PUT macro instruction is issued to place the work JOE into the JOT.

For non-3800 printers, if neither the $N or $I commands have been issued, PRPUEXIT issues a $#REM macro instruction to remove the completed work JOE. For all devices, the PDEALLOC subroutine releases control blocks and and buffers required for processing. The print/punch processor issues a $DORMANT macro instruction and returns to its primary entry point to select a new job.

PADDPQE: 3800 PQE Allocation Subroutine

PADDPQE allocates PQEs. PADDPQE checks the PQE free queue to see if it's empty. If it is, PADDPQE attempts to get the next PQE extent, and if unsuccessful, PADDPQE gets the next PQE in that extent and again checks for an empty queue. If no free PQE is found in all the extents, PADDPQE issues the CLEAR PRINT CCW to print the page buffer and frees the associated PQEs. Then PADDPQE again checks the PQE free queue for a free PQE. When found, PADDPQE initializes it and chains the PQE to the PQE active queue. PADDPQE then returns to the caller.

PDELPQE: 3800 PQE Deallocation Subroutine

PDELPQE deallocates PQEs. On entry, register 1 contains the address of the first PQE to free and register 0 contains the number of PQEs to free. For the requested amount, PDELPQE unchains PQEs from the PQE active queue and returns them to the PQE free queue. On exit, register 1 contains the address of the active PQE previous to the first freed PQE.

PBLOCK: Block Letter Routine

PBLOCK creates the block letter output on the header and trailer pages. PBLOCK is invoked by the $PBLOCK macro. (User exit routines can also use PBLOCK by invoking it via the $PBLOCK macro.) Up to eight characters can be specified in the $PBLOCK macro for output. PBLOCK scans the input field for the first blank character encountered. If there are no blanks, it takes the first eight characters. These characters can be slanted, straight, centered, or not centered depending upon how the $PBLOCK macro is coded.

SEPPDIR: Remote Setup Header Routine

SEPPDIR is invoked by the $SEPPDIR macro from a user exit routine to build a peripheral data information record (PDIR) for remote SNA devices requiring one (that is, remote SNA devices with spooling capabilities that are being used).

PFRESMFB: Free SMF Buffer Subroutine

PFRESMFB frees the SMF buffers. PFRESMFB, using compare and swap logic, chains the SMF buffer addressed by register 1 to the SMF buffer free queue. PFRESMFB then issues the $POST SMF macro instruction to notify JES2 that the resource is available.

PRPUDSV: Device Setup Verification Subroutine

PRPUDSV ensures that the device controlled by the print/punch processor is set up with the requested forms and, when appropriate, the forms control buffer (FCB) and universal character set (UCS) buffer. The FCB must also include the proper index byte if the device is a 3211 printer.

A parameter list is supplied containing the required setup, which is compared with the DCT that specifies actual physical setup. If a mismatch requires operator intervention, message $HASP190 is issued and processing is halted until the operator indicates otherwise.

Note: If the SETUP=NOHALT parameter was specified on the device initialization statement or set by issuing a $T command for the device, JES2 will not issue the $HASP190 message when encountering a requested output control environment change for that device and will immediately proceed to transmit output to it.

The most obvious response to the $HASP190 message is for the operator to perform the requested setup and then use the start ($S) command to continue processing. Optionally, the operator can override any of the setup requirements before issuing the start command or can suspend processing completely by using the cancel ($C), interrupt ($I), or restart ($E) commands. Next, the UCS is loaded, if supported by the device, and if at least one of the following is true:

- This is the first use of the printer since JES2 was started (and the UCS image ID is not 0).
- A change in UCS identification has occurred.
- The operator has issued a $T command to this device with the operand "T=" since the last UCS loading.
- The last attempt to load the UCS failed.

If the requested UCS image is not in SYS1.IMAGELIB, message $HASP180 is issued, followed by a setup message, and the device is stopped to allow specification of a valid UCSB id.

Because loading of UCS and FCB images from SYS1.IMAGELIB involves many embedded waits that the JES2 main task cannot allow, an image loader subtask is attached by HASPPRPU to perform the loading function.

After UCS loading, an attempt is made to load the FCB and index value for 3211 Printers if any of the following is true:

- This is the first use of the 3211 Printer since JES2 was started.
- A change in FCB identification or index value has occurred.
- The operator has issued a $T command to the device with the operand "C=" since the last FCB loading.
- The last attempt to load the FCB failed.
- The last loading of the FCB was done with an index value different from that contained in the system copy of the FCB image.

If the requested FCB image is not in the library, message $HASP180 is issued (followed by a setup message), and the device is stopped to allow specification of a valid FCB ID. Having completed the above task, PRPUDSV returns control to the calling point in the print/punch processor.

P3800DSV: 3800 Printer Device Setup Verification Subroutine

P3800DSV uses a setup parameter list supplied by the caller to ensure the proper setup status of a 3800 printer allocated to the print processor. The caller-supplied parameters are compared with values in the DCT that specify the actual physical setup. If a mismatch requires operator intervention (a mismatch of forms, flash, or burster status), message $HASP190 is issued, and processing is halted until started or terminated by the operator. P3800DSV issues a display status code order to the 3800. This causes the 3800 display panel to show F1, F2 or F8 and the printer to go into a not-ready status.

Note: If the SETUP=NOHALT parameter was specified on the device initialization statement or set by issuing a $T command for the device, JES2 will not issue the $HASP190 message when encountering a requested output control environment change for that device and will immediately proceed to transmit output to it.

Next, the status indicated by the DCT is compared with the status indicated by the unit control block (UCB) to determine whether SETPRT (SVC 81) is required. If so, the UCB/FCB image loader subtask, HASPIMAG, is used to call SETPRT with a parameter list specifying the necessary changes for the 3800 printer.

If an error occurs during subtask processing, message $HASP151 is issued, identifying the image name and the type of failure that occurred. SETPRT writes the exact text of the $HASP151 message on the printer. JES2 follows this with a $HASP154 message on the printer identifying the device name, job name, job ID and room number.

PQECINIT: 3800 PQEC Initialization

PQECINIT initializes the PQE checkpoint (PQEC). PQECINIT first acquires a PQE by calling the PADDPQE subroutine. PQECINIT then initializes the PQE as a checkpoint PQE (PQEC) by using information from the checkpoint area of the PCE. The PQEC acts as a holding place for checkpoint data. This routine is called on an interval based on the CKPTPGS setting for this data set.

PQEDINIT: 3800 PQED Initialization

PQEDINIT initializes the data set PQE (PQED). PQEDINIT first acquires a PQE by calling the PADDPQE subroutine. PQEDINIT then initializes the PQE as a PQED using information from the checkpoint area of the PCE. The PQED contains checkpoint information that remains constant through the printing of its associated data set.

After initializing the PQED, PQEDINIT calls the PQECINIT subroutine to acquire the PQEC representing the beginning of the data set. Then PQEDINIT calls the PPGIDIO subroutine to obtain the 3800 page ID's associated with the PQEC.

Finally, if the data set is being warm started, PQEDINIT calls the PRECOMP and PADDPQE subroutines to initialize a PQEC representing the warm start location of the JOE.

PPGIDIO: 3800 Page ID I/O Routine

PPGIDIO, called by PQEDINIT, activates the I/O necessary to obtain the 3800 page ID and FCB line position associated with each PQE. This allows JES2 to associate its spool information with the data transmitted to the 3800 but not yet stacked. This I/O is activated by calling the PPPUT subroutine with the following CCWs:

- Request-printer-information order of the execute-control CCW
- TIC CCW
- Sense intermediate buffer CCW

The 3800 page ID and FCB line positions are read into four bytes of storage immediately following the TIC CCW.

PPQMGR: 3800 Pending Page Queue Management Routine

PPQMGR manages 3800 pending page queues. On input, register 0 contains the 3800 page ID of the page currently being stacked on the 3800 printer. The PPQMGR routine compares this stacker page ID against the PQE page IDs to determine which PQEs in the pending page queue have had their associated pages stacked. For these PQEs, the PPQMGR routine performs the following functions:

- For an SMF type-6 PQE (PQES), PPQMGR issues the $QUESMFB macro to write the SMF type-6 record.
- For a checkpoint PQE (PQEC), PPQMGR saves the PQEC address for checkpoint processing. When all stacked PQECs have been processed, PPQMGR calls the PCKPTNI subroutine to issue a checkpoint against the latest stacked PQEC.
- For an end-of-JOE indication, PPQMGR performs the following functions:
  - Processes any deferred repositioning commands, by issuing the $#PUT and $#CHK macro instruction
  - Issues the $#REM macro instruction to remove the JOE from the JOT (unless the $#PUT macro instruction has been issued for a deferred command against the JOE)
- For all PQECs that are not needed for use as a backspace table, PPQMGR calls the PDELPQE subroutine to free the PQEC. PPQMGR maintains the following PQEs for use as a backspace table.
  - PQED of the data set currently being stacked.
  - PQEC representing the beginning of the data set.
  - n number of most recently stacked PQECs at m page id intervals. Where n and m are the values contained in the $BSPNTE and $BSPGT fields of the HASP control table respectively.
  - Before exiting, PPQMGR calls the PCKPTNI subroutine to issue a checkpoint for the most recently stacked PQEC.

PCKPTNI: PPQMGR Checkpoint Subroutine

PCKPTNI updates the checkpoint (CHK) with information from the PQEC and PQED and returns to the caller.

PLOCATE: Locate 3800 Transfer Station Routine

PLOCATE determines which physical page is at the 3800 transfer station or, if the paper is jammed, the 3800 fuser station. It does this by using the request printer information (BFWSENS) data in the buffer work area ($BFW).

PLOCATE also determines whether a repositioning command can be processed (based upon whether the repositioning command falls within the job presently at the 3800 transfer station). If so, the PP3800R bit is set and control is returned to the caller. If not, PLOCATE issues the $HASP152 message indicating that the command is rejected.

PLOCATE also processes $N commands by either issuing the $#ADD macro instruction or by increasing the deferred command count in the 3800 pending page queue.

P3800CMD: 3800 Command Processing

P3800CMD, called by PPDSEND, processes all 3800 repositioning commands. P3800CMD first issues the purge-page-buffer order of the execute-control CCW to purge any data currently in the 3800 page buffer. P3800CMD then determines the specific repositioning command being processed and passes control to one of the following routines.

P3800CEI - 3800 $C, $E, $I Routine: If the command is $C, P3800CEI indicates cancellation in the 3800 pending page queue and issues the $HASP170 message indicating that the printer was deleted.

If the command is $E, the P3800CEI routine attempts to add the JOE to the JOT (via the $#ADD macro). If successful, the $HASP170 message is issued. If unsuccessful, the deferred command count in the 3800 pending page queue is increased and the $HASP170 message is issued.

If the command is $I, P3800CEI indicates interruption of the JOE, increases the deferred command count in the 3800 pending page queue, and P3800CEI issues the $HASP170 message indicating that the printer was interrupted.

PBSPACE - 3800 Backspace Routine: PBSPACE processes the $B command and 3800 paper jams. PBSPACE locates the PQEC previous to either the target page of a $B command, or the fuser page for a paper jam. (All physical pages within a copy group are counted when copy grouping.) PBSPACE finally issues either the $HASP170 message for printer backspaced, or the HASP153 message for paper jam.

PFSPACE - 3800 Forward Space Routine: PFSPACE processes the $F command and the 3800 cancel key. The 3800 cancel key is processed as a $F, D command. PFSPACE first determines the PQEC previous to the target page, and then issues the $HASP170 message indicating that the printer was forward spaced.

All specific command routines exit by passing control to the 3800 reschedule routine (PGETMAPV). PGETMAPV first determines what, if any, FCB mapping requirements exist. Then PGETMAPV calls the PRECOMP routine to adjust 3800 page IDs in the 3800 pending page queue to reflect the repositioning. Next, the current JCT is freed. Then, all JOEs in the 3800 PPQ that have not been printed are rescheduled by issuing the $#PUT macro instruction. Finally, the reschedule routine passes control to the PJOSETUP routine.

PJOSETUP initiates print resumption by causing the job's JCT to be read, by calling the PALLOC subroutine to allocate resources for the job, and finally by setting up the PCE checkpoint area from the target PQE.

3800 command processing concludes with the 3800 PPQ truncation routine (following PRESETAL), which truncates the 3800 pending page queue to reflect repositioning. Control is returned to the caller.

PALLOC: Local Device Buffer Allocation

PALLOC allocates buffers for local devices. Local devices request, through $GETBUF, either one or two JES2 buffers depending upon options chosen during JES2 initialization. For processes that will despool track cells, each request is for a multiple JES2 buffer chain. Additionally, PALLOC requests a print/punch buffer to build printer or punch CCWs. For impact printers and punches, PALLOC uses this buffer to build two CCW chains of lengths determined by the CCWNUM parameter on the PRINTDEF or PUNCHDEF statements.

For a 3800 printer, PALLOC obtains a page buffer to build the two CCW chains. At the end of each 3800 CCW chain, PALLOC builds a NOP CCW, a request printer information order of the execute-control CCW, and a sense intermediate buffer CCW to obtain 3800 information needed for checkpoint determination. PALLOC then initializes and closes the buffer work area to retain the sensed 3800 information.

PALLOC obtains one additional print/punch buffer for building the track-cell input CCWs, if needed.

To facilitate the use of EXCPVR for local devices, PALLOC fixes in main storage all data buffers, and CCW buffers, and the data extent block (DEB) for the duration of the processing of the data sets described by the work JOE.

PDEALLOC: Free Print/Punch Resources

PDEALLOC frees the control blocks and buffers required for print/punch processing that were allocated by PALLOC.

PRECOMP: Recompute 3800 Page ID

PRECOMP adjusts the reposition page ID for all PQEs. This is done to show repositioning or gaps in spool data not reflected by the 3800 paper line on a warm started data set.

PMAPFCB: 3800 Map FCB Routine

PMAPFCB is called from the mainline print loop when 3800 repositioning is needed. Printer repositioning is accomplished by reading forward on the spool to the restart point. These logical pages, however, do not necessarily match the physical 3800 pages. PMAPFCB matches each print line to the FCB of the data set to determine the actual repositioning. When a print line crosses the end of the FCB, the number of pages left to map is decreased. When this page count reaches zero, the repositioning is finished and printing resumes.

PHEADER: Produce Job Header

For impact printers, PHEADER calls PJODMSG to issue the $HASP150 message for the job on the device. For 3800 printers $HASP150 is issued when the start of the job reaches the transfer station.

PHEADER gets setup information from the characteristics JOE and invokes the device setup verification routine, PRPUDSV. (First, for a 3800 printer, forms flashing and copy modification are suppressed, and the character set for the 3800 is forced to the installation default. For a 3211 printer, the index value is forced to one.) On return from setup verification, the separator exit point PTESTSEP (for exit 1) allows the installation to add to or replace the standard separator page via installation exit routines. If the installation exit routine returns a return code of 0 or 4, JES2 creates a separator page based on the flag settings in field DCTPPSWS. If the return code is 8, JES2 does not create a separator page. If the return code is 12, JES2 unconditionally creates a separator page. Depending upon device type, PHEADER invokes either PUNCHSEP to produce a separator card, or PRINTSEP to produce a separator page.

PJODMSG: Issue Print/Punch Job Sign-on Message

PJODMSG issues message $HASP150 to provide the operator with the output job name, the output device name, and the number of lines, or pages, or cards about to be processed. For impact printers and punches, PJODMSG issues this message at job selection time. For 3800 printers, PJODMSG issues the message when the job first appears at the transfer station.

PQECOMP: 3800 PQE Completion

PQECOMP is called out of PPCHECK to complete any PQEs assigned during the building of a CCW area. At checkpoint intervals and at the beginning and end-of-data, PQECOMP completes PQEs in the pending page queue. Most of the data kept in the PQE is often from existing JES2 control blocks. However, PQECOMP retrieves the FCB line index and channel page ID from the hardware. PPEIDIO places request printer information and sense intermediate buffer CCWs in the CCW area when the PQE is created. At channel end or program control interrupt (PCI), the information needed is in the CCW area and PQECOMP moves it to the PQE to complete initialization.

PQECOMP also chains all PQEs for job start into the page queue header. If any PQEJs are at the transfer station, PQECOMP calls PJODMSG to issue the $HASP150 message. Finally PQECOMP calls PPQMGR to process any stacked ids.

PRPURCV0: Print/Punch Processor Recovery Routine

This routine performs the error recovery processing for the print/punch processor. The $ESTAE environment is established (via a $ESTAE macro) at label PGOTPDDB and cancelled (via a $ESTAE macro) at label PESTCAN2. This routine attempts recovery from program checks. It simulates a $I command to interrupt the I/O at the point of the error and requeues the output to recover from the error. This is done for the JOE being processed at the time of the error. PRPURCV0 also marks the JOE SELECT=NO and returns the JOE to the JOT.

For program check errors, PRPURCU0 distinguishes between a non-3800 and 3800 device. For a non-3800 device, the ESTAE is cancelled at PESTCONT and output processing flushes the punch or terminates the printer listing. For a 3800, PESTCNZR is the resume address. At PESTCNZR, PRPURCV0 continues recovery by calling a second level recovery routine at PESTREC. PESTREC simulates a $I (via $HASP197) and sets the corresponding JOE non-selectable. Processor termination continues.

HASPIMAG: UCS/FCB Image Loader Subtask

The image loader subtask is part of the HASPPRPU assembler module. Entry point HASPIMAG is identified to the operating system and attached by HASPINIT. It establishes $STABEND as its ESTAE routine. When attached, the image loader uses the IMGLIB system macro instruction to open SYS1.IMAGELIB. It returns a successful completion code to HASPINIT and waits to be posted with a JES2 buffer address. When the image loader subtask is posted, the post code is assumed to be either 0 or a JES2 buffer address. A 0 implies a requested shutdown and causes the subtask to issue IMGLIB to close SYS1.IMAGELIB and to exit. If the post code is a buffer address, the address points to a JES2 buffer that contains a list suitable for use by a BLDL macro (issued by the subtask), which locates and verifies an image on SYS1.IMAGELIB.

If the BLDL is not successful, an abnormal return is given to the posting JES2 processor, and the image loader subtask waits for another post. If the BLDL is successful, the subtask uses the MVS LOAD macro instruction to cause the requested UCS or FCB image to be loaded into storage from SYS1.IMAGELIB. The requested image is copied from the loaded module into the JES2 buffer and then deleted from the load location through a DELETE macro.

In the case of the 4245, the image name prefix passed in the buffer equals 'UCSS' or 'UCSG'. The SETPRT macro is invoked to access and verify the image. The BLDL is not issued.

A successful completion code is moved into the JES2 buffer, and the posting JES2 processor is posted. The image loader subtask then waits for another post.

Support is provided for the 3800 Printing Subsystem to communicate with the SETPRT (SVC 81) function of MVS.

For 3800 repositioning, HASPIMAG loads the FCB image into the caller's buffer for FCB mapping purposes.

The HASPIMAG ESTAE routine will close and reopen SYS1.IMAGELIB and retry the failing load. This is to handle the case for impact printers accessing images on extents added since SYS1.IMAGELIB was originally opened. If errors persist, the ESTAE routine will issue a $$WTO macro instruction and post HASPPRPU with an error return code.

Note: The DEVFCB= parameter on the PRTnnnn and Rm.PRn initialization statements (also $T-able) indicates the device default FCB value (DEVFCB), which is the sole FCB value to be used in the event that the output element has not explicitly declared "FCB=" on the JCL.

HASPHOPE: Output Processor

System output consists of operator console messages, job statistics, messages from the operating system, and data sets written by the program. Operator console messages are saved in JES2-defined data sets by WTO/WTOR JES2 communications. Job statistics are maintained and updated in the job control table (JCT) by various JES2 processors. Messages from the operating system are written directly on a spool volume just as any other data set. Data sets to be processed as output are placed on a JES2 spool volume by the HASCJBST module during job execution.

There are 36 classes of system output permitted in the operating system. The JES2 output processor maintains an output queue that corresponds to each class for local output and remote output, plus a single network output queue for all data that is routed to other nodes. The programmer defines a system output data set by using the DD statement or the JCL OUTPUT statement. The programmer specifies the class to which the system data set belongs in the parameter associated with the SYSOUT keyword or the CLASS= operand on the JCL OUTPUT statement. The HASCDSAL module assigns a peripheral data definition block (PDDB) to each SYSOUT data set, as well as one to the console messages and operating system messages pertaining to the job. At termination, the job is transferred from the execution queue to the output queue for analysis by the output processor.

The task of the output processor is to analyze the PDDBs built for the job during execution and to build a set of job output elements (JOEs) that represent unique print/punch requirements. The JOEs are placed in the job output table (JOT), which contains all output requirements currently available to be processed by JES2 print/punch processors and JES2 SYSOUT transmitters.

To ensure that the integrity of the job queue and the JOT is maintained when multiple systems are active in a multi-access spool environment, the $QSUSE macro is issued prior to any change or reference to either table.

HASPHOPE: Output Processor Main Entry Point

At its entry point (OPINIT), HASPHOPE acquires access to checkpoint data (via $QSUSE) and checks $FLAG1 in the HCT to see if any spun data sets still needs to be processed because of a full JOT ($UNSPUN in $FLAG1 is set to 1 or $PRUNSP in $FLAG1 is set to 1). If spun data sets still need to be processed, HASPHOPE invokes OPSPIN to process the work. If no unspun work exists, the $QGET macro is used to search the JES2 job queue for new work from the $OUTPUT queue. If no job is available, the $WAIT macro is used to release control until the status of some job in the queue is unchanged. If work exists, HASPHOPE invokes OPMAIN to process the work.

OPSPIN: HASPHOPE activates itself via $ACTIVE and gets a buffer for a spun IOT if it is not already available. HASPHOPE then scans the JQEs checking for unspun IOTs. For each unspun IOT, HASPHOPE builds (via $#BLD) a prototype JOE pair from the PDDB in the spin IOT and adds (via $#ADD) the JOEs to the JOT. If the JOEs are successfully added to the JOT, HASPHOPE allocates a spool record (via $#ALCHK) and writes the IOT. Then HASPHOPE checkpoints the JQE.

If the JOEs are not successfully added, HASPHOPE tries the process later. HASPHOPE processes some special cases. If HASPHOPE reads an IOT for a job that has been cancelled by the operator, the job is moved to the $PURGE queue. If the job was in $HARDCPY and if the job has no more JOEs or held data sets, HASPHOPE continues the processing of unspun data sets until either the JOT becomes full or until the spun data sets have all been processed.

When the JOT is full or no spun data sets exist, HASPHOPE frees the spin IOT buffer and attempts to get a job to process its output.

OPMAIN: Having obtained a job, the output processor issues a $ACTIVE macro instruction to indicate to the dispatcher that JES2 is not dormant and a $TIME macro instruction to get a starting time and date for the output processor.

The $QGET macro returns the address of the JQE if a job is available for output processing. All references to job data by the output processor are made through the JQE, which contains the track address (on a spool volume) of the job control table (JCT). The job lock is obtained via ($GETLOK). This routine issues the $JCTIO macro instruction to read the JCT from the spool volume into a JES2 buffer. ($FREEBUF frees the buffer when no longer needed.)

A check is made to ensure that the data read from the spool volume is a valid JCT and belongs to the job being processed; if not, the routine exits to the OPNOJCT routine for error processing.

At the normal completion of JCT checks, the job level copy count, message class, and default job forms ID are copied from the JCT to the processor control element (PCE) used as a work area. The JCT contains the track address of the first input/output table (IOT) that contains the PDDBs. The IOTs are chained, each containing the track address of the next IOT in the chain.

If the NOTIFY=userid keyword was specified on the JOB statement for a user on this node, an attempt is made to locate an active time-sharing user of the ID specified. If the user is found and is busy on another system within a multi-access spool configuration, the job's system affinity is altered to match the system on which the user is active, and the job is returned to the $OUTPUT queue.

The TSO SEND command is used to display a job-ended message on the user terminal if the NOTIFY=userid is on this node. If the NOTIFY=userid is on another node, the JES2 remote console processor is used to send a job termination message back to the user on the originating node. Exit point OPNEXIT (for exit 16) is taken to allow an installation exit routine to modify the notification messages, replace them with different ones, or bypass issuing certain messages.

If the userid is not active in the MAS, either the job system's affinity is altered to be that of the system the job originated on (OUTDEF BRODCAST=NO), or the NOTIFY is issued to the shared broadcast data set from this system (BRODCAST=YES).

OPNOTX: Buffers are obtained and all IOTs for the job are read into buffers before the scan of PDDBs begins. When the amount of buffer storage is inadequate and attempts to get move are unsuccessful, a $DISTERR macro is issued to alert the operator to the problem (using the $HASP183 message). The IOTs read into the JES2 buffers are checked for validity. If the validity check for any IOT fails, a $DISTERR macro instruction is issued to alert the operator, and processing is continued if at least one IOT was valid. Absence of any valid IOTs results in the same processing path as was taken for an invalid JCT. The TGMSPMSK mask in the track group map (TGM) portion of the allocation IOT indicates which spools this job has space allocated on. This mask is logically ORed into the JQE spools-used mask so that the JQE has a record of the spools this job is using.

After all valid IOTs have been read, JES2 checks for conditionally-purgeable output. If the job executed on this node and has completed normally, any PDDB that is destined to print on this node and has a conditionally-purgeable output class will be marked as not containing SYSOUT so that the data set will not be printed. Next, the job class is checked to see if it is a conditionally-purgeable job class. If it is, and the job has completed normally, and there are no non-empty, non-spin data sets other than system data sets, this job will be conditionally purged.

OPSCAN: Two prototype JOEs are built for the first PDDB that does not have the NULL flag set. The first JOE (work JOE) contains routing information and SYSOUT class, and the second JOE (characteristics JOE) describes the device setup such as forms, FCB, UCS flash frame, and external writer IDs, as well as the burst specification necessary to process this PDDB. The $#ADD macro instruction is used to add the work prototype JOE to the JOT in the SYSOUT class queue specified in the PDDB (if the JOE routing is for the local node) or in the network queue (if the JOE routing is for another node). The $#ADD macro instruction also adds the SYSOUT characteristics JOE to the characteristics JOE queue. Each $#ADD macro instruction on behalf of the job is done "job copy" times to allow parallel processing by the print/punch processors. If the $#ADD macro indicates that the JOT currently has no available space for insertion of JOEs, the routine waits ($WAIT JOE) until space becomes available. A spool record is allocated (for output checkpointing) via a call to the $#CHK service routine for each work JOE that is added to the JOT.

Having added this processing requirement to the JOT, the routine sets the NULL flag in the PDDB along with the NULL flags in all subsequent PDDBs that represent the similar class, writer name, PRMODE, route, and setup characteristics. The first PDDB that does not meet the above test is used as a restart point for the next prototype JOE built and $#ADD macro instruction. When all PDDBs have been set to NULL, the JCT for the job is updated on the spool volume by the addition of output processor start/stop times.

OPJLOG: The output processor calls OPJLOG when processing the JES2 job log data set. If job statistics and messages are outstanding from earlier phases, OPJLOG adds these to the JES2 job log using data set services.

PDBCRTHD: The output processor calls PDBCRTHD when processing a held data set. PDBCRTHD creates a held PDDB for the held data set. It also puts creation data (the time of creation and status of the held data set) for the held data set into the PDDB in the IOT buffer and increases the OPHLDCT held data set count field.

OPWRTR: After all other output processing is done, OPWRTR (while under the job lock) writes to the spool the chain of IOTs created during output processing. Then it sets the hold count in the JQE (JQEHLDCT) to the number of held data sets. Once the write of the IOTs is completed, the JQE is checkpointed.

OPNOJCT: As each JOE is added to the queue, it is also chained to the JQE. During processor termination, the job is placed in the $PURGE queue if the job has no JOEs and the hold counter is 0; otherwise, the job is placed in the $HARDCPY queue while awaiting print/punch processing or SYSOUT transmission. Finally, a $FREEBUF macro instruction is issued to free each IOT buffer used during JOE creation, $DORMANT is issued to reduce the active processor count, and the routine branches OPINIT to await further work.

HASPFSSP: JES2 - Functional Subsystem Service Processor

The functional subsystem service processor:

- Coordinates the initiation and termination of the functional subsystem (FSS).
- Services requests for output groups (as made via GETDS and RELDS in HASPFSSM).
- Processes operator commands directed to devices associated with the FSS.
- Communicates orders to the FSS.

Device PCEs associated with functional subsystem address spaces execute code in HASPFSSP. When a printer runs in FSS-mode, the entry point in the PCE points to HASPFSSP and the FSS-mode flag is set in the printer DCT.

Initialization and Dispatching

HASPFSSP initialization receives control at HA$PFSSP, on the first dispatch of a PCE associated with a FSS-mode printer device that is running in page-mode. Its other entry point, DYNFSS, receives control to look for an FSSCB matching the FSSNAME that was passed.

HA$PFSSP - Initial Entry Point

HASPFSSP establishes global addressability to the functional subsystem control block (FSSCB) and the functional subsystem application control block (FSACB) and establishes temporary addressability to the printer DCT, pointed to by the PCE. The PCE work area and the PCE's JQE indicator are set to zero.

HASPFSSP next checks to see if an FSACB already exists; this indicates whether a hot start of JES2 occurred while a functional subsystem application (FSA) was running. If a JES2 hot start did occur, HASPFSSP determines if one of the following conditions exist:

- An FSS-level order is outstanding. The FSSXB parameter list is checked to see if this PCE issued the order. If this PCE issued the order, HASPFSSP returns the contents of the FSWORDID field in the FSSXB parameter list of the PCE work area.
- An FSA-level order is outstanding. The FSA order parameter list in the FSAXB is returned.
- The device had been started. The current set up for the device is restored to the DCT using information from the FSAXB.

If JOEs have not as yet been assigned to the FSA, HASPFSSP issues a $GETUNIT to assign the DCT to the FSA device and a $ACTIVE macro to indicate that the functional subsystem service processor PCE is active. If the FSA device was not started and a stop FSA order is not outstanding, HASPFSSP sets the DCTSTART bit on in the DCT to indicate that the device needs to be started.

If a hot start of JES2 did not occur while an FSA was running (PCEFSACB = 0), then HASPFSSP checks whether the device is already in use, draining, in hold status, or is paused. If any of these conditions exists, HASPFSSP enters a $WAIT on UNIT loop until the device associated with the DCT is available. If none of the device conditions are present, HASPFSSP sets the DCTFLAGS byte to 0 and begins the device processing.

FSPMAIN - Dispatching Loop: FSPMAIN identifies the beginning of a loop in HASPFSSP that controls the processing for the device. (FSPMAIN is also the retry point set up by the ESTAE for HASPFSSP. The sequence of processing in this loop is as follows:

1. A test is made to determine if the system is draining; if it is, the DCTSTART bit is set off in the DCTFSSFL field to prevent device starts. If the system is not draining, a $CALL macro is issued to invoke FSPORDER, which processes commands and orders.
2. On return from command or order processing performed by FSPORDER, a check is made to determine if the FSACB exists; if it does not exist, a $WAIT on FSS or FSSXECB is issued. When the wait is posted, processing starts at the beginning of the loop at FSPMAIN. If the FSACB exists but the device is not yet started, a $WAIT on JOT or FSAXECB is issued. When the wait is posted, processing again begins at the beginning of the loop at FSPMAIN.
3. If the FSACB exists and the device is started, then a $CALL is issued to invoke FSPRELDS to process any pending FSA RELDS requests associated with the device.
4. A check is made to determine if the device is draining; that is, if the FSAQUIES bit is on in FSAFLAG1. If the device is draining, a $CALL is issued to invoke FSPORDER to attempt to stop the device and a $WAIT on JOT or FSAXECB is issued. When the wait is posted, processing begins again at the beginning of the loop at FSPMAIN.
5. If the device is not draining, then a $CALL is issued to invoke FSPGETDS to process FSA GETDS requests. Upon return from FSPGETDS, a $WAIT on JOT is issued. When the wait is posted, processing begins again at the beginning of the loop at FSPMAIN.

FSPORDER - Command/Order Management: FSPORDER determines if any command processing is necessary on behalf of the functional subsystem. Processing is required if there exists a response to a previously issued order, if a command has been issued against a device that requires an order to be issued, or if the functional subsystem needs to be started.

FSPORDR0, FSPORDR1 - Response Check Processing: At these labels, HASPFSSP checks whether there are any outstanding responses to FSS orders that need to be processed. At FSPERRCK, HASPFSSP checks whether the FSS/FSA abended or abnormally disconnected.

Response processing takes place when functional subsystem orders have been issued and responses to them are outstanding. FSPORDER checks the FSSOROUT bit in FSSFLAG1 flag byte; if the bit is on, a response to an FSS-level order is outstanding. FSPORDER issues $CALL to invoke FRSPORDR to process the response. If the return code from FRSPORDR is 0, FSPORDER checks for more eligible orders to process; if the return code is not 0, FSPORDER stores register 6, containing the FSACB address, in the PCE current save area, restores the caller's registers, and returns to the caller.

FSPSORDR - FSS Order Processing: If FSPSORDR determines that FSS-level orders are required, FSPSORDR passes control to the appropriate routine to issue the order.

FSPAORDR - FSA Order Processing: If operator commands have been issued against the device, FSPAORDR determines if any FSA-level orders are required and passes control to the appropriate routine to issue the order.

FSPSTFSS - Start FSS Processing: FSPSTFSS is invoked as a result of issuing the JES2 start command to start the device; if the FSS is not already active, the start command is prepared for the FSS and subtasked by the FSMGCR routine. The processor (HASPFSSP) is then marked as active.

FSPSTFSA - Start FSA Processing: FSPSTFSA prepares the parameter list for the start FSA order and invokes FSPORCMS to issue the order. An appropriate form of the $HASP700 message is issued for errors in this start processing.

FSPSTDVC - Start Device Processing: FSPSTDVC prepares a parameter list for the start device order and invokes FSPORCMS to issue the order If FSPORCMS processing was successful, FSPSTDVC initializes the current device defaults and device settings in the FSAXB. If FSPORCMS processing is not successful, FSPSTDVC indicates that the device is not started and processing is initiated to stop the FSA. An appropriate form of the $HASP700 message is issued for errors in this start processing.

FSPSPDVC - Stop Device Processing: FSPSPDVC prepares a parameter list for the stop device order and invokes FSPORCMS to issue the order. HASPFSSP waits for all data sets for the current JOE to finish printing before the stop device order has its effect. HASPFSSM indicates that all the data sets have finished printing for a specific JOE in its GETDS routine. An appropriate form of the $HASP701 is issued for errors in this stop processing.

FSPSPFSA - Stop FSA Processing: FSPSPFSA prepares a parameter list for the stop FSA order and invokes FSPORCMS to issue the order. If the order is successfully processed, FSPSPFSA issues a $STIMER to start the timer to ensure that the FSA disconnects in a certain amount of time. If the order is not successful or the timer expires, FSPSPFSA terminates stop processing. An appropriate form of the $HASP701 message is issued for errors in this stop processing.

FSPSPFSS - Stop FSS Processing: FSPSPFSS prepares a parameter list for the stop FSS order and invokes FSPORCMS to issue the order. If the stop order is successful, FSPSPFSS issues a $STIMER to start the timer and ensure that the functional subsystem disconnects within a given interval of time. If the stop order fails or the timer expires, FSPSPFSS turns off the functional subsystem drain bit (FSSDRAIN) in the FSSCB and issues a diagnostic message. An appropriate form of the $HASP702 message is issued for errors in this stop processing.

FSPRPTDV - Query Order Processing (for $N): FSPRPTDV prepares a parameter list for the query order and invokes FSPORCMS to issue the order.

FSPSETDV - Set Order Processing (for $T): FSPSETDV prepares a parameter list for the set order and invokes FSPORCMS to issue the order.

FSPOIRDV - Operator Intervention Order Processing: FSPOIRDV prepares a parameter list for the operator intervention order and invokes FSPORCMS to issue the order.

FSPBKFDS - Backspace/Forward Space Synch Order Processing: FSPBKFDS prepares a parameter list for the synch order to backspace or forward space the FSA device and invokes FSPORCMS to issue the order. An appropriate form of the $HASP152 message is issued if an error occurs.

FSPCEI - $C, $E, $I Synch Order Processing: FSPCEI prepares a parameter list for the $C, $E, or $I synch order and invokes FSPORCMS to issue the order. An appropriate form of the $HASP152 message is issued if an error occurs.

FSPHALTD - Halt Device Synch Order Processing: FSPHALTD prepares a parameter list for the synch order to halt the FSA device and invokes FSPORCMS to issue the order. The FSACB is marked to indicate that the device is halted; this causes any GETREC in HASPFSSM to enter a wait state.

FSPCANJB - Purge Output Synch Order Processing: FSPCANJB prepares a parameter list for the synch order to purge output if a $CJ, P command was issued and output was returned in the most recent GETDS request. FSPCANJB invokes FSPORCMS to issue the order.

FSPORCMS - FSIREQ Cross Memory Services: FSPORCMS invokes cross memory services for FSIREQ REQUEST=ORDER requests on behalf of the various order processing functions of HASPFSSP; by issuing the FSIREQ macro, FSPORCMS invokes (via cross memory services) a corresponding order processing routine that receives control in HASPFSSM. If the FSIREQ fails, then the $HASP703 message is issued to explain the error condition.

FRSPORDR - Response Processing: FRSPORDR invokes the proper response processing routine to process responses that come in for outstanding orders. FRSPORDR uses the order id associated with the order to locate the response routines in a table of addresses. The following response routines are used:

- FRSPPFSS - for a stop functional subsystem order
- FRSPSFSA - for a start functional subsystem application
- FRSPPFSA - for a stop functional subsystem application
- FRSPSDEV - for a start device order
- FRSPPDEV - for a stop device order
- FRSPQERY - for a query order
- FRSPSET - for a set order
- FRSPSYNC - for a synch order
- FRSPOPIN - for an operator intervention order

FSPGETDS - Get Data Set (GETDS) Processing: FSPGETDS attempts to select a JOE from the JES2 JOT and pass it to the FSA via the JIB; FSPGETDS does this for each JIB on the FSA's request stack. After selecting a JOE, FSPGETDS marks the JOE as allocated to an FSA.

FSPRELDS - Release Data Set (RELDS) Processing: FSPRELDS processes the JOE associated with each JIB on the FSA's return stack. FSPRELDS returns the JIB to the functional subsystem's free JIB stack. FSPRELDS purges the JOE if the JIB is complete or cancelled; otherwise, FSPRELDS releases the JOE in the JOT for use later (when processing starts from a point that is checkpointed by the FSA).

FSPPOST - Post FSA Task Processing: FSPPOST allows a functional subsystem service processor, running in the JES2 address space to post the FSA task in HASPFSSM, running in the functional subsystem address space. If the FSS has abended, FSPPOST issues a $GETASCB macro to obtain the ASID of the functional subsystem address space and issues a $XMPOST macro to post the FSA task.

Specifically, FSPPOST checks to determine if the FSA is active or if it has terminated. If the FSA is active, FSPPOST invokes cross memory services (PC) to invoke HASPFSSM. HASPFSSM issues the FSIREQ TYPE=POST macro. If the FSA has terminated, FSSPPOST performs a cross memory post of the appropriate ECB to signal that disconnect processing in HASPFSSM should finish. (For more detail see FSMCONCT/FCNDISCN in HASPFSSM.)

FSPFAILS - Start FSS Failed: FSPFAILS issues the $HASP700 error message indicating the nature of the failure.

FSTOFAIL - Start FSA Failed: FSTOFAIL dechains the FSACB, frees the FSACB storage, and indicates that the device was not started; FSTOFAIL also issues $HASP700.

FSTDFAIL - Start Device Failed: FSTDFAIL initiates processing to stop the FSA and issues the $HASP700 error message.

FPAFAIL - Stop FSA Failed: FPAFAIL terminates stop processing and issues the $HASP701 and $HASP702 error messages.

FPSFAIL - Stop FSS Failed: FPSFAIL turns off the FSS drain bit and issues the $HASP701 and $HASP702 error messages.

DYNFSS

DYNFSS looks for an FSSCB that matches the name of the FSS that was passed. If one is found, its address is returned to the caller. If one is not found and allocation of the new FSSCBs is allowed (it is not allowed on a hot start), a new FSSCB is allocated. The new FSSCB is formatted and inserted alphabetically into the FSSCB chain, and its address returned to the caller.

HASPFSSM: JES2 - Functional Subsystem Support Module

HASPFSSM includes support for the functional subsystem interface (FSI) routines. The following HASPFSSM routines support the FSI:

- FSMCONCT/FCNDISCN - CONNECT/DISCONNECT

  Establishes and terminates the connection between the functional subsystem application (FSA) and JES2.
- FSMGETDS - GETDS

  Handles requests for data sets made by the functional subsystem application (FSA).
- FSMRELDS - RELDS

  Handles termination of processing of output data sets by the FSA.
- FSMGETRC - GETREC

  Handles FSA retrieval requests for records from output data sets.
- FSMFRERC - FREEREC

  Frees storage occupied for the records obtained via FSA GETREC requests.
- FSMCHKPT - CHECKPOINT

  Handles FSI checkpoint requests.
- FSMORDER - ORDER

  Processes operator commands that are directed to the FSA.
- FSMPOST - POST

  Asynchronously informs the FSA that its GETDS request has been satisfied.
- FSMSEND - SEND

  Handles communication between JES2 and the FSA and functional subsystem.

HASPFSSM also includes the following resource management routines that are necessary to efficiently use resources that are used between JES2 and the functional subsystem:

- Quick cell management (FSMGETQC, FSMFREQC, FSMBLDQC, FSMQCT, FSMGTBLK, FSMRTBLK)
- Save area management (FSMSAVE, FSMRETRN)
- I/O Services (FSMCBSET, FSMCBIO, FSMCBCK, FSMBTG)
- Initializing and using I/O buffers for spool reads and writes (FSMCBSET, FSMCBIO)
- Completing I/O when reading and writing to spool (FSMCBCK)
- Handling bad track groups (FSMBTG)
- Miscellaneous support that includes:
  - Issuing catastrophic error messages and abends (FSMCATER)
  - Accessing FSI control blocks (FSMFSLNK)
  - Obtaining and freeing the MVS local and CMS locks (FSMGETLK, FSMFRELK)

- Issuing SJF termination request to free SJF storage and perform cleanup (FSMSJTER)
- Issuing the $HASP150 message when the FSA indicates that the data set is apparent on the device (FSMSM150)

FSMCONCT/FCNDISCN - CONNECT/DISCONNECT Service Routine

The connect service routine establishes the FSI between the functional subsystem address space and JES2. The FSA issues the FSIREQ REQUEST=FSICON macro to begin the connection process. The FSIREQ macro expands to an SSI call with a function code of 53 and is directed to the subsystem named in the function dependent area of the FSI parameter list. As a result, SSIFSCNT in HASCSIRQ receives control. SSIFSCNT loads HASPFSSM and branches to the connect service routine FSMCONCT, in HASPFSSM. FSMCONCT then checks whether the request is for a disconnect or a connect.

For a connect and disconnect request, FSMCONCT acquires storage (via GETMAIN) for a save area and various parameter lists for use by disconnect processing. If the GETMAIN fails, an error return code is set in register 15 and return is made to HASCSIRQ. If the storage is acquired, FSMCONCT checks whether this is a connect request for a functional subsystem application (FSA) or for a functional subsystem (FSS).

For a FSS connect, FSMCONCT acquires storage (subpool 230) for a functional subsystem vector table (FSVT), functional subsystem control tables (FSCTs), functional subsystem extension block (FSSXB), entry descriptor table (ETD), data control block (DCB), and data extent block (DEB); these control areas are then initialized. Then FSMCONCT initializes the exit information table (XIT) with entries corresponding to the defined exit points that exist within HASPFSSM and initializes the queue control tables (QCTs) and their associated free cell stacks. FSMCONCT then initializes the functional subsystem cross memory environment.

When it initializes the cross memory environment for functional subsystem connect processing, HASPFSSM:

- Builds an entry table (using ETCON) in the functional subsystem address space.
- Reserves an authorization index (using AXRES) for the functional subsystem address, which JES2 uses to allow the the functional subsystem to program transfer (PT) to JES2.
- Sets the functional subsystem authorization index (using AXSET).
- Reserves a system level linkage index (using XRES) for the functional subsystem address space so that JES2 can program call (PC) to the functional subsystem address space.
- Creates and initializes the entry table with JES2 PC routines (using ATSET).
- Establishes authority to allow JES2 to SSAR to the functional subsystem address space.

HASPFSSM then returns to HASCSIRQ, with the response pending bit off in FSSFLAG1.

For an FSA connect, FSMCONCT searches the FSACB chain for the FSACB that has an FSAID equal to FSIFSAID. Register 6 will contain the address of the FSACB if one is found or zero if no FSACB is found. Then, for both the FSA connect or FSS connect, FSMCONCT issues the $TRACE macro (id=16) to trace this connect.

FSA connect proceeds at FCNFSA. FCNFSVT is invoked to verify whether or not the FSA is connected. FCNFSVT issues a $GETASCB TYPE=HOME to acquire the address of the FSVT (register 10) and FSVT entry for this FSA (register 3). FCNFSA then acquires storage (via GETMAIN) from subpool 230 for the FSA's FSCT, the JES2 FSCT and FSAXB, zeroes the acquired storage and

- Initializes the FSA FSCT, copying FSI routine addresses supplied by the FSS
- Initializes the JES2 FSCT, copying FSI routine addresses from the HFCT
- Stores the address of the FSACB in the JES2 FSCT
- Initializes the functional subsystem application extension control block (FSAXB)
- Turns off the response pending bit in the FSSCB

HASPFSSM then posts HASPFSSP (using FSSEDECB) and returns to the caller (HASCSIRQ).

For a disconnect request, FSMCONCT acquires a disconnect parameter list and branches to FCNDISCN in HASPFSSM. FCNDISCN checks whether the request is to disconnect the FSS or FSA. For a FSS disconnect, FCNDISCN, using the disconnect parameter list, disconnects all FSAs (via FSIREQ TARGET=JES, REQUEST=FSIDCON) that are still connected to the FSS. Then FCNDISCN uses the extended ECB in the FSSCB (FSSXECB) and calls the post JES2 routine (FSMPSTJ2) to post JES2; (this ensures that HASPFSSP is re-dispatched if it is expecting the FSS disconnect.) If JES2 is not ready to receive the disconnect request (the FSSDCONX bit is off in the FSSFLAG3 byte of the FSSXECB), FCNDISCN issues a WAIT macro against the FSSEDECB to wait until JES2 is ready.

When JES2 is ready to accept the disconnect request (FSSDCONX is on in FSSFLAG3), FCNDISCN proceeds at FCNERDC2 to terminate the cross-memory environment, destroying the entry table for the functional subsystem address space and freeing the authorization index and linkage index. Then, the quick cell pools and their QCTs are freed along with the FSVT, the FSCT for the functional subsystem, the FSCT for JES2, the FSSXB, and the ETD. Finally, the return code for the disconnect is set in SSOBRETN, the response pending bit is set off in FSSFLAG1 to indicate a successful disconnect, and return is made to HASCSIRQ to free the HASPFSSM storage.

FSMGETDS - GETDS

GETDS processing handles requests for data sets that are made by the functional subsystem application (FSA) that is running in the functional subsystem address space. When the FSA issues the FSIREQ FUNCTION=GETDS, the GETDS routine at FSMGETDS in HASPFSSM receives control (via the $FSILINK entry service routine) to process the request.

FSMGETDS issues a $GETLOCK to obtain the local lock to serialize the active queue between GETDS and RELDS and then scans the active JIB queue that is chained from the FSACB via the FSAACTQS field. FSMGETDS finds the JIB that exists just in front of a completed JIB or a cancelled JIB. FSMGETDS then issues $FRELOK to free the local lock; at this point, register 4 contains the address of the last processed or cancelled JIB (if any) and register 5 contains the address of the eligible JIB (if any). The eligible JIB is then processed.

FSMGETDS checks the eligible JIB to see if it has been initialized. If the JIB has not been initialized, FSMGETDS further checks to see if the device being processed by the FSA is to be drained; if it is to be drained, the JIBs on the request queue are placed on the free queue via the use of a $RETBLK macro. The FSACB is marked to indicate that a stop device may be issued. If no JIB is already assigned to the FSS, the active JIB queue (FSAACTQS) is cleared by calling FSMJIBRT for each JIB, and the FSA is marked as drained by turning on the FSADRAIN bit in the FSAFLAG1 of the FSACB. (Otherwise, RELDS processing clears the active JIB queue.) FSMGETDS then cross memory posts the JES2 main task and the functional subsystem is told to wait for a return post from JES2, indicating that there is work to do.

If the device is not draining and the JIB is not yet initialized, FSMGETDS initializes the JIB and the spool DEB. FSMGETDS issues a $BUFIO to read the JCT and IOT. The job name, the job ID, and the address of the JMR area from the JCT are set in the JIB. If the $JOECKV flag indicates that a valid checkpoint record exists for the JOE, another $BUFIO is issued to read the checkpoint record. FSMGETDS then prepares to invoke the JSPA modification exit (exit 23), to allow the installation to modify the JSPA.

The exit point JSPAXIT for the JSPA modification exit (exit 23) is then taken. If JOE separator pages are required (as indicated by the return code from the installation exit) the FSAFJSPG flag is set to 1 in the FSAFLAG2 byte of the FSACB; the JESNEWS data set is not printed if a job separator page is not requested. FSMGETDS then issues a $CALL for FSMFINDP to find the first PDDB to assign from the JOE.

Data set assignment occurs at label FGDS200 in FSMGETDS processing. FSMGETDS:

- Merges data set characteristics from the characteristics JOE into the PDDB (Data set characteristics can be modified via the $TO command.)
- Checks data set characteristics against the current device setup and determines if setup processing is required.
- Allocates the selected data set, acquiring and initializing the GETREC related control blocks (GCB, ACB, DEB, an SDB)
- Performs a "fake open" for the data set, and acquires and chains the required protected I/O buffers for despooling.
- Reads the SWB chain for the data set and constructs a SWB list and chains it to the GETDS parameter list.

  Note: A catastrophic error X'F04' can occur during the updating of the SWB list, specifically during the building of a temporary keylist table when SJF extract services return an error return code.
- Builds the RPL chains and the GETREC index tables and issues a GET macro for data set input.
- Locates the next PDDB (by invoking FSMFINDP) in anticipation of the next GETDS call.
- Verifies the printer's authority to select the data set and saves security token information for later authority verification by PSF. (The PRTAUTH routine issues a SAF call using the data set token from the PDDB.)

- Fills in the GETDS parameter list with values that are to be returned to the FSA including a unique data set identifier.
- Determines whether or not another JOE is to be requested of the JES2 main task. If a JOE is to be requested, FSMGETDS queues a JIB request to the FSA request queue (FSAREQQS).

Finally, FSMGETDS returns to the caller, indicating whether a data set has been assigned.

FSMRELDS - RELDS Service Routine

FSMRELDS in HASPFSSM supports the FSI RELDS function as related to the functional subsystem. The basic function of FSMRELDS is to release storage resources that are associated with a data set that was previously assigned via the FSI GETDS function. FSMRELDS:

- Locates the JIB and GCB control blocks corresponding to the data set that is to be released.
- If the data set is being processed as an incomplete RELDS, resets the data set sequence number (JIBDSEQN) to the sequence number for this data set, provided that the sequence number is less than the current JIBDSEQN value; this is done so that FSMGETDS selects the appropriate data set on the next GETDS call.
- Voids any outstanding GET requests for the data set.
- Returns storage that is associated with RPL chains and index tables.
- Issues a "fake close" to unchain and return the I/O buffers and mark the SDB as closed.
- Issues an SJFREQ request to delete SWBs associated with this data set.
- Unallocates the data set, unchaining and freeing (via $RETBLK) the GETREC-related control blocks (GCB, ACB, DEB, and SDB).
- Decreases the data sets' assigned count in the JIB; if the count is zero and the JOE has been completely processed or cancelled, frees the storage associated with the JIB (that is, the JCT and IOT buffers) and places the JIB on the JIB return queue (FSARETQS).
- Frees the checkpoint record resources that are associated with the JIB.
- For a draining FSA device, returns all JIBs on the active JIB queue and marks the device as drained in the FSACB; that is, the FSADRAIN bit is set in FSAFLAG1 of the FSACB.

FSMRELDS returns to the caller.

FSMGETRC - GETREC Service Routine

FSMGETRC supports the FSI GETREC function related to a functional subsystem. The basic function of FSMGETRC is to give the functional subsystem access to data records of a JES2 SYSOUT data set that was previously assigned to the functional subsystem via the FSI GETDS function. Processing overhead is minimized in accessing the data records because FSMGETRC returns only data record addresses to the caller and does not move data from the I/O buffers. Records are obtained through the GET (locate mode) HASP access method.

On entry, FSMGETRC checks whether the FSA device is to be halted (that is via a $Z command). The FSAHALT bit in the FSAFLAG1 of the FSACB is on when the device is to be halted. If the FSA device is to be halted, then FSMGETRC issues a WAIT macro to wait for the device to be restarted. If the FSA device is not to be halted, FSMGETRC accesses the GCB associated with the data set and validates it. For an invalid GCB, FSMGETRC indicates that no index is returned (the GLRNOI bit is turned on in GLRFLGS1), a return code of 4 is set in register 15, and returned to the caller. If the GCB is valid, FSMGETRC determines what kind of request is being made. The request can be a specific record request or a request to get the next record. If neither is indicated, then FSMGETRC assumes that this is a request for the next record.

To process a request for the next data record, FSMGETRC first checks to see if an active RPL chain exists, and if so, ensures that the I/O has completed for the RPL before assigning the associated records to the request. If an active RPL chain does not exist, FSMGETRC invokes the HASP access method (HAM) via a GET macro to fill an RPL chain before assigning the associated records to the request. FSMGETRC assigns records to the FSA by copying record pointers from the RPL chain to the IDX entries and marks the RPL chains in the GCB to indicate that the RPL is assigned to the FSA.

To process a request for a specific data record, FSMGETRC issues a POINT macro to position the data set to the requested record, issues a CHECK macro to wait for the positioning to complete, then issues a GET macro to obtain the specific data record(s). FSMGETRC then assigns records as above.

If an I/O error occurs during the processing of the GETREC request, FSMGETRC fills the index table (IAZIDX) with a pointer to the $HASP185 message and returns to the caller.

FSMFRERC - FREEREC Service Routine

The FSI FREEREC routine, FSMFRERC releases for reuse by GETREC processing the storage areas associated with data records that were previously assigned to the functional subsystem through the GETREC routine. These storage areas include the I/O buffers; FSMFRERC marks the RPL chain as unassigned and issues the ENDREQ macro to release I/O buffers (via HAM).

FSMCHKPT - Checkpoint Service Routine

FSMCHKPT in HASPFSSM supports the FSI CHECKPOINT routine. The basic function of FSMCHKPT is to write out the FSI checkpoint control blocks to the JES2 spool data set.

Upon entry, FSMCHKPT establishes JIB and GCB addressability for the data set that is being processed, and then checks whether a checkpoint buffer has already been acquired and formatted. If the JIB, GCB, or CHK record can't be validated, FSMCHKPT sets a return code of 4 and returns to the caller. If a checkpoint buffer is not available, FSMCHKPT acquires one through a $CALL to FSMCBSET. When a buffer is returned, FSMCHKPT zeroes the JES-dependent work area (CHKJESWK) and indicates in the JIB that a buffer is available (the JIBFCPB bit in the JIBFLG1 byte is set on).

If a checkpoint buffer is available, FSMCHKPT processes the I/O status and error conditions, if any, that are associated with it. If the status of the buffer indicates that:

- A bad checkpoint spool record exits, FSMCHKPT sets a flag in the checkpoint parameter list, indicating a permanent I/O error and then sets a successful return code (register 15 = 0) and returns to the caller.
- I/O is not yet complete for the buffer, FSMCHKPT sets a successful return code (register 15 = 0) and returns to the caller. If the checkpoint parameter indicates that a forced checkpoint write is to be performed, then FSMCHKPT waits for the completion of the I/O and then writes the checkpoint record.
- I/O completed abnormally, FSHCHKPT sets a flag in the JIB to indicate that a bad checkpoint record exists and then calls FSMBTG to process the I/O error.

After establishing addressability to the checkpoint buffer, FSMCHKPT updates the JES-dependent area of the buffer and issues a $BUFIO macro to write the checkpoint record to spool. If the write to spool was successful, FSMCHKPT returns to the caller with a return code of 0 in register 15. If the write to spool failed, FSMCHKPT sets error indication flags in the JIB, queues a TGB for bad track processing and issues the $HASP370 error message. FSMCHKPT then sets a return of 4 in register 15 and returns to the caller.

FSMORDER - Order Service Routine

The FSI order routine, FSMORDER is called from HASPFSSP via cross memory services. HASPFSSP invokes FSMORDER on behalf of JES2's processing of operator commands that are directed to FSA devices operating under the control of the functional subsystem.

Upon entry, FSMORDER saves the cross memory environment of its caller (using a PCLINK) and then obtains a save area to be used by the order routine that is to process the order being received from JES2. This save area is in the FSSXB if the order is for the functional subsystem and in the FSAXB if the order is for the functional subsystem application (FSA).

FSMORDER then prepares to issue the FSIREQ macro for either the FSS order or the FSA order. To determine the type of order, FSMORDER examines the FSID within the order parameter list. If the FSA portion of the FSID is zero, then FSMORDER issues the FSIREQ macro for the functional subsystem, using the FSSXB. Register 1 points to the functional subsystem parameter list and register 13 points to a save area (addressable from the FSSXB). If the order is for an FSA, then FSMORDER issues the FSIREQ macro for the functional subsystem application, using the FSAXB. Register 1 points to the FSA parameter list and register 13 points to a save area the order routine can use when it gets control.

Results of the FSIREQ macro call can be as follows:

- The FSIREQ is successful and an immediate response is given.

  If the order is an FSS-level order, FSMORDER exits to the caller. If the order is an FSA-level order the response area is processed before returning to the caller.
- The FSIREQ is successful and a response is to be given later.

  FSMORDER returns to the caller whether it's an FSA-level order or an FSS-level order.

- The FSIREQ is unsuccessful.

  FSMORDER returns to the caller with register 15 containing a nonzero return code.

FSMPOST - POST Service Routine

The FSI post routine, FSMPOST, in HASPFSSM notifies the functional subsystem or the functional subsystem application of the completion of a request that was made via the FSIREQ REQUEST=FSIPOST macro call.

Upon entry, FSMPOST saves the cross memory environment (using a PCLINK) and establishes addressability to the FSACB, FSSCB, the FSACB extension, and the FSI POST parameter list (in the FSAXB). FSMPOST then moves the reason for the post from the FSAXB to the FSI POST parameter list and issues the FSIREQ REQUEST=FSIPOST macro to initiate the post. After restoring the cross memory environment of the caller, FSMPOST returns to the caller via a program transfer (PT).

FSMSEND - SEND Service Routine

The SEND FSI routine, FSMSEND, receives control as a result of an FSIREQ REQUEST=FSISEND macro call. FSMSEND sends a response to an order that was processed asynchronously. To do this, FSMSEND first checks to see whether the response that is to be sent is a functional subsystem (FSS) response or a functional subsystem application (FSA) response. For a FSS response, FSMSEND stores in register 1 the address of the extended ECB that is to be posted (FSSXECB) and calls (via $CALL) FSMPSTJ2 in HASPFSSM to asynchronously post the processor PCE associated with the functional subsystem order. For a FSA response, FSMSEND first uses the FSMRESP subroutine to process the response area, stores in register 1 the address of the extended ECB that is to be posted (FSAXECB), and calls (via $CALL) FSMPSTJ2 in HASPFSSM to asynchronously post the processor PCE associated with FSA order.

If the request is for the termination of the FSA, bit SNDTYFIT in the SNDTYPE field of the FSI SEND request parameter list (IAZFSIP) is turned on. When FSMSEND finds this bit on, the quiesce bit, defined by FSAQUIES in $FSACB, is turned on in the flag byte FSAFLAG1 of the FSACB of the device. FSMSEND then cross-memory posts the FSA processor to terminate.

Quick Cell Management

FSMGETQC - Get Quick Cell Routine: FSMGETQC is used to obtain one or more quick cells from a predefined pool of cells. If the cell pool has not been initially created or if the cell pool becomes exhausted, FSMGETQC builds or extends the cell pool according to the characteristics defined in the QCT for this quick cell type. FSMGETQC is invoked when the $GETQC macro is issued.

FSMGETQC issues a catastrophic error code of X'F00' when one of the following conditions exist:

- The number of quick cells requested on the $GETQC macro is not in the valid range permitted.
- A GETCELL request is made for the indicated quick cell and the GETCELL fails but not because the quick cell pool is exhausted.

FSMGETQC manages the use of cells by chaining the quick cells together and placing each chain of quick cells on a push-down stack specified in the QCT. FSMGETQC then manipulates this stack and chain structure for each quick cell request.

FSMFREQC - Free Quick Cell Routine: FSMFREQC is entered when the $FREQC macro is issued. FSMFREQC returns one or more quick cells to their proper cell pool, depending on the options indicated by the $FREQC macro issuer. FSMFREQC issues the FREECELL macro to free the cell.

FSMFREQC issues catastrophic error code X'F01' if one of the following conditions should occur:

- The requested number of cells that is to be freed is not within the valid limits.
- The FREECELL macro invocation to free the quick cell fails.

FSMBLDQC - Build Quick Cell Pool Routine: FSMBLDQC builds quick cell pools and their extensions. FSMBLDQC obtains the characteristics associated with the cell pools from the QCT for the quick cell type specified on the $BLDQC macro. FSMBLDQC runs under the local lock.

Initially, FSMBLDQC issues a $GETASCB macro to obtain in register 7 the ASCB address of the address space that issued the $BLDQC macro. Using the QCT, whose address is in register 6, FSMBLDQC calculates the size of the quick cell pool, performs a MODESET to get in key 0 and issues a GETMAIN macro to obtain the storage from subpool 5, storage key 1. Then, switching back to the caller's key (via MODESET), FSMBLDQC issues a BLDCPOOL macro to build the quick cell pool, specifying the number of cells and each cell's size. After the quick cell pool is built, FSMBLDQC returns to the caller.

FSMBLDQC issues the $ERROR macro with a catastrophic error code of X'F02' if one of the following conditions occurs:

- The GETMAIN for the storage for the quick cell pool fails.
- The BLDCPOOL macro to ultimately construct the quick cell pool fails.

FSMQCT - Initial Setup of QCT Routine: FSMQCT sets up the quick cell control table (QCT) by acquiring storage for the QCTs, storing the address of the first QCT in the HFCT, and moving all QCTs into the acquired storage. FSMQCT issues a GETMAIN macro for subpool 230 storage and moves the QCTs defined at FSMQCT01 into this acquired storage; FSMQCT01 in HASPFSSM defines contiguous QCTs via the $QCTGEN macro. FSMQCT then initializes the following QCT entries:

- Save areas
- I/O buffers
- JIBs
- RPLs
- GETR control blocks
- SJFP control blocks

FSMQCT issues the $ERROR macro for a catastrophic error of X'F03' when the GETMAIN for the QCT storage fails.

FSMGTBLK - Get Storage Cell from a Free Pool Routine: FSMGTBLK obtains a specified number of predefined storage cells from one of the free pools of quick cells created by an FSI CONNECT; the FSI service routines use these storage cells for control blocks, buffers, and data areas. The type of cell to be obtained is passed to FSMGTBLK in a half-word parameter list generated by the $GETBLK macro, which is used to invoke FSMGTBLK. The type indicator in the parameter list indicates a particular quick cell control table (QCT) that defines the size, chaining offsets, and the free chain stack header for quick cells of that type.

FSMGTBLK clears all storage cells before passing them to the caller.

FSMRTBLK - Return Storage Cell to a Free Pool Routine: FSMRTBLK returns a number of predefined storage cells to one of the free pools of quick cells established by the FSI CONNECT; these pools of quick cells are used by FSI service routines for control blocks and buffers. FSMRTBLK is invoked when the $RETBLK macro is issued. The type of cell to be returned is passed to FSMRTBLK in a halfword parameter list. The type indicator in the parameter list indicates the particular quick cell control table (QCT); the QCT defines the size, chaining offsets, and the free chain stack header for the cells of that type.

If the cell pointed to by register 1 is the first cell of a chain, then the entire chain is returned to its respective free queue.

Save Area Management

FSMSAVE - Acquire Save Area Routine: FSMSAVE processes save area and register linkage for FSI routines. FSMSAVE uses a stack of unused save areas in its processing and attempts to take one off the top of the stack for each save request. The stack header is called HFSAVSTK and it resides in the HFCT. If the stack of save areas is empty, FSMSAVE acquires more save area space by issuing the $GETQC TYPE=SAVE macro while holding the local lock. After obtaining a save area for the request, FSMSAVE checks to see if tracing is requested by the caller, and if so, issues the $TRACE macro for trace id 11 if the requester is a functional subsystem application (FSA) or 1 if the requester is other than the FSA. For the FSA requester, FSMSAVE completes the formatting of the trace table entry (TTE) prior to issuing a $TRACE RELEASE, releasing the allocated TTE for processing.

FSMRETRN - Return Save Area Routine: FSMRETRN processes save area and register return linkage for FSI routines. FSMRETRN traces the return, if tracing is active, pushes the returned save area onto the save area stack, restores registers of the routine being returned to, and returns to the code that originally called the FSI routine. FSMRETRN uses the $TRACE ID=12 macro for a return to an FSA and $TRACE ID=2 macro for a return to other than an FSA. For $TRACE ID=12, FSMRETRN formats the allocated trace table entry (TTE) prior to issuing a $TRACE RELEASE to release the TTE for processing.

Miscellaneous Support

FSMCATER - Catastrophic Error Message and ABEND Routine: FSMCATER formats and issues the $HASP750 message with the catastrophic error code, reason code, and text that was specified by the issuer of the $ERROR macro instruction. FSMCATER requests an SVC dump using the text of the $HASP750 message as a title. FSMCATER then issues the abend.

FSMCBSET - I/O Buffer Setup for Spool Read/Write Routine: FSMCBSET issues a $GETBLK TYPE=BUF to acquire an I/O buffer and initialize an IOB in the buffer. The IOB is initialized with the ECB address passed on input in register 0, the addresses of the CCW templates, and the address of the DCB associated with the functional subsystem. FSMCBSET returns to the caller with the initialized buffer in register 1.

FSMCBIO - HASP Control Block Read/Write Routine: FSMCBIO checks to see if the read or write option was specified by the caller (register 1 is zero for a read and contains the address of the I/O buffer for a write). For a read, FSMCBIO acquires an I/O buffer (via $GETBUF) and calls FSMCBSET to format the I/O buffer. For both the read and write option, FSMCBIO then uses the buffer to initialize for the read or write I/O. The MTTR track address supplied in register 2 is converted to a full disk address (MBBCCHHR), rotational position sensing (RPS) is set if available for the device and the channel program is activated by issuing the EXCP macro.

FSMCBIO issues message $HASP363 if an invalid track address is detected.

FSMCBCK - Check Read/Write Completion Routine: FSMCBCK issues a WAIT macro to wait for the completion of I/O. The wait is issued for the ECB whose address is supplied in register 0. When the wait is posted, FSMCBCK checks to see if I/O did occur. If I/O did not occur, FSMCBCK issues message $HASP370. If I/O did occur and the requested I/O operation was a write, FSMCBCK returns to the caller. For a read I/O operation, FSMCBCK validates the JCT, IOT, CHK, and SWB and issues $HASP364 if any one of these control blocks is invalid. If the validation is successful, FSMCBCK returns to the caller with a successful return code.

FSMBTG: Bad Track Processing Routine: FSMBTG queues a TGB on the I/O error queue so that bad track processing can occur by the JES2 main task. To do this, FSMBTG issues a $GETMAIN to acquire storage for a TGB cell (subpool 241, key 1), queues the bad spool record address to the I/O error queue, and issues a $$POST (SVTSPOOL) to post the HASP spool manager in the JES2 main task to perform bad track processing. FSMBTG then returns to the caller with the $$POST return code in register 15.

FSMFSLNK - Access FSI Control Block Routine: FSMFSLNK is invoked for each FSI service requested. FSMFSLNK accesses the control blocks that are required to process the request.

After saving the caller's registers, FSMFSLNK validates the function id of the caller; if the function id is invalid, FSMFSLNK sets an error return code in register 4. If the function id is valid, FSMFSLNK establishes addressability to the FSCT and the FSACB and checks to see whether the FSA is connected; if the FSA is not connected, FSMFSLNK sets an error return code in register 4; otherwise, a successful return code is set in register 4.

FSMFSLNK then uses the $TRACE ID=14 macro or the $TRACE ID=15 macro to trace the FSIREQ request. ID=14 is used for low frequency or high importance FSIREQs, such as GETDS, RELDS, and SEND. ID=15 is used for the other FSIREQs, such as ORDER, GETREC, and FREEREC.

Finally, FSMFSLNK establishes the needed values for the specific FSIREQ routine that is to be invoked. The return code stored in register 4 is moved into register 15 and return is made to the caller.

FSMGETLK - Acquire MVS Local and CMS Locks Subroutine: FSMGETLK issues a MODESET macro for key 0 and then issues a SETLOCK OBTAIN for the local or CMS locks. After the lock is successfully obtained, FSMGETLK issues a MODESET macro to reestablish the PSW key of the caller before returning to the caller.

FSMFRELK - Free the MVS Local and CMS Locks Subroutine: FSMFRELK issues a MODESET macro for key 0 and then issues a SETLOCK RELEASE to release the local or CMS locks. After the lock is successfully released, FSMFRELK issues a MODESET macro to reestablish the PSW key of the caller before returning to the caller.

FSMSJTER - SJF Terminate Subroutine: FSMSJTER issues the SJF terminate request (SJFREQ REQUEST=TERMINATE) to terminate SJF, free SJF local storage and perform needed SJF cleanup; FSMSJTER then returns to the caller.

FSMSM150 - Job On Device Message Routine: FSMSM150 issues the $HASP150 message for the first data set in a JIB that is assigned to the FSA.

HASPRTAM: Remote Terminal Services

Module HASPRTAM provides the following functions for JES2 remote and networking communications:

- The line manager to control line use and allocation for binary synchronous communication (BSC) devices and to manage sessions and logical lines for Systems Network Architecture (SNA) devices
- $EXTP service routines for the network job route receiver and transmitter
- The remote console processor to provide for communication between JES2 and remote consoles

HASPMLLM: HASP Line Manager

The line manager is the central control routine for all remote job entry terminal support and network job entry support in JES2. It schedules all communication requests, handles communication error recovery, and interfaces with VTAM for synchronous data link control (SDLC) communication. The line manager is both work and time driven, and it contains several routines that process different kinds of work.

MSEARCH: Mainline General Event Handler Routine

The MSEARCH routine is the central control routine for the line manager and schedules all other routines within the line manager on a demand basis. The routine consists of two segments (MSCANEXT and MBUFSRCH) that schedule routines based on external events, time intervals, and queued work. External events are signaled to the line manager by flags (MLMSCANI/ MLMSCANR) from other processors and exit routines (for example, HASPCOMM, HASPVTAM, HASPBSC, and HASPSNA.) Timing events are internal to and maintained by the line manager. Some routines, notably buffer processing, are executed whenever work exists for them on an associated work queue. Additionally, the line manager may periodically execute certain scan routines by internally setting request flags.

MSAFCHK: Security Authorization Routine

MSAFCHK is called by BSC and SNA signon routines, BSC line start SNA autologon support, and the remote console processor to initiate SAF requests for RJE terminals and remote nodes.

MSAFCHK obtains the storage necessary for the SAF call, issues one or modify form RACROUTE requests, and queues the request to a subtask by issuing the $SEAS macro. The $SEAS uses WAIT=NO to avoid placing the line manager or remote console processor in a wait state.

The signon work element (SWEL) is added to the queue requested by the caller and the $XECBSRV service is called to request posting of the caller's PCE when the subtask completes.

Upon exit, register 15 contains one of the following return codes:

- 0 - $SEAS processing was successful.
- 4 - $SEAS processing was successful and the SQD was posted.
- 8 - A $GETWORK/$GETMAIN failure occurred or $SEAS could not obtain storage for the SQD.

MSWELSCN: SWEL Scan Routine

MSWELSCN scans SWELs for active $SEAS requests (for BSC dedicated line start and SNA autologon) to see if they have completed yet. This routine is called by MSCANEXT when there are elements on a queue of active $SEAS requests. The completion of a $SEAS request results in the dispatch of HASPMLLM, the line manager, which waits for the posted element.

MSWELSCN removes SWELs with posted SQDs from the active queue and places them on a posted queue where they can be processed the next time the checkpoint is owned. MSWELSCN also requests ownership of the checkpoint if any elements are on the posted queues and the checkpoint is not already owned.

MSCANEXT: Main Scan Driver Routine

The MSCANEXT routine is defined between the labels MSCANEXT and MSCANXIT. This routine schedules all of the non-buffer-related routines in the line manager. It is controlled primarily by a table (MSCANTBL), which contains entries for all such routines. The table specifies both the criteria for the execution of each routine and the necessary environment. The main scan driver, using the table, determines if a routine should be executed based on event flags and/or work queue contents, establishes required addressability and other register contents, and invokes the routines. The request flags, event indicators, and work queues referenced by the main scan routine are described below.

The scan request flags, set in both MLMSCANR (set by processors of the line manager PCE external to RTAM) and MLMSCANI (set by the line manager) are:

- MLMSSUNT: Scan inactive logon and SNA line device control tables (DCTs)
- MLMSBUNT: Scan inactive BSC line device control tables (DCTs)
- MLMSSLNE: Scan active systems network architecture (SNA) logical lines
- MLMSSIDL: Scan idle (started but not connected) SNA line DCTs
- MLMSBACT: Scan active binary synchronous control (BSC) lines
- MLMSRAT: Scan the remote attribute table (RAT) for autologon remote work stations
- MLMSSLOG: Scan active SNA logon DCTs

The line manager event indicators, found in PCE flag byte MLMEVNTI, are:

- MLMEPJOB: Job output table (JOT) post occurred
- MLMETIME: One second time interval elapsed
- MLMEDISC: 32-second (disconnect interval) time interval elapsed.
- MLMECKPT: The shared queues are owned
- MLMEALM: Remote is in autologon mode
- MLMEMXSS: MAXSESS exceeded

The line manager work queues in MLLM PCE work area:

- MLMSNAIL: Idle SNA line DCT queue
- MLMSNAAL: Active SNA line DCT queue
- MLMBSCAL: Active BSC line DCT queue
- MLMSNALG: Active SNA logon DCT queue
- MLMICEQ: Line manager interface control element (ICE) exit queue
- MLMLOGQ: Line manager logon DCT exit queue
- $RJECHEQ: Common remote job entry (RJE) buffer queue
- MLMASWLQ: Active SWEL queue

- MLMPSWLQ: Two queues:
  - MLMPSWLB: Posted SWELs for BSC dedicated line start
  - MLMPSWLS: Posted SWELs for SNA autologon

MBUFSRCH: Mainline Buffer Processing Routine

MBUFSRCH is defined between the labels MBUFSRCH and MBUFPROC. MBUFSRCH schedules all buffer processing for the line manager. Buffers that require processing are queued to the line manager buffer queue ($RJECHEQ) by the EXCP channel end appendage and the VTAM request completion exit for BSC and SNA, respectively. After reordering this queue last-in-first-out to first-in-first-out (LIFO to FIFO), this routine examines each buffer to determines its type (BSC or SNA) and establishes the environment for and passes control to the appropriate buffer processing routine (HASPBPRO for BSC or HASPSPRO for SNA).

If the buffer represents an active terminal-initiated signon (buffer points to a SWEL), the buffer processing routine is called only if the SWEL has been posted and if the checkpoint is owned.

MTIMSRCH: Mainline Post-Processing and $WAIT Routine

MTIMSRCH is defined between the labels MTIMSRCH and MLLMWAIT. After the line manager has completed all required scans and processed any queued buffers, it reinitializes its timer, if necessary, and waits ($WAIT) to be dispatched again. When it is dispatched, it returns to the mainline general event routine (MSEARCH).

HASPRBUF: Buffer Queue Routine

To avoid line manager waits, HASPRBUF is called when the shared queues are needed but not owned or when a SAF request has been initiated for a signon. HASPRBUF queues a buffer to the $RJECHEQ and posts (via $POST) the checkpoint processor. HASPRBUF then returns to line manager buffer processing.

HASPROUT: Network Job Route Routines

These $EXTP service routines are used by the network job route receiver and transmitter to reroute a /*XMIT job for local execution. A /*XMIT job can be created in two ways:

1. By a job submitted using a /*XMIT JES2 control statement
2. By JES2 receiving a network job that is not destined for execution on this node (JES2 is serving as an intermediate node in a network)

In either case, the /*XMIT job does not pass through normal input processing and is not syntax checked. This is desirable so that JES2 is a transparent node. If a /*XMIT job is rerouted for local execution, it must be resubmitted through input processing.

The network job receiver (NJR) and transmitter provide the link for re-submission. The network job transmitter retrieves the /*XMIT job from spool and builds the NJE records for transmission. Each record is passed to the corresponding receiver, which submits the job for input processing. The network job receiver and transmitter work similar to normal receivers and transmitters. One major exception is in the $EXTP interface. The NJR processor control elements (PCEs) use the common $EXTP interface in HASPNUC. If the device is a NJR device control table (DCT), control is then passed to HASPROUT.

Upon entry to HASPROUT, register 14 points to a halfword index into ROUTAB. This index indicates what service routine is desired: OPEN, GET, PUT, CLOSE or negative CLOSE.

ROUTOPEN: Network Job Route OPEN Routine

ROUTOPEN is called by HASPROUT to process OPEN requests. If the $EXTP OPEN is for a transmitter, the hold, drain, and EOF bits are turned off to indicate that the transmitter is ready for work.

ROUTGET: Network Job Route GET Routine

ROUTGET is called by HASPROUT to process GET requests for the NJR receiver. If the transmitter has indicated end-of-file, a negative condition code is returned through the ROUTXEX exit. Otherwise, the DCTRBFF bit indicates if data is available for the receiver. If DCTRBFF is off, the receiver waits ($WAIT) until posted ($POST) by the NJR transmitter.

If the data is available, the SRCB is stored in register 0, as required by the HASPRDR RGET routine.

The DCTRBFF bit is reset to show that GET processing is complete.

ROUTPUT: Network Job Route PUT Routine

ROUTPUT is called by HASPROUT to process PUT requests for the network job transmitter. Upon entry to the ROUTPUT routine, the receiver DCTRBFF bit is checked. If it is on, the transmitter waits until GET processing is complete before sending another record. If DCTRBFF is off, the CCW is compared to a truncation CCW. The record is ignored if truncation is desired.

Next the record is transferred from the transmitter PCE work area to the receiver PCE work area. The SRCB is stored in JTWRSCB and DCTRBFF is set. The network job receiver is posted ($POST) for work.

ROUTCLOS and ROUTNCLO: Network Job Route CLOSE Routine

ROUTCLOS and ROUTNCLO are called by HASPROUT to process CLOSE requests. If either routine is entered for a transmitter, DCTRBFF is checked and if necessary, the transmitter waits ($WAIT) until the receiver GET is complete. Then the end-of-file is set on and the network job receiver is posted ($POST) for work.

For receiver CLOSE or negative CLOSE, the DCTHOLD flag is set on.

HASPROUT Exit Routines

ROUTXIT, ROUTXEX, and ROUTXAB are commonly used exits for all HASPROUT service routines. ROUTXIT returns with a positive return code and ROUTXEX with a negative return code, and ROUTXAB with a condition code of 0.

HASPMCON: Remote Console Processor

The remote console processor processes all console messages, commands, and command responses to and from remote terminals, other members of a multi-access spool configuration, and other members of an NJE network. The routine optionally saves messages to remote SNA work stations and BSC terminals - other than sign-on multileaving terminals - for later printing on the terminal printer. Additionally, HASPMCON receives and sends network path manager connection information across the shared spool.

The remote console processor receives control when one of the following conditions occurs:

- A network path manager buffer is placed on the $NPMMSG queue.
- A console message buffer (CMB) is placed on the queue of CMBs for remote terminals, nodes, or shared-spool members ($BUSYRQ).
- Commands or messages are received over the shared-spool.
- A command is received from a remote work station.
- A nodal message record (NMR) is received over an NJE line.

Support for MLWTO message is provided as follows. Each job entry subsystem must receive a complete set of NMRs representing all lines of an MLWTO message before attempting to forward a message. When an MLWTO message transmission begins, no other command or message may interrupt its transmission. The receiving JES2 protects itself from console lockout due to incomplete MLWTO transmission by either truncating the MLWTO or restarting the transmission line when a length delay occurs or inconsistent transmissions are received.

Whenever the remote console processor determines that a member of a multi-access spool configuration has failed, an exit to the network path manager (HASPNMDN) is taken so that this change in the NJE network can be recorded.

HASPMCON initializes itself for processing by acquiring a dedicated buffer pool and allocating it as follows:

- IOT buffer - 4K (RCPIOT)
- Special input (RCPININ) and output (RCPIOOUT) buffer - 4K
- Reserved buffer (RCPRESV) - 4K
- Shared queue output buffer (RCPSOOUT) - 4K
- Shared queue input buffer (RCPSIIN) - 4K

HASPMCON acquires buffer storage for message spooling, initializes remote console DCT and PC15 fields, and creates (if non-existent) the shared communication queue data set JQE.

MCL: Write Network Path Manager Buffers to Spool Routine

MCL examines the $NPMMSG queue and invokes the spooling output function to process any network path manager buffers that have been queued. The buffers are written to the shared-spool and picked up by the remote console processor in the appropriate shared spool member.

MCS: Handle Output CMB Traffic Routine

MCS examines the $BUSYRQ queue and, upon encountering a message queued for a remote terminal connected to the local node, determines the current status of that terminal. If the terminal is an active BSC multileaving terminal connected to the local system and has an operator console, an attempt is made to write directly to the console. A device control table (DCT) representing that specific remote terminal is constructed, the DCT is chained to the other DCTs for the remote work station, and a $EXTP OPEN macro instruction is issued. All queued messages for this remote work station are then written to the terminal ($EXTP PUT), the DCT is closed through a $EXTP CLOSE macro instruction, and the temporary DCT is removed from the chain.

MCS also determines whether or not message spooling is to take place, and, if so, prepares for MCOSPOOL.

MCOSPOOL: Remote Console Message Spooling Routine

If message spooling is allowed ($SPOLMSG was not set to zero at JES2 initialization), MCOSPOOL gets control. MCOSPOOL is invoked to spool messages for the following reasons:

- The remote is inactive.
- The remote console is not operational or does not exist.
- The remote is not a BSC multileaving work station.
- The message count is at the limit (RMTMSG parameter on the TPDEF initialization statement).
- The line DCT is inactive.
- The line DCT is restarting.
- The line is signing off.

In these cases, the messages can not be transmitted to the remote console and must be spooled for transmission or for printing by HASPPRPU.

MCOSPOOL writes messages to the remote message data set associated with the remote terminal. The remote message data set is structured as a special job so as to allow HASPOOL to move it if required; it has a JQE, IOT, and PDDB.

The remote message JQE is created whenever messages must be written for a particular remote terminal; this JQE always exists on the $HARDCPY queue. If the remote message JQE has been previously created (RATRMJQEI0) or has been found by a job queue search, the JQE is reacquired via $GETLOK.

Note: The MSGPRT=NO parameter on the RMTnnnn initialization statement causes JES2 to discard remote console messages if the message would normally have been spooled for later display on a remote printer. (The $T RMTnnnn command can also be used to set the MSGPRT= parameter.)

MCOSPOOL writes messages to the remote message data set beginning with the last record existing on the spool. MCOSPOOL reads the IOT for the remote message data set and obtains the track address of the last message record written to spool from the (only) PDDB. This message record is read, and the message(s) to be spooled are added to it. The message buffer is rewritten to spool. Up to three message buffers may be filled and written before MCOSPOOL waits for the I/O completion(s). If the number of allocated message records exceeds $SPOLMSG, the first message record is reread and the new messages placed into it, thus overlaying the oldest messages in the data set.

The maximum number of allowed messages is 254, or the number of TGAEs. $SPOLMSG is initially set to 254.

When all messages to be spooled have been written to the message data set, the IOT is rewritten, a $FRELOK is issued for the remote message JQE, and a $#POST is issued to post the remote printers.

MCSPATH: Output to Other Node - Determine Path Routine

If the CMB represents a message or command destined for another node, the node information table (NIT) is examined to determine the path to the node. If the NIT indicates that the node is unreachable and the CMB is flagged as containing a command (messages for unreachable nodes are merely thrown away), the CMB command is converted into a path lost message ($HASP243) and queued (via $BUSYRQ) for delivery to the originator of the command.

If the NIT indicates that the node is reachable, the path DCT address is extracted from the appropriate NIT entry. If the DCT is a dummy DCT, the node is reachable via the shared spool; control is passed to the spool output routine (see below). Otherwise, the command or message is to be transmitted over an NJE line to an adjacent node. A DCT representing a remote console is built and chained to the other NJE device DCTs for the NJE line, and a $EXTP OPEN macro instruction is issued. All queued CMBs destined for this node are converted into network message records (NMRs) and transmitted via remote terminal access method (RTAM) ($EXTP PUT) to the next member of the NJE network in the direction of the ultimate receiver.

When all messages and commands have been successfully transmitted, the DCT is closed ($EXTP CLOSE) and removed from the NJE device DCT chain.

MCSSO: Spool Output Routine

If the path to another member of the network is through the shared-spool connection, or if the message or command is destined for another member of the shared spool, MCSSO is invoked. Qualifying CMBs are placed in a spool buffer, which is written to the shared spool. After the writing is completed, the remote console processor is responsible for updating the console queue sub-entry (which contains spool record pointers) corresponding to its member number in the destination member's shared queue element (QSE).

When queuing commands issued at one member for execution on another member ($M commands), the issuer's token is also written to spool for use by MCISI on the target member.

MCISI: Spool Input Routine

MCISI is invoked to read commands and messages from the shared spool. The records were written to the spool by the remote console processors in other members of the shared-spool configuration. Additionally, this remote console processor itself may have spooled incoming RJE and NJE commands and messages for later processing by the spool input function (see MCINSI). Spool records are read from disk. A pointer to the first record to read is obtained from the sending member's console queue sub-entry in the local member's QSE. If the spool record is identified as a network path manager buffer, an exit is made to the network path manager (HASPNBUF) for processing of the network control record. Otherwise, CMBs are removed from the spool record and, if destined for a remote, local, or system node, are chained to the $BUSYRQ queue for subsequent processing by the output CMB function (see "MCS: Handle Output CMB Traffic" preceding).

If destined for the local system, the processing depends on whether the CMB contains a command. If the CMB contains a command, the command authority of the originating console (COMAUTH) is combined with the command authority of the originating node as defined by the local node and contained in the NIT (NITFLAG). If the command comes from another node, a SAF VERIFYX call is initiated by calling MSAFCHK to get a token for the source node. Queuing of the command is performed later by MCINN. Commands from other sources are queued to the CCTCOMMQ queue after obtaining the token from either the spool record (for commands from other members of the same node) or from the DCT (for commands issued by remote terminals at this node). If the command is not a JES2 command, it is passed to MVS for processing (SVC 34) provided the command source has network and system authority; otherwise, it is rejected. Messages destined for the local system are queued for display on local consoles by the $WTO task ($BUSYQUE).

MCINN: Check Pending SAF Request Routine

MCINN checks the queue of SAF requests for commands received from other nodes. SAF requests that have completed are removed from the queue and their associated CMBs are queued to the command processor's command queue.

MCINSI: Receive Transmissions Routine

MCINSI is invoked to receive commands and messages over RJE and NJE lines. The RTAM $EXTP GET interface is used to obtain incoming records, which are moved into newly acquired CMBs for further processing. If no CMBs are available or if the interface is currently receiving or spooling a multiline write-to-operator (MLWTO) message, incoming records are moved to a spool buffer until the end of the MLWTO is received, or until CMBs become available. If the spool buffer is exhausted, it is written to the shared spool with the updated local member's console queue sub-entry in the local member's shared queue element (QSE). The spooled buffer is processed later by the spool input function (see "MCISI: Spool Input" earlier).

If spooling of incoming RJE and NJE commands and messages is unnecessary, the resulting CMBs are queued for processing in the same manner described above for spool input.

HASPBSC: BSC Service Routines

HASPBSC consists of the following routines for BSC remote terminal access method (RTAM) communications:

- $EXTP service routines
- $EXTP, SMF, and DCT initialization/termination subroutines
- BSC line manager scan routines
- Channel-end processing routine

HASPBSCA: Entry Point for $EXTP Service Routines

HASPBSCA is entered from the $EXTP routine in module HASPNUC and invokes requested $EXTP service routines. At entry, register 14 points to a halfword index value. This index is an offset into BSCTAB to indicate which $EXTP service is requested. HASPBSCA initializes common registers and enters the desired service routine. The following routines support BSC $EXTP services.

BSCOPEN: BSC Remote Terminal Open Routine

The $EXTP OPEN service routine converts the line from idling mode to transmit or receive mode. For a multileaving device, the routine also generates the permission request to begin a new function.

When an NJE transmitting function sub-component wants to begin transmitting a unit of work, BSCOPEN is entered by the $EXTP OPEN macro call. BSCOPEN prepares a request-to-initiate function control record and waits ($WAIT) for a response. The RTAM in the receiving JES2 attempts to locate resources to receive the unit of work. If reception is to be permitted, the receiving JES2 responds with the normal RJE permission-to-initiate function control record. Otherwise, the receiving JES2 responds with negative permission. The line manager channel-end processor indicates acceptance or rejection of the request and posts ($POST) the requesting processor; it then gives control to BSCOPEN. BSCOPEN passes condition code indicators to the calling transmitter sub-component to indicate whether the OPEN was accepted.

BSCGET: BSC Remote Terminal Get Routine

The $EXTP GET service routine converts data received from the line into EBCDIC images suitable for processing by the JES2 processors. This conversion includes deblocking, decompression, and conversion from line code to EBCDIC.

When receiving records, BSCGET is entered by the $EXTP GET macro call. BSCGET decompresses and passes logical records one at a time to the caller. When the buffers are emptied, reading of the next block is scheduled. When the processor must wait for incoming buffers, BSCGET issues a $WAIT macro instruction. The line manager channel-end processor posts ($POST) the processor when the data arrives. In an NJE environment, two types of end of file may be received at the end of data. Normal end-of-file is received when the transmitting system wishes the receiver to queue the unit of work on direct access for further processing. It is expected that the receiver processor respond with a $EXTP CLOSE macro call to indicate the work is safely stored. Abnormal end-of-file is received when the transmitting system wishes the receiver to discard all records received within the unit of work. It is expected that the receiver processor responds with either $EXTP CLOSE or $EXTP NCLOSE macro calls. The reception of data, normal end of file, or abnormal end of file is conveyed to the caller via condition codes.

BSCPUT: BSC Remote Terminal Put Routine

The $EXTP PUT service routine converts data from EBCDIC into the form required for transmission to the terminal. This conversion includes compression, blocking, and conversion from EBCDIC to line code.

When transmitting records, BSCPUT is entered via the $EXTP PUT macro call. BSCPUT compresses logical records one at a time, and the results are placed in an output buffer. When a record does not fit into a buffer, the buffer is scheduled for transmission and the processor waits ($WAIT). The line manager channel-end processor posts ($POST) the processor when transmission is complete. The new buffer is obtained, and record compression continues.

If the line manager channel-end processor receives a receiver-cancel control record, the MDCTSTAT DCTERMMR flag is set to indicate an error. BSCPUT, when posted ($POST), passes back the error (via condition codes) to the transmitter processor.

BSCCLOSE/BSCNCLO: BSC Remote Terminal Close Routines

The $EXTP CLOSE/$EXTP NCLOSE service routines convert the line from transmit or receive mode to idling mode.

When transmission of a unit of work is complete, BSCCLOSE is entered via a $EXTP CLOSE macro call to indicate normal end of file, or BSCNCLO is entered via a $EXTP NCLOSE macro call to indicate an abnormal end of file. The close routine sets an abnormal indicator if NCLOSE is used, and the end-of-file record is appended to the end of the current buffer. In the case of NJE, a response is required; therefore, the processor waits ($WAIT). When posted ($POST) by the line manager, positive or negative acceptance is indicated to the caller via condition codes.

When reception of a unit of work is complete, BSCCLOSE is entered via a $EXTP CLOSE macro call to indicate normal acceptance of an end of file, or BSCNCLO is entered via $EXTP NCLOSE to indicate rejection of an attempt to open transmission, a rejection of the current work, or a rejection for end of file. For NJE receivers, the close routine gets a buffer and queues an acknowledge-transmission-complete or receiver-cancel record. Control is returned to the caller.

BSCWRITE: Path Manager Write Routine

RTAM allows the path manager to queue buffers via the BSCWRITE routine to a line for transmission. For a BSC line that is in the prepare for sign-on condition, the channel-end processor later changes its normal SETMODE, ENABLE, and READ sequence to a SETMODE, ENABLE, WRITE, and READ sequence, writing a SOH-ENQ control sequence. If RTAM is on the other end of the line, it aborts its current activity, which should also be the normal prepare sequence, and responds with ACK0. RTAM then transmits the buffer queued by the path manager. If the line is in normal multileaving mode at the time a buffer is queued for transmission by the path manager, RTAM transmits the buffer on its next opportunity to transmit as a normal data transfer.

BSCREAD: Path Manager Read Routine

RTAM recognizes the receipt of network connection control records and queues them to the path manager for action. When the path manager is finished with the buffer, it frees the buffer for more input via the BSCREAD routine. For a BSC buffer, the teleprocessing buffer is marked empty and the MINITIO subroutine is invoked. BSCREAD attaches transmitter/receiver DCTs to the line as required.

BSCSUB: BSC $EXTP Subroutines

The following RTAM BSC subroutines (as distinguished from the device-dependent subroutines associated with each primary service routine) perform common services for the primary routines. Included are common exit subroutines, which return control and a condition code as follows:

MBTAMXIT - normal return, positive condition code
MBTAMXEX - exception return, negative condition code
MBTAMXAB - abnormal return, zero condition code
MBTAMXOV - overflow return, all ones condition code
MBTAMREQ - HASPRBUF return, condition code not applicable
MBTAMRQN - HASPRBUF return without checkpoint request (return and condition codes are not applicable).

MDISCON: Remote Disconnect Subroutine

When MDISCON is entered on behalf of an NJE member, an exit is taken to the path manager (HASPNDCN) so that path control information can be altered to reflect the loss of connection. If the connection is not an NJE connection, MDISCON checkpoints an indication that the remote is disconnected. MDISCON then issues a disconnect message and returns to the caller for an NJE connection. For a non-NJE connection, MDISCON establishes exit point MDSXITA (for exit 17) to allow an installation exit routine to perform further sign-off processing. After the exit routine returns, MDISCON writes a type 48 SMF record (via the $QUESMFB macro) and returns to the caller.

MSIGNON: SIGNON Statement Processing Subroutine

The MSIGNON subroutine receives the address of a /*SIGNON statement in register 1 and a RACROUTE call is issued to determine if the remote terminal has sufficient authority to logon. If the line used to read the sign-on statement was defined as a dedicated line, the sign-on is ignored and the subroutine returns control to its caller without further processing.

For non-dedicated lines (if SAF is controlling the sign-on), MSIGNON will be called at least twice:

1. To start the sign-on process and initiate the SAF call.
2. After the SAF call completes, to finish sign-on processing.

Processing for the two calls is referred to "front-end" and "back-end" processing, respectively. (There may be additional calls if the checkpoint is not owned on the initial call.)

On the first call, the MABORT and MDISCON subroutines are called to disconnect any other remote terminal that may have been attached to this line. Before any further sign-on processing takes place, exit point MSOXITA (for exit 17) is taken; an installation exit routine (if defined) then gets control to replace or modify the standard JES2 checking of the sign-on card. When control is returned from the exit routine a branch table is used based on the exit routine's return code. The following processing occurs based on the return code from the exit routine:

- RC=0 - continue with MSIGNON sign-on processing
- RC=4 - continue with MSIGNON sign-on processing
- RC=8 - terminate sign-on processing and issue message $HASP202 indicating an invalid sign-on. MSIGNON then returns to its caller.
- RC=12 - bypass MSIGNON sign-on processing but perform password checking
- RC=16 - bypass MSIGNON sign-on processing and password checking

For return codes 0 and 4, MSIGNON checks for a correct RAT entry and remote number to correspond to the remote device attempting to sign-on; if no corresponding RAT entry or remote number is found, MSIGNON issues $HASP202 to indicate an invalid sign-on and returns to its caller.

For return codes 0, 4, and 12, MSIGNON calls MSAFCHK to initiate a SAF request, passing the remote name from the RAT (as a user ID) and the old (and possibly new) passwords. For return code 16, the SAF call is made, but a "bypass password checking" indicator is set. If the SAF call subtask is completed by the time MSAFCHK returns, MSIGNON continues with back-end processing. Normally, however, the SAF check will not be complete and MSIGNON will return indicating that a requeue of the buffer is needed.

When the SAF check has completed (and the checkpoint is owned), MSIGNON will be called again to perform back-end processing. The $SEAS service is called to finish analysis of the SAF request. Based on the SAF return code, processing continues with one of the following:

- 0 - Complete the sign-on.
- 4 - Use JES2 RJE password checking.
- 8 - Reject the sign-on request.

If validation (JES2 or SAF) is successful, MSIGNON locates and examines the device control tables (DCTs) representing the remote terminal and line. If the specified terminal is already attached to another line, or if a DCT for the terminal cannot be found, MSIGNON issues the $HASP202 error message and returns control to its caller. Otherwise, MSIGNON attaches the designated terminal to the line.

After sign-on processing is complete, exit point MSOXITB (for exit 17) is taken; an installation exit routine (if defined and enabled) then gets control to perform additional checking. When control is returned from the exit routine MSIGNON writes a SMF record to record the sign-on and issues a type 47 message to the local and remote operators confirming the remote sign-on.

MINITIO: Multileaving Input/Output Interface

MINITIO analyzes the status of a multileaving remote terminal and takes appropriate action to minimize degradation while ensuring maximum line throughput. Based on the status of output processors, the status of the remote terminal, and the status of input and output buffers queued within JES2, MINITIO either transmits ACK0 to the terminal, transmits a text buffer to the terminal, or initiates a 1-second delay.

MEXCP: Remote Terminal Input/Output Interface

MEXCP interfaces with the remote terminal access method through the standard $EXCP input/output interface. In addition to initiating I/O, MEXCP also provides the multileaving block control byte sequence count and the BSC 2770/2780/3780 parity check (ACK0-ACK1) conversion.

MCCWINIT: Channel Command Word Sequence Setup Subroutine

MCCWINIT is passed a sequence type in bits 24-27 of register 1. MCCWINIT constructs a channel command word (CCW) chain, based on that value, and returns. Figure 3-7 depicts the various CCW sequences that can be constructed by MCCWINIT.

MSMFWRIT: Common SMF Write Routine

MSMFWRIT is a common routine used to queue SMF records for recording. It completes all type 47 and 48 SMF records by adding the EBCDIC device/remote identifier. It calls the JES2 common SMF queue routine, which interfaces with the SMF writer SVC, by issuing a $QUESMFB macro instruction.

DCT Initialization/Termination Subroutines

These subroutines are used by the BSC scan routines and the channel-end processing routines to allocate, deallocate, initialize, and release DCTs.

MBDCTGET: Line Allocation and Initialization Subroutine

MBDCTGET allocates a line DCT to the line manager by issuing a $GETUNIT macro instruction. If the line is a dedicated line, MBDCTGET then calls MSAFCHK to initiate a VERIFYX for the remote associated with the line. No password checking is done.

If the line is not dedicated, MBDCTGET calls MBDCTCMP to complete DCT initialization.

MBDCTCMP: Complete Line Initialization Subroutine

This subroutine is called by MBDCTGET (for non-dedicated lines) and by HASPBSLN (for dedicated lines) to complete line initialization. MBDCTCMP uses the $GETBUF macro to allocate a page-fixed teleprocessing buffer, which remains fixed until the line is drained. Then MBDCTCMP initializes the DCT and writes a type 47 SMF record to indicate the line start. The PCE-active count is increased with a $ACTIVE macro instruction.

The BSC CCW initialization routine (MCCWINIT) is used to establish the JES2 BSC prepare sequence CCWs in the line buffer. The prepare sequence is scheduled by calling the BSC common EXCP routine (MERREXCP). This enables (that is, ENABLECCW) the communication adapter and prepares the line to receive incoming data.

BSC Prepare Sequence (Code = C)

BSC Prepare Sequence (Code = C)

CCW	Command	Data Address	Flags	Internal Code	Byte Count
IOBCCW1	DISABLE	0	60	C0/D0	1
IOBCCW2	SET MODE	LCBMCB	60	C1	1
IOBCCW3	ENABLE	0	60	C2	1
IOBCCW4	NOP/WRITE	MBSCSYN/MSOHENQ	60	CA	4/2
IOBCCW5	NOP/WRITE	MBSCENQ/MBSCEOT	60	CA	1
IOBCCW6	READ	TPBUFST	20	C4	BUFSIZE on TPDEF

BSC Multileaving Terminal Sequence (Code = 9)

CCW	Command	Data Address	Flags	Internal Code	Byte Count
IOBCCW1	ENABLE	0	60	92	1
IOBCCW3	NOP	MBSCSYN	60	99	4
IOBCCW3	WRITE	LCBRCB	60	99	2
IOBCCW4	READ	TPBUFST	20	94	BUFSIZE on TPDEF
IOBCCW5	NOP	MBSCSYN	60	98	4
IOBCCW6	WRITE	TPBUFST	60/A0	98	*_*
IOBCCW7	WRITE	METBSEQ	60	98	2
IOBCCW8	READ	TPBUFST	20	84	BUFSIZE on TPDEF

BSC Hardware Terminal Write Sequence (Code = 8)

CCW	Command	Data Address	Flags	Internal Code	Byte Count
IOBCCW1	ENABLE	0	60	82	1
IOBCCW2	NOP	MBSCSYN	60	89	4
IOBCCW3	WRITE	LCBRCB	60	89	2
IOBCCW4	READ	TPBUFST	20	84	BUFSIZE on TPDEF

BSC Hardware Terminal Read Sequence (Code = 8)

CCW	Command	Data Address	Flags	Internal Code	Byte Count
IOBCCW1	ENABLE	0	60	A2	1
IOBCCW2	NOP	MBSCSYN	60	AA	4
IOBCCW3	WRITE	MBSCENQ	60	AA	1
IOBCCW4	READ	LCBRCB	20	A6	2
IOBCCW5	NOP	MBSCSYN	60	A8	4
IOBCCW6	WRITE	TFBUFST	60	A8	*_*
IOBCCW7	WRITE	METBSEQ	60	A8	2
IOBCCW8	READ	LCBRCB	20	A5	2

CTCA Prepare Sequence (Code = C)

CCW	Command	Data Address	Flags	Internal Code	Byte Count
IOBCCW1	SENSE	LCBMCB	60	C2/D2	1
IOBCCW2	WRITE	MNAKSEQ/MSOHENQ	60	CA	2
IOBCCW3	CONTROL	0	60	C7	1
IOBCCW4	READ	TPBUFST	20	C4	BUFSIZE on TPDEF

CTCA Multileaving NJE Sequence (Code = 9)

CCW	Command	Data Address	Flags	Internal Code	Byte Count
IOBCCW1	SENSE	LCBMCB	60	92	1
IOBCCW2	WRITE	LCBRCB	60	99	2
IOBCCW3	CONTROL	0	60	97	1
IOBCCW4	READ	TPBUFST	20	94	BUFSIZE on TPDEF
IOBCCW5	SENSE	LCBMCB	60	92	1
IOBCCW6	WRITE	TPBUFST	60	99	*_*
IOBCCW7	CONTROL	0	60	97	1
IOBCCW8	READ	TPBUFST	60	84	BUFSIZE on TPDEF

Figure 3-7. Remote Terminal CCW Sequences

MBDCTFRE: Line Deallocation Subroutine

After the line is drained, MBDCTFRE frees the line device control table (DCT) by issuing a $FREUNIT macro instruction and decreasing the PCE-active count.

HASPBACT: Active BSC Line DCT Scan Routine

HASPBACT is called from the line manager scan routine (MSCANEXT) in HASPRTAM. HASPBACT is executed at 1-second intervals, or by external request (command processor) whenever there are active BSC line DCTs present on the line manager active BSC line queue (MBSCACT). It is used to perform two basic functions for active BSC lines. First, it services requests for a delayed wait-a-bit sequence (indicated by flag MDCTIMER) for any active multileaving terminals by setting up and executing (using MERREXCP) the appropriate CCW sequence. Second, it checks I/O activity on each line to perform the disconnect interval function.

HASPBUNT: Inactive DCT Scan Routine

HASPBUNT is responsible for acquiring all newly started ($S) BSC lines; it is executed only at command processor request. The common DCT chain (DCTCHAIN) is searched for available BSC line DCTs that are not yet allocated. If such a DCT is found, the line DCT initialization routine (MBDCTGET) is called to acquire and initialize the DCT.

Note: For leased lines, initialization is completed by HASPBSLN.

HASPBSLN: Post-$SEAS Processing for Dedicated Line Start

For dedicated lines, HASPBSLN cannot complete initialization because a SAF call must be made. For these lines, the line manager calls HASPBSLN when the SA call is complete and the checkpoint is owned.

For each completed SAF request, HASPBSLN calls $SEAS to analyze the SAF return code and continues accordingly:

- 0 and 4 - Continues the line start.
- 8 - Mark the line status as drained.

If the line start is to continue, the remote's PCEs are obtained, the token address is placed in the RAT, reader DCTs are processed (for RC=0 only), and MBDCTCMP is called to finish line start processing.

HASPBPRO: BSC Buffer Processing Routine

HASPBPRO maintains the BSC line protocols and controls error recovery; it runs under control of the line manager PCE which, in routine MBUFPROC in module HASPRTAM, has dequeued a buffer from its work queue, MBUFQUE, and called HASPBPRO.

BSC buffers can be placed on the $RJECHEQ (they are transferred to MBUFQUE in MBUFPROC) in two ways: by the IOS channel-end appendage in HASPNUC at I/O completion; and, in a multi-access spool configuration, by the line manager if a sign-on or disconnection is being processed and the shared queues are not owned.

Throughout the description of HASPBPRO, 'multileaving remote' means either a work station running a remote terminal program (RTP) workstation package or a network job entry node (NJE, NJI).

HASPBPRO handles the I/O interface to four general types of devices:

- Hardware terminals (2770, 2780, 3780) using the BSC protocol
- Multileaving terminals (programmable terminals running the remote terminal program (RTP) work station packages) using the multileaving protocol
- Block multiplexer channel-to-channel adapters (CTCs) connecting two NJE nodes and running the multileaving protocol with CTC line control
- NJE nodes using the multileaving protocol

HASPBPRO maintains two communications protocols:

- BSC line protocol (for hardware terminals). Supported BSC responses and framing characters are:

  ENQ EOT ACK0 ACK1 NAK WAK
  STX ETX SOH ETB

  The transparent form of these characters is also supported.

- Multileaving protocol. For discussion of the multileaving protocol and framing, see Appendix A.

HASPBSC supports two sets of channel command word (CCW) sequences:

- Prepare sequence used for line initialization, transmission initialization and termination (OPEN and CLOSE processing), and some forms of error recovery.
- Read/write sequence used for data transmission and reception.

Each protocol has its own set of CCWs for prepare and read/write sequence processing. For a breakdown of the CCW chains, see "Remote Terminal CCW Sequences" earlier in the HASPBSC description.

HASPBPRO is divided into three logical sections: prepare sequence processing, read/write sequence processing, and support routines. The prepare sequence processing section is divided into a general section - successful I/O completion handling - and error recovery. The read/write sequence processing section is divided into a general section - multileaving successful completion - hardware terminal successful completion, and error recovery. CTC processing is handled as a subset of the multileaving processing.

Note that error recovery usually involves restarting the line or restarting or retrying an I/O operation. There are several routines in HASPBPRO that the line manager uses to effect error recovery. HASPBPRO restarts a line by branching to any of the following labels: MDRAINLN, MFORCERL, MRSTLINE. MRSTLINE, the main routine, aborts all active functions on the line (via subroutine MABORT) and disconnects the attached remote terminal (via MSFORCE). The line is put back in the initial prepare sequence, and if the line is a dial line, the phone is hung up.

I/O is retried by a call to MRETRYL (which logs an error first) or MRETRY. MRETRY retries the current I/O operation in the error recovery sequence (the CCW pointed to by IOBRSTRT). I/O is restarted by a call to MRESTART, which causes the I/O to be reissued starting with a read CCW. Both MRETRY and MRESTART call MNEWSTRT, which issues the actual EXCP. After the EXCP has been issued, control is returned to MBUFNEXT.

HASPBPRO uses the following register conventions:

R2	last executed CCW
R3	line DCT
R4	BSC buffer address
R5	address of code table
R6	work and return register

Upon entry to HASPBPRO, common error and operator command checking is done and buffer tracing is activated. An $M01 catastrophic error occurs if the line manager determines that more channel ends have been received than EXCPs issued by JES2 (master I/O count, $EXCPCT), or by the line (active buffer count, DCTBUFCT). The line is drained if the CCW address in the channel status word (CSW) is found to be zeros or if an SIO condition code of busy has been returned.

If the operator has issued a "$TLNE, e=y" (DCTLOGAL on) command, a $HASP094 message is generated to log the channel-end information. The buffer contents are traced if the operator has issued a "$TLNE, tr=i" (DCTRACT on) command and has started event trace for ID4. The line is restarted without processing the contents of the buffer if a disastrous error has occurred (for example, an interface control check), if the sense shows command reject, or if DCTRSTRT is on (set by $ELNE, $PLNE, $SLNE, or by internal line manager error recovery).

MBSCPREP: BSC Prepare Sequence Processing

MBSCPREP processes BSC buffers when the internal CCW code in the last executed CCW indicates a prepare sequence (MPREPSEQ). If the CCWs are in read/write sequence, control is passed to the BSC read/write sequence processing routines at MBCERDWR.

The prepare sequence is used when a line or CTC is started, when a hardware terminal is idling, and during open and close processing for hardware terminals. The only BSC control characters valid are ENQ and EOT (for hardware terminals), SOH-ENQ (for multileaving remotes and NJE terminals), ACK0, and NAK.

If a CTC line is being started, the start line logic has issued a stand-alone CTC sense command. When the sense completes, the results are analyzed by HASPBPRO. If the sense is nonzero, it indicates the CTC at the other end had a CONTROL CCW outstanding. The stand-alone sense issued by this node has caused the CONTROL to complete and the other CTC now has a read outstanding. In this case, MBSCPROC causes a WRITE-CONTROL-READ-CCW chain to be issued. If the sense is zero, the CTC at the other end has not been started by JES2. HASPBPRO then issues a CONTROL-READ CCW string. The CONTROL remains outstanding until the other end is started and issues the stand-alone sense. This process is designed to ensure that the two CTCs will be synchronized with each other with sense completing control, and read complimenting write. For further detail on the CTC CCW sequences, refer to "Remote Terminal CCW Sequences" earlier in the HASPBSC description.

If the last CCW is not a CTC sense command, a check is made to see if the line requires drain (DCTDRAIN) or sign-off (DCTSOFF) processing. If it does, the appropriate action is taken by the MDRAINED routine. If a unit exception occurred, control is passed to MBSCPRUE for unit exception processing. If the I/O completed on a disable or set mode command, the modem was not successfully enabled, and the operation is retried.

The current TOD clock value is placed in MDCTIMOK on the first successful I/O completion for a started line. This value is updated on every successful I/O completion in the BSC read/write sequence and is used in conjunction with the RMT parameter DISCINTV to determine if communication with the other end has been lost. If a unit check has occurred, control is passed to the unit check processing routine, MBSCPRUC.

MBSCPTUC - BSC Prepare Sequence Processing, CSW Status = 0C00: MBSCPTUC processes I/O completions with channel end and device end when a remote terminal or node is either connected to the line or is trying to become connected. The response from the attached device is analyzed, and appropriate action is taken.

If an SOH-ENQ has been received, the line is attached to a multileaving work station or a network node, which is requesting permission to sign-on. All activity on the line is aborted, and an ACK0 is sent. The remote is expected to respond with a sign-on card.

If an SOH-ENQ was sent (indicated by MCPUSEQ set on), a check is made to see if an ACK0 was responded. If so, a branch is taken to MBSCPUSA to set up and send the sign-on record.

If the line is a CTC and a buffer containing a sign-on to be transmitted is queued to the DCT (MDCTOBUF), an SOH-ENQ sequence is set up to be transmitted after a 2-second delay.

An ENQ can be received from a hardware terminal only. It indicates two conditions; either the remote terminal is signing on, or a signed-on remote terminal has work to send and is bidding for the line. In either case, a branch is taken to MSTARTRD where sign-on processing is initiated (via a branch to MSONOPEN), or if the remote is already signed-on, the reader processor control element (PCE) is posted for work.

ACK0 or ACK1 can also be sent by hardware terminals in response to an ENQ sent by JES2 to initiate work for a remote printer or punch. A branch to MSTARTPP is taken to post the appropriate processor for work.

If a NAK is received, several actions are possible. If the terminal is not signed-on, or is a multileaving remote terminal, the I/O is retried via a branch to MRETRY. If the remote terminal is a signed-on hardware terminal, processing continues at MBSCPTST as if a read timeout had occurred. A response other than SOH-ENQ, ENQ, ACK0, ACK1, or NAK is invalid, and if one of these is received, an error is logged and the I/O is retried (MRETRYL).

MBSCPRUE - BSC Prepare Sequence Unit Exception Processing, CSW Status = 0D00: A unit exception usually occurs because an EOT has been sent from the remote terminal, causing a unit exception to occur on a read command. A branch is taken to MRETRYL to log an error and retry the I/O. A unit exception can also occur on a write command when JES2 writes to the line while the control unit is reading from the remote terminal (the remote and processor are out of synchronization), this causes the unit exception to occur on a write command. A branch is taken to MRESTART to read the data transmitted from the remote.

MBSCPRUC - BSC Prepare Sequence Unit Check Processing, CSW Status = 0E00: In unit check processing, the resulting sense is interrogated, and appropriate error recovery initiated. The bulk of the processing is for a time-out. If the unit check occurred while JES2 was dialing the remote terminal, an error is logged, and the line restarted (by MFORCERL). The I/O is retried for a sense that indicates lost data (by MRETRY) without logging an error. If the sense shows intervention is required and the abortive disconnect option has been specified on the LINE initialization parameter, an error is logged, and the line restarted (by MFORCERL). If an abortive disconnect is not specified (DCTPNADS is not on), the intervention required is processed as a time-out. If the sense does not indicate one of these errors, an error is logged and the I/O is retried.

MBSCPRUC uses two routines to process time-outs:

- MBSCPNAD: MBSCPNAD processes the error recovery for time-outs on lines attached to both hardware and multileaving remote terminals. If the line is a dial line and the modem line is not open, no communication has yet taken place between the remote and JES2. The I/O is retried (by MRETRY) if autodial is not selected. If JES2 is dialing the line, the dialing CCW sequence is reinitialized and retried (by MNEWSTRT). The I/O is also retried (by MRETRY) if the last CCW executed was not a read, if no remote terminal is signed-on the line, or if the signed-on remote terminal is a multileaving terminal or terminal of a network node.

- MBSCPTST: MBSCPTST processes a read time-out, intervention required (without the abortive disconnect feature), or NAK received on a hardware terminal. If ENQ or EOT is being sent by JES2 (IOBCCW5's op code is a write), the ENQ/EOT is flip-flopped. (Thus, if an EOT was initially sent, JES2 sends ENQ, and vice versa.) If EOT is now to be sent, it is transmitted immediately by branching to MRETRY.

  The possible work indicator (MDCTJOB) in the line DCT is interrogated to see if any global JOT posts have occurred since the last line manager scan. If no new output has been queued in the JOT, the I/O is retried (by MRETRY).

The remaining processing in MBSCPNAD (starting at label MBSCPNOP) determines if work has been queued to this remote terminal (via $#GET HAVE=NO) and sends an ENQ (via MRETRY) to the terminal if there is work in the JOT. First, a check is made of the interval elapsed since the last successful transmission (using MDCTIMOK and comparing the interval against $WAITIME) to see if JES2 is allowed to initiate output to the remote. If $WAITIME has not expired, the line must be left open for remote transmission; the I/O is retried (by MRETRY) leaving a read hung on the line.

If $WAITIME has expired, the line manager checks for spooled messages queued for this remote terminal concerning print jobs and punch jobs. If no work is queued, MDCTJOB1 is reset in the line DCT; the write ENQ (if any) is changed to a no-op, and a read is hung on the line (by MRETRY). If work is queued, an ENQ is sent.

MBCERDWR: BSC Read/Write Sequence Processing

MBCERDWR transmits and receives data, notifies the input and output PCEs of successful data transmission and reception, and effects error recovery. If a unit exception or unit check for either multileaving or hardware terminals occurs, control is passed to MBSCRWUE (unit exception) or MBSCRWUC (unit check). If the channel end is for a hardware terminal, control is passed to MBCE27X0.

MBCENSF - BSC Multileaving Read/Write Sequence Processing, CSW Status = 0C00: In the multileaving read/write sequence routines, the received information is interrogated and appropriate action taken. Valid multileaving control characters are: SOH-STX (non-transparent text), DLE-STX (transparent text), ACK0 (transmission acknowledged or synchronized idle state), and NAK. ACK1 and WAK are not supported.

If the I/O completed on a CCW other than a read, an error is logged and the line restarted (by MFORCERL). (Multileaving always issues CCWs in write/read pairs.) Control is passed to MBCETEXT for text processing if SOH-STX or DLE-STX is received. If an ACK0 is received, control is passed to MBCEACK0. (The receipt of text in response to a block of text constitutes an implied ACK0.)

MBCENSF uses the following routines for BSC multileaving read/write sequence processing:

- MBCENAK0: MBCENAK0 processes responses that are not text or ACK0. If the response is a request to sign on (SOH-ENQ), control is passed to MBCPUST to abort all activity on the line and set up to read the sign-on record. If neither a NAK nor any of the above valid responses have been received, an error is logged, and a NAK written (by MCWRNAKL).

  If a NAK is received, JES2 must determine what information was lost. If the line is a CTC, there is no data recovery. An error is logged, and the line restarted (by MFORCERL). If JES2 last wrote a WAIT-A-BIT indicating that JES2's input resources were busy, or if the last write was an ACK0, a branch is taken to MBCENAK to initiate a line pause and to post output processors if work is available for them. If the last write type was text, JES2 retransmits the text. If JES2 last wrote a NAK to a network node, a branch is also taken to MBCENAK. If a NAK was written to a remote work station, JES2 resends the NAK (via MRETRY).

- MBCETEXT: MBCETEXT maintains and validates the block sequence count fields (BCB), analyzes the control records (RCB), and posts the appropriate processor for work. For non-CTC lines, the last byte of the buffer is required to be an ETB; if it is not, an error is logged, and a NAK is written (by MCWRNAKL).

  In block sequence count verification there are four possible exception actions:

  1. The remote can request that the count be reset.
  2. While receiving a job, if the receive block sequence count shows a duplicate block received, the block is thrown away, and buffer processing continues.
  3. If the receive block sequence count is in error, a message is issued ($HASP094), and the job is deleted (MRSTRD).
  4. During job transmission, if a block sequence error is detected by the remote terminal, JES2 issues a message, and the job is interrupted by simulating a $I command to the remote device (via MRSTPP).

- MBCENSCK, MBCEDATA, MBCEDISC: These routines dispatch the work in the buffer to the appropriate processors using MBCNTRLR and MBCNEXTR in the BSC common subroutines. The data records and console commands in one input buffer can be destined for different DCTs. Once the input DCT has been found for the first unit of work in the buffer, the buffer is allocated to that DCT (DCTBUFAD), and the device's processor control element (PCE) is posted for work. The line manager posts the output processors for completed work and searches for new work added to the JOT. The line manager has nothing more to do with the additional data records (after the first) in the buffer. When the reader PCE is activated, $EXTP GET logic processes the text in the buffer that is destined for that DCT. If an RCB is found that does not match the input DCTs, MBCNTRLR and MBCNEXTR are called under control of the device PCE. The buffer is allocated to the new DCT, and its PCE is posted. This process is repeated until the buffer is empty. The line manager initiates a timed delay on I/O for the line after the call to MBCNEXTR to allow time for the input PCEs to empty the buffer.

If JES2 last sent a WAIT-A-BIT, the contents of the buffer are ignored, and processing continues at MBCEDISC. If a control record indicates a sign-on record, MSIGNON is called directly from MBCNTRLR. If a control record indicates disconnect, the DCTRSTRT bit is turned on in the line DCT, and the line is restarted (by MRSTLINE) at MBCEDISC.

- MBCEACK0: MBCEACK0 posts active output for successful text transmission; it is part of text processing (SOH-STX and DLE-STX) or is branched to directly if an ACK0 is received. (Text received in response to text is an implied ACK0.)

  When JES2 sends text to a remote, the buffer contains data destined to only one remote. Multiple output processors can be active at the same time filling their own buffers. These filled buffers are queued to the line DCT for output at MDCTOBUF.

  At MBCEACK0 a test is made to see if the last JES2 write to the line was text. If so, accounting is done to show that the time of successful text transmission (by MDCTIMOK), to indicate output was sent in the line DCT (by DCTPOST), and to show that I/O is complete in the remote DCT (pointed to from the buffer chained to MDCTOBUF). The chain of output buffers is processed to see if any console message buffers are queued; if not, the console message count in the line DCT (MDCTCMCT) is zeroed. The first buffer on the MDCTOBUF chain has been successfully sent and is freed (via $FREEBUF).

- MBCENTXT: If the last JES2 write was not an ACK0 response, text processing is rejoined at MBCENTXT to post the output processor for work. The output processor is posted under the following conditions:

  1. A WAIT-A-BIT has not been received.
  2. I/O for the device is complete (DCTPOST is on).
  3. The FCS allows work for that device to be transmitted.

  If any of these conditions are not met, processing continues at MBCENAK. If the PCE is posted, all output and I/O post indicators (DCTPBUF and DCTPOST in the line and remote DCTs) are reset.

- MBCENAK: MBCENAK initiates the next I/O operation and dispatches output processors if a global JOT post has occurred. Control is passed here directly if:

  1. A NAK is received and the last write type was ACK0
  2. A NAK is received and the last write was NAK on a line connecting network nodes (to avoid a NAK-NAK loop)
  3. A WAIT-A-BIT is received

  MBCENAK is also part of text processing.

  If a WAIT-A-BIT was not received, subroutine MCCWINIT is called to reinitialize the CCWs for a multileaving read/write sequence. If a request for a line pause has occurred (MDCTIMER on), I/O is not initiated at this time. Otherwise, subroutine MINITIO is called to issue an EXCP. MINITIO sends the first buffer queued to MDCTOBUF or, if there is none, sends an ACK0. (If text was received, a line pause has been requested, and I/O is not initiated at this time.)

  Next MDCTJOB is tested to see if any work has been added to the JOT; if not, multileaving read/write processing for this buffer is complete, and control is returned to MBUFPROC in HASPRTAM. Processing is also complete if the line is connected to a network node, the line is draining (DCTDRAIN), or the line is signing off (DCTSOFF).

If a JOT post has occurred, the device PCEs are not posted if the device is a remote reader, is in use, or is drained. After all eligible output PCEs have been posted, control is returned to MBUFPROC in HASPRTAM.

- MRSTRD, MRSTPP: These subroutines are called from the block sequence verification section (MBCETETB) if a block sequence error is detected. If the remote terminal detects an error, MRSTPP is called to simulate a $I command to the remote terminal. This causes the remote terminal output PCE to back up to the previous checkpoint when it retransmits the job. If JES2 detects a block sequence error, MRSTRD is called, and the job is deleted.

MBCE27X0 - BSC Hardware Terminal Read/Write Sequence Processing, CSW Status = 0C00: Hardware terminal processing, unlike multileaving, has separate CCW sequences for data reception and data transmission. Some common error checking is performed before control is passed to the specific routines to handle transmission and reception. Valid BSC control characters are: ENQ, EOT, STX, ETX, ACK0, ACK1, WAK, and transparent encodings of these control characters.

If an SOH-ENQ (a multileaving sequence) is received, an error is logged, and the line restarted (MFORCERL). If JES2 is writing to the remote terminal, control is passed to MBCEHDWR at this time.

Further error processing is done when JES2 receives input from a remote terminal. If an ENQ is received, the I/O is retried (by MRETRY). If an SOH is received, the attached header is skipped, and the next block read (by MBCENULL). If an invalid character is received, an error is logged, and a NAK written (by MWRNAKL).

MBCE27X0 uses the following routines for BSC hardware read/write sequence processing:

- MBCEHDRD - BSC Hardware Read Sequence Text Processing: This routine processes both transparent text (STX) sent from the terminal. If the last byte in the transmission is an ENQ, the remote terminal has aborted transmission by sending a STX-data-ENQ sequence, a NAK is then written (MWRNAK). If the last byte received is an ETX, DCTETX is set in the line DCT. The ETX received indicator differentiates between an end of transmission and a request to suspend the output data stream when an EOT is received (see "MBSCRWUE — BSC Read/Write Sequence Unit Exception Processing").

  If data is present in the buffer, a branch is taken to MNORMAL to post the remote reader PCE for work. (MNORMAL is a common routine for both successful reception and transmission processing). If an SOH is received or if no data is present in the buffer, subroutine MEXCPNXT is called to process the next block in the buffer. Control is then returned to MBUFPROC in HASPRTAM.

- MBCEHDWR - BSC Hardware Write Sequence Processing: This routine processes terminal responses to text sent by JES2. The responses can be either 1 or 2 bytes. The only response that causes the next block to be sent is a properly sequenced ACK.

  If the first byte is a DLE, valid responses are ACK0, ACK1, and WAK. If the expected ACK is received, control is passed to MNORMAL to prepare to send the next text block. If the response is WAK, indicating that the terminal has correctly received the text but is requesting that JES2 wait before continuing, JES2 delays by sending an ENQ (via MWRENQ). If JES2 has sent an ENQ and not received a response of ACK or WAK, an error is logged, and another ENQ is sent (by MWRENQL). If the block count is off or the wrong ACK is received, an error is logged, and the I/O retried. If any response but ACK0, ACK1, or WAK is received, it is invalid; an error is logged, and an ENQ written (by MWRENQL).

If the first byte is not a DLE, the response is either a NAK or invalid. If NAK is received and a previous EOT has also been received, an ENQ is written (by MWRENQ). If the NAK is in response to an ENQ indicating that the terminal is denying JES2 control of the line, the ENQ is retransmitted (by MRETRY). Otherwise, an error is logged, and the last text retransmitted (by MRETRYL). If the response is not a NAK, it is invalid; an error is logged, and an ENQ written (by MWRENQL).

MBSCRWUE - BSC Read/Write Sequence Unit Exception Processing, CSW Status = 0D00: A unit exception can occur if JES2 writes to the line while the remote terminal is writing. If this happens, the I/O is reinitiated, starting with a read command to clear the data in the control unit. A unit exception can also occur when an EOT is received by the control unit. If the terminal is a multileaving remote, an EOT is invalid; an error is logged, and a NAK written.

For a hardware terminal, an EOT can be received in a transmit or receive sequence. If JES2 is transmitting (the EOT occurred on a read response), the terminal is indicating that the device is temporarily unavailable (for example, a paper jam has occurred). A check is made to see if the device suspend feature is allowed for the active remote terminal (DCTSUSPD is set on by specifying SUSPEND on the RMT.PR or RMT.PU initialization parameter). If the suspend feature is allowed, a $I command to the device is simulated. In both cases an ENQ is written (By MWRENQL). If the device is interrupted, it is possible that a new job of higher priority will be transmitted before the interrupted job is retransmitted.

If JES2 is receiving from the terminal (the EOT occurred on a read data), the remote terminal is either ending transmission and relinquishing control of the line or is aborting the transmitted block of text. If an ETX was previously received (DCTETX on), the EOT indicates the end of transmission. A branch is taken to label MREADEOT (in MNORMAL) to post the input processor to close. If a previous ETX has not been received, the EOT indicates the terminal is temporarily unable to transmit. JES2 sets up to issue an ACK0 response upon receipt of an ENQ from the terminal and branches to MRDTXT to read for the ENQ.

MBSCRWUC - BSC Read/Write Sequence Unit Check Processing, CSW Status = 0E00: Read/write sequence unit check processing requires different recovery techniques for hardware terminals and multileaving remote terminals. Some common checking is performed, and then processing is split. MCWRNAKL handles the multileaving interface and MBUCHDWR the hardware.

If an "intervention required" is received, a check is made of DCTPNADS to see if the abortive disconnect option was selected for this line (ADISCON selected on the LINE initialization parameter). If this option was selected, an error is logged, and the line restarted (by MFORCERL); if it was not, a branch is taken to MBINTREQ where a NAK is sent. If the failing command is an enable, the enable is retried.

MCWRNAKL processes unit checks for multileaving remote terminals. An error is logged (by MCERRLOG), and if the line type is a CTC, the line restarted (MRSTLINE). Multileaving requires no further analysis of the unit check sense. The

CCWs are reinitialized (via MCCWINIT), and a NAK transmitted. If the buffer is not empty, the data is read in over IOBCCW7 and IOBCCW8 until the buffer is emptied. The NAK is then sent.

MBUCHDWR processes unit checks for hardware terminals. The sense is analyzed, and error recovery is effected on the combination of command type and sense. The error recovery actions are reflected in the following table:

Command Type	Sense	Action	Routine
Read	LD/TO	Read text	MRDTXT
Read	BO/DC/OR	Write NAK	MWRNAKL
Read response	BO/DC/OR/LD/TO	Write ENQ	MWRENQL
Read response to ENQ	BO/DC/OR/LD/TO	Write ENQ	MWRENQL
Write	BO	Write ENQ	MWRENQL
Write response	BO	Restart from read	MRESTART
Write ENQ	BO	Write ENQ	MWRENQL

Several important sequences should be noted. If JES2 is reading text and times-out, nothing is sent; the read text is reissued. If JES2 is sending text and times-out on the read response, an ENQ is sent. For the most part, error recovery consists of sending an ENQ.

HASPBSC Common Service Routines

The common service routines are called by the read/write processing routines to effect error recovery and complete normal data transmission and reception. In most cases these routines use the restart/retry subroutines MNEWSTRT and MRETRY. Control is returned to MBUFPROC in HASPRTAM.

MWRENQL is called from the hardware terminal unit check processing routine to log an error and write an ENQ. The ENQ is written by MWRENQ, described later.

MRDTXT is called from the hardware terminal unit check processing routine to log an error and reissue the I/O at a read text CCW. The routine calls MNEWSTRT to issue the EXCP.

MBSCPUST is used by the multileaving read/write routines to read the sign-on card. All active functions on the line are aborted via a call to MABORT. The CCWs are initialized and executed via a branch to MRETRY.

MRESTART is called to initiate I/O with the next read CCW. MNEWSTRT is called to issue the EXCP.

MWRNAKL, MWRNAK, and MWRENQ are called by both the hardware and multileaving routines to issue a NAK or ENQ (hardware terminals only). At MWRNAKL an error is logged. The I/O is initiated by call to MNEWSTRT.

MBINTREQ is called from both hardware and multileaving read/write sequence unit check processing to handle an intervention required condition or a unit check on an enable command. The routine sets up to send a NAK and calls MNEWSTRT.

MSTARTRD is called to start a hardware terminal reader when an ENQ has been received. If the remote terminal is not signed on, a branch is taken to MSONOPEN to initiate sign-on processing. Otherwise, subroutine MSTUNIT is called to post the remote reader processor control element (PCE) for work. If the device is unavailable, an EOT is returned to the remote terminal.

MSTARTPP is called to start a hardware remote printer or punch from prepare sequence processing if an ACK0 or ACK1 is received. If MDCTJOB is not on in the line DCT, no additional work has been added to the JOT, and an EOT is transmitted to the terminal. An EOT is also sent if the remote terminal is not signed on. MSTARTPP calls MSTARTPU to check for jobs queued for punching or printing. MSTUNIT is called to post the device PCE if separator pages are specified and messages are spooled for the remote terminal and if a punch or print job is waiting output ($#GET HAVE=NO). If no work is queued for this remote terminal, MDCTJOB is reset, and an EOT is written (by MRETRY).

MSTARTPU issues a $#GET to locate available work for printer or punch DCTs. If the shared queues are owned, MSTUNIT is called to start the device. Otherwise, MSTUNIT is called by the hardware device start routine described above to post the associated PCE for work. If the device DCT indicates that it is either in use or drained (DCTINUSE or DCTDRAIN), control is returned to the caller so that an EOT can be sent. If the device is eligible, DCTHOLD is reset and the associated PCE posted for work. A line pause is requested for this line, and control is returned to MBUFPROC in HASPRTAM.

MNORMAL (MREADEOT) is called by both hardware read and write processing to process successful text transmission and reception. MDCTIMOK is set to the current TOD clock value. A secondary entry point (MREADEOT) is entered from unit exception processing upon receipt of a valid EOT. The buffer is posted complete, and the line manager prepares to post the active PCE. If no device is active (DCTINUSE on), the remote terminal is attempting to sign on, and a branch to MSONGET is taken to read the sign-on card. If an active device is found, its processor control element (PCE) is posted, and control is returned to MBUFPROC in HASPRTAM.

MRETRYL, MRETRY, MNEWSTRT are called to retry or restart an I/O operation. On entry to MRETRYL, an error is logged. If MRETRYL or MRETRY are called, the I/O is issued starting at the CCW pointed to by IOBRSTR in MNEWSTRT. If MNEWSTRT is called directly, register 1 is loaded with the address of the CCW to be executed (usually a read CCW). After the I/O is issued, control is returned to MBUFPROC in HASPRTAM.

MCOMMREJ, MDRAINLN, MFORCERL, MRSTLINE are the entries to the line restart and disconnection routine. MCOMMREJ handles a command reject by restarting the line or, if the error occurred on a disable command, by draining the line. MDRAINLN is entered if the line is to be drained. MFORCERL is called to log an I/O error on the console. The line is restarted at MRSTLINE by aborting all active functions (MABORT) and disconnecting the remote terminal (MSFORCE). The line returns to the prepare sequence if it is not drained. If the line is a switched line, the phone is hung up. Control is returned to MBUFPROC in HASPRTAM.

MLLMRCV0: Multileaving Line Manager Processor Recovery Routine

MLLMRCV0 performs the error recovery processing for the line manager processor. MLLMRCV0 is invoked via the $ESTAE recovery mechanism in the event of a system abend or JES2 catastrophic error occurring during line manager execution. MLLMRCV0 attempts to recover only from $M01 catastrophic errors, which are too many channel ends on an RJE line. The DCTDRAIN and DCTRSTRT flags are set on in the line DCT simulating the $E LNEn command for the line in error. The processor is resumed at label MRSTLINE in the BSC channel end processing routine MBSCPROC. The $HASP227 message is issued indicating the recovery action.

HASPSNA: SNA Service Routines

HASPSNA handles all SNA remote terminal access method (RTAM) communications using the following routines:

- $EXTP service routines
- $EXTP, SMF and session control subroutines
- SNA line manager scan routines
- Buffer processing routine
- VTAM application program interface (API)

HASPSNAA: Entry Point for $EXTP Service Routines

The $EXTP routine in HASPNUC calls HASPSNA for the SNA portion of the remote terminal access method (RTAM). Register 14 points to a halfword index that is an offset into SNATAB. This offset indicates which $EXTP service routine is to be used. Common registers are initialized and the desired service routine is entered.

The following routines support SNA $EXTP terminal operations.

SNAOPEN: SNA Remote Terminal/SNA NJE OPEN Routine

SNAOPEN is used to establish a session either between a JES2 NJE processor and a remote terminal or between two JES2 NJE processors. The direction of flow on a remote terminal session may be either outbound from a print/punch processor to a remote printer or punch, or inbound from a remote reader to a reader processor.

The data flow indicator, DCTPOUTB in field MDCTSEL, is used to distinguish inbound from outbound $EXTP OPEN calls. At JES2 initialization, HASPINIT sets DCTPOUTB to 0 for DCTs representing readers, and to 1 for DCTs representing printers and punches.

Opening for Inbound Data Stream: Opening for inbound flow occurs when a remote terminal receives (at a time when no outbound data flow is in progress) a BDS request: a request to begin a data set. The line manager initially handles that request, which is indicated in a function management header (FM header). When SNAOPEN is entered, the line manager has already established the necessary connections between the ICE, representing the session, and the DCT, representing the remote device. SNAOPEN first determines whether the buffer containing the BDS request is still present. If it is not, the attempt to allocate the reader was timed out by the line manager and the session is no longer available; the open call is terminated with condition code 0, indicating that the open request was unsuccessful. If the buffer is present, SNAOPEN need only determine whether the FM header through which the remote work station transmitted the BDS request was accompanied by data. Because the line manager has already removed the FM header from the request and adjusted the request length accordingly, the length is 0 if no data was present; SNAOPEN exits to MVRELBUF, which releases the buffer that contained the BDS request, and returns to the caller of SNAOPEN. If data was present, it is retrieved and processed through subsequent $EXTP GET requests. In that case, SNAOPEN returns to its caller directly, without freeing the buffer.

Opening for Outbound Data Stream: When a remote work station is ready to receive data, and a print/punch processor is waiting to send data to that remote work station, the processor is posted by the line manager and issues a $EXTP OPEN call. Before posting the processor, the line manager has committed the device to the session by storing a pointer to the ICE in the DCT; however, the session cannot be committed to the device until a begin data set request has been sent to the remote work station and accepted, as indicated by a positive response. (Through remote operator action, a need to send a data set inbound might arise while JES2 is preparing to send a data set outbound. By convention, inbound data sets have precedence over outbound data sets up to the moment when a positive response to an outbound BDS request is received.)

To open data flow outbound, the routine first determines whether the session is already allocated. If so, one of two things has occurred:

1. If the current DCT is not the allocated DCT (which must therefore be a reader), an inbound request indicating begin bracket and begin data set has been received on this session in the time between the line manager's post of the print/punch processor and the processor's issuance of a $EXTP OPEN call. The open routine returns to its caller with condition code 0, indicating that the open request was not honored.

2. If the current DCT is the allocated DCT, then the outbound stream being opened has interrupted another outbound stream. The suspending processor, not the line manager, has allocated and posted the current processor. the open routine continues as if the ICEALLOC indicator were off.

If the ICEALLOC indicator is off, indicating that the session is still available, SNAOPEN gets an output buffer, links to the MVFMHBDS routine to cause a request including a BDS FM header or begin destination select FM header to be built and sent, and on regaining control (which occurs only when a positive response to the BDS request is received), returns to the calling processor.

Opening for SNA NJE Data Stream: SNAOPEN processes a $EXTP OPEN call from a processor connected to an SNA NJE line. An immediate return is made if the caller is the remote console processor. Otherwise (if called by an NJE transmitter or receiver), a request parameter list (RPL) is acquired and initialized with pointers to the appropriate control blocks. If called by a receiver, the stream control send routine (MSCSEND) is invoked to send a permission-to-allocate-granted stream control record. If called by a transmitter, MSCSEND is invoked to send a request-to-allocate-job/SYSOUT-stream control record. The transmitter/receiver is placed in the $WAIT state until posted ($POST) with a response to the request or incoming data. When posted, the DCTERMNR flag in the receiver DCT is checked. If it is off, a normal return is made (MVTAMXIT); if it is on, an error return is made (MVTAMXAB).

SNAGET: SNA GET Routine

SNAGET is entered as a result of a $EXTP GET macro instruction request issued by a reader, job receiver, SYSOUT receiver, or remote console processor. SNAGET transfers a data record from a buffer (together with an RPL) received by the line manager to a caller-supplied buffer. The data is decompressed, decompacted (if necessary), and translated from EBCDIC to ASCII, if required. SNA standard character string (SCS) control sequences (special bit strings indicating transparency, secure string reader data, interchange record separator, forms feed, or new line) are deleted from the data stream. Records that span request units (RJE only) are reassembled in the caller's buffer.

Special SNA NJE GET Logic: For SNA NJE, each data record in the buffer is preceded by a 3-byte record identifier (RID) appended to the record prior to compression/compaction by the SNA NJE PUT routine at the transmitting node. The RID identifies the type of data found in the record, the appropriate job, SYSOUT, or console stream, and contains the length of the decompressed/decompacted data record.

Although the RID has undergone compression/compaction at the transmitting node, it must also go through decompression/decompaction at the receiving node. The sneak-a-peek routine, MNSKPEEK (discussed later with other SNA RTAM routines), is invoked to decompress/decompact an RID into a 3-byte work area (RPLRID) in the RPL associated with that buffer. MNSKPEEK is invoked from the RID analysis routine, MNSRANAL, the first time an incoming RPL buffer is examined. Subsequently, it is invoked by the SNAGET routine prior to returning a data record to a caller, in order to decompress/decompact the RID for the next record in the buffer. Thus, on entry to the SNAGET routine for SNA NJE, the RPLRID contains the decompressed/decompacted 3-byte RID associated with the next unprocessed record in the RPL buffer.

The RPLRID field is examined to determine if the associated record is to be processed by the caller (that is, if the stream identifier in the RID matches that contained in the caller's MDCTRCB field). The caller continues to execute the $EXTP GET macro instructions until either the end-of-file is detected or the RID indicates that the next record is not for that caller.

If the record can be processed by the caller, a check is made for a zero-length data record (end-of-transmission or transmitter-cancel); if the record is of zero-length, the appropriate end-of-file return is made to the caller but only after first calling MNSKPEEK to decompress/decompact the next RID (if the RPL buffer contains more data) and then calling MNSRANAL to analyze the RID and process the next record. If the record is a non-zero-length data record, the length of the record is obtained from the RPLRID field, and common NJE/RJE SNAGET logic is invoked (via a branch to the MVGMVE routine) to decompress/decompact the required number of bytes from the RPL buffer into the caller-supplied area. Before returning to the caller, SNAOPEN invokes MNSKPEEK to decompress/decompact the next RID into the RPLRID field.

If the record cannot be processed by the caller (that is, if the stream identifier in the RID does not match that contained in the caller's MDCTRCB field), MNSRANAL is invoked to examine the RID and to take the appropriate action. Upon return from MNSRANAL, the caller is put into the $WAIT state (unless it is the remote console processor) until the next RID for that caller is processed.

SNA RJE GET Logic: An inbound stream is initiated when the remote terminal sends in an FMH1 that indicates begin destination select (BDS) for the associated reader. The request header (RH) that accompanies this FMH1 should indicate only in chain (OC) function management data (FMD), definite response (DR1) and begin bracket (BB) ... RH=0B8080. After the response is received from JES2, the first inbound data record will be issued. This record will carry an RH that indicates first-in-chain (FIC) ... RH=029000. Get processing for a session is finished when an end destination select (EDS) and an end-of-chain are indicated together, and when the last byte of data received in an input buffer has been placed in a reader buffer and returned to the reader processor. The GET routine is reentrant to permit simultaneous handling of data for several sessions.

Upon entry, the GET routine determines whether register MBUF contains an RPL address. If that register contains 0, the routine enters MVREQBUF to cause a buffer to be removed from the inbound queue for this session and its FM header (if any) to be decoded. If MBUF contains 0 on return from MVREQBUF, the job with which this data is associated is being purged. The GET routine links to MVFREPRG to release any further buffers queued to the DCT for processing and then returns to the caller.

SNA NJE GET Logic: SNA NJE uses LU0 protocol. LU0 protocol is not architecturally-defined per SNA protocol. The two application programs that are the session partner end-users communicate using a form of BSC multileaving protocol. SNA NJE does not use bracket protocol as does SNA/RJE LU1. After initial session establishment, there are no function management headers (FMH) used, no brackets, no chains, and no definite response requests (DR1). All transmissions carry a request header in the form: RH=039000. To begin an inbound transmission to either the job or SYSOUT receiver, the sender sends RTIF; the receiver sends either PTIF or receiver cancel. If a PTIF is sent, the next transmission is always the network job header (NJH). Various NMR, NCC, and normal data records may be interspersed in the following transmissions. The actual data set or job transmission ends when the job trailer (NJT) and the EOT record is processed (RIDSRCB=00).

SNAPUT: SNA PUT Routine

SNA PUT is invoked by the print/punch processors, remote console processors, and by JOB and SYSOUT transmitter processing through the JES2 macro $EXTP PUT. The SNAPUT routine converts channel command words (CCWs) into SNA request units (RUs) and calls RPL services to send those RUs to the designated receiver.

SNA PUT is divided into two basic layers. The upper layer decodes the CCW, obtains and initializes buffers, calls the lower layer, passes finished RUs to RPL services, and handles exceptional conditions. The lower layer, also known as the RU composer, performs the actual transformation of channel commands into request units. This transformation includes the suppression of trailing blanks, conversion to SCS representation, compression and compaction (if permitted under the bind session parameters governing the session's operation).

PUT Routine Upper Layer

SNAPUT: If this is an NJE session, SNAPUT branches to the MVPINSTR label and continues processing. If this is not an NJE session, SNAPUT tests the DCTPSUSP indicator in the DCT to determine whether the outgoing data stream has been suspended. (The data stream is suspended, for example, if a change of direction has occurred.) If the data stream is suspended, the routine branches to MVREQUME to wait until processing of the current data stream is resumed.

Next, the ICEWTRSP indicator is tested to determine whether a change direction request (a reversal of the roles of sender and receiver) or some other request has been initiated on the processor's behalf by the line manager and whether the positive response to that request has been received. If a request sent by the line manager is still pending, SNAPUT branches to label MVREQRSP to wait until the response is received.

Next the ICEINSTR indicator is tested. If it is on, the current PUT processing is part of an ongoing data stream; stream resumed processing is bypassed. If the indicator is off, the stream is now being resumed after a previous suspension. SNAPUT enters MVRPLGET to get a buffer and on return, turns on the ICEOCPND indicator and enters MVFMHBLD to cause a resume device selection function management header to be built and transmitted. Because ICEOCPND is on, the header is sent immediately as an only-in chain request.

MVPCPACT: The ICERCPTN and DCTACPTN indicators are compared to determine if a compaction table change is required. If a table change is needed, MVPCPACT branches to MVPUTCPA, ending the current chain by sending a compaction table function management header if one is needed, or just resetting the composer state if a header is not needed.

MVPINSTR: To continue PUT processing, MVPINSTR tests the command type of the CCW passed to the PUT routine by the processor. Seven types of commands are recognized: NOP, truncate, send FM header, write, space immediate, skip immediate, and load forms control buffer (FCB). (Truncate and send FM header are special JES2 CCWs, using command codes X'FF' and X'FE' respectively. Both indicate that the current buffer is to be truncated and sent immediately; X'FF' passes no data, while X'FE' passes an FM header that is sent following the truncated buffer.) MVPINSTR branches to MVPNOPCC for a NOP CCW, to MVPRTUNC for a truncate or send FM header CCW, or continues processing if another type of CCW is received; another type of CCW exits when a buffer is available; if so, MVPINSTR branches to MVPUTBUF.

MVPRPLGT: If no output buffer is available, SNAPUT enters the MVRPLGET routine to obtain a buffer, then sets DCT and internal indicators to reflect data stream characteristics. If the outgoing data stream is being translated to an alternate code, an indicator is set to eliminate checking by the PUT routine for transparent character strings in the data. If alternate code translation is not in effect, checking for transparent strings is allowed in punch data streams. Conversely, for printer data sets, an indicator is set to allow output records to span RU boundaries. Spanning is never allowed for punch data sets. Initial values in the RU composer work area are set for all data sets by calling MVPRPLAN.

MVPUTBUF: MVPUTBUF checks for an NJE session; for an NJE session MVPUTBUF builds an RID (3 bytes, with byte 0 = the record control block from the transmitter DCT, byte 1 = the subrecord control byte from CCW, and byte 2 = the length from CCW minus 1), and moves the RID in front of the data string. The CCW data pointer points to the RID, and the data length is updated to reflect the addition of the RID. A branch is taken to MVPNIMNJ where the data string with the appended RID is passed to the RU composer for compression/compaction into the RPL buffer.

If this is not an NJE session and an output buffer is available, MVPUTBUF prepares to invoke the RU composer. For a space immediate command, no further preliminary processing is required. For a skip immediate command, the preceding command is made immediate. If the result is an immediate skip to the same channel, the current command is redundant and is discarded; the routine branches to MVPNMORE to return to the caller without invoking the RU composer. For a load FCB command, MVPUTBUF branches to MVPUTFCB for special processing. For all other command types, including non-redundant skips, MVPUTBUF enters the MVPNIMED. (If the device is a card punch, MVPUTBUF first changes the command type to force an inter-record separator to be transmitted.)

MVPNIMED: MVPNIMED compares the maximum record length acceptable by the remote device (the "device width"), previously extracted from the device control table (DCT), with the data length of the current command. If necessary, the data length field is changed to cause the maximum rather than the original amount of data to be transmitted. Note that the NJE processors enter this routine at

MVPNIMNJ. The load of the data address and length from the CCW is bypassed as both values have been adjusted to allow for the RID which has been appended to the record.

MVPUT: MVPUT represents the separation between the upper and lower layers of $EXTP PUT processing. All routines that require construction of an RU enter this routine, which invokes the lower-layer RU composer through a branch to label MVKOMPOZ. When MVPUT regains control, indicator SCWMORE is on in the composer's current RU, if the output record could not be completely contained after processing by the composer. (If SCWMORE is off, the current record has been fully processed, and MVPUT branches to MVPNMORE). If composer invocation did not cause the output record to be processed completely, the routine determines whether RU spanning is allowed for this data stream and if not, branches to MVRPLSND, first adjusting the line register so that MVRPLSND returns control to MVPRPLGT. The effect is to cause the current RU, without the partial current output record, to be sent. Reentry at MVPRPLGT causes MVPUT to obtain another output buffer and reprocess the current record, which will fit entirely in the RU.

If RU spanning is allowed, MVPUT enters the MVRPLGET to obtain another output buffer, initializes the buffer, and enters MVRPLSND after adjusting the link register to return control to MVPUT. In this case, MVRPLSND transmits the current RU, including the composed portion of the current output record, and then returns control to MVPUT, which invokes the composer to place the remainder of the current record in the newly obtained buffer.

MVPNOPCC: Entry at this label occurs when an NOP CCW is received by the PUT routine. MVPNOPCC enters MVPNMORE, first entering MVRPLGET to get an output buffer if none is available.

MVPNMORE: MVPNMORE is entered when processing of the current output record is complete, or if the record was an NOP CCW requiring no processing. MVPNMORE checks whether this is an NJE session; if it is, MVPNMORE branches to MVPNMFUL and continues processing. If this is not an NJE session, processing continues as follows.

If the ICEBREAK indicator is on, the remote operator has signalled the computer (for example, by pressing the 3774 ATTN key twice in succession) to indicate that a high priority interruption is requested. MVPNMORE branches to MVPENDCH to force an end-of-chain. If the ICEBREAK indicator is not on, a test is made to determine if the RU is full. If full, a branch to MVPUTRPL is taken to send the RU; otherwise, control is returned to the print/punch processor via a branch to MVTAMXIT.

MVPNMFUL: When a logical page-end has not been reached (or if testing was bypassed), an indicator is tested to determine whether the current RU if full. If it is, MVPNMFUL branches to MVPUTRPI to cause the RU to be transmitted. Otherwise, MVTAMXIT is entered to return to the caller and solicit the next output record.

MVPUTCPA: MVPUTCPA is entered when it is necessary to change the compaction table in effect. The compaction table change is handled before processing the current CCW. The routine first checks whether a previous attempt to change compaction tables failed. If so, the ICECPT field (the address of the table in effect for the session) is reset to point to the table in use before the change.

A compaction table FM header is not needed if DCTACPTN indicates only that compaction should stop. This suppress request is indicated by DCTACPTN changing to 0 on this CCW where it was nonzero on the previous CCW. An FM header is also not needed if DCTACPTN indicates compaction is to resume with the table that was in effect before suppression was requested. This reactivate request is indicated by DCTACPTN being equal to the table number in the ICEACPTN field on this CCW, where on the previous CCW, DCTACPTN was 0.

If an FM header is not needed, MVPNMFUL branches to MVPUTCPR, setting ICERCPTN equal to DCTACPTN to show that the request has been handled, resetting the composer state in the buffer work area as needed. Control then returns to MVPNMFUL for CCW handling. If an FM header is needed, control is passed to MVPTRUNC, forcing an end of the current RU chain. ICERCPTN is left unequal to DCTACPTN to instruct MVPUTFMH (which gets control after response to an end-of-chain) that a table change is still pending and a type 3 FM header must be built and sent.

MVPTRUNC: MVPTRUNC is entered when a CCW with the special truncate command code is received from the processor, or when a function management header must be sent. If this is an NJE session, control is returned to the processor via an immediate branch to MVTAMXIT; otherwise, the current chain is to be ended and transmitted. If an end of chain has already been sent and no new chain has been started, end-of-chain processing is bypassed by a branch to MVPUTFMH. Otherwise, MVPTRUNC enters MVPENDCH to end the current chain, first entering MVRPLGET to obtain an output buffer in which to build the end-of-chain request if an output buffer is not already available.

MVPENDCH: The ICEINCHN indicator is tested. If it is on, indicating that a current RU chain exists, MVPUTEOC is entered, turning on ICEOCPND, the end-of-chain-pending indicator in the ICE. An end-of-chain is forced when MVPUTRPL causes the current RU to be sent. If ICEINCHN is off, the current RU's length is examined. If that RU (which would have been first-in-chain) contains data, MVPUTEOC is entered; the RU is sent as only-in-chain (a single RU chain).

If no new RU chain has yet been started, or if the current RU is the first of a potential new chain but does not yet contain any data, MVFREOUT is entered, freeing the empty buffer. The linkage register ML is set such that MVFREOUT returns control to MVPUTFMH.

MVPUTRPL: MVPUTRPL invokes MVRPLSND to cause the current RPL to be scheduled through VTAM. On return, the ICEINCHN indicator is tested to determine whether the RU just transmitted was one in a chain of RUs (and was not the last in that chain). If ICEINCHN is on, the routine branches to MVTAMXIT to return to its caller and solicit the next output record. Otherwise, MVPUTFMH is entered.

MVPUTFMH: If this is an NJE session, control is returned to the processor; otherwise, MVPUTFMH is entered when either an RU chain has been ended or conditions exist that require sending of a function management header. First, ICERCPTN and DCTACPTN are compared; if they are unequal, control passes to MVPUTCPT to send a compaction table FM header. Next, the current CCW code is checked for the special value X'FE'; if it is present, the issuer of $EXTP PUT has built an FM header, and MVPUTFMH copies and ends it without modification. After sending or for CCW codes other than X'FE', control passes to MVPITRPT.

MVPITRPT: The ICESIGNL, ICEBREAK, and ICEOUTBK indicators are tested to determine whether an interruption has been requested. If no interruption is outstanding, SNAPUT exits through a branch to MVTAMXIT. Otherwise, the current data stream must be suspended. MVPSUSPD is entered if an outbound stream is yielding to an inbound stream; MVPSUSP1 is entered if an inbound stream is yielding to an outbound stream.

MVPSUSPD: MVPSUSPD is entered when the outbound data stream must be suspended to permit an inbound data stream to be processed. The change direction indicator (ICECHDIR) in the ICE is set on, because the receiver of the outbound data stream is now to become the sender and the ICESIGNL and ICEBREAK indicators are cleared. Control then passes to MVPSUSP1.

MVPSUSP1: MVPSUSP1 is entered directly, bypassing MVPSUSPD when the outbound data stream must be suspended to permit another outbound stream to be processed. A buffer is obtained, and the end-of-chain indicator (ICEOCPND) is set on to ensure that the next request in the outbound chain is sent at once and as the end of the on-going chain. The routine then enters MVFMHBLD with register 1 indicating that an FM header for suspend device selection is to be built. MVFMHBLD constructs and transmits a request containing the FM header, together with the end of chain and, if required, change direction indication. As a result, the current outbound data stream is suspended, and control is not returned to MVPSUSPD until that data stream is to be resumed.

When MVPSUSPD regains control, transmission of the interrupted (suspended) outbound data stream is to be resumed. The routine turns on the ICEOCPND indicator to force immediate transmission, adjusts the link register so that return from the FM header build routine is to MVPITRPT, and calls MVFMHBLD to build and transmit a request indicating that device selection is to resume.

MVPUTFCB: MVPUTFCB is entered when SNAPUT receives a load FCB CCW. MVPUTFCB determines whether transmission for this session is in alternate code or EBCDIC. If data is being transmitted in EBCDIC, no special processing is required. MVPUTFCB returns to MVPNIMED to continue PUT processing. (Return is by way of MVPUTFNA, which converts the RTAM simulated load FCB code X'61' to the true load FCB channel command code X'63' by turning on the immediate command bit.)

If alternate code is in use, it is necessary to transmit any alternate code data already accumulated in the output buffer and enter EBCDIC mode temporarily. The load FCB function for a remote printer requires the transmission of a set vertical format (SVF) standard character string control sequence, for which there is no alternate code equivalent. Therefore, the routine determines the length of the current RU. If the length indicates that the RU contains no data, the alternate code indicator in the DCT is turned off, an internal indicator is set to indicate that no more data can be placed in the RU, and MVPUTFCB branches to MVPNIMED by way of MVPUTFNA to cause the request to be sent. If the current RU does contain data, the routine enters MVRPLSND to cause the RU to be sent and gets another buffer by calling MVRPLGET. The register that points to the CCW is reinitialized, the alternate code indicator in the DCT is turned off, and the internal indicator is set to ensure that no data is added to the RU containing the SVF request. (The alternate code indicator is turned on by the MVPRPLGT routine when the next output buffer is obtained.) The routine then returns to MVPNIMED through MVPUTFNA, changing the command as described above.

MVPRPLAN: MVPRPLAN is called for each new buffer. It is also called when a change in the DCTACPTN field causes, within a single buffer, a switch from compaction to compression-only mode or vice versa. This routine stores the maximum-length-RU end address and the initial composer planner routine address into the buffer work area. The planner address stored corresponds to either compaction empty state (MVKQEY) or compression-only empty state (MVKPEY).

MVPUTCPT: MVPUTCPT is entered when MVPUTFMH determines that a compaction table FM header must be sent. MVPUTCPT obtains a buffer and calls MVFMHCPA to compute the address of the desired table. If the table does not exist, control passes to MVPIVCPN, which suppresses compaction, displays a diagnostic error message ($HASP211) containing the invalid compaction table number, discards the unused buffer, and returns to MVPINSTR to handle the original CCW.

If the table does exist, MVPUTCPT stores its address in the ICECPT field and calls the MVFMHCPT routine to construct the type 3 FM header from the table. MVFMHCPT exits to MVRPLSND, which sends the header and waits for a positive response before returning. The new compaction table number is stored in the ICERCPTN and ICEACPTN fields, showing the table change was successful. The routine then returns to MVPINSTR to handle the original CCW.

PUT Routine Lower Layer

The RU composer is a set of routines that convert channel commands to SNA request units. Conversion is a two-step process, consisting of a feeder step and a buffer-fill step. In the feeder step, the channel command is represented as 1, 2, or 3 character strings, depending upon the type of command:

Command Type:	**Character String(s):**
Immediate CCWs	SNA character string (SCS) control string
Write CCWs (no transparency required)	Text string plus SNA character string (SCS) control string (not included for NJE)
Write CCWs (transparency required)	Transparency frame plus text string plus SNA character string (SCS) control string (not included for NJE)

The feeder routine passes these source strings as they are generated to the buffer-fill sub-layer. In the buffer-fill step, the buffer-fill sub-layer transfers the source string data to the RU. If the entire string cannot fit in the current RU, composer processing is interrupted and control returns to the caller with the SCWMORE flag set. The caller obtains a new buffer, copies the composer work area into (preserving the values needed to process the remainder of the source string), sends the old buffer, and calls the composer again. Upon reentry, the composer recognizes this as a continuation call by the setting of the SCWMORE flag, and resumes processing in the buffer-fill sub-layer. RU overflow is then made transparent to the feeder step.

If cross-RU spanning of logical records is not allowed for this device, the composer recognizes this from the setting of the SCWNSPAN flag to 1. In this case, the buffer-fill routine handles RU overflow by returning to the caller with SCWMORE set, but without changing the RU length from it original value at entry to the composer. The caller tests the SCWNSPAN flag, finds it on, sends the old buffer, and then reenters SNAPUT at the beginning, as in the case when no buffer is currently owned by the DCT. In short, RU overflow is prohibited when spanning. This causes any composition done for this CCW to be thrown away, and the entire CCW to be handled using a new buffer.

When the feeder has completed passing the control string to the buffer-fill sub-layer, the SCWMORE flag is turned off, indicating that composing is complete. If the current RU is now full, the SCWFULL flag is turned on to cause the caller to send the buffer. The new RU length is stored in field RPLRLEN in the RPL.

Interface with Upper Layer: The RU composer communicates with its caller through 2 bytes in the composer work area: SCWFLAGS, which contains the composer flags, and SCWLCCC, which contains the channel command code; and through registers 14 and 15 which contain, respectively, the address and length of the text pointed to by the CCW. Registers 14 and 15 are ignored for immediate channel commands. The RU composer also relies on:

- ICERULEN: The ICE field that contains the maximum RU length acceptable on the current session. Logon processing sets ICERULEN to the value specified as BUFSIZE on the RMT nnn initialization statement by copying the RATBUFSZ field.
- MDCTFMT: The DCT format indicator byte that contains the compression and compaction switches. For RJE, the MDCTFMT compression and compaction indicators are set by the SNA header services routine MVFMHBDS during open processing when it forms the FM header properties byte; the indicators are set depending upon the session bind image, which defines the mode of operation for the session. For NJE, the compression indicator (DCTPPRES) is always on and the compaction indicator (DCTPCPCT) is turned on at connection time if output compaction is to be performed on this session.
- RPLAREA: A field that contains the address of the beginning of the RU. RPLAREA is initialized to the address of RPLBUFST by the $BFRBLD macro.
- RPLRLEN: A field that contains the length of the current RU. This field is initialized to 0 by the $BFRBLD macro.
- SCWRUEND: A field that contains the address of the rightmost byte of the maximum-length RU. MVPRPLAN sets this field to the sum of the values in RPLAREA and ICERULEN, less 1.
- SCWPLAN: A field that contains the address of the planner routine corresponding to the present composer state. This field is used only if compression or compaction is active. MVPRPLAN initially sets this field to the empty state.

Determining String Control Byte Usage: The above paragraphs have described the general design of the composer and apply whether or not string control bytes (SCBs) are being used. There are in fact two feeders and buffer-fill sub-layer combinations: one that generates compression and compaction SCBs, and one that does not. The routines that do *not* generate SCBs are identified by label beginning with MVKD (feeder) and MVKU (buffer-fill). The feeder routine for compression is identified by labels beginning with MVKF.

The buffer-fill sub-layer for the SCB case is structured into four kinds of routines: emitter, planner, move, and send routines. There is only one emitter routine (MVKEMITR) and one send routine for compression and compaction (MVKZCNUE). The planner consists of as many regular routines as there are states in the compression strategy, plus an additional suspend routine for each state where it might be possible to add to an SCB with unused capacity created by a previous source string. The move routines perform the actual transfer of data from the source strings to the RU and handle the creation and updating of SCBs.

The four kinds of routines interact in the following way. The emitter determines how many consecutive occurrences of the same character appear in the source string at this point. This information is passed in registers to the routine corresponding to the current planner state, which switches states and calls move routines according to the strategy implemented; in effect, planner routines determine the type of SCB to be formed and where it should be formed. Planner routines call the send routine when the RU becomes full.

Planner routines for compression are identified with labels beginning with MVKP. The planner routines for compression are:

- MVKPEY: For the empty state in which there is no current SCB
- MVKPNC: For the state in which source bytes are being accumulated under a non-compressed, non-duplicate SCB (an SCB representing a string of non-compressed, non-duplicate characters)
- MVKPNCS: For the state in which a non-compressed SCB has been suspended at the end of a source string, although the SCB still has some unused capacity

The move routines for compression are identified with labels beginning with MVKV. These move routines are:

- MVKVPR: For prime duplicate SCBs
- MVKVDU: For non-prime duplicate SCBs
- MVKVNC: For non-compressed SCBs

The planner routines for compaction are identified with MVKQ. The four planner routines are:

- MVKQEY: For the empty state in which there is no current SCB
- MVKQNC: For the non-compressed state in which source bytes are being accumulated under a non-compressed, non-duplicate SCB
- MVKQCA10: For the even compact state in which source bytes are being processed under a compaction SCB
- MVKQCA11: For the add compact state in which source bytes are being processed under a compaction SCB

Odd compact state differs from even compact state in that an odd (unpaired) master character is left over awaiting possible pairing with another master character on the next emission.

For states where a non-compressed SCB or a compaction SCB has been suspended at the end of a source string, though the SCB still has unused capacity, there are four additional planner routines:

- MVKQNCS: For suspending in non-compressed states
- MVKQCAS: For suspending in compact even states

Five mover routines for compaction are identified with labels beginning with MVKW:

- MVKWPRO and MVKWDUO: Alternate entry points for the compression mover routines MVKVPR and MVKVDU
- MVKWNC: For storing non-compressed SCBs.
- MVKWCT: For storing compaction SCBs
- MVKWCA: For converting a string of already-stored duplicate, prime duplicate, and non-compressed SCBs into a compacted string

Planner-End Exits: A planner-end exit and feeder-end exit are associated with each regular planner routine. A branch to the planner-end exit or to the feeder-end exit is located at the entry point of the planner routine, less 4 bytes (planner-end exit) or 8 bytes (feeder-end exit). The function of the exits is to store an SCB with the proper count and cause source data accumulated under a planned SCB to be moved to the RU prematurely, that is, before the strategy actually requires it. This is necessary when the end of a source string is reached. For example, if an uncompressed SCB still has unused capacity at the end of a source string, the planner has not invoked the move routine yet.

When the emitter detects the end of source string, it returns to the feeder level, which in turn either passes the next string (text or control) or returns to the original caller of the composer (PUT upper level). In any case, the storage occupied by the current source string may be altered and is sure to be discontinuous with the next source string to be passed. To ensure that all data has been moved out of the current source string before the next source string is passed, the feeder invokes either the planner end exit or the feeder-end exit.

Common Subroutines Overview: Both SCB-generating and non-SCB-generating feeders make use of two common subroutines: MVKOTPCY and MVKOCCDE.

MVKOTPCY determines whether it is necessary to generate a transparency frame and count sequence for a write command and returns with a condition code indicating its result:

- Minus: The CCW length was 0. The feeder routine is expected to bypass processing the text string entirely because it is null.
- Zero: The CCW length was nonzero, but either the composer flag SCWPUTRN was off, indicating that transparency checking should not be performed, or the transparency check was performed but no values below X'40' were found in the text string.
- Plus: The CCW length was nonzero. The transparency check found values in the text string below X'40'. The address and length of the text string in registers R14 and R15 have been saved in composer work area locations SCWINPT and SCWLINPT. R14 now contains the address of the transparency frame and count sequence in the composer work area and R15 contains the value 2.

MVKOCCDE decodes the channel command and returns with registers RSRCE and RLSRCE set to the address (into the composer work area) and length of the SCS equivalent code string, respectively.

Register Conventions: Certain key quantities are maintained in registers throughout composer processing. Register usage for the non-SCB-generating feeder and buffer-fill routine is confined to those registers in RTAM usually available: 14, 15, 0, 1, 2, and 6. For the SCB-generating feeder and buffer-fill sub-layer, it is necessary to use additional registers, requiring the composer to use the refresh and restore subroutines (MVREBASE, MVRESTOR, and MVREFRSH) before returning to PUT upper layer code (for R3, R5, R8, R10, and R12).

Command Registers 14-2 and 6: R1 contains the number of bytes left in the SNA request unit (RU) - (in the non-SCB case) - or the address of the last byte in the RU (in the SCB case). R2 contains the address of the next unused byte in the RU. R14 and R15 contain the address and length of that part of the source string remaining to be processed. R6 is used by the feeder level to link to subroutines and to the buffer-fill sub-layer.

Registers Used in Compression-Only: When SCBs are being generated, six additional registers are required. Of these, 8 and 9 (RCHAR and RLCHAR) are used by the emitter to execute the compare long instruction. Upon exit from the emitter, register 8 (RCHAR) points to an instance of the emission character, while register 11 (RNUM) contains a copy of the length used in the compare long instruction. When the new length (in RLSRCE), changed by the compare long, is subtracted from the old length, the result is the emission duplication factor less 1. Register 0 (RSCBL) contains the current unused SCB capacity when in the non-compressed, non-duplicate SCB state. A branch on count (BCT) instruction is used to decrease the unused capacity by 1 and simultaneously test if the SCB is full. Register 12 contains the address of the planner routine corresponding to the current planner state.

Register Used in Compaction: Compaction uses the following registers along with the registers described in the previous paragraph: register 5 (RCPT) contains the address of the active compaction table; registers 0 and 10 (RCPZ and RCPA) are used as work registers while in the compaction SCB state; register 9 (RGO) is used in planner routines to perform six-way branches on the emission character attribute.

SNACLOSE: SNA CLOSE Routine

MVCNJE - SNA NJE CLOSE Routine: MVCNJE is called by an NJE processor to terminate an incoming or an outgoing NJE stream. If called by the remote console processor (RCP), a check is made to determine if the console is inbound or outbound. If outbound, a normal return to the RCP is made (via a branch to the MVTAMXIT routine) after first calling the RPL send routine (MVRPLSND) to send the last buffer if the remote device control table (DCT) currently points to an RPL buffer. If this is an inbound console, either the buffer is empty or the next record in the buffer is not for the remote console processor (that is, it is not a all-system message record (NMR)). The RPL buffer is returned to the beginning of the interface control element (ICE) inbound queue and the allocated RCP indicator (ICERCON) is turned off to indicate that this session is no longer dedicated to the RCP. The RID analysis routine (MNSRANAL) is called to process the next record, either in this buffer or in the next buffer, if any. Upon return from MNSRANAL, a normal return to the RCP is made via a branch to MVTAMXIT.

If it is called by an NJE transmitter and the DCTERMNR flag is on in the transmitter DCT, MVCNJE makes an abnormal return to the transmitter via a branch to MVTAMXAB. Otherwise, the stream control send routine (MSCSEND) is invoked to send the appropriate stream control or special (zero-length) data record, as follows:

- RECEIVER CANCEL: If this is a receiver DCT and the DCTERMNR indicator is on
- ACKNOWLEDGE EOT: If this is a receiver DCT and the DCTERMNR indicator is not on
- TRANSMITTER CANCEL: If this is a transmitter DCT and the NCLOSE macro is issued (DCTADS indicator is on)
- END-OF-TRANSMISSION: If this is a transmitter DCT and the CLOSE macro instruction is issued (DCTADS indicator is not on)

Upon return from MSCSEND, an immediate return is made to the caller (via a branch to MVTAMXIT) if the caller is a receiver; if the caller is a transmitter, the caller is placed in the $WAIT state until posted ($POST) with the expected response, at which time control is returned to the caller.

MVCRJE - SNA Remote Terminal CLOSE Routine: The following paragraphs discuss the SNA remote terminal CLOSE routine.

Closing an Inbound Data Set: At entry to MVCRJE, the DCTPOUTB indicator in the MDCTSEL field of the DCT is tested. If the indicator is off, the data set being closed is an inbound data set that has just been terminated with a request carrying the end device select (EDS) indication. In this case, the CLOSE routine need only release the inbound buffer used to transmit the EDS request. The link register is reset (at label MVCINBND) so that control returns to the caller of the CLOSE routine after the buffer is released, and MVRELBUF is called to release the buffer, completing inbound close processing.

Closing an Outbound Data Set: If the DCTPOUTB indicator is on at entry to the CLOSE routine, an outgoing data set is to be closed. This requires that the current RU chain (if any) be ended and that the data stream be terminated through an EDS or equivalent request.

The DCTPSUSP indicator is tested to determine whether the data stream now being closed was previously suspended (temporarily stopped). If DCTPSUSP is on, the routine waits at label MVREQUME until processing for the suspended data stream is to be resumed.

Next, the ICEWTRSP indicator is tested to determine whether a change of direction (reversal of sender and receiver) or other request has been issued by the line manager and has not had a positive response yet. If the response to a request initiated by the line manager is still outstanding, the routine waits at label MVREQRSP until the line manager has received the positive response.

Next, the ICEINSTR indicator is tested. If the data stream is already flowing on this session (the in-stream condition), flow need not be resumed; but if the in-stream indicator is off, a resume device select (RDS) indication must be sent. The routine enters MVRPLGET to get an output buffer, turns on the ICEOCPND indicator to cause the next request to be sent with the end-of-chain indication on, initializes a parameter register with the RDS action code, and enter MVFMHBLD to cause a request to be sent indicating end-of-chain and RDS.

MVCINSTR: With the in-stream condition ensured, the CLOSE routine tests the DCTFLUSH indicator in the MDCTSTAT field of the DCT, branching to label MVCTRUNC if the indicator is off; this means that the data stream is not to be aborted. If the stream is to be aborted, stream termination is to occur through an abort device select (ADS) rather than an EDS indication. If no output buffer is available in which the ADS request can be built, MVCINSTR branches to label MVCENDBF to obtain a buffer and send the ADS indication. If two buffers are available, MVFREOUT is entered to free the first buffer, with the link register loaded so that MVFREOUT passes control to MVCENDDS. If only one buffer is available, MVCINSTR enters MVCENDDS directly.

MVCTRUNC: MVCTRUNC is the continuation of close processing entered when an ongoing data stream exists and no request to abort the stream is outstanding. If an output buffer is available, MVCTRUNC branches to label MVCENDCH to attempt to end the current chain. If no buffer is available, one is obtained through entry to MVRPLGET, and on return the ICEINCHN indicator is tested. If no chain is currently being processed, MVCTRUNC enters MVCENDDS to terminate the current data set by sending an only-in-chain EDS request. If a chain is being processed, it must be ended before the stream can be terminated. MVCENDCH is then entered.

MVCENDCH: MVCENDCH is entered to end the chain, if any, currently being processed for the data set being closed. If the ICEINCHN indicator is off, the current RU, which would have begun a new chain, is examined. If that RU does not yet contain any data, it is ignored; MVCENDDS is entered to terminate the data stream. If ICEINCHN is on, indicating that a chain is being processed, or is off but the current (new or first) RU contains data, the ICEOCPND indicator is set on and MVRPLSND is entered. The current output buffer is scheduled to be sent with the end of chain indicated.

MVCENDBF: This entry point to MVCENDDS is used by MVCINSTR and MVCENDCH when EDS is to be sent but no output buffer is available. It consists of a call to MVRPLGET to obtain a buffer, followed by entry to MVCENDDS.

MVCENDDS: The ICESDCT pointer is tested to determine whether a previous data stream flowing on this session has been suspended. If so, MVCENDDS branches to MVCNOTEB. If no suspended data stream exists, the primary protocol indicators (set to reflect session parameters when the session was created) are tested. If the protocol in effect permits JES2 to send the end bracket (EB) indication to the remote sender, the EB-pending indicator (ICEEBPND) is set on. If the EB indication must come from the remote work station, the change direction pending indicator (ICECHDIR) is set instead. In either case, MVCDSTRM is entered.

MVCNOTEB: MVCNOTEB examines all DCTs on the suspended data stream queue. If a suspended outbound data stream is found, MVCNOTEB enters MVCDSTRM to send an EDS indicator without changing direction. Otherwise, MVCNOTEB branches to label MVCHGDIR to set the change direction indicator to allow an inbound stream to be resumed.

MVCDSTRM: MVCDSTRM terminates transmission of the current data stream by causing an end device select (EDS) or abort device select (ADS) indication to be sent. At entry, the link register is loaded with the address of the normal exit routine MVTAMXIT, so that any path out of this routine ends with a return to the invoker of the CLOSE routine. Next, the parameter register is loaded with the EDS action code and the ICEOCPND indicator is set on to cause the next request to be transmitted to end the chain. MVCDSTRM branches to MVFMHBLD to cause a terminating request to be built and sent. Before MVFMHBLD is entered, however, the $EXTP index value in the halfword proceeding the return point is tested to determine the type of $EXTP call. If the request is not CLOSE, NCLOSE is assumed and this data stream is to be aborted. The EDS action code in the MVFMHBLD parameter register is replaced by the ADS action code.

SNANCLO: SNA Negative CLOSE Routine

For a SNA remote terminal, SNANCLO is entered via a $EXTP NCLOSE macro call. It sets the DCTADS bit and enters SNACLOSE. For SNA NJE, SNANCLO is entered when a $EXTP NCLOSE is issued by an NJE processor. If it is called by a transmitter, an immediate branch is taken to enter SNACLOSE at MVCNJE. Otherwise, for a receiver, the receiver cancel flag (DCTERMNR) is turned on and the DCTINUSE indicator in the DCTSTAT flag byte is tested. If not on, SNACLOSE is entered at label MVCNRCVR to perform receiver close logic. If the DCTINUSE indicator is on, the DCTEOF indicator is tested to see if an end-of-file was detected. If not, the NJE RID analysis routine (MNSRANAL) is invoked to examine the next RID and take the appropriate action for the record headed by that RID. If the DCTEOF indicator is on or if not on, upon return from MNSRANAL, a branch is taken to enter SNACLOSE at MVCNJE.

SNAREAD: Path Manager Read Routine

This routine is used by the network path manager to release SNA buffers (RPLs) via MUFREEIN. SNAREAD sets the compress flag (DCTPPRES) on in the transmitter DCT and attaches the transmitter and receiver DCTs to the line if necessary. SNAREAD then examines the interface control element (ICE) to determine whether compaction is to be performed on output data and sets the compact flag in the transmitter DCT accordingly.

SNAWRITE: Path Manager Write Routine

The network path manager uses SNAWRITE to pass RPLs to VTAM. SNAWRITE calls the RPL send routine (MVRPLSND) to schedule the RPL for transmission.

SNASUB: SNA $EXTP Common Exit Routines

Exit routines exist for each condition under which RTAM processing completes. Each exit restores the caller's registers, turns off the DCTRTAM indicator to show that the processor is no longer within RTAM, sets the condition code, and returns to the caller. Exit labels and conditions code settings are:

- MVTAMXIT: Normal exit; condition code positive (2)
- MVTAMXEX: Exception exit; condition code negative (1)
- MVTAMXAB: Abnormal exit; condition code zero (0)
- MVTAMXOV: Overflow exit; condition code "ones" or "overflow" (3)

MVRPLSND: RPL Send Routine

MVRPLSND is called to schedule a teleprocessing (TP) buffer for VTAM execution. (A TP buffer includes a request parameter list (RPL), a work area also used as a register save area, and a data buffer.) The TP buffer containing the RPL to be scheduled for execution by VTAM is pointed to by R4 and the DCTBUFAD field of a DCT. The DCT in turn is pointed to by R3. The interface control element (ICE) representing the session is pointed to by R5.

MVRPLSND: This is the primary entry point for the RPL send routine. Before the RPL can be scheduled for execution, several tests are required. To prevent JES2 from using too many TP buffers for requests queued for scheduling, the number of buffers queued for outbound transmission (ICEOUTCT) is compared against a limiting value (ICEOUTLM). If the limit has been reached, MVRPLSND branches to the MVRPLWT entry point to wait ($WAIT) for the number of queued buffers to be reduced. Note that for an NJE session the buffer limit test is bypassed if currently running under the line manager or the remote console processor's PCE. Also, if no $WAIT is necessary, the ICEOCPND indicator in the ICE send status flag (ICESNDST) is turned on, register 15 (MVRPLCHX table index) is loaded with the index value of 2 (this causes the RPL to be flagged as only-in-chain and the ICEOCPND indicator to be turned off), and the RJE specific logic is bypassed with an immediate branch to MVRPLCHN.

For an RJE session, if no $WAIT is necessary, the content of the buffer's RPLNEXT field is placed in DCTBUFAD, and MVRPLSND determines whether this RPL represents an RU that is part of a chain of RUs.

If the current RPL represents an RU that is part of an RU chain and is not the first in the chain, a test of outstanding response limits need not be performed. Only one response is returned for an entire chain; the limit check is made when the first RU in the chain is processed. If ICEINCHN indicates that the current RPL's RU is part of a chain, the routine branches to MVRPLHDR to continue scheduling. Otherwise, the response limit check is performed.

A count of outstanding requests requiring definite responses is kept in the ICE (ICERSPCT). The count is increased by the RPL send routine when a request requiring a definite response is scheduled; the count is decreased by the line manager when a definite response is received. Before adding to the outstanding response count, the routine compares the current value of the counter (ICERSPCT) with the maximum outstanding request value (MDCTCHLM), which is always 1. If there are already as many outstanding responses as are permitted for this session, MVRPLSND branches to the MVRPLWTR entry point to wait until the count has been reduced. In this case, the wait routine sets on the waiting for response indicator (ICEWTRSP) in the ICE, which informs the line manager that the associated processor is awaiting a response and that the line manager should post the processor when the next response is received.

In addition, two secondary entry points are provided: MVRPLWTR and MVRPLWT.

MVRPLWTR: This entry point is used when an RPL is to be scheduled only after a response has been received for the last RPL previously scheduled for this session. MVRPLWTR turns on the wait for response (ICEWTRSP) indicator in the session's ICE and enters the MVRPLWT entry point to wait for the response to be received. In addition, MVRPLWTR stores the contents of MBUF in DCTBUFAD. If MVRPLWTR is entered from MVRPLSND, DCTBUFAD points not to the current RPL, but to the RPL pointed to by the current RPLNEXT field. Storing MBUF in DCTBUFAD restores the pointer to the current RPL until the wait is satisfied. Finally, MVRPLWT is entered.

MVRPLWT: This entry point is used when some specific event must occur before the current RPL can be scheduled. Examples of these events are a receipt of the response associated with a previously scheduled RPL, or a reduction of the number of outstanding responses below the limit. MVRPLWT waits for the specific event ($WAIT WORK), then upon regaining control restores the access method registers using the MVREFRSH routine and tests the ICECNCEL indicator to see if a negative response for this session was received by the line manager during the wait. If so, the $EXTP call is aborted through a branch to MVREQCAN; otherwise, processing continues normally by entering the main routine at MVRPLSND.

MVRPLOCH: When it has been determined that another outstanding response is permissible (or when this entry point is used by an SNA service subroutine), MVRPLOCH increases the count of outstanding responses by 1, and continues to process the RPL for scheduling.

MVRPLHDR: MVRPLHED performs further RPL initialization to prepare the request for scheduling. First an index value of 4 is set in register 15. (The index value is used at label MVRPLCHN, as described below.) The ICEINCHN indicator is tested. If no chain is currently being processed, the index value is reset to 0, and label MVRPLFC1 is entered to test for actions that may be required after a chain has ended.

MVRPLFC1: If a function management header is pending, the RPLFMHDR indicator is set on. If end bracket is pending, the RPLEB indicator is set on.

MVRPLFC2: If the ICEINBRK indicator in the ICEFLAGS field is off, indicating that no bracket has begun on this session, the begin-bracket pending indicator (ICEBBPND) and outbound data stream indicator (ICEOUTBD) in the ICEFLAGS field are turned on along with the begin-bracket indicator (RPLBB) in the RPL. This ensures that when the RPL is scheduled, a begin-bracket indication is transmitted. If ICEINBRK is on, the session is already in bracket state, and MVRPLNFC is entered.

MVRPLNFC: The ICEOCPND indicator in the ICESNDST field is tested. If the indicator is off, the current chain is not ending and MVRPLNFC branches to MVRPLNEC. If the indicator is on, a previously executed routine has determined that the chain should be ended with the current RU. The index value in register 15 is increased by 2, and the RPLVTFL2 field of the RPL is marked to indicate that a definite response to the current request is required; that is, after the current RPL has been scheduled and the corresponding request has been transmitted, JES2 must receive a specific indication (the definite response 1 indication) that the request was received successfully. MVRPLNFC next determines whether its caller was the line manager. If some other processor was the caller, that processor waits until the required definite response is received; the link register is adjusted so that return from MVRPLNFC is not to its caller, but to MVREQWT, the wait for response routine (which returns control to MVRPLNFL's caller after the wait has been satisfied). If the line manager is the caller, however, waiting is not permitted and the link register remains unchanged. In either case, MVRPLNFL turns on the RPLCMD indicator in the RPLRH3 field if the ICEFLAGS indicator ICECHDIR indicates that a change-direction operation is pending, then it enters MVRPLNEC.

MVRPLNEC: If the DCTPALTC indicator in the MDCTFMT field is off, data to the current device is being transmitted in EBCDIC and MVRPLNEC branches to MVRPLCHN. If the indicator is on, translation to alternate code is necessary. The length of the current RU is tested and is reduced by the length of a function management header if one is included in this RU. If any data remains (that is, if the RU did not consist solely of an FM header), the data is translated into alternate code form before MVRPLCHN is entered.

MVRPLCHN: The index value calculated in the preceding steps is used to set the RPL indicator that reflects the current RU's position in the chain and to set the ICE indicator that reflects the session's state. The effect of each possible value is:

0 The RPL indicates that this RU is first-in-chain. The ICEINCHN indicator is turned on.

2 The RPL indicates that this RU is only-in-chain. The end-of-chain pending indicator (ICEOCPND) is turned off.

4 The RPL indicates that this RU is middle-of-chain (that is, other than first or last). The ICE indicators are not changed.

6 The RPL indicates that this RU is last-in-chain. The ICEINCHN indicator and the ICEOCPND indicator are both turned off.

MVRPLOCR: When an RU (possibly modified) is to be sent, the count of active outbound requests (ICEOUTCT) is increased by 1.

MVRPLRQN: If the RPL containing the RU is to be queued to the outbound queue, the RPL is marked as a send request, and the send control sequence is stored in the RPLSEQTP field for later examination by the line manager. (This entry point to the RPL queueing routine is also used when successfully received incoming request is to be converted to a response and placed on the outbound queue.)

MVRPLQOB: The RPL scheduler next determines whether the RPL may be scheduled at once or must be queued. The address of the line DCT, the address of the ICE, and the session's communication ID are stored in the RPL, then the ICEOUTBF and ICEOUTHD pointers are examined to determine whether an outbound transmission is already in progress on this session. If one is not, the current RPL can be scheduled at once and the routine branches to MVRPLXIN. If a previous outbound RPL for this session is still being processed, the current RPL is added to the ICE's queue of outbound requests.

MVRPLXIN: As a final check before actually passing the RPL to VTAM, the routine tests the ICEABORT and ICECLOSE indicators and, if the session is terminating (abnormally or normally), branches to MVFREEOQ to release the buffers on the outbound queue. If both indicators are off, the address of the current buffer is stored in the ICEOUTBF field, indicating that an outbound transmission is in progress, and the RPLBRANC indicator in the RPLEXTDS field is turned on; this indicator causes a branch, or fast-path, entry to VTAM's RPL processing routines.

MVRPLXEC: MVRPLXEC initializes the RPL and creates the register environment required for the interface with VTAM.

The address of the RPL exit routine, VEXITRPL, is stored in the RPLECB field. VEXITRPL is the routine to which VTAM returns control when the request represented by the RPL has been completed.

The address of the current processor control element (PCE) is stored in the RPL, and register 1, a VTAM parameter register, is loaded with the address of the current RPL. MBUF, the RPL pointer, is loaded with the address contained in the current RPL's RPLNEXT field, which is the address of the next buffer in the buffer chain, or 0 if there is none.

The sequential access and asynchronous processing indicators are turned on.

VTAM parameter register 0 is loaded with the request type that was stored in the RPLREQ field by the routine that submitted the RPL to the RPL scheduler. (MVRPLXEC is entered by any SNA routine that requires that an RPL be scheduled. The request type field may contain any of the request types used by JES2, such as send, receive, reset-continue-specific, or close destination.)

Register 13 is loaded with the address of a standard 18-word save area (also contained in the RPL, at RPLSAVEA) for use by VTAM. Register 13, the PCE base register, is reloaded with the PCE address from the RPL when HASPSNA is reentered.

Finally, the access control block (ACB) is referenced, the address of the interface routine in VTAM contained in ACBINRTN is obtained, and VTAM is entered through a BALR instruction.

When VTAM returns control to JES2, the RPL scheduler restores the PCE address to register 13. VTAM return code register 15 is tested. If the return code is 0, control is returned to the calling processor. If the return code is nonzero, the requested operation was not started successfully; MVRPLXEC determines whether a disastrous error has occurred or the request can be tried again.

If the return code returned by VTAM is not X'04', or is X'04' but the recovery action return code returned in register 0 is not lower than X'14', an unrecoverable error (possibly a JES2 logic error) has occurred. The routine branches to $R01, at which point the $ERROR macro instruction is issued to cause the JES2 catastrophic error routine to be entered with JES2 error code $R01. If there is no error, a compare and swap instruction is used to add the buffer containing the RPL to the line manager's $RJECHEQ queue in order to cause the RPL to be rescheduled by the line manager. The line manager is posted for work ($POST WORK), and the routine returns to the calling processor.

MVRPLKOQ: Outbound Queue Restart Subroutine

MVRPLKOQ is entered from the line manager's VTAM interface/request end processor, MSNAPROC, to determine whether the outbound queue needs to be restarted. The ICEOUTBF and ICEOUTHD fields are tested. If ICEOUTBF is nonzero, an outbound request is active in VTAM and no restart is required; control is returned to the request end routine. Similarly, control is returned if ICEOUTHD contains 0, indicating that no request is available to be scheduled. However, if no outbound request is being processed by VTAM and ICEOUTHD contains the address of a buffer ready for processing, the outbound queue is restarted (or purged if the session is aborting).

The ICEABORT and ICECLOSE indicators in field ICESTAT are tested. If the session is being terminated, MVRPLKOQ branches to MVFREEOQ to release the buffers on the outbound queue. Otherwise, the first buffer on the queue is removed, the current contents of the MBUF register are stored in this buffer's RPLNEXT field, and MVRPLKPQ branches to MVRPLXIN to cause the RPL contained in that buffer to be scheduled. Because MVRPLXIN replaces the RPL address passed to it in MBUF with the contents of the RPLNEXT field, the net effect is that the contents of the MBUF are not changed by the call of MVRPLKOQ. When MVRPLKOQ is invoked from the line manager's request end routines at MSNAPSND, the initiation of the next send by MVRPLQOK does not destroy the address (in R4) of the just-completed send being handled at MSNAPSND.

MVRPLGET: RPL Buffer Get Subroutine

MVRPLGET is entered when a SNA routine requires a VTAM RPL buffer, that is, a buffer in which a VTAM RPL is to be built and initialized.

A get buffer request ($GETBUF TYPE=VTAM) is executed, with MVRPLNO specified as the address to which control is to be returned if no buffer is immediately available. MVRPLNO, if entered, waits until a buffer has been released by another processor ($WAIT BUF). The ICECNCEL indicator is then tested because an exception response for a previous transmittal may have been received by the calling outbound processor while that processor was waiting for buffers. If ICECNCEL is on, MVRPLGET branches to label MVREQCAN to abort the current $EXTP request. Otherwise, MVRPLGET is reentered to repeat the get-buffer request, attempting to acquire the just-released buffer; this cycle is repeated, if necessary, until a buffer is actually obtained.

When a buffer has been obtained, the address of the access control block (ACB) is extracted from the logon DCT and stored in the RPL. Next, R4 is tested to determine whether a current buffer existed when MVRPLGET was entered. If so, the address of the new buffer is placed in the RPLNEXT field of the current buffer. If not, the address of the new buffer is placed in MBUF and in the DCTBUFAD field of the DCT. After the address of the new buffer has been stored, control is returned to MVRPLGET's caller.

MVFREEIQ: Free ICE Inbound Queue Subroutine

MVFREEIQ is entered from the session abort subroutine to release the session buffers that are on the remote console processor queue (based on $MCONMSG), on the network path manager queue (based on $NPMVINQ), or on the interface control element (ICE) inbound queue. The first part of MVFREEIQ scans the remote console processor queue. Because BSC and SNA buffers are intermixed on the queue and do not use the same chain fields, the routine tests each buffer's type and loads the address of the next buffer from the appropriate chaining field for that type. If an SNA buffer belonging to the session being aborted is found (that is, a buffer whose RPLICE field contains the same value as R5), the buffer is removed from the queue. The contents of the R4 are stored in the buffer's chain field, and R4 is loaded with the address of the buffer. At exit from the scan of the remote console processor queue, MVFREEIQ begins the scan of the network path manager queue at label MVFRENPM. If a buffer belonging to the session being aborted is found, the buffer is removed from the queue. The contents of R4 are stored in the buffer's chain field, and R4 is loaded with the address of the buffer. At exit from the scan of the network path manager queue, R4 contains either the same value it did on entry to MVFREEIQ (assumed to be 0), or the address of the first of a chain of RPLs removed from the remote console processor queue and the network path manager queue.

In the second part of MVFREEIQ, any RPLs pointed to by R4 are concatenated with the chain of RPLs on the ICE inbound queue. If the resulting chain is not empty, the number of buffers queued is determined, and the value in ICEINCT is reduced by that amount. (ICEINCT contains the count of the number of inbound data and synchronous data flow control requests queued on the ICE inbound queue or the line manager's work queue).

Because ICEINCT is increased in the asynchronously scheduled RPL exit (VRPLAOK), ICEINCT is updated through compare and swap of the fullword whose rightmost halfword is ICEINCT. The leftmost halfword is the inbound activity limit field, ICEINLM; the compare and swap instruction operates on the label ICEINLM, although the value of this high-order halfword is not changed when the low-order halfword is decreased.

Finally, MVFREEIQ clears the ICEINHD and ICEINTL fields, removing the buffers from the queue, and exits to MVFREPRG to release the dequeued buffers; MVFREPRG returns control to the MVFREEIQ caller.

MVFREEOQ: Free ICE Outbound Queue Subroutine

MVFREEOQ is entered from any RTAM routine that finds that the current session is being terminated. MVFREEOQ removes buffers from the ICE outbound queue and, if necessary, adjusts RTAM counters that were increased when those buffers were placed on the outbound queue.

The outbound queue pointer, ICEOUTHD, is cleared, removing any existing buffers from the queue. Then, if that queue was empty and no current (not yet queued) buffers exist, control is returned to the invoking routine. If there is a current buffer or the outbound queue was not empty, a single buffer chain is formed, consisting of the current buffer and/or the contents of the outbound queue. Because the RPLs contained in these buffers are not to be scheduled for execution, each buffer is examined and these actions are taken:

- If the RPL represents a response to a data flow control request that was received inbound, it is ignored.

- If the RPL represents the response to an inbound data RU, it is considered to have been sent, because the session is being terminated. The inbound request count, ICEINCT, is decreased by 1.
- If the RPL represents an outbound request (rather than a response), the outbound queue length counter, ICEOUTCT, is decreased by 1 to reflect the fact that this request will not be sent.
- If the outbound request was a data RU that was to be sent as last-in-chain or only-in-chain, in which case it carries a request for definite response, the count of outstanding requests requiring definite responses (ICERSPCT) is decreased by 1.

When all buffers in the chain have been processed, MVFREEOQ exits to MVFREPRG to release the buffers; MVFREPRG returns control to the routine that invoked MVFREEOQ.

MVFRELBF: Release Inbound Buffer Routine

MVFRELBF is used to release teleprocessing buffers and to generate responses for those requests for which a response is required. Upon entry, register MBUF and field DCTBUFAD contain the address of the buffer to be freed. The buffer is disassociated from the owning DCT; the buffer address field of the DCT (DCTBUFAD) is loaded with the address contained in RPLNEXT, the address of the second buffer on the DCT chain. When the passed buffer is the only buffer in the DCT chain, the DCTBUFAD is set to 0. Next, MVFRELBF sets to 0 the RPLNEXT field of the current buffer and enters MVFRESPD.

MVFRESPD: This is a secondary entry point to MVFRELBF, used to release successfully processed input buffers. If a definite response is required, a positive response is sent; if only exception responses are desired, the buffer is simply freed. The exception-response-only indicator, RPLEX, is tested; if the indicator is on, further response testing is bypassed, and MVFRESPD branches to label MVFREEIN to release the buffer without sending a response. If RPLEX is off, entry is at label MVFRESPN.

MVFRESPN: This secondary entry point to MVFRELBF (in addition to being entered from MVFRESPD when the RPLEX indicator is not on) is used to send the required form of positive response before releasing the buffer. The buffer passed to MVFRELBF has been used for a receive operation with a definite response requested. MVFRESPN changes the RPL type (RPLSRTYP) from a receive to a send response and tests the definite response type indicator, RPLVTFL2. If definite response 1 or 2 was requested, MVFRESPN branches to MVRPLRQN to schedule the response RPL for transmission. (JES2 does not support the no-response mode of operation. If neither response type has been requested, MVFRESPN enters MVFREEIN to release the buffer without sending a response. When MVFRESPN is entered, at least one form of definite response is always requested.)

MVFRESPN is used to dispose of unsuccessfully-processed buffers, provided that the caller sets the RPLEX indicator. If a definite response is permitted, the correct type of exception response (type 1 or 2) is sent.

MVFREEIN: First, the RPLDFASY indicator is tested. If it is on, this RPL represents an asynchronous request and a branch to MVFREPRG is taken. Otherwise, the RPL being released represents a data transmission or transmission of a synchronous data flow control request. MVFREEIN decreases by 1 the count of outstanding inbound data RPLs; that count was increased by 1 when the RPL now being processed was placed on the line manager's work queue, $RJECHEQ, by the RPL exit. Next, the status of the ICERCVSP indicator is tested to determine whether the session has been set to receive-specific mode. (All sessions normally operate in receive-any mode. The RPL exit routine issues a RESETSR request to switch a session to receive-specific mode if the session's inbound queue becomes congested. Because JES2 never issues a receive-specific request to VTAM, placing a session in receive-specific mode has the effect of stopping JES2 from receiving further inbound data on that session until the number of inbound requests already outstanding has been reduced to an acceptable level.) If the session is in receive-any mode, MVFREEIN branches to MVFREPRG to release the buffer.

If the session has been placed in receive-specific mode, MVFREELM determines whether receive-any mode can be reentered. Receive-any mode can be entered if the number of outstanding inbound data requests (ICEINCT) does not exceed one-half the session inbound limit (ICEINLM). If the inbound queue still exceeds one-half the session limit, MVFREEIN branches to MVFREPRG to free the buffer. If not, MVFREEIN converts the current buffer into a RESETSR (reset mode to receive-any) request and branches to MVRPLQOB to schedule the request; MVRPLQOB returns control to the caller of MVFREEIN.

MVFREELM: Receive Any Level Check Subroutine

MVFREELM is entered from HASPSPRO and MSNAPDMG. MVFREELM attempts to keep the number of receive-any RPLs available to VTAM at a maximum that is defined by the limit MDCTRALM. (VTAM can move data directly into a user data area without intermediate moves to and from VTAM's pageable buffer pool, provided that a receive RPL is active. It is desirable, therefore, that active receive RPLs always be available.)

At entry, register MBUF is tested. If that register contains the address of a teleprocessing buffer, MVFREELM is functionally equivalent to MVFREPRG and branches to MVFREPR1 to purge the buffer. If MBUF contains 0, this call furnishes an early opportunity to get an additional receive-any RPL into the system if one is needed. MVFREELM accesses the logon device control table (DCT) and compares the total number of buffers currently active in VTAM as receive-any RPLs (MDCTRACT) with the limit MDCTRELM. If the limit has been reached, no additional buffers are assigned, and returns to its caller.

If the limit has not been reached, MVFREELM attempts to get another buffer ($GETBUF TYPE=VTAM), returning to the caller if the attempt is unsuccessful. If a buffer is obtained, however, MVFREELM stores the address of the access control block (ACB) in the buffer's RPL and branches to MVFRECQE.

MVFREOUT: Free Outbound Buffer Subroutine

MVFREOUT is entered from the SNA PUT or CLOSE routines when a buffer originally intended for outbound use is no longer required. The buffer is removed from the remote DCT's RPL chain, and its chain pointer (RPLNEXT) is cleared. MVFREOUT then enters MVFREPRG to release the buffer.

MVFREPRG: Buffer Purge Routine

MVFREPRG is the main entry point for buffer purging. MVFREEIN and MVFREOUT (and routines that enter those routines) dispose of a single buffer. MVFREPRG, however, can dispose of multiple buffers, depending upon the contents of MBUF (the first buffer in the chain, if any) and the RPLNEXT field of the RPL of each buffer in the chain (if any).

At entry, R4 is tested. If that register contains 0, the buffer chain has no members, and return to the caller is immediate. Otherwise, the register contains the address of a buffer to be released and MVFREPR1 is entered.

MVFREPR1: MVFREPR1 is entered from MVFREPRG, or from any SNA routine that can bypass the null-chain test at MVFREPRG because it has already been determined that R4 points to at least one buffer to be released. The address of the logon DCT is obtained from the ACB. If the logon DCT is not active, MVFREPR1's function of making buffers available for receive-any use is not required; MVFREPR1 branches to MVFREPR2 to free the buffer to which R4 points. If the logon DCT is active, MVFREPR1 performs the processing required for buffers in receive-any use.

A buffer in receive-any use is always in the receive-any queue (the queue of buffers pointed to by field MDCTRABF of the logon DCT and chained together through RPL fields RPLFCHN and RPLBCHN). While a buffer is active (that is, while the RPL it contains is being processed by VTAM) the buffer appears only on the receive-any queue. After the buffer is returned to JES2 by VTAM and until it is freed, it also appears on an additional queue, chained through the RPL's RPLNEXT field. (The use of different chain pointers makes it possible for the same buffer to be a member of two different queues. The additional queue on which the buffer appears depends upon the stage of JES2 processing that has been reached, as follows:

- When a receive-any buffer is returned to the RPL request completion exit (VEXITRPL) by VTAM, the exit routine places the buffer on the line manager's I/O completion queue, $RJECHEQ.
- When the line manager accepts the contents of $RJECHEQ for processing (at MBUFSRCH in the general event handler section at entry to the line manager), the buffer becomes a member of the line manager's work queue. This queue consists of the former contents of $RJECHEQ, reordered into first-in-first-out order; the queue header is MBUFQUE in the line manager processor control element (PCE) work area.
- When JES2 processing of the buffer contents is complete, the buffer is placed on the receive-ahead queue, the queue of buffers available for receive-any use when required by VTAM. The queue head is field MDCTRQBF in the logon DCT.

To perform receive-any buffer processing, MVFREPR1 compares the count of receive-any RPLs active in VTAM (MDCTRACT) with the limit for active receive-any RPLs (MDCTRALM). If the count of buffers in active use is lower than the limit, the buffer now being processed is retained, reinitialized as a receive-any buffer, added to the receive-any queue, and passed to VTAM.

If the count of active RPLs is at its limit, the buffer is not required for immediate receive-any use but may be required for future (receive-ahead) use. The count of buffers designated for receive-ahead use, MDCTRQCT, is compared with the limit, MDCTRQLM. If the count is less than the limit, a branch to MVFRECQ0 occurs. Otherwise, both the active count and the count of receive-ahead RPLs are at their limits, and MVFREPR2 is entered.

MVFREPR2: If the logon DCT is inactive, or enough buffers for future use have already been assigned, MVFREPR2 tests to see if a SWEL exists. (A SWEL exists when the buffer represents a logon in progress.) If a SWEL exists, a $SEAS is issued to analyze the completed SAF request and free resources. The return code is ignored; the SWEL, WAVE, and token area are freed.

Following the SWEL processing (or if there is no SWEL), MVFREPR2 loads R4 with the contents of the RPLNEXT field of the current buffer, then issues a $FREEBUF macro instruction to release the current buffer. On regaining control, MVFREPR2 branches to MVFREPRG to process the next buffer on the chain. (If the just-freed buffer was the last or only in the buffer chain, its RPLNEXT field contained 0. MVFREPRG is entered with R4 containing 0 and returns control to the caller.)

MVFRECQE: If the buffer being processed is to be added to the active queue, the high-order byte of its address in R4 is modified, and processing continues at MVFRECQ0. (The effect of this modification is described under "MVFRECQ1" in this section.)

MVFRECQ0: If the count of available buffers is at or below the lower limit, MVFRECQ0 does not release the current buffer but does modify its RPL. RPL options set are ANY (except inbound data from any source), and Q (wait for data if none is immediately available); the RPL is marked as available for synchronous (normal request), asynchronous (expedited request), or response use, and the address of the buffer's data area is placed in the RPL. The normal sequence indication is set for the line manager's use, the address of the logon DCT is stored in the RPL, and the RPLREQ (RPL type) field is set to receive. MVFRECQ0 then enters MVFRECQ1.

MVFRECQ1: The current buffer is added to the head of the active queue pointed to by MDCTRABF, and the contents of R4 are examined. If the buffer just added to the active queue is for immediate use, the high-order byte of R4 will have been set to a nonzero value, as in the case where entry to this point in the routine is by way of MVFRECQE, described earlier. A branch to MVFRECVX occurs, leading to entry to MVRPLXEC and execution of a receive-any request.

If an immediate receive-any is not required, the current buffer is added to the chain of buffers queued for future use-the queued ahead queue. MVFRECQ2 is entered.

Note that a compare and swap instruction is used to update both the MDCTRABF and MDCTRQBF fields, because the RPL exit routine, which is scheduled asynchronously, may change the contents of either field. The RPL exit routine does not change the address portion of MDCTRABF, but may change the active receive-any count (MDCTRACT) in the high-order byte. The exit routine does, however, change both the address portion of MDCTRQBF and the count field (MDCTRQCT) when it removes an RPL from the receive-ahead queue and passes it to VTAM, making it active.

MVFRECQ2: The buffer is added to the head of the queue pointed to by the MDCTRQBF field, and the count of buffers queued for future use is increased by 1. MVFREPRG is entered with the RPLNEXT contents of the current buffer in R4. The next buffer in the chain is added to the appropriate queue, or MVFREPRG returns to the caller if no further buffers remain to be processed.

MVFRECVX: This entry point is used to cause a receive-any request to be presented to VTAM. If a single buffer is passed to this entry point, an immediate branch to MVRPLXBR occurs; the RPL is scheduled and control returned to the caller. If a chain of buffers is involved, MVFRECVX links to MVRPLXBR to schedule each RPL, then branches to MVFREPRG to handle the rest of the chain.

SNA $EXTP Subroutines

The SNA $EXTP subroutines are used to build, process, and analyze: function management headers, outbound requests, compaction headers, SNA NJE RIDs, and data streams.

MVFMHBLD: Function Management Header Build Routine

The function management header (FM header) is a control area, defined by systems network architecture (SNA), that defines the logical device that is to receive a data stream and describes some of the characteristics of that data stream. The FM header is sent within a request unit, like data, and can be accompanied by data. JES2, however, never sends (but may receive) data in the same request unit that contains an FM header.

For SNA remote job entry (RJE), every outbound data set transmitted by JES2 begins and ends with an FM header. The beginning FM header contains a begin destination select (BDS), the selected device type (printer, punch, exchange diskette, basic exchange diskette, or console), and specifies the device's sub-address (for example, 0 for Rn.PR1). In addition, the FM header indicates the outbound data stream characteristics (compression, compaction) and for punch exchange and basic exchange devices, the logical record length of the data to follow.

By architectural rules, the BB and EB indicators mark the beginning and end of a unit of work, where the sender and the receiver have a common interpretation of what constitutes a unit of work. JES2 defines that unit of work as a data set: a unit of job output that would usually be delimited by output separators. The architecture also defines BDS and end device select (EDS) as the delimiters for a data stream to be sent to a single device. JES2 further defines BDS and EDS as marking (like BB and EB) the beginning and end of a data set.

EDS marks the normal end of a data stream, indicating that the complete data set has been transmitted. The suspend device select (SDS) indication marks the temporary end of a data stream, which will be resumed later; the resume device select (RDS) indication marks the resumption of a suspended data stream. Finally, abort device select (ADS) is the indication sent to terminate a data stream before that stream has been fully processed; this indication is given in such a way that transmission of that data stream cannot be resumed.

MVFMHBDS: This entry point ot the FM header build routine is used by the SNA $EXTP OPEN service routine, SNAOPEN, when an outbound data set is opened. The BDS indicator is set, and the bind session image is examined. The compression indicator, DCTPPRES in field MDCTFMT of the device control table (DCT) of the remote device, is turned on if all of the following are true; alternate code (rather than EBCDIC) is *not* to be used, the bind session image indicates that JES2 may compress outgoing data, and the device DCT indicates that the device accepts compressed data. The address of the DCT is stored in the interface control element (ICE), and the ICEALLOC indicator is turned on, indicating that the session is in-use (allocated). MVFMHBLD is then entered.

MVFMHBLD: This entry point is used by all SNA routines when an FM header is apparently required. The bind session image is examined, and if the common protocol section indicates that FM headers are not to be used, a branch to MVFMHNOP occurs, bypassing FM header construction. Otherwise, a prototype FM header is moved into the current RU. The prototype defines the FM header as type

1 (device selection or allocation) and defines its length as 6 bytes (fixed, for type 1). The reserved fields of the header are set to 0 by the prototype.

The prototype is then initialized to form the specific header required. The data stream code (for example, BDS if MVFMHBLD was entered through MVFMHBDS) is stored in the properties field, FMHPROPS, of the FM header; this code is passed in register 1 to MVFMHBLD by the invoking routine. The data stream code is also placed in field ICESNDST, the send status field of the ICE, to indicate that transmission of an FM header is pending. MVFMHBLD examines the DCT and turns on the compression and compaction indicators in the FM header if the data sent to the device can be compressed or compacted. The contents of the MDCTSEL field, which indicates the remote device type and logical sub-address of the device, are placed in the header. For punch, exchange, and basic exchange devices the logical record length of the data to follow is placed in the BDS header. Finally, the length of the FM header is stored in field RPLRLEN of the RPL, and MVFMHNOP is entered.

MVFMHNOP: The ICEOCPND indicator is tested. If it is on, end-of-chain is pending and this FM header is to be sent; MVFMHNOP branches to MVRPLSND, which sends the FM header and returns control to the caller of MVFMHNOP. Otherwise, the FM header has been constructed but is not to be sent at this time, and MVFMHNOP returns control to its caller.

MVREQRSP: Wait for Response Subroutine

MVREQRSP is entered when a SNA routine has sent a request outbound indicating that a definite response is required. MVREQRSP waits for the required response and then determines whether the chain that included the request was processed successfully or unsuccessfully.

MVREQRSP: This entry point is used by the $EXTP PUT and $EXTP CLOSE service routines, SNAPUT and SNACLOSE. It is invoked in two situations.

In the first case, a processor has issued a PUT request, has regained control, and is waiting. HASPSNA receives and records an exception response to the current chain for that processor. When the processor reenters HASPSNA through a PUT or CLOSE request, the corresponding service routine tests the ICECNCEL indicator for the session and finds that an exception response has been received. MVREQRST is entered to wait for the end-of-chain RU to be purged or for the cancel request to be acknowledged.

In the second case, the processor is waiting (not in HASPSNA and between chains) when a SIGNAL is received for this session by the line manager. The line manager sends a stand-alone change-direction request on the session's behalf, requiring a response. Before the response arrives, the processor is posted ($POST) and reenters HASPSNA. At entry, the PUT or CLOSE routine finds the ICEWTRSP indicator on, indicating that a response is outstanding. The service routine enters MVREQRSP to wait for the response before processing the PUT or CLOSE request. (In the case of change direction, receipt of a positive response leads to an implied suspension of the data stream, with a further wait for stream resumption.)

At entry, MVREQRSP checks the ICECNCEL indicator, branching to MVREQRS1 if it is on. If it is off, MVREQCAN is entered.

MVREQCAN: This entry point is used by other routines within HASPSNA that wait at points where an exception response might be received.

After such routines have been posted ($POST), they test the ICECNCEL indicator. If it is on, the normal processing path for such a routine is abandoned and MVREQCAN is entered.

MVREQCAN transfers the calling routine's return address from R6 to R2 and links to MVFREPRG to discard any buffers queued to the DCT for the remote device. The DCTBUFAD field in the DCT is cleared, and the MVREQRS1 routine is entered.

MVREQRS1: MVREQRS1 checks the ICEWTRSP indicator. If it is off, the chain is complete and MVREQRS1 branches to MVRSTATO to update the session status. If ICEWTRSP is on, the return address in R6 is saved in DCTEWF and MVREQWT is entered to wait for a response, which is the purging of the end-of-chain RU or acknowledgment of the CANCEL request.

MVREQWT: This entry point is used by the RPL scheduler, MVRPLSND, which has already saved R6 in the DCT. The ICEWTRSP indicator in the session's send status field, ICESNDST, has already been turned on to show that a response is outstanding for this session. MVREQWT waits ($WAIT WORK) for the response to be received. The processor control element (PCE) representing the processor on whose behalf MVREQWT was entered is posted by the line manager when the response is received, and control is returned to MVREQWT. On regaining control, MVREQWT tests the ICEWTRSP indicator. If the indicator is still set, the event that caused the $POST was not the receipt of a response for which MVREQWT was waiting, and the wait is repeated until ICEWTRSP is turned off.

MVRSTATO - Responses to Outbound Requests: Processing continues at this label, which is also used as an entry point by the request end processor. The ICECNCEL indicator in field ICESNDST is tested. If that indicator is on, the request was not accepted and a branch to MVRSTATC occurs. Otherwise, a branch to MVRSTATE is taken, with the following qualification.

If the accepted outbound chain carried a begin-bracket, the ICERTRPD indicator representing the ready-to-receive-pending state is cleared. Once an outbound bracket has been successfully initiated, it no longer serves any purpose for the remote work station to send a RTR.

MVRSTATI - Responses to Inbound Requests: This entry point is used by SNA routines to dispose of an inbound buffer by using that buffer to send an outbound response to the request. At entry to MVRSTATI, register 15 contains the address of the HASPSNA subroutine that disposes of the inbound buffer. This subroutine saves the return address of caller, enters the caller-designated buffer disposal routine, reloads the return register, and continues. MVRSTATI checks to see if this is an SNA NJE session; if it is, MVRSTATI branches to label MSNAXCLS to terminate the session. Otherwise, the receive-status field ICERCVST in the ICE is examined. If the ICECNCEL indicator is off, an incoming RU chain was received successfully and accepted, and processing continues at MVRSTATE.

If the ICECNCEL indicator is on, the caller of MVRSTATI is in the process of cancelling a chain of RUs because an error has been detected or the sender has requested that the chain be cancelled. In this case, the ICEINCHN indicator is tested. If ICEINCHN is on, the chain has not yet ended, and MVRSTATI branches to MVREQBUF to look for the next buffer in the chain. The session remains in purging-chain state. If ICEINCHN is off, the chain is ended. Updating the session status to rescind the bad chain continues at MVRSTATC and MVRSTATE.

MVRSTATC: This provides an entry to MVRSTATE for chains that indicate that an exception response was received or sent. MVRSTATC replaces the address of the good chain transition table in register 14 with the address of the bad chain transition table.

MVRSTATE: MVRSTATE is common to inbound and outbound request processing and to successful and unsuccessful completion. MVRSTATE stores a session state indicator in ICESNDST for outbound requests, or ICERCVST for inbound requests, and branches to the appropriate entry point for further processing. Figure 3-8 summarizes, with respect to the contents of the RU that was sent or received, the session state that is indicated and the entry point at which processing resumes. These entry points are described on the following page.

	SUCCESSFUL		**UNSUCCESSFUL**	
RU Contained[1]	**New Session State[2]**	**Processing Continues at[3]**	**New Session State[2]**	**Processing Continues at&**
RDS request	ICEINSTR	MVRSTIST	ICENOFMH	MVRSTSDS
EDS request	ICENOFMH	MVRSTEND	ICEINSTR	MVRSTIST
BDS request	ICEINSTR	MVRSTIST	ICENOFMH	MVRSTPAB
ODS request	ICENOFMH	MVRSTEND	ICENOFMH	MVRSTPAB
SDS request	ICENOFMH	MVRSTSDS	ICEINSTR	MVRSTIST
ADS request	ICENOFMH	MVRSTADS	ICEINSTR	MVRSTIST
CDS request	ICEINSTR	MVRSTIST	ICEINSTR	MVRSTIST
Out-of-stream data	ICENOFMH	MVRSTNOP	ICENOFMH	MVRSTNOP
In-stream data	ICEINSTR	MVRSTIST	ICEINSTR	MVRSTIST

[1] **RU Contents**

RDS: Resume destination select
EDS: End destination select (normal end)
BDS: Begin destination select
ODS: One-chain destination select (BDS and EDS)
SDS: Suspend destination select
ADS: Abort destination select (abend)
CDS: Continue destination select

- Out-of-stream data: Any data RU received or sent when no data stream is established; this is an implied beginning or resumption of a data stream.
- In-stream data: Any data RU received or sent as part of an ongoing data stream (the normal case).

[2] **Session States**

- The ICEINSTR state exists when a data stream has been established on this session. BDS, CDS, or RDS has been sent and accepted; CDS, SDS, EDS, or ADS has not been sent or has been sent but rejected.
- The ICENOFMH state exists when no data stream is currently established on this session. BDS or RDS has not been sent and accepted. No previous data stream existed, or a previous SDS, EDS, or ADS was sent and accepted.

[3] **FM-Header-Pending States**

The bit combinations B'0000', in the 4 left-most bits of field ICERCVST or ICESNDST, represent states where an FM header is pending (has been sent or received but not yet confirmed by a positive response) as follows:

0000 - Resume destination select
0001 - End destination select
0010 - Begin destination select
0011 - One-chain destination select
0100 - Suspend destination select
0101 - Abort destination select
0110 - Continue destination select

The FM-header-pending states for RDS and BDS resolve to in-stream after successful chain completion, or to out-of-stream after chain failure. The FM-header-pending states for EDS, ADS, and SDS resolve to out-of-stream after successful chain completion and to in-stream after chain failure. The FM-header-pending state for ODS resolves to out-of-stream regardless of the outcome. The FM header pending state for CDS resolves to in-stream regardless of the outcome.

Figure 3-8. MVRSTATE Decision Table

MVRSTPAB: The processor that issued the $EXTP request to cause this RU to be sent or received is to be aborted. The DCTABORT indicator is set, and processing resumes at label MVRSTEND.

MVRSTADS: A request to abort a data stream has been processed successfully. MVRSTADS sets the DCTDELET indicator to cause the processor to delete the job that produced the current data set. MVRSTADS then branches to MVRSTEND.

MVRSTIST: This entry point is reached either when acceptance of a good chain causes the in-stream state to be entered, or when rejection of a bad chain causes a reversion to the in-stream state. The state change set by the FM header has already been set by the MVRSTATE routine; MVRSTIST handles the *implied* stream state change represented by end-bracket or change-direction indicators:

- For good chains, an end bracket is an implicit abort of the current stream; a change-direction is an implicit suspension of the current stream. (In this usage, explicit means with an FM header and implicit means without an FM header.)
- For bad chains, an end-bracket carried on the CANCEL RU instruction is an implicit abort of the current stream; a change-direction carried on the CANCEL RU is an implicit suspension of the current stream. The bracket direction indicators carried on the failing chain itself are ignored.

MVRSTIST tests the ICEEBPND and ICECHDIR indicators. If both are off, a branch to MVRSTNOP occurs. If either is on, suspension of the data stream is implied. The ICE is marked as out-of-stream with no FM header pending.

MVRSTIMP: Register 0, which is stored in the ICESDCT field, is loaded with the implied suspend indication (ICESIMPL and ICESUSPD both on). MVRSTSDS is entered at a point past where MVRSTIMP reinitializes register 0.

MVRSTSDS: If this entry point is entered directly, rather than from label MVRSTIMP, an explicit suspend is indicated. The suspend indication, ICESUSPD, is loaded into register 0. In either case, suspend indicator DCTSUSP in field MDCTSTAT is turned on, and the DCT for the remote device is pushed down onto the session's suspend queue. The address of the DCT is placed in field ICESDCT on the ICE, and the balance of the queue is chained from the DCT's MDCTSDCT field. The implicit or explicit suspend indication in register 0 is stored in leftmost byte of ICESDCT. Processing continues at MVRSTEND.

MVRSTEND: The pointer to the ICE in the DCT is cleared, if entry is from other than label MVRSTSDS; this is because the remote device is no longer the user of this session. Regardless of the point of entry, the pointer to the DCT in the ICE is cleared. The allocated indication in the ICE, ICEALLOC, is turned off, and the ICENOFMH indicator is tested. If ICENOFMH is on, the suspension was caused by a request to start a different data stream in the same direction of flow, typically, a BDS request not preceded by an SDS request. If ICENOFMH is off, this is an explicit suspension, or an implicit suspension caused by receipt of an end-bracket or a change-direction request. A branch to label MVRSTNOP is then taken.

MVRSTNOP: If ICECHDIR indicates that a change of direction is to be performed, the current setting of the ICEINBND, ICEOUTBD, and ICEREVFL indicators is reversed, changing the direction of flow. After the change of direction, or if no change was requested, the ICEBBPND and ICEEBPND indicators are tested. If both are on, a change or bracket state is being requested, and the setting of ICEINBRK is reversed. Next, any pending conditions (ICEBBPND, ICEEBPND, and ICECHDIR) are cleared, and the ICECNCEL flag is tested. Processing continues at MVRSTRMO, if this chain is not being cancelled.

If ICECNCEL is on, showing that the chain is being cancelled, it is cleared. The caller's base register, MBASE1, is then tested; if the high-order bit is on, the caller is a processor that issued a $EXTP CLOSE request or if the caller is the line manager's request end processor. A branch to label MVRSTRMC then occurs. The MDCTTYPE indicator, DCTPOUTB, is checked to positively identify the caller as a processor that has issued an $EXTP OPEN or $EXTP PUT request. If DCTPOUTB is off, the data stream is inbound, and the routine branches to label MVRSTRMC. Otherwise, the caller's return register is adjusted to point to the abnormal (condition code 0) exit routine, MVTAMXAB, to indicate to the calling processor that the OPEN or PUT request has failed. Finally, DCTFLAGS are tested to distinguish between status indication X'0811' (associated with the 3775 CANCEL key) X'0825' (associated with the 3775 STOP JOB key), and other error types. If either of the two specific status indications is found, the DCTFLUSH indicator in field MDCTSTAT is set to cause any further PUT processing (for instance a printer-deleted message and the trailer page) to be suppressed.

MVRSTRMC: Processing of a cancelled chain is concluded by a branch to MVRSTRM1 to find the next data stream. If the session is still within the bracket, the processing of a cancelled chain is concluded by a branch to MVRSTRM1 to start the next data stream. If the failure was caused by an actual error rather than bracket contention, a branch to MICEREQZ is taken to send a null RU to end the chain.

MVRSTRM0: Entry at this label normally occurs after successful chain completion. If the session is not within brackets, MVRSTRM0 branches to MICEAEB to abort any suspended streams possibly cut short by a premature end bracket.

MVRSTRM1: If the ICEINSTR indicator shows that a data stream is still established, control is returned to the caller. If no stream has been established, MVRSTRM1 enters MVREQSTR to attempt to start a new stream.

MVDECFMH: SNA Function Management Header Decode Routine

MVDECFMH is entered when a FM header is received through normal SNA receive processing. Upon entry, the FM header is checked to ensure that it has arrived as a first-of-chain request unit and that the common bind protocols allow the use of FM headers. Next, the type of FM header is determined and the appropriate routine is invoked according to the following table.

FMH type	Routine
0	Invalid
1	MVDTYPE 1
2	Invalid
3	MVDTYPE 3
4	MVDTYPE 4

MVDTYPE1: Type 1 FM Header Processing Routine

MVDTYPE1 is entered when a type 1 FM header (data stream header) is received through the normal SNA receive processing. First, the specific type of FM header is determined. Valid type 1 FM headers are:

RDS	- Resume destination select
EDS	- End destination select
BDS	- Begin destination select

ODS	- One chain destination select (Begin and End)
SDS	- Suspend destination select
ADS	- Abort destination select
CDS	- Continue destination select

The receive session status indicator, ICERCVST, is updated to reflect the FM header pending state. (Refer to Figure 3-8 under "FM Header Pending States".) The FM headers are processed as follows.

If the header is an RDS, the ICEALLOC indicator is tested to ensure that the session is not allocated. If the session is allocated, a 1008 negative response (invalid FM header) is returned. If the RDS is valid, the suspended remote DCT chain, pointed to by ICESDCT, is searched to find the remote DCT whose data stream is to be resumed. Once the remote DCT is found, it is allocated to the session by requeuing the DCT to the ICE at ICERDCT and turning on the ICEALLOC indicator. Next, the waiting processor is posted ($POST) to resume the inbound data flow for its device.

If the FM header is a BDS or ODS, a new device will be allocated to the session. If an active destination already exists and the FM header is not for a console, the current destination is suspended (implied suspend). If the FM header is for a console and an active destination already exists, a 1008 negative response is returned. Once the FM header is validated, the remote DCT chain, pointed to by the line DCT, is searched to find an eligible remote DCT that can accept the incoming data stream. Once the remote DCT is found, the ICE and the remote DCT are chained to each other and the ICEALLOC indicator is turned on to indicate that the session is allocated. The corresponding processor is then posted ($POST) to start the inbound data flow for this device.

If the FM header is a CDS, the data to follow is to be treated as a continuation of the current data stream. The CDS header must flow only in an in-chain state. If the FM header is a CDS header and is for console, it is rejected with a 1008 negative response returned.

The RDS, BDS, ODS, and CDS FM headers contain the compression and compaction characteristics of the data to follow. If the common protocols section of the bind are compatible, these characteristics are reflected in the remote DCT. If they are not, a 1008 negative response is returned. The BDS, ODS, and CDS FM headers also contain the logical record length of the data to follow. This is recorded in the remote DCT, at MDCTRECL. If the logical record length is zero, a default of 80 is used.

If the FM header is an SDS, the ICEALLOC indicator is tested to ensure that the session is allocated. If the session is not allocated a 1008 negative response (invalid FM header) is returned. The receipt of this type of header eventually causes the remote DCT to be marked as suspended and to be placed on the ICE suspend queue pointed to by ICESDCT. The corresponding processor is caused to wait ($WAIT) and eventually resumes processing upon receipt of a RDS for the data stream.

If the FM headers are EDS or ADS, the ICEALLOC indicator is tested to ensure that the session is allocated. If not, a 1008 negative response (invalid FM header) is returned.

Receipt of an EDS causes the associated remote processor to close normally and the job to become eligible for processing. Receipt of an ADS causes the associated remote processor to close abnormally and the job to be deleted. The session is deallocated.

MVDTYPE4: NJE FM Header Receive Processing Routine

MVDTYPE4 is entered when a type 4 FM header (NJE management header) is received through normal SNA receive processing.

Upon entry, the FM header is checked for proper length and whether the header is the only type 4 header received for this session. Next, the RU buffer size is selected. The allowable range is 256 to $BFSZSNA. The value received in the header (FMHBFSIZ) is compared to the value specified locally ($BFSZSNA); the smaller of the two values is used. Finally, the feature (FMHFEAT) and the record identification format (FMHRIDFM) bytes are tested. The FMHOPTMZ and FMHNETOP flags must be on in the FMHFEAT flag byte. FMHRIDFM must have the FMHRID1 flag on. The feature byte is moved to the application table (APT) entry (APTFEAT). If the type 4 NJE header is followed by a concatenated type 3 compaction header, MVDTYPE3 is entered to validate the header.

After processing the type 4 header (and if necessary, the concatenated type 3 header), MVFRESPD is invoked to send a positive response to the header, and MNJENPMC is invoked to check for a possible call of the network path manager initial sign-on routine (HASPNSNR).

MVDTYPE3: NJE FM Compaction Header Processing Routine

MVDTYPE3 is entered when a type 3 compaction header is received. The header is validated, and the compaction data is used to build a decompaction table in an acquired area whose address is saved in the ICE (ICEDCPT).

After successfully validating a type 4 header, the ICEFMHR4 indicator is turned on to indicate successful receipt of the type 4 header. The ICEFMHRV indicator (successful receipt of all headers) is turned on at the same time if the FMHFEAT byte indicates that the other node does not support compaction (FHMCMPTN=0), or if a type 3 header is ever received. However, only the type 4 header need be received to invoke the network path manager initial sign-on routine (HASPNSNR).

MNJENPMC: Network Path Manager Call Subroutine

MNJENPMC is called upon receiving and/or sending a positive response to an NJE FM header. Note that this routine is called from receive processing when an FM header 4 is received. For send processing, both FM header 4 and FM header 3 (if required) must have been sent and responded to before MNJENPMC is called. If this is the high node (the node with the alphabetically-higher node name), the path manager initial sign-on support routine (HASPNSNR) in the HASPNET module is called to send an NJE sign-on record to the other node.

Return offsets are 0 for an error return (unable to initiate sign-on due to system resource shortage) and plus 4 for a normal return.

MNSRANAL: SNA NJE RID Analysis Routine

MNSRANAL examines an NJE record identifier (RID) that has been decompressed/decompacted into the RPLRID field by the sneak-a-peek routine (MNSKPEEK) and takes the appropriate action for the record headed by that RID. MNSRANAL is invoked by the SNA RECEIVE processing routine, the $EXTP GET routine, or the $EXTP CLOSE routine.

MNSRANAL examines the RID of the current record. If the RPL has not been through the decompress/decompact processing, the sneak-a-peek routine is called to decompress/decompact the first RID into RPLRID. If the RID indicates that this is an NJE data record, another check is made for a all-system message record (NMR), which is used to send operator commands and messages. If it is an NMR, the RPL is removed from the ICE inbound queue, the allocated-to-remote-console-processor indicator (ICERCON) in the NJE flag byte (ICENJEF2) is turned on, and MVDCNSLN is invoked to queue the RPL to the remote console processor (via $MCONMSG), to post ($POST) the processor PCE, and to return to the caller of MNSRANAL.

If the RID indicates that this is an NJE data record for a job/SYSOUT receiver, the transmitter and receiver DCTs chained off the line DCT is searched to find the appropriate DCT; the RPL address is stored in the receiver DCT (DCTBUFAD); the receiver's PCE is posted ($POST); and control is returned to the invoking routine. If the DCTERMNR flag is on (cancelled by the receiver), the data record is skipped over (via a call to MNSKPREC), and the next RID (if any) is examined.

If the RID indicates a control record, a check is made to determine whether it is an NJE network topology record. If it is, the RPL is removed from the ICE inbound queue and is queued immediately for the network path manager via $NPMVINQ; the network path manager's PCE is posted ($POST); and control is returned to the appropriate invoking function.

If the RID indicates a stream control record, the NJE stream control record processor (MNSSCRP) is invoked to process the record. Upon return from MNSSCRP, the RPL is removed from the ICE inbound queue, released (stream control records are always the last logical records in an RU), and the next RPL (if any) on the ICE inbound queue is processed.

MNSSCRP: NJE Stream Control Record Processor

MNSSCRP processes stream control records obtained by the NJE RID analysis routine (MNSRANAL). It performs the appropriate stream control function, depending upon the type of stream control record. See Figure 3-9 for these functions. MNSSCRP branches to the appropriate stream control processor routine depending on the stream control ID.

MNSKPREC: SNA NJE Skip-a-Record Routine

MNSKPREC is called from the RID analysis routine (MNSRANAL) when the decompressed/decompacted record identifier (RID) indicates that the record is a job or SYSOUT data record and the matching receiver DCT has its DCTERMNR (receiver cancel) flag on. The record currently pointed to must be skipped over in the buffer. The buffer cannot be ignored because there may be records for other receivers past this record.

MNSKPREC saves registers via a $SAVE macro instruction, loads the saved GET routine registers from the RPL work area (registers were saved after sneak-a-peek processing on the RID), initializes register RPC (processor buffer count) to the contents of the RIDLEN field plus 1 (length of record to be skipped), initializes register RPP (the processor buffer pointer) to 0 (indicating that the record is to be skipped), turns on the SGWSKPRC flag and branches into the SNA GET routine at label MVGMVE to skip over the record.

Figure 3-9. Stream Control Record IDs and Functions

Stream Control ID	Meaning	Function Performed
X'80'	Hold stream	Find matching transmitter DCT and turn on DCTHOLDS flag
X'90'	Request to allocate job or SYSOUT stream	Find matching receiver DCT and post ($POST) for $EXTP OPEN; if not found, queue RPL to path manager which sends X'B0' (permission denied)
X'A0'	Permission to allocate granted	Find matching transmitter DCT and post ($POST) transmitter PCE
X'B0'	Permission to allocate denied, or receiver cancel	Find matching transmitter DCT, turn on DCTERMNR flag, and post ($POST) transmitter PCE
X'C0'	Acknowledge EOT	Find matching transmitter DCT and post ($POST) transmitter PCE
X'D0'	Release stream	Find matching transmitter DCT and turn off DCTHOLDS flag

Note: The logic at exit from the GET routine ensures a return to the caller of MNSKRPEC if the SGWSKPRC flag is on.

MNSKPEEK: SNA NJE Sneak-a-Peek Routine

MNSKPEEK is called from the RID analysis routine (MNSRANAL) the first time the RPL is examined and from the SNA $EXTP GET routine (RTAMVGET) after decompressing/decompacting (of skipping) a record prior to returning that record to the caller. MNSKPEEK checks to see if there is more data in the RPL; if there is, it invokes $EXTP GET logic to decompress/decompact the next 3 bytes of data into the RPL work area (RPLRID).

MNSKPEEK saves registers via the $SAVE macro instruction and checks the SWGNJE flag to see if this RPL has ever been through $EXTP GET processing. If it has, it loads registers from the RPL GET work area to restore the GET processing status.

MNSKPEEK loads register RPP (the processor buffer pointer) with the address of RPLRID (a 3-byte field in the RPL work area), and loads in register RPC (the processor buffer count) with a 3 (the length of RPLRID). It then turns on the sneak-a-peek flag and tests the SWGNJE flag. If the SWGNJE flag is on, MNSKPEEK branches to the $EXTP GET routine at label MVGMVE. If the flag is off, SWGNJE is turned on and MNSKPEEK branches to the $EXTP GET routine at label MVGSCN.

Note: The GET routine exit logic ensures a return to the caller of MNSKPEEK if the SGWSNKPK flag is on.

MSCSEND: SNA NJE Stream Control Send Routine

MSCSEND is called by the SNA $EXTP OPEN routine (RTAMVOPE) and by the SNA $EXTP CLOSE routine (SNACLOSE) to send an NJE stream control record.

On input, register 0 contains a 3-byte stream control RID (right-adjusted). R4 contains the address of an RPL (if available) to which the stream control record is to be added. If R4 is 0, MSCESEND gets and initializes an RPL.

MSCSEND adds a compression string control byte (SCB) to the front of the RID, adds the 4-byte record to the RPL buffer, invokes the RPL send routine which sends the data or queues it for output, and returns to the appropriate back level caller via R6.

SNA Session Control Subroutines

The SNA session control subroutines are used by the SNA $EXTP routines, scan routines, and buffer processing routine for controlling the activity of SNA sessions. They perform the basic session management functions associated with the interface control element (ICE), which shows the JES2 internal representation of SNA sessions.

MICEREQU: Queue ICEs to the Line Manager Subroutine

MICEREQU is defined between the labels MICEREQU and MICEUPDT. This is a multiple-entry subroutine (MICEREQU, MICEREQ1, MICEREQ2) that is called from the line manager and RTAM to queue ICEs to the line manager ICE exit queue (MICEQUE).

MICEMELT: Subroutine to Free ICEs

MICEMELT is used after a session has been terminated (CLSDT completed) in order to return the ICE to the pool of free ICEs ($ICETRAY). For NJE, any storage acquired for a decompaction table is freed via a FREEMAIN macro instruction. Because requests may still be in VTAM, this routine examines the outstanding inbound and outbound request parameter list (RPL) counters in the ICE before freeing the ICE. If the counters indicate the RPLs are still outstanding, the ICE is queued to the line manager ICE exit queue (MICEQUE) by calling MICEREQ1. The ICE is then reexamined by MICEMELT at every line manager dispatch (at least once per second) and is freed when no more RPLs are outstanding.

MICEGIBB: Delayed Request Subroutine

MICEGIBB is defined between the labels MICEGIBB and MICEGPND. This is a multiple-entry subroutine (MICEGIBB, MICEGIEB, MICEGICD, MICEGSND), which is used by the line manager to issue certain requests (begin bracket, end bracket, change direction, CLSDST, TERMSESS, and null request unit) when no teleprocessing buffer is immediately available. MICEGIBB queues the ICE for the requesting session to the line manager ICE exit queue (MICEQUE) by calling MICEREQ2 until a buffer is available. The request is tried again by the line manager by calling MICEGIBB at each dispatch.

MICETRAP: Session Abort/Abandon Routines

MICETRAP is defined between the labels MICETRAP and MICEAFRQ. MICETRAP, MICEABRT, MICEABDN, MICENET, and MICEAEB are used to terminate and clean up session usage and/or to disconnect (unbind) sessions. The line manager uses these routines primarily for error-recovery purposes, and most entries disconnect the session as well as terminate data flow. RTAM also uses these routines to terminate session data flow, particularly when an unexpected end bracket (EB) is received.

Normally, all of these routines cause any RJE device DCTs that are still connected to the ICE (pointed to by ICERDCT or ICESDCT) and any network transmitter device to abort their data flows and purge any buffers they may own (pointed to by DCTBUFAD). Any buffers the ICE points to (inbound, outbound, and special sequence) are also purged. These routines, however, attempt to save one buffer to be used for CLSDT in the ICE indicators (ICEFLAGS). This buffer indicates that the session is to be unbound (ICEABORT, ICEDRAIN, or ICECLOSE). When processing for a remote or network device is completed, the processor control element (PCE) associated with the device is posted ($POST) for I/O and WORK as required. MICENETA is entered to reset the ICE status flags. Next, MICEADED is entered at label MICEAFRQ to determine the status of the session and to clear the inbound and outbound session queues as required.

After this clean-up action, MICETRAP examines the indicators (ICEFLAGS) and, if necessary, schedules execution of the session disconnect routine, MICEDISC. Because MICEDISC can only be executed by the line manager, and because the MICEABRT is also used by the $EXTP routines, control cannot always be passed directly from MICEABRT to MICEDISC. If MICEABRT or MICEABDN has been called by RTAM, it uses MICEREQ2 to queue the ICE to the line manager exit queue (MICEQUE) before returning to the caller. The line manager then calls the disconnect routine for RTAM at its next dispatch.

MICEDISC: Session Disconnect Routine

MICEDISC is defined between the labels MICEDISC and MICEDREM. This routine is entered either directly from the session abort/abandon routines (MICEABRT and MICEABDN) or from the ICE exit scan routine of the line manager (MICEPROC). MICEDISC is used to disconnect the ICE from the line DCT and, if this is the last session active on the line, to disconnect the RJE device DCTs and network device DCTs from the line DCT. Before disconnecting the ICE from the line DCT, all statistical data collected in the ICE is added to the appropriate line DCT count fields, and message $HASP210 is issued to inform the operator that the session has been disconnected (logged off). MICERDSC resets all pointer connections between the device DCTs and the line DCTs.

For a SNA NJE session, processing starts at label MICERDSC, which calls the path manager subroutine HASPNDCN to clean up the network control blocks. The transmitter/receiver DCTs, disconnected from the line, are returned to the pool of free network DCTs ($NETDCTS). The DCTPHASP flag is turned off, indicating that the line is no longer being used for an NJE connection.

For RJE, the SAF token is freed, all pointers to the RJE device DCTs and the line DCT are reset in the RAT (unless the line is leased or shared), and the checkpoint record remote status bits (pointed to by $RMTSON) are reset. When the process of disconnecting the RJE and NJE sessions is complete, exit point MICEXIT (for exit 18) is taken (if enabled) giving control to an installation exit routine. The installation exit routine returns and the disconnect message $HASP203 is issued. Indicators in the line DCT and RJE device DCTs are then reset. If the line DCT command flags (DCTFLAGS) indicate that the line should be drained (DCTSPAT), $FREUNIT and $DORMANT macro instructions are executed; otherwise, the now idle line DCT is requeued to the line manager's idle SNA line DCT queue (MSNAIDL). If other sessions are still active on the line, the ICE is dequeued from the line DCT, and the line DCT active sessions count (MDCTSCNT) is decreased; the remote terminal is not disconnected. After the ICE is disconnected from the line DCT, MICECLOS is entered to actually disconnect (unbind) the session.

MICECLOS: Session Close (CLSDST) Subroutine

MICECLOS is defined between the labels MICECLOS and MICETERM. This is a two-entry subroutine (MICECLOS and MICEBLD) that is used to issue a CLSDST or a TERMSESS VTAM request to disconnect an SNA session. The primary entry, MICECLOS, checks flags in the ICE (ICESTAT) to determine if a CLSDST or a TERMSESS is required. For NJE sessions, MICECLOS determines whether the application is primary or secondary. Because the secondary application cannot issue a CLSDST, an indication is set to show that a TERMSESS sequence is required. If a CLSDST or a TERMSESS is required and a buffer is available (usually preserved by MICEABDN or MICEABRT), the CLSDST or TERMSESS request is built in the buffer and passed to VTAM by calling the RPL scheduler (MVRPLXEC). If, however, no teleprocessing buffer is available, MICEGBUF is called to obtain a buffer from the teleprocessing buffer pool (via $GETBUF) and reenter MICECLOS at its secondary entry (MICECBLD). If no buffers are available in the buffer pool, MICECLOS queues the ICE to the line manager ICE exit queue (MICEQUE). The line manager then tries the request again at its next dispatch. Eventually, the CLSDST is built by MICECBLD or the TERMSESS is built by MICETERM and scheduled by the RPL scheduler.

MICESHUT: Send Request Shutdown Subroutine

MICESHUT is called when the DCT for an NJE line shows a drain pending ($P line issued), a sign-off pending in the absence of the restart flag ($T line, D=Q), or a request-shutdown data flow control received from the secondary logical unit (LU).

On entry to MICESHUT, the ICENJEF1 flag is tested to determine whether the application is the primary or the secondary LU on the session.

If the application is the secondary LU and a request for shutdown of data flow control has not been sent to the primary LU, MICESHUT branches to MICEGBUF to get an RPL. At label MICESHT4, MICESHUT builds the request for shutdown of data flow control RU in the RPL and branches to label MVRPLOCR to send the request. If a request for shutdown has been sent (ICEQUIET indicator present), control is returned to the caller.

If the application is the primary LU, a check is made to determine whether any transmitters and receivers are connected to the line. If none are connected or if the devices that are connected are no longer in use, a branch is taken to MICESHUT2 when the ICERSHUT flag is reset, the ICEDRAIN and ICECLOSE flags are set on (request for session closedown), and a branch is taken to label MICENETA.

If any of the network devices are still in use, the line manager line scan request is set on, and control is returned to the caller.

MICETIME: Session Delay Interval Subroutine

MICETIME is defined between the labels MICETIME and MICEDLAY. This multiple-entry subroutine (MICETIME, MICEDLAY) is used to cause a session to remain idle (outbound allocation prohibited) for a specified interval. The main entry, MICETIME, always uses the interval specified by either the WAIT parameter on the TPDEF initialization statement, or the WAITTIME parameter of the RMTnnn JES2 initialization statement contained in the HASP communications table ($WAITIME) or the remote attribute table (RATWTIME). The secondary entry, MICEDLAY, allows an interval to be passed to MICETIME in register 1.

SMF Recording Subroutines

These subroutines are used exclusively by the line manager to produce SMF records for significant RJE events. The SMF records they produce are described in *System Management Facilities (SMF).*

MSMFSTRT: SMF Line Start Record Subroutine

MSMFSTRT is entered whenever any SNA line is started in order to write a type 47 SMF record for the line start event. As with all the SMF recording subroutines, MSMFSTRT calls MSMFWRIT to actually write the SMF record.

MSMFSTOP: SMF Line Stop Record Subroutine

MSMFSTOP is called to record the line stop event any time a SNA line is stopped or restarted. It writes a type 48 SMF record that includes the statistical totals kept in the line DCT since it was first started. MSMFSTOP passes control to MSMFTERM to complete the record.

MSMFTERM: Line Stop/Remote Disconnect SMF Record Routine

MSMFTERM is entered either to write a type 48 remote disconnect event record or (from the SMF line stop subroutine, MSMFSTOP) to complete a type 48 line stop SMF record. MSMFTERM passes control to the common line manager SMF record write routine (MSMFWRIT) to queue the SMF record for recording.

MSMFWRIT: Common SMF Write Routine

MSMFWRIT is a common routine used by all SMF recording routines to queue SMF records for recording. MSMFWRIT completes all SMF records by adding the EBCDIC device/remote identifier; it calls the JES2 common SMF queue routine, which issues the JES2 $QUESMFB macro instruction to invoke the SMF writer.

Common DCT Initialization/Termination Routines

These routines are used by the SNA $EXTP routines, scan routines, and buffer processing routine to allocate, deallocate, initialize, and release DCTs. They provide common DCT initialization services and perform statistical totaling associated with SMF recording.

MLDCTGET: Line/Logon DCT Common Allocation Routine

MLDCTGET is used to allocate all SNA line and logon DCTs to the line manager; it allocates the DCTs by issuing a JES2 $GETUNIT macro instruction and then increases the PCE active count with a $ACTIVE macro instruction. After the DCT is allocated, its statistical counters are reset, and MLDCTGET exits to one of three routines that initialize either logon or line DCTs (MLOGSTRT or MSNASTRT respectively).

MLDCTREL: Common DCT Free Routine

MLDCTREL commonly is used to free line and logon DCTs after they are drained by issuing a $FREUNIT macro instruction and decreasing the PCE active count. However, the code in the line manager does not only perform this function; the session abort/abandon routines (MICEABRT and MICEABDN) also issue $FREUNIT and $DORMANT when an SNA remote terminal is being disconnected from a line that was drained while the line was active.

MSNASTRT: SNA Line DCT Initialization Routine

MSNASTRT is entered by the common line/logon DCT allocation routine (MLDCTGET) whenever an SNA line DCT is allocated to the line manager. It places the newly allocated line DCT on the line manager idle SNA line queue (MSNAIDL) and calls the SMF start line subroutine (MSMFSTRT) to record the event with a type 47 record.

MLOGSTRT: SNA Logon DCT Initialization and Open Routine

MLOGSTRT is entered from the common logon/line DCT allocation routine (MLDCTGET) after a new logon DCT has been acquired for the line manager. It initializes the DCT and causes the associated VTAM access control block (ACB) to be opened. MLOGSTRT initially acquires a teleprocessing buffer and, using MLOGPOST, signals the VTAM ACB subtask to open the ACB. The subtask uses the pre-allocated buffer to issue a SETLOGON OPTCD=START VTAM request after the ACB is open.

MLOGPOST: VTAM ACB OPEN/CLOSE Subtask Interface Routine

MLOGPOST is the primary interface routine for the VTAM ACB OPEN/CLOSE subtask. Whenever it is necessary for the subtask to OPEN or CLOSE a VTAM ACB associated with a logon DCT, MLOGPOST is called to place that logon DCT on a special HASP communications table (HCT) queue ($VLOGQUE) and post the ACB subtask.

MLOGSTOP: SNA Logon DCT Termination and ACB CLOSE Routine

MLOGSTOP is defined between the labels MLOGSTOP and MLOGSDEQ. MLOGSTOP is used whenever it is necessary to drain a logon DCT. All receive-any pre-queued buffers pointed to by the logon DCT (MDCTRQBF) are freed by placing them on the line manager buffer queue ($RJECHEQ) with a simulated VTAM RTNCD/FDBK2, indicating that they were purged by CLOSE processing. After the logon DCT has been removed from the line manager active logon DCT queue (MSNALOG), the VTAM ACB subtask interface routine (MLOGPOST) is called to signal the subtask to close the associated ACB.

MSNASTOP: SNA Line DCT Termination Routine

MSNASTOP is defined between the labels MSNASTOP and MSNASDSC. MSNASTOP is used to drain SNA line DCTs; it removes the line DCT (which must be idle) from the MSNAIDL queue and, using the common logon/line DCT free routine (MLDCTFRE), releases the DCT from the line manager use. A call is made to MSMFSTOP to write a type 48 SMF record that contains the statistical totals accumulated in the line DCT during operation.

HASPSLOG: Active SNA Logon DCT Scan Routine

Selected for execution by external trigger from the command processor (HASPCOMM), HASPSLOG is executed for each active SNA logon DCT. The primary purpose of this scan routine is to process the restart ($E) and drain ($P) commands for logon DCTs. When a restart command is detected for a logon DCT, the session abort routine (MICEABRT) is called to force all sessions associated with the specified application interface to disconnect immediately. Drained logon DCTs are processed by calling MLOGSTOP only if no session is currently associated with them (MDCTICE=0).

HASPSLNE: Active SNA Line DCT Scan Routine

HASPSLNE is executed for all active SNA line DCTs at least once every second. Before checking idle sessions, the scan routine performs the disconnect interval check by comparing the noted time of last activity (MDCTIMOK) in the line DCT against the current time. Also, each line DCT is checked to see if operator restart ($E) has occurred. Lines without recent activity (as determined by the DISCINTV=operand of the RMTnnn initialization statement) or that have been restarted by the operator, are disconnected using the session abort routine (MICEABRT) to terminate all of their associated sessions.

If this is an NJE session, a check is made to see if an orderly shutdown is required. If the DCTDRAIN or DCTSOFF flag in the line DCT is on, or if the ICERSHUT flag in the ICE is on, the orderly shutdown routine (MICESHUT) is invoked to initiate an orderly shutdown. Otherwise, the MDCTNJEH flag is checked to see if any NJE headers are to be sent and, if so, the NJE FM header send processing routine (MSLNENJH) is entered. Otherwise, a branch is taken to MSLNENXT to process the next line DCT.

If this is an RJE session, HASPSLNE has several interrelated functions, but it is mainly concerned with allocating idle SNA sessions to outbound data flows whenever the job output table (JOT) posts a signal that output work or messages are queued for transmission to the remote terminal.

MSLNTDCT performs its main function of initiating outbound transmissions by examining all active SNA line DCTs that indicate possible work. Each idle session, represented by an ICE on the line DCT ICE chain (MDCTICE), is considered for allocation to outbound work. Some ICEs in the chain are not eligible because they are already allocated (ICEAVAIL), held (ICEHOLD), or waiting for the expiration of a time delay. ICEs in time delay are further checked to see if the delay interval has expired and they will be made available. For each eligible ICE in the chain, a subscan of all the RJE device DCTs is done. If MSLNTDCT determines that work exists, as indicated from $#POST, it $POSTs the device. If it cannot be determined whether work exists, this scan uses the $#GET service to determine if any outbound work is queued for the RJE device. This subscan uses the $#GET queue service macro instruction to determine if any outbound work is queued for the RJE device. If so, the ICE is pre-allocated to the RJE device DCT, and the associated processor PCE is posted ($POST). If no work is found for any RJE device associated with an active line, the possible work indicator is reset, and the scan continues with the next active SNA line DCT.

The final processing performed by HASPSLNE is special outbound interrupt processing (MSLNTICE). If messages are queued for an LUTYPE1 terminal with the console feature, and no unallocated sessions are available to transmit these messages, HASPSLNE attempts to locate an active outbound session that can be interrupted for this high-priority transmission.

MSLNENJH: NJE FM Header Send Processing Routine

MSLNENJH is entered by the line manager when, while scanning the SNA line DCTs, it finds the MDCTNJEH flag in the line DCT set, indicating the need to send NJE FM headers.

Upon entry, the pointer to an outbound compaction table (ICECPT) is checked; if the pointer is 0, MSLNENJH has been entered to build and send an NJE FM header (type 4) and an immediate branch is taken to MNHGETR to get an RPL. Otherwise, MSENDCH is entered to send a compaction header (type 3).

MSENDCH: The ICEFMHR4 indicator is checked to determine if a type 4 header has been successfully received from the other node. If a type 4 header has not been received, a branch is taken to MNHDELAY to request a timer interrupt line manager dispatch and continue active SNA line scanning with the next line DCT. If a type 4 header has been received, the application table (APT) features byte (APTFEAT) is checked to see if the other NJE node supports compaction (FMHCMPTN indicator on). If the NJE node does not support compaction, the outbound compaction table pointer (ICECPT) is zeroed out, MNJENPMC is called to check for a possible invoking of the path manager sign-on routine (this is done only if a positive response has been received for the type 4 header sent to the other node), the MDCTNJEH indicator is turned off, and the active SNA line scan continues with the next line DCT. If the NJE node supports compaction, a branch is taken to MNHGETR to get an RPL.

MNHGETR: The $GETBUF macro instruction is executed to acquire an RPL. If successful, the RPL is linked to the appropriate control blocks and ICECPT is checked; if nonzero, a branch is taken to MNHBLDC to build a compaction header (type 3). Otherwise, a type 4 NJE header is built in the RPL. If a type 3 compaction header is needed, a pointer to the appropriate compaction table is stored in the ICE (ICECPT), before invoking MVRPLSND to send the type 4 header, indicating that a type 3 header is to be sent. Upon return from MVRPLSND, ICECPT is checked to see if a type 3 header is to be sent, and if so, a branch is taken to the top of the NJE FM header send processing routine (MSENDCH). Otherwise, the routine exits to MSLNENXT to continue the line scan with the next DCT.

MNHBLDC: MNHBLDC is entered from MNHGETR after getting an RPL when a compaction header is to be built. It invokes MVFMHCPT to build a type 3 compaction header and to send it. Upon return from MVFMHCPT, the MDCTNJEH indicator is turned off and an exit is taken to label MSLNENXT to continue the active SNA line scan with the next line DCT.

HASPSIDL: SNA Idle Line DCT Scan Routine

HASPSIDL is executed only when requested by the command processor to process drain ($P) commands for idle SNA line DCTs. Drained idle line DCTs are terminated by calling MSNASTOP.

HASPSUNT: Inactive DCT Scan Routine

This is the line manager scan routine responsible for acquiring all newly started ($S) SNA lines and logon DCTs. HASPSUNT examines all logon and line DCTs on the common DCT chain (DCTCHAIN) starting with the first logon DCT which is pointed to by the HCT ($LOGNDCT). If a line or logon DCT is located that is available but not yet allocated, HASPSUNT acquires and initializes it for the line manager by calling the common line manager DCT allocation routine (MLDCTGET). MLDCTGET passes control to the appropriate DCT initialization routine, which establishes DCT values and adds the DCT to a line manager active DCT queue.

HASPSACB: Line Manager Logon DCT Exit and ACB Subtask Completion Scan Routine

HASPSACB is used to service the line manager processing related to logon DCTs; it is executed by the common scan driver in HASPRTAM (MSCANEXT) whenever any logon DCT is present in its work queue (MLOGQUE). Work is entered into MLOGQUE from two sources: the VTAM ACB subtask and the VTAM TPEND exit routine. HASPSACB determines the type of service a queued logon DCT requires by examining an exit code (MDCTXCOD) in the high-order byte of the logon DCT exit chain field (MDCTEXIT).

The ACB subtasks queue logon DCTs to the work queue (MLOGQUE) when they have completed OPEN or CLOSE processing. After successful OPEN processing, the logon DCT is added to the SNA active logon DCT queue (MSNALOG), and MSMFSTRT is called to write a type 47 SMF record. Unsuccessful OPEN processing causes message $HASP092 to be issued, and the logon DCT to be drained. Completed CLOSE processing uses MSMFSTOP to write a type 48 SMF record and may drain the logon DCT if necessary (CLOSE processing failure or DCTDRAIN set); otherwise, the logon DCT is rescheduled for OPEN processing by calling MLOGSTRT ($E LGNn processing).

When a logon DCT is queued by the TPEND exit, a secondary exit code is used to determine what kind of TPEND action has occurred. The three possible types of TPEND are: orderly shutdown ('Z NET' VTAM operator command), quick shutdown ('Z NET, QUICK' VTAM operator command) and VTAM abends. For quick and abend termination, JES2 immediately aborts all sessions associated with logon DCT by calling the session abort routine (MICEABRT) for all ICEs on the logon DCT active ICE queue (MDCTICE). Orderly shutdown causes all sessions immediately to be marked as draining (indicated by DCTDRAIN + DCTINUSE), which causes them to disconnect whenever data flow is finished (end bracket).

HASPSICE: ICE Exit Scan Routine

HASPSICE is responsible for processing several types of work, mainly session control associated with ICEs. The ICEs are queued to a common service queue (MICEQUE) from the sources listed below and are distinguished via an exit code (ICEXTCOD) in the high-order byte of the ICE exit chain field (ICEXTCHN). The main scan driver (MSCANEXT) selects the scan for execution whenever ICEs are queued to MICEQUE.

Code	Queued by	Name
1x	Logon exit	(VEXITLGN)
2x	Autologon RAT scan	(MRATPROC)
3x	Delayed request routine	(MICEGBUF)
4x	RELREQ exit	(VEXITRLR)
5x	SCIP exit	(VEXITSCP)
6x	LOSTERM exit	(VEXITLST)
7x	Disconnect routine	(MICEMELT)
8x	$SN exit	(HASPSNET)
9x	SCIP exit (BIND)	(VEXITSCP)
Ax	SCIP exit (UNBIND)	(VEXITSCP)
Bx	NS exit	(VEXITNS)

The logon exit, the autologon RAT scan, and the $SN exit use the ICE exit function (codes 1, 2, and 8, respectively) to begin the logon cycle. ICEs queued for logon are processed by a routine within the ICE exit scan routine called MICELOGN. If MICELOGN is entered because of an RJE logon, it initiates the logon cycle by getting a teleprocessing buffer and initializing it with an INQUIRE VTAM request. This INQUIRE request is used to obtain the device characteristics of the terminal (session) that is to be connected. The completion of this INQUIRE request causes the SNA buffer processing routine (HASPSPRO) to continue the logon or autologon cycle. If MICELOGN is entered because of a start networking ($SN) command to establish an SNA session with another NJE node, MICELOGN gets a teleprocessing buffer and using a hard-coded BIND image, initializes the RPL for an OPNDST ACQUIRE request and branches to the VTAM interface to issue the OPNDST request.

As previously described in "MICEGBUF: Delayed Request Routine, " when certain VTAM requests must be issued by the line manager and no teleprocessing buffer is available, the request must somehow be tried again after buffers are available. MICEGBUF performs this function by queueing the ICE passed to it by its caller to the ICE exit queue. The line manager then recalls MICEGBUF from the ICE exit scan at each dispatch until the request is complete and the ICE is removed from the queue.

All three of the VTAM session-related exits (LOSTERM, RELREQ, and SCIP) as well as the $SN pseudo exit (HASPSNET) use the ICE exit function of the line manager to perform the bulk of their JES2 processing. In this way, the amount of code executed outside the control of the JES2 main task is minimized. Because there are several LOSTERM conditions, the exit code for LOSTERM contains, in addition to the LOSTERM identifier (6x), the entry code that was passed to the LOSTERM exit by VTAM. This code is adjusted to fit into the low-order 4 bits of the exit code. Most LOSTERM conditions result in the termination of the session by calling the session abort routine (MICEABRT). The SCIP exit (scheduled by VTAM) is treated as an unconditional abort request because it is only scheduled by VTAM to indicate an interface programming error. The RELREQ service is used by applications outside of JES2 to request that JES2 release its use of a specific VTAM logical unit (LU.) The ICE exit routine responds to RELREQ conditions by calling a session control routine (MICETRAP) that causes the session to begin the disconnect process after any current activity (end bracket) is completed.

Like the delayed request routine (MICEGBUF), the ICE free routine (MICEMELT) must sometimes wait until VTAM buffers, which were cancelled by CLSDST, are purged from the network. By queuing the ICE to the line manager ICE exit queue (MICEQUE), MICEMELT reschedules its own execution at each line manager dispatch. Then, MICEMELT reexamines the ICE outstanding RPL counters when all requests are complete and frees the ICE.

MICEBIND: SNA NJE OPNSEC Processing Routine

MICEBIND, an ICE exit routine (invokable at an entry in the ICE exit routine branch table — MICEBTAB), handles a BIND request from another application and responds by issuing a VTAM OPENSEC request to establish an SNA application-to-application session. Input to MICEBIND is an ICE on the MICEQUE (obtained by the VTAM SCIP exit routine.)

Upon receiving control, MICEBIND gets an RPL and searches the application table for a match against the application specified in the BIND request unit. If it is unable to find the application name, if it is already in session with this application, if the OPENSEC macro instruction has already been sent or is pending, or if the BIND parameters are unacceptable, MICEBIND issues a SESSIONC macro instruction to reject the BIND request.

If MICEBIND determines that it can accept the BIND, it obtains an available line DCT, calls APPLDYN to find an application program table (APT) entry with the name in ICESYMB (or to find a node with that name and build a dynamic SPT), initializes the RPL for an OPNSEC request, and branches to the VTAM interface to issue the OPNSEC.

MICEUBND/MICENS: SNA NJE Unbind and Cleanup RU Processing Routine

The MICEUBND/MICENS ICE exit routines are invoked by HASPSICE when the exit code in the ICE indicates that UNBIND was received in the SCIP exit, or a CLEANUP RU was received in the $SN exit. HASPSICE uses a branch table (MICEBTAB) to locate the address of MICEUBND/MICENS. MICEUBND/MICENS checks the ICETERMS indicator to determine whether the application requested TERMSESS. If TERMSESS was not requested, MICEUBND/MICENS branches to MICEABDN to free any resources associated with the session. If TERMSESS was requested, the ICETSC indicator is checked to determine whether the TERMSESS request completed. If the TERMSESS request completed, a branch is taken to MICEABDN;

otherwise, the ICE is requeued via MICEREQ2, pending completion of the TERMSESS request.

HASPSRAT: Autologon RAT Scan Routine

HASPSRAT is primarily responsible for beginning automatic logon processing. A companion routine, HASPSSAL, completes the automatic logon initiation. The automatic logon feature of JES2 SNA RJE support makes JES2 capable of automatically contacting and connecting remote terminals for which output work or messages are queued. The scan is executed whenever an operator command enabling the autologon feature ($T RMTn or $S RMTn) is executed or the device has been set to autologon mode by the RMTnnnn initialization statement. The line manager also internally requests an autologon scan when certain RJE resources are freed or when output work is queued. However, because some significant events may take place externally (for example, through VTAM) and without any indication to the JES2 system or because messages have been queued for the remote, the scan is dispatched every 32 seconds, by default. An installation can specify the interval in a range between 10 and 600 seconds. When the scan is dispatched, it examines each entry in the remote attribute table (RAT) to determine if the corresponding remote terminal is eligible for automatic connection (autologon).

Remote terminals that are eligible for autologon are currently disconnected SNA remote terminals for which a logical unit name (LUNAME = operand of the RMTnnn initialization statement) has been specified, and for which the operator has entered a $S RMTn or $T RMTn command or the device has been set to autologon mode by the RMTnnnn initialization statement. Remote terminals that are already connected can become eligible for autologon any time after they are disconnected. Once an eligible remote terminal is located, a test is performed to determine if the terminal should and can be connected. Remote terminals that have been started (via $S RMTn) have an autologon attempt made regardless of whether work exists for any of their RJE devices. However, those in autologon mode are connected only if output or messages are queued for transmission, as set by the $#POST routine. The RAT is first checked to see if output exists, and if it does exist the autologon attempt is made immediately. If no output exists, the scan looks for messages for the remote terminal and if they exist, the logon is attempted. If they do not exist, JES2 decides whether a full queue scan is necessary. A full queue scan is only necessary the first time through autologon or after a disconnect. If no queue searches are required, JES gets the next RAT. If output is found on the queue searches, the autologon is attempted.

The autologon attempt begins with a SAF call to determine if the remote terminal has sufficient authority to logon. If so, HASPSRAT then examines the availability of line DCTs, ICEs, and logon DCTs to establish the connection. If any resource is unavailable, the connection is not attempted at this time. Additionally, if the shared queue (used for MAS configurations only) remote status bits (pointed to by $RMTSON) indicate that the remote terminal is connected to another system in the configuration, it is not connected. Remote terminals that are eligible for connection, but for some reason are not connected, are reexamined for possible connection at a later dispatch of HASPSRAT.

EXIT point MALGXIT is taken (for exit 18) to invoke the installation exit routine. If the return code from the exit routine is 8, the autologon terminates. Otherwise, HASPSRAT initiates a SAF VERIFYX call (without password checking), marks the remote "signed on" in $RMTSON and steps to the next RAT. Parameters passed to MSAFCHK indicate that the SWEL is to be queued to the active SWEL queue (MLMASWLQ).

HASPSSAL: Post-$SEAS Processing for Autologon

HASPSSAL is responsible for completing the initiation of autologons started by HASPSRAT. It is invoked when the $SEAS call issued by HASPSRAT has completed and the SWEL has been moved from the MLMASWLQ queue (active SWELs) to the MLMPSWLS queue (SNA-posted SWEL).

HASPSSAL removes the posted SWELs from the queue and, for each, calls $SEAS to analyze the results of the SAF call. If the SAF return code is 8, the autologon is aborted, the RAT is marked "not autologon", the remote is marked "not signed-on" in $RMTSON, and the DCT is freed. Otherwise (SAF RC=0 or 4), the PCEs are obtained, and the pre-connection of the RAT, DCT, and ICE (which was started by HASPSRAT) is completed. The ICE is then queued to the ICE exit queue to begin the logon cycle.

HASPSPRO: SNA Buffer Processing Routines

Buffers containing request parameter lists (RPLs) (that is, SNA teleprocessing buffers) represent requests that have been completed or that have not been completed and are to be reissued. Further buffer processing is performed according to the state of the session, as indicated by the ICEINDEX field, which indicates the current state, and the RPLSEQTP field, which reflects the state of the session at the time the request was issued.

Before the common register environment is set up for the request-end routines, receive-any requests are singled out by testing the RPLSEQTP field and removed from the receive-any queue. (The preceding and following RPLs on the "receive any" queue are located directly through the RPLBCHN and RPLFCHN pointers.) MVFRECHK is called at this time to check if another "receive-any" should be issued to maintain the number of simultaneously active receives at the prescribed limit.

The registers commonly used throughout the request-end are then initialized. The VTAM return code is used as an index to a branch table to determine the following points where processing continues:

- MSNAPAOK: The request was completed normally (+0).
- MSNAXRSP: An exception response or request was received (+4).
- MSNAXTMP: A temporary storage shortage condition in VTAM was encountered (+8).
- MSNAPDMG: A data integrity problem was encountered (+12).
- MSNAPENV: An environment error, or an error that resulted in lost data integrity was reported (+16).
- MSNAPBUF: A logic or timing error was encountered (+20).

MSNAPAOK: Normal Request Completion

Requests that completed normally are further divided into special sequence requests and normal sequence requests. Special sequence requests are those that do not occur during normal data transmission, such as OPNDST, CLSDST, and INQUIRE. These are detected by testing RPLSEQTP for a nonzero major code and passing control to MSNAPSPC, where another branch table identifies the specific routine for each request type. Normal sequence requests consist of SEND, RECEIVE, and RESETSR macros issued to control inbound pacing. In this case control passes to four subsequence entry points with the following labels:

- MSNAPRCV: Receive complete
- MSNAPSND: Send complete
- MSNAPRCS: Receive complete and RESETSR CS (session reset to continue-specific mode) issued to hold pacing
- MSNAPSCA: RESETSR CS (session reset to continue-any mode) issued to release pacing

For special sequence requests, control passes to MSNALOGN for logon sequence requests and MSNACLOS for session closedown requests. MSNACLOS is an alternate name for MICEMELT which, after ensuring that all outstanding RPLs for the session have completed, disconnects an ICE from the logon DCT and frees it. MSNALOGN branches on the minor code portion of RPLSEQTP to pass control to five subsequence entry points with the following labels:

- MSNALDEV: INQUIRE DEVCHAR (device characteristics) complete; issues INQUIRE SESSPARM
- MSNALPAR: INQUIRE SESSPARM (session parameters) complete; issues OPNDST
- MSNALOPD: OPNDST (open destination) ACCEPT processing complete; session now ready for normal sequence requests
- MSNALODQ: OPNDST ACQUIRE (for SNA NJE application-to-application session) complete; session now ready for normal sequence
- MSNALOPS: OPNSEC (open secondary) complete; session now ready for normal sequence

MSNALPAR: Inquire Session Complete Routine

MSNALPAR obtains control upon successful completion of the INQUIRE SESSION PARAMETERS request. MSNALPAR is normally entered twice for the logon of an RJE device. The first entry starts the logon and initiates a SAF call. Since the line manager PCE cannot $WAIT, the $SEAS call issued by MSAFCHK (which issues the prior SAF call) specifies WAIT=NO. The second entry occurs when the SAF call has completed. On this entry, the results of the SAF call are analyzed by $SEAS and the logon is completed.

On the first entry, MSNALPAR locates the user logon data and invokes exit 18 to allow an installation exit routine to affect the logon parameters and subsequent processing. Based on the exit's return code, MSNALPAR will:

- 0 or 4 - Continue with normal logon.
- 8 - Terminate the logon.

- 12 - Continue with the logon using the exit-supplied remote (RAT) and the passwords provided by the remote user.
- 16 - Continue with the logon using the exit-supplied remote (RAT) and no password checking.

If the logon is to continue (RC is not 8), MSAFCHK is called to issue a SAF VERIFYX call with a user ID from the RAT supplied by the exit (RC=12 or 16) or teh RAT corresponding to the remote name specified on the logon (RC=0 or 4). If the exit return code is 16, SAF password checking is bypassed; otherwise, the old password (and new password, if present) is passed as well. Normally, the SAF call will not complete before MSAFCHK returns, so MSNALPAR returns requesting that the buffer be requeued.

When the SAF cal' completes, MSNALPAR is entered for the second time. This time, MSNALPAR issues a $SEAS to analyze the completed SAF request. Based upon the SAF return code, MSNALPAR:

- 0 - Completes the logon.
- 4 - Falls back to non-SAF password checking, checking the password against the one in the RAT (if any).
- 8 - Rejects the logon.

After logon processing completes, exit point MSNALTXT2 (for exit 18) is taken to invoke an installation exit routine. The installation exit routine returns with a continue normal logon processing indication (return code 0 or 4).

MSNALODQ/MSNALOPS: OPNDST/OPNSEC Completion Processing Routine

MSNALOD/MSNALOPS obtains control upon successful completion of an OPNDST or OPNSEC macro instruction via an entry in the logon sequence branch table (MSNALTAB). MSNALODQ/MSNALOPS updates control blocks to indicate that the SNA session has been established and turns on the appropriate indicator in the line DCT (MDCTNJEH) to trigger the transmission of the JES2 NJE FM headers.

MSNAPRSP: SNA Positive Response Handling Routine

MSNAPRSP checks for an NJE session and invokes MNJENPMC when all function management (FM) headers have been successfully sent and received. MNJENPMC checks for a possible call of the network path manager initial sign-on routine (HASPNSNR).

HASPVTAM: Open/Close ACB Routine

HASPVTAM communicates with VTAM, performing functions equivalent to the OPEN ACB and CLOSE ACB macros. Because those functions require that the invoking routine wait, HASPVTAM executes as a subtask, attached by HASPINIT when JES2 is started. HASPVTAM specifies $STABEND as its $ESTAE routine. $STABEND performs no recovery for HASPVTAM but does provide problem determination information and terminates the HASPVTAM subtask if a failure occurs.

HASPVTAM: At initial entry (upon being attached by HASPINIT), HASPVTAM performs standard and RTAM initialization. It clears $SNAECB, the event control block (ECB) through which the line manager indicates to HASPVTAM that work is available, and posts $PSNAECB to indicate to HASPINIT that the subtask has been attached and is active. HASPVTAM then enters VACBWAIT to wait for work.

VACBWAIT: VACBWAIT issues a WAIT macro instruction, specifying $SNAECB as the event control block. This is the point at which VACBWAIT waits for work after each processing cycle. $SNAECB is posted by the line manager when a $S LGN1 or $P LGN1 operator command has been received. Each time it receives control, VACBWAIT tests the $SYSEXIT indicator in the $STATUS field of the HASP communications table (HCT) to determine whether JES2 is terminating; if it is, VACBWAIT frees its save area and returns to its caller. Otherwise, VACBWORK is entered.

VACBWORK: VACBWORK clears its ECB ($SNCECB) and determines whether the logon device control table (DCT) is on the $VLOGQUE. If it is not, VACBWORK branches to VACBWAIT to wait for work. If the logon DCT is on the queue, it is removed, and VACBPROC is entered.

VACBPROC: The DCTSINON indicator in field MDCTSTAT of the logon DCT is tested. If that indicator is on, the ACB has already been opened and this is a close ACB request. VACBPROC branches to VACBCLOS. Otherwise, a register in initialized with the open ACB complete exit code, and open processing is performed.

A GETMAIN macro instruction is issued to acquire storage, and an ACB is built and initialized. Among the information placed in the ACB is the password, (if any), which is stored in the logon DCT, and the address of VACBINVL (in this routine), to which VTAM is to return control if the ACB is found to be invalid. An OPEN macro instruction is issued; VACBPROC tests the ACB indicators to determine whether the OPEN processing was successful. If open processing was not successful, VACBFAIL is entered. Otherwise, the DCTSINON indicator in the logon DCT is turned on to show that the ACB has been opened; processing then continues.

The logon DCT, the request parameter list (RPL) to which it points, and the ACB are interconnected. The RPL is initialized to indicate that SETLOGON is to be performed, and VACBPROC links to the VTAM interface. When control is returned, VACBPROC tests the completion code, branching to VACBPOST to post the line manager if the request completed successfully. If the request did not complete successfully, it is tried again, unless the error is such that it cannot be attempted. In that case, DCTABORT is set on. The completion code returned by the VTAM interface is combined with the SETLOGON action code in register 2, and VACBPOST is entered to post the line manager.

VACBINVL: This entry point is used by VTAM when open processing is impossible because of an error in the ACB passed to the VTAM interface. A return code of X'20' is stored in register 15, indicating that the OPEN request failed, and control is returned to VTAM.

VACBCLOS: This entry point is used for CLOSE processing. At entry, the DCTSOFF indicator in the logon DCT is tested. If that indicator is off, CLOSE processing is not required, and VACBCLOS branches to VACBWAIT to wait for more work. Otherwise, VACBCLOS stores the CLOSE ACB complete exit code conditionally in register 2 and issues the CLOSE macro instruction. If the CLOSE processing is successful, VACBCLOS branches to VACBFREE to free the ACB; otherwise, VACBFAIL is entered.

VACBFAIL: The completion code returned by VTAM is combined with the exit code in register 2, and the DCTABORT indicator in the logon DCT's MDCTSTAT field is turned on. VACBREE is entered.

VACBFREE: The ACB is freed, through a FREEMAIN macro instruction, and the pointer to the ACB in the logon DCT is cleared. Processing continues at VACBPOST.

VACBPOST: VACBPOST branches to VDCTPOST to post the line manager, then returns to VACBWAIT to await further work.

VTAM API Routines

The following text describes the processing of the application program interface (API) routines.

VEXITLGN: Logon Exit Routine

VEXITLGN is entered from the VTAM application program interface (API) when an SNA remote work station logs on to VTAM.

VEXITLGN examines the logon device control table (DCT) and returns immediately to VTAM (in effect, ignoring the logon attempt) if the access control block (ACB) is available or if the DCTDRAIN indicator is set, indicating that shutdown of the JES2/VTAM interface is in progress. Otherwise, the logon counter(field MDCTLOGN) is increased by 1, and the next available interface control element (ICE) is removed from the $ICETRAY. (If no ICE is available, a branch to VLOGFAIL occurs.) The ICE is cleared and then initialized with the address of the logon DCT and the information returned by VTAM (the address of the symbolic name of the remote work station, stored in ICESYMB). Finally, VEXITLGN sets the logon exit routine identifier in register 2 and exits to VICEPOST to post the line manager. VICEPOST returns control to the VTAM API.

VLOGFAIL: If the logon is rejected because no ICE was available, contents of DCT field MDCTNICE is increased by 1. A GETMAIN macro is issued, and the $BFRBLD macro instruction is issued to construct a request parameter list (RPL) in the space obtained. The RPL is initialized as a CLSDST request and passed to the VTAM API. If the CLSDST fails but the VTAM return code indicates that it should be reissued, the CLSDST request is reissued until it succeeds or fails permanently. If the CLSDST request is successful or is unsuccessful and cannot be tried again, the RPL is freed, and the logon exit returns control to VTAM.

APPLDYN

APPLDYN is called from MICEBIND and SNASNET to find an application program table (APT) entry matching the application name that is passed. If an application name is passed without a node number, the existing APTs are searched. APTs are chained, not contiguous. Return codes 12 or 16 are returned to the caller if a match is found but the node is invalid. If a match exists and the node is valid, the APT and the NIT are returned to the caller, with a return code of zero. If no matching APT exists, the NITs are searched; if the OWNNODE NIT matches, return code 12 is returned. If another NIT matches, and APT is built, the APT and the NIT are returned with a return code of zero. If no NIT matches, the return code is 8.

If a node number is also passed, along with the application name, the existing APTs are searched. If a matching APT is found, its node is reset and it is returned, even if its node value is invalid. The NIT address is also returned (even if it is invalid). If no matching APT exists, one is built and its node is set and returned (even if it is invalid). The NIT address is also returned (even if it is invalid). Return code 4 is set if a GETMAIN for an APT fails.

SNASNET: $SN Exit Routine

When a start networking command ($SN A=applname) is issued to establish a SNA session with another NJE node, the NJE command sub-processor (HASPCNT1 in HASPCOMM) invokes SNASNET.

SNASNET verifies that the VTAM interface is usable and the the DCT line is valid. If a line DCT is not supplied by the caller, this routine obtains one from the SNA idle queue. It calls APPLDYN to find or build an APT. Then an interface control element (ICE) is obtained from the queue of available ICEs ($ICETRAY), initialized, and updated with the appropriate $SN command information. The line specified by the operator (or any qualifying SNA line, if none was specified) is removed from the SNA idle queue (MSNAIDL) and added to the SNA active queue (MSNALNE). A pointer to the line device control table (DCT) is stored in the ICE (ICELDCT). Finally, SNASNET sets the $SN exit routine identifier in register 2 and enters VICEPOST. VICEPOST queues the ICE to the MICEQUE queue, posts ($POST) the line manager, and returns control to the HASPCOMM command sub-processor.

VEXITLST: LOSTERM Exit Routine

VEXITLST is entered from the VTAM API when a SNA work station becomes a lost terminal (that is, when the SNA work station is not available). In the NJE environment, the LOSTERM exit at the primary logical unit is scheduled when a TERMSESS is issued by the secondary logical unit.

VEXITLST examines the logon DCT and returns immediately to VTAM if the ACB is unavailable. Otherwise, VEXITLST performs basic RTAM register initialization, loads register 2 with the LOSTERM exit routine identifier combined with the reason code passed by VTAM, and enters VICEPOST to post the line manager. VICEPOST returns control to the VTAM API.

VEXITNS: Network Services Exit Routine

VEXITNS is entered from the VTAM API and is scheduled for both primary and secondary applications when a session outage occurs or as a result of a VARY INACT command with the forced or reactive operand.

VEXITNS examines the request parameter list (RPL) passed by VTAM. If the RPL is a cleanup request unit (RU), indicating that VTAM has terminated the session, the routine performs basic RTAM register initialization, loads register 2 with the $SN exit routine identifier, and enters VICEPOST to post the line manager. VICEPOST returns control to the VTAM API.

If the RU passed by VTAM is a network services procedure error RU, VICEPOST returns to VTAM with registers 15 and 0 set to 0.

If the RU received is not one of the expected types, VICEPOST returns to VTAM with register 0 set to 0 and register 15 set to 4.

The registers on entry are as follows:

Register	Contents
1	Address of parameter list First word - Address of ACB Second word - Communications identifier (CID) of session Third word - User field/ICE address Fourth word - Reserved Fifth word - Address of read-only RPL (cleanup RU)
2-13	Unpredictable
14	Return address
15	Entry address

Return registers are as follows:

Register	Contents
0	Zeros
15	Return code 0 - Good return 4 - RU passed by VTAM not recognized by application

VEXITTND: TPEND Exit Routine

VEXITTND is entered from the VTAM API when VTAM is terminating either because the operator has requested a normal or a quick shutdown or because of abnormal termination.

VEXITTND examines the logon DCT and returns immediately to VTAM if the ACB is unavailable. Otherwise, VEXITTND performs basic register initialization, loads register 2 with the TPEND exit routine identifier combined with the reason code passed by VTAM, and enters VICEPOST to post the line manager. VICEPOST returns control to the VTAM API.

VEXITSCP: SCIP Exit Routine

VEXITSCP is entered from the VTAM API when a session control request unit (RU) is received by VTAM for the JES2 application.

VEXITSCP examines the logon DCT and returns immediately to VTAM if the application control block (ACB) is not available; otherwise, VEXITSCP performs standard register initialization and examines the request parameter list (RPL) passed to it by VTAM. This is a read-only RPL containing the session control RU received by VTAM for JES2.

If the request is a request recovery (RQR), VEXITSCP sets the SCIP exit (RQR) identifier in register 2, enters VICEPOST to queue the ICE to the MICEQUE queue, and posts ($POST) the line manager.

When VEXITSCP is entered due to the receipt of a BIND request unit (resulting from another application issuing an OPNDST ACQUIRE macro instruction), an ICE is obtained from the $ICETRAY, initialized, and updated with the appropriate BIND information.

VEXITSCP sets the SCIP exit (BIND) identifier in register 2, enters VICEPOST to queue the ICE to the MICEQUE queue, and posts ($POST) the line manager. If no ICE is available, the BIND is rejected via a SESSIONC request.

If the SCIP exit routine is entered due to the receipt of an UNBIND request unit (received when the primary session issues a CLSDST macro instruction), VEXITSCP sets the SCIP exit (UNBIND) identifier in register 2 and enters VICEPOST; VICEPOST then returns control to the VTAM API.

All other session control requests are ignored by VEXITSCP; control is returned to the VTAM API with no further action.

VEXITRLR: RELREQ Exit Routine

VEXITRLR is entered from the VTAM API when another application, possibly a diagnostic application, has requested ownership of a remote work station logged on to JES2 through VTAM.

VEXITRLR examines the logon DCT and returns immediately to VTAM if the ACB is unavailable. Otherwise, MDCTSNCT, the count of active sessions, is obtained from the logon DCT. The address of the associated terminal's symbolic name, passed by VTAM, is compared with the symbolic name address in field ICESYMB of each active ICE. If no match is found, control is returned to VTAM. If a match is found, VEXITRLR places the RELREQ exit routine identifier in register 2 and enters VICEPOST to post the line manager; VICEPOST then returns control to the VTAM API.

VEXITRPL: Request Completion Exit Routine

VEXITRPL is entered from the VTAM API at the completion of any RPL request. The primary function of VEXITRPL is to respond to the completion of a send or a receive request by scheduling immediately the next applicable send or receive request. Because VEXITRPL executes under a service request block (SRB), causing each request completion to trigger the next initiation within the SRB, VEXITRPL significantly reduces dispatching and housekeeping overhead.

In the course of processing, VEXITRPL queues every completed request on the line manager's channel end queue. $RJECHEQ (through a branch to VBUFPOST), continues processing. VEXITRPL takes only that action and returns control to VTAM in these cases:

- If the request that completed was a special sequence request. (Special sequence requests are those that do not occur during normal data transmission, such as INQUIRE SESSPARM, or OPNDST. Normal sequence requests are send and receive requests, including RESETSR requests).
- If a send or receive request completes, but the RPL return code indicates an error higher than X'04'.
- If a receive request completes, and the queue of receive-ahead buffers is empty.
- If a send request completes, and the queue of outbound requests for the session is empty.

In all other cases, VEXITRPL links to VBUFPOST to cause the current buffer to be passed to the line manager for further disposition. VEXITRPL then starts a new send or receive operation.

The principal difference between receive and send processing is that on completion of a receive request, any buffer on the receive-any queue, MDCTRQBF, can be passed to VTAM to cause the next receive operation to be started. On completion of a send request, only buffers on the outbound queue for the same session can be passed to VTAM. Also, receive-completion processing includes precautions against overloading any session's inbound queue.

Whenever a receive operation (other than a response or an asynchronous data flow control) is completed, VEXITRPL increases the corresponding session's count of outstanding inbound requests, ICEINCT, and compares the count against its limit, ICEINLM. If the limit has not been reached, the next receive operation is started. If the limit has been reached, inbound flow to JES2 on this session must be temporarily halted. The RPL is reinitialized as a RESETSR request, indicating that the session be placed in continue-specific mode and is passed to VTAM. (Note that this RPL may still contain data transmitted by the remote sender. When the reset request completes, even if an error occurs, the buffer is placed on the line manager's $RJECHEQ and is processed as a received data buffer.)

Resetting a session to continue-specific mode permits the count of outstanding inbound requests to fall to an acceptable level, because no receive-specific request is ever issued by JES2; thus, JES2 receives no more inbound traffic on the session until the receiving processor has handled the existing inbound requests. The remote sender may still transmit inbound data to VTAM, but VTAM does not pass that data to JES2 until the session is returned to continue-any mode.

HASPCKPT: Checkpoint Processor

HASPCKPT manages access to all information in the JES2 checkpoint records: job queue, job output table, shared queue elements (QSEs), master track group map, remote message spooling queues, and other miscellaneous control information. Other processors use the $QSUSE macro instruction to request update access to these areas. HASPCKPT handles communications among the systems in a multi-access spool configuration, detects errors in the checkpoint data set, and performs recovery procedures for any errors found.

HASPCKPT represents the major portion of code dealing with the JES2 checkpoint data set. The checkpoint PCE executes code in HASPCKPT to ensure that timely access is made to the JES2 checkpoint data set, and to ensure that other members of the multi-access spool complex are given the opportunity to access the job queues.

JES2 can use either one or two checkpoint data sets, in either of two modes of operation.

The first mode of operation, DUPLEX mode, uses the data set(s) as a primary data set with an optional backup, or alternate, data set. In this mode, most activity takes place on the primary data set -- updates made by other systems are read from this data set, updates made by this system are written to this data set. The backup data set is written occasionally to ensure that the second physical copy of the checkpoint data set is relatively up to date. In DUPLEX mode, all queue updates are read and written in 4K blocks, that is, entire 4K pages are read and written, even if only one control block on a given page was changed.

The second mode of operation, DUAL mode, uses both data sets as primary data sets, that is, updates are read and written to both data sets. (Although DUAL mode can be used with only one data set, this configuration is recommended only as a temporary measure when a data set must be taken out of commission, for example, while a DASD volume or control unit undergoes maintenance.) In DUAL mode, the two data sets "flip-flop" control back and forth between them:

- Updates are read from one data set, those updates and any updates made to the job queues by this system are written to the other data set.
- The next system to gain control of the job queues reads from the other data set and writes its updates to the first data set.

In DUAL mode, updates made by the previous system are read from, and updates made by this system are written to, the data set in the change log. The change log is used to compress changed control blocks together. Instead of changes being written in separate 4K pages, the changed control blocks are gathered together and written. In this way, a single 4K page containing, say, 30 control blocks, can be written where 10-20 pages would have had to be written before.

Although DUAL mode uses two primary data sets, ownership of the queues is still given to one system at a time, that is, it is not possible for one system to own CKPT1 while another system owns CKPT2.

Any time HASPCKPT performs a checkpoint I/O, if an I/O error occurs, the checkpoint reconfiguration dialog is called to resolve the problem. Also, each time a write occurs to a checkpoint data set, and each time a Read 1 is executed, the dialog will be called if the operator entered a $TCKPTDEF,RECONFIG=YES command. (See "KDIALOG" documentation in the section that describes HASPCKDS.)

This section describes the following portions of HASPCKPT:

- Checkpoint PCE Initialization
- Main Loop
- Major subroutines
- KREAD1
- KTRK1IO
- KREAD2
- KAFTRD2
- KPRIMW
- KWRITE
- KBLDCHLG
- KBLDCCWS
- KFORMAT
- KTRACE
- KIOERROR
- KBLOB

Checkpoint PCE Initialization

During checkpoint PCE initialization, timer queue elements (TQEs) are initialized for the various interval timers that are used, the I/O copy of the checkpoint is initialized, cross-system $#POST propagation is set up, the page pointer list (PPL) is GETMAINed (used by KBLDCHLG), and the main loop is entered to start this system's write cycle.

When the PCE is initialized, the system has control of the checkpoint data set and an up-to-date copy of the queues is in memory.

Main Loop

The main loop in HASPCKPT:

- Gains control of and obtains the current copy of the checkpoint

  The KREAD1 and KREAD2 routines are called to RESERVE the CKPT1 volume, obtain the software lock, and read into storage any updates made by other systems.

- Processes multi-access spool communication

  The SPOOL processor is $POSTed if there is work for it to do. The QSE control blocks (shared queue elements) are examined to determine if other systems are active or inactive. If another system has joined the complex, the network path manager is notified to add the new system to the network. If a system previously in the complex is no longer active, the network path manager is notified to delete the system from network processing. This code then informs the rest of this system that the job queues are available for processing, propagates $POSTs and $#POSTs from other members of the complex to this system, starts the minimum hold timer and a timer to force a checkpoint write if too much time elapses before a checkpoint write occurs. Finally, it calls KPRIMW to ensure that the down-level data set is brought up to date.

- Makes updates to the data sets as long as this system is allowed. It $WAITs until one of the first three conditions is true and the fourth condition is also true:

  1. A checkpoint write is requested
  2. The timer goes off indicating a write must occur
  3. The timer goes off indicating this system must give up control of the checkpoint data set
  4. All other JES2 PCEs have completed all the work available to them and are $WAITing for events to occur (for example, for SPOOL I/O).

  When these conditions are satisfied, the main loop calls KBLOB to replenish the track groups available to this system, calls KBLDCHLG to build the change log, and calls KWRITE to write all updates to the current data set. After the write is scheduled, any PCEs or other address spaces that are waiting for track groups are notified that spool space is again available. If the minimum hold interval has expired, the code indicates that the queues are no longer available to be updated. Finally, KWRITE is called again to $WAIT for the write to complete. When the write is complete, KPRIMW is called to ensure that the down-level data set is brought up to date, if necessary, and to ensure that the pages which were fixed for the I/O are freed. If the minimum hold interval did not expire, the code returns to wait until conditions are appropriate for another write.

- Releases control of the data sets when the cycle is complete

  When the minimum hold interval has expired, KTRK1IO is called to release the software lock on the up-to-date checkpoint data set, and the CKPT1 volume is RELEASEd. The change log is cleared, the page address lists (PALs) are FREEMAINed, and the page pointer list (PPL) is cleared. If JES2 is shutting down, the checkpoint PCE is shut down.

- Waits to regain control of the queues

  The PCE $WAITs until the minimum amount of time ($MINDORM) has expired before checkpoint processing is again allowed to attempt to regain control of the queues. If no other PCEs require access to the checkpoint, the checkpoint PCE $WAITs until the maximum amount of time has expired before checkpoint processing must attempt to regain access to the queues.

Major Subroutines

KREAD1

This section describes the processing associated with the READ 1 phase of the checkpoint cycle, that is, the initial read of the lock record, the contents of track 1, the change log, and the setting of the lock record. The processing described here is most often invoked during the normal checkpoint cycle by the checkpoint processor.

READ 1 processing initiates two channel programs against two separate data sets in the dual data set configuration, and reads the change log portion from track 1 of the data set. Error processing handles the dual I/O completions and the extended reads. These functions are contained in the KTRK1IO routine and in turn translated into the channel program built for the read of track 1.

The content of the lock record key is defined to allow indication of several states of the data set when operating in dual or duplex modes. The routine processing, data set format, and channel program structure are described in the following sections.

Routine KREAD1: KREAD1 acquires the MVS reserve for the checkpoint data set, invokes KTRK1IO to acquire the checkpoint lock and read the master and change log records, determines the state of the checkpoint lock, and either retries the lock read or returns to its caller. If the lock is obtained, it is validated, and the HFAM (Hasp file allocation map) data checked for forwarded data set(s) or a change in data set status. If the lock is not obtained, KREAD1 informs the operator of the wait and retries the lock read operation once each second. The operator is re-informed of the wait every 32 seconds. If, during the retry processing, an $ESYS,RESET= command is issued, this processing will perform the lock reset operation.

Routine KTRK1IO: The KTRK1IO routine is a general routine which is used to perform I/O operations for the first track of the JES2 checkpoint. KTRK1IO may be used to read or write either of the check and lock records, read the master and change log records, or to test and set the lock record. These operations may be done for one or more records in a single call to KTRK1IO, for example, test and set the lock record, read the check record, and read the master and change log records. The I/O operations may be done to a single checkpoint data set, or to both data sets simultaneously.

The KTRK1IO routine is usually invoked to acquire the checkpoint data set software lock, and, if successful, to read the master record and change log data from the to-be-read-from data set for a dual data set configuration and from the primary data set for a primary/duplex data set configuration. The DASD reserve must be obtained, if required, prior to the invocation of KTRK1IO.

Input to the KTRK1IO routine consists of pointers to the $CKGPAR control blocks (CKGs) for either or both of the checkpoint data set pointers to the comparand values for the lock record key, pointers to the to-be-written values for the lock record key, and a set of processing control flags. The various control flags passed on input are:

Control Flag **Meaning**

READ-CHECK If set, read the check record from the data set(s). Otherwise, do not read the check record.

WRITE-CHECK If set, write the check record from the data set(s). Otherwise, do not write the check record.

If both READ-CHECK and WRITE-CHECK flags are set, then the check record will be read.

Control Flag **Meaning**

TEST-LOCK If set, the lock record (key and data) will be compared to the given lock key comparand value, and, if matching, any further I/O operations indicated will be performed. (If the key value does not match the given comparand, the lock record will be read only.) If this flag is not set, any further I/O operations are performed as indicated.

Note that the following operations are performed conditionally if TEST-LOCK is specified.

Control Flag **Meaning**

READ-LOCK If set, the lock record record will be unconditionally read. Otherwise, the lock record is not read (except if TEST-LOCK is specified); this flag is ignored if TEST-LOCK is specified.

READ-MASTER If set, read the master record (record 3) from the data set(s). If not set, do not read the master record.

READ-LOG If set, read the change log data from the data set(s). If not set, do not read the change log.

WRITE-LOCK If set, write the lock record key value passed as input to the lock record key and data area. If not set, do not write the lock record.

MASTER-SLI If set, suppress length errors reading the master record (record 3) from the dataset(s). If not set, allow length errors to occur.

The routine processes the I/O request by first setting the CKB IOBs and channel program packets to the structure indicated by the input control flags. The required I/O(s) are then initiated via native EXCPVR requests using the separate DCBs and DEBs OPENed for the data sets. The I/O for the data set referenced by the first CKG in the input parameter list is initiated first.

ECBs for the I/O requests are established in the CKB and are directly waited on ($WAIT XECB) after a timer is started. Upon completion of the I/O request(s) any I/O errors detected by the abnormal channel end appendage(s) are recognized, and/or the status of the read completion(s) are determined.

The final read completion status and an indication of the data set(s) read is returned to the caller via a return code and status flags.

The possible return codes returned by KTRK1IO are:

Code **Meaning**

RC=0 The read request completed according to the input specified. The in-storage data set contains the data read from the checkpoint data set.

RC=4 I/O error encountered. The type of error is indicated by further status flags (CKBFLAG3). See below.

RC=8 The CKG addresses passed were either both zero, or one of the CKBs pointed to was found to be invalid. This error is caused by a bad parameter list pointer and/or storage overlay.

RC=12 An I/O operation was found to be already outstanding on a checkpoint data set.

The possible I/O errors (RC=4) detected by KTRK1IO are:

Status **Meaning**

CHECK-I/O-ERROR An I/O error was encountered reading/writing the check record for the data set. No data was read/written.

KEY-SEARCH-ERROR An I/O error was encountered searching for the lock record key of the data set. The key was not found on track 1. Any previous operations were successful.

MASTER-READ-ERROR An I/O error was encountered reading the master record for the data set. No data could be read. Any previous operations were successful.

LOG-READ-ERROR An I/O error was encountered reading the change log for the data set. Some change log records may have been read. (A count of the number of change log read is set in the CKB.) The check and master records were read successfully.

LOCK-READ-ERROR An I/O error was encountered reading the lock record for the data set. No data could be read. Any previous operations were successful.

LOCK-WRITE-ERROR An I/O error was encountered writing the lock record for the data set. No data was written. Any previous operations were successful.

Track 1 and change log format: Three distinct types of records occupy the first track of the data set (track 1).

The check record is contains the data set name and volume serial of the active data set. (It may refer to the data set containing the check record or to a "forwarded" data set in the case of a checkpoint data set reconfiguration.)

The lock record key contains an indication of several states associated with the configuration of the checkpoint data set in dual or duplex mode of operation. The lock record key is also used as a means of preventing data set access in the case of a DASD reserve failure, and as a means of determining whether or not to read the entire contents of track 1.

The lock record data area is the same size as the check record, so that both records can be written in one locate record domain (in ECKD mode).

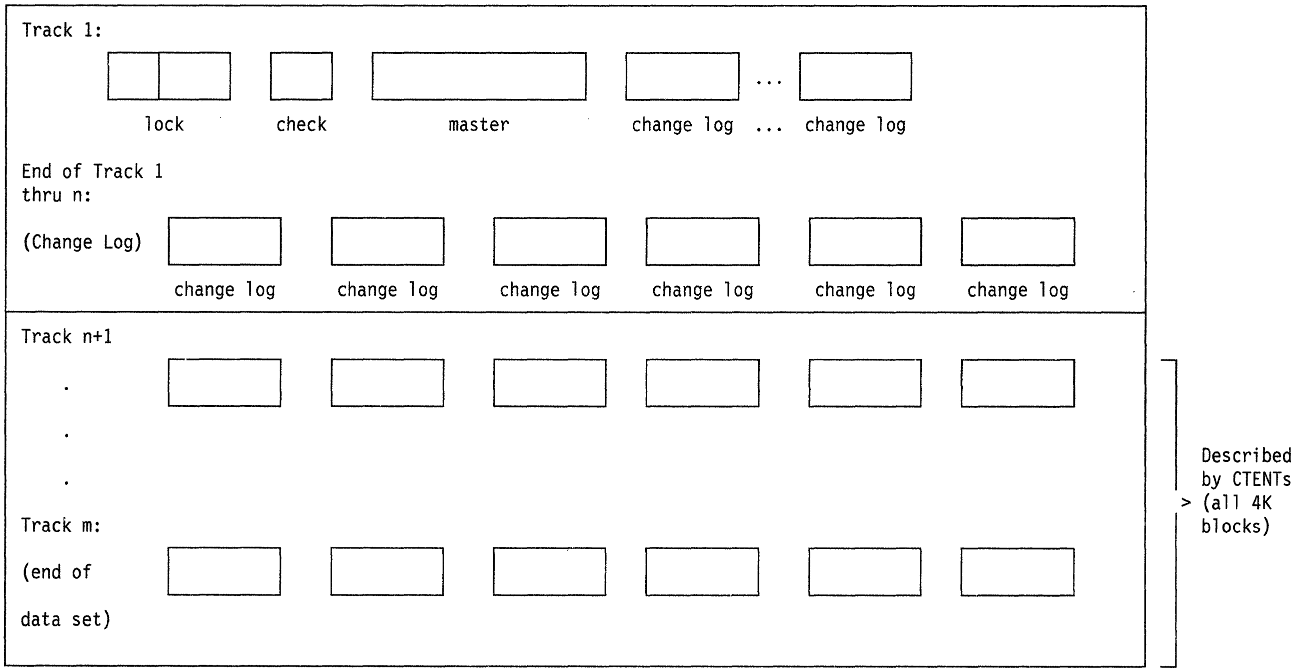
New records which contain the change log data are formatted onto track 1 after the existing master record.

Figure 3-10 depicts the records formatted onto the first track of a checkpoint data set in the order that they exist on the data set.

```
Lock    | 00000000 0000001v |     | 00000000 0000001v SYSID |
         Key                       Data

         lv = 00 - indicates that this data set is to-be-read-from
                   (or primary).
         lv = 01 - indicates that this data set is to-be-written-to
                   (or duplex).
         lv = 02 - indicates that the data set has been reallocated to the
                   dsn and volume named in the Check record.

Check   | dsn and volser (in HFAM format) | level | ck |

         level = A four-byte binary integer indicating the current level of
         the checkpoint 4K page data area.

         ck = 1-7F - incremental write verification sequence no.

         ck = 81 - indicates format write done.

Master  | HCT checkpointed data ...  (mstr ck)  (CTLBs) (etc.) |

Log     | Change log data  (4K length)                          |

          As many log records as will fit on track 1 are formatted.
```

Figure 3-10. Format of Checkpoint Data Set Track 1 Data Records

The format of the checkpoint data set is shown in Figure 3-11. (Also refer to the debugging section in Chapter 5 for a figure that shows the format in more detail.)



Note Here, a single block is an unkeyed data record, a double block is a key record.

Figure 3-11. Checkpoint Data Set Format

Channel Program Description: The channel program constructed to read the track 1 data is built according to the input control flags passed to the KTRK1IO routine. In the simpler cases, the channel program will be a sequence of chained single record reads/write CCW operations. For the TEST-LOCK operation, however, the channel program constructed is more complex.

Basically, the TEST-LOCK channel program acts much like the 370 COMPARE AND SWAP CPU instruction. If the value of the lock key matches the comparand value given on input, the check, master and log records are read and the lock record written (set) to indicate ownership of the data set by the reading system, otherwise ONLY the check and lock records are read (and the lock is NOT altered). The master/log records are read only if indicated by the input parameters passed to the routine.

The channel program is designed to perform the read of track 1 with minimum I/O service time. The channel commands are chained to test the lock, read the check, master, and log records in sequence, and write the lock, in a single rotation of the DASD device if possible.

The channel program itself is built from several common CCW "packets" (a set of CCWs command chained together that perform a single I/O operation - read, write, search key - for a single record of the data set). The CCW packets are formatted depending on the type of DASD device on which the data set resides. If the device supports CKD I/O commands only, the common CCW packets consist of the standard SEEK, SET SECTOR, and SRCH ID EQ CCWs preceding the operation CCW. If the device supports the Extended-CKD command set, then the LOCATE RECORD command is chained to operation CCW and its parameter list set to perform the SEEK, SET SECTOR, etc. orientation. Note that whole CCW packets are not always chained together beginning with the first CCW in the packet; any CCW within the packet may be the target of a TIC command from another packet.

More precisely, the following standard CCW packets are formatted for the single record operations:

Standard Single Record Read/Write Packet (SRW):

1. IDAW List
2. SEEK (CKD) or LOCATE RECORD (ECKD)
3. TIC *+8 (i.e., NOP if CKD) or TIC CCW7 (ECKD)
4. SET SECTOR
5. SRCH ID EQ
6. TIC *-8
7. READ or WRITE + DATA or KEY+DATA
8. TIC (next packet)

Standard Keyed Record Search Packet (KRS): (used only for lock record)

1. (Unused)
2. SET SECTOR or NOP
3. TIC *+8 (i.e., NOP)
4. SRCH ID EQ R2
5. TIC *-8
6. SRCH KEY EQ (argument = XL8'0')
7. TIC (if KEY .NE. arg)
8. TIC (if KEY .EQ. arg)

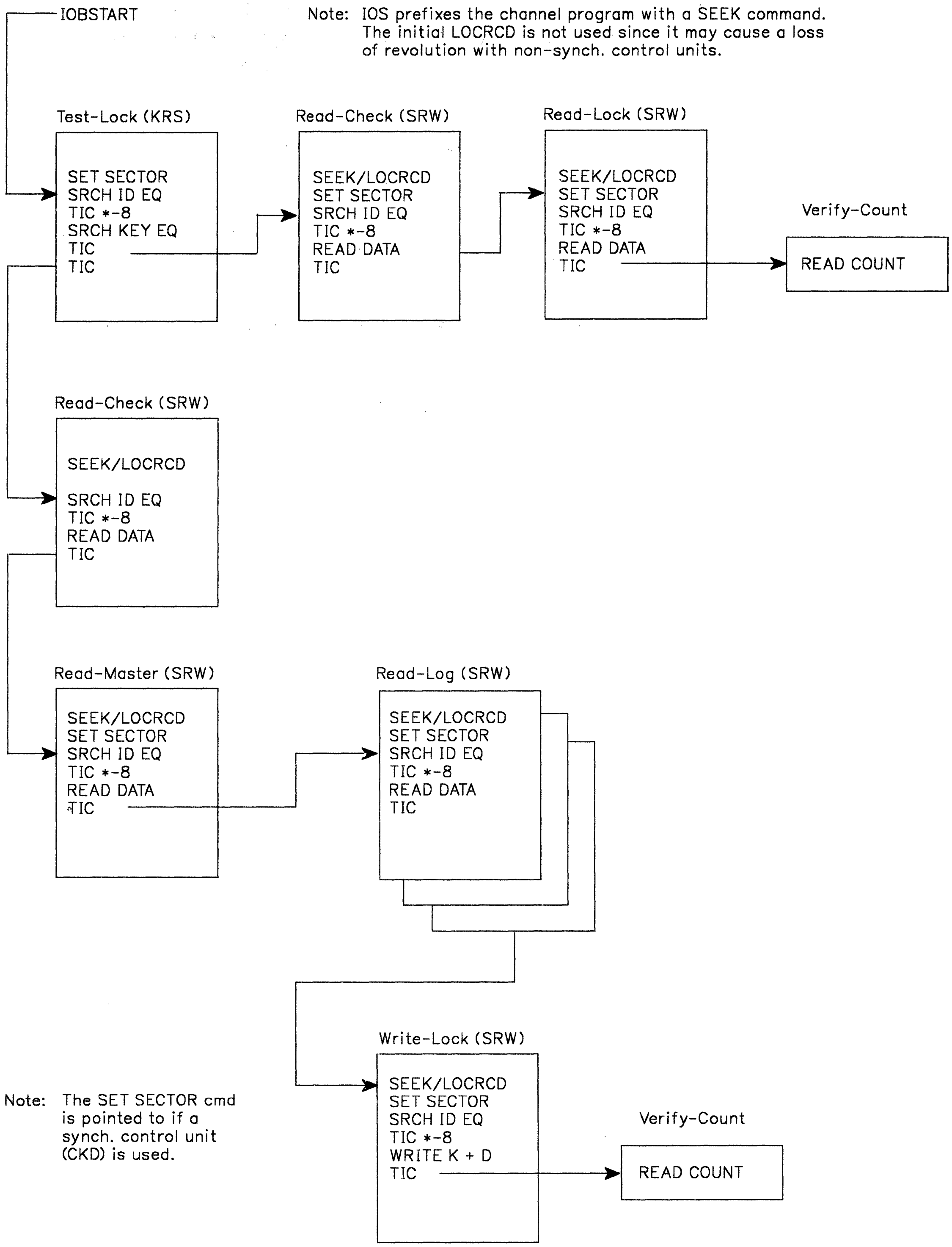
Standard Single Record Format Write Packet (SFW):

1. IDAW List
2. SEEK (CKD) or LOCATE RECORD (ECKD)
3. TIC *+8 (i.e., NOP if CKD) or TIC CCW7 (ECKD)
4. SRCH ID EQ
5. TIC *-8
6. WRITE COUNT+(KEY)+DATA
7. WRITE + DATA
8. TIC (next packet)

The channel program to read track 1 via the TEST-LOCK operation is depicted in the following figure. It is constructed by linking (via the TIC commands) several of the standard CCW packets defined above.

Note: The Read 1 channel program is considered a 'write' operation by a buffered DASD controller because of the final Write-Lock packet; therefore does not take advantage of buffered data for its read operations.

Note: The channel program as depicted below will require, on the average, .5+1+c number of disk revolutions to complete (where c is the partial revolution required to rewrite the lock record).



Note: IOS prefixes the channel program with a SEEK command. The initial LOCRCD is not used since it may cause a loss of revolution with non-synch. control units.

Note: The SET SECTOR cmd is pointed to if a synch. control unit (CKD) is used.

Figure 3-12. Channel Program for Read 1

The channel program as depicted in Figure 3-12 will complete the read of the entire contents of track 1 and write the lock record if, and only if, the lock record key value is all zeros. Otherwise, only the check record data area and the lock record key and data areas are read.

The lock record key value as defined in Figure 3-10 is written with three possible values, each value a combination of settings of the "lv" byte within the key. The four possible values and their meanings are defined as follows:

LV **Meaning**

00-00 Lock unheld, data set is to-be-read-from (or primary if CKPT1).

00-01 Depends on lock value of other data set:

If 00-00, lock unheld, data set is to-be-written-to (or duplex if CKPT2).
If 00-01, lock just obtained, data set status must be determined from level numbers.

00-02 Data set is forwarded. Forwarded data set must be read to determine lock and level status.

The following figures show the channel programs for the Intermediate, Final and Primary Writes, and for Read 2.

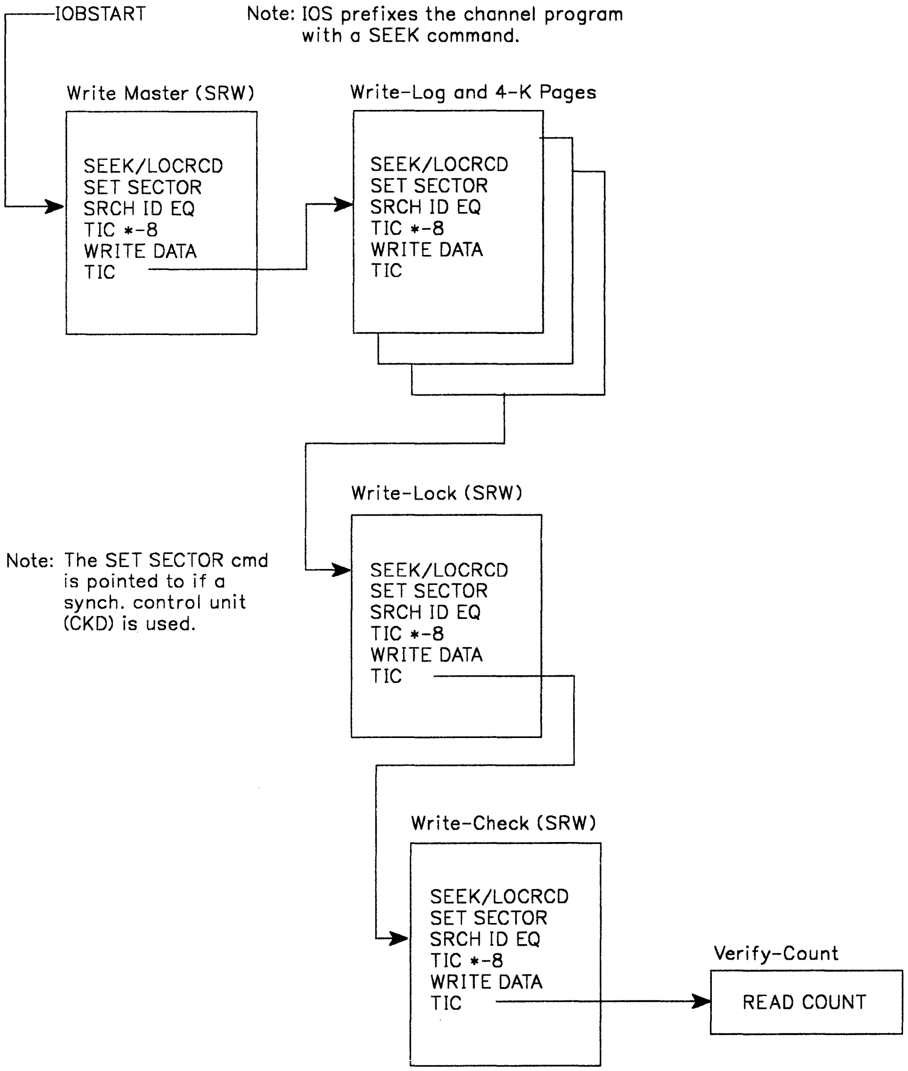


Figure 3-13. Channel Program for Intermediate, Final, and Primary Writes

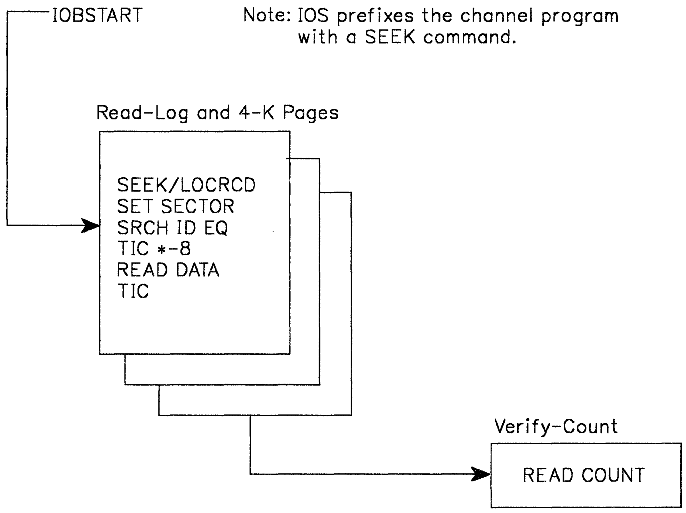


Figure 3-14. Channel Program for Read 2

CCW Packet Build Processing

Several routines process the building and setting of the CCW packets necessary to perform the various required I/O operations to the checkpoint data set. The required I/O operations on the checkpoint data set include read, write, and format-write.

Separate CKBs containing only the Read 1 channel program CCW packets and all data areas are formatted, one for each checkpoint data set. A single set of CCW packets for the regular 4K records (and all track 1 and change log records) are built and formatted separately from the CKBs for the read 1 CCWs.

In addition, the CKB contains storage for an ECB. The ECB is used to monitor checkpoint I/O completions and is separate from the normal $HASPECB used for checkpoint (and spool) I/O. This separation of the checkpoint I/O related ECBs promotes improvement in the checkpoint processor's response to checkpoint I/O completions.

Routine KBLDCKB: The KBLDCKB routine is responsible for acquiring (GETMAINing), building (initial formatting), and page-fixing the CKB. It is invoked primarily from JES2 initialization, but may also be invoked from error recovery support when a new checkpoint data set is dynamically allocated. The CKB is anchored to the CKW.

The size of the CKB depends on the size of the change log defined and the number of 4K records contained in the checkpoint data (this number depends on the sizes of the various JES2 checkpointed control block tables, the job queue (JQE), job output queue (JOT), etc.).

The caller of KBLDCKB is responsible for ensuring that all I/O requests using the CKB are complete prior to the call.

The routine initializes the CCW packets and the associated data packets in the checkpoint buffer (CKB). These packets are used for all I/O operations on both the primary and alternate checkpoint data sets. Each CCW packet is used for I/O operations on a specific record of either checkpoint data set. Each CCW packet is initialized as shown in Figure 3-15.

```
CCW   SEEK,*-*,CC,6
CCW   LOCRCD,*-*,CC,16
CCW   TIC,*-*,0,0
CCW   SRCH+ID+EQ,*-*,CC,5
CCW   TIC,*-8,0,0
CCW   TIC,*-*,DC,8
CCW   READ+DATA,*-*,CC,4096
CCW   TIC,*-*,0,0
```

Figure 3-15. Standard Format-Write CCW Packet

The type of the first and second CCWs in each packet is dependent on the device type of the alternate and primary checkpoint data sets, respectively. For count key data architecture (CKD) devices, a SEEK CCW is used. For extended CKD devices, a LOCATE RECORD CCW is used. The third CCW is a TIC either to the SEARCH ID CCW for CKD devices or to the WRITE CCW for extended CKD devices. The packets are initialized for either format-write or read operations during initialization. For format-write operations, the sixth CCW becomes a WRITE COUNT-KEY-DATA CCW that writes the count portion of the record and is data-chained to the next CCW that writes the data portion of the record. Later, the checkpoint processor (in module HASPCKPT) alters the packets for non-format I/O operations.

Associated with each CCW packet is a data packet that contains the parameter lists for LOCATE RECORD, SEEK, SET SECTOR, and SEARCH ID. It contains the addresses (in CCHHR format) of the record on both the primary and alternate checkpoint data sets. Where CC is the cylinder ID, HH is the head ID, and R is the record.

KBLDCKB uses special CCW packets (CKBLOCKV, CKBLOCKR, CKBLOCK, CKBFMT, CKBCHECK, and CKBVERP) in the CKB for locking, formatting, and I/O verification. These special packets are initialized along with CCW packets used for reading and writing the master checkpoint record and all of the 4K checkpoint records. The special CCW packets are initially set up for both read and format-write operations. They are later altered by the checkpoint processor for other I/O operations.

Routine KSETTRK1: The KSETTRK1 routine is used to set the various elements of the CKB to proper values prior to the initiation of an the Read 1 I/O operation.

Both IOBs for the two checkpoint data sets are set, and the CCW packets for the Read 1 channel programs are set to required values, depending on the type of Read 1 to be performed. (See "KREAD1" for the various types of READ 1s that can be requested.)

Routine KBLD4KP: The KBLD4KP routine is used to initialize the CCW packets for the 4K records prior to the initiation of a checkpoint data set.

I/O Appendages: I/O errors are detected after each main CCW packet, that is, after the reads of the check, lock, master, and change log records. Several attempts will be made to retry the I/O operation for any I/O error detected (beyond the retry attempts made by IOS).

If, after retrying, the I/O remains unsuccessful, the packet in error is recorded in the CKB and the I/O posted complete. For write errors, a determination is made (if possible) as to whether any data was actually transferred to the device. If no data was transferred, the error will be treated as a soft error for purposes of recovery. That is, it will be recognized that the data set may still be read after such an error.

KREAD2

The KREAD2 routine:

- Reads all changed 4-K records into the I/O area, including changed change log records which do not reside on track 1
- Checks the integrity of checkpoint
- Moves all newly read job queue records to the actual queue area and updates them from the change log (using KAFTRD2)

The KREAD2 routine invokes KSETPAKS to ensure that the CCW packets are set to the correct dataset, then calls KBLDCCWS to build a channel program which will read in the changed 4-K records.

The Read 2 I/O is then started (via EXCPVR). If an I/O error occurs, the I/O is retried a number of times. If the I/O completes normally, KIOVERFY is called to verify that the I/O ran to completion. If the checkpoint reconfiguration dialog was called because of an I/O error and control returns to this routine, it must be the case that the dialog lost control of the dataset. In this case, a special return code is returned to the caller (the main loop, generally), which must obtain control of the dataset again from the beginning.

When the Read 2 I/O completes successfully, various checks are made to ensure that the checkpoint dataset is still intact.

Finally, KAFTRD2 is called to apply the change log which was read in to the 4K pages, and to page-free or release those pages which were read during Read 2 and which will not be written in the primary write.

KAFTRD2

The KAFTRD2 subroutine:

- Validates and applies the change log to the 4K pages
- Copies the 4-K pages which were read in during Read 2 from the I/O area to the actual 4-K area
- Page-frees or page-releases the pages that were read during Read 2 and which will not be written in the primary write.

To make the update from the change log, the control block must be copied to the 4-K pages. At this stage of processing there are two sets of 4-K pages which may have valid data, namely the "actual" area and the I/O area. Note that it is inappropriate to first move the 4-K pages from the I/O area to the "actual" area and then to apply the updates from the change log, because the I/O pages that were read in will be used to perform the primary write.

In order to reduce paging, when performing the updates, the only piece of the control block to be moved to the I/O copy of the dataset will be that piece contained in a page that has been read in (and is thus page-fixed). Similarly, the only piece of the control block to be moved to the actual copy of the dataset will be that piece that is not moved to the I/O copy.

Later, the I/O copies of the read-in pages will be copied into the actual copy; any control blocks which were copied from the change log into the I/O copy will take their place in the actual copy at that time.

There are thus four cases to consider:

1. Control block entirely within a page that was read in read 2
2. Control block has front part in a page that was read in read 2, but not back part
3. Control block has back part in a page that was read in read 2, but not front part
4. Control block is not at all within a page that was read in read 2

In case 1, the entire control block will be moved to the I/O area.

In case 2, the front part of the control block will be moved to the I/O area, the back part of the control block will be moved to the actual area.

In case 3, the back part of the control block will be moved to the I/O area, the front part of the control block will be moved to the actual area.

In case 4, the entire control block will be moved to the actual area.

KPRIMW

The KPRIMW routine:

- Handles changes to CKPTDEF,DUPLEX=
- Does a primary write, if required
- Frees all pages that are fixed

If the complex is in DUPLEX mode, the KPRIMW routine ensures that this system's data set allocation matches the allocation desired as specified by the DUPLEX=YES/NO parameter on the CKPTDEF statement.

In other words, if the mode is duplex and the operator requested a change to the duplexing status on this member (that is, requested that there be a change from duplexing on this member to not duplexing, or vice versa) then the following must be done:

- If the change is from duplexing on this member to not duplexing on this member, this routine:

  1. Unallocates CKPT2 (if currently allocated)

  2. Indicates in the QSE that 1 data set is allocated

  3. Skips actual primary write I/O (no reason to do a primary write if only one data set is allocated in DUPLEX mode)

- If the change is from not duplexing on this member to duplexing on this member, this routine :

  1. Ensures that both CKPT1 and CKPT2 are in use
  2. Allocates CKPT2
  3. Initializes the CKB for CKPT2
  4. Indicates in the QSE that 2 datasets are allocated
  5. Continues with the primary write

A primary write is performed if any of the following are true:

- The system is shutting down
- A read 2 was just completed
- An intermediate write was just completed, and the primary write counter has dropped to zero

If a primary write is to be performed, then the KPRIMW routine:

- Frees any pages that are fixed that will not be written in the primary write
- Calls KBLDCHLG to update the change log
- Calls KWRITE to perform the actual primary write

Finally, after the primary write is complete, the KPRIMW routine page-frees any pages in the I/O copy of the checkpoint that are fixed.

KWRITE

The KWRITE routine is responsible for writing all changed change log and 4K records to the checkpoint data set. The KWRITE routine updates the write buffers for the check and lock records, and calls KBLDCCWS to build the channel program that is invoked to perform the I/O. The master record is then copied from the actual area to the I/O area, and the I/O is initiated via EXCPVR.

After the I/O has been initiated, the KWRITE routine processes primary writes differently than intermediate and final writes. When a primary write is performed, this routine waits for the I/O to complete, and returns to the caller. When this routine is entered for the first time to perform an intermediate or final write, it initiates the I/O and immediately returns to the caller. When re-entered for an intermediate or final write, it waits for completion of the I/O and then proceeds as with primary writes.

After the write is complete, the KWRITE routine $POSTs any PCEs waiting on the CKPT resource. The KIOVERFY routine is then called to ensure that the I/O ran to completion, and control returns to the caller.

If an I/O error occurs during the write, the KFORMAT routine is called. If the KFORMAT routine indicates that the format write completed successfully, control proceeds as if the original write succeeded.

KBLDCHLG

The KBLDCHLG routine builds the change log and the $PPL/$PAL structure based on information found in the $CALs. The $CALs are used to identify which control blocks must be written to the checkpoint dataset in the next write. The $PPL/$PAL structure is used to quickly find a particular control block in the change log. They are used to eliminate duplicate entries for individual writes, and, in case of change log overflow, they are used to ensure control block integrity. The KBLDCHLG routine uses the following logic:

If the complex is in DUPLEX mode, the $CKPT service routine should not have produced any CALEs. If there are any CALEs, KBLDCHLG will abend with a $K21 error code. Otherwise, KBLDCHLG returns to the caller.

If the complex is in DUAL mode, all of the updates to the checkpoint are in the $CALs unless the $CKPT routine ran out of storage for new $CALs. If this is the case, KBLDCHLG refreshes any existing change log entries if the page the control block came from was updated. In other words, if a page was updated (its CTLB was marked by $CKPT) and if there are entries in the change log for that page (its PPL entry is not zero), then each change log entry for that page is refreshed.

For each entry in the CALs, KBLDCHLG does the following:

- Validates the CALE (abend with a $K21 if invalid)
- Marks the extra control bytes (CTLBX) for the page(s) corresponding to this control block
- Adds an entry to the change log for this control block
- Updates the PPL/PAL structure to indicate the control block is in the change log.

The following conditions will alter this sequence of events for a given control block:

1. If this control block is already in the portion of the change log that will be written out during the next write, then the existing control block in the change log is refreshed by the control block from the actual copy of the checkpoint, and a new change log entry is not created.
2. If there is no more room in the change log, the following steps are taken:
   - The CTLBs for the pages this control block resides on will be marked with X'FF' indicating that they need to be written
   - Each control block in this control block's PALE chain will have its corresponding CTLBs marked with x'FF'
   - Each control block in the change log in this control block's PALE chain will be marked to be ignored by the next system
   - The PPL entry for this control block's page will be set to zero, indicating that this page has no valid entries in the change log.

Notes:

1. The excess $CALs are $FREMAINed by KBLDCHLG; the $PALs are $GETMAINed by KBLDCHLG as they are needed, and are $FREMAINed in the main loop after the checkpoint reserve is released upon completion of the final write.
2. All control block copies are made by calling the KBCMVBYX routine.

KBCMVBYX

The KBCMVBYX subroutine copies a control block of variable size by executing an MVC instruction in a loop.

KBLDCCWS

The KBLDCCWS routine examines the CTLBs and CLCBs, and builds a channel program which will read/write each page whose control byte indicates that an I/O is required for that page. As the control bytes are examined and the channel program is built, a PSL is built containing an entry for each page to be written/read.

The PSL is then used to page-fix the I/O area and to copy the actual 4-K pages to the I/O area in preparation for the I/O. The records to be read or written are determined by:

- If Read 2, the $SIDAFF bit is looked for in the I/O area control bytes

- If intermediate or final write, the $SIDAFF bit is looked for in the actual control bytes
- If this is the primary write, the high-bit (X'80') is looked for in the actual control bytes.

All CCW chains terminate with the I/O verification packet. In addition, the lock-record write packet, and the check-record write packet precede the verification packet for intermediate, final and primary write chains.

Note: Before KBLDCCWS is called, KSETPAKS must be invoked to ensure that the CCW packets used to perform the I/O point to the correct data areas.

The Control Bytes (CTLBs)

The CTLB is a data area containing one byte for each page in the checkpoint data set. The bits within those bytes have the following meaning:

- Bit X'80':

  This bit is used to control what pages are written during primary writes. This bit is set to 1 at the following times (both on this system and others):

  - (DUAL mode):

    The page was updated by an entry in the change log during this system's KAFTRD2 routine

    The page was updated by an entry in the change log during the previous system's KAFTRD2 routine

    The page was marked by some system because the change log overflowed.

  - (DUPLEX mode):

    The page was marked by some system because that system updated a control block within the page.

  If this bit is ever set to 1, the bits in this byte will be reset in the following manner:

  - Case 1: The page was updated by an entry in the change log during this system's KAFTRD2 routine.

    - The KPRIMW routine on this system will call KBLDCCWS (which will reset the $SIDAFF bit and the affinity bit for the last system that wrote the checkpoint) and will write the page to the appropriate data set. (For example, if the byte was X'FF', this $SIDAFF bit is X'02', and the previous system's affinity bit is X'01', the byte will now be X'FC'.) In this way, if the next system to get control of the checkpoint is the system that last wrote it, that system won't read in any pages that were updated by its change log.
    - The KPRIMW routine on the next system will call KBLDCCWS (which will reset the primary write bit - X'80'), and write the page to the appropriate data set. (For example, if the byte was X'FC', the byte will now be X'7C'.)

  - Case 2: The page was updated due to a change log overflow or (if in DUPLEX mode) a normal control block update. In both cases, the page was updated on the system that previously held the checkpoint.

  - The KPRIMW routine on this system will call KBLDCCWS (which will reset the primary write bit - X'80'), and write the page to the appropriate data set. (For example, if the byte was X'FC', the byte will now be X'7C'.)

- The X'7F' bits:

  These bits control the reading and writing of pages on the various members of the multi-access spool complex. (JES2 supports up to 7 members, among other reasons, because there are 7 bits.)

  These bits are set to 1 for the following reasons:

  - (DUAL mode):

    The page was updated by an entry in the change log during some system's KAFTRD2 routine

    The page was marked by some system because the change log overflowed.

  - (DUPLEX mode):

    The page was marked by some system because that system updated a control block within the page.

  If these bits are ever set to 1, the bits in this byte will be reset in the following manner:

  - Case 1: The page was updated by this system and this system is doing an intermediate/final write

    - The KWRITE routine on this system will call KBLDCCWS (which will reset the $SIDAFF bit only), and write the page to the appropriate data set. (For example, if the byte was X'FF', and the $SIDAFF bit is X'02', the byte will be X'FD'.)

  - Case 2: The page was updated by another system and this system is doing a Read 2.

    - The KREAD2 routine on this system will call KBLDCCWS (which will reset this system's $SIDAFF bit), and read the page from the appropriate data set. (For example, if the byte was X'6E', and that system's $SIDAFF bit is X'02', the byte will now be X'6C'.)

KCCWADJ

The KCCWADJ subroutine takes two standard 4-K read/write CCW packets and chains them together. The routine takes into account such things as a change in track, record addresses, etc., when chaining one packet to the other.

KSETPAKS

The KSETPAKS routine ensures that the CKC CCW packets are set to the dataset whose CKB address is specified. If $CKBCRNT does not match the caller's CKB, it is assumed that the CKC is not set to the dataset requested, and the CCW packets are adjusted to:

1. Reset the seek and search addresses
2. Reset the packets for ECKD vs CKD devices
3. Reset the data packet addresses

KFORMAT

The KFORMAT routine formats (or re-formats) a specified checkpoint data set. The format operation is accomplished by modifying the CCW packets in the $CKC area to a "format-write" sequence (using write count+key+data CCWs), issuing the I/O, then resetting the packets back to the standard read/write sequence. The data set to be formatted is represented by the CKG passed as a parameter to KFORMAT by the caller.

KTRACE

The KTRACE routine, if the checkpoint performance trace id is currently active, is responsible for filling in appropriate fields in the checkpoint trace work area, based on the type of I/O just done, and issuing the $TRACE macro.

Several other routines in HASPCKPT gather checkpoint performance data during the checkpoint cycle. For example, at the time of each I/O, the time the I/O is initiated and the time the I/O completes is saved. Also, during each cycle, the time we know we can allow queue updates and the time we must disallow queue updates are saved. The KBLDCCWS, KAFTRD2, and KBLDCHLG routines all count and save such information as the number of pages transferred in each section of the checkpoint, and the number of control blocks in the change log which belong to each section of the checkpoint.

KIOERROR

The KIOERROR routine analyzes the I/O error detected, indicates the type of error in the CKBFLAG3 field, and reports the error to the operator by issuing the HASP291 message.

KBLOB

The BLOB is a CSA-resident data area containing a table of track group block entries (TGBEs) representing spool space. When a JES2 processor (PCE) or another address space needs spool space, the BLOB is examined, and a track group represented by a TGBE is allocated to the PCE (allocated by the $TRACK routine in HASPTRAK), or address space (allocated by the $STRAK routine in HASCSRIC) requesting space.

Before each intermediate or final checkpoint write, routine KBLOB is called to replenish the BLOB. When replenishing the BLOB, the KBLOB routine checks the TGBEs so that only empty entries will be refilled. Those entries that were not removed by $STRAK or $TRACK will remain in the BLOB.

The allocation is made in such a way that a "round-robin" effect will exist; that is, TGBEs in the BLOB are allocated from all volumes that have free space. In addition, all volumes must be allocated from equally. To do this, the CCTSRCH field in the $HCCT is used. The CCTSRCH field in the $HCCT contains the address of the TGBE that $STRAK and $TRACK use to begin their search for a TGB. Using compare-and-swap logic, these routines then update this field to point to the next entry.

This puts a restriction on the size of the BLOB, the way the BLOB is filled, and on the way TGBEs are removed from the BLOB. Due to this restriction, the size of the BLOB is forced to always be at least as big as the number of allocated spool volumes. In addition, the BLOB will be partitioned. That is, there are a certain number of entries per volume in the BLOB and specific entries belong to specific volumes. Each volume has the same number of entries as every other volume.

The BLOB is partitioned in such a way that each entry is associated with a particular spool volume. Each volume has $NUMTGBE (set from SPOOLDEF TGBPERVL) entries in the BLOB. The nth volume will use every $NUMTGBE entry starting from entry n. This is done (rather than grouping track groups for a given volume) to spread the allocated track groups equally across all volumes.

Before the major loop in KBLOB is entered, checks are done to ensure that all the spool volumes in the BLOB still have space, the size of the BLOB hasn't changed, no volumes were added or deleted, and no volumes just became free after being in a full state. If all those checks are OK, then the main loop is entered. If any of those checks fail, then what is represented in the BLOB is not totally correct and some adjustments have to be made.

There are two spool masks and two counts of spool volumes that are maintained by KBLOB processing. The CKPBLMSK is a mask of all spool volumes which have space in the BLOB. The CCTMTSPL mask is a mask of all spool volumes that have space in the BLOB and/or in the track group map. CKPCNT is a count of spools that are in the BLOB and $SPLCNT is a count of spools with space.

When a volume becomes full, KBLOB turns off that volume's bit in the CCTMTSPL mask and decrements the $SPLCNT field. When a volume that has been full gets back some track groups, KBLOB turns on that volume's bit in the CCTMTSPL mask and increments the $SPLCNT field. When a volume is halted or deleted, the SPOL PCE turns off the bit in the CCTMTSPL mask and decrement $SPLCNT. When a volume is added, KBLOB turns on the CCTMTSPL bit and increments $SPLCNT.

The size of the BLOB can be changed using an operator command, but it will always be a multiple of the number of allocated spool volumes because the command specifies entries per volume.

HASPCKDS: Checkpoint Support Routines

Application Program Support

HASPCKAP Subtask

HASPCKDS contains the HASPCKAP subtask. The HASPCKAP subtask (attached by IRMVS in HASPIRMA with a $DTEDYNA) optionally gets storage for and maintains a third copy of the checkpoint so that authorized application programs (such as SDSF) can use it.

When attached, HASPCKAP determines if a $KAC is chained off the $SVT. (A $KAC is an extended-CSA control block that describes the copy of the checkpoint.) If not, storage for one is obtained and it is initialized. The subtask then gets storage for the copy of the checkpoint data (master record and 4-K pages). A page service list (PSL) is also GETMAINed. The application copy's bits in the control bytes ($CTLBXs) are set to zero (see below), the initial move of the checkpoint data is done, and $DTEDYN is posted. The subtask then waits for work.

The application copy of the checkpoint is updated under two conditions:

- When JES2 releases the checkpoint data set lock. This is based on the MASDEF HOLD and DORMANCY parameters.
- During intermediate writes, but not more than approximately once per second. (An intermediate write is scheduled when a checkpointed control block is altered by an operator command or when a JQE is altered by the execution processor.) A $STIMER interval schedules an intermediate write after a maximum of 5 seconds.

When posted, the subtask checks to see if the application wants JES2 to update the copy (the application may have specified a time later than the current time). If the update is not wanted, the subtask posts the JES2 main task and waits. If an update has been requested, the subtask indicates that an update is in progress and zeros the wait time field. It updates the level of the copy and saves the current checkpoint level number. The master record is then updated. A PSL is constructed to release any pages that were updated by the main task. If no pages were updated, then no pages are released and none is copied. All the pages in the copy that need to be updated are released and the updated information is copied in.

When all the updates are made, the update-in-process indicator is turned off. If the DEBUG function is on, a CLCL is issued to ensure the new copy matches the real 4-K pages. If the two do not match, an MVS ABEND (068, reason code, 100) is issued. Finally, the application copy's bits in the control bytes are cleared, and the JES2 main task is posted.

Standard ESTAE support is used for recovery. If there is an abend while the subtask is initializing, the recovery routine turns on a failure-to-initialize indicator and posts $DTEDYN. If the abend is during normal processing, all of the application copy's bits in the the control bytes are set to 1, and the bad-copy-indicator is turned on. The next time the subtask is posted, it will attempt to recopy the entire checkpoint. The bad-copy-indicator is turned off, if the recopy is successful.

$CTLBXs

The $CTLBXs are a set of control bytes used by the HASPCKPT PCE to support trace id 17. They are also used to support the HASPCKAP subtask to keep track of the pages changed since the last time the copy was updated. If the page is not changed since the last update to the copy, they will be zero. If the last change was because of a HASPCKAP problem, the application copy's bits will be set to 1. All other ways of changing the page will have values equated in $HASPEQU, as follows:

- In $CKPT (in HASPNUC), if a 4-K page is being marked as changed (this is DUPLEX mode or a CAL was not available), the equate is CKPCLCKP.
- In KAFTERIO (in HASPCKPT), when the I/O CTLBs are marked as the change log is applied after a read, the equate is CKPCLRDC. Before a 4-K page is copied from the I/O area to the actual 4-K pages after a read, the equate is CKPCLRDP.
- In KBLDCHLG (in HASPCKPT), as a change log entry is built and the pages that contain the control block is marked, the equate is CKPCLBCL.

Determining the Version of JES2

An eight-character version number is also available to authorized applications. To determine the level of the JES2 subsystem, the application finds the SSCVT for the JES2 subsystem. Then it loads the SSCTSUSE field. If this field is zero, this is a pre-2.2.0 system. If this field is non-zero, this is the address of an 8-byte character field containing "SP 2.2.0", referring to a JES2 Version 2, Release 2, system, or containing 8 characters that refer to a subsequent release of JES2.

Other Routines in HASPCKDS

HASPCKDS also contains routines used to initialize a checkpoint data set (dynamic allocation and building of control blocks), suspend the use of a checkpoint data set. It also contains the checkpoint dialog routines, and the KRESERVE and KRELEASE routines.

CKBINIT

This routine sets up the internal control blocks to do I/O to the checkpoint data set. It checks the size of the variable control blocks (that is, the change log) and ensures the size of the in-storage copy matches the size specified in the master record. If the size of the control block has changed, the in-storage copy is $FREMAINed and a new one $GETMAINed. If the $GETMAIN fails, the old size is $GETMAINed and processing continues.

If the original control block did not exist, no $FREMAIN is done. If the $GETMAIN fails for the new control block, control is returned to the caller and no other control blocks are are changed.

A new CKB is built using the routine, KBLDCKB.

CKPTALOC

This routine performs the dynamic allocation of a checkpoint data set. It performs tests to verify that the data set is usable for the checkpoint function. It performs the OPEN for the data set. First, it establishes an ESTAE. If CKPTALOC abends, KALESTAE gets control.

CKPTUNAL

This routine performs dynamic unallocation of a checkpoint data set. It first CLOSEs the data set, then $FREMAINs the DCB, CKB, TOT, and deallocates the data set.

CKPALCLN

This routine cleans up after a failed CKPTALOC call. It $FREMAINs any storage that was obtained, using the checkpoint allocation work area (KAWA) to determine the processing CKPTALOC did.

KDIALOG

This routine attempts to satisfy the requirement for a checkpoint data set reconfiguration. This requirement results from one of three conditions:

- A permanent I/O error has occurred on a checkpoint data set
- JES2 initialization has determined that it needs help establishing the proper checkpoint reconfiguration
- An operator has requested ($T CKPTDEF,RECONFIG=YES) that the checkpoint configuration be changed. An ESTAE environment is established and the code responds to various operator responses to messages. Code at KDLIOERR handles I/O error situations. Code at KDLINITC handles initialization reconfiguration. Code at KDLRECON handles the operator-initiated reconfiguration dialog.

  When control returns after reason-specific processing, all messages that may still be pending are DOMed. $QSONDA is turned off. If there is a residual I/O error pending processing, the dialog is invoked again.

KDIALOG service routines include:

- KDLWTO, which sends messages to the operator using $BLDMSG
- KDLWTOR, which sends a WTOR to the operator using $BLDMSG
- KDLREPLY, which processes operator responses to a WTOR
- KNULLCHK, which determines if the HASP file allocation map entry (HFAME) describes a null file. If so, message HASP282 is issued.
- KDSLOC, which prompts the operator for checkpoint data set specifications if the data set described by the HFAME cannot be found by CKPTALOC.
- KDLESTAE, which handles abends occurring during processing of I/O error recovery or RECONFIG=YES support.

The routine also contains:

- KBLDCKB, which obtains a CKB control block, fixes it in central storage, and builds the CCW packets associated with track one for a particular data set. The CCW packets are formatted and filled in with data that remains constant through checkpoint I/O processing.
- KBLD4KP, which obtains storage for the 4-K record CCW area, page-fixes the storage in central storage (for use by EXCPVR), and builds the CCW packets used for the 4-K record I/O for a particular data set. The CCW packets are formatted and filled in with data that remains constant through checkpoint I/O processing. In KBLD4KP, the packets are built in a one-to-one correspondence with the CCW data packets built in KBLDCKB.

KRESERVE: Reserve the Shared Checkpoint Volume Service Routine

The KRESERVE routine saves the caller's registers and issues the MVS RESERVE macro instruction to reserve the data set whose CKB is in register 1. The MVS RESERVE sets a return code in register 15. If the return code is zero, indicating that the checkpoint is reserved, control is returned to the caller with register 15 set to zero. If the return code is not zero, indicating that the checkpoint is not reserved, the KRESERVE routine returns with register 15 set to 4 and the address of the XECB to $WAIT on in register 1.

KRELEASE: Release the Shared Checkpoint Volume Service Routine

The KRELEASE routine saves the caller's registers, issues the MVS DEQ macro instruction to release the reserve on the data set whose CKB is in register 1, and returns to the caller.

KNOP: Issue a NOP CCW to Get the Hardware Reserve

The KNOP routine saves the caller's registers and determines if the data set whose CKB was passed in register 1 is on a shared device. If not, control is returned to the caller with register 15 set to zero. If the data set is on a shared device, a NOP CCW is issued to the data set. Control is returned to the caller with register 15 set to 4 and the address of the XECB to $WAIT on in register 1.

HASPWARM: Warm Start Processor

A JES2 warm start can be invoked for a variety of reasons. It is performed automatically after the JES2 initialization. It can also be invoked as the result of a $E SYS command. In the first case, one of several environments may exist after JES2 initialization:

- An initial program load (IPL) has been performed, and a JES2 cold start has been requested.
- An IPL has been performed, and a JES2 warm start has been requested.
- JES2 has been restarted, without an intervening IPL, following an orderly shutdown of JES2 via a $P JES2 command.
- JES2 has been restarted, without an intervening IPL, following the abnormal termination of the JES2 subsystem.

For restart following an orderly shutdown, the operator can request only a cold or warm start. If the operator requests a warm start, JES2 determines what kind of warm start to perform according to the state it was in when it went down and how it went down. Following an abnormal termination of JES2 and a subsequent restart with no intervening IPL, JES2 will restart with a hot start; all jobs, started tasks, and TSO logons, which had been executing at the time of the JES2 termination, continue to execute. Unless these jobs require additional JES2 services, such as for spool space after the existing track group block has been exhausted, the termination and restart of JES2 should be transparent to them. At worst, these jobs are forced to wait until the JES2 hot start or all-systems warm start is complete.

In a multi-access spool environment, the $E SYS command can be used at one system to request the recovery of work in another system in the node. Normally, this would only be done when the warm-started system is not in a condition to have an IPL performed. This condition would exist, for example, if the failed system had experienced a hardware error, preventing the CPU from being IPLed.

The logic of the warm start processor is based upon a shared-spool environment. When it performs a cold start, other systems in the node are assumed to be dormant. However, a warm start may have to proceed within a system that is itself operational (that is, as the result of a $E SYS command) or within a node in which one or more systems are operational. Such a warm start is referred to hereafter as a single-system warm start. If a warm start is performed following a $S JES2 command and all other systems in the multi-access spool configuration are dormant, then such a warm start is referred to as a all-systems warm start.

For an all-systems warm start where all spool volumes are online, the warm start processor reconstructs the track group bit map in order to recover track groups lost due to prior system failures. This reconstruction process involves freeing up any unused tracks as follows:

- A temporary track group map is constructed with each existing track group marked free.
- All IOTs are read one by one.
- Any track group indicated as in use by an IOT is marked as allocated in the temporary map.

- After a successful completion of the warm start, if all spool volumes are online, the master track group map is updated from the temporary map.

Lost tracks result from abnormal termination of JES2 at a time when jobs are being read in but before their control blocks have been checkpointed. For all other warm starts, the master track group map is left untouched.

Warm start processing uses multiple PCEs. The main warm start PCE is attached during HASPIRMA initialization. On entry to HASPWARM, this PCE performs the non-I/O bound warm start functions, such as the JOT warm start, first. For JQE processing, the main PCE dynamically attaches a number of other warm start PCEs of the same type, using the $PCEDYN macro (serviced by PCEDYN in HASPDYN). These are called clone PCEs. Both the main PCE and the clone PCEs process JQEs, using a common JQE index to run through the elements sequentially.

After the job queue has been processed, the main warm start PCE waits on the RMWT queue until all the clone PCEs have been detached. After the job queue has been processed, a clone PCE issues the $PCETERM macro. The $PCETERM macro issues a $WAIT to the clone PCE on the PCETM queue and then issues a $POST to the resource manager. The resource manager will then detach and PCEs waiting on the PCETM queue.

When all clones have been detached, the main PCE finishes performing the remaining HASPWARM functions. $HASP492 informs the operator that warm start processing is complete.

Figure 3-16 on page 3-384 shows the processing flow of the major routines in HASPWARM. Text describing the flow follows.

```
ENTRY
 |
ACTIVE
 |
CLONE? YES --> NQJQEWRM
 |
 NO
 |
NQRMTON
 |
COLD OR QUICK START? YES--> NQCLNUP
 |
NQPURUP
 |
HOT START? NO-->NQJOTWRM
 |
NQFSSHS
 |
NQJOTWRM
 |
HOT START? NO-->NQPREJQE
 |
NQPSOQ
 |
NQPSJB
 |
NQPREJQE
 |
NQJQEWRM
 |
CLONE?  YES-->PLACE ON TERMINATE QUEUE
 |
NQSPOF
 |
NQCLNUP
 |
PROCESSOR
CLEANUP
 |
DORMANT
 |
WAIT ESYS
```

Figure 3-16. Processing flow for HASPWARM

The initial entry point of HASPWARM is HA$PWARM. The processor is marked active. It is then determined whether this is the main PCE (the warm start communication area (WCA) does not exist) or a clone PCE (the WCA does exist). The WCA keeps track of the latest JQE being processed and the number of JQEs left to process. It is obtained by $GETWORK just before job queue processing and released by $RETWORK just after job queue processing. The WCA is mapped by $WARMCA. If this is the main PCE, processing continues with normal warm start processing. If this is a clone PCE, control branches to do just job queue processing.

NQRMTON -- Gets control on all starts: cold, quick, all-system, single-system, hot. If remote terminals are generated for the system, the warm start processor resets the remote sign-on table. If a single-system warm start or hot start is being performed, then only those system IDs in the table corresponding to terminals signed on to the warm-started system are reset. Otherwise, the entire sign-on table is reset.

NQPURUP -- Gets control on all-system, single system, and hot starts. The QSEs are searched to check if any jobs failed while purging. See $PURGER.

NQFSSHS -- Gets control on hot starts. For a hot start, the warm start processor acquires storage for an SRB and determines if a functional subsystem address space was created and still exists (functional subsystem address spaces, if active, are waiting for the JES2 main task services or a post). The warm start processor schedules the SRB to those functional address spaces; the SRB reconnects the functional subsystem cross memory environment and resets its authority. If the reconnection process fails, the warm start processor terminates the functional subsystem with a '02C' abend and issues the $HASP413 message. If the reconnection is successful, the warm start processor checks for any FSAs for which no DCTs have been defined and terminates with a '02C' abend those FSAs that have no DCTs defined; the $HASP415 message is issued to indicate this.

NQJOTWRM -- Gets control on all-systems, single system, and hot starts. NQJOTWRM scans the class queues for work JOEs representing active devices. For a single-system warm start or hot start, only JOEs representing activity on the warm-started system are processed. When a appropriate busy work JOE is found, it is tested for local, remote, or network job entry (NJE) transmission activity. If it is local, its active device count in its characteristic JOE is decreased by one. If it is local or remote, the JOE's priority is recomputed, the JOE is marked ready, and a message is issued to the operator that a job was printing/punching. If NJE SYSOUT transmission was in progress, the JOE and any other JOEs that the SYSOUT transmitter might have acquired are marked ready, and a message is issued to the operator that a job was on a SYSOUT transmitter.

If the job was being dumped (HASPWARM checks the device id of every busy JOE on the class queues) a message is issued to the operator that the job was on the SYSOUT transmitter being dumped. Then these jobs are also returned to the JOT. If held data sets were being dumped (indicated by JQEs on the hardcopy queue that are marked busy and have an offset in the LCK), a message is also sent to the operator. HASPWARM then issues a QPUT to put the JQE back on the hardcopy queue, reset the busy bit, and zero the JQE offset in the LCK.

NQPSOQ -- Gets control on a hot start. Any waiting requestors on the CANCEL/STATUS and PSO queues are posted and the request elements removed.

NQPSJB -- Gets control on a hot start. The CCTJPCLS queue is searched for any partially-selected SJBs. If any are found, the SJB partially-selected bit is reset, the token field is cleared, and if the associated JQE is marked busy on this system, the JQE busy field is reset.

NQPREJQE -- Gets control on all-systems, single-system, and hot starts. This subroutine gets the storage for the WCA and initializes it. This subroutine determines the number needed and attaches the clone PCEs. The number is based on the number of available spool volumes and I/O buffers. If an error occurs while trying to attach PCEs, warm start processing will continue with the number of PCEs already attached. There will be at least one.

NQJQEWRM -- Gets control on all-systems, single-system, and hot starts. At this point, the warm start processors are ready to perform a warm start of the JES2 job queue. In general, this consists of removing any job hold queue locks or held data set use locks held by jobs which were active in the system being warm started and of requeuing, where required, jobs for the next or current phase of processing. The WCA is used as a serialization tool in this processing.

The processing of jobs in the JES2 job queue differs according to the type of warm start being performed. The following lists possible considerations of the warm start processor for each job:

1. Reallocate the spool space represented in the job's spin and normal input/output tables (IOTs).
2. Ensure that the JOT is current with respect to the job's spin IOTs.
3. Remove the job's hold data set in-use locks.
4. Ensure that the held data set's queue for the job is current with respect to the spin IOTs and the name IOT.
5. Requeue a job on the reader, job receiver, or SYSOUT receiver for purge processing.
6. Requeue a job on the job transmitter for transmission.
7. Requeue a normal job in JCL conversion for conversion processing.
8. Requeue a started task or logon job that was awaiting execution or awaiting JCL conversion for purge processing.
9. Requeue a job that was terminating execution for output processing.
10. Requeue a job that was executing (but not terminating) for execution processing or for output processing.
11. Release jobs held because of duplicate job name.
12. Reestablish time-exceeded monitoring for a job in execution.
13. Requeue a job that was in output processing for processing, removing any JOEs that were created for the job if its IOT had not been written yet.
14. Requeue a job in purge processing for processing.

For a single-system warm start following an IPL or resulting from a $E SYS command, all of the above considerations apply except for the first one and the last two.

For a JES2 hot start, items 2-7 and the last three considerations apply.

For an all-systems warm start, all considerations apply except for requeuing a job for purge processing.

If any spool volumes for a given job are not online, the job's spool control blocks, JCT, and IOT might not be accessible. In this case, the JCT and IOT are not read and HASPWARM checks to see if all of this job's spool volumes still exist. If they do, HASPWARM marks the job not busy and requeues it. If any spools do not exist, HASPWARM purges the JQE and the tracks are recovered when the track group map is reconstructed.

For an all-systems warm start, all jobs in the JES2 job queue are processed with the above considerations. In general, only those jobs active on the system being warm-started are examined for requeuing in a single-system warm start or hot start. However, items 2, 3, and 4 are considered for all jobs because they do not necessarily apply to jobs owned by one system. For example, PDDB in-use locks may be held by a system being warm-started, although the job is still in execution on another system.

The shared communication queue messages are processed as part of job queue processing. If this is an all-systems warm start and the track group map is not being rebuilt, all messages queued to all QSEs are deleted. If this is a single-system warm start or a $E SYS start, only the warm starting system's queued messages and reset-spools-used mask are deleted.

NQSPOF -- Gets control on all-systems, single-system, and hot starts. For those devices that are active, the DSN and the system id in the spool offload LCK table are cleared. For an all-systems warm start, the entire table is cleared.

NQCLNUP -- Gets control on all starts: cold, quick, all-systems, single-system, and hot. When all jobs in the job queue have been processed or examined, the warm start processor proceeds with its termination processing. If a single-system warm start was performed as the result of a $E SYS command, a type 43 system management facilities (SMF) record can be produced. If a cold start or an all-systems warm start was performed, a calculation is made of the available track groups. HASPWARM queues any bad track groups not specified on the BADTRACK initialization parameter to the HASPSPOL processor (via the $BLDTGB macro) so that recovery can be attempted. HASPWARM then frees the storage (FREEMAIN) for the BADTGS temporary map, and marks any track groups marked as bad in the bad track group map as allocated. If JES2 is being started following an IPL as the primary system, the master scheduler is posted with post code=0 to start up SYSLOG. If JES2 is being started as the primary subsystem and this is not a hot start, the master scheduler is posted with post code=4 to restart SYSLOG. For a cold or quick start, or an all-systems warm start, the checkpoint device reserve count is decreased, allowing other systems in a MAS complex to access the job queue.

HASPMISC: Miscellaneous Services

The following describes the resource manager, the time excess processor, the priority aging processor, the SMF accounting subtask, and the network accounting conversion routines.

HASPRESM: Resource Manager

This processor handles resource threshold management. It gains control every eight seconds to check the state of these JES2 resources: console message buffers (CMBs), logical buffers (LBUFs), teleprocessing buffers (TPBFs), system management facility buffers (SMFBs), job queue elements (JQEs), job output elements (JOEs), spool space track groups (TGs), job numbers (JNUM) and trace tables (TTAB). The percentages are updated every eight seconds. If any shortage is detected, HASPRESM issues the $HASP050 action message. This message is updated every 32 seconds. HASPRESM also detaches PCEs on the PCE termination (PCETM) queue. This queue is checked every 8 seconds or when the resource manager is posted. The other resources are only checked every 8 seconds.

HASPTIME: Time Excess Processor

This processor monitors a user job's real time (clock-on-the-wall) in execution and, when appropriate, issues a message to the system operator indicating that the job has exceeded its estimated real execution time.

When the JES2 execution processor provides a user job to the HASCJBST job select routine, the select routine reads in the job-related subsystem control blocks and performs certain other functions before returning to the MVS initiator, which initiated the job select request. If the job select routine completes its job request processing normally, it stores the complement of the time estimate for the job, which is obtained from the JCT, in the SJBXSTIM field of the job's subsystem job block (SJB) to indicate that the job is beginning execution. The job select routine then posts ($$POST) the JES2 task, which causes the time excess processor to be dispatched.

When dispatched, the time excess processor runs the chain of SJBs representing jobs in execution by class. The SJBSTQE and SJBXSTIM fields are used to monitor a job's real execution time. When SJBXSTIM is a negative binary value, the processor obtains the time estimate by re-complementing the value. This estimate is provided to the $STIMER routine in the SJBSTQE field. If the job is still in execution when the time estimate elapses, the time excess processor is posted ($POST WORK) by the timer processor. Then the time excess processor issues a time-exceeded message to the operator. It then replaces the initial time estimate with the value in $ESTIME. This represents the installation-determined interval between time exceeded messages for a given job. This new value is provided to the $STIMER routine. If the job is still in execution when the message interval elapses, the time excess processor, dispatched following a post by the timer processor, issues another time-exceeded message and executes another $STIMER macro instruction with the same time interval. This sequence is repeated until the job terminates.

HASPGPRC: Priority Aging Processor

The function of the priority aging processor is to regularly increase the priority of a job and output in such a way that its position in the JES2 job queue or output queue is enhanced with the passage of time. This is accomplished by regularly passing through the JES2 job queue and increasing the priority field of all job queue elements whose priorities fall between upper and lower limits. These limits, as well as the time interval, are installation variables. At this same time interval, a job's job output elements are also priority aged, up to a maximum of 255. In a multi-access spooling environment, only one system performs the function of priority aging.

When HASPGPRC is dispatched, it first examines the JES2 job queue to determine if it is empty. If empty, HASPGPRC waits ($WAIT) for a job to be added to the queue before continuing processing.

If the job queue is not empty, the shared queue elements (QSEs) are then scanned for active systems. If the first active system encountered in the QSEs is a different from the system on which this processor is operating, the priority aging process is bypassed to prevent multiple updating of the job queue.

Next, HASPGPRC searches through the JES2 job queue for all jobs. The job queue element (JQE), whose priority field (JQEPRIO) is less than the JES2 initialization parameter PRTYHIGH and greater than PRTYLOW, will be priority aged. HASPGPRC stores the new priority in the JQE and issues a $CKPT to checkpoint the JQE. Any job output elements (JOEs) that are chained off a JQE will be priority aged. The new JOE priority is stored in the JOE and a $CKPT is issued to checkpoint the JOE. The interval timer is then reset, and HASPGPRC enters a $WAIT until the timer interval expires.

Because the priority of the JQE is represented by the 4 leftmost bits of JQEPRIO, adding 1 to this field has no immediate effect on the priority. After this operation is repeated 16 times, however, the actual value of the priority is increased by 1. The value of the time interval is actually 1/16th of the interval implied by the JES2 initialization parameter PRTYRATE. This effect tends to smooth out the process of priority aging by creating less impact when an interval expires. This also applies to the JOEPRID field for output.

HASPACCT: SMF Subtask

The function of the JES SMF writer (HASPACCT) is to check the $SMFBUSY queue, take buffers off the queue, call the IEFUJP exit if necessary, interface with the MVS system management facilities (SMF) writer, and place freed buffers on the $SMFFREE queue. HASPACCT runs as a subtask of the JES2 main task, even though it is a part of the HASJES20 load module. HASPACCT is identified and attached by HASPINIT and detached by HASPNUC if JES2 is stopped. The program is activated by a MVS POST macro when any JES2 main task routine is ready to write a JES2 SMF record.

HASPACCT: Initial Entry Point

HASPACCT performs standard register save area initialization, obtaining its own save area through a GETMAIN macro instruction. Then HASPACCT issues an ESTAE identifying $STABEND as its recovery routine. If an error occurs during HASPACCT processing, ACCESTAE will get control and percolate the error, requesting recording. The address of an event control block (ECB) is passed to HASPACCT in register 1 by the caller. HASPACCT posts that ECB to indicate to the caller that HASPACCT has received control, then waits on its own ECB ($ACCTECB) in the HASP communications table (HCT) to be posted for work by the JES2 main task. Upon again receiving control, HASPACCT enters its main processing loop, in which it continues whenever dispatched until JES2 is terminated through a $P JES2 command.

AFREESMF: Main Processing Loop

At label ARECYCLE, the $SMFBUSY field in the HCT is tested. If that cell is 0, indicating that no SMF buffer is ready to be written, AFREESMF tests the $SYSEXIT indicator in the HCT $STATUS field to determine whether the $P JES2 command has been issued. If so, AFREESMF issues the RETURN macro instruction to return control to the caller with condition code 0.

If $SMFBUSY is non-zero, AFREESMF obtains the address of the first buffer in the chain (the contents of $SMFBUSY) and steps through the chain until the last buffer is reached. Because HASPACCT operates asynchronously with respect to JES2 main task routines, which may alter the buffer chain, AFREESMF uses a compare-and-swap instruction to reset chain pointer $SMFBUSY to 0, indicating that all buffers have been removed from further processing by HASPACCT. If the compare-and-swap fails, one or more buffers have been added to the top of the chain while the chain was being processed. In this case, AFREESMF reenters the main processing loop at ARECYCLE and repeats the chain search.

If the chain was successfully removed from the $SMFBUSY queue, AFREESMF examines the first buffer and, if the buffer contains an SMF record, branches to AWRITSMF immediately. If the buffer contains a copy of the common exit parameter area (the job management record section of a job control table), however, a parameter list is prepared for the IEFUJP user exit. The parameter list, pointed to by register 1 when IEFUJP is called via the SMFEXIT macro, consists of two fullwords: the first word is the address of the common exit parameter area, and the second is the address of the SMF RDW of the type 26 purge record. If upon return from the IEFUJP routine, register 15 contains the value 4, the type 26 record is not written; write processing is bypassed through a branch to AUNQSMF. Otherwise, processing continues with SMF write processing at AWRITSMF.

AWRITSMF: AWRITSMF tests a flag in the SMF buffer to determine whether the record should be suppressed. If the flag is on, writing is bypassed through a branch to AUNQSMF. If the flag is not set to 1, the routine determines if the SMF record is being written on behalf of a JES2 batch job, an STC, a TSU or a non-job related request. After write affinity for the SMF record has been determined, the routine issues the SMFEWTM macro instruction to cause the record to be written.

AUNQSMF: AUNQSMF places the SMF buffer on the $SMFFREE queue after the record has been written, or if writing was bypassed. If there was a job management record (JMR) buffer in addition to the SMF record buffer, it is also freed. Then, if the $SMFFREE queue was empty, the JES2 main task is posted ($$POST TYPE=SMF). After the post, or if the free queue was not empty, AUNQSMF returns to the main processing loop at AFREESMF.

HASPNACT: NETACCT Conversion Routines

Two lookup routines are contained in the HASPMISC module in a single entry point called HASPNACT. The address of the common entry point to both routines is contained in the $NACT cell in the HCT. The calling sequence for both routines is the same; register 0 points to the location that contains (or will contain) the 8-byte network account number, and register 1 points to the location that contains (or will contain) the JES2 account number. The presence or absence of each of these parameters determines which lookup routine (JES2-to-net or net-to-JES2) in invoked. If the network account number is present (first byte is neither blank nor X'00'), the net-to-JES2 routine is invoked; if a match is found for the network account number, the corresponding JES2 account number is placed in the location pointed to by register 1, overriding any existing JES2 account number. If the network account number is not present but the JES2 account number is, the JES2-to-net routine is invoked; if a match is found for the JES2 account number, the corresponding network account number is placed in the location pointed to by register 0.

Each routine uses a binary search algorithm to search its appropriate lookup table. The values used to control the binary search loop were set in the header of each table by HASPIRRE when the table was built. After performing the search, control is returned to the caller with one of the following codes in register 15:

Code	**Meaning**
0	Entry found; conversion performed as requested
4	Necessary table not defined at initialization time
8	Entry not found; no conversion performed
12	Neither JES2 nor network account number supplied

HASPEVTL: Event Trace Log Processor

HASPEVTL maintains the JES2 event trace log facility. It consists of:

- HA$PEVTL -- which manages the trace tables
- TRGETTB -- which gets more table storage if needed
- LOGGER -- which creates the started task logging job
- TTABFMT -- which calls the various formatting routines
- TROPEN, TRCPUT, TREOB, TRCLOSE, TRCKPT, and T$TRACK -- which are the various data management routines.
- TRCLGDIE -- which terminates the logging started task
- TRCERR -- which is the recovery routine
- Conversion routines -- which convert hexadecimal data to printable EBCDIC data.

An initially-generated started task control (STC) job (job name $TRCLOG) is started and, based upon certain criteria, the trace log data set is spun off to that job multiple times during the course of running JES2. JES2's security token is associated with the $TRACE STC. This is the token that will be used when SAF is called for early verification and for printing the data set.

The operator communicates with HASPEVTL via the $T TRACEDEF, $D TRACEDEF, $T TRACE(x) and $D TRACE(x) commands. The operator can control the trace logging on a dynamic basis with this command, which allows the changing of the SYSOUT class of the log data set, along with stopping the logging process or merely spinning off the current trace log, and the adding or deletion of trace tables. Through the TRACEDEF initialization statement, the system programmer specified the initial SYSOUT class; the size of the log data set in terms of lines of print so that, periodically, logs are spun off to the above SYSOUT class; whether or not the log process automatically starts after JES2 initialization; and the number of trace tables.

HASPEVTL Initialization

The initial entry point of HASPEVTL is HA$PEVTL, which is the JES2 trace table manager. It is responsible for serializing (using ENQ/DEQ logic and a set of rules involving the table pointers) the use of extended common service area (ECSA) trace tables among the address spaces that are tracing. The ECSA tables form a single-threaded circular queue. The $HCCT points to the tables that are currently being logged and filled. CCTTRLGG points to the table that is full and ready to be logged or currently logging. CCTTRTBL points to the table that is available for data additions. When a table becomes full and the next table pointed to is the log table (meaning all tables are full), trace entries are discarded. The CCTTRPLG field points to the table that precedes the log table. The only time this field is non-zero is when JES2 is attempting to decrease the number of trace tables.

HA$PEVTL is dispatched just after warm start processing as part of JES2 initialization.

On entry, HA$PEVTL establishes TRCERR as the recovery routine. It tests to see if more tables are needed. If needed, HA$PEVTL gets and initializes more tables, calling TRGETTB. If not, control passes to label TRCENQQQ.

TRCENQQQ tests to see if the current table (the one being written to) is equal to the logging table (the one being formatted and written out). If so, a $TRACE macro specifying TYPE=TRUNC is issued to advance the current trace table pointer so that this table can be formatted. If not, control passes to label TRCTRND, where the ENQ is issued to exclusively enqueue on the table. If the ENQ is not successful, the return code is checked: if it is 4, the routine waits to be posted. If the return code is unexpected, a $ERROR is issued. The routine abends with a system completion code of X '02D', and the return code is passed via the reason parameter on the ABEND.

If the ENQ is successful, but if logging is not supposed to be on, the logger routine (LOGGER) is not called, and the table is dequeued (DEQ) by the code at TRCDEQQQ. If logging at this point is supposed to be active, the routine waits for more work, and calls LOGGER when there is work. If logging is not supposed to be active, and a logging job has been started, TRCLGDIE is called to terminate logging. If the table should be freed (which the operator can request), the code at TBLFREE is entered, and a $FREMAIN is issued.

TRGETTB: Obtain Table Storage Routine

TRGETTB issues a $GETMAIN for the table storage. If the $GETMAIN is not successful, message $HASP652 is issued. If there are at least 3 existing tables, no attempt is made to get another table, and control returns to the caller. If there are fewer than 3 existing tables (which is below the minimum value), tracing and logging is turned off, and the tables are marked to be $FREMAINed before returning to the caller.

If the $GETMAIN for more table storage is successful, the new table is initialized. If there are no current tables, in the queue, CCTTRLGG and CCTTRTBL are updated. If there are tables already on the queue, this new table is added in front of the current table. When no more tables need to be obtained, control passes to TRCENQQQ.

LOGGER: Logging Routine

LOGGER is responsible for creating the logger STC job by calling HASPRJCS in HASPRDR if an STC has not already been created. The job header and trailer are set up for the $TRCLOG job, and the job is queued to the *in execution* queue. The "JES2 EVENT TRACE LOG NOW ACTIVE" message is issued, and TROPEN is called to open the trace log data set. If an STC already exists, the routine checks to see if the timer interval has expired. If not, the unexpired timer is cancelled. If so, T$TIMER is entered to check if trace entries are being discarded. If not, check to see if message $HASP654 is outstanding. If not, the formatting routine is called. If it is outstanding, the routine checks to see if 30 seconds has expired. If not, the message is retained, and the formatter is called. If so, the message is $DOMed. If entries are being discarded, the latest time the message was issued is recorded, and if it is outstanding, it is not reissued and TTABFMT (the formatting routine) is called at label TRCNOMSG. Otherwise, it is issued and then the formatting routine is called. A timer is also set for 19 seconds so that logging will not have to wait for a page to fill before a trace table is formatted.

The return code is tested on return from TTABFMT, and TRCTERM is called if the return code is an error return code. Then, routine TREOB is called, which writes a buffer to spool using a call to T$TRACK.

TTABFMT takes every trace table entry (TTE) and uses the appropriate formatting routine to format the trace information. Entries are written to the log data set until $TRLOGSZ is reached or until a $T TRACEDEF,SPIN command spins off the data set with a call to TRCLOSE, in which case a new data set is opened with a call to TROPEN so that logging may continue. A return is then made to HA$PEVTL.

Trace Log Processor Format Subroutines

A set of subroutines is provided to format individual segments of the trace output. These formatting routines are pointed to by the TID table pair entries. Each trace table entry (TTE) is formatted into a header line and, optionally, additional information lines that include blocks of data formatted into a standard dump format. The individual trace ID formatting routines call subroutines in this section to build the trace output.

The formatting routines are:

TROUT000: Format TRACE EVENTS DISCARDED information

TROUT001: Format $SAVE invocations for trace IDs 1, 11, and 18

TROUT002: Format $RETURN invocations for trace IDs 2, 12, and 19

TROUT003: Format $DISTERR invocations for trace ID 3.

TROUT004: Format BSC buffer trace at MBSCPROC for trace ID 4.

TROUT005: Format RTAM SNA buffer trace at MSNAPROC for trace ID 5.

TROUT006: Format JES2 initialization invocations for trace ID 6.

TROUT007: Format JES2 termination invocation for trace ID 7.

TROUT008: Format JES2 symbol trace for trace IDs 8, 9, and 10.

TROUT013: Format JES2 $EXIT macro invocations for trace ID 13.

TROUTFSI: Format $FSILINK macro invocations (ID 14),

TROUT017: This routine formats and prints the following type of trace records for trace ID 17:

- READ1
- READ2
- Primary write
- Intermediate write
- Final write.

TROUT020: Format JES2 $#GET service call information.

TRCHDR: Subroutine to Build the Standard Trace Header Segment

TRCHDR builds the standard trace header segment. This segment appears on every trace header line:

```
HH.MM.SS.TH     ID=*** - TIDNAME JOBNUMBER PCENAME
or
HH.MM.SS.TH     ID=*** - TIDNAME ASID NUM  PCENAMEX
```

where ID is the identification from the trace table entry ID (TTEID) of the trace table element, and the 8-byte name is the name from the $TRIDTAB entry.

The remaining subroutines and the text they create are as follows:

Subroutine	Text
TRCAT	AT symbol name
TRCPCE	PCE=pce-addr
TRCREGS	register 14, register 15, register 0, register 1
TRCIN	IN symbol name

TRCREGS: TRCREGS builds the following line:

```
R14-R1 = ******** ******** ******** ********
```

Register 3 points to the values used in the display line.

TRCPHEX: TRCPHEX converts 4 bytes of hexadecimal data to printable EBCDIC. It is invoked by macro $TRBIN2X, which is an internal macro used only by the event trace processor.

TRCPHEXB: TRCPHEXB converts any number of bytes of hexadecimal data to EBCDIC. The EBCDIC data is placed into the output record at the location pointed to by register TRCB. It is assumed that the result will not exceed the length of the output record. On return, register 3 (which on entry pointed to the bytes to be converted) has been increased to point past the bytes converted, and TRCB has been increased to point past the result in the output record. The value in register 3 points to the next field to be looked at by the calling routine. (This may be another value to be converted to EBCDIC, in which case TRCPHEXB will again be called.) The value in register TRCB prevents previously converted EBCDIC data from being overlayed with the results of the next conversion.

TRCDUMP: TRCDUMP formats a block of hexadecimal data to printable EBCDIC in standard dump format.

TEBCDIC: EBCDIC Translate Table

This translate table is used by TRCDUMP for formatting the EBCDIC portion of the dump lines. The table translates all characters to blanks except the following, which retain their own value: the letters A-Z; the numbers 0-9; and the special characters, left and right parenthesis ( ), plus sign +, minus sign -, ampersand &, dollar sign $, slash /, comma ,, period ., single quote ′, and the = sign.

TRVARFMT: Variable Format Fields Routine

TRVARFMT provides table-driven formatting for any number of fields appended to a standard, fixed format, trace table entry. Any of the trace table entry formatting routines (TROUTnnn) can branch to this routine after having formatted the fixed portion of the entry.

On entry, register 3 points past the fixed portion of the entry to TTEVFNUM, the number of items to be processed.

TRVARFMT determines whether or not variable fields exist by checking bit TTE1VARF in TTEFLAG1. Only if this bit is on is any processing performed. The one-byte field TTEVFNUM indicates the number of items to be processed. All items are contiguous, the first item immediately follows TTEVFNUM. Each item has the following format:

TTEVFLG1	TTEVTABI	TTEVDLEN	TTEVDATA
flags	table index	length of data	data
1 byte	1 byte	1 byte	

The TTEVTABI field value is used to index either a standard formatting table (TVFTB000) or a table associated with the trace entry id (TVFTBnnn). The standard table is used if bit TTEV1STD in TTEVFLG1 is on. The addresses of these tables are located in a vector (at TVFTBVEC) indexed by either 0 or the entry id. (This vector can have no missing entries up through the highest id for which a formatting table exists.)

The formatting table entry consists of flags indicating how the data is to be formatted and an 8-byte label field which may be blank. Dump and character formatting are supported. Dump format output always begins on a new line. If the label field is non-blank, it is produced on a new line by itself and then the dump format output begins on the following line.

Trace Log Data Management Subroutines

The following text describes the subroutines that handle the data management of the trace log data set.

TROPEN: Subroutine to Open the Trace Log Data Set

TROPEN initially obtains the storage for a spin IOT. It then creates a single peripheral data definition block (PDDB) within that IOT and sets the SYSOUT class, routing, and spin indication.

TROPEN ensures that the size of the TGAE area is the largest possible while leaving enough space for one PDDB. Next, a data buffer is obtained and the initial track address of the data set is obtained, via the T$TRACK subroutine, and placed into field PDBMTTR. The data buffer is initialized with the job and data set keys assigned and with the trace table log title line. The new spin IOT and updated JCT are rewritten to the spool, and control returns to the caller. If any errors are detected during the open processing (for example, if no storage is available), the TRCFLAG1/TRC1ERR flag is set, and the condition code is set to non-zero.

TRCPUT: Subroutine to Add a Record to the Trace Log Data Set

TRCPUT adds a record to the output data buffer. The length is passed in register 0. The left-most byte of register 0 contains the machine carriage control for the output record, and if it is not supplied, a write-space-1 control is assumed. For each record, a JES2 logical record control (LRC) block is built specifying the maximum logical record length (121), machine carriage control, the actual length of the data record, and the actual machine carriage control character. For each record added, the PDDB record count is increased by one. If this count exceeds the LOGSIZE specification from the TRACE initialization statement, the current data set is spun off via TRCLOSE and TROPEN. However, because the output for a particular trace ID can create multiple records, an attempt is made to keep all the output for a trace table entry together; therefore, the trace log data set temporarily remains open until the last line of the entry is processed (TRCFLAG1/TRCLASTL).

When no room remains in the buffer for trace records, TREOB is called to write the current buffer to the spool, and the buffer is reinitialized for use.

TREOB: Subroutine to Obtain Track Address for Buffer

TREOB obtains a track address for the next data buffer via T$TRACK and places this address into HDBNXTRIC, the chain pointer in the current data buffer. However, if this request is for the final buffer of the data set (indicated by the leftmost bit of register 14 set to 0), the chain field is set to 0.

The data buffer is written to the spool via $EXCP. To facilitate the reuse of this data buffer, register TRCB is reset to the first text area in the buffer, and control returns to the caller.

TRCLOSE: Subroutine to Close Trace Log Data Set

TRCLOSE performs the function of closing the current trace log data set and spins it to the current trace SYSOUT class. Provisions are made for held SYSOUT classes that are available for access by time-sharing users.

If the data set was never opened (TRCFLAG1/TRC1OPEN equal to 0), TRCLOSE exits immediately; otherwise, a check is made for an empty data set. If an empty data set is found, the spin IOT is freed from the common storage area via the FREEMAIN macro instruction, and control returns to the caller.

The final output buffer is written to the spool via a call to TREOB specifying the termination option (BALR, instead of BAL). The current SYSOUT class is obtained from $TRCLASS in the HCT and inserted into the PDDB. The lines/cards count is updated by the contents of field PDBRECCT, as is the total output count for the job.

If the SYSOUT class is a DUMMY class, all acquired spool space for the data set is purged ($PURGE), and control returns to the caller. The trace log data set is purged ($PURGE), and control returns to the caller. The trace log data set is physically written to the spool as a track-celled data set; however, the track cell indicator in the PDDB is set only if the SYSOUT class is eligible.

If the SYSOUT class is a held class and if the data set is printed at the local node, the hold indicator is set in the PDDB, the hold count is increased by one in the JCT, and a track is obtained for a hold queue table (HQT), if one has not already been obtained by a previous trace log data set. If the SYSOUT data set is held and, in addition is printed at another node, the hold at-destination indicator is set (PDB2HDST), and normal spin processing continues.

The spin IOT is placed onto the SVTSPIOT queue, and the JES2 execution processor is posted ($POST) to perform actual spin/hold processing.

Finally, the $HASP801 message is issued to inform the operator that a trace log data set has been queued for output and, optionally, that it is held. The open indicator is reset, and control returns to the caller.

TRCKPT: Subroutine to Interrogate Checkpoint Indicators

TRCKPT interrogates the checkpoint indicators in both the JCT (JCT1CKPT) and the IOT (IOT1CKPT) and rewrites them to the spool, if they are set. The $DISTERR macro instruction is issued if an error occurs during the I/O operation, and the TRC1ERR indicator is set.

T$TRACK: Subroutine to Obtain Track Address

T$TRACK obtains, via the $TRACK macro instruction, a new track address and returns it to the caller in register 1. If the $TRACK routine returns with condition code 0, a new track group was allocated and, therefore, the IOT is rewritten to the spool via TRCKPT to reflect the change in the track group map.

TRCLGDIE: Trace Termination Routine

TRCLGDIE is called to terminate the logging function. It $DOMs the outstanding $HASP654 message, calls TRCLOSE at label TRCTERM to spin off the current data set, terminates the logging job, and frees all acquired resources, and returns to HA$PEVTL.

TRCERR: Trace Error Recovery Routine

TRCERR obtains control from HASPTERM after it has taken an SDUMP. The purpose of TRCERR is to disable the the trace facility and allow JES2 to continue to run without it. The trace facility can be restarted by an operator command.

HASPSPOL: Spool Manager Services

HASPSPOL: HASP Spool Manager Processor

HASPSPOL supports the dynamic addition and deletion of spool data sets. HASPSPOL recovers or removes bad track groups on those spool volumes that have been found on the spool space. Bad track group isolation is an attempt to limit the effects of a single or small failure in the spool system.

HASPSPOL isolates track groups in which an I/O error has occurred from those which are available for allocation. HASPSPOL isolates the track groups only until I/O operations once again can be executed successfully. HASPSPOL attempts to return these track groups to availability as soon as the job that encountered the error, purges. If the tracks cannot be returned to the job at this point, they may be returned later during a warm or cold start. However recovery of bad track groups is attempted only when a job no longer owns them because the verification operation destroys all data currently contained in the records. To verify that the track in which the error occurred is now usable, HASPSPOL format writes the entire track and then rereads the records. If this verification operation is successful on all tracks in the track group, the track group is removed from the bad TGM and returned to the master TGM as available. HASPSPOL issues the $HASP251 message to inform the operator when the tracks are available, returned, or removed. Any track groups that fail this verification will remain in the bad TGM.

HASPSPOL is posted ($$POST) by HASCSRIC when work is available on the CCTIOERR queue. This is a queue containing track groups that must be placed in the bad TGM. Work on this queue is processed at SMDEQSV.

The processor is also posted ($POST) by the JES2 main task when work is available on the $SPOOLQ. The queue represents the bad track groups needing recovery. Work on this queue is processed at SMDEQSP.

HASPSPOL is posted ($POST) by the checkpoint processor or the command processor when work is available on the $DASWRKQ queue. The queue represents spool extents that require processing to dynamically start, drain, or halt the extent in the JES2 complex. This queue is processed at SMCMND.

SMDEQSV: SVTIOVERR Processing

At label SMQUEUED, HASPSPOL checks for work that is queued on the CCTIOERR queue and the $SPOOLQ queue. IF HASPSPOL finds work on the CCTIOERR queue, it branches to label SMDEQSV. At label SMDEQSV, HOSPOOL dequeues a TGB from the CCTIOERR queue and calls the $TGMSET service routine to mark the bad track group as bad in the bad track group map. This is done by setting the bit to 0 that represents the bad track group. HASPSPOL then frees the ($FREMAIN) the TGB and looks for more work at SMQUEUED.

SMDEQSP: $SPOOLQ Processing

If HASPSPOL finds work on the $SPOOLQ queue, it branches to label SMDEQSP. At label SMDEQSP, HASPSPOL dequeues a TGB from the $SPOOLQ queue. HASPSPOL then puts all necessary information into the SPL work area and posts (POST) the HOSPOOL subtask so that HOSPOOL can attempt to recover the bad track groups. HASPSPOL then waits ($WAIT) until it is posted (POST) by the HOSPOOL subtask. When posted by HOSPOOL, HASPSPOL checks to see if the recovery attempt was successful. SMWAIT issues the $HASP251 message to indicate whether the recovery attempt was successful or unsuccessful.

If the recovery attempt was successful, HASPSPOL removes the track group from the bad track group map and puts it in the good track group's map ($TGMAP) to make it available for allocation. It then frees ($FREMAIN) the TGB and looks for more work on the $SPOOLQ queue.

If the recovery attempt was unsuccessful, HASPSPOL frees ($FREMAIN) the TGB and looks for more work on the $SPOOLQ queue; the bad track group remains in the bad track group map.

SMCMND: Spool Command Entry

If HASPSPOL finds work on the $DASWRKQ, processing of spool addition and deletion commands will occur at this label. At this label, ownership of the shared queues is acquired and preparations are made to loop through the $DASWRKQ. Processing then continues at SMCLOOP. $DASWRKQ contains volumes for which an addition or deletion command is being processed.

SMCLOOP: The next DAS on the work queue is obtained. If the end of the queue has been reached, processing continues at SMQUEUED. If a DAS is obtained, the DADCKALL subroutine is called to determine the current command status (if any) and on return further checks are made to see if this is a new command, if an error during start processing has occurred (on this or other systems), or if final command processing for this volume should occur. If none of the above apply, the return code from DADCKALL is used to index into the branch table at SMCGO. The subroutines called from SMCLOOP (SMCFINAL, SMCWORK, SMCERR and SMCNEW) supply a return code that is used to index into this branch table. The return code for the table at SMCGO is as follows:

RC=0 All processing for this extent is complete. Use SMCFINAL to process completion messages and perform any other final processing.

RC=4 This JES2 system is to process the spool command at SMCWORK.

RC=8 Currently, there is no more that can or has to be performed on this system. Go to SMCLOOP to get another DAS from the $DASWRKQ and continue looking for work.

RC=12 An error has occurred in start processing on this or another system in the multi-access spool configuration. Go to SMCERR to deallocate the volume on this system (if necessary), issue an error message (if necessary), and return the volume to its previous status.

SMCNEW: Process New Command

Processing occurs here when DASFLAG2 indicates there is a new command. If DASBUSY indicates another system is processing (allocating or deallocating) a previous command, the new command will be deferred by returning with a return code of 8. If processing can proceed, DASFLAG is marked to indicate the new command. DADAVAIL is called to recompute the allocated and total track groups for the extents that can have space allocated. DADCKALL is then called to determine the processing required.

SMCWORK: Command Processing Routine: SMCWORK checks the DASFLAG to see if the DASTART flag is set to 1. If it is, this indicates a start command, and SMCWORK invokes SMCSTART to begin processing. If DASTART is zero, then a deallocation is required. SMCWORK calls SMCDEL to perform the deallocation because the command is either DRAIN or HALT.

SMCSTART: Start $DASWRKQ Processing Routine: SMCSTART allocates (and formats if necessary) the extent for the spool volume for every active system in the multi-access spool complex; the extent then becomes active (or begins to drain or halt depending on the type of start spool command). If a restart of a halted volume is taking place, all systems in the complex can attempt allocation at the same time because the extent information is already available and the volume will not be formatted. If the volume to be started is not previously known, one system in the complex must complete allocation of the volume before other systems can process start spool commands. This first system sets up extent information in the DAS and performs any necessary formatting.

If the volume is previously known, allocation of the volume occurs at SMSTALOC. If this is the first time the extent is being allocated, DADGTUCB is called to find the UCB. If formatting was requested, the SPL is marked to indicate that formatting is required. DADWTCMD is called to post HOSPOOL. If the extent was allocated successfully, DADXTENT is called to set up the extent information in the DAS and find a slot in the TGM for this extent. On return, a check is made to determine if the extent fit in the TGM. If the extent fit, DADTGM is called to format the slot in the TGM and the bad TGM.

If the extent size had to be reduced, the operator is given the option of using the volume with the reduced extent size or of not using the volume. If the operator replies to use the volume, processing continues at SMSTTGM. If the operator replies not to use the volume, processing continues at SMSTPRG where the SPL is set up, the volume is deallocated (by calling DADWTCMD), and the DAS is reset for reuse (by calling DADREMVE).

SMSTALOC: Extent Allocation Routine: SMSTALOC invokes DADGTUCB to find the UCB for the volume to be allocated. Then SMSTALOC calls DADWTCMD to post the HOSPOOL subtask to allocate the volume. SMSTALOC checks to see if the volume is formatted and if no errors have occurred thus far, indicates on the DAS that the volume is allocated. SMSTALOC then invokes DADDEB to store the extent that was allocated in the DEB. After the spool volume has been successfully allocated processing continues at SMSTEND. SMSTEND updates the CSA copy, and the JES2 SJXB copy of the DEB to reflect this allocated extent processing continues at SMWKEND.

SMCDEL: Deallocate Routine: SMCDEL determines if deallocation of a volume is ready to occur. Deallocation cannot occur on any system until all systems have re-BLOBed and no jobs on that volume are active for a halting volume or remain on that volume for a draining volume. If the re-BLOB processing on all systems hasn't yet occurred, processing is deferred by returning with a return code of 8. If any other systems in the multi-access spool configuration have started or completed deallocation, this system may begin the deallocation process by branching to SMDEALOC. Otherwise, processing continues at SMJOB to determine the job status.

SMJOB: Job Check Routine: SMJOB ensures that no jobs are active on a halting volume before any system begins deallocation for a halting volume or that no job remains on the draining volume before any system begins deallocation on a draining volume. When this routine begins processing, DASJOBNO contains the job number of the lowest numbered active job in the system (for a halt command) or the lowest numbered job in the system (for a drain command).

SMJOB locates a job using $QLOC. If the job can't be located DASJOBNO is increased and processing continues. If a job is located, the JQESPOLS mask is used to determine if the job resides on the volume. If it does not, processing continues at SMNXTVOB. If it does, the JQE is checked to determine if the job should be moved to another volume. If the JESNEWS data set or spooled message STCs are on the volume, they are moved to another volume at this time by the DADMOVER routine. Because the scan of the job queue starts with the lowest job number, and because jobs to be moved may not have the lowest job number until jobs with lower numbers are processed, the move might not happen immediately after the drain command is issued. If a read error occurs during the move process, the job is removed via a $QREM. If a write error occurs, the move is attempted again. If no tracks are available, further processing is deferred by returning with a return code of 8 (to be attempted again later when SMJOB is entered again on this or another system).

If the job does not have to be moved, processing continues at SMNOTMVE. If the volume is draining, further processing is resumed (and a retry later takes place, beginning with this job). If the volume is halting and the job or its JOEs are active, further processing is deferred. Otherwise, the job number is increased at SMNXTJOB and processing continues. When the entire job queue is completed (that is, for a halting volume, the job queue contains no active jobs on that volume, or, for a draining volume, the queue contains no more jobs associated with this volume because all jobs associated with this volume have either executed or have been moved), the volume is drained and the deallocation process can continue at SMDEALOC.

SMDEALOC: Volume Deallocation Routine: If HASPSPOL determines that a spool volume is ready to be deallocated, SMDEALOC is invoked to perform the deallocation. DADGTUCB is called to get the UCB for the volume that is to be deallocated and DADWTCMD is invoked to post HOSPOOL, which performs the deallocation. Processing continues at SMWKEND.

SMWKEND: Final SMC Work Processing: DADCKALL is called to determine if final processing should occur. Exit is made to SMCWORK.

SMCFINAL: Final Spool Command Processing Routine

SMCFINAL performs final start, drain, halt, and re-BLOB spool processing.

- Final Drain Processing

  At SMCFDRN, HASPSPOL calls DADREMVE to remove the volume from the DAS entries, issues the $HASP806 message, calls DADSPALE to log the track group usage, and calls DADKBYTE to recompute the byte available on the spool.

- Final Halt Processing

  At SMCFHALT, HASPSPOL calls DADREMWQ to remove the DAS from the work queue and issues the $HASP630 message.

- Final Start Processing

  At SMCFSTRT, HASPSPOL checks to see if this is just a start or a start with drain.

- Final Re-BLOB Processing

  HASPSPOL indicates that a search through the job queue for a drain or halt command can begin.

For a start with drain, HASPSPOL ensures that work gets selected for restart on the inactive volume by posting the JQE for jobs on the draining volume. Then, HASPSPOL posts the processors for these jobs.

For just a start, HASPSPOL does not post any work to be started. Instead, HASPSPOL calls DADAVAIL to recompute total and allocated track groups on the volume, calls DADREMWQ to remove the DAS from the work queue, calls DADSPACE to log the track group map space utilization, calls DADKBYTE to recompute the bytes on the spool, and issues the $HASP630 message.

HASPSPOL Subroutines

DADAVAIL computes the allocated and total track groups for the extent.

DADCKALL determines whether processing for the current command has been completed.

DADCKTGM checkpoints part or all of the TGM. It is passed an address which is within the TGM and a length which represents the amount of the TGM to be checkpointed. This routine issues a $CKPT for each portion of the passed segment of the TGM that is in its own page.

DADDEB initializes the direct access DEB extent information.

DADGTUCB finds the UCB for the volume id.

DADKBYTE computes K bytes of space on defined spool volumes.

DADMOVER is invoked to move a job that is affected by dynamic spool deletion commands. DADMOVER copies all of the job's spooled data sets and JES2 control blocks (JCT, IOTs) to currently active spool volumes; it is assumed that the job has no JOEs built for it.

The job lock of the job to be moved (the 'argument' JQE) is acquired via $GETLOK. A separate JQE is then built and added to the $INPUT job queue to serve as the anchor for the job copy. In case of an error in DADMOVER, this JQE (the "copy" JQE) is automatically purged by the warm start processor. The copy JQE is built to resemble the argument JQE except that its JQEJNAME is set to "$$$$MOVE". A new allocation IOT is created for use by $TRACK when allocating track addresses for the job copy. The argument JCT (if existing) is read, updated with new track addresses, and both the new IOT and JCT are written to spool. Both the regular and spin IOT chains are then copied via the DMVIOTC routine.

If the copy is successful, the argument job's IOT is processed by $PURGER and its JQE is removed by $QREM. The argument job's job number is contained in field JQEINJNO of the copy JQE. If the copy fails, the copy JQE is purged if the job numbers in the argument and copy JQE match. If they do not, an error message is issued.

DADREMVE removes extents from the DAS and compresses the master track group map and the bad track group map.

DADREMW0 removes the DAS from the work queue.

DADSPACE logs the space utilization of the track group map.

DADSPLST resets the SPL control block for reuse.

DADTGM adds the bit map for a new extent to the master track group map.

DADWTCMD posts and waits on HOSPOOL.

DADXTENT initializes JES2 and MVS data areas for each spool extent.

HOSPOOL: Dynamic Spool Allocation Subtask

The dynamic spool allocation subtask, HOSPOOL, supports the dynamic allocation of the JES2 checkpoint volume, checkpoint data set, and spool volumes with their corresponding spool data sets. It provides information for the construction of control blocks used in controlling these data sets and provides the function of testing and/or formatting the spool volumes. By providing these capabilities in a subtask, the processing of newly mounted spool volumes is allowed to proceed in parallel. The subtask can also attempt recovery of track groups on which I/O errors have occurred.

Information controlling the operation of the subtask is provided in a dynamic spool allocation work area (SPL), described by the $SPL macro expansion.

This work area resides in the work area extension of the HOSPOOL DTE.

Subtask Entry

On entry, HOSPOOL establishes $STABEND as its ESTAE routine in case of abnormal termination of the subtask. If HASPIRDA attached the subtask, a branch is taken to SPDYLIST, otherwise SPWAIT is given control. HOSPOOL sets a subtask flag byte, SPLFLG2, to indicate any failure to allocate the indicated volume and data set.

SPWAIT: Wait for Work

HOSPOOL uses SPWAIT to wait for work when none is available to process. SPWAIT first posts the HOSPOOL processor and issues a WAIT macro instruction. After returning from the WAIT, SPWAIT tests to determine if HOSPOOL is to be detached, and, if so, branches to SPEXIT. It then determines if SPBADTRK processing is required and, if so, goes to SPBADTRK. Otherwise, control goes to SPDYLIST.

SPDYLIST: Volume Allocation

HOSPOOL next constructs a dynamic allocation parameter list using information in the SPL. If SPL1UNAL is on, control goes to SPUNALOC. It then issues a DYNALLOC macro instruction to allocate the volume and its data set (SYS1.HASPACE). Note that the data set name can be modified by the DSNAME parameter on the SPOOLDEF initialization statement.

SPOBTAIN: Data Set Verification/Analysis

An OBTAIN macro instruction is now issued to obtain information about the allocated data set. An OBTAIN error results in message $HASP414 being issued, followed by SPUNALOC. If a spool volume is being processed, a direct-access device control block (DCB) and data extent block (DEB) are constructed in the SPL.

Information required for DAS entry construction is computed and saved in the SPL. If a checkpoint volume is being processed, control is passed to SPGXTENT to obtain the data extent information. Otherwise, the TGSIZE parameter value is used to determine the number of tracks per group for the spool voiume.

SPDASVLS: DAS Value Acquiring

HOSPOOL checks the data set type. If it's a checkpoint data set, the record size is set to 4K. If it's not a checkpoint data set, HOSPOOL checks the device type. If the device type is a cellular device, HOSPOOL computes the records per track. If the device is a count-key-data DASD device HOSPOOL computes its records per track. Finally, HOSPOOL computes the tracks per group for the spool volume.

SPGXTENT: Data Set Statistics Gathering

At SPGXTENT, data set extent information is obtained. Then, if a spool volume is being processed, a track group bit map for the volume is created in field SPLTGM.

SPSETIOB: Direct-access IOB Construction

A direct-access input/output block (IOB) is constructed and, if formatting is not demanded, the count portion of the first record on the last track of the first extent of SYS1.HASPACE is read. A read error results in a status bit being set in SPLFLG2. If formatting is not allowed, control is passed to SPGRDS. If formatting is demanded, an I/O error must have occurred on the read, or the length of the block read was not the same as the BUFSIZE specification; an attempt is made to format the volume.

SPFORMAT: SYS1.HASPACE Formatting

Formatting is preceded by message $HASP423. Space for a channel program is obtained via the SPSETCCW subroutine. A channel program is then constructed, providing for format writing one track at a time.

An error in formatting results in the message $HASP418 being issued, followed by SPUNALOC processing.

SPUNALOC: Termination Processing

If any processing errors have already occurred, the volume (for $Z or $P) is deallocated and the SPLFLG2 bit is set to 1 in the SPL2UNAL field. Otherwise, control goes to SPEXITCK.

SPEXITCK: Exit from Subtask

SPEXITCK restores the registers previously saved upon entry at the calling exit point. After restoring the registers, SPEXITCK returns control to JES2.

SPBADTRK: Bad Track Processing

SPBADTRACK formats the track group and rereads it to determine if the track group can be returned to service. If SPBADTRK encounters an I/O error, or if the data read is incorrect, it sets SPLFLG2 flag on. HOSPOOL branches to SPWAIT to wait for more work.

HOSPOOL Subroutines

SPEXCP initiates an I/O operation, waits for its completion, and returns.

SPWTO issues a message to the operator and returns.

SPBLDIOB constructs a direct access IOB.

SPFMTRK formats a track and checks for an I/O error.

SPRDCNT builds CCWs to read a record.

SPSETCCW sets up the format channel program.

SPFREEWK frees work storage ($FREMAIN).

SPESTAE handles subtask abends.

HASPSTAM: Spool Transfer Access Method

HASPSTAM consists primarily of three sections: HASPSTAM $EXTP service routines, HASPXFRM spool transfer I/O manager, and the HASPOFF spool transfer subtask, The HASPSTAM $EXTP routines (HASPXFRA) are a set of $EXTP service routines (OPEN, CLOSE, GET, PUT, and NCLOSE) that are called as subroutines from the spool offload receivers and transmitters. These service routines perform the same logical functions, as the ones in HASPBSC and HASPSNA. The difference is that records are written to and read from a BSAM format spool offload data set instead of a TP line.

The spool transfer I/O manager (HASPXFRM) runs as a JES2 processor and performs functions analogous to the HASPRTAM line manager. Among other things, it

- Processes BSAM I/O completions
- Passes input buffers to the appropriate receiver DCT
- Posts ($POST) awaiting transmitters when their buffers have been completely written
- Interacts with the spool transfer subtasks
- Assists in starting and draining spool offload devices

The spool transfer subtask (HASPOFF) is used to perform certain data management requests that could incur waits. These requests include:

- Allocation/Deallocation
- OPEN/CLOSE
- BSAM CHECK processing.

HASPSTAM $EXTP Routines

The HASPSTAM $EXTP service routines are called by the spool offload receivers and transmitters to perform I/O operations for spool offload using a BSAM format.

HASPXFRA: HASPSTAM $EXTP Entry Point

HASPXFRA is entered by the $EXTP service routine (HASPEXTP) in HASPNUC in response to a $EXTP request by a spool offload receiver or transmitter. Based on the service option in R14, the address of the specific $EXTP service (OPEN/CLOSE/GET/PUT/NCLOSE) is extracted from table XFRTAB. HASPXFRA then sets up addressability for the offload and receiver/transmitter DCTs (hereafter referred to as R/T DCT) and exits to the specified $EXTP service routine.

HASPXOPE: $EXTP OPEN Service for Spool Offload

HASPXOPE is used by the $EXTP service routines to provide OPEN service for spool offloading. HASPXOPE initializes the offload and R/T DCTs for subsequent GET/PUT processing. It increases the OPEN count (XDCTOPCT) in the offload DCT and marks the R/T DCT as being opened.

HASPXGET: $EXTP GET Service for Spool Offload

HASPXGET examines the queue of input buffers at DCTBUFAD in the receiver DCT. If there are no input buffers waiting to be processed, HASPXGET issues a $WAIT macro instruction to wait for work. This wait will be satisfied by the HASPSTAM I/O manager after an input buffer is read and SBUFPASS queues it for this DCT.

If an I/O error occurs during a READ operation, the waiting processor is posted and a negative condition code is passed back to the receivers to indicate the error.

If the skip flag is set (DCTSKIP) by the receivers on the next $EXTP GET call, HASPXGET calls SBUFPASS to free the buffer and issues another READ, passing a new buffer to the receiver.

If either receiver detects a job verification error, it will set a bit in the offload DCT (XDCT1VER). On the next $EXTP GET call, HASPXGET calls SBUFPASS to free the buffer and signals the I/O manager to drain the device.

After obtaining a new buffer, or if a buffer currently exists, HASPXGET examines the next logical record. If it is an EOB (end of buffer), HASPXGET calls SBUFPASS to free it and read another. If the record is for this receiver DCT, HASPXGET deblocks it and moves it into a return area. The record SRCB is stored for return in R0.

If HASPXGET detects an EOF (end of job) while it is deblocking the record, the EOF type (normal/abnormal) is obtained. If this was the last record, the buffer is given to SBUFPASS to be freed. If a normal EOF was found, the HASPSTAM exception exit (STAMXEX) is taken to indicate end of file. Otherwise the HASPSTAM abnormal exit (STAMXAB) is taken.

When the end of a job is encountered (EOF), HASPXGET halts the offload device if both receivers are now drained and, issues message $HASP582.

HASPXPUT: HASPSTAM PUT Service for Spool Offload

HASPXPUT examines the CCW pointed to by R0. It uses this CCW as a parameter list to $EXTP PUT to indicate the I/O operation, data length and data address. If the CCW op code is NOP, the HASPSTAM normal exit routine (STAMXIT) is immediately called. If the op code indicates a buffer truncation, the HASPSTAM normal exit routine (STAMXIT) is immediately taken.

If a buffer doesn't exist, HASPXPUT obtains one for this transmitter DCT in which records will be put to write to the tape.

At label STUFFBUF, the next record is put into the current buffer if it fits. STUFFBUF increases the record count and takes a normal HASPSTAM exit.

SPUFLUSH: Terminate Buffer

SPUFLUSH writes the buffer to the offload data set. If I/O errors were detected on previous writes, the HASPSTAM abnormal exit (STAMXAB) is taken. Otherwise, SPFWRITE is called to initiate I/O for this buffer, and the current PUT request is retried.

HASPXNCL/HASPXCLO: $EXTP NCLOSE/CLOSE Service for Spool Offload

HASPXNCL and HASPXCLO are used by the $EXTP service routines to close or NCLOSE open DCTs.

At entry, a flag is set to indicate whether this is a normal or negative CLOSE.

If the DCT was never opened, HASPXCLO takes a normal exit (STAMXIT). Otherwise, the R/T DCT is marked closed and the OPEN count in the offload DCT (XDCTOPCT) is decreased.

For a receiver CLOSE, the queue of awaiting input buffers is examined. If it is empty, DCTHOLD is set on and the STAMXIT exit is taken. If a buffer has been attached to the receiver DCT, a check is made to see if the receiver is draining. If not, DCTHOLD is turned off to allow this buffer to be processed, and exit STAMXIT is taken.

If the receiver is draining, SDRNRCV is called to check if the other receiver is drained. If not, the buffer is detached, and SBUFPASS is called to free the buffer; then another buffer is read.

If the other receiver happens to be drained, then chain the buffer to the offload DCT and halt the offload device to stop the read operation.

For a transmitter CLOSE, the EOF and EOB records are inserted into the buffer and SBFWRITE is called to write out the buffer. If this is a negative CLOSE, the EOF record indicates an abnormal termination. A $WAIT is then issued to wait for work. This wait is satisfied by the spool transfer I/O manager when a buffer write completes. If the queue of awaiting output buffers is empty, normal exit is taken. Otherwise the $WAIT macro instruction is reissued. When HASPXCLO exits, all data is secure on the offload data sets.

HASPSTAM Exit Routines

The three exit routines set positive, negative, or zero condition codes and invoke the $RETURN service. These routines are:

- STAMXIT - Normal exit and sets a positive condition code
- STAMXEX - EOF (end of job) detected on GET, sets a negative condition code
- STAMXAB - Abnormal exit and sets a zero condition code

SDRNRCV: Test Receivers Drained Status Subroutine

SDRNRCV checks the status of the two receivers. When the receiver being tested is detected as drained or draining, the routine checks the status of the other receiver. If the other receiver is drained, the offload device is halted.

SGETOBUF: Obtain Output Buffer Subroutine

HASPXPUT calls SGETOBUF to obtain an output buffer (via the $GETBUF macro instruction specifying TYPE=SPXFR) and chains it to the transmitter DCT. SGETOBUF examines the buffer count field in the offload DCT to see if it is below the buffer limit. If so, SGETOBUF gets a buffer and initializes it, and increases the buffer count (DCTBUFCT) in the offload DCT. If the buffer count is not too high, SGETOBUF waits ($WAIT) for work. This wait will be satisfied by the spool transfer I/O manager (at HASPXFRM) when a previously obtained buffer is written out and freed.

SQUEBUF: Queue Buffer on DCT Queue Subroutine

SQUEBUF is used by the HASPSTAM service routines to queue a buffer onto the end of a chain of buffers. The address of the head of the chain of buffers is located in the DCT. The DCT address and buffer head offset are passed to SQUEBUF so that this routine will queue buffers for both input and output operations.

SDRANXFR: Drain offload and Associated R/T DCTs Subroutine

SDRANXFR is called by SBUFEMTY, SBUFPROC, SDCTSCAN, or STIMRCHK to drain the offload and its associated R/T DCTs. SDRANXFR sets on the drain bits in the offload and all R/T DCTs to indicate that they are to be drained. If the open count in the offload DCT is non-zero, SDRANXFR exits because there is still processing taking place. If this is an input operation, SDRANXFR calls SFORCDRN to purge any remaining input buffers. The offload subtask is posted for closing and termination.

SDRANXFER calls SSETLCK before it issues a CLOSE to indicate that the device is no longer in use. SDRANXFR then posts the JOT and JOB resources to force the transmitters into their idle loop, and issues a $FREUNIT for any inactive transmitters or receivers so that message $HASP097 can be issued.

SSETLCK: Set Lock Subroutine

When an offload device is drained, SSETLCK indicates this in the LCK so that another device can start, and checkpoints the LCK.

SFORCDRN: Purge Remaining Input Buffer Subroutine

SFORCDRN is called by SDRANXFR to purge remaining input buffers prior to draining the offload and its associated R/T DCTs. SFORCDRN frees all waiting input buffers (via $FREEBUF) for all receiver DCTs associated with a given offload DCT.

SBFWRITE: Initiate I/O for Output Buffer Subroutine

SBFWRITE is called by SPUFLUSH to initiate I/O for an output buffer. SBFWRITE moves the buffer from the transmitter DCT to the end of the queue of buffers awaiting I/O in the offload DCT. SBFWRITE then issues the BSAM WRITE macro instruction to initiate I/O for the buffers. SBFWRITE calls SECBSET to initialize the ECB contained in the buffer.

SINITRD: Initiate I/O for Input Buffer Subroutine

SINTRD is called by SBUFPASS and SUBTCOMP to initiate more input buffer I/O. SINITRD checks to see if the number of buffers in use by this offload DCT is at its maximum or if an EOF has occurred on the offload data set. In either case, SINITRD immediately exits. If it's not an EOF or if the number of buffers used is not a maximum, SINITRD obtains a buffer via a $GETBUF macro instruction specifying TYPE=SPXFR. SINITRD then increases by one the offload DCT buffer count (DCTBUFCT) and places the buffer at the end of the I/O pending queue (XDCTBUFQ) in the offload DCT. SINITRD then issues BSAM READ for the buffer and calls SECBSET to initialize the ECB contained in the buffer. The entire process is then repeated for each input buffer beginning at the initial check until there are no more input buffers to process.

SECBSET: Initialize Extended ECB Subroutine

SECBSET is called by SBFWRITE and SINITRD to initialize the ECB contained in the DECB for the spool offload device. This is an extended ECB specifying that post exit processing should take place. After this initialization, an exit routine in HASPNUC is given control when the ECB is posted by BSAM to indicate I/O completion. This exit routine queues the buffer for the spool transfer I/O manager.

SECBSET operates in supervisor state key 0 and uses a local lock when accessing the ECB. If the ECB has already been posted, SECBSET queues the buffer to the $XFRBEND queue in the HCT using compare and swap. Otherwise, SECBSET initializes the ECB to indicate that post exit processing is to take place. The local lock is released, and the key is restored to the HASP key.

SBUFPASS: Pass Buffer to Correct Receiver DCT or Free if Empty Subroutine

SBUFPASS is called by HASPXGET and SBUFPROC to pass a buffer to the correct DCT or to free the buffer if it is empty or if EOF is encountered. If the read for this buffer resulted in an EOF, the EOF indicator (DCTEOF) in the offload DCT is set to 1. If EOF is encountered or the buffer is empty, or the maximum READ error count has been reached, the passed buffer is freed and the buffer count in the offload DCT (DCTBUFCT) is decreased by 1.

Any open receivers must be $POSTed so that they can issue an $EXTP CLOSE to drain the devices. If the buffer count becomes zero (indicating I/O has quiesced and the end of file was encountered on the offload data set), SDRANXFR is called to terminate operations. Otherwise, SINITRD is called to initiate the processing of more input.

If SBUFPASS is called from HASPXGET by a receiver skipping buffers to the next job, the buffer is freed and SINITRD is called to read another.

But if the device is halted, no new READ is issued until the device is restarted for receiving.

If EOF is not encountered and the buffer is not empty, SBUFPASS scans the receiver DCTs for a DCT whose RCB matches the RCB of the next record in the buffer; this is done in order to determine which receiver is to process the buffer. If such a DCT is found, SBUFPASS queues the buffer to the DCT, resets the DCTHOLD bit and posts ($POST) the DCT's associated PCE so that the receiver can process the buffer. If no matching DCT is found, a disastrous error (SBFRCBER) is issued and the offload device is drained.

If SBUFFPASS is called from HASPXGET because of a job verification error, it issues message $HASP590 and $POSTs the open receivers with a negative return code so that they can be closed and the devices drained.

Spool Transfer I/O Manager

The HASP spool transfer I/O manager runs as a HASP processor under the standard sub-dispatching structure of the JES2 main task. It is responsible for completing processing of certain events, some of which occur outside the JES2 main task. This completion processing consists mainly of updating and manipulating control blocks and in some cases posting ($POST) other processors to notify them that these events have occurred.

There are four events that cause this processor to take action. They are:

1. BSAM has signaled I/O completion for a spool offload buffer. Completed buffers are queued to $XFRBEND in the HCT by a post exit routine in HASPNUC. A non-zero value in $XFRBEND indicates one or more of these events have occurred.
2. The operator has issued a start, drain, or halt command for an offload DCT. A bit in the $STIMASK in the HCT indicates one or more of these events has occurred.
3. The offload I/O subtask has finished its processing on behalf of a given DCT. When finished with processing, the subtask queues DCTs to the $XFRDEND queue in the HCT. A non-zero value in this field indicates one or more of these events has occurred.
4. A 32-second timer has expired. This event is detected by examining this processor's TQE.

HASPXFRM: Processor Initialization

HASPXFRM establishes addressability, issues an IDENTIFY macro instruction for the offload I/O subtask (HASPOFF) and initializes the TQE.

SEARCH: I/O Manager Scanner/Dispatcher

SEARCH is entered at the end of processor initialization to detect events and dispatch the appropriate event processors. SEARCH is table driven; the scan table that it uses contains three elements per entry:

1. A list of bits, which must be on, in the $STIMASK.
2. An instruction which, when executed, must produce a non-zero condition code.
3. The address of the event processor.

If conditions 1 and 2 are met, the event processor is given control.

Upon entry, SEARCH resets the $$POST element in the $HCCT to allow subsequent $$POSTs to be issued. If the TQE indicates that the 32-second timer has expired, SEARCH sets the timer scan bit on in a temporary copy of the $STIMASK. The other $STIMASK bits are merged into this copy and the $STIMASK is reset.

SEARCH processes the scan table and calls the appropriate event processor. After completion of table processing, SEARCH tests to see if the timer has expired or if any $STIMASK bits have been set on. If either condition is true, SEARCH is again entered.

If there are any active offload DCTs and the timer has expired, the timer is restarted and SEARCH waits ($WAIT) for work. This wait is satisfied by various components of JES2 when an event occurs requiring this processor's action. SEARCH is entered when the wait is satisfied.

SBUFPROC: Buffer I/O Completion Event Processor

SBUFPROC is called by SEARCH to copy the queue of buffers at $XFRBEND and to reverse their order from LIFO to FIFO. The buffers are then processed one by one. If the buffer is an output buffer, SBUFPROC examines the associated R/T DCT to see if it is waiting. If it is waiting, $BUFPROC posts ($POST) its PCE. If this is not the first buffer on the offload DCT I/O pending queue, then SBUFPROC issues a catastrophic error $X01. If the I/O was successful, SBUFPROC dequeues and frees the buffer.

If the buffer is an input buffer and the I/O is successful, it is given to SBUFPASS to be passed to the appropriate receiver. If this is not the first buffer on the offload DCT I/O pending queue, then $BUFPROC issues a catastrophic error $X01.

If the I/O was not successful, (detected by a bit, SPBSYNAD, set on in the buffer by a SYNAD routine), the error count is incremented, and this routine sets an I/O error flag in each open receiver DCT. SBUFPROC calls SBUFPASS to process the error. To indicate the error, SBUFPROC issues error message $HASP587. The message will indicate if the limit of ten consecutive errors has occurred. If, before the limit is reached, an I/O attempt is successful, the count of errors is cleared to zero.

SDCTSCAN: Complete Processing of Start or Drain Command

SDCTSCAN is called by SEARCH to scan for all offload DCTs in starting or draining status.

It also serializes the starting of devices among members of a MAS installation and verifies that data set names are unique.

For a starting DCT, SDCTSCAN issues a $GETUNIT macro instruction to obtain the offload DCT, and chains the starting DCT to the $XFRACTV queue.

Before attaching a spool transfer subtask, the I/O error count is reset. This is used to count the number of READ errors for a load operation. Then, SDCTSCAN attaches a spool transfer subtask for this DCT and signals the spool transfer subtask (by setting XDCTSUBR=SBTOPENR) to do allocate/open processing for the offload data set. If this initialization cannot take place, disastrous errors are issued and a $FREUNIT macro instruction is issued for the DCT.

If the device is restarting after a $Z OFFn command (bit XDCTSTRT in the DCT is turned on), SDCTSCAN does not call the subtask to reallocate the data set. It just posts any startable receiver if the device was receiving or any startable transmitter if the device was transmitting.

For a draining DCT, SDCTSCAN calls SDRANXFR to drain the offload and its associated R/T DCTs.

SUBTCOMP: Subtask Completion Processor

SUBTCOMP is called by SEARCH to process the queue ($XFRDEND) of completed offload DCTs.

For a offload DCT that has just completed OPEN processing, SUBTCOMP checks to see if the OPEN was successful. If it was not successful, SUBTCOMP performs CLOSE completion processing. If the OPEN was successful and the OPEN was for input, SUBTCOMP calls SINITRD to initiate input I/O and then starts and posts ($POST) those receivers that have bit DCTSTRT turned on in their DCTs. If the OPEN was successful and was for output, SUBTCOMP starts and posts ($POST) those transmitters that have bit DCTSTRT turned on in their DCTs.

For a offload DCT that has just completed CLOSE processing, SUBTCOMP removes the DCT from the $XFRACTV active queue. The drain bit (DCTDRAIN) is then set to 1 in the offload DCT and a $FREUNIT macro instruction is issued for the offload DCT.

If there was an error during allocate/open processing, the device is not started. Instead, SSETLCK is called to clear the LCK and indicate the device is no longer in use.

STIMRCHK: Timer Completion Processor

STIMRCHK is called by SEARCH to scan the queue of active offload DCTs. For each DCT with the drain bit set on, STIMRCHK checks to see if any of its receivers or transmitters are active. If none of these receivers or transmitters are active, STIMRCHK calls SDRANXFR to drain the offload device

Spool Transfer Subtask

The spool transfer subtask is used to perform data management tasks that could incur waits (for example, allocation, deallocation, OPEN, CLOSE, BSAM check and EOV processing).

HASPOFF: Spool Transfer Initial Entry Point

HASPOFF is entered to store registers and establish initial addressability. The DTE work area is then initialized from the work area prototype contained in module HASPSTAM. HASPOFF then adjusts addressability so that the work area prototype actually forms the DSECT for the work area. Finally, HASPOFF issues an ESTAE to establish an abnormal end exit.

SBTDISP: Subtask Dispatcher

SBTDISP is entered by subtask initialization and SPTRCOK (after the wait for work is satisfied) to mark the DCT as being in subtask processing, examine the request code in the DCT, and dispatch the appropriate request handler.

SBTRETC: Subtask Termination

SBTRETC is called to terminate the subtask. SBTRETC calls SBTRETRN, which will return the DCT to the main task. SBTRETC frees the work area and issues the RETURN macro instruction to terminate the subtask.

SBTOPEN: Allocate, Open and Ready Offload Data Set for Processing

SBTOPEN is called to construct a dynamic allocation parameter list in order to allocate the offload data set with a disposition of DISP=(OLD), and to issue the DYNALLOC SVC. If this DYNALLOC fails with a locate error, but a unit name has been specified on the UNIT= parameter of the OFFLOADn initialization statement and the offload was started as a TYPE=TRANSMIT, then the dataset is allocated with a DISP=(NEW,CATLG). If the dynamic allocation fails, SBTOPEN calls DYNALERR to format the error message using the TSO DAIRFAIL routine. The subtask is then terminated. If the allocation is successful, SBTOPEN initializes the DCB with the proper DDNAME.

SBTRCOK: Subtask Successfully Complete

SBTRCOK is entered from SBTOPEN when SBTOPEN has successfully read or written a descriptor record. SBTRCOK sets the DCT subtask return code field to zero and returns the DCT to the main task. The spool transfer subtask then waits for more work. This wait is satisfied by SDRANXFR and SBUFPROC.

SBTCLOSE: Close and Deallocate Offload Data Sets

SBTCLOSE is entered by the spool I/O manager to issue the CLOSE macro instruction, build a dynamic allocate parameter list, and issue a DYNALLOC macro instruction to deallocate the offload data set. Control then passes to SBTRETC.

SBTCHECK: Perform Check for Spool Offload Buffer

SBTCHECK is called by the spool I/O manager to issue the CHECK macro instruction for the first buffer on the I/O pending queue of the given DCT. If either the SPBSYNAD or SPBEODAD routines are entered, corresponding bits are set in the SPBFLAG1 flag in the buffer. SBTCHECK requeues the buffer to the $XFRBEND queue and control passes to SBTRCOK.

SBTRETRN: Return DCT to JES2 Main Task

SBTRETRN is called by SBTRETC and SBTRCOK to return a DCT to the main task. The passed return code is stored in the DCT subtask return code field (XDCTSUBC) and a bit (XDCTSUBT) is reset in the DCT to indicate that subtask processing is complete. The DCT is placed on the $XFRDEND queue using compare and swap logic. The spool transfer I/O manager is then posted by issuing a $$POST macro instruction.

SBTESTAE: Subtask Abnormal and Exit Routine

SBTESTAE is called by the recovery termination manager (RTM) when the spool I/O manager discovers an abnormal exit condition. SBTESTAE issues the SETRP macro instruction to invoke the SBTRETRY retry routine if an SDWA was provided by the caller. If no SDWA was passed, SBTESTAE returns with return code information that causes SBTRETRY to be invoked.

SBTRETRY: Abnormal End Retry Routine

SBTRETRY is entered by the recovery termination manager (RTM). SBTRETRY reestablishes addressability and formats a message indicating the type of abnormal end. SBTRETRY then prints the message using the $$WTO macro instruction, sets the subtask return code to X'FF', and passes control to SBTRETC, where the subtask is terminated.

DYNALERR

This routine calls the TSO DAIRFAIL routine to format the error message if dynamic allocation fails. Parameters for the DAIRFAIL routine include the address of the SVC99 request block, the address of a field containing the SVC99 return code, a caller ID number, and the address of a buffer in which to return the formatted error message.

HASPRAS: Error Service Routines

Contains service routines for error situations in the JES2 main task and its subtasks.

$ESTAE Services

The $ESTAE service routines maintain the processor recovery element (PRE) stack. A PRE represents a recovery routine and contains its entry point address. It also provides an area (PRETRACK) in which the processor can maintain serviceability data and/or status information required by the recovery routine. If the processor is maintaining serviceability data, it must set PRELOGLN (1 byte) to the number of bytes of PRETRACK to be written to SYS1.LOGREC. LOGREC recording is accomplished by setting RECORD=YES in the $SETRP macro in the JES2 ESTAE exit. $LGRR is the DSECT mapping macro that defines the LOGREC record format.

The PRE stack has a LIFO organization. The topmost PRE represents the most recently established recovery environment and identifies the routine that is to receive control in event of an error.

Each time an $ESTAE macro is issued with RECADDR= specified, it creates a PRE (this is the only way a PRE is created). Zero, 1, or more PREs can be created each time a $SAVE macro is issued. The PREs can be created before the processor has acquired a processor save area (PSV). PREs are associated with the save area level at which they are created; that is, the information contained in a PRE reflects the status of the environment at the time the last $SAVE macro was issued. PREs created within subroutine code that is logically bracketed by $SAVE and $RETURN macros are automatically cancelled in $RETURN processing (the $ESTAE macro with CANCEL specified is actually issued in $RETSAVE).

$ESTAER: Establish $ESTAE Processor Recovery Element

$ESTAER is called by a $ESTAE macro with RECADDR= specified to establish the processor's recovery element (PRE). $ESTAER acquires a PRE via a $GETWORK macro, initializes it, and adds it (LIFO) to the processor's PRE stack.

$ESTACAN: Cancel $ESTAE Processor Recovery Element

When $ESTAE macro with CANCEL is specified, this routine cancels the most recently established PRE. If the most recent PRE was not created by the last $SAVE macro that was encountered, catastrophic error $ER1 results. Otherwise, $ESTACAN removes the most recent PRE from the processor's PRE stack and frees it with a $RETWORK macro.

$ESTAREP: Replace $ESTAE Processor Recovery Element

When $ESTAE macro with REPLACE, RECADDR= is specified, this routine replaces the address of the recovery routine in the most recently established PRE with the address specified by RECADDR=. If the most recent PRE was not created by the last $SAVE macro that was encountered, $ESTAREP issues catastrophic error $ER1. Otherwise, the address of the recovery routine in the most recent PRE is replaced with the address specified via the RECADDR= operand.

$STABEND: Common JES2 Subtask Estae Routine

Each subtask establishes $STABEND as its recovery routine and provides its DTE address with the ESTAE macro. It provides diagnostic information for problem determination (and ensures that diagnostic information is recorded in SYS1.LOGREC) and attempts subtask recovery. A system dump can be obtained based on installation-defined recovery options (RECVOPTS). Each subtask DTE specifies its own retry, VRA formatting, and clean-up routines. These routines are called by $STABEND. If recovery is not possible, $STABEND terminates the JES2 main task with a $ERROR Z03.

Error Routines

The following routines handle error conditions that occur during JES2 processing.

$DISTERR: Disastrous Error Routine

When reading a job control table (JCT) or input/output table (IOT), this routine is entered from a processor whenever a physical I/O error is detected. The operator is notified of the error and processing continues, although JES2 should be quiesced and restarted as soon as practicable to recover any direct-access space that may have been lost as a result of the error.

This routine also provides a dump (using the $SDUMP macro instruction) based on installation-specified recovery options. The dump title contains the subsystem name, the error symbol name, and error module name.

When this routine is entered, the symbol name and module name are moved into the message from the $DISTERR macro expansion. A $WTO is then issued to notify the operator of the error and control is returned to the calling processor. The message to the operator is as follows:

```
$HASP096 DISASTROUS ERROR AT SYMBOL symbol IN CSECT module
```

This message also provides job information if JOB=YES was specified on the $WTO macro instruction.

$IOERROR: Input/Output Error Logging Routine

This routine is entered whenever an unrecoverable input/output error occurs on a JES2 spooling volume, or whenever a line error occurs that requires the attention of the operator. A message is generated describing the error, and this message is routed to the operator through the operator's console. The routine then returns without taking any further action.

When this routine is entered, register 1 contains the address of the input/output block (IOB) that is associated with the failing input/output operation. The channel status, channel command code, sense information, track address, and line status are retrieved from the IOB and are formatted; the unit address and volume serial are obtained from the unit control block (UCB); the device name (if applicable) is acquired from the device control table (DCT); and the message is written to the operator's console.

For all errors on a spool volume, this routine locates the track group that contains the failing record. When the defective track group has been located from information in the IOB, the appropriate bit in the bad track group map is flagged off. Finally, the spool I/O error is queued to the HASPSPOL processor so that it can remove the bad track group from service.

The format of the message generated is described in *System Codes*.

$SDUMP Service Routine

$SDUMP is a service routine called via the $SDUMP macro. It provides a central service of requesting SVC dumps for all of the JES2 main task (not just during termination or recovery).

$SDUMP determines the location information of the $SDUMP macro that called it and formats and issues the $HASP080 message. If a dump title was not passed or if only text to be appended to the default title was passed, $SDUMP builds a default dump text consisting of the $HASP080 message text. An ASID list is built of from 1 to 3 address space identifiers, based on the options specified with the macro.

$SDUMP issues the MVS SDUMP macro, passing the title and ASID list it has built. The SDATA options passed to SDUMP are (ALLPSQ, RGN, TRT, SQA, CSA, LPA, SWA, LSQA). If the dump fails, the next $SDUMP action depends on the ERROPT operand specified in the $SDUMP macro. $SDUMP either issues the $HASP081 message and returns to the caller, or issues WTOR message $HASP089, which allows the operator the ability to specify that $SDUMP should retry the dump.

HASPMSG

HASPMSG contains a routine that supports the message building function of $SCAN for many checkpoint messages. It also includes $SCANTAB entries for these messages and their pre- and post-scan routines. HASPMSG contains $MSGDISR, which is called from the SCAN facility to construct the list form (MF=L) of the WTO, WTOR, or MLTWO macros, and to execute the MF=E form of the macro.

Chapter 4. Directory

The following directory lists the main entry points available within JES2. Given are the modules in which the entry point exists, the entry point name, and a brief description of the function performed by the code.

The following information to aid you in understanding and trouble-shooting JES2 is available on microfiche:

MVS/ESA Information	**Microfiche Order Number**
Data Areas	LYB8-1850

Figure 4-1 (Page 1 of 13). JES2 Directory Information

Object Module	Entry Point	Function
HASPNUC	$WAIT	Puts the current processor in the $WAIT state by queueing the PCE to itself, and dispatches other ready processors.
	$POST	Makes an individual processor ready for CPU time by removing the specified processor control element (PCE) from its current queue and queueing it to the $READY queue.
	HASPPXIT	Notifies a processor that there is work.
	$GETWORK	Obtains a work area.
	GTWKTABL	Defines work area pools.
	$RETWORK	Returns a work area.
	$EXCP	Schedules I/O for JES2 main task through EXCP or EXCPVR.
	HASPEXTP	Service entry routine: Processes all $EXTP service requests to open or close a remote device, or to get a record from, or put a record to, a remote device.
	$QACT	Indicates a JQE is active and possibly obtains an extension for it.
	$QADD	Adds a new job queue element (JQE) to the active job queue.
	$QJIX	Obtain a job number and add a JQE to the job queue index (JIX).
	$QPUT	Releases control of a JQE on the active job queue.
	$QGET	Obtains control of a JQE on the active job queue.
	$QREM	Removes a JQE from the active job queue.
	$QLOC	Locates a JQE by job number.
	$QMOD	Modifies a JQE on the active job queue.
	$CKPT	Schedule a checkpoint for an altered element.
	$QSUSE	Requests access to job queue and JOT queue checkpoint records.
	$CHECK	Checks for completion of the checkpoint write associated with a passed token.
	$GETUNIT	Obtains control of a JES2 device.
	$FREUNIT	Releases control of a JES2 device.
	$STIMER	Schedules a JES2 timer queue element (TQE).
	$TTIMER	Returns remaining time on a JES2 TQE.
	$TIMER	Timer processor: Handles STIMER completions.
	$STCK	Entry to store clock service routine
	DATECONV	Day to date conversion routine
	$GETSMFB	Obtains a JES2 system management facilities (SMF) buffer from the free pool.
	$QUESMFB	Queues a JES2 SMF buffer to the $$MFBUSY queue for processing by the HASPACCT subtask.
	$GETSAVE	Obtains a JES2 save area and save registers.
	$RETSAVE	Returns a JES2 save area.
	$RETURN	Returns to the calling routine.
	$PGSRVC	Issues an MVS PGSER macro to free, fix, or release storage
	$GETBUFR	Gets a buffer from a buffer pool.
	$FREEBFR	Returns a buffer to its buffer pool.
	$BFRBLDR	Constructs buffer prefix: Entered through the $BFRBLD macro instruction constructs an IOB or RPL beginning at the first byte of a JES2 buffer for the purpose of reading into or writing from that buffer.
	$GFMAIN	Provides a branch entry interface for MVS GETMAIN/FREEMAIN services.
	$GETLOKR	Obtains the MVS CMS lock.
	$FRELOKR	Releases the MVS CMS lock.
	$GETJLOK	Gets the JES2 job lock for a JQE.
	$FREJLOK	Frees the JES2 job lock.

Figure 4-1 (Page 2 of 13). JES2 Directory Information

Object Module	Entry Point	Function
	HASPATTN	Entry for unsolicited device ends: Provides support for the automatic starting of paused output devices and of readers.
	$IOAPPEN	JES2 input/output appendage table.
	$CKAPPEN	I/O appendage vector table and appendage routines for CKPT I/O.
	$POSTEX	Posts the spool transfer function for I/O started by the main task.
	$FIXEND	Indicates end of HASPNUC fixed storage requirement.
	$ASYNC	Asynchronous I/O processor: Handles the synchronization of I/O interruptions with their associated JES2 processors.
	$JESEFF	Entry to JES2 main task exit effector. This is the interface to user exit routines for the $EXIT macro.
	$JCTIOR	JCT I/O exit routine: It is entered after the successful read of a JCT from the spool or before a write of a JCT to the spool.
	$DYN	Entry to MVS to dynamically allocate and deallocate JES2 devices.
	$GETUCBS	Obtains a UCB address using IOS UCB scan routine.
	$GETUCON	Provide entry for recall of $GETUCBS
	$FREUCBS	Frees storage for the UCB parameter list.
	HASP	Initial HASJES20 module entry point.
	HOSALLOC	Dynamic allocation/unallocation subtask.
	$DSOPEN	Open routine for the job log.
	$DSPUT	Put routine for the job log.
	$DSCLOSE	Close routine for the job log.
	$MODCHK	Module verification service.
	$MODLOAD	Module load service.
	$MODELET	Module deletion service.
	MOD875	Issues the HASP875 message for errors detected by $MODLOAD and $MODCHK.
	$XEBKIL	Removes $XECBs from all JES2 dispatcher queues.
	$SEAS	Security authorization service.
	$SEASMSG	Issues the HASP077 error message for security processing failure.
	SUBDEST	Subtasks $DEST authorization processing.
	$SUBIT	General purpose subtask queuing service.
HASPBSC	HASPBSCA	Entry point for BSC $EXTP service routines.
	MWRSPCCW	RTAM BSC CCW skeleton.
	HASPBACT	BSC active line DCT scan routine. Called by the line manager scan routine.
	HASPBUNT	BSC inactive line DCT scan routine. Called by the line manager scan routine.
	HASPBPRO	BSC channel end processing routine. Called by the line manager scan routine.
	MLLMRCV0	Performs error recovery processing for the line manager processor.
	HASPBSLN	Completes start of dedicated lines after $SEAS request for the remote has completed.
HASPCKPT	HA$PCKPT	Checkpoint processor: Writes checkpoint job queue and job output table (JOT) queue records.
	KTRK1IO	Performs I/O for track 1.
	KFORMAT	Formats the specified checkpoint data set.
	KIOERROR	Analyzes I/O errors.
	KREAD2	Reads changed 4K records into the I/O area.
HASPCKDS		Checkpoint data set routines that support checkpoint processing, including the checkpoint reconfiguration operator dialog.
	CKBINIT	Initializes internal control blocks to do checkpoint I/O.
	CKPTALOC	Performs dynamic allocation of the checkpoint data set.
	CKPTUNAL	Performs dynamic unallocation of the checkpoint data set.
	CKPTALCLN	Cleans up after a failed dynamic allocation attempt.
	KDIALOG	Dialog routine to satisfy the requirement for a checkpoint data set reconfiguration.
	KBLDCKB	Obtains a CKB and builds CCW packets associated with track 1.
	KBLD4KP	Builds CCW packets used for 4K records.
	KRESERVE	Issues the MVS RESERVE macro on the checkpoint data set.
	KNOP	Issues a NOP CCW to get a hardware reserve
	KRELEASE	Issues the MVS DEQ macro to release the checkpoint data set.
	HASPCKAP	Subtask that maintains a third copy of the checkpoint for use by applications.
HASPMSG	$MSGDISR	Called from the $SCAN facility to build WTO line(s).
HASPCOMM	HA$PCOMM	Handles all JES2 commands.
	CONSCHK	Checks if this a defined MCS console.
	CWTO	Writes to the operator.
	CWTOT	Writes to the operator.
	COFCVE	Converts halfword to EBCDIC.
	COFEDTR	Converts fullword to EBCDIC.
	COFRTC	Converts binary route codes for messages.

Figure 4-1 (Page 3 of 13). JES2 Directory Information

Object Module	Entry Point	Function
	COFJID	Obtains job id information for messages.
	$JCANR	Cancels or stops a job.
	CSCANDSP	Command processor command display routine.
HASPCNVT	HA$PCNVT	JCL conversion and authorization checking entry.
HASPCNVS	HOSCNVT	JES2 JCL conversion processor subtask; works in conjunction with the JES2 JCL conversion processor; opens and closes, and in some cases validates, the data sets used by the MVS JCL converter. The converter is entered from the subtask and returns to the subtask.
HASPSUBS	HA$PSUBS	JES2 generalized subtask; attached by HASPDYN, HA$PSUBS initializes a subtask and provides for generalized subtasking of calls to a specified routine. It also provides subtask ESTAE recovery.
HASPCON	$WTOR	Schedules MVS commands or messages for the operator.
	HASPWQUE	Queues a CMB.
	$WTOCR	Schedules MVS commands or messages for the operator using the caller's CMB.
	$GETCMBR	Gets a CMB from the free queue.
	$DOMR	Deletes operator requests.
	$FRECMBR	Frees a console message buffer (CMB).
	$HASPWTO	Issues SVC 34 and 35 instructions to pass commands and operator messages to MVS.
HASPTRAK	$PURGER	Frees space in the SYS1.HASPACE data set.
	$GBLDTGB	Builds TGBs and queues them to the HASPSPOL processor.
	$TRACKR	Obtains space in SYS1.HASPACE data set
	$TGMSET	Sets a bit indicating a bad track group in bad track group map.
	HASPVPRG	Frees all tracks acquired for a job, queue an SMF type 26 purge record and optionally a JMR for output and notify the operator that a job is purged.
HASPFSSM	FSMCONCT	Connects JES2 to the functional subsystem.
	FSMARR	Cross-memory services ESTAE routine (FSMORDER/FSMPOST to HASPFSSP).
	FSMORDER	Processes JES2 operator commands related to the functional subsystem.
	FSMPOST	Notifies the functional subsystem when a request completes.
	FSMGETQC	Obtains one or more quick cells from a predefined pool of cells.
	FSMFREQC	Returns one or more quick cells to their associated pool.
	FSMBLDQC	Builds quick cell pools and their associated extensions.
	FSMQCT	Sets up the quick cell control table.
	FSMSAVE	Handles save area and register linkage for JES2 FSI routines.
	FSMRETRN	Handles save area and register return linkage for JES2 FSI routines.
	FSMCATER	Issues catastrophic error messages and abends.
	FSMERROR	Common error routine for unsupported FSI routines.
	FSMRCRTN	SRB routine to reconnect the cross-memory environment.
	FSMCBIO	Reads and writes JES2 control blocks.
	FSMCBCK	Checks read and write completions and the EBCDIC identifiers of JES2 control blocks.
	FSMFSLNK	Accesses the control blocks required by any FSI services routine.
	FSMGETLK	Gets the MVS local and CMS locks.
	FSMFRELK	Frees the MVS local and CMS locks.
	FSMGTBLK	Acquires a storage cell from a free pool.
	FSMRTBLK	Returns a storage cell to the free pool.
HASPFSSP	HA$PFSSP	Processes functional subsystem requests as a JES2 processor.
	DYNFSS	Searches for/adds FSS address spaces
HASPDYN	$PCEDYN	Dynamically attaches/detaches PCEs.
	$PCEDYDC	Dynamically attaches/detaches PCEs for DCTs.
	$DTEDYNA	Attaches DTEs.
	$DTEDYND	Detaches DTEs.
	$DCTDYN	Finds and chains DCTs.
	$DCBDYN	Attaches or detaches a DCB and DEB for a specific DCT.
	$DESTDYN	Dynamically adds JES2 destinations ids.
	DDYNRDR	Initializes local reader DCT.
	DDYNPUN	Initializes local punch DCT.
	DDYNPRT	Initializes local printer DCT.
	DDYNOFF	Initializes offload DCT.
	DDYNOJT	Initializes offload job transmitter DCT.
	DDYNOST	Initializes offload SYSOUT receiver DCT.
	DDYNOJR	Initializes offload job receiver DCT.
	DDYNOSR	Initializes offload SYSOUT receiver DCT.
	DDYNLNE	Initialize remote/network line DCT.
	DDYNLGN	Initialize SNA LOGON DCT.

Figure 4-1 (Page 4 of 13). JES2 Directory Information

Object Module	Entry Point	Function
HASPJOS	$#BLD	JOE build routine; entered through the $#BLD macro instruction; builds a pair of JOEs (a work JOE, and a characteristics JOE).
	$#ADD	Job output element add routine; entered through the $#ADD macro instruction; copies the data represented by a work JOE prototype and a characteristics JOE prototype into two JOEs, which reside in the JOT.
	$#PUT	Put JOE routine; entered through the $#PUT macro instruction; returns a work JOE to the JOT class queue so that processing of the output work it represents can be continued at a later time.
	$#MOD	Modify JOE routine; entered through the $#MOD macro instruction; unchains a work JOE from its current queue and places it on a new queue based on its (possibly modified) SYSOUT class and route code.
	$#REM	Remove JOE routine; entered through the $#REM macro instruction; removes a work JOE (for which all output has been processed) from the JOT class queue and adds the JOE to the JOT free queue for reuse.
	$#CAN	Cancel JOE routine; entered through the $#CAN macro instruction; removes all JOEs for a job.
	$#GET	Get JOE routine; entered through the $#GET macro instruction; selects a work JOE from the JOT that most closely matches the setup, class, and routing characteristics of the requester as described by the device control table (DCT). A security check is also performed.
	GTSCREEN	Screens a JOE against a work selection list.
	$#POST	Posts specific processors to attempt selection of output.
	$#CHK	Reads/writes a spool output checkpoint record.
	$#ALCHK	Allocates a spool record for output checkpointing.
	$#NEWS	JESNEWS data set support.
	$#PDBCAN	Marks all non-held PDDBs (for a JOE) as non-printable.
	$#JOTBLD	Builds the JES2 job output table during a cold start
	$#JOTCHK	Verifies the JOT on a all-system warm start
	JOTVERIF	Verifies job output queues.
	JOTREBLD	JOT rebuild routine.
HASPMISC	HASPRESM	Checks the state of JES2 resources for shortages.
	HASPTIME	Monitors real time of job execution
	HASPGPRC	Priority ages jobs in the JES2 job queue at a specified time interval.
	HASPACCT	Writes SMF records and, if applicable, calls the IEFUJP user exit.
	HASPNACT	Converts the JES2 account number to the network account number, or vice versa.
HASPEVTL	HA$PEVTL	Trace table manager and creates a formatted log of TTEs.
	TRGETTB	$GETMAINs ECSA trace tables.
	TTABFMT	Invokes following formatting routines for trace table entries.
	TROUT000	Formats "trace events discarded" information
	TROUT001	Formats trace table entries
	TROUT002	Formats trace table entries
	TROUT003	Formats trace table entries
	TROUT004	Formats trace table entries
	TROUT005	Formats trace table entries
	TROUT006	Formats trace table entries
	TROUT007	Formats trace table entries
	TROUT008	Formats trace table entries
	TROUT013	Formats trace table entries
	TROUT020	Formats trace table entries
	TROUTFSI	Formats trace table entries
	TROUT017	Formats trace table entries
	TRCDUMP	Formats and prints lines in a standard dump format.
	TRCPUT	Adds a record to the current buffer.
HASPNPM	HASPNPMP	Network path manager: • Builds and processes network connection control records. • Maintains routing tables based on control information regarding connections and disconnections. • Promulgates status information to path managers at connected nodes to ensure that the picture of the network is consistent and accurate.
	HASPNDCN	Updates the NAT when a line is disconnected; determines whether a path should be completely disconnected or whether another should be made primary.
	HASPNMUP	Records the fact that a multi-access spool member is up and is part of the network.
	HASPNMDN	Records the fact that a multi-access spool member is down and is no longer part of the network.

Figure 4-1 (Page 5 of 13). JES2 Directory Information

Object Module	Entry Point	Function
	HASPNBUF	Processes a spool buffer containing network connection control records transmitted between shared spool members and updates the spool member's nodes attached table (NAT).
	HASPNSNR	Initial sign-on support subroutine: Builds a sign-on record and queues it for transmission.
HASPNET	HASPNJT	Network job transmitter: • Transmits an image of a job's JCL and instream data sets to another node for execution. • Responds to operator commands affecting job transmission. • Queues a job for purge if positive acknowledgement to an end of file was received from the associated job receiver, otherwise requeues the job for transmission. • Aborts a job when permission to transmit is denied.
	HASPNST	Network SYSOUT transmitter: • Transmits regular and spin SYSOUT data sets over NJE lines to one or more destination nodes. • Responds to operator commands affecting SYSOUT transmission. • Queues a job for purge when last work for the job is processed and positive acknowledgement to an end of file is received from an associated SYSOUT receiver, otherwise requeues the work for transmission. • Aborts a job when permission to transmit is denied.
	HASPNSR	Network SYSOUT receiver: • Receives a job's generated SYSOUT data sets from a SYSOUT transmitter on another NJE node. • Writes received data sets to spool. • Prepares received data sets for local output or transmission to other NJE nodes.
HASPPRPU	PRPUXSRV	Entry point for exit services routines.
	HASPPPI1	Print/punch processor entry point: • Acquires output work from the JOT using its DCT as a parameter list. • The work described by a JOE is written to a printer or punch in compliance with the user's data set descriptions. • When it completes output, it transfers the job to the $PURGE queue.
	PRPUT	Entry to PPPUT branched to as a result of a $PRPUT macro instruction. Provides a standard interface for use by exit routines.
	HASPIMAG	SYS1.IMAGELIB loader subtask entry point: • Is identified and attached as a subtask by module HASPINIT. • Loads forms control buffer (FCB) or universal character set (UCS) images from SYS.IMAGELIB for 3800 printers. • Interfaces with the SETPRT function without causing the main JES2 task to wait.
	PRTASUB	Print security authorization routine.
	PPDATE	Entry to service routine that formats the date.
	PPMSGDSP	Message display routine.
HASPHOPE	HA$PHOPE	Output processor: • Converts the output requirements for a job described by the peripheral data definition blocks (PDDBs) in the job's input/output table (IOT) to JOEs. • As each JOE is added to the JOT, it immediately becomes available for selection by a print/punch processor. • If any spin data sets are queued to $UNSPUNQ, they are converted to JOEs.
HASPRAS	$ESTAER	Establishes a $ESTAER environment.
	$ESTACAN	Cancels a $ESTAER environment.
	$ESTAREP	Replaces a $ESTAER environment.
	$DSTERR	Displays $HASP096 message.
	$IOERRTN	Displays $HASP094 message.
	$SDUMP	Issues an SVC dump.
	SDMPOPER	Sets up and issues WTOR SDUMP message.
	$STABEND	Abend services for subtasks.
HASPRDR	HA$PRDR	Input service processor and network job receiver: Reads jobs into the JES2 subsystem, spools JCL and input data sets, and queues the jobs for execution services.
	HASPRTRM	Terminates input service.
	HASPRCCS	JES2 control card scan subroutine: Processes JES2 control language statements.
	HASPRDDS	Scans DD* card.

Figure 4-1 (Page 6 of 13). JES2 Directory Information

Object Module	Entry Point	Function
	HASPRJCS	Job initialization subroutine: Constructs the basic job control blocks, scans the JOB statement, and initiates processing of the job.
	HASPRSCN	JOB card accounting field scan subroutine: Scans the accounting field of the JOB statement and sets the related job parameters.
HASPRTAM	HASPMLLM	Manages physical and logical teleprocessing lines, and performs error recovery and logging when needed.
	MSAFCHK	Obtains and initializes a SWEL, WAVE, and security token; issues a SAF call.
	LMRPCED	Provides the interface to $PCEDYN to attach remote device PCEs.
	HASPRBUF	Requeues buffers on $RJECHEQ for $EXTP routines.
	HASPROUT	Handles rerouted XMIT jobs through use of the $EXTP routines.
	HASPMCON	For multileaving terminals with consoles, provides immediate interface (through $EXTP routines) With the JES2 command processor and JES2 console output for all other terminals, provides console output spooling to be processed later as printed output.
	HASPMDCN	Remote console processor line disconnect: Handles disconnection of the line when RJE work station or NJE node. HASPMDCN sends a final RJE sign-off record.
HASPSCAN	$SCAN	Facility to set or display control block values based on initialization statements.
	$SCANB	Routine to back up the control block field about to be changed by a scan setting.
	$SCAND	Routine to permit a pre- or post-scan exit, and HASPSCAN, to add or replace the text resulting from a display request.
	$SCANCOM	Subroutine to find the first non-blank comment character.
	SCNDIAGT	Diagnostic message address table.
	SCNDGRTN	Builds diagnostic phrase that requires a dynamic variable.
	SCNDBRNG	Builds diagnostic message stating value is not within the binary range.
HASPSERV	SERVXSRV	Contains addresses of common service routines.
	SRVM630	Formats $HASP630 message.
	SRVMHOLD	Modifies hold for jobs or SYSOUT.
	SRVMCLAS	Modifies class for jobs.
	SRVMXEQN	Modifies the route code for jobs.
	SRVMSAF	Changes system affinity.
	SRVMROUT	Modifies route code of data sets.
	SRVPRSCN	Scan PRMODE operand and create PRMODE table.
	WSPBRS	Printer work selection burst routine.
	WSSERV	Work selection service routine.
	WSNXT	General work selection work routine.
	WSCOMP	General work selection comparison routine.
	WSFLG	General work selection flag routine.
	WSRNG	General work selection range routine.
	WSCLASS	Work selection class routine.
	WSVOL	Work selection volume routine.
	WSROUT	Work selection route code routine.
	WSSAF	Work selection system affinity routine.
	WSPRMD	Work selection PRMODE routine.
	WSJOEPRI	Work selection priority routine for SYSOUT.
	WSJOBPRI	Work selection job priority routine.
	WSLIM	Work selection limit routine.
	WSPUCS	Printer work selection UCS routine.
	WSPFLSH	Printer work selection flash routine.
	WSPFCB	Printer work selection FCB routine.
	WSPPFRMS	Printer/punch work selection forms routine.
	WSLASH	Work selection slash routine.
	SRVPRSCN	Scan PRMODE operand and create PRMODE table.
	HASPLIST	$L support.
	SUBRRT	Subtasks the $REROUTE routine (HASPSSRV).
HASPSSRV		JES2 subtask services module; provides data set fake open/close, performs message logging, validates jobs/SYSOUT, and authorizes network rerouting of jobs/SYSOUT.
	DSFOPEN	Data set fake open routine.
	DSFCLOSE	Closes a data set opened by DSFOPEN.
	PSAFSCAN	PDDB scan and SAF call routine.
	JOBVALM	Job validation and error message processing routine.
	NEWSCRE	Initialize JESNEWS PDDB, request create authorization and create the JESNEWS data set.
	RPDBSEC	System PDDB initialization processing routine.
	SYSOVFY	SYSOUT validation routine.

Figure 4-1 (Page 7 of 13). JES2 Directory Information

Object Module	Entry Point	Function
	$LOGMSG	Puts job-related messages into the SYSMSG data set and, optionally, performs a WTO.
	$REROUTE	Authorizes the final destination for a job or data set rerouted via $R or $TO command.
HASPSNA	HASPSNAA	Entry point for SNA $EXTP service routines.
	HASPSLOG	SNA active logon scan routine. Called by line manager scan routine.
	HASPSLNE	SNA active line scan routine. Called by line manager scan routine.
	HASPSIDL	SNA idle line scan routine. Called by line manager scan routine.
	HASPSUNT	Inactive SNA line and logon DCT scan routine. Called by line manager scan routine.
	HASPSACB	Logon DCT exit and ACB completion scan routine. Called by line manager scan routine.
	HASPSICE	ICE exit scan routine. Called by line manager scan routine.
	HASPSRAT	Searches remote attribute table for autologon remote terminals. Called by line manager scan routine.
	HASPSPRO	SNA buffer processing routine. Called by line manager scan routine in HASPRTAM.
	HASPVTAM	OPEN/CLOSE ACB routine. Attached at initialization to run as a subtask of the JES2 task and communicates with VTAM.
	SNASNET	$SN command exit routine. Called by HASPCOMM while processing the $SN command with the A= operand to initiate the NJE application-to-application session.
	APPLDYN	Find/build an APT for an APPL.
HASPSPOL	HA$PSPOL	HASPSPOL processor
	DADAVAIL	Computer total and allocated track groups.
	DADCKTGM	Checkpoints the TGM.
	DADDEB	Initializes direct access DEB extent information.
	DADKBYTE	Computes thousands of bytes of spool.
	DADREMVE	Removes extent from DAS.
	DADREMWQ	Removes DAS from work queue.
	DADSPACE	Log track group map space utilization.
	DADSPLST	Resets SPL control block.
	DADTGM	Initializes track group map for this extent.
	DADXTENT	Initializes extent information.
	HOSPOOL	Dynamic spool allocation subtask.
HASPSXIT		Contains all exits for $SCAN
		Set-related exits:
	PREAPPL	Pre-scan for the APPL statement
	PREAUTH	Pre-scan for the AUTH parameter on JOBCLASS statement
	PREFSSDF	Pre-scan for the FSSDEF statement
	PREJRNG	Pre-scan for the RANGE operand
	PRELIMIT	Pre-scan for the LIMIT and PLIMIT parameters
	PRELINE	Pre-scan for the LINEnnnn statement
	PRELOAD	Pre-scan for the LOADMOD statement
	PRELOADR	Pre-scan for the LOADMOD statement ROUTINES operand
	PRENODE	Pre-scan for the NODE statement
	PRENODES	Pre-scan for the STATUS parameter on the NODE statement
	PRERDEV	Pre-scan for the RnnnnDVX statement
	PRERMT	Pre-scan for the RMTnnnn statement
	PRERMTAL	Pre-scan for the AUTOLOG sub-operand on the RMTnnnn statement
	PRERMTST	Pre-scan for the STATUS sub-operand on the RMTnnnn statement
	PRERNG	Pre-scan for the RANGE sub-operand of output devices
	PRETDFID	Pre-scan for the ID sub-operand on the TRACEDEF statement
	PRETRCID	Pre-scan for the ID sub-operand of the TRACEDEF statement
	PSTAPPL	Post-scan for the APPL statement
	PSTBADTR	Post-scan for the BADTRACK statement
	PSTCAT	Post-scan for the TSUCLASS and STCCLASS statements
	PSTCHARS	Post-scan for the COMPACT CHARS sub-operand
	PSTCKPT	Post-scan for the CKPTDEF statement
	PSTCKPTN	Post-scan for CKPTN and NEWCKPTN operands on CKPTDEF statement
	PSTCNCHR	Post-scan for the CONCHAR on the CONDEF statement
	PSTCOMP	Post-scan for the COMPACT statement
	PSTCONCT	Post-scan for the CONNECT statement
	PSTDEST	Post-scan for the DESTID statement
	PSTEST	Post-scan for the TEST parameters
	PSTEXRTN	Post-scan for the ROUTINE sub-operand of EXITnnn statement
	PSTFEN	Post-scan for the FENCE parameter on the SPOOLDEF statement
	PSTFFSDF	Post-scan for the FSSDEF statement
	PSTHOLD	Post-scan for the HOLD operand

Figure 4-1 (Page 8 of 13). JES2 Directory Information

Object Module	Entry Point	Function
	PSTJRNG	Post-scan for the RANGE operand
	PSTLGNA	Post-scan for the $S, $P, and $E LOGONn statements
	PSTLIMIT	Post-scan for the DEVICE LIMIT/PLIM/RANGE sub-operands
	PSTLINE	Post-scan for the LINEnnnn statement
	PSTLINEA	Post-scan for the $S, $P, and $E LINEnnnn statements
	PSTLINST	Post-scan for the LINEnnnn sub-operand
	PSTLOAD	Post-scan for the LOAD statement
	PSTLSPIN	Post-scan for the SPIN parameter on the TRACEDEF statement
	PSTMDRC	Post-scan for the MOD=ROUTECDE sub-operand
	PSTMDSAF	Post-scan for the MOD=SYSAFF statement
	PSTNODE	Post-scan for the NODEnnnn statement
	PSTNOTAB	Post-scan for the TRACEDEF statement
	PSTNETAC	Post-scan for the NETACCT statement
	PSTNRT	Post-scan for the NET RECEIVER/TRANSMITTER counts
	PSTPRMD	Post-scan for the PRMODE sub-operand of output devices
	PSTPRTY	Post-scan for the PRTYRATE statement sub-operand
	PSTRDR	Post-scan for the RDRnn statement
	PSTRECV	Post-scan for the RECVOPTS statement
	PSTRMT	Post-scan for the RMTnnnn statement
	PSTRMTAL	Post-scan for the AUTOLOG sub-operand on the RMTnnnn statement
	PSTRMTA	Post-scan for the $S and $P RMTnnnn statements
	PSTRC	Post-scan for the ROUTECDE statement
	PSTTGBE	Post-scan for the TGBPERVL parameter on the SPOOLDEF statement
	PSTSELCT	Post-scan for the SELECT sub-operand for remote devices
	PSTSAFF	Post-scan for the OFFn.JT and OFFn.JR SYSAFF sub-operand
	PSTSPL	Post-scan for the SPOOLNUM operand on the SPOOLDEF statement
	PSTTRANS	Post-scan for POSTing of offload transmitters
	PSTVOL	Post-scan for the VOLUME sub-operand on the DEVICE statement
		Display-related exits:
	PREDAUTH	Pre-scan in support of the AUTHORITY operand
	PREDBADT	Pre-scan in support of the BADTRACK parameter
	PREDCHAR	Pre-scan in support of the CHARS operand
	PREDCOMP	Pre-scan in support of the COMPACT parameter
	PREDCNCT	Pre-scan in support of the CONNECT parameter
	PREDEST	Pre-scan in support of the DESTID DEST parameter
	PREDEXIT	Pre-scan in support of the EXIT parameter
	PREDSBEX	Pre-scan in support of the EXIT parameter
	PREDMDRC	Pre-scan in support of the MOD=ROUTECDE sub-operand
	PREDNET	Pre-scan in support of the NETACCT parameter
	PREDPSWD	Pre-scan in support of the PASSWORD operands
	PREDPRMD	Pre-scan in support of the PRMODE parameter
	PREDRNG	Pre-scan in support of the RANGE VECTOR operands
	PREDRC	Pre-scan in support of the ROUTECDE parameter
	PREDSESN	Pre-scan in support of the SESSIONS operand
	PREDSTAT	Pre-scan in support of the STATUS operand
	PREDSLCT	Pre-scan in support of the SELECT parameter
	PREDSAF	Pre-scan in support of the SYSAFF parameter
	PREDWS	Pre-scan in support of the WS parameter
	PREWS	Pre-scan in support of the WS sub-operand for output devices
	PREZEROF	Pre-scan to zero the upper and lower RANGE/PLIM/LIMIT DCT fields.
HASPSTAM	HASPXFRA	Entry point for spool offload $EXTP service routines.
	HASPXFRM	Manages spool offload I/O processing, including dumper and loader start and drain and buffer completions.
	HASPOFF	Spool offload subtask to handle allocation, open, close and deallocation of the offload data set.
	DYNALERR	Formats DYNALLOC error message
HASPTABS	HASPTABR	Accesses the HASP and installation table pairs.
	DPRTWS	Local printer default WS list.
	DPUNWS	Local/remote punch and remote printer default WS list.
	DFOFFJWS	Offload JT and JR default WS list.
	DFOFFSWS	Offload ST and SR default WS list.
	TIDTABD	Indicates end of HASP trace ID table.
	$BITSON	Bits on in a byte.
	$OUTTAB	Default priority table for lines and pages.

Figure 4-1 (Page 9 of 13). JES2 Directory Information

Object Module	Entry Point	Function
	$TIMETAB	Default priority table for estimated elapsed time.
	$QINDEX	Job class queue header index table.
	$#INDEX	SYSOUT class queue header index table.
HASPSTAB	HASPOPTT	OPTIONS $SCAN table
	HASPMPST	MAIN parameter statement $SCAN table
	HASPAPLT	APPL parameter statement sub-operand $SCAN table
	HASPBADT	BADTRACK parameter statement sub-operand $SCAN table
	HASPBUFT	BUFDEF parameter statement sub-operand $SCAN table
	HASPSTCT	STC class attribute statements sub-operand $SCAN table
	HASPTSUT	TSU class attribute statements sub-operand $SCAN table
	HASPCATT	JOBCLASS attributes statements sub-operand $SCAN table
	HASPCKTT	CKPTDEF parameter statement sub-operand $SCAN table
	HASPKPNT	CKPTDEF CKPTn= parameter statement sub-operand $SCAN table
	HASPEKNT	CKPTDEF NEWCKPTn= parameter statement sub-operand $SCAN table
	HASPCOMT	COMPACT parameter statement sub-operand $SCAN table
	HASPCNDT	CONDEF parameter statement sub-operand $SCAN table
	HASPCONT	CONNECT parameter statement sub-operand $SCAN table
	HASPDEST	DESTID parameter statement sub-operand $SCAN table
	HASPELCT	ESTLNCT parameter statement sub-operand $SCAN table
	HASPEBYT	ESTBYTE parameter statement sub-operand $SCAN table
	HASPEPGT	ESTPAGE parameter statement sub-operand $SCAN table
	HASPEPNT	ESTPUN parameter statement sub-operand $SCAN table
	HASPETMT	ESTIME parameter statement sub-operand $SCAN table
	HASPXITT	EXITnnn parameter statement sub-operand $SCAN table
	HASPFSST	FSSDEF parameter statement sub-operand $SCAN table
	HASPPITT	Innnnn parameter statement sub-operand $SCAN table
	HASPINRT	INTRDR parameter statement sub-operand $SCAN table
	HASPJOBT	JOBDEF parameter statement sub-operand $SCAN table
	HASPJPYT	JOBPRTY parameter statement sub-operand $SCAN table
	HASPLNET	LINEnnnn parameter statement sub-operand $SCAN table
	HASPLODT	LOADMOD parameter statement sub-operand $SCAN table
	HASPLOGT	LOGONn parameter statement sub-operand $SCAN table
	HASPMAST	MASDEF parameter statement sub-operand $SCAN table
	HASPNJET	NJEDEF parameter statement sub-operand $SCAN table
	HASPNODT	NODEn parameter statement sub-operand $SCAN table
	HASPNETT	NETACCT parameter statement sub-operand $SCAN table
	HASPOFLT	OFFLOADn parameter statement sub-operand $SCAN table
	HASPOFFT	OFFn.XX parameter statement sub-operand $SCAN table
	HASPOJRT	OFFn.JR parameter statement sub-operand $SCAN table
	HASPOJTT	OFFn.JT parameter statement sub-operand $SCAN table
	HASPOSRT	OFFn.SR parameter statement sub-operand $SCAN table
	HASPOSTT	OFFn.ST parameter statement sub-operand $SCAN table
	HASPOJMT	OFFn.JR MOD= parameter statement sub-operand $SCAN table
	HASPOSMT	OFFn.SR MOD= parameter statement sub-operand $SCAN table
	HASPOUTT	OUTDEF parameter statement sub-operand $SCAN table
	HASPOPYT	OUTPRTY parameter statement sub-operand $SCAN table
	HASPPART	INITDEF parameter statement sub-operand $SCAN table
	HASPPCDT	PCEDEF parameter statement sub-operand $SCAN table
	HASPOPDT	OPTSDEF parameter statement sub-operand $SCAN table
	HASPPTDT	PRINTDEF parameter statement sub-operand $SCAN table
	HASPPRTT	PRTnn parameter statement sub-operand $SCAN table
	HASPPUNT	PUNnn parameter statement sub-operand $SCAN table
	VECTPPXX	VECTOR device statement sub-operand $SCAN table
	HASPRDVT	RnnnnDVx parameter statement sub-operand $SCAN table
	HASPPUDT	PUNCHDEF parameter statement sub-operand $SCAN table
	HASPRPRT	RnnnnPRx parameter statement sub-operand $SCAN table
	HASPRPUT	RnnnnPUx parameter statement sub-operand $SCAN table
	HASPRRDT	RnnnnRDx parameter statement sub-operand $SCAN table
	HASPRDRT	RDRnn parameter statement sub-operand $SCAN table
	HASPRCVT	RECVOPTS parameter statement sub-operand $SCAN table
	HASPSBDT	SUBTDEF parameter statement sub-operand $SCAN table
	HASPRMTT	RMTnnnn parameter statement sub-operand $SCAN table
	HASPSMFT	SMFDEF parameter statement sub-operand $SCAN table
	HASPSPDT	SPOOLDEF parameter statement sub-operand $SCAN table

Figure 4-1 (Page 10 of 13). JES2 Directory Information

Object Module	Entry Point	Function
	HASPSCTT	OUTCLASS parameter statement sub-operand $SCAN table
	HASPTPDT	TPDEF parameter statement sub-operand $SCAN table
	HASPTSTT	TESTDEF parameter statement sub-operand $SCAN table
	HASPTRCT	TRACEDEF parameter statement sub-operand $SCAN table
	HASPTGLT	TRACEDEF LOG subscan sub-operand $SCAN table
	HASPSTAT	TRACEDEF STATUS subscan sub-operand $SCAN table
	HASPTRIT	TRACEn parameter statement sub-operand $SCAN table
	HASPSSIT	SSI parameter statement sub-operand $SCAN table
HASPTERM	$ABEND	JES2 ESTAE exit.
	HASPTRCA	Entry for $ERRTRCA in HCT
	$HEXIT	Entry to normal recovery/termination for clean-up processing.
	HEXTINIT	Termination processing for unsuccessful initialization.
	FSJBHASB	Frees the SJBs and HASB for the JES2 address space.
	EX26ESTA	ESTAE routine for exit 26.
	ERMODULE	Subroutine to locate an error load module.
	ERMODMAP	Subroutine to locate an error entry point.
	ERBASOFF	Subroutine to format the assembly base and offset.
	ABNDRATE	Subroutine to check for a RECVOPTS error rate excession.
HASPWARM	HA$PWARM	Warm start processor; provides JES2 warm start function.
HASPXEQ	HASPEXEC	JES2 execution processor; supports requests to spin or hold subsystem data sets and requests for job selection, job termination, and job requeuing.
	HASPTIME	JES2 time excession processor; monitors a job's real time in execution and issues WTOs to the operator when a job exceeds its estimated real execution time.
HASPSTAC	HA$PSTAC	STATUS/CANCEL processor; removes SJBs from the STATUS/CANCEL queue, scans for jobs to process, performs security processing for jobs, and records processing information in SJBs.
HASPPSO	HA$PPSO	JES2 process-SYSOUT processor; processes requests for subsystem data sets from conversational terminal systems and from external writers.
	HASPJDR	Job disposition processor; provides an interface between HASPCOMM and HASPPSO for disposition of held data sets.
HASPINIT		JES2 initialization; this series of routines accepts initialization options, reads initialization parameters, examines system control blocks, and (from these input sources) sets up the control blocks required for JES2 processing.
HASPIRA	!RA	Entry to initialization administrator.
	NBFBUILD	Subroutine to generate buffer pools.
	NSSSM	Subroutine to load and validate HASPSSSM load modules.
	IRMLIST	List of MIT addresses and MODMAP entry offsets for all initialization assembly modules.
	NQUERY	Subroutine for WTORs.
	NRDCTINT	Subroutine to initialize remote device DCTs.
	$RWL	Remote device lookup table.
	$RWLRDRS	Remote readers.
	$RWLPRTS	Remote printers.
	$RWLPUNS	Remote punches.
	ISSIMSG	HASP430 initialization message.
	NGTMNMSG	HASP432 initialization message.
	NRTEMSG	HASP444 initialization message.
	IHOTSMSG	HASP490 initialization message.
	NEXITEM	HASP864 initialization message.
HASPIRMA	IROPTS	Process JES2 initialization options.
	IRMODCHK	Verifies that this version of JES2 is compatible with the installed level of MVS; verifies the level of JES2 modules.
	IRSSI	Finds this subsystem's CVT (SSCT).
	IRSETUP	Pre-PARMLIB storage allocations.
	IRDCTDCB	DCT,DCB,DEB formatting routine.
	IRDCTCP	Completes DCT initialization.
	IRCSA	Creates console message buffers.
	IRPCE	Builds non-dynamic PCEs.
	IRURDEV	Unit record device allocation.
	IRMVS	Identifies JES2 post-exit routines to MVS.
	IRFINAL	Internal reader build.
HASPIRPL	IRPL	Initialization routine to process initialization parameter statements.
	IRPOSTPL	Post-process parameters set in IRPL.
	HPARMDCB	IRPL data area; HASPPARM DCB.

Figure 4-1 (Page 11 of 13). JES2 Directory Information

Object Module	Entry Point	Function
	HLISTDCB	IRPL data area; HASPLIST DCB.
	NPLLOG	Displays PARMLIB initialization statements to the console.
HASPIRDA	IRDA	Performs DASD initialization.
HASPIRRE	IRNJE	Completes final NJE initialization.
	IRRJE	Completes final RJE initialization.
HASPAM	HASPAMI	Provides support routines for use of GET/PUT/ENDREQ data management macros with JES2 SYSIN, SYSOUT data sets.
	HAMNULL	Provides for immediate return to user if any macros are invoked before the data set is opened or after it is closed.
	HINTRDR	Provides support for internal reader processing.
	HERNOEOD	Return point if EOD is encountered.
	HAMAVT	Appendage vector table used for I/O completion exits of HAM-generated I/O.
	HCNVFDAD	Routine to convert the track address from JES2 MTTR form to MVS MBBCCHHR form.
HASCDSAL	SSIALOC	SSI function routine for subsystem data set allocation.
	SSIALUNA	SSI function routine for subsystem data set deallocation
	$PDBBLD	Build and initialize a PDDB.
	HALFDSNR	Find data set number.
	HALJMERG	Merge PDDB values into JFCB.
	HALOCRP	Merge OCR values into PDDB.
	HALOPDBI	complete initialization of a PDDB.
	HALSSALP	Merge SSOB values into a PDDB.
	HALUNAL	Subsystem data set unallocation service routine.
	HAOUTSCN	Scan output DD reference.
	HIOTSPIN	Create a common storage spin IOT and chain it to $HCCT so that HASPXEQ can process it.
HASCDSOC	SSIDAOPN	Calls DSOPEN to open a subsystem data set or internal reader.
	SSIDACLO	Closes a data set.
	SSIDARES	Restarts subsystem data sets.
	SSIDACKP	Checkpoints subsystem data sets.
	DSOPEN	Opens a subsystem data set (SYSIN, SYSOUT, PSO, INTRDR).
	HFOPSUB	"Fake opens" special subsystem data sets.
	HIORALE	Ensures that this data space is permitted to access the internal reader PBUF data space.
	SSVCLSC	Closes all data sets opened for the converter subtask.
	SSVOPNC	Opens all data sets for the converter subtask.
HASCJBST	SSIJSEL	SSI job select routine.
	SSIJTERM	SSI job termination routine
	SSIRRREQ	SSI request job id routine.
	SSIRRRET	SSI return job id routine.
	SSIRQRNQ	SSI requeue job routine.
	HJE000	Job termination.
	HFJOBLOG	Place header and messages into job log.
	HJSMAKSL	Create a PDDB slot for a data set.
	JBFOUND	Set up job for execution.
	JBSELECT	Select job.
	JOBSTATS	Update JCT statistics.
	JSREOPN	Reposition system data sets.
	JSOPSSDS	Open subsystem data sets.
	MRGSWBS	Merge SWB into PDDB.
HASCJBTR	SSIENEOM	SSI function routine for end-of-task processing.
	SSIETEOT	SSI function routine for end-of-memory processing.
	EOTFDCON	Issue FSIREQ disconnect request.
HASCLINK	$CRETRN	JES2 return linkage.
	SSSMTABI	JES2 subsystem initialization table.
	$CRETSAV	Return save area routine.
	$CSAVE	JES2 save linkage routine.
	$GETHP	High private cell pooling routine.
	$HGFMAIN	Branch-entry GETMAIN/FREEMAIN routine
	$MLTFBUF	Free multiple buffer routine.
	$SSIBEGN	Subsystem function initialization.
	$SSIEND	Subsystem function termination.
	$FRECEL	Returns a CSA cell to the CSA cell pool.
	$GETCEL	Gets a CSA cell from the CSA cell pool.
	GETTRE	Gets a TCB recovery element.

Figure 4-1 (Page 12 of 13). JES2 Directory Information

Object Module	Entry Point	Function
	RECOVERY	Generalized recovery routine for subsystem functions.
	SSIFINE	SSI function cleanup routine.
	SSISETUP	Setup routine for SSI functions.
HASCSIRQ	SSISOUT	Process SYSOUT routine
	SSICSTAT	SSI function routine for status.
	SSICSCAN	SSI function routine for cancel.
	SSIWTA	SSI function routine for WTOs.
	SSICMD	SSI function routine for commands.
	SSIUSUSE	Destination verification.
	SSIFSCNT	FSI connect/disconnect routine.
	TSCNVJB	Convert job id to binary job number.
	TSQUEUE	Queue SJB to HASP work queue.
	USERDEST	Verify destination and convert to binary route code
	USERSUB	User/subtask exit effector.
HASCSRDS	$CBIO	Control block I/O routine.
	$FNDRIOT	Locate reusable spin IOT.
	$IOTBLD	Create and chain an IOT.
	$PDBFIND	Locate a PDDB.
	$SDBFREE	Free an SDB.
	$SDBINIT	Initialize an SDB.
	$VERIFY	Verify a control block.
	DSNCMP	Compress a SYSIN/SYSOUT data set name.
	DSNVFY	Verify a SYSIN/SYSOUT data set name.
	HALCLASS	Determine if SYSOUT class is in hold status.
	HALDEST	Determine destination of a held data set.
	HCBCK	Checkpoint all control blocks pointed to from SJB.
	HCBFM	Free a JES2 control block.
	HCBGM	Get a JES2 control block.
	HFCLSUB	Fake close a data set.
	HFCLTRNC	Truncate an unprotected buffer.
	HJSRETAB	Rebuild SDB track allocation.
	HONEWOUT	Open a new output data set.
	HOOLDINP	Reopen an old input data set.
	HOOLDOUT	Reopen an old output data set.
	MTTRVAL	Validate a track address.
HASCSRIC	$$POST	Cross-memory posts JES2 to post a PCE or resource
	$RACROUT	Issues a SAF call to RACROUTE services.
	$STRAK	User environment spool space allocation.
	$SVJLOK	Get the job communication queues lock.
	$SVJUNLK	Release the job communication queues lock.
	$TRACER	Obtain a JES2 trace table entry.
	$TRAREL	Release a JES2 trace table.
	$VFLI	Simulate a VFL instruction.
	$XMPOST	Cross-memory post services routine.
	HASJFREQ	Initialize SJF parameter list to resolve OUTPUT JCL references.
	HKYMERGE	Merge OUTPUT JCL keywords into PDDB.
	HOSWB	Open processing for OUTPUT card SWBS.
	HSJFLSP	Free SJF storage.
	PRTAUTH	Print security authorization routine.
	PPSOSJB	Purge the PSO chained from the SJB.
	PSQUEUE	Queue a PSO request to the JES2 main task.
	SSVXDEF	Propagate defined exit point bit map in MIT to XIT.
	TQUEBTG	Check for successful I/O.
	TSETLOCK	Use the MVS SETLOCK macro to get the local and CMS lock.
	TSFRELOK	Release the local and CMS lock.
	FINDLMT	Open a new output data set.
	TRCKDATA	Called when trace data does not fit in a trace table. TRCKDATA issues message $HASP381 and turns off the trace id.
HASCSRJB	$JBIDBLD	Creates an 8-character EBCDIC job id.
	$SJBFIND	Finds the LOJ, first, last or SSIB SJB.
	$SJBLOCK	Gets the SJB lock for an SJB.
	$SJBRQ	Requeues the SJB to the specified queue.
	$SJBUNLK	Releases the SJB lock.
	HETSOUT	Saves interrupt status.

Figure 4-1 (Page 13 of 13). JES2 Directory Information

Object Module	Entry Point	Function
	SJBFREE	Dechains and frees the storage of an SJB, SJXB, and SJF work area.
	SJBINIT	Gets storage for and initializes the SJB, SJBX, and SJF work area.
	TSHABDQ	Removes an TSO-queued SJB from the chain.
	TSUABQS	Scans for TSO-queued SJB on a user abend.
HASCDSS	DSPSERV	Data space services routing routine.
	DSCREATE	Data space create routine.
	DSDELETE	Data space delete routine.
	DSSRB	Data space SRB routine.
	DSRMTR	SRB resource termination routine.
	DSFRR	SRB FRR recovery routine.
	DSGET	Data space ASCB/TCB get routine.
	DSDSP	Issues the DSPSERV macro to create or delete a data space.
	DSALE	Creates or deletes access to a data space.
	DSESTAE	Data space service ESTAE routine.
HASPBLKS	HA$PBLKS	Control block formatting routine.

Chapter 5. Diagnostic Aids

Diagnosis is the task of describing a programming problem, identifying the source of the problem, and solving the problem or reporting it to IBM.

Introduction

This section describes a general approach and specific resources and techniques that can be used to diagnose problems with JES2. The introduction provides a general methodology for problem solving in the JES2 address space. The next section tells you what documentation you need when you report you problem. The following section describes some of the structural features of JES2 from a diagnostic perspective. Familiarity with this structure can be helpful when diagnosing problems. Subsequent sections offer specific topics that address specific diagnostic tools.

Diagnosis can be a difficult task. But you increase the difficulty if you do not diagnose in a disciplined way. Discipline cannot replace experience or intuition, but it can structure your diagnosis effort and save you valuable time.

The diagnostic procedure can be divided into the following basic tasks:

1. Obtain an exact description of the perceived problem condition and what led up to it
2. Identify an external symptom from this context
3. Gather pertinent information the system has provided
4. Analyze the information
5. Identify further needed information
6. Use available diagnostic tools to gather that information
7. Identify the basic functional area and then the module in control when the problem occurred
8. Pinpoint the problem to a module
9. Report the problem to your IBM Support Center and supply any needed information.

Describing the Problem Condition

This is often the most important step in the process of diagnosing problems. Because it is the first step, a wrong move at this point can waste hours of effort. To correctly identify a problem, you need an exact sense of what the conditions were at the time of the problem and what events led up to this condition. To do this, it is best to get a description of the problem as it was perceived by an eyewitness. You will want a description that provides a context from which to start, such as:

> "System is looping; can't get in from the console."
> "Console locked out."
> "The job won't cancel."
> "The command does not work."
> "Bad output."
> "Nothing is running."

The list is endless, of course. But your objective is to fit one (or more) of these description to one of the external symptoms described below. To do this, ask questions:

- When did you notice the problem?
- Has the system told you anything? Messages? Codes? Dumps?
- Did you notice anything unusual before the problem occurred?
- Did you do anything unusual before the problem occurred?
- What have you done to correct the condition?

Gather this kind of information and produce an exact description of the condition and the events surrounding it. It can be used as a convenient checklist as you do the next task.

Identifying the External Symptom of your Problem

The objective of this task is to match your description to one of the following externals symptoms:

- Enabled wait -- the system is not executing any work and when it takes interrupts, nothing happens. Something appears to be stuck.
- Disabled wait -- the system stops with a disabled PSW that has the wait bit on. This can be either an explicit and intentional disabled wait or a situation that occurs because the PSW area has been overlaid.
- Disabled loop -- this is normally a small (fewer than 50 instructions) loop in disabled code.
- Enabled loop -- this is normally a large loop in enabled code (but may include disabled portions -- loops as a result of interrupts).
- Program check -- the program is automatically cancelled by the system, usually because of improper specification or incorrect use of instructions or data in the program. The program check message gives the location of the failing operation and the condition code. If the JCL includes a SYSABEND, SYSMDUMP, or SYSUDUMP DD statement, a dump will be issued.
- ABEND -- the system has encountered an error it cannot recover from. Message $HASP088 gives you information about the abend.
- An error message -- the system has encountered an error and issued a message about the error. $HASP095 is issued when catastrophic errors occur. $HASP096 is issued when disastrous errors occur.
- An incorrect or unexpected command response -- the system is not doing what you have requested
- Incorrect output -- a job or jobs are not producing required SYSOUT
- A job failure -- a job does not execute successfully
- An RJE or NJE failure -- a break occurs in communication between a workstation and its host or between nodes in a network
- A maintenance-related problem -- the system experiences an incompatibility between different release levels or fixes

Gathering Pertinent Information about your Problem

To diagnose your problem, you need information. Generally, for the above problems, you do the following:

Abend, Loop, or Wait: To analyze an abend, check the JES2 section of *MVS System Codes* for an explanation of the abend code. Then, if the system did not produce a dump, obtain a dump using the MVS DUMP command and specify SDATA=RGN. Consult *MVS System Commands* for further information, and see "Dumps and the $SDUMP Command" later in this chapter for more about dumps.

Incorrect Output: Incorrect output can be characterized as missing records, duplicate records, or invalid data with sequence errors, incorrect values, format errors, or meaningless data. Remember that if a program or job has apparently executed successfully, incorrect results will be noticed only when the output is actually looked at or used at some future time.

Incorrect or Unexpected Command Response: Check the syntax of the command you entered and ensure that it is correct, and identify on what kind of device it was entered and under what conditions (such as, a local or remote console, or TSO terminal).

A Job-oriented Failure: Check the failing job's JCL. The system tells you right away about a JCL error, but your JCL may not be incorrect, only inappropriate for how you want your job to be processed. Also check the control statements you used. Use the JCL manual.

Incorrect or Unexpected Message: Check to be sure that it is a JES2 message. All JES2 messages are preceded by the prefix $HASP (assuming that the dollar sign ($) has been left as the default console character defined by JES2). Consult *JES2 Messages* for the modules that detect, contain, and issue the message. Given the processing that you believe was going on when the message was issued, is it probable that those modules were executing. If not, something went wrong with the flow of control; that is, unexpected calls or branches were made.

A Remote or Networking Problem: Networking-related initialization statements are often the problem. (See "Remote or Networking Problems" later in this chapter for more information about this topic.) Check these statements, look at the console sheet, and SNA/VTAM messages that the system issued.

Maintenance-related Problem: Understand SMP messages; assembly listings should show the errors. Keep track of program temporary fixes (PTFs), which provide source updates and resultant object modules. For more information about PTFs, see "Problem Reporting" later in this chapter.

Information the System Provides

The system provides three kinds of resources to diagnose problems:

- Input information you can check to ensure that the system is operating with the data it requires. This includes:
  - Assembled code, including parameters passed or received, data areas, tables, save areas
  - Initialization statements and parameters
  - JCL statements and control statements
  - Commands.
- Status information, which includes:
  - Messages
  - Codes
  - Logs
  - Traces.
- Requested information. This includes the following diagnostic facilities:
  - Dump Facility
  - IPCS Exit
  - SLIP Command
  - DEBUG Facility
  - $TRACE Facility
  - Patching Facility.

Messages

JES2 generates many messages to inform the operator and the user about system conditions. They are documented in *MVS Message Library: JES2 Messages*. There are three kinds of messages:

Informational messages tell you about system conditions that require no action on your part.

Action messages tell you that you must perform some action in order for processing to continue.

Error messages tell you that an error has occurred during processing. Error codes on message $HASP095 and register contents and PSW content on message $HASP088 can tell you a great deal about your problem. These messages and their associated codes are documented in *JES2 Messages*

The message "explanation," "system action," and "response" tell why a message was issued and the appropriate action to take. Some messages also direct you to the "Problem Determination" tables at the end of *JES2 Messages* These tables tell you what you must do before contacting IBM for support (also see "Problem Reporting" later in this chapter).

Codes

The system provides system codes that can help you determine the nature and cause of a problem. All system codes are documented in *MVS/Message Library: System Codes*. There are several types of codes:

- A system completion code is a three-digit hexadecimal number issued by the system when a module issues an ABEND macro instruction. This instruction causes a task or an address space to abnormally terminate. The code tells you why the task or address space terminated. You can find it in field TCBCMP of the terminated TCB. Codes for subtask terminations are also located in the ECB specified when the subtask was attached.
- A wait state code signals that the system has entered a wait state. The code is found in the PSW.

The only system codes that JES2 issues are 02C and 02D. 02C indicates a problem with a functional subsystem address space, and is accompanied by one of several messages (see *System Codes*). HASPFSSM detects the problem and issues the code or HASPWARM detects the problem and causes CALLRTM to issue the code. 02D is issued for a $PJES2,ABEND command.

JES2 issues many abend codes. They are issued with message HASP095 or HASP088. See the JES2 section of *MVS Messages* for these codes and their explanations.

Logs

Information is logged in the JES2 message log, the JES2 job log, the system log, and SYS1.LOGREC. Note that if something goes wrong involving a JES2 device, LOGREC will not show it.

Identifying Further Needed Information

If the information you have collected does not give enough to isolate your problem, the next step is to gather further information by using available diagnostic aids that MVS and JES2 provide. "JES2 Diagnostic Tools" later in this chapter, provides information about these aids.

Isolating the Functional Area and the Module

At any one of the points you have reached so far, you may have identified the module during whose processing the problem occurred. This may have been indicated by a dump or message. If not, and if all you have is a general sense of what function the system was performing when the problem occurred, a knowledge of JES2 processing is vital. This manual is an important resource for gaining that knowledge. Chapter 1 contains an overview of the structure of JES2. Figure 1-7 maps the structure against the functions of JES2. Using that figure as a starting point, you can easily match modules to functions as you read Chapters 2 and 3. Chapter 4 is a directory of entry points in JES2. It lists all object modules and the names of entry points within the object modules and provides brief descriptions of the functions of the entry points.

Problem Reporting

When you have a problem you cannot diagnose or fix, you need to report it to the IBM Support Center. If you have tried to diagnose your problem and produced documentation about the problem, IBM can provide maximum and efficient support.

General Checklist

Provide IBM support with:

- Your correct name and phone number
- FMID and release-related information about your installation. **Be sure that your MVS maintenance level is the same as your JES2 maintenance level.**
- Required documentation

  System log -- showing time before and after the failure

  Console log -- all messages and commands

  JES2 initialization parameters

  A dump of JES2 -- for all abends and most other failures; include the JES2 address space, the related user address space (for TSO), and for storage problems, CSA.

  A current assembly listing of the appropriate module or user exit

  The correct PTF level or PUT tape level of JES2 and MVS

  JCL and control statements for the failing job if the failure is job-oriented.

Specific Documentation Requirements

- Abends/waits/loops -- provide a dump (see "Introduction" at the beginning of this chapter for recommendations about dumps). **Note: When you report information in your dump, use sequence numbers, not displacements. It is also important to provide the name of the caller of the failing module. Problems can often originate in the caller.**
- Incorrect output -- provide details about the device type, and examples of actual versus expected output
- Command response incorrect -- provide exact format of the failing command, and where entered, that is, local or remote console, TSO, etc., and the console log
- Job-oriented failure -- provide failing job's JCL and any JES2 control statements
- Unexpected or incorrect message -- provide the exact format and content of the message, and the command if the message was issued in response to a command
- RJE or NJE failures -- provide exact copy of JES2 parameters, and your level of VTAM or TCAM if this is a SNA-related failure. Provide a dump, if you encountered an abend or wait. The console log is mandatory; traces may be required. The current assembly listings are also needed.
- Maintenance-related problems -- provide a current CDS listing, and the exact SMP error messages, if any. Also provide assembly listing showing errors.

- APAR fixes

  Source updates
  No superzaps
  Include "APARNUM" source statement
  Include only "real" prerequisites.

- PTFs -- provide source updates plus resultant object modules. PTFs are created monthly for each FMID. Quarterly level set PTFs and PE buckets supersede all monthly PTFs since the last level set. PTF charts are available to check maintenance level, verification of FMID, and are in PE buckets and PSP files.

Note that statement numbers or offsets are useless unless related to a label or sequence number in the current assembly listing.

Specific Documentation for Failures in Certain Modules

Problems involving HASPRTAM

- A trace of line activity. Acceptable data includes MVS CCWTRACE, 3705 TRACE, GTF TRACE macros. (See "Diagnostic Aids for Remotes and Networking Lines" in "Introduction" for more information.
- Console sheet from the remote terminal if there is one

Problems involving HASPXEQ or HASPSSSM

- JCL from the user program.

Problems involving the multi-access spool environment

- Turn on JES2 debugging by using the $T Debug=Yes command.
- Initialization parameters for all members in the MAS configuration.

Note: See *JES2 Initialization and Tuning* for performance considerations.

Problems involving HASPPRPU or HASPRDR

- JCL for all the jobs involved.
- System output from the failing jobs.

Note that if you have user modifications, a listing of each modified module directly or indirectly involved in the failure is needed.

The Structure and Processing of JES2

This section provides more detail from a diagnostic perspective about the structure of JES2.

The Task Structure of JES2

For diagnostic purposes, it is important to remember that most JES2 modules execute under the main task, except modules loaded in the link pack area or common storage, which execute under the user task, and modules attached as subtasks by modules that execute under the main task. The JES2 main task provides the basic functions of reading and spooling job input, converting JCL, selecting jobs from the JES2 job queue for MVS to execute, spooling and writing job output, and purging jobs. The modules that perform these functions are called processors and are represented by processor control elements (PCEs). JES2 attaches nine subtasks. See Figure 1-7 for more detail about the subtasks.

Dispatching

PCEs can be on a variety of queues (see Chapter 1) and are dispatched by the JES2 dispatcher. The JES2 dispatcher allocates time to the JES2 main task processors. When a processor is eligible for dispatching, its PCE is on a dispatcher queue called the $READY queue. When a processor is waiting on an event, it is ineligible for dispatching. If the processor is waiting for a resource, its PCE is chained to the designated resource queue; if the processor is waiting for a specific event, its PCE is queued to itself via a specific event wait field, "PCEEWF", in the PCE. See the description of the $WAIT macro in *JES2 Customization* for a list of events and resources a PCE can be waiting on. The currently active processor's PCE is at the top of the $READY queue and is addressed by the $CURPCE field in the HCT. Major queue and event control fields in the HCT and HCCT are shown in Figure 1-9.

$WAIT

A processor that is currently active remains so until it issues a $WAIT macro instruction, at which time the dispatcher is entered at entry point $WAIT (for a specific event), or $WAITR (for a general resource). The dispatcher continues to dispatch eligible processors from the $READY queue until the queue is empty. At this time control is passed to the dispatcher's resource posting routine, which looks for waiting PCEs that have been posted for events and are therefore eligible for dispatching. All eligible PCEs are moved to the $READY queue, and control passes back to the dispatcher.

$$POST

The JES2 dispatcher can be notified of work from within its own address space by the $POST macro. In addition, the dispatcher can be notified of work from other address spaces or from subtasks within its address space by the $$POST macro, which causes a HASPSSSM interface routine to cross-memory post the JES2 main task. In this case, the dispatcher post promulgation routine, which receives control when the resource posting routine runs out of work propagates event posts from the HCCT fields used by HASPSSSM interface routines to the HCT fields in the JES2 address space. Here the resource posting routine can pick them up and mark the corresponding processors eligible for dispatching. Control then returns to the dispatcher.

JES2 WAIT

When the JES2 dispatcher determines that there is no more work to be done, it issues an MVS WAIT macro, and waits to be posted for more work. When a $$POST macro is issued, the dispatcher post promulgation routine receives control and transfers the event notifications to the HCT, where they are picked up by the resource posting routine. The corresponding PCEs are transferred to the $READY queue.

JES2 Dispatcher Queue Structure

JES2 dispatcher resource queues are double headed and double threaded. Each PCE (as shown in Figure 5-1) has a chain field to the following PCE entry and one to the preceding entry on the queue. In the special case of the first PCE (referred to as PCE zero), the preceding entry field points to the queue headers, offset so that the queue header appears to be a PCE itself. The last PCE has a following entry field that points back to the queue header. The queue header itself is double, with pointers to the first and last PCEs in the chain. An empty queue has both queue header fields pointing to itself, offset to appear as PCE zero. A PCE that is not on a queue has both its preceding and following entry fields pointing to its origin. In addition to the chain fields, each PCE has a PCEEWF field, which contains information about the type of event the processor is waiting for. Figure 5-2 provides an example of a dump of JES2 processor queue chains.

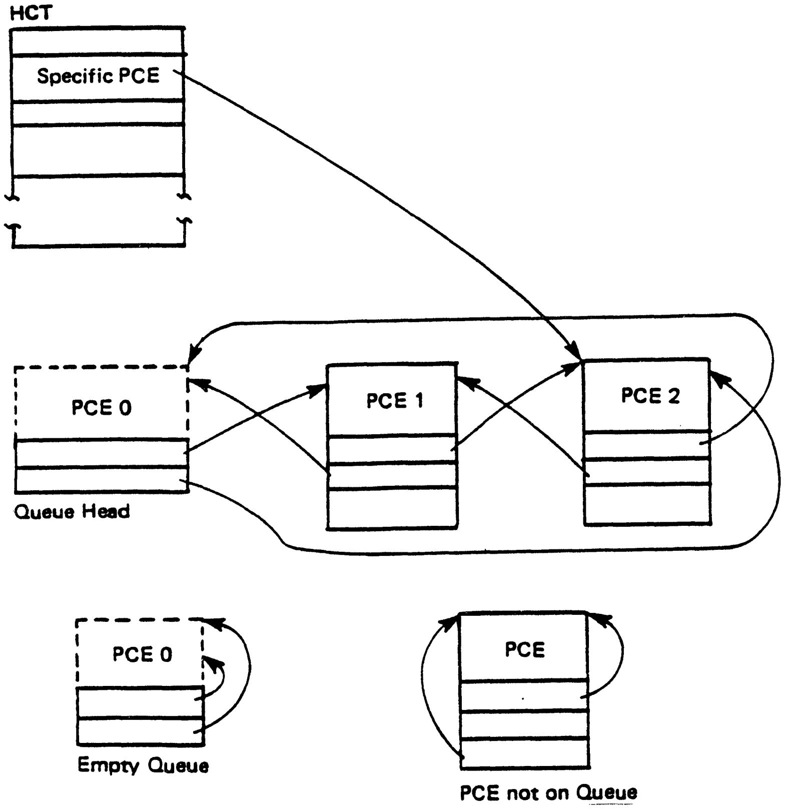


Figure 5-1. JES2 Processor Control Element Relationships

HASP* Communication Table

```
                                                                      1.                         2.
001258A0 TO NEXT LINE ADDRESS SAME AS ABOVE
001258C0 18 00000000 00000000 946CBC20 3EF49000  D5F2D4F1 00124230 00010100 00000101  *.............4..N2M1............*
001258E0 18 00000A64 000009C4 007A1200 006400FA  00240000 10000000 00000001 00000000  *.......D........................*
00125900 18 00000000 00000000 80402010 08040201  00170EF8 00000000 00000000 0002D201  *......... .........8..........K.*
00125920 18 D0F6888C 58F0B2D8 000100FF 00030000  0000012C 00000000 0018CD48 0018C8D8  *.6...0.Q......................HQ*
00125940 18 0018C6C0 0018C1C8 0018C120 00000000  0018D358 0018DFA0 0018E210 0018D148  *..F...AH..A.......L.......S...J.*
00125960 18 00000000 0018DBC0 0018CB18 0018CCA0  0018D8B8 0018D400 0018D800 0018D9D8  *..................Q...M...Q...RQ*
00125980 18 0018E158 0018CD48 FFFBFFFF 00000000  00000000 0012593C 0012593C 00125944  *................................*
001259A0 18 00125944 0012594C 0012594C 0019C4B0  0018F7F0 0012595C 0012595C 001C24C8  *..............D...70...........H*
001259C0 18 0018CCA0 0018E370 001921E0 00125974  00125974 0012597C 0012597C 00125984  *......T.........................*
001259E0 18 00125984 0018D400 0018D400 00125994  00125994 0012599C 0012599C 001259A4  *......M...M.....................*
00125A00 18 001259A4 001259AC 001259AC 001259B4  001259B4 001259BC 001259BC 001259C4  *................................*
00125A20 18 001259C4 0018CD48 0018CD48 000028C1  00597D00 00000000 00000000 00000000  *...D...........A................*
00125A40 18 00000000 0A3A0000 00000958 00000000  00000034 00000000 00000000 00000000  *................................*
00125A60 18 00000000 00000000 000008F0 00000000  00000068 00000548 000005E4 00000000  *...........0...............U....*
00125A80 18 00000000 00000000 00000000 00000000  00000000 00001728 00000000 00000000  *................................*
00125AA0 18 00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  *................................*
00125AC0 18 00000000 00000000 00000000 00000000  00000000 00000000 00000000 0000182C  *................................*
00125AE0 18 00000000 00000000 00000000 00000000  000011E0 00000000 00000000 00000000  *................................*
00125B00 18 00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  *................................*
00125B20 TO NEXT LINE ADDRESS SAME AS ABOVE
```

Processor Control Element

```
                                                                                                  3.
0019C2C0 TO NEXT LINE ADDRESS SAME AS ABOVE
0019C420 18 00000000 00000000 00000000 00000000  00000000 00000000 00000000 0019C1E8  *..............................AY*
0019C440 18 20000000 00000000 001975B8 0019C1E8  00120100 00010001 00010000 00010000  *..............AY................*
0019C460 18 00010000 0019C77C D9F14BD9 C4F14040  00000000 00000000 00000000 00000000  *......G.R1.RD1  ................*
0019C480 18 5093C0C4 00000000 0019C77C 89210008  01030800 91000150 00010000 00010001  *...D......G.....................*
0019C4A0 18 00010001 7F08C1C1 000FE000 00000000  D7C3C540 00000000 00000000 50149AE8  *......AA........PCE............Y*
0019C4C0 18 00149B54 00000000 0019C77C 50149AE8  00000000 00000000 00000000 0014BA70  *..........G....Y................*
0019C4E0 18 00000000 0014AA70 0014CA70 00000000  00125000 00149A70 0019C4B0 0019C4B0  *..........................D...D.*
0019C500 18 0019C1E8 0019C880 0019C880 00125954  00000000 00000000 00000000 00000000  *..AY..H...H.....................*
0019C520 18 00000000 00008207 00000000 00000000  0019C4B0 00000000 00000000 00000000  *..................D.............*
0019C540 18 0019C4B0 000102CC 0019C77C 00000000  00000000 00000000 00000000 00000000  *..D.......G.....................*
0019C560 18 00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  *................................*
               4.       5.       8.       6.       7.
```

Legend:

1. JES2 processor wait queues
2. Queue is empty if first and last pointers point to queue-X'58'.
3. Register 15 in the PCE is the resume point when the processor is dispatched.
4. PCEPREV points to the previous PCE in the chain of all PCEs.
5. PCENEXT points to the next PCE in the chain of all PCEs.
6. PCEPCEA points to the next PCE waiting on this queue.
7. PCEPCEB points to the processor queue-X'58' if this is the first PCE on the queue.
8. PCEID, 8207 = local printer.

Figure 5-2. Example Dump of JES2 Processor Queue Chains

"Restricted Materials of IBM"
Licensed Materials – Property of IBM

Control Block Overviews

This section presents summary information about important JES2 control blocks and contains figures that show the pointers between them. Figure 5-3 provides an overview of the major JES2 control blocks. Figure 5-4 shows RJE/NJE-related control blocks. You can further consult the "JES2 Data Areas" microfiche for a mapping of all JES2 control blocks, or look at HASPDOC, which contains an assembly of each control block in JES2. Use your own HAPSDOC to find the offsets of field names.

The following presents a synopsis of the major JES2 control blocks and their use. The control blocks are grouped according to the function they perform.

Organizational

Abbr.	Name	Storage Type	Primary Use
HCCT	Subsystem Vector Table	CSA subpool 228	Major directory for CSA.
HCT	HASP Communication Table	JES2 address space, HASPNUC	Major directory for HASJES20. Contains queue headers, control blocks, pointers, module and entry pointers, and system parameters.
HFCT	HASP Functional Subsystem (FSS) Communications Control Block	FSS address space	Major directory for HASPFSSM. Contains queue headers, control block pointers, and entry point addresses for FSI and FSS functions.
MODMAP	HASP Module Map	JES2 address space, HASPTABS	Contains the addresses of HASP modules and entry points.
MIT	Module Information Table	JES2 address space, All modules	Contains module information including the exit bit map.
MCT	Master Control Table	JES2 address space, HASPTABS	Contains pointer to various HASP tables and their user extensions.
HAVT	HASP Address Space Vector Table	CSA SUBPOOL 241	Contains information about other address spaces.
HCCT	HASP Common Storage Communication Table	CSA subpool 228	Contains address of data structures used by HASC modules.
HASB	HASP Address Space Block	ECSA	Contains addresses of data structures in CSA.
HASXB	HASP Address Space Extension Block	JES2 address space	Contains addresses of JES2 control blocks in the user area.
CADDR	JES2 Common Storage Address table	Common storage	Contains addresses of routines in common storage called by $CALL.
SVTC	SSVT Function Matrix	JES2 address space, HASPIRMA	Contains the SSI function matrix.

Processor Management

Abbr.	Name	Storage Type	Primary Use
PCE	Processor Control Element	JES2 address space	Unit of JES2 dispatcher. Has associated work space and save areas for JES2 processors.
XECB	Extended Event Control Block	CSA SUBPOOL 231	Provides an area as a wait element for the JES2 dispatcher.

Buffer Management

Abbr.	Name	Storage Type	Primary Use
BUFFER	Buffer	JES2 address space	Basic building block for JES2 control blocks (JCT, IOT, Special).

Job Management (transient)

Abbr.	Name	Storage Type	Primary Use
JCT	Job Control Table	SYS1.HASPACE User address space during XEQ	Primary job-oriented control block. Contains accounting information and pointers to other job information.
IOT	Input Output Table	SYS1.HASPACE User address space during XEQ	Contains job DASD information and PDDBs for input/output data sets.
PDDB	Peripheral Data Definition Block	SYS1.HASPACE User address space during XEQ	Describes a job input or output data set.
OCT	Output Control Table	SYS.HASPACE User address space route, during XEQ	Contains ouptut control records (OCRs) to describe data output records (forms, etc.).
SJXB	Subsystem Job Extension Block	User address space	Contains control blocks for JES2 to perform I/O in the user address space.

Job Management (resident)

Abbr.	Name	Storage Type	Primary Use
JQE	Job Queue Element	Checkpoint data set & JES2 address space	Represents a job in process. Resides on appropriate job queue chain.
JIX	Job Queue Index	Checkpoint data set & JES2 address space	Provides an index to the job queue by job number.
JOT	Job Output Table	Checkpoint data set & JES2 address space	Central control block for all JES2 output. Contains three kinds of JQEs.
JOE	Job Output Element	Checkpoint data set & JES2 address space	Represents output data set by units of work, characteristics of data set, and class of output.
SJB	Subsystem Job Block	CSA subpool 231	Represents a job in process to OS/VS2 used by HASPSSSM interface routines.
SDB	Subsystem Data Block	User address space subpool 229	Used by HASPSSM to control processing of data set using HASP access method (HAM).

Job Management (miscellaneous)

Abbr.	Name	Storage Type	Primary Use
PIT	Partition Information Table	CSA	Completely describes a JES2 logical partition, its job classes and current state.
CAT	Class Attribute Table	JES2 address space	Describes the attributes of a job class.
SCAT	SYSOUT Class Attributes Table	In SSVT space	Describes output classes by print, punch, plot, etc. characteristics.

Unit Management

Abbr.	Name	Storage Type	Primary Use
DCT	Device Control Table	JES2 address space	Represents a unit record device or RJE line. Contains all the information necessary to set up EXCP.
RAT	Remote Attribute Table	JES2 address space	Consists of one entry per remote workstation, containing attributes of workstation.

Multi-system Management

Abbr.	Name	Storage Type	Primary Use
QSE	Shared Queue Control Element	Checkpoint data set & JES2 address space	One per system of a multi-access spool environment, containing identification and cross-system communication paramemters.

Console Management

Abbr.	Name	Storage Type	Primary Use
CMB	Console Message Block	CSA SUBPOOL 231	Used by the command processor to process commands, responses, and messages.

Spool Management

Abbr.	Name	Storage Type	Primary Use
DAS	Direct Access Spool Data Set Control Block	JES2 address space, CSA: SP231, SP241	Used by the spool processor to maintain spool space.
CKB	Checkpoint Buffer	Checkpoint data set & JES2 address space	Contains the I/O parameters for checkpointing.

Functional Subsystem Interface (FSI)

Abbr.	Name	Storage Type	Primary Use
FSSCB	Functional Subsystem Control Block	CSA	Represents a functional subsystem (FSS) address space. There is one FSSCB for each FSSDEF initialization statement.
FSSXB	FSSCB Extension	FSS address space	Contains information for FSI orders, including parameter lists and response areas.
FSACB	Functional Subsystem Application Control Block	CSA	Represents a functional subsystem application (FSA). There is one FSACB for each application in an FSS address space.
FSAXB	FSACB Extension	FSS address space	Contains information for FSI orders, including parameter lists and response areas.
FSSWORK	FSS PCE Work Area	JES2 address space	Contains a mapping of the PCE work. Used vby HASPFSSP during FSI processing.
JIB	JOE Information Block	FSS address space	Used to pass job output element (JOE) information between JES2 and an FSS address space.

Recovery Management

Abbr.	Name	Storage Type	Primary Use
PRE	Process recovery element	JES2 address space	Maintain information required for recovery.
ERA	Error recovery area	JES2 address space	Provides the interface between the processor recovery routine and JES2 recovery management.
LGRR	LOGREC record	JES2 address space	Contains JES2 serviceability data.
TRCA	Terminations recovery control area	JES2 address space	Contains termination recovery data.
TRE	TCB Recovery Element	JES2 address space	Contains information about global resources the TCB holds. It is passed as a parameter to the ESTAE of a HASC module.

Subtask Management

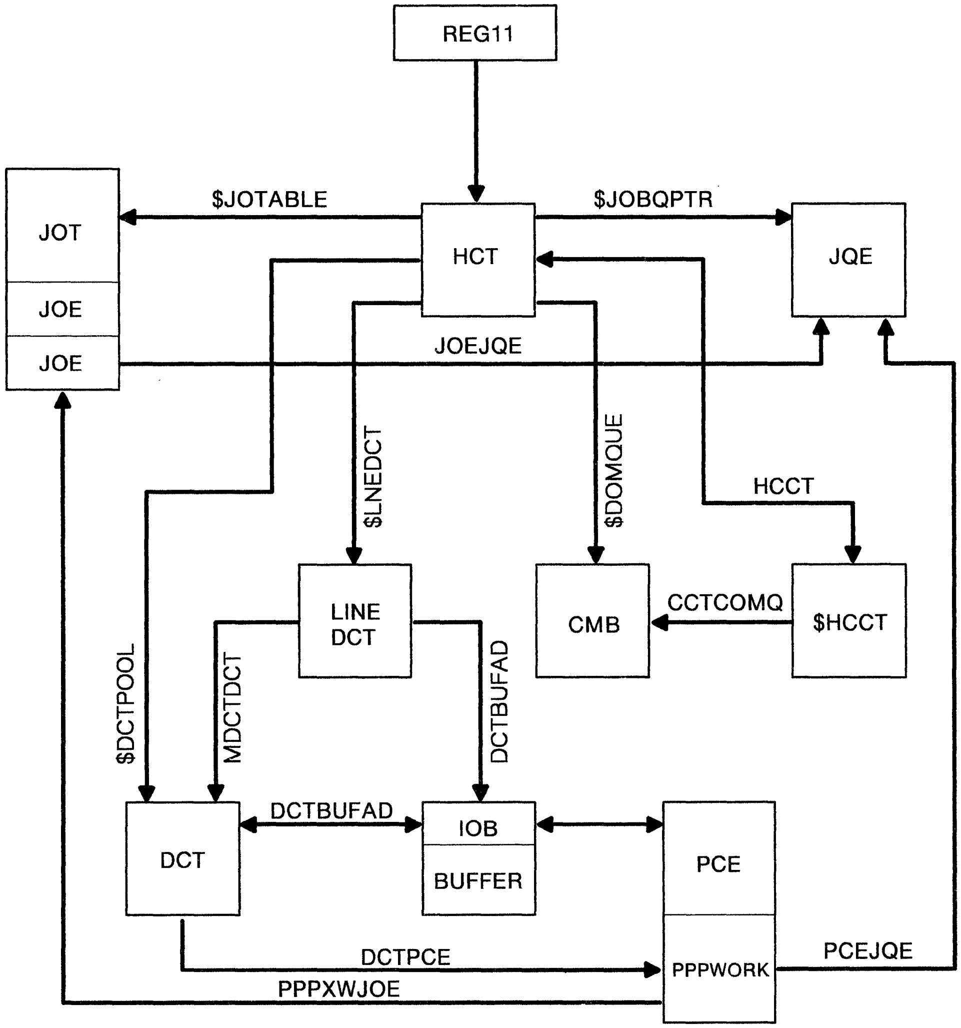
Abbr.	Name	Storage Type	Primary Use
DTE	Daughter Task Element	JES2 address space	Central means of communication between the JES2 main task and its subtasks.
SQD	Subtask Queue Descriptor	JES2 address space	Contains information to be queued to the subtask work queue for a general purpose subtask

Checkpoint Management

Abbr.	Name	Storage Type	Primary Use
HFAM	HASP File Allocation Map	Checkpoint data set	Contains information required for the checkpoint data set
KAWA	Checkpoint allocation work area	JES2 address space	$GETMAINed by CKPTALOC to provide a work area for checkpoint information.
CAL	Change log address list	JES2 address space	Used to build the change log
PAL	Page address list	JES2 address space	Contains one entry for each control block in the change log
PSL	Page Service List	JES2 address space	Contains a list of start and end addresses for each piece of storage to be serviced by the PGSER macro.

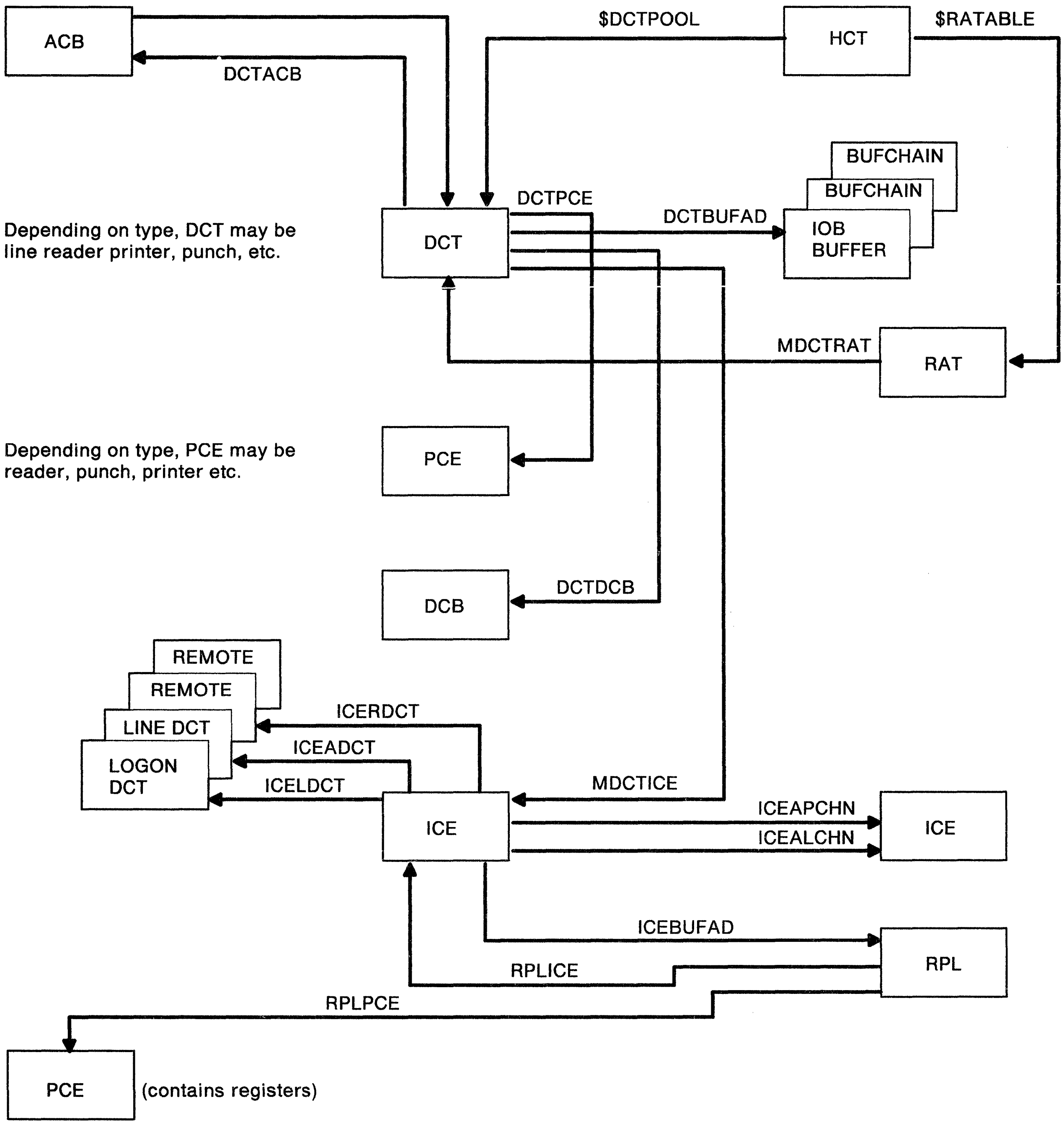
Security Management

Abbr.	Name	Storage Type	Primary Use
General:			
WAVE	Work Access Verification Element	JES2 address space	Contains list form of RACROUT request types for $RACROUT routine.
SAFINFO	SAF Information Parameter List	JES2 address space	Contains SAF information for JOBVALM and SYSOVFY.
Remote Networking:			
CAPE	Communications Access Parameter Element	JES2 address space, HASPRTAM, BSC, SNA	Contains the MSAFCHK parameter list used by the Line Manager and the Remote Console Processor.
SWEL	Signon Work Element	JES2 address space, HASPBSC, SNA	Contains information collected by front end RJE signon.



Note: *Use your own HASPDOC to find the offsets of field names*

Figure 5-3. Pointers Between Major JES2 Control Blocks



Note: *Use your own HASPDOC to find the offsets of field names*

Figure 5-4. RJE/NJE-Related Control Blocks

Important Fields in Major Control Blocks

ACB (access method control block)

ACBLNDCT -- address of the VTAM logon DCT

BUFFER (BSC)

IOBECBCC -- I/O completion code
BUFECBCC -- I/O completion code
BUFSTART -- start of buffer workspace

BUFFER (SNA)

RPLBUFST -- start of VTAM buffer work space (see MVS mapping for an RPL)

CMB (console message buffer)

CMBCMB -- address of the next CMB
CMBMSG -- console message

DCT (device control table)

DCTSTAT -- status of the device
DCTPCE -- address of the PCE associated with this device
DCTBUFAD -- address of the buffer
DCTACB -- address of the ACB
DCTDCB -- address of the DCB
DCTBUFCT -- the count of active buffers
DCTDEVTP -- the device type
DCTFLAGS -- operator command flags
DCTCHAIN -- address of the next DCT
DCTDEVN -- the device name

HCT (HASP communication table)

$HASPMAP -- address of JES2's module directory
$WAIT -- entries to the JES2 dispatcher
$GETBUF -- entries to various service routines
$WTO -- TCB and ECB address for subtasks
$HCCT -- HASP common communication table
$HASPTCB -- control block directory
$NUMRDRS -- configuration constraints
$PRIRATE -- operating constraints
$REGSAVE -- miscellaneous control fields
$COMMPCE -- processor PCE addresses
$CURPCE -- address of the current PCE
$HASPECF -- dispatcher event control field
$READY -- queue header for PCEs eligible for dispatching
$SAVEBEG -- checkpoint record

Fields in the HCT related to RJE/NJE

$LNEDCT -- first LINE DCT
$LOGNDCT -- first LOGON DCT
$RATABLE -- address of the remote attribute table
$NITABLE -- address of the node information table
$NATABLE -- address of the nodes attached table
$RMTSON -- remote signon table

$ICETRAY -- first free interface control element) ICE address
$MCONMSG -- address of queue of input messages from remote consoles
$RJECHEQ -- queue of buffers (IOBs and RPLs) for the line manager
$VLOGQUE -- queue of LOGON DCTs for OPEN/CLOSE VTAM ACB subtask
$BUSYRQ -- queue of output messages for remote console processor
$MLLMPCE -- line manager PCE
$MCONPCE -- remote console PCE

ICE (interface control element)

ICESTAT -- ICE status indicators
ICEAPCHN -- address of the next ICE chained to LOGON DCT
ICEALCHN -- address of the next ICE allocated to a line
ICEXTCHN -- address of the next ICE on the line manager work queue
ICEINHD -- inbound buffer queue head
ICBOUTHD -- outbound buffer queue head
ICEADCT -- address of the logon DCT
ICELDCT -- address of the line DCT
ICERDCT -- address of the remote DCT
ICESDCT -- address of the first suspended DCT

JCT (job control table)

JCTJQE -- offset of JES2 job queue entry (JQE)
JCTJOBID -- JES2-assigned job identification
JCTJNAME -- job name from the job card

JOE (job output element): Work JOEs are chained in the class queues. The JOE counter in the JQE contains the number of work JOEs for the job that is represented by that JQE.

JOENEXT -- next work JOE
JOEFLAG -- flags
JOECHAR -- characteristics JOE
JOEROUT -- the remote ID of the data
JQEJQE -- address of the JQE

Characteristic JOEs are chained in the characteristics queues. The work JOE points to them. Since many work JOEs might require the same characteristics, a characteristics JOE might be pointed to by many work JOEs.

JOENEXT -- next characteristics JOE
JOEFORM -- forms ID
JOEFCB -- FCB ID
JOEUCS -- UCS ID

Save Area Linkage Conventions

All JES2 processors, at one time or another, require the services of various subroutines or sub-processors. These can be local subroutines or JES2 central services. In turn, these called subroutines may require the services of yet other subroutines. It is the responsibility of the called subroutine to make the internal characteristics transparent to the caller. (The caller must be aware of external interfaces such as parameters and return codes.) The called subroutine accomplishes this transparency by saving the caller's environment on entry and restoring it on return. The $SAVE and $RETURN macros are provided for this purpose. See *JES2 Customization* for information about issuing these two macros.

During execution of any JES2 processor, the address of its processor control element (PCE) is in register 13 even though the processor might be several levels down in subroutines. The PCE itself contains an MVS save area. This save area in the PCE is *not* used by the JES2 $SAVE and $RETURN macros, but is used when routines are called that use MVS save area conventions, and by the $WAIT routine.

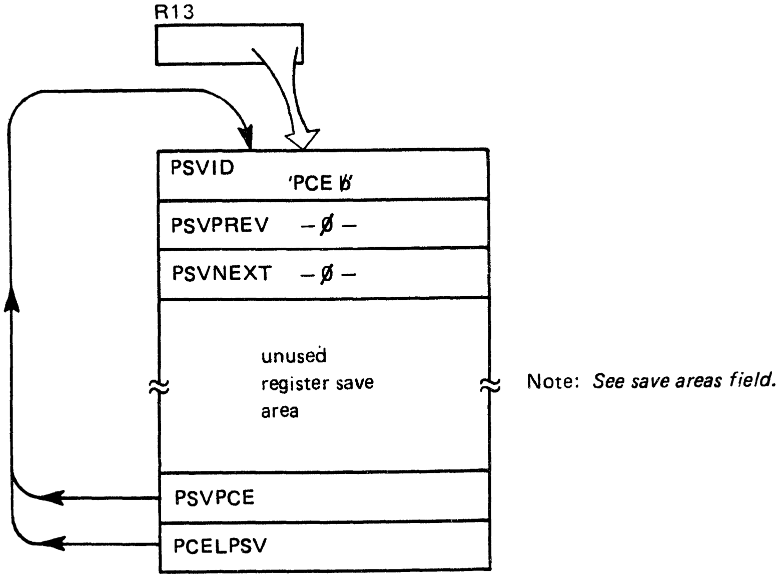


Figure 5-5. PCE-level Save Area in the PCE

When a JES2 subroutine is called, it issues a $SAVE macro. $SAVE obtains a JES2 save area and places it on the end of the current processor's save area chain. Therefore, register 13 points to the processor's save area chain.

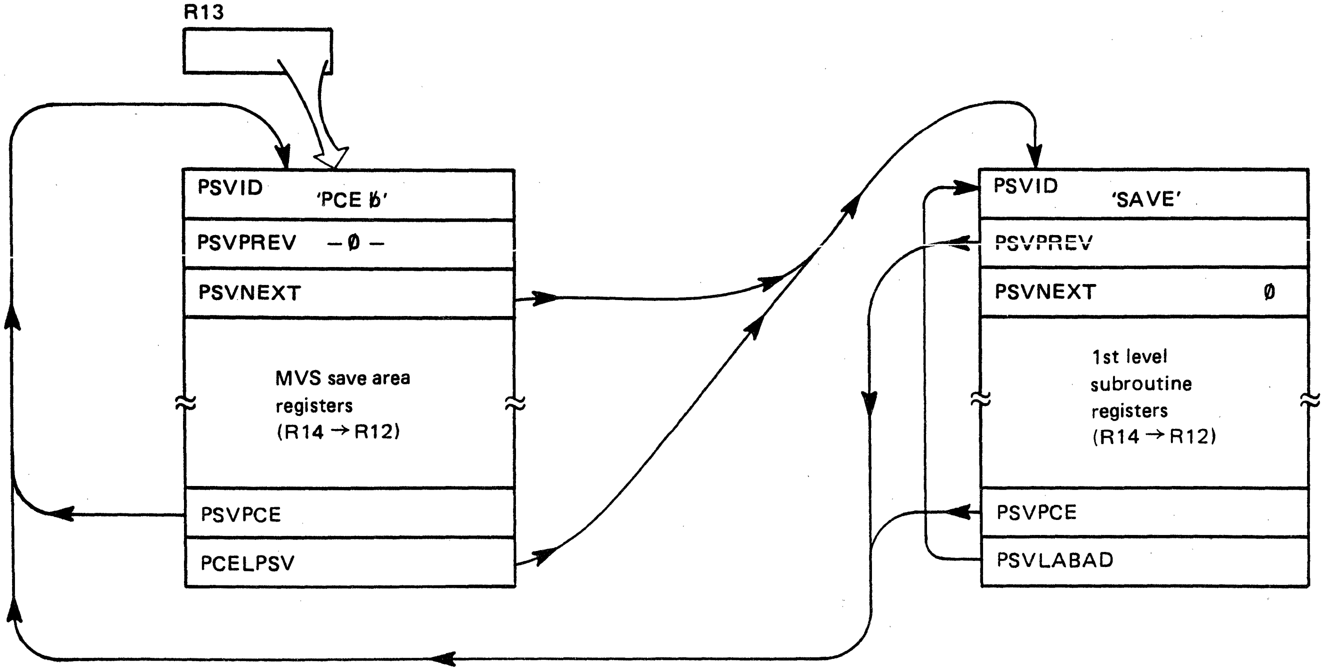
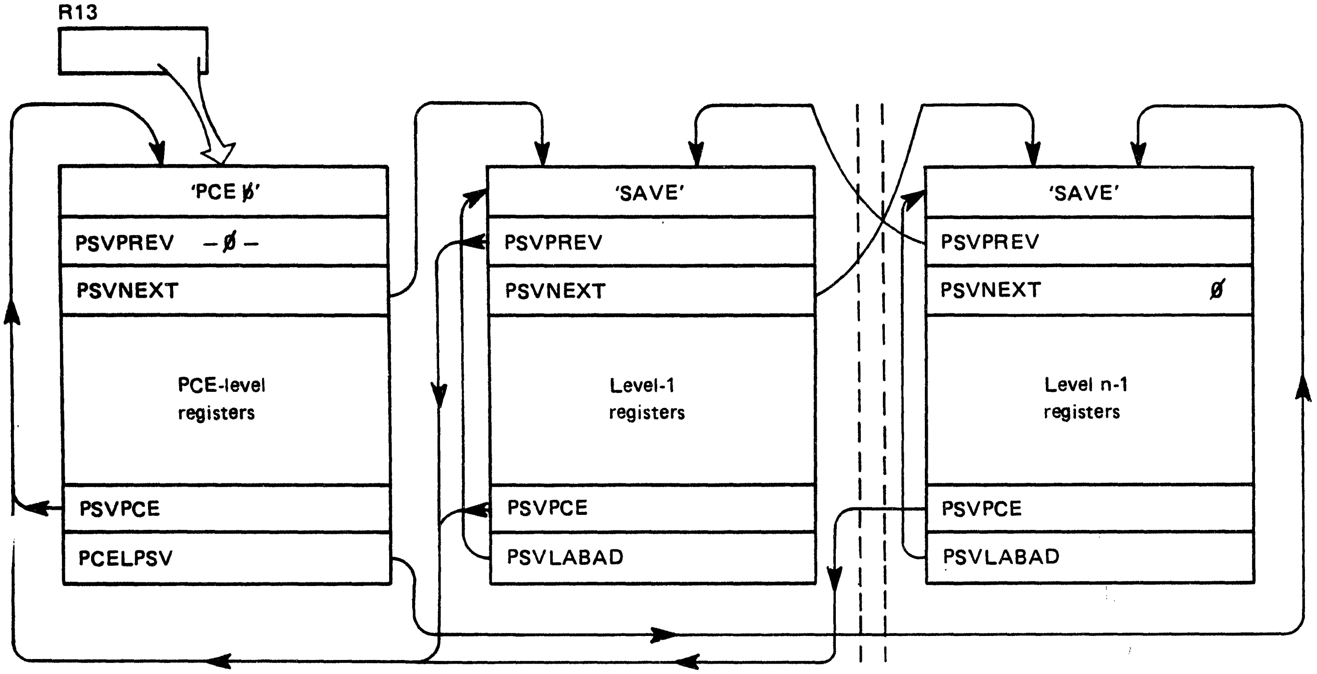


Figure 5-6. Save Area Chain for First-Level Subroutine

Processing occurs on each successive level as shown in Figure 5-7. A $SAVE macro is invoked, and a new JES2 save area is obtained and chained on.



Note: *PSVLAST is only meaningful in the PCE save area.*

Figure 5-7. Save Area Chain for Successive Levels of Subroutine

The $RETURN macro instruction restores the previous level's registers and unchains the save area. The following paragraphs describe the various save area fields.

PSVID: This field contains the characters "PCE" in the PCE save area, or "SAVE" in other save areas on the processor's save area chain.

PSVPCE: This field contains the address of the processor control element (PCE), and has the same value in all save areas on the processor's save area chain.

PSVPREV: This field contains the address of the previous save area, that is, the one belonging to the previous level that issued a $SAVE macro instruction. It is possible, via this field, to start at any level and trace the path to that level. PSVPREV contains 0 when the processor is executing on the PCE-level.

PSVNEXT: This field contains the address of the next save area on the processor's save area chain. It is possible, via this field, to start at the PCE-level and trace the flow of the processor through all levels.

PSVNEXT contains 0 in the final save area on the processor's chain. (This field remains 0 while the processor is executing on the PCE-level.)

PCELPSV: This field contains the address of the last (final) save area on the processor's save area chain. Therefore, this field points to the current save area.

PSVLABAD is the beginning address ($SAVE ID) of this same area.

Remote Job Entry and Networking Problems

Initialization statements, traces, and miscellaneous tools can help you diagnose remote job entry and networking problems.

Initialization Statements: The following JES2 initialization statements are the single most meaningful items of documentation obtainable when diagnosing remote terminal-oriented JES2 problems.

- LINEnnnn statement describes to JES2 the device address and optional features associated with a real TP adapter.
- RMTnnnn statement describes to JES2 the type of remote terminal and its associated features.
- Rnnnn.PRm statement describes the characteristics of one printer attached to the terminal represented by RMTn. Multiple statements may be used.
- Rnnnn.PUm statement describes the characteristics of one punch attached to RMTn. Multiple statements may be used.
- Rnnnn.RDm statement describes the characteristics of one reader attached to RMTn. Multiple statements may be used.

Traces: The following traces can be used to trace the indicated items:

$TRACE

BSC TP buffers
SNA RPLs

CCWTRACE

BSC only
CCWs and data

3705 EP trace

BSC only

GTRACE macro

These traces can be put anywhere to trace anything

VTAM traces

Internal (API, PIU) -- traces JES2 and VTAM interfaces
External (RNIO, BUFFER) -- traces lines between JES2 and VTAM (both data and control)

Miscellaneous Diagnostic Aids: The following miscellaneous items can be used to understand the current state of the system and what commands were issued before the problem occurred.

- A JES2 dump.
- The console log.
- JES2 $DISPLAY commands:
  $DU,RMTnn
  $DU,LOGON1
  $DF or $DJ or $DN
  VTAM display commands (DISPLAY NET, ID=)
- "Appendix B. External Writer" may also be useful for diagnosis.

JES2 Error Services

When JES2 encounters an error, it enters one of the following routines, which comprise JES2 error services:

- Disastrous error routine
- Processor retry routine
- JES2 ESTAE routine
- $SDUMP routine
- JES2 exit routine
- Input/output error-logging routine.

The following brief descriptions of the error routines explain their functions and what can be done when one of these routines is entered.

Disastrous Error Routine: This routine is entered at entry point $DSTERR in HASPNUC whenever a physical I/O error occurs, or whenever a logical error is detected when reading a job control table (JCT) or an input/output table (IOT). The symbol and module names are moved into the message from the $DISTERR macro expansion. A $WTO is issued to notify the operator of the error, and control is returned to the calling processor. The message to the operator is as follows:

```
$HASP096 DISASTROUS ERROR AT SYMBOL symbol IN CSECT module
```

JES2 should be quiesced and restarted as soon as it is practicable in order to recover any direct-access space that might have been lost as a result of the error.

Processor Retry Routine: This routine is entered whenever JES2 encounters a catastrophic error or system abend. The processor retry routine (established by the $ESTAE macro) executes under control of the processor that was in control at the time of the error. The installation can construct error retry logic for dealing with programming errors that normally result in catastrophic error or abend termination.

On the $ESTAE macro instruction, the installation specifies the address of the retry routine, whether or not an SVC dump is to be taken, and, optionally, if message $HASP070 is to be issued. Message $HASP070 requires the operator to specify if an SVC dump is to be taken and if JES2 should continue with the recovery attempt or terminate.

JES2 ESTAE Routine: This routine is entered at entry point $ABEND in HASPTERM whenever JES2 abends for any reason and a processor retry routine did not intercept the error. The catastrophic error routine is called with an error code of ABEND and control is passed to the JES2 exit routine.

$SDUMP Routine: This routine is entered at entry point $SDUMP in module HASPTERM when a $SDUMP macro is issued to request an SVC dump. The $SDUMP routine issues an operator message that indicates the SVC dump was requested and by whom. It then sets up the dump title, and, if required, builds the default title with the dump requester information in the $HASP080 message.

The $SDUMP routine sets up the ASID list that was requested by the caller of the SVC dump and then takes the dump. (The dump does not include the nucleus, unless the operator's dump options include the nucleus.) The $SDUMP routine waits for the dump to complete. The ECB option is used because the ASIDLST option is specified.

If the SVC dump fails and the caller specified an ERROPT option of WAIT, message $HASP089 is issued to the operator's console. The operator can retry the dump or bypass the dump.

JES2 Exit Routine: This routine is entered from the catastrophic error routines whenever JES2 is to terminate under abnormal circumstances, and whenever a $P JES2 command is successfully executed. When entered from the catastrophic error routine, the following WTOR message is issued:

```
$HASP098 ENTER TERMINATION OPTION
```

The routine waits for the operator to respond with one of the following replies:

- EXIT
- PURGE
- DUMP text

If the reply is EXIT, the subsystem vector table ($HCCT) termination complete flag is set to 1. Control is returned to the system in the case of JES2 error detection by an SVC 3 instruction with register 15 set to 24; or, in the case of a JES2 task abend, by a branch to the location in register 14.

If the reply is PURGE, the routine attempts to clean up commonly addressable control blocks. If the subsystem is the primary subsystem: the UCB attention index values are set to zero; tasks waiting for CANCEL/STATUS, process-SYSOUT, and storage cell expansion queues are posted; a system management facility (SMF) record may be optionally written, and JES2 subtasks are terminated and detached. Control is then returned to the system as with the EXIT option.

If the reply is DUMP, a SDUMP macro instruction is executed with the text (if any) used as the header. Processing continues as with the PURGE reply.

If entry to the routine is through the normal execution of the $P JES2 command, processing is the same as with the PURGE option for abnormal terminations, except that control is returned to the system by an SVC 3 instruction with register 15 set to zero.

I/O Error Logging Routine: This routine is entered at entry point $IOERROR in module HASPNUC when an unrecoverable I/O error occurs on a JES2 spooling volume, or when a line error occurs, which might require the attempt of the operator. A $HASP094 message to the operator is generated containing:

- The channel status, channel command code, sense information, track address, and line status are retrieved from the IOB (pointed to by register 1) and formatted.
- The unit address and volume serial are obtained from the UCB.
- The device name (if applicable) is acquired from the device control table (DCT).

Common JES2 Subtask ESTAE Routine: Each subtask establishes $STABEND as its recovery routine. When the subtask issues the ESTAE macro it passes $STABEND the address of its DTE. This routine provides diagnostic information for problem determination (and ensures that diagnostic information is written in SYS1.LOGREC) and attempts subtask recovery. A system dump can be obtained based on installation-defined recovery options (RECVOPTS). Each subtask DTE specifies its own retry, VRA formatting, and clean-up routines. These routines are called by $STABEND. If recovery is not possible, $STABEND terminates the JES2 main task with a $ERROR ZO3.

Miscellaneous Hints on JES2

Starting JES2 - Enqueue Wait on STCQUE: The installation can choose the option to manually start JES2 by changing MSTRJCL with AMASPZAP. When MSTRJCL is changed, JES2 parameters are entered on an operator-issued START command (that must be issued before MVS processing can occur). If the operator misspells JES2 on the START command (such as entering JES), a wait occurs. In this case, there is no indication of this other than that STC (IEESB605) is exclusively enqueued on SYSIEFSD STCQUE behind IEEVWAIT, which does not release the resource until JES2 is initialized. The command scheduling control block (CSCB), pointed to by the parameter list to IEFJSWT, is not formatted in the dump.

Therefore, if an enqueue wait on STCQUE occurs, the START command might have been entered incorrectly. This is also true for any normally started tasks (such as mounts or installation started tasks) that cannot be started until the primary job entry subsystem is started.

Using Dumps, Traps, and Traces

The following describes where to locate important JES2 dump information, how to trap program errors, and event tracing in JES2.

Dumps

One of the most important things to remember about dumps is to find the PCE involved, then use the register save area to get into the code or the module involved. Register 14 tells you where control came from; register 15 tells you where control is to go.

The first part of CSECT HASPNUC (which is the first in the load module HASJES20) is the JES2 HASP communications table (HCT). The HCT contains the addresses of many JES2 control service sub-programs and contains most of the globally used status bytes, counters, control block chain pointers, and queue pointers. CCTHCT in the $HCCT points to the HCT. $HCCT in the HCT points to the HCCT.

If the dump has formatted MVS control blocks, such as those printed by IPCS, the multiple task structure of JES2 will be apparent. The main task is the load module named HASJES20. There are always at least five subtasks present: HASPWTO, HASPACCT, HASPIMAG, and HOSCNVT, and HOSPOOL. These identifiers can be found in the NM field of the formatted PRB.

During the execution of instructions in HASPSSSM, register 11 usually points to the $HCCT, register 13 usually points to an available save area and register 10 usually points to the system job block (SJB). In the HASP access method (HAM), register 10 points to the system data block (SDB). Similarly, within the main JES2 task, register 11 points to the HCT and register 13 points to a PCE.

To locate the save areas, refer to "Save Area Linkage Conventions" described earlier in this chapter.

If you need a dump, you can request one in the following ways (the first one is best, the next one is second best etc.).

- Reply DUMP to the HASP098 message.
- Issue the $SDUMP command. The $SDUMP service routine issues an MVS SDUMP command (see "$SDUMP Service Routine"in Chapter 3 for more information).
- Issue a $PJES2,ABEND command -- this may not give you an exact picture of storage at the time of the abend.
- Issue a stand-alone dump (SDUMP).

The IPCS Exit

The JES2 dump formatting service formats JES2 storage-resident control blocks. It is useful in initial problem determination using JES2 dumps. It can be modified to suit an installation's particular requirements either by adding control block printouts to the dump or deleting those not needed. The dump service executes as a user exit of the IPCS print dump utility.

This dump service formats control blocks including:

- JESCT data: The JES communication table
- SSCT data: Subsystem communication vector tables
- SSVT data: Subsystem vector table, including:
  - SSI matrix
  - function routine addresses
- HCCT data: HASP common communication table, including:
  - estimated count fields
  - $$POST elements
  - job service queues
  - spool management pointers
  - SYSOUT class attribute table
- MODMAP data: Addresses of JES2 modules
- HCT data, including:
  - addresses of JES2 services
  - addresses of JES2 control blocks
  - address of the track group map
  - initialization parameters, grouped by
    configuration constraints
    operating constraints
    estimated count fields
    other
  - PCE addresses and queue heads
  - checkpointed variables
  - job queue heads
- QSE data: MAS complex control fields -- all 7 QSEs are formatted
- Special processor PCE control blocks: The current PCE is formatted, followed by all other non-device related PCEs.
- DCTs with associated PCEs
- DTEs -- JES2 daughter task elements (subtasks)

Installing the Dump Service

The code for the dump service comes in both source and object form. The standard JES2 SMP installation procedure will place the modules on DASD, assemble and link them. If you have bypassed this procedure, you can link the dump formatting modules using member JES2BLD, which comes with JES2 source code in data set HASPSRC.

HASPBLKS is the module that governs the formatting activities and is the one invoked by the JES2 keyword. HASPBLKS uses IPCS storage access and print services to do its work.

HASPBLKS verifies that the version of JES2 which is about to be formatted is the same as the version of the JES2 IPCS exit modules. If the two versions are not the same, the JES2 IPCS exit will issue a message to terminate processing.

HASPBLKS verifies each of the other IPCS modules it loads as each of these modules is loaded. If a module is not at the same version level as the HASPBLKS module, an error message is generated and HASPBLKS continues to the next formatting module.

The exit control table (ECT) entry for JES2 in the module AMDPRECT turns off the MVS/370 compatibility flag. If this flag is on, a IPCS exit can access only 24-bit data.

HASPBLKS loads the following formatting table modules: (The subroutine that formats the name and address of a JES2 control block for each of these formatting modules generates an 8-character address)

- HASPFMT0, which formats the JESCT, JESPEXT, and SSCT
- HASPFMT1, which formats the SSVT, HCCT, and CADDR
- HASPFMT2, which formats the MODMAP, HCT, and PADDR
- HASPFMT3, which formats all QSEs
- HASPFMT4, which formats special processor PCEs
- HASPFMT5, which formats DCTs and their PCEs, including the offload device DCTs
- HASPFMT6, which formats DTEs and dumps associated work areas
- HASPFMT7, which formats checkpoint-related control blocks: HFAM, CKG, CKB, CKW, and KAC
- HASPFMT8, which formats HASB, HASXB, SJB, and SJXB

These are the macros used by these modules:

- $FIND, which builds DSECT addresses
- $TITLE, which creates titles for dump listings
- $DUMP, which causes data field maps to be built
- $TEST, which causes conditional mapping of data
- $LINE, which creates a new print line
- $PAGE, which creates a new page

Running the IPCS Service

You can obtain a dump either by issuing an MVS operator DUMP command or by waiting until JES2 abends and causes a system dump. The operator dump command is "DUMP COMM=(title)". You reply to the message that follows with the ASID belonging to JES2. Use the display command with the ALL option ('D A,ALL') to find out the ASID of JES2. If JES2 abends, JES2's ESTAE routines will issue an SDUMP macro to cause a system dump.

Make sure that the system dump provides the RGN, CSA, and NUC options. You need RGN for the JES2 address space, CSA for subpool 241 data, and NUC for JESCT. You can verify that you have the needed options with the "display dump options" operator command ("D D,O").

Following is an example of a job that could be used to format and print JES2 control blocks from SYS1.DUMP00. The subsystem name is JES2, the ASID where JES2 was running is X'E', and all of the format modules are to be used.

```
//MYIPCS   JOB
//IPCS     EXEC PGM=IKJEFT01,REGION=4096K
//STEPLIB  DD  DSN=SYS1.LINKLIB,DISP=SHR
//SYSTSPRT DD  SYSOUT=*
//IPCSSTOC DD  SYSOUT=*
//IPCSPRNT DD  SYSOUT=*
//SYSPROC  DD  DSN=SYS1.SBLSCLI0,DISP=SHR
//SYSTSIN  DD  *
  IPCS
  SETDEF NOCONFIRM PRINT NOTERMINAL DA('SYS1.DUMP00')
  VERBEXIT JES2 'SUB=JES2,ASID=E,FMT=ALL'
  END
/*
```

The format of the VERBEXIT subcommand in IPCS and the VERBEXIT control card illustrated above is:

```
  VERBEXIT JES2 'SUB=,ASID=,FMT='
or
  VERBEXIT HASPBLKS 'SUB=,ASID=,FMT='
```

The verb name JES2 can be substituted for the format module name HASPBLKS.

The options are as follows:

SUB = subsystem name
The subsystem name is the name of the JES2 subsystem (typically "JES2"). If no subsystem name is given or if the SUB= option is omitted, the default subsystem name is JES2. If no other options are given, the ASID will be supplied by IPCS/SNAP and the format option will be FMT=ALL.

ASID = address space identifier (in hexadecimal)
The ASID is either that of the JES2 subsystem itself or that of the user. If a user's ASID is given, then the control blocks produced by HASPFMT8 (HASB, HASXB, SJB, and SJXB) will represent the user's address space. The default format option is FMT=COM. If you wish to override this, you must provide a specific FMT= option:

- FMT=COM causes the use of format modules HASPFMT0, 1, and 8.
- FMT=ALL causes the use of all format modules (0-8).

Note: It is possible to specify the SUB= and ASID= options representing different address spaces. Make sure that both options represent the same address space identifier.

All options are positional. Following are examples of different ways to specify JES2 options:

```
  VERBEXIT JES2 'SUB=JESA,ASID=F'
  VERBEXIT JES2 'SUB=JES2,FMT=COM'
  VERBEXIT JES2 'ASID=E,FMT=ALL'
  VERBEXIT JES2 'ASID=D'
  VERBEXIT JES2 'FMT=COM'
```

For more information about the IPCS service, see *IPCS User's Guide*, (GC28-1833).

The Program Event Recording (PER) SLIP Command

If you are not familiar with the SLIP command facility, see *MVS Diagnostic Techniques* and *MVS System Commands* for an introduction and a full explanation of the SLIP command. The following text assumes you know this information.

The general form for setting a trap is:

```
SLIP SET,event,ENABLE,ID=xxxx,ACTION=xxxx,other parameters,
MATCHLIM=xx,END
```

The system responds when the command is entered. If no ID was used, an ID is assigned by the system. This ID identifies the trap and is used to enable, disable, or delete the trap. To display a trap, use D SLIP, which gives you summary information on all SLIP traps, or issue D SLIP=nnnn, which gives you detailed information on SLIP ID nnnn.

The PER SLIPs

Only one PER SLIP event may be monitored at any given time, although more than one trap may be defined. This means that only one trap can be enabled at a time, all others must be disabled. Three types of PER events can be monitored.

1. SA (storage alteration)
2. SB (successful branch)
3. IF (instruction fetch)

Several types of ACTION may be taken, including:

SVCD -- schedule an SVC dump
WAIT -- put the system in a wait state
TRACE -- write a GTF trace record
TRDUMP -- write a GTF trace record and then schedule an SVC dump
IGNORE -- use with other traps to filter out unwanted conditions.

The SVC dump is the default. If specified or defaulted, the dump is scheduled for the address space in which the trap was matched. These dumps do not cause an abend.

Printing a SLIP Dump

The registers and PSW at the time of this kind of dump are not reliably in the TCB and its RBs. They are stored in a 4K buffer pointed to by CVTSDBF in the CVT. This buffer is in CSA, so your SDATA options must include CSA. When you print the dump you must either print all of CSA or print whatever region storage is requested including the CVT. Then you must do another print run specifying the specific address of the SDUMP buffer. In the buffer, EBCDIC character eyecatchers precede the PSW and the registers. The sequence of the registers is 0-15.

The GTF and AMDPRDMP Options for SLIP TRACE

If ACTION=TRACE or ACTION=TRDUMP, GTF must first be started with option TRACE=SLIP. To print the trace records using AMDPRDMP, you must specify EDIT DD=input ddname,SLIP.

The Storage Alteration Trap

The following example traces storage alteration of a field in the HCT, a halfword at offset X'154':

```
SLIP SET,SA,EN,
     ACTION=TRACE
     TRDATA=(STD,REGS,E51F4,E51F5),
     JOBNAME=JES2
     RANGE=(E51F4,E51F5),          (two bytes)
     MATCHLIM=50                   (quit after fifty times)
     END
```

You may not want to trace every time this storage is altered, only the time some instruction turns it into a negative number.

In this case, the DATA= parameter can be used to limit the trap matches to storage alteration of the area indicated by RANGE= only when the area indicated by DATA= is greater than, less than, or equal to the indicated value, whichever you specify. The following example tests for a "less than" condition:

```
SLIP SET,SA,EN,
     ACTION=SVCD,      (produce a dump if storage is altered)
     SDATA=(RGN,CSA),
     JOBNAME=JES2,
     RANGE=(E51F4,E51F5),
     DATA=(E51F4,LT,0000),    (halfword of 0s, default hex)
     END
```

The DATA Keyword:

The form of the DATA keyword is

```
DATA=(target, operator, value)
```

or

```
DATA=(target(b),operator,value)
```

target -- indicates a storage location, a register, or an indirect pointer.

b -- indicates a binary comparison starting with bit "b"

operator -- indicates the relationship between the "target" and the "value"; that is, whether the target is equal to (EQ), not equal to (NE), greater than (GT), or less than (LT) the "value".

Examples

```
DATA=(15R(28),GT,0100)
```

This tests whether bits 28-31 of register 15 are greater than B'0100'

```
DATA=(6R%,EQ,4B3628)
```

This tests whether the three bytes pointed to by register 6 are equal to X'4B3628'

```
DATA=(150,EQ,FF,151,NE,00
```

This is two one-byte tests, one at storage location X'150' and one at location X'151'. Both conditions must exist (that is, storage location X'150' must contain FF and location X '151' must not be equal to 00) in order for the trap to match and take whatever action you specified.

Note: To trap storage alterations, you must know the virtual addresses of the storage area you are monitoring.

The Successful Branch Trap

You may want to trace the path taken through some routine:

```
SLIP SET SB,EN,ACTION=TRACE,
     TRDATA=(STD,REGS),
     JOBNAME=JES2,           (trace path through . . .
     RANGE=(E6582,E6D18),     . . . the job queue manager)
     END
```

The Instruction Fetch Trap

To trap execution of a given range of instructions (for example, entry to an ESTAE routine), use this form:

```
SLIP SET,IF,EN,
     ACTION=SVCD,
     SDATA=(RGN,CSA),
     JOBNAME=(JES2),
     RANGE=(startaddr,endaddr),
     END
```

Note: 'RANGE' may also be a single instruction.

To trap instructions in HASPSSSM or any other module loaded into the link pack area (LPA), you specify:

```
SLIP SET,IF,EN,
     ACTION=SVCD,
     SDATA=(RGN,CSA)
     JOBNAME=(jobname)
     LPAMOD=(HASPSSSM,offset1,offset2),
```

Note that, if you do not specify JOBNAME, the trap springs every time the LPAMOD is entered on behalf of any address space. Note also that instead of the RANGE parameter and its virtual addresses, the LPAMOD parameter uses offsets into whatever module is specified as the start and end of the storage to be monitored.

SLIP Trap for a Specific Message

To save time looking for the module associated with a message, you can set traps for any MVS message. To do so, code:

```
SLIP SET,IF,EN,ADDRESS=(XXXXXX),DATA=(1R%+6,EQ,YYYYYYYY),
     ACTION=(SVCD),END
```

XXXXXX is the instruction 41C0BFFF at offset X'26' or X'2E' in module IEAVVWTO (IGC0003E in LPAMAP). A ZAP dump of the module will give the correct offset.

YYYYYYYY is the hex representation of the third, fourth, fifth, and sixth characters of the message number to be trapped. For example, for message IEF270I, the hex representation for F270 would be C6F2F7F0. This format is necessary because the DATA keyword can compare only four bytes of data.

Note that displacement +6 must be changed to +14 when the message is a WTOR. SVCD is the default on the ACTION keyword. Other actions are possible.

The ACTION = IGNORE Specification

This is used to set a SLIP trap for a certain range but filter out a sub-range(s) that you don't want trapped for whatever event and action you have specified on the command. For example, the following traps:

```
SLIP SET,IF,EN,ACTION=TRACE,
     TRDATA=(STD),
     JOBNAME=JES2,
     RANGE=(E5000,107000),           (trace all in this range . . .
     END

SLIP SET,IF,EN,ACTION=IGNORE,
     JOBNAME=JES2,
     RANGE=(E5628,105230),           . . . except this range . . .
     END

SLIP SET,IF,EN,ACTION=IGNORE,
     JOBNAME=JES2,
     RANGE=(105400,106000),          . . . and this range)
     END
```

The previous traps would trace instructions in these ranges: E5000-E5627, 105231-1053FF, and 106001-107000.

Indirect Addressing in SLIP Traps

Indirect addressing is often used with the TRDATA and DATA parameters. For example, if you are tracing in HASPTERM and want to trace the SDWA (to which register 1 points), you can specify:

```
. . . TRDATA=(1R%+0,1R%+100), . . .
```

This will trace X'101' bytes of data pointed to by register 1.

If you know the SDWA resided at location X'946EF0', you could have used direct addressing and specified:

```
. . . TRDATA=(946EF0,946FF0), . . .
```

Shorthand is permitted in indirect addressing, for example:

```
. . .TRDATA=(1R%,+100), . . .
```

The location in the first member of the pair defaults to '+0' and the register in the second pair uses the register coded on the first pair as a default.

Direct and indirect addressing can be mixed:

```
. . . TRDATA=(2R%+0,+30,9F6E0,9F6F8), . . .
```

In this example, the first pair uses indirect addressing. It specifies X'31' bytes of storage pointed to by register 2. The second pair directly addresses storage between the virtual addresses coded.

Pointing Through a Chain of Pointers

You can also trace storage through chain of pointers. For example, if register 10 points to the CVT:

```
. . .TRDATA=(10R%+128%+18%+10%.. .
```

points to the SSVT for JES2. At X'128' bytes into the start of the CVT (pointed to by register 10), is the address of JESCT. At X'18' bytes into the JESCT is the address of the SSCVT. At X'10' bytes into the SSCVT is the address of the SSVT.

Contents of Trace Records

Trace records are produced by the ACTION=TRACE or ACTION=TRDUMP specifications. GTF must be active. A maximum of 256 bytes plus a 16-byte header may be traced.

There are four types of trace records. The standard trace record contains:

Hex Offset	**Contents**
0	16-byte GTF header
10	ASCB address
14	CPU ID
16	Jobname
1E	Trap ID
22	ASID
24	JOBSTEP program name
2C	TCB address
30	System mode indicators/error indicators
34	SLIP flag
35	Data unavailable (used with DATA= keyword)
37	Load module and offset
43	Instruction address
47	Instruction trapped
4D	Target address of an "execute" (EX) instruction
51	Target instruction of an "execute" (EX) instruction
5	Beginning of the range for an SA trap
5B	Data for an SA trap
5F	Program old PSW and interrupt code
6B	PER interrupt code and trap mode
6D-87	Reserved

SLIP User Trace Record

The contents of the SLIP user trace record are:

Registers (optional)
User-defined fields (191 bytes with registers, 256 without)
A one-byte length field precedes the registers and the data

SLIP Standard/User Trace Record

The contents of the SLIP standard/user trace record are:

SLIP standard record fields
User-defined fields (71 bytes with the registers, 136 without)
A one-byte length field precedes the registers and data

Using the DEBUG Keyword with TRACE

Using the DEBUG option with TRACE produces a fourth type of record, the GTF SLIP DEBUG record, every time the trap is tested.

JES2 DEBUG Functions in a Multi-access Spool Configuration

JES2 systems in a multi-access spool configuration share a single job queue, job output table, and master track group map, all of which are kept on the JES2 checkpoint data set. In addition, the checkpoint data set contains shared system queue elements (QSEs) and other miscellaneous information needed for inter-system control. (Figure 5-8 illustrates the checkpoint format.) The checkpoint data sets are allocated to one processor at a time.

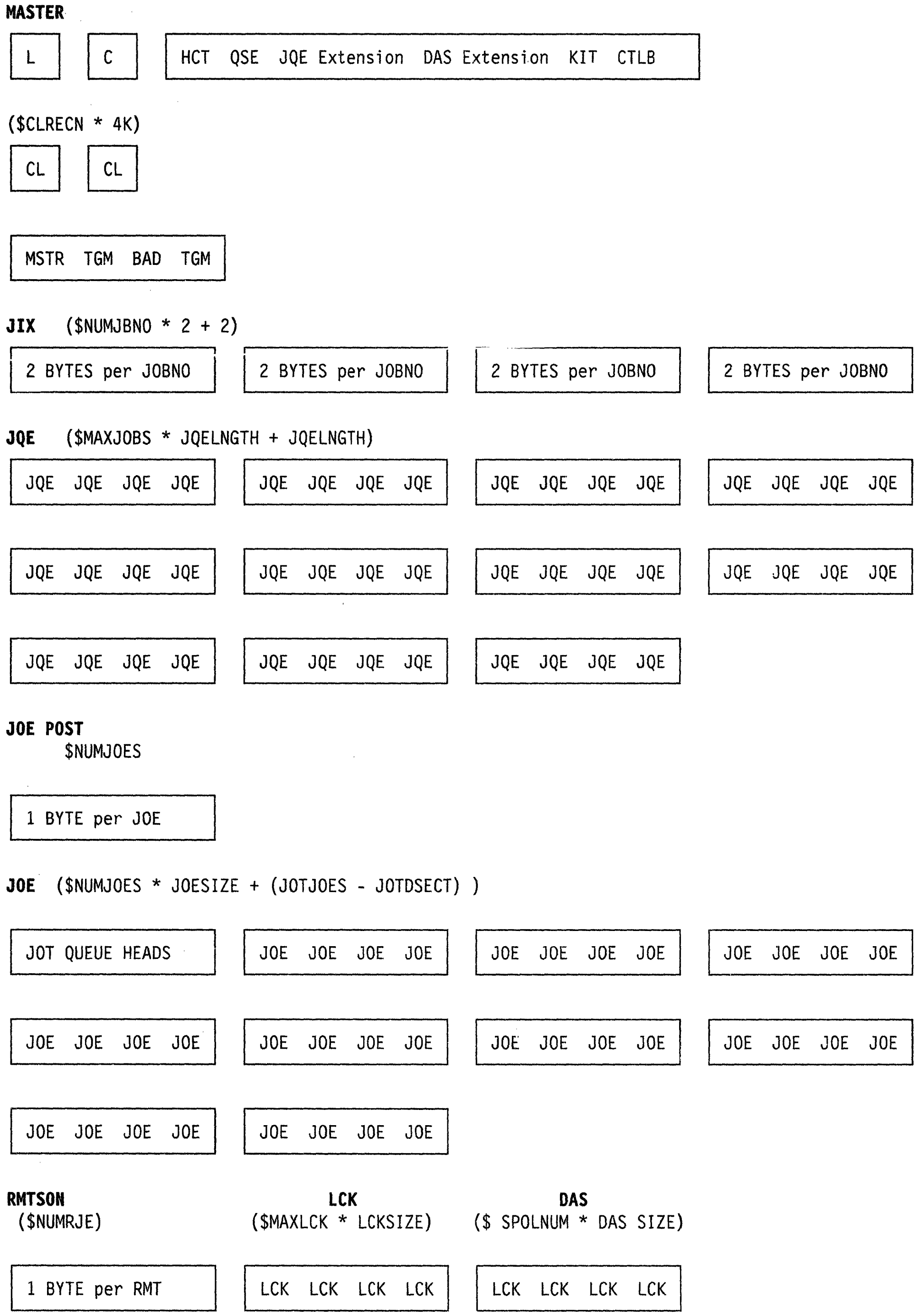


Figure 5-8. Checkpoint Format

Access to any part is controlled by a system's JES2 checkpoint processor, found in the HASPCKPT module. If debug mode is specified (DEBUG=YES), HASPCKPT can detect illegal updates to the checkpoint data set. HASPCKPT does this by copying its in-storage checkpoint data to a 4K I/O area before writing it to DASD. When this system regains control of the checkpoint data set, it compares its current checkpoint data set with its saved copy (the data in the I/O area). If the data is not the same, an invalid update has taken place. Similarly, during a write operation, HASPCKPT compares each 4K checkpoint area that was not flagged for change with its corresponding copy in the I/O area. If they don't match, an invalid update has taken place. If the checkpoint record was flagged for change, it creates a new I/O copy.

The processor has four major sections:

- Initialization
- Read
- Write
- Release

Initialization: Initialization is executed once when the processor is first activated by the JES2 dispatcher.

Read: This function is executed at the beginning of each shared queue ownership period by systems in a multi-access spool configuration. If initialization parameter DEBUG=YES, the 4K pages are compared with copies saved just prior to the last write of a checkpoint. A mismatch indicates an invalid change to these shared queue areas and JES2 is terminated with a $K01 catastrophic error.

After a successful read completion, the time stamps in this system's QSE, and in any QSE for which the $ESYS command had been entered, are compared with a time stamp saved in the HCT. A mismatch indicates that another system has illegally taken ownership of a QSE owned by this system, or that the reserve mechanism (hardware or IOS) has failed to prevent simultaneous access to the MAS configuration checkpoint records. JES2 terminates with a $K03 catastrophic error.

Write: This function is executed repeatedly as a loop in response to various requests by other processors or timers. In a multi-access spool environment, the loop operates only during an ownership or hold period.

If the initialization parameter is DEBUG=YES, the saved copies of the 4K pages are compared with the current 4K pages. If a record has been modified, but its corresponding checkpoint control byte does not indicate this, JES2 is terminated with a $K05 abend. A $K05 abend indicates a failure to issue a $CKPT macro instruction prior to executing a $WAIT macro instruction, after first modifying the 4K pages.

The current hardware time-of-day (TOD) clock is recorded in the HCT, along with this system's QSE and a QSE for which any $ESYS command had been entered. In multi-access spool systems, these time stamps are verified following the next read operation to ensure the integrity of QSE ownership.

If the parameter is DEBUG=YES, copies of the 4K pages to be written are saved. In multi-access spool systems, these are used prior to the next read operation to detect invalid updates to shared but not owned information.

Release: READ2, KPRIMW, and KFORMAT (see HASPCKPT) page-free and page-release the I/O area records after I/O is complete. The system default on the MVS PGFREE service is RELEASE=YES so that non-fixed pages are not written to the page data set when they are paged out. If the DEBUG option is chosen, the virtual copy of the record in the I/O area must be retained for comparison, that is, the fixed pages are freed but not released. With DEBUG=YES, the MVS PGFREE service uses RELEASE=NO, which guarantees that the content of the virtual page remains intact when the area no longer remains in central storage. This, however, results in a considerable amount of paging I/O.

DEBUG and HASPNUC

Because the $GETWORK routine in HASPNUC obtains a member of a pool of fixed-length areas, the area provided and then returned might be larger than the area requested and used. If DEBUG=YES is specified, a check is made to ensure that the area used did not exceed the area requested. If the area used is not the same size as the area requested, catastrophic error $GW3 is issued.

The Usefulness of DEBUG

The DEBUG facility is most useful for trapping unauthorized alteration of a JQE or JOE. It can detect a failure to issue a $QSUSE or a $CKPT macro. It can also detect random overlays of the 4K pages. However, the facility cannot detect all unauthorized alterations and its limitations should be understood.

If a routine changes a JQE or JOE without issuing $QSUSE, it is possible that the routine already has exclusive control of the checkpoint data set. In this case, DEBUG will not detect the change. And if a routine changes a JQE or a JOE without first having issued a $CKPT, and another JQE or JOE in the same block was validly altered during the same checkpoint cycle, the error will not be detected. As a result, many unauthorized changes or overlays may remain undetected.

Modifying DEBUG

The operator can change the current setting of the DEBUG statement by issuing the $T command (either $T DEBUG=YES or $T DEBUG=NO). The $D DEBUG command can be used to display the current setting of the DEBUG statement. Message $HASP827 will appear displaying the setting.

Tracing and the $TRACE Facility

For general information about tracing events in JES2 processing, see *SPL: JES2 Initialization and Tuning*. This section explains what events can be traced and provides examples of trace output for certain trace ids.

TRACE IDs

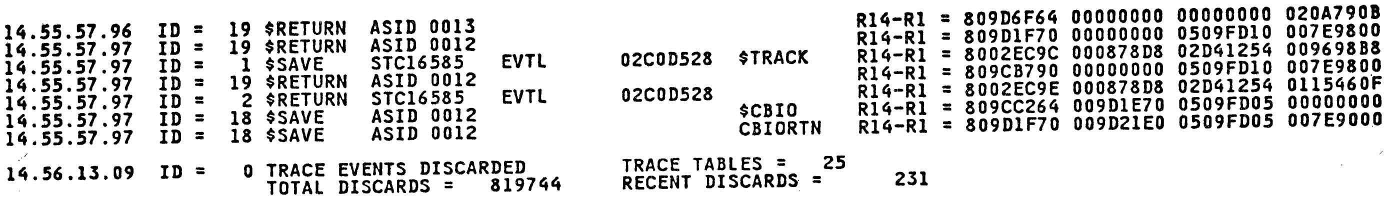
Figure 5-9 summarizes the various types of information available from the standard JES2 tracing facility.

Figure 5-9. Trace Functions

Trace ID	Type of Information
0	Don't trace but spin-off the current trace log data set
1	Trace the $SAVE macro usage of all PCEs.
2	Trace the $RETURN macro usage of all PCEs.
3	Trace JES2 disastrous error results
4	Trace channel ends for BSC remote lines (BSC buffer trace)
5	Trace RPL completions for SNA remote lines (SNA buffer trace)
6	Trace JES2 initialization processing
7	Trace JES2 abnormal termination
8	Trace specific debugging locations in JES2
9	Trace specific debugging locations in JES2
10	Trace specific debugging locations in JES2
11	Trace the use of the $SAVE macro in all PCEs with TR=P. Also trace all SSI routines with TRACE=YES.
12	Trace the use of the $RETURN macro in all PCEs with TR=P. Also trace all SSI routines with TRACE=YES. address space
13	Trace exit point invocations
14	Trace the occurrences of GETDS, RELDS, and SEND FSI requests
15	Trace the occurrences of GETREC, FREEREC, and CHKPT FSI requests
16	Trace the occurrences of connect and disconnect requests between JES2 and the functional subsystem
17	Trace checkpoint performance
18	Trace HASPSSSM $SAVEs in all SSI functions
19	Trace HASPSSSM $RETURNs in all SSI functions
20	Trace $#GET calls
21-255	Currently not used. If you use them, start with 255 and decrease the numbers.

Trace ID = 0 Example

Figure 5-10 is an example that shows Trace ID=0. It shows the number of trace tables in effect, and total and recent discards. TOTAL DISCARDS is the number of entries discarded since tracing was turned on. RECENT DISCARDS is the number of entries discarded because of the current lack of an available table. Be aware that adding the recent number to the total number does not always add up to the next total. The reason is that the recent number is kept in the trace table page and the total number is kept in the $HCCT.



```
14.55.57.96  ID =  19 $RETURN  ASID 0013                                 R14-R1 = 809D6F64 00000000 00000000 020A790B
14.55.57.97  ID =  19 $RETURN  ASID 0012                                 R14-R1 = 809D1F70 00000000 0509FD10 007E9800
14.55.57.97  ID =   1 $SAVE    STC16585   EVTL      02C0D528  $TRACK     R14-R1 = 8002EC9C 000878D8 02D41254 009698B8
14.55.57.97  ID =  19 $RETURN  ASID 0012                                 R14-R1 = 809CB790 00000000 0509FD10 007E9800
14.55.57.97  ID =   2 $RETURN  STC16585   EVTL      02C0D528             R14-R1 = 8002EC9E 000878D8 02D41254 0115460F
14.55.57.97  ID =  18 $SAVE    ASID 0012                      $CBIO      R14-R1 = 809CC264 009D1E70 0509FD05 00000000
14.55.57.97  ID =  18 $SAVE    ASID 0012                      CBIORTN    R14-R1 = 809D1F70 009D21E0 0509FD05 007E9000

14.56.13.09  ID =   0 TRACE EVENTS DISCARDED        TRACE TABLES =   25
                      TOTAL DISCARDS =   819744     RECENT DISCARDS =     231
```

Figure 5-10. TRACE ID=0

Trace ID = 1, ID = 2, ID = 18, ID = 19 Example

Figure 5-11 is an example of trace output for trace IDs 1 ,2, 18, and 19. An explanation of the fields appear after the figure.

In addition to the tracing of $SAVEs and $RETURNs for the routines listed in the figure, $SAVEs and $RETURNs associated with a functional subsystem are also traced.

```
     A           B              C    D      E         F                           G

10.25.31.46  ID =   1 $SAVE               COMM     CWTO      R14-R1 = 401B0EE2 001B0EE8 00000012 00000000
10.25.31.46  ID =   1 $SAVE               COMM     $WTO      R14-R1 = 901B0F2E 001BED58 00208DF0 00208E3C
10.25.31.46  ID =   1 $SAVE    STC  0001  EVTL     $TTIMER   R14-R1 = 401C4102 001A7318 00000000 0020A450
10.25.31.46  ID =   1 $SAVE    STC  0001  EVTL     $TRACK    R14-R1 = 501C52B4 001A7AD2 00227068 0087D080
10.25.31.46  ID =   2 $RETURN  STC  0001  EVTL               R14-R1 = 501C52B4 001A7AD2 00227068 00017102
10.25.31.53  ID =   1 $SAVE    STC  0001  EVTL     $TRACK    R14-R1 = 501C52B4 001A7AD2 00227068 0087D080
10.25.31.53  ID =   2 $RETURN  STC  0001  EVTL               R14-R1 = 501C52B4 001A7AD2 00227068 00016001
10.25.31.66  ID =   1 $SAVE    STC  0001  EVTL     $TRACK    R14-R1 = 501C52B4 001A7AD2 00227068 0087D080
10.25.31.66  ID =   2 $RETURN  STC  0001  EVTL               R14-R1 = 501C52B4 001A7AD2 00227068 00016003
10.25.32.02  ID =   1 $SAVE    STC  0001  EVTL     $JCTIOR   R14-R1 = 501C5262 001A92A4 00000001 00015601
10.25.32.12  ID =   1 $SAVE    STC  0001  EVTL     $WTO      R14-R1 = 401C51EC 001BED58 00000028 0026C058
10.25.32.13  ID =   1 $SAVE    STC  0001  EVTL     $TRACK    R14-R1 = 501C52B4 001A7AD2 00227068 0026B080
10.25.32.13  ID =   2 $RETURN  STC  0001  EVTL               R14-R1 = 501C52B4 001A7AD2 00227068 00015702
10.25.32.13  ID =   1 $SAVE    STC  0001  EVTL     $TRACK    R14-R1 = 501C52B4 001A7AD2 00227068 00A49080
10.25.32.13  ID =   2 $RETURN  STC  0001  EVTL               R14-R1 = 501C52B4 001A7AD2 00227068 00016401
10.25.32.13  ID =   1 $SAVE    STC  0001  EXEC     $GETJLOK  R14-R1 = 401F782A 001A8504 00000018 00227068
10.25.32.13  ID =   1 $SAVE    STC  0001  EXEC     $#BLD     R14-R1 = 601F783E 00000068 0087D0D4 00208830
10.25.32.13  ID =   1 $SAVE    STC  0001  EXEC     $#ADD     R14-R1 = 401F7858 001C1B60 00208830 00208884
10.25.32.13  ID =   1 $SAVE    STC  0001  EXEC     $GETBUF   R14-R1 = 601C2F20 001A7D8E 00000001 00000028
10.25.32.13  ID =   1 $SAVE    STC  0001  EXEC     $JCTIOR   R14-R1 = 501C2F76 001A92A4 00000001 00000000
10.25.32.14  ID =   1 $SAVE               CKPT     $PGFIX    R14-R1 = 801AF176 001A7A2A 000013E9 00220000
10.25.32.14  ID =   1 $SAVE               CKPT     KWRITE    R14-R1 = 901AE548 002540CC FFFFFBE9 00221800
10.25.32.14  ID =   1 $SAVE               CKPT     $PGFIX    R14-R1 = 801AEB6E 001A7A2A 00001000 00247000
10.25.32.16  ID =   1 $SAVE               CKPT     $PGFIX    R14-R1 = 801AEB6E 001A7A2A 00001000 00240000
10.25.32.16  ID =   1 $SAVE               CKPT     KWRITE    R14-R1 = 801AE5CE 00000000 7F000000 7F000000
10.25.32.23  ID =   1 $SAVE    STC  0001  EVTL     $JCTIOR   R14-R1 = 501C5262 001A92A4 00227068 00015601
10.25.32.27  ID =   1 $SAVE    STC  0001  EXEC     $TRACK    R14-R1 = 401C2FE4 001A7AD2 00227068 0087D080
10.25.32.27  ID =   2 $RETURN  STC  0001  EXEC               R14-R1 = 401C2FE4 001A7AD2 00227068 00016002
10.25.32.27  ID =   1 $SAVE    STC  0001  EXEC     $FREEBUF  R14-R1 = 401C30D8 001A8038 00000030 0026D000
10.25.32.27  ID =   1 $SAVE    STC  0001  EXEC     $PGRLSE   R14-R1 = 501A819C 001A79E2 00001000 0026D000
10.25.32.41  ID =   1 $SAVE               CKPT     $PGFREE   R14-R1 = 701AEC6E 001A7A06 00001000 00240000
10.25.32.41  ID =   1 $SAVE               CKPT     $PGFREE   R14-R1 = 701AEC6E 001A7A06 00001000 00247000
10.25.32.41  ID =   1 $SAVE               CKPT     $PGFREE   R14-R1 = 801AEA78 001A7A06 000013E9 00220000
10.25.32.41  ID =   1 $SAVE               CKPT     $TTIMER   R14-R1 = 401AF22C 001A7318 7F000000 002094A0
10.25.32.41  ID =   1 $SAVE               CKPT     $STIMER   R14-R1 = 501AF21A 001A71B6 FFFFE890 002094A0
10.25.32.41  ID =   1 $SAVE               CKPT     $PGFIX    R14-R1 = 801AF176 001A7A2A 000013E9 00220000
10.25.32.41  ID =   1 $SAVE               CKPT     KWRITE    R14-R1 = 901AE548 002540CC FFFFFBE9 00221800
10.25.32.41  ID =   1 $SAVE               CKPT     $PGFIX    R14-R1 = 801AEB6E 001A7A2A 00001000 00247000
10.25.32.42  ID =   1 $SAVE               CKPT     KWRITE    R14-R1 = 801AE5CE 00000000 7F000000 7F000000
10.25.32.51  ID =   1 $SAVE    STC  0001  EVTL     $STIMER   R14-R1 = 601C415E 001A71B6 00000000 0020A450
10.25.32.60  ID =   1 $SAVE    STC  0001  EXEC     $FREJLOK  R14-R1 = 501F7A08 001A857C 00000030 00227068
10.25.32.60  ID =   1 $SAVE               EXEC     $QGET     R14-R1 = 401F6E1E 001A6ACE 0087E000 00000041
10.25.32.60  ID =   1 $SAVE               PRINTR1  $GETBUF   R14-R1 = 401CED96 001A7D8E 00000024 00000020
10.25.32.61  ID =   1 $SAVE               PRINTR1  $#GET     R14-R1 = 501CBB84 001C2468 00000024 0020AE84
10.25.32.61  ID =   1 $SAVE    STC  0001  PRINTR1  $GETBUF   R14-R1 = 401D273C 001A7D8E 00000001 00000068
10.25.32.61  ID =   1 $SAVE    STC  0001  PRINTR1  $PGFIX    R14-R1 = 401A7EC0 001A7A2A 00001000 0026E000
10.25.32.61  ID =   1 $SAVE    STC  0001  PRINTR1  $GETBUF   R14-R1 = 401D2754 001A7D8E 00000001 00000068
10.25.32.61  ID =   1 $SAVE    STC  0001  PRINTR1  $PGFIX    R14-R1 = 401A7EC0 001A7A2A 00001000 0026F000
10.25.32.61  ID =   1 $SAVE    STC  0001  PRINTR1  $GETBUF   R14-R1 = 501D2782 001A7D8E 00000001 00000063
10.25.32.62  ID =   1 $SAVE    STC  0001  PRINTR1  $PGFIX    R14-R1 = 401A7EC0 001A7A2A 00000448 00374448
10.25.32.62  ID =   1 $SAVE    STC  0001  PRINTR1  $PGFIX    R14-R1 = 401D288E 001A7A2A 00000020 0084A334
10.25.32.62  ID =   1 $SAVE    STC  0001  PRINTR1  $#JCTRDR  R14-R1 = 401CBD66 001C32DE 0039452D 00227068
10.25.32.62  ID =   1 $SAVE    STC  0001  PRINTR1  $GETBUF   R14-R1 = 401C33D8 001A7D8E 00000000 00000020
10.25.32.62  ID =   1 $SAVE    STC  0001  PRINTR1  $JCTIOR   R14-R1 = 601C33EA 001A92A4 00000000 00270000
10.25.32.62  ID =   1 $SAVE    JOB  0007  PRINTR2  $#GET     R14-R1 = 601C14EE 001C2468 000000
10.25.32.62  ID =   1 $SAVE    JOB  0007  PRINTR2  $FREUNIT  R14-R1 = 401C150C 001A70BE 000000
10.25.32.74  ID =   1 $SAVE               CKPT     $PGFREE   R14-R1 = 701AEC6E 001A7A06 000010
10.25.32.74  ID =   1 $SAVE               CKPT     $PGFREE   R14-R1 = 801AEA78 001A7A06 000013
```

Figure 5-11. $SAVE/$RETURN Traces

A Time-of-day clock value when the $TRACE was executed.

B Trace identifier

ID = 1 $SAVE - $SAVE trace
ID = 2 $RETURN - $RETURN trace
ID = 18 $SAVE
ID = 19 $RETURN

C Address space identifier or job identifier associated with the program that issued the $TRACE (if available).

D PCE name (if available) associated with the $TRACE.

E PCE address

F EBCDIC name of the routine in which the $SAVE or $RETURN was issued.

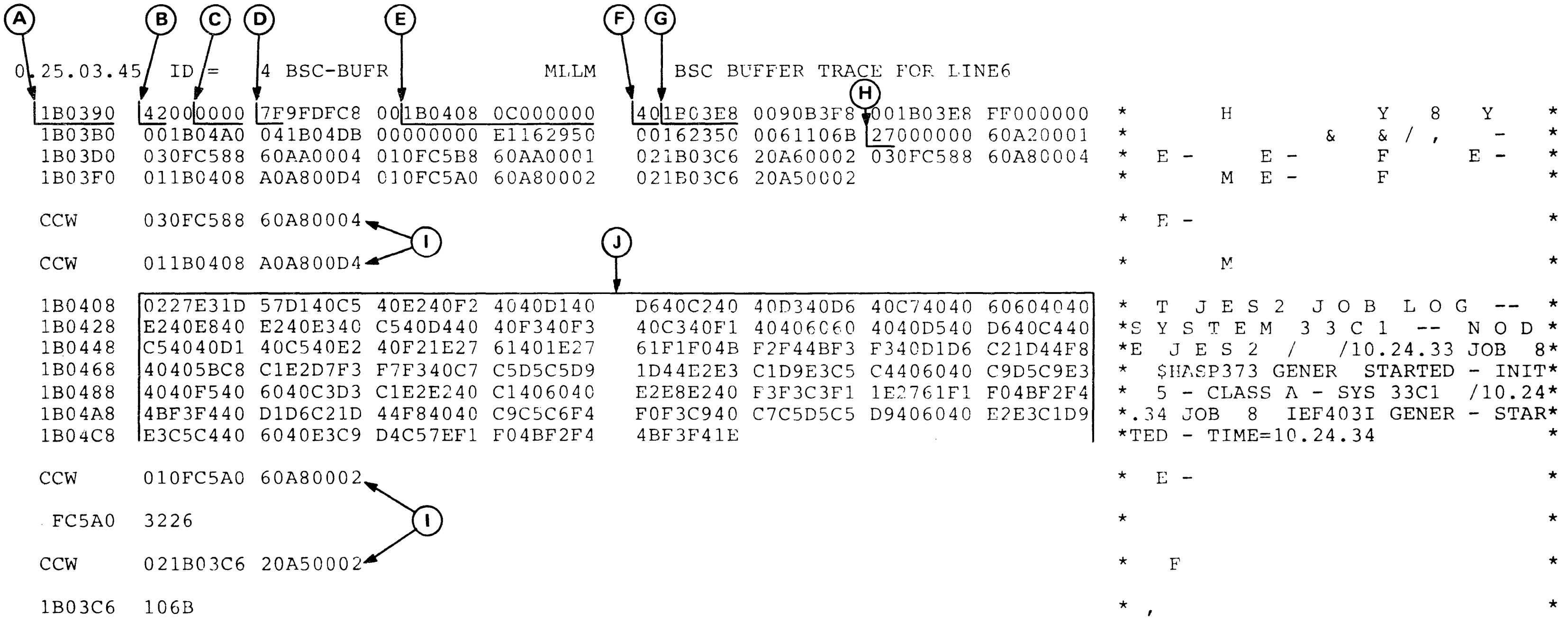
G Contents of registers 14, 15, 0, and 1.

Trace ID=3

Traces with ID=3 trace disastrous error results, including errors associated with a functional subsystem. A trace point in routine FSMCBCK in HASPFSSM traces I/O errors or the encounter of invalid control blocks. The buffer at the time of the error is traced.

Trace ID=4 Example

Figure 5-12 is an example of trace output for trace id=4, a BSC buffer trace. A description of how to interpret the trace output follows the example.



```
0.25.03.45  ID =  4 BSC-BUFR                 MLLM     BSC BUFFER TRACE FOR LINE6
 1B0390   42000000 7F9FDFC8 001B0408 0C000000   401B03E8 0090B3F8 001B03E8 FF000000  *       H          Y  8   Y    *
 1B03B0   001B04A0 041B04DB 00000000 E1162950   00162350 0061106B 27000000 60A20001  *               &  & / ,     -  *
 1B03D0   030FC588 60AA0004 010FC5B8 60AA0001   021B03C6 20A60002 030FC588 60A80004  * E -     E -     F       E -   *
 1B03F0   011B0408 A0A800D4 010FC5A0 60A80002   021B03C6 20A50002                    *       M E -     F             *

 CCW      030FC588 60A80004                                                          * E -                           *

 CCW      011B0408 A0A800D4                                                          *       M                       *

 1B0408   0227E31D 57D140C5 40E240F2 4040D140   D640C240 40D340D6 40C74040 60604040  * T J E S 2  J O B  L O G  --  *
 1B0428   E240E840 E240E340 C540D440 40F340F3   40C340F1 40406060 4040D540 D640C440  *S Y S T E M  3 3 C 1  --  N O D*
 1B0448   C54040D1 40C540E2 40F21E27 61401E27   61F1F04B F2F44BF3 F340D1D6 C21D44F8  *E  J E S 2  /  /10.24.33 JOB  8*
 1B0468   40405BC8 C1E2D7F3 F7F340C7 C5D5C5D9   1D44E2E3 C1D9E3C5 C4406040 C9D5C9E3  *  $HASP373 GENER  STARTED - INIT*
 1B0488   4040F540 6040C3D3 C1E2E240 C1406040   E2E8E240 F3F3C3F1 1E2761F1 F04BF2F4  *  5 - CLASS A - SYS 33C1  /10.24*
 1B04A8   4BF3F440 D1D6C21D 44F84040 C9C5C6F4   F0F3C940 C7C5D5C5 D9406040 E2E3C1D9  *.34 JOB  8  IEF403I GENER - STAR*
 1B04C8   E3C5C440 6040E3C9 D4C57EF1 F04BF2F4   4BF3F41E                             *TED - TIME=10.24.34            *

 CCW      010FC5A0 60A80002                                                          * E -                           *

 FC5A0    3226                                                                       *                               *

 CCW      021B03C6 20A50002                                                          *  F                            *

 1B03C6   106B                                                                       * ,                             *
```

Figure 5-12. BSC Buffer Trace

This trace can assist you in diagnosing BSC problems. Both inbound and outbound buffers are traced when this trace is activated. Both network job entry (NJE) lines and remote job entry (RJE) lines can be traced. A breakdown of the trace output follows.

The input to this trace point is the buffer which contains the IOB. The IOB contains the channel program that executed on the line being traced. Refer to *Debugging Handbook* for more details about the IOB. See *Principles of Operation* for more information about CCWs and CSWs.

A In this column is the address of the IOB or data being traced. You will see trace entries with the same addresses as they are reused.

B I/O completion flags (IOBFLAG1). Successful completion = X'42' Error = X'46'.

C First 2 sense bytes of the I/O operation. See sense information for individual devices (IOBSENS1 and IOBSENS2).

D ECB completion codes (IOBECBCC) and the HASP ECB pointer (IOBECBPT). Common completion codes are:

X'41'	-- I/O error. Check IOBSENS1 and IOBSENS2
X'44'	-- IOB intercept. CSW is not valid
X'48'	-- I/O was purged
X'7F'	-- Normal completion
X'FF'	-- Requeued BSC buffer (a JES2 only value)

E This is the CSW without its first byte. It starts at !OB+9 and is 7 bytes in length. The address in the first 3 bytes points 8 bytes past the last CCW executed. The CSW count field is the residual date count of the CCW that completed. The actual number of bytes read or written is the difference between this value and the CCW byte count in the CCW just completed.

F SIO condition code (IOBSIOCC), successful in this instance (40 = CSW stored).

G Address of the start of the channel program (IOBSTART).

H Channel programs (IOBCCW1 through IOBCCW8)

I CCW entry. These are the CCWs from the channel program. It starts at X'1B03E8' with a NOP CCW and ends at X'1B0400' with a READ CCW. The READ or WRITE data associated with the CCW follows the CCW entry.

J This is the data that was associated with the WRITE CCW located at X'1B03F0'. When this trace entry was made, this data was located at X'1B0408' and was X'D4' bytes in length, as indicated by the CCW byte count in the WRITE CCW that precedes it.

See Figure 3-7 for an explanation of byte 5 of the CCW (TP op code).

Refer to the description of message $HASP094 in *JES2 Messages* for additional information about some of the fields listed above.

Trace ID = 14 Example

The following figures are examples of trace output for trace id = 14, a GETDS, RELDS, ORDER, ORDER RESPONSE, CHKPT, and SEND FSI request trace. A description of how to interpret the trace output follows the examples. The letters in the explanation correspond to letters in the figure.

In addition to providing you with the contents of registers 14, 15, 0, and 1 and the order parameter list, these traces provide:

DCT fields:

> DCTFLAGS, DCTSTAT, DCTFSSFL, DCTFLAG1

FSSCB fields:

> FSSFLAG1, FSSFLAG2, FSSFLAG3

FSACB:

> The FSACB is traced if one exists. Only FSACB flags are traced from the common order routine in HASPFSSP. The entire FSACB is traced from the order cross memory and response routines in HASPFSSP.

FSIP:

> The FSIP is traced if a FSACB exists.

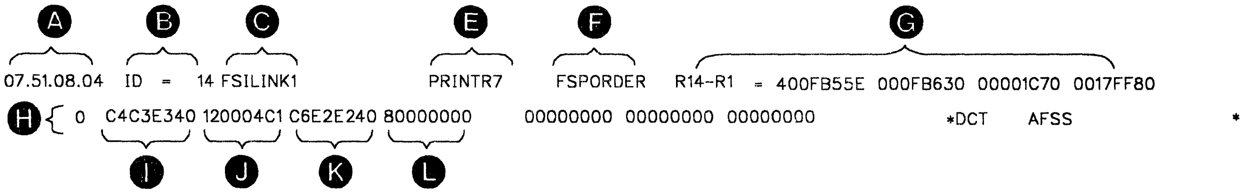
TRACE ID = 14 trace points for ORDERS exist in HASPFSSP, FSI ORDER (FSPORDER) and ORDER CROSS MEMORY (FSPORCMS) routines. These routines call HASPFSSM to issue an FSI order or service request.

The functional subsystem interface parameter list (FSIP) is traced only in the ORDER CROSS MEMORY routine.

TRACE ID = 14 trace points for ORDER RESPONSES exist in HASPFSSP RESPONSE routines.

TRACE ID = 14 trace points for GET DATASET (GETDS), RELEASE DATASET (RELDS), CHECKPOINT (CHKPT) and SEND FSI REQUEST exist in HASPFSSM.

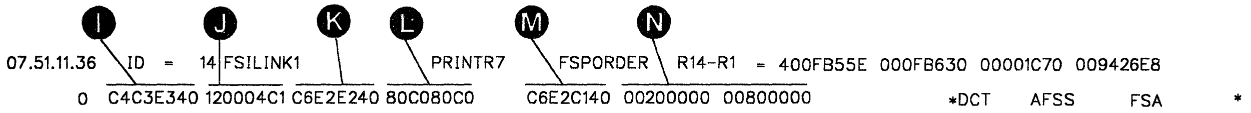
Figure 5-13 is an example of trace output for trace id = 14, an FSPORDER trace, when an FSA is not active. Note the absence of the FSA eyecatcher and FSA flag bytes of zeroes.



```
07.51.08.04  ID  =  14 FSILINK1          PRINTR7    FSPORDER  R14--R1  =  400FB55E 000FB630 00001C70 0017FF80
   0  C4C3E340 120004C1 C6E2E240 80000000      00000000 00000000 00000000            *DCT    AFSS             *
```

Figure 5-13. An FSPORDER Trace When an FSA Is Not Active

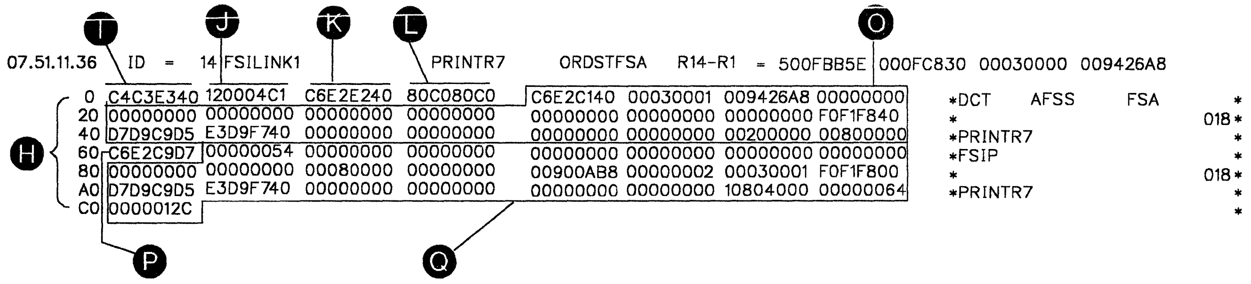
Figure 5-14 is an example of trace output for trace ID=14, an FSPORDER trace, when an FSA is active.



```
07.51.11.36  ID  =   14 FSILINK1              PRINTR7      FSPORDER   R14-R1  = 400FB55E 000FB630 00001C70 009426E8
          0  C4C3E340 120004C1 C6E2E240 80C080C0   C6E2C140 00200000 00800000           *DCT    AFSS     FSA      *
```

Figure 5-14. An FSPORDER Trace When an FSA Is Active

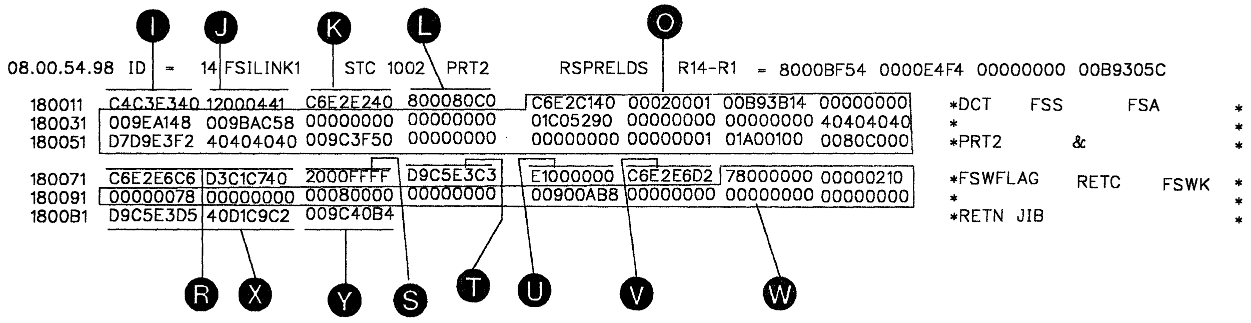
Figure 5-15 is an example of trace output for trace ID=14, an FSPORCMS trace.



```
07.51.11.36  ID  =   14 FSILINK1              PRINTR7      ORDSTFSA   R14-R1  = 500FBB5E 000FC830 00030000 009426A8
          0  C4C3E340 120004C1 C6E2E240 80C080C0   C6E2C140 00030001 009426A8 00000000  *DCT    AFSS     FSA      *
         20  00000000 00000000 00000000 00000000   00000000 00000000 00000000 F0F1F840  *                      018*
         40  D7D9C9D5 E3D9F740 00000000 00000000   00000000 00000000 00200000 00800000  *PRINTR7                  *
         60  C6E2C9D7 00000054 00000000 00000000   00000000 00000000 00000000 00000000  *FSIP                     *
         80  00000000 00000000 00080000 00000000   00900AB8 00000002 00030001 F0F1F800  *                      018*
         A0  D7D9C9D5 E3D9F740 00000000 00000000   00000000 00000000 10804000 00000064  *PRINTR7                  *
         C0  0000012C                                                                                             *
```

Figure 5-15. An FSPORCMS Trace

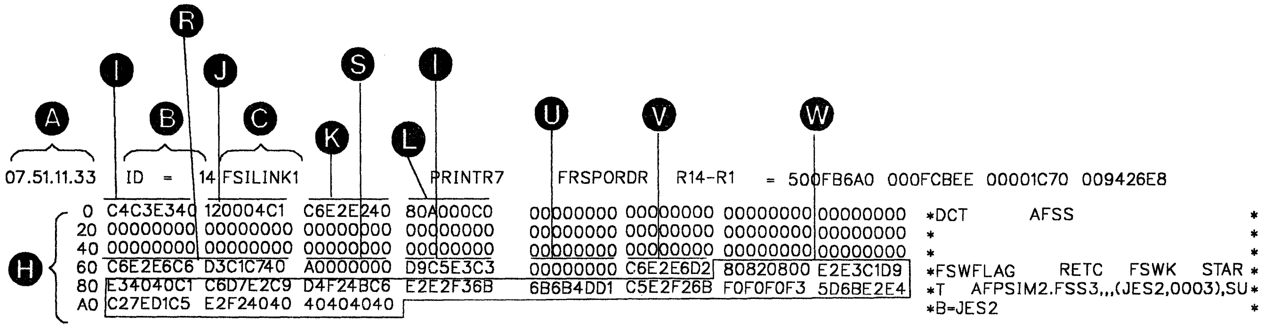
Figure 5-16 is an example of trace output for trace ID=14, an FRSPORDR trace from the RELDS response routine.



```
08.00.54.98  ID  =   14 FSILINK1   STC  1002 PRT2        RSPRELDS   R14-R1  = 8000BF54 0000E4F4 00000000 00B9305C
     180011  C4C3E340 12000441 C6E2E240 800080C0   C6E2C140 00020001 00B93B14 00000000  *DCT    FSS      FSA      *
     180031  009EA148 009BAC58 00000000 00000000   01C05290 00000000 00000000 40404040  *                         *
     180051  D7D9E3F2 40404040 009C3F50 00000000   00000000 00000001 01A00100 0080C000  *PRT2      &              *

     180071  C6E2E6C6 D3C1C740 2000FFFF D9C5E3C3   E1000000 C6E2E6D2 78000000 00000210  *FSWFLAG  RETC    FSWK    *
     180091  00000078 00000000 00080000 00000000   00900AB8 00000000 00000000 00000000  *                         *
     1800B1  D9C5E3D5 40D1C9C2 009C40B4                                                 *RETN JIB                 *
```

Figure 5-16. An FRSPORDR Trace from the RELDS Response Routine

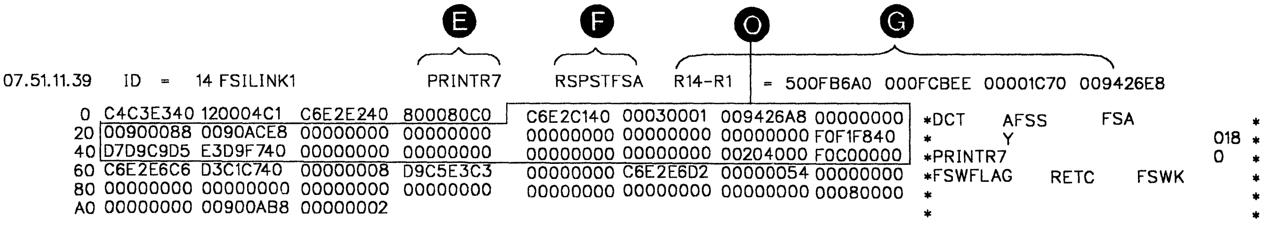
Figure 5-17 is an example of trace output for trace ID=14, an FRSPORDR trace when an FSA is not active.



```
07.51.11.33  ID  =   14 FSILINK1              PRINTR7      FRSPORDR   R14-R1  = 500FB6A0 000FCBEE 00001C70 009426E8
          0  C4C3E340 120004C1 C6E2E240 80A000C0   00000000 00000000 00000000 00000000  *DCT      AFSS                *
         20  00000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000  *                             *
         40  00000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000  *                             *
         60  C6E2E6C6 D3C1C740 A0000000 D9C5E3C3   00000000 C6E2E6D2 80820800 E2E3C1D9  *FSWFLAG    RETC   FSWK   STAR*
         80  E34040C1 C6D7E2C9 D4F24BC6 E2E2F36B   6B6B4DD1 C5E2F26B F0F0F0F3 5D6BE2E4  *T  AFPSIM2.FSS3,,,(JES2,0003),SU*
         A0  C27ED1C5 E2F24040 40404040                                                 *B=JES2                       *
```

Figure 5-17. An FRSPORDR Trace When an FSA Is Not Active

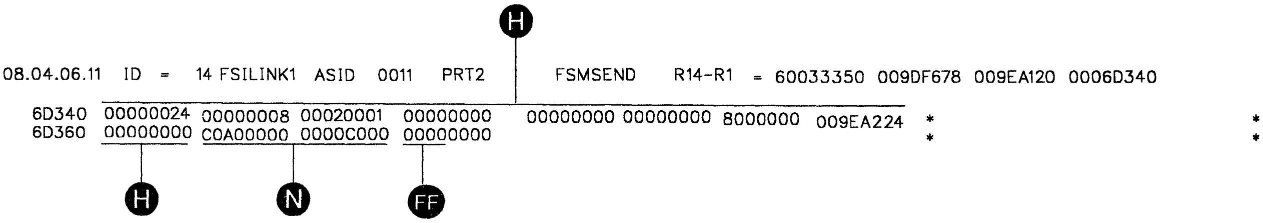
Figure 5-18 is an example of trace output for trace id=14, an FRSPORDR trace when an FSA is active.



```
07.51.11.39   ID  =   14 FSILINK1              PRINTR7      RSPSTFSA    R14-R1  = 500FB6A0 000FCBEE 00001C70 009426E8

  0 C4C3E340 120004C1 C6E2E240 800080C0    C6E2C140 00030001 009426A8 00000000  *DCT   AFSS      FSA          *
 20 00900088 0090ACE8 00000000 00000000    00000000 00000000 00000000 F0F1F840  *       Y                018 *
 40 D7D9C9D5 E3D9F740 00000000 00000000    00000000 00000000 00204000 F0C00000  *PRINTR7                  0 *
 60 C6E2E6C6 D3C1C740 00000008 D9C5E3C3    00000000 C6E2E6D2 00000054 00000000  *FSWFLAG    RETC     FSWK     *
 80 00000000 00000000 00000000 00000000    00000000 00000000 00000000 00080000  *                            *
 A0 00000000 00900AB8 00000002                                                  *                            *
```

Figure 5-18. An FRSPORDR Trace When an FSA Is Active

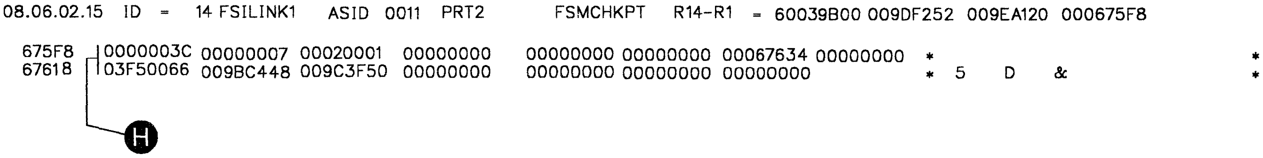
Figure 5-19 is an example of trace output for trace id = 14, an FSISEND trace.



```
08.04.06.11   ID  =   14 FSILINK1  ASID  0011  PRT2        FSMSEND   R14-R1  = 60033350 009DF678 009EA120 0006D340

 6D340  00000024 00000008 00020001 00000000    00000000 00000000 8000000  009EA224  *                            *
 6D360  00000000 C0A00000 0000C000 00000000                                         *                            *
```

Figure 5-19. An FSISEND Trace

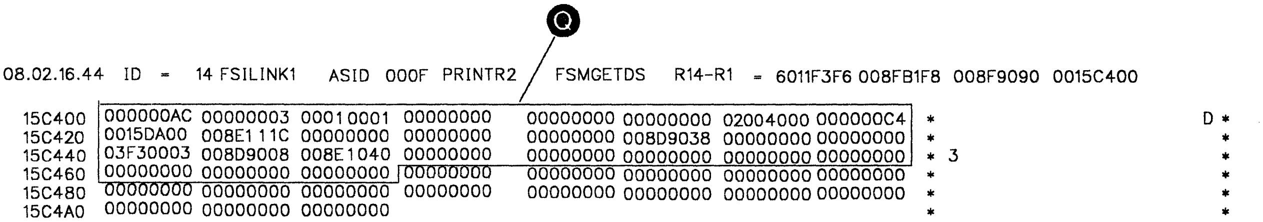
Figure 5-20 is an example of trace output for trace id = 14, an FSICKPT trace.



```
08.06.02.15   ID  =   14 FSILINK1   ASID  0011  PRT2       FSMCHKPT   R14-R1  = 60039B00 009DF252 009EA120 000675F8

 675F8   0000003C 00000007 00020001 00000000    00000000 00000000 00067634 00000000  *                            *
 67618   03F50066 009BC448 009C3F50 00000000    00000000 00000000 00000000           *  5     D     &             *
```

Figure 5-20. An FSICKPT Trace

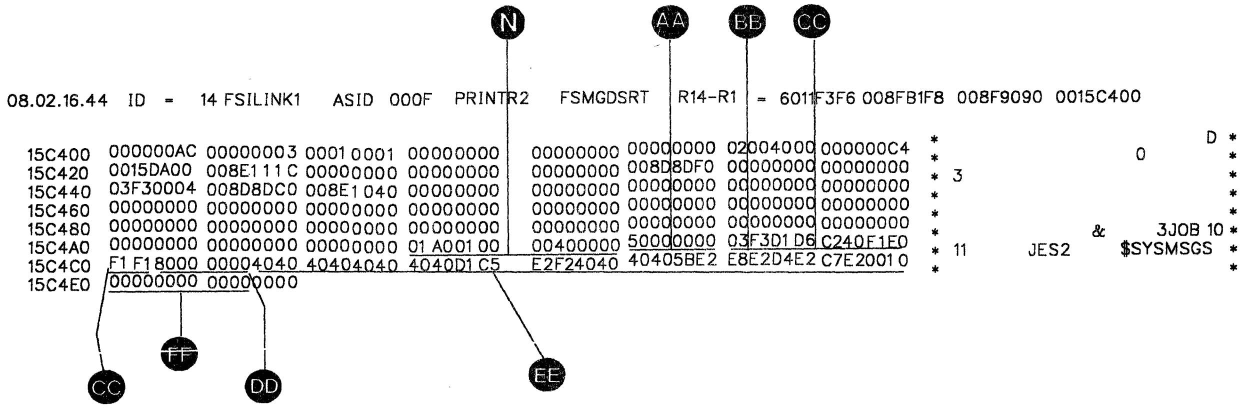
Figure 5-21 is an example of trace output for trace id = 14, a GETDS, RELDS, SEND FSI request trace. This trace is taken before a data set is allocated and contains only the GETDS FSIP.



```
08.02.16.44  ID  =   14 FSILINK1   ASID  000F  PRINTR2   FSMGETDS   R14-R1  = 6011F3F6 008FB1F8 008F9090 0015C400

 15C400  000000AC 00000003 00010001 00000000    00000000 00000000 02004000 000000C4  *                         D *
 15C420  0015DA00 008E111C 00000000 00000000    00000000 008D9038 00000000 00000000  *                           *
 15C440  03F30003 008D9008 008E1040 00000000    00000000 00000000 00000000 00000000  * 3                         *
 15C460  00000000 00000000 00000000 00000000    00000000 00000000 00000000 00000000  *                           *
 15C480  00000000 00000000 00000000 00000000    00000000 00000000 00000000 00000000  *                           *
 15C4A0  00000000 00000000 00000000                                                  *                           *
```

Figure 5-21. A GETDS, RELDS, SEND FSI Request Trace Before Data Set Allocation

Figure 5-22 is an example of trace output for trace id = 14, a GETDS, RELDS, SEND FSI request trace. This trace is taken only if a data set has been allocated, and contains the FSIP, FSA flags, JIB job number, JIB job ID, GCB flags GCBDSPN.



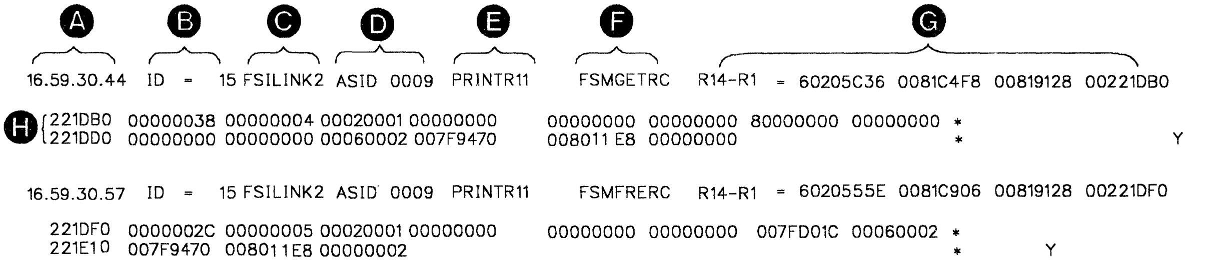
```
08.02.16.44  ID  =  14 FSILINK1   ASID  000F   PRINTR2    FSMGDSRT    R14-R1  =  6011F3F6 008FB1F8  008F9090  0015C400

 15C400  000000AC 00000003 0001 0001 00000000   00000000 00000000 02004000 000000C4  *                         D *
 15C420  0015DA00 008E111C 00000000 00000000    00000000 008D8DF0 00000000 00000000  *                  0        *
 15C440  03F30004 008D8DC0 008E1040 00000000    00000000 00000000 00000000 00000000  * 3                         *
 15C460  00000000 00000000 00000000 00000000    00000000 00000000 00000000 00000000  *                            *
 15C480  00000000 00000000 00000000 00000000    00000000 00000000 00000000 00000000  *                            *
 15C4A0  00000000 00000000 00000000 01 A001 00  00400000 50000000 03F3D1 D6 C240F1E0 *           &       3JOB 10 *
 15C4C0  F1 F18000 00004040 40404040 4040D1 C5   E2F24040 40405BE2 E8E2D4E2 C7E2001 0 * 11     JES2     $SYSMSGS   *
 15C4E0  00000000 00000000                                                           *                            *
```

Figure 5-22. A GETDS, RELDS, SEND FSI Request Trace After Data Set Allocation

Trace ID = 15 Example

Figure 5-23 is an example of trace output for trace id = 15, a GETREC and FREEREC FSI request trace. A description of how to interpret the trace output follows the example. The letters in the explanation correspond to letters in the figure.

A trace point also exists in FRSPORDR. This routine processes responses to orders. This trace point will give you the response area that is mapped by the IAZRESPA macro, and flags FSAFLAG and FSAFLAGS in the FSACB.

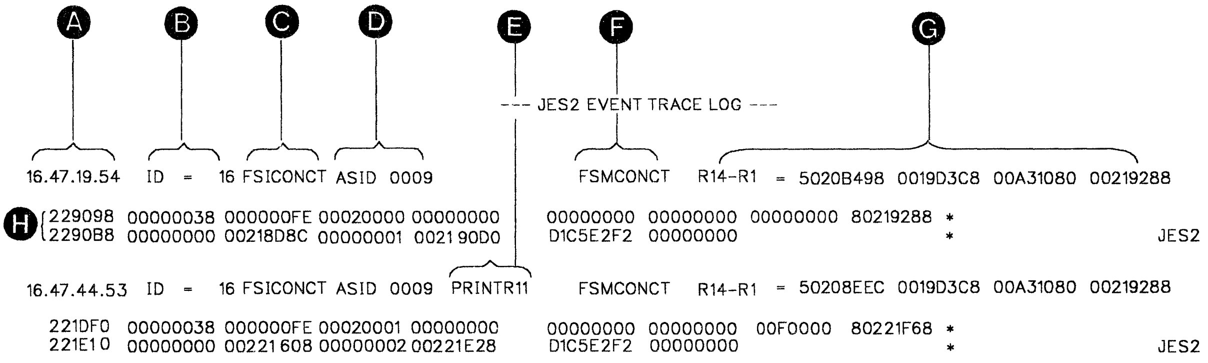


```
16.59.30.44  ID  =   15 FSILINK2  ASID  0009   PRINTR11      FSMGETRC   R14-R1  = 60205C36  0081C4F8  00819128  00221DB0

 {221DB0  00000038 00000004 00020001 00000000        00000000  00000000  80000000  00000000  *
 {221DD0  00000000 00000000 00060002 007F9470        008011 E8  00000000                      *                    Y

16.59.30.57  ID  =   15 FSILINK2  ASID  0009   PRINTR11      FSMFRERC   R14-R1  = 6020555E  0081C906  00819128  00221DF0

  221DF0  0000002C 00000005 00020001 00000000        00000000  00000000   007FD01C  00060002  *
  221E10  007F9470  008011E8 00000002                                                          *         Y
```

Figure 5-23. GETREC/FREEREC Trace

Trace ID = 16 Example

Figure 5-24 is an example of trace output for trace id = 16, a CONNECT and DISCONNECT FSI request trace. A description of how to interpret the trace output follows the example. The letters in the explanation correspond to letters in the figure.



```
                                               --- JES2 EVENT TRACE LOG ---

16.47.19.54   ID  =   16 FSICONCT  ASID  0009                  FSMCONCT   R14-R1  = 5020B498  0019D3C8  00A31080  00219288

 {229098  00000038  000000FE 00020000 00000000       00000000  00000000  00000000  80219288  *
 {2290B8  00000000  00218D8C  00000001  0021 90D0     D1C5E2F2  00000000                      *                    JES2

16.47.44.53   ID  =   16 FSICONCT  ASID  0009  PRINTR11      FSMCONCT   R14-R1  = 50208EEC  0019D3C8  00A31080  00219288

  221DF0  00000038  000000FE 00020001  0000000C       00000000  00000000   00F0000   80221F68  *
  221E10  00000000  00221608 00000002 00221E28        D1C5E2F2  00000000                      *                    JES2
```

Figure 5-24. CONNECT/DISCONNECT Trace

Trace ID = 14, 15, 16 Descriptions

A Time-of-day clock value when the $TRACE was executed.

B Trace identifier

ID = 14 for GETDS/RELDS/ORDER/RESPONSE/CHKPT trace.

ID = 15 for GETREC/FREREC trace.

ID = 16 for CONNECT/DISCONNECT trace.

C Functional subsystem interface function name.

FSILINK1

FSILINK2

FSICONCT

D Address space identifier associated with the functional subsystem.

E Device name associated with the functional subsystem.

F Trace symbolic name associated with the trace entry.

ORDSTFSA for START FSA ORDER trace

ORDSTDEV for START DEVICE ORDER trace

ORDSPDEV for STOP DEVICE ORDER trace

ORDSPFSA for STOP FSA ORDER trace

ORDSPFSS for STOP FSS ORDER trace

ORDQUERY for QUERY ORDER trace

ORDSET for SET ORDER trace

ORDINTV for INTERVENTION ORDER trace

ORDSYNC for SYNCH ORDER trace

RSPSTFSA for START FSA RESPONSE trace

RSPSTDEV for START DEVICE RESPONSE trace

RSPSPDEV for STOP DEVICE RESPONSE trace

RSPSPFSA for STOP FSA RESPONSE trace

RSPSPFSS for STOP FSS RESPONSE trace

RSPQUERY for QUERY RESPONSE trace

RSPSET for SET RESPONSE trace

RSPINTV for INTERVENTION RESPONSE trace

RSPSYNC for SYNCH RESPONSE trace

RSPRELDS for RELDS RESPONSE trace

G Contents of registers 14, 15, 0, 1.

H Trace FSI parameter, flags and miscellaneous data area.

I DCT eyecatcher (to help you locate DCT flags in trace data).

J DCT flags. The data is four bytes, containing DCTSTAT, DCTFLAGS, DCTFLAG2, and DCTFSSL.

K FSS eyecatcher (to help you locate FSSCB flags in trace data).

L FSSCB flags. The data is four bytes, containing FSSTYPE, FSSFLAG1, FSSFLAG2, and FSSFLAG3.

M FSA eyecatcher (to help you locate FSACB flags in trace data.

N FSACB flags. The data is eight bytes, containing FSAFLAG1, FSAFLAG2, FSAFLAG3, FSAFLAGO, FSAFLAGI, and FSAFLAGR. The remaining two bytes are not used.

O FSACB. The FSACB is X'50' bytes in length and traced in its entirety.

P FSIP eyecatcher (to help you locate the FSIP in trace data).

Q Functional Subsystem Interface Parameter List (IAZFSIP). The length of the FSIP is variable. The length of IAZFSIP (and the length of the data traced) can be found in the low order positions of the first four bytes of trace data.

R FSWFLAG eyecatcher (to help you locate the FSWFLAGS trace data)

S Functional Subsystem Support Processor PCE Work Area (FSSWORK) flags. The data is four bytes in length and contains FSWFLAG, a reserved byte, and FSWORDID.

T RETC eyecatcher (to help you locate RESPRETC in trace data).

U Return Code of requested function. The data is four bytes in length and contains the RESPRETC field of IAZRESPA.

V FSWK eyecatcher (to help you locate the FSWRK area in trace data)

W Functional Subsystem Work Area (located in FSSWORK). The data is X'3C' bytes in length and contains the response to an order.

X RETN JIB eyecatcher (to help you locate a pointer to the JIB (JOE information block) being returned in the trace data).

Y The address of the JIB being returned. The data is four bytes in length and contains the address of the JIB that is being removed from the FSACB return JIB stack (FSARETQS).

AA JIB flags. The data is four bytes in length and contains JIBFLG1, JIBFLG2, JIBFLG3, and a reserved byte.

BB JIB jobnumber. The data is two bytes in length and contains the HASP jobnumber (JQEJOBNO) in hexadecimal.

CC JIB jobid. The data is eight bytes in length and contains the HASP job identifier (JOB, STC, or TSU) and the decimal equivalent of the JIB jobnumber in EBCDIC.

DD GCB flags. The data is four bytes in length and contains the GCB flags.

EE GCB Dataset ID. The data is 32 bytes in length and contains the GCB fields: GCBDSPN (dataset proc. name), GCBDSSN (dataset step name), GCBDSDD (dataset ddname), and GCBPRIO (dataset priority).

FF FSA JOE Count. The data is two bytes in length and contains a count of the JOEs assigned to this FSA.

Checkpoint Tracing

$TRACE ID 17 (CKPTPERF) provides information about checkpoint performance. Each record in the trace output has eight 8-byte fields separated by blanks. The formatted output records appear in groups of at most 5 (more if the user has added CTENTS). The first field in the first record of each group contains one of the following values, and determines the meaning of the contents of the rest of the fields:

- READ 1 -- this means that this group (1 record long) describes information about the first read for this checkpoint cycle.
- READ 2 -- this means that this group (5 records long) describes information about the second read for this checkpoint cycle.

- PRIMARY -- this means that this group (5 records long) describes information about the primary write for this checkpoint cycle.
- INTERMED -- this means that this group (5 records long) describes information about the intermediate write for the checkpoint cycle.
- FINAL -- this means that this group (5 records long) describes information about the final write for this checkpoint cycle.

The meaning of the fields in the various records for the different groups is described in the following sections.

READ 1

Record 1:

- Field 1: Contains the EBCDIC value "READ 1 ".
- Field 2: Contains the time (in tenths of milliseconds) that passed from the $EXCP to start the Read1 I/O until the I/O completed and the CKPT PCE got dispatched.
- Field 3: Contains the number of used pages in the change log.
- Field 4: Contains the current value of MINHOLD.
- Field 5: Contains the current value of MINDORM.
- Field 6: Contains the current value of MAXDORM.
- Field 7: Contains the number of change log records read in Read 1.
- Field 8: Contains the EBCDIC name ("CKPT1 " or "CKPT2 " of the data set that contained the current copy of the queues when Read 1 was performed.

READ 2

Record 1:

- Field 1: Contains the EBCDIC value "READ 2 ".
- Field 2: Contains the time (in tenths of milliseconds) that passed from the $EXCP to start the Read2 I/O until the I/O completed and the CKPT PCE got dispatched.
- Field 3: Contains the number of pages in the change log (total pages, not used pages).
- Field 4: Contains the number of control blocks in the change log.
- Field 5: Contains the number of PCEs defined to this system.
- Field 6: Contains the number of PCEs that are $WAITing for access to the checkpoint.
- Field 7: Contains the maximum length of time (in tenths of milliseconds) that a PCE was $WAITing for access to the checkpoint.
- Field 8: Contains the average length of time (in tenths of milliseconds) that the PCEs were $WAITing for access to the checkpoint.

Record 2:

- Field 1: Contains the number of used bytes in the change log.
- Field 2: Contains the length of time (in tenths of milliseconds) that this system did not hold the checkpoint.
- Field 3: Contains the number of pages which would have been read if the complex had been in Duplex mode. (The field is meaningless if the complex is in Duplex mode.) This field may be low if the change log overflows.
- Field 4: Contains the EBCDIC name ("CKPT1 " or "CKPT2 ") of the data set Read 2 was performed against.

Record 3:

- Field 1: Contains the number of pages read for the first CTENT.
- Field 2: Contains the number of control blocks in the change log for the first CTENT.
- Field 3: Contains the number of pages read for the second CTENT.
- Field 4: Contains the number of control blocks in the change log for the second CTENT.
- Field 5: Contains the number of pages read for the third CTENT.
- Field 6: Contains the number of control blocks in the change log for the third CTENT.
- Field 7: Contains the number of pages read for the fourth CTENT.
- Field 8: Contains the number of control blocks in the change log for the fourth CTENT.

Record 4-N: These records have the same format as Record 3, but for the fifth through eighth, etc., CTENTs.

PRIMARY WRITE

Record 1:

- Field 1: Contains the EBCDIC value 'PRIMARY'.
- Field 2: Contains the time (in tenths of milliseconds) that passed from the $EXCP to start the primary write I/O until the I/O completed and the CKPT PCE was dispatched.
- Field 3: Contains the number of used pages in the change log.
- Field 4: Contains the number of control blocks in the change log.
- Field 5: Contains the number of PCEs defined to this system.
- Field 6: Contains the number of PCEs that are $WAITing for checkpoint write completion.
- Field 7: Contains the maximum length of time (in tenths of milliseconds) that a PCE was $WAITing for checkpoint write completion.
- Field 8: Contains the average length of time (in tenths of milliseconds) that the PCEs were $WAITing for checkpoint write completion.

Record 2:

- Field 1: Contains the number of used bytes in the change log.
- Field 2: Contains the EBCDIC value 'PRIO AGE' if priority aging contributed to this write
- Field 3: Contains the number of times the Checkpoint PCE put itself at the bottom of the Ready queue before performing this write
- Field 4: Contains the number of pages which would have been written if the complex had been in Duplex mode. (This field is meaningless if the complex is in Duplex mode.) This field may be low if the change log overflows.
- Field 5: Contains the level number of the data set.
- Field 6: Contains the EBCDIC name ('CKPT1 ' or 'CKPT2 ') of the data set the primary write was performed against.

Record 3:

- Field 1: Contains the number of pages written for the first CTENT.
- Field 2: Contains the number of control blocks in the change log for the first CTENT.
- Field 3: Contains the number of pages written for the second CTENT.
- Field 4: Contains the number of control blocks in the change log for the second CTENT.

- Field 5: Contains the number of pages written for the third CTENT.
- Field 6: Contains the number of control blocks in the change log for the third CTENT.
- Field 7: Contains the number of pages written for the fourth CTENT.
- Field 8: Contains the number of control blocks in the change log for the fourth CTENT.

Record 4-N: these records have the same format as Record 3, but for the fifth through eighth, etc., CTENTs.

INTERMEDIATE WRITE

Record 1:

- Field 1: Contains the EBCDIC value 'INTERMED'.
- Field 2: Contains the time (in tenths of milliseconds) that passed from the $EXCP to start the Intermediate Write I/O until the I/O completed and the CKPT PCE got dispatched.
- Field 3: Contains the number of used pages in the change log.
- Field 4: Contains the number of control blocks in the change log.
- Field 5: Contains the number of PCEs defined to this system.
- Field 6: Contains the number of PCEs that are $WAITing for checkpoint write completion.
- Field 7: Contains the maximum length of time (in tenths of milliseconds) that a PCE was $WAITing for checkpoint write completion.
- Field 8: Contains the average length of time (in tenths of milliseconds) that the PCEs were $WAITing for checkpoint write completion.

Record 2:

- Field 1: Contains the number of used bytes in the change log.
- Field 2: Contains the EBCDIC value 'PRIO AGE' if priority aging contributed to this write.
- Field 3: Contains the number of times the checkpoint PCE put itself at the bottom of the Ready queue before performing this write.
- Field 4: Contains the number of pages which would have been written if the complex had been in Duplex mode. (This field is meaningless if the complex is in Duplex mode.) This field may be low if the change log overflows.
- Field 5: Contains the level number of the data set.
- Field 6: Contains the EBCDIC name ('CKPT1 ' or 'CKPT2 ') of the data set the intermediate write was performed against.

Record 3:

- Field 1: Contains the number of pages written for the first CTENT.
- Field 2: Contains the number of control blocks in the change log for the first CTENT.
- Field 3: Contains the number of pages written for the second CTENT.
- Field 4: Contains the number of control blocks in the change log for the second CTENT.
- Field 5: Contains the number of pages written for the third CTENT.
- Field 6: Contains the number of control blocks in the change log for the third CTENT.
- Field 7: Contains the number of pages written for the fourth CTENT.
- Field 8: Contains the number of control blocks in the change log for the fourth CTENT.

Record 4-N: these records have the same format as Record 3, but for the fifth through eighth, etc., CTENTs.

FINAL WRITE

Record 1:

- Field 1: Contains the EBCDIC value 'FINAL   '.
- Field 2: Contains the time (in tenths of milliseconds) that passed from the $EXCP to start the final write I/O until the I/O completed and the CKPT PCE got dispatched.
- Field 3: Contains the number of used pages in the change log.
- Field 4: Contains the number of control blocks in the change log.
- Field 5: Contains the number of PCEs defined to this system.
- Field 6: Contains the number of PCEs that are $WAITing for checkpoint write completion.
- Field 7: Contains the maximum length of time (in tenths of milliseconds) that a PCE was $WAITing for checkpoint write completion.
- Field 8: Contains the average length of time (in tenths of milliseconds) that the PCEs were $WAITing for checkpoint write completion.

Record 2:

- Field 1: Contains the number of used bytes in the change log.
- Field 2: Contains the length of time (in tenths of milliseconds) that this system held the checkpoint.
- Field 3: Contains the EBCDIC value 'PRIO AGE' if priority aging contributed to this write.
- Field 4: Contains the number of times the checkpoint PCE put itself at the bottom of the Ready queue before performing this write
- Field 5: Contains the number of pages which would have been written if the complex had been in Duplex mode. (This field is meaningless if the complex is in Duplex mode.) The field may be low if the change log overflows.
- Field 6: Contains the EBCDIC name ('CKPT1   ' or 'CKPT2   ') of the data set the final write was performed against.

Record 3:

- Field 1: Contains the number of pages written for the first CTENT.
- Field 2: Contains the number of control blocks in the change log for the first CTENT.
- Field 3: Contains the number of pages written for the second CTENT.
- Field 4: Contains the number of control blocks in the change log for the second CTENT.
- Field 5: Contains the number of pages written for the third CTENT.
- Field 6: Contains the number of control blocks in the change log for the third CTENT.
- Field 7: Contains the number of pages written for the fourth CTENT.
- Field 8: Contains the number of control blocks in the change log for the fourth CTENT.

Record 4-N: these records have the same format as Record 3, but for the fifth through eighth, etc., CTENTs.

$TRACE ID = 20

This trace ID traces all $#GET calls made by devices, including local and remote print and punch devices, spool offload SYSOUT transmitters, and NJE SYSOUT transmitters. It also provides the job number, the class, and route code of the output selected. It provides counts such as the number of elements searched before work is found, the total number of elements, and the number of elements in use. NJE transmitters also run JQE-JOE chains and display the count of JOEs on the chain and the count of those selected. The trace record also contains the work selection list. Because NJE does not use the work selection services, the WS list for NJE devices is always null.

This record provides a means to analyze selection criteria and queue search overhead to tune work selection criteria.

The format of the trace record is, in three cases, as follows:

1. **The Trace Record for a PRPU Processor**

```
ID = 20  $#GET   JOB 1234   PRT2   $#GET CALL FOR PRT2
WS = (W,R,Q,PRM/UCS,P)
OUTGRPS DEFINED = nnnn     OUTGRPS IN USE  = nnnn
OUTGRPS SCANNED = nnnn     OUTGRPS THRU WS = nnnn
OUTGRP MASK = FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFF
CLASS = A     ROUTE = 00010000   FLAGS= 20A48000
```

Note that:

a. The class and route code belong to the JOE that has been selected.

b. The outgroup mask is the mask used by the work selection services to determine the best output element to select. The formatted mask is of the JOE selected.

c. The flags describe certain characteristics of the $#GET call and return code; the $GTW macro documents them.

d. If no work is selected, there will be no jobid, class, route code, or mask formatted.

2. **The Trace Record for a Spool Offload Device (Held Output)**

```
ID = 20  $#GET  JOB 1234 OFF1.ST  $#GET CALL for OFF1.ST
WS = (W,R,Q,PRM/UCS,P)
OUTGRPS DEFINED      = nnnn     OUTGRPS IN USE   = nnnn
OUTGRPS SCANNED      = nnnn     OUTGRPS THRU WS  = nnnn
OUTGRPS ON JOB CHAIN = nnnn     OUTGRPS SELECTED = nnnn
JOBS DEFINED         = nnnn     JOBS SCANNED     = nnnn
JOBS THRU WS         = nnnn
JOB MASK = FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFF
FLAGS= 20A48000
```

   Note that:

   a. When a selectable job with held output is found, JES2 runs the JOE chain for that job. Thus, the mask describes the JQE, and the trace record includes the count of JOEs for the job.

   b. If the device is not set up to take non-held work, the job's JOE chain and selected counts are irrelevant and not formatted. If no work is found, the job chain and these selected counts are also omitted.

3. **The Trace Record for a Spool Offload Device (Non-held Output)**

```
ID = 20  $#GET  JOB 1234 OFF1.ST  $#GET CALL for OFF1.ST
WS = (W,R,Q,PRM/UCS,P)
OUTGRPS DEFINED      = nnnn     OUTGRPS IN USE   = nnnn
OUTGRPS SCANNED      = nnnn     OUTGRPS THRU WS  = nnnn
OUTGRPS ON JOB CHAIN = nnnn     OUTGRPS SELECTED = nnnn
JOBS DEFINED         = nnnn     JOBS SCANNED     = nnnn
JOBS THRU WS         = nnnn
OUTGRP MASK = FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFF
CLASS = A     ROUTE = 00010000   FLAGS= 20A48000
```

   Note that:

   a. Because no JQEs were selected, the output queues were scanned: the mask, class, and route represent the JOE selected.

The Patching Facility

The JES2 patching facility can be used (when JES2 is started) in order to make dynamic changes to JES2 code. One patch space is reserved in module HASPNUC. Additional patch spaces can be reserved using the JES2 macro $PATCHSP.

The JES2 patching facility permits patches to be applied (at the time JES2 is initialized) through statements in the JES2 parameter library. Patches can be applied to any module in JES2 or to any absolute storage address available from the address space into which JES2 is loaded.

These patches are temporary (that is, valid only until a module is reloaded), and patches to modules in the JES2 address space must be applied every time that JES2 is started. Modules that are marked REFRESH (refreshable) should not be patched because a system refresh nullifies the effect of the patch.

When pages in the pageable link area (PLPA) are not paged out, any patches applied to modules residing in this area are not effective once the page in which the patch resides has been reloaded. For this reason, modules in the SYS1.LPALIB data set (for example, HASPSSSM) must be fixed through an entry in the

SYS1.PARMLIB fix list before patches are applied or HASPSSSM should be directly loaded into the CSA.

The patch statements in the JES2 parameter library can be specified in either the JES2 patch format or in AMASPZAP format however, all patches in the JES2 patch format should appear first.

Refer to *JES2 Initialization and Tuning* for the details of using the patching facility.

Diagnosing Routines Without Source

For some routines, IBM supplies only the object code and not the source code. Therefore, you cannot diagnose problems with these routines by referring to the source code. Instead, you need to gather diagnostic information and report this information to IBM.

This diagnosis section will help you gather information to report to your IBM support center. Using the information you provide, your support center will give you an answer to the problem with the routine.

Recognizing the External Symptoms

When a problem occurs in the JES2 routine for which there is no IBM-supplied source, JES2 issues the following message:

```
$HASP477 DATA SPACE {CREATE | DELETE} FAILED FOR INTERNAL
         READER PROTECTED BUFFERS, RC=rc
```

The return code specified in the $HASP477 message indicates why the routine failed. (See the explanation for $HASP477 in *JES2 Messages* for a list of return codes and their meanings.) This return code is part of the information that you need to report to your IBM support center.

If the return code is:	You need to:
4 or 8	Contact your IBM support center and provide them with the message number and this return code value. This failure does not generate SYS1.LOGREC information.
12 - 76	Identify the important fields in the SYS1.LOGREC and provide your IBM support center with this additional information. (RC = 16 indicates that more than one failure has occurred and you need to identify the important SYS1.LOGREC fields for each.) Refer to "Interpreting SYS1.LOGREC" for information about gathering this data.

Interpreting SYS1.LOGREC

If the return code from the $HASP477 message is 12 to 76, you need to gather information from your SYS1.LOGREC. See *MVS/ESA Using Dumps and Traces* for complete information about reading SYS1.LOGREC symptom report records. The SYS1.LOGREC entry for this problem will have a system abend code of **0F7**. Use this abend code to isolate the SYS1.LOGREC for this specific problem.

Use the following example of output to identify the information that you need to report to your IBM support center. This example is the section of the detail edit report (produced by EREP) showing the SDWA. Depending on an indicator in the

SDWA, the SDWA is displayed as either a hexadecimal dump or in key, length, and data format.

```
HEXADECIMAL DUMP

  HEADER
    +000  4C831800 00000000  0086251F 23380882    *<C.......F.....B*
    +010  FF022047 30810000                       *........*

  JOBNAME
    +000  D4E8D1D6 C2F1F2F3                       *MYJOB123*

  SDWA BASE
    +000  E2D9F3F0 F8F1F0F2  F2F0F4F7 FFFFFFF2    *SR3081022047...2*
    +010  FF802755 C4667601  40404040 40404040    *....D...        *
    +020  4040D5D6 D5C1D4C5  4040F5F7 F5F2C8C2    *  NONAME  5752HB*
    +030  C2F3F3F3 F0400080  00000000 00000000    *B3330 ..........*
    +040  F1F00030 00640070  002100D4 001800F5    *10.........M...5*
     ...
    +170  00000000 00000000  00000000 00000000    *................*
    +180  00000000 00000000  00000000 00000000    *................*
    +190  00000000 00000000  00000000 00000000    *................*

  VARIABLE RECORDING AREA (SDWAVRA)
    +000  KEY: 00     LENGTH: 08
    +002  F1F2F3F4 F5F6F7F8                       *12345678*

    +00A  KEY: 00     LENGTH: 14
    +00C  00000000 00000000  00000000 00000000    *................*
    +01C  00000000                                *....*

  SDWA FIRST RECORDABLE EXTENSION (SDWARC1)
    +000  00000000 00000000  00000000 00000000    *................*
    +010  00000000 00000000  00000000 00000000    *................*
     ...
    +080  00000000 00000000  00000000 00000000    *................*
    +090  00000000 00000000                       *........*

  SDWA SECOND RECORDABLE EXTENSION (SDWARC2)
    +000  00000000 00000000  00000000 00000000    *................*

  SDWA THIRD RECORDABLE EXTENSION (SDWARC3)
    +000  00000000 00000000  00000000 00000000    *................*
    +010  00000000 00000000  00000000 00000000    *................*

  ERRORID
    +000  00000000 00000000  00000000 00000000    *................*
```

Figure 5-25. Hexadecimal Dump in the Detail Edit Report

The variable recording area (SDWAVRA) is the section of the SDWA that you will need to have when you contact IBM system support. It contains the following $HASP477-specific information:

- Either of the following component descriptions:

  COMPON=JES2 DATA SPACE SERVICES SRB,
  COMPID=SC1HA,ISSUER=HASCDSS(DSFRR)

  COMPON=JES2 DATA SPACE SERVICES,
  COMPID=SC1HA,ISSUER=HASCDSS(DSESTAE)

- The data CSECT name ($DSWA).
- The DSPSERV parameter list, if applicable.
- The ALESERV parameter list, if applicable.

Appendix A. Multileaving

Multileaving is a computer-to-computer communication technique developed for use by JES2. Multileaving is fully synchronized, two-directional transmission of a variable number of data streams between two or more computers, utilizing binary synchronous communication (BSC) facilities.

Multileaving Philosophy

The basic element for multileaving transmission is the character string. One or more character strings are formed from the smallest external element of transmission - the physical record. These input physical records may be any standard record media (card images, printed lines, tape records, etc.). For efficiency in transmission, each record is reduced to a series of character strings of two basic types: a variable length non-identical series of characters, and a variable number of identical characters. Because of the frequency of blank characters, a special case is made for identical characters when the duplicate character is a blank. An 8-bit control field, termed a string control byte (SCB), precedes each character string to identify the type and length of the string. Thus, a string of nonidentical characters is represented by an SCB followed by the non-duplicate characters. A string of consecutive, duplicate, non-blank characters can be represented by an SCB and a single character; the SCB indicates the duplication count and the character following indicates the character to be duplicated. In the case of an all-blank character string, only an SCB is required to indicate both the type and number of blank characters. A data record to be transmitted is, therefore, segmented by the transmitting program into the optimum number of character strings to take full advantage of the identical character compression. A special SCB is utilized to indicate the grouping of character strings that compose the original physical record. The receiving program can then reconstruct the original record for processing.

In order to allow records of various media to be grouped together in a single transmission block, an additional 8-bit control field precedes the group of character strings representing the original physical record. This field, the record control byte (RCB), identifies the general type and function of the physical record (input stream, print stream, data set, etc.). A particular RCB type has been designated to pass control information between the various systems. To provide for simultaneous transmission of similar functions (such as multiple input streams), a stream identification code is included in the RCB. A second 8-bit control field, the subrecord control byte (SRCB) is included immediately following the RCB. This field supplies the receiving program with additional information concerning the record; for example, in the transmission of data to be printed, the SRCB can carry carriage control information.

For multileaving transmission, a variable number of records can be combined into a variable block size, as indicated previously, (that is, RCB, SRCB, SCB1, SCB2,...SCBn, RCB, SRCB, SCB1,...etc.). The multileaving design provides for two (or more) computers to exchange transmission blocks containing multiple data streams in an interleaved fashion. To allow optimum use of this capability, however, a system must have the capability to control the flow of a particular data stream while continuing normal transmission of all others, such as during the simultaneous transmission of two data streams to a system for immediate transcription to I/O devices of different speeds. To meter the flow of individual data streams, a function control sequence (FCS) is added to each transmission block. The FCS is a sequence of bits, each of which represent a particular transmission stream. The receiver of several data streams can temporarily stop the transmission of a particular stream by setting the corresponding FCS bit off in the next transmission to the sender of that stream. The stream can subsequently be resumed by setting the bit on.

Finally, for error detection and correction purposes, a block control byte (BCB), is added as the first character of each block transmitted. The BCB, in addition to control information, contains a modulo 16 block sequence count. This count is maintained and verified by both the sending and receiving systems to prevent lost or duplicated transmission blocks.

In addition to the normal binary synchronous text control characters (STX, ETB, etc.), multileaving uses two of the BSC control characters, ACK0 and NAK. ACK0 is used as a filler by all systems to maintain communications when data is not available for transmission. NAK is used as the only negative response and indicates that the previous transmission was not successfully received. The following indicates the format of a typical multileaving transmission block.

Multileaving Protocol for JES2 NJE

The following tables show the protocols used for JES2.

Multileaving Buffer Format

Characters

DLE	BSC control character
STX	BSC control character
BCB	Block control byte
FCS	Function control sequence
FCS	Function control sequence (continued)
RCB	Record control byte for record 1
SRCB	Subrecord control byte for record 1
SCB	String control byte for record 1
data	Character string
SCB	String control byte for record 1
data	Character string
SCB	Terminating SCB for record 1 (end-of-record)
RCB	Record control byte for record 2
SRCB	Subrecord control byte for record 2
SCB	String control byte for record 2
SCB	Terminating SCB for last record
RCB	Transmission block terminator (end-of-block)
DLE	BSC control character
ETB	BSC control character

Block Control Byte (BCB)

Binary	**Meaning**
r... ...	Reserved (must be 1)
.xxx	Control information as follows:
	.000 .... Normal block
	.001 .... Bypass sequence count validation
	.010 cccc Reset expected block sequence count to cccc
	.011 .... Reserved for future use
	.100 .... Reserved for future use
	.101 .... Not used
	.110 .... Not used
	.111 .... Reserved for future use
.... cccc	Modulo 16 block sequance count

Function Control Sequence (FCS)

Binary	Meaning
r... .... r...	Reserved (must be 1... .... 1... ....)
.0..	Normal state
.1..	Suspend all stream transmission (WAIT-A-BIT)
..rr .... ..rr	Reserved for future use
.... .1..	Remote console stream identifier
.... 1...	Function stream identifier for: RJE input stream number 1 RJE print stream number 1 NJE job transmission stream number 1
.... .1..	Function stream identifier for: RJE input stream number 2 RJE print stream number 2 RJE punch stream number 7 NJE job transmission stream number 2 NJE SYSOUT transmission stream number 7
.... ..1.	Function stream identifier for: RJE input stream number 3 RJE print stream number 3 RJE punch stream number 6 NJE job transmission stream number 3 NJE SYSOUT transmission stream number 6

Binary	Meaning
....1	Function stream identifier for: RJE input stream number 4 RJE print stream number 4 RJE punch stream number 5 NJE job transmission stream number 4 NJE SYSOUT transmission stream number 5
.... 1...	Function stream identifier for: RJE input stream number 5 RJE print stream number 5 RJE punch stream number 4 NJE job transmission stream number 5 NJE SYSOUT transmission stream number 4
.... .1..	Function stream identifier for: RJE input stream number 6 RJE print stream number 6 RJE punch stream number 3 NJE job transmission stream number 6 NJE SYSOUT transmission stream number 3
....1.	Function stream identifier for: RJE input stream number 7 RJE print stream number 7 RJE punch stream number 2 NJE job transmission stream number 7 NJE SYSOUT transmission stream number 2
....1	Function stream identifier for: RJE punch stream number 1 NJE SYSOUT transmission stream number 1

Record Control Byte (RCB)

Binary	Hex	Meaning
0000 0000	00	End-of-block
rrrr rrrr	01-8F	Reserved for future use
1001 0000	90	Request to initiate function (SRCB = RCB of function)
1010 0000	A0	Permission to initiate function (SRCB = RCB of function)
1011 0000	B0	Negative permission or receiver cancel (SRCB = RCB) of function)
1100 0000	C0	Acknowledge transmission complete (SRCB = RCB of function)
1101 0000	D0	Not used
1110 0000	E0	BCB sequence error
1111 0000	F0	General control record
1001 0001	91	RJE console message
1rrr 0001	A1-F1	Reserved for future use
1001 0010	92	RJE operator command
Irrr 0010	A2-F2	Reserved for future use
liii 0011	93-F3	RJE input record where iii contains identification information
liii 0100	94-F4	RJE print record where iii contains identification information
liii 0101	95-F5	RJE punch record where iii contains identification information
liii 0110	96-F6	Data set record where iii contains identification information
liii 0111	97-F7	Terminal message routing request where iii contains identification information
liii 1000	98-F8	NJE input record where iii contains identification information
liii 1001	99-F9	NJE SYSOUT record where iii contains identification information
1001 1010	9A	NJE operator command/NJE console message
Irrr 1010	AA-FA	Reserved for future use
1001 1011	9B	Reserved
Irrr 1011	AB-FB	Reserved for future use
Irrr 1100	9C-FC	Reserved for future use
Irrr 1101	9D-FD	Reserved for future use
Irrr 1110	9E-FE	Reserved for future use
Irrr 1111	9F-FF	Reserved for future use

Subrecord Control Byte (SRCB)

RCB	SRCB
00	None
90	RCB of function to be initiated
A0	RCB of function to be initiated
B0	RCB of function to be cancelled
C0	RCB of function which is complete
E)	Expected count (received count is in BCB)

F0	An identification character as follows:
	A = Initial RJE sign-on
	B = Final RJE sign-off
	C = Print initialization record
	D = Punch initialization record
	E = Input initialization record
	F = Data set transmission initialization
	G = System configuration status
	H = Diagnostic control record
	I = Initial network sign-on
	J = Response to initial network sign-on
	K = Reset network sign-on
	L = Accept (concurrence) network sign-on
	M = Add network connection
	N = Delete network connection
	O-R = Reserved for future use
	S-Z = Unused

91	1000 0000 (X'80')
92	1000 0000 (X'80')
93-F3	1000 0000 (X'80')
94-F4	Carriage control information as follows: 1010 00nn Space immediately nn spaces (not used) 1011 cccc Skip immediately to channel cccc (not used) 1000 00nn Space nn lines after print 1000 1100 Load printer FCB image 1001 cccc Skip to channel cccc after print 1000 0000 Print and suppress space
95-F5	1000 1111 (X'8F')
96-F6	Undefined
97-F7	Undefined

RCB	SRCB
98-F8	NJE input control information as follows:

1000 0000	Normal input record
1100 0000	Job header
1110 0000	Data set header
1101 0000	Job trailer
1111 0000	Data set trailer (not used)

99-F9	NJE SYSOUT control information as follows:

10cc 0000	Carriage control type as follows:	
	1000 0000	No carriage control
	1001 0000	Machine carriage control
	1010 0000	ASA carriage control
	1011 0000	Reserved for future use
11cc 0000	Control record as follows:	
	1100 0000	Job header
	1110 0000	Data set header
	1101 0000	Job trailer
	1111 0000	Data set trailer (not used)
1000 ss00	Spanned record control as follows:	
	1000 0000	Normal record (not spanned)
	1000 1000	First segment of spanned record
	1000 0100	Middle segment of spanned record
	1000 1100	Last segment of spanned record

9A	1000 0000 (X'80')
9B	1000 0000 (X'80')

String Control Byte (SCB)

Binary	Meaning
0000 0000	End-of-record If first SCB, this also indicates end-of-file
0100 0000	Abort transmission
100b bbbb	bbbbb blanks are to be inserted
101d dddd	The single character following this SCB is to be duplicated ddddd times
llcc	The cccccc characters following this SCB are to be inserted

Standard Data Record Format

Characters

LRECL	Original record length
CCTL	Carriage control (only present if indicated in SRCB)
data	Record text

Spanned Data Record Format

Characters

SEGL	Segment length (1 byte) (present in all segments)
LRECL	Total original record length (2 bytes) (only present in first segment)
data	Segment text

Channel-to-Channel Adapter Support

The remote terminal access method (RTAM) supports the channel-to-channel adapter as a communication line. The support is for 360 mode operations over block multiplexer channels. Instead of using channel command words (CCWs) designed for binary synchronous communication (BSC) line transmissions, the channel-to-channel adapter sequences are as follows:

Opcode	Definition	Comments
X'14'	Sense	This CCW causes other end of adapter control CCW to fall through to the read CCW
X'01'	Write	Writes data to the adapter
X'07'	Control	Causes the channel program to wait until other end of adapter issues sense; block multiplex channel is free for other programs
X'02'	Read	Read data from adapter

The prepare to sign-on sequence sends the BSC characters SYN-NAK instead of being passive as with BSC lines. If one end of the adapter is placed back into prepare for sign-on (sends the SYN-NAK), the RTAM at the other end does likewise. Because there are no channel-end interrupts when both ends of the adapter are not being instructed, operators wanting to drain the line device control table (DCT) must also enter a line restart ($E line) after the drain command on the last active end of the adapter.

Multileaving in BSC/RJE

While the JES2 support for programmable BSC work stations is completely consistent with the multileaving design, it does not utilize certain of the features provided in multileaving. These feature limitations include:

- The transmission of record types other than print, punch, input, console, and control is not supported.
- The only general control record types utilized are the terminal sign-on and terminal sign-off.
- Only SCB count units of 1 are utilized.
- No support is included for column binary cards.
- Multiple RCB types in an output buffer are not supported.

Multileaving Protocol for SNA NJE

Within an SNA NJE session, multiple streams have been defined to carry different types of data. NJE transmits three types of data: job, SYSOUT, and console data. Although JES2 only defines these three stream types, the JES2 user may specify more than one stream of each type. The data's stream is identified in a portion of a 3-byte record header known as the record identifier. An RU can be composed of any number of data records with their record identifiers; those data records need not be of the same stream.

When a job enters a node through NJE, JES2 performs a SAF call to RACF passing the user id, password, group id, and SECLABEL from the networking header on the early verification SAF call made during input service. The SAF call verifies that the user is valid and returns a default security token for the job.

Jobs entering from another node may not pass the verify call. If SAF/RACF rejects the request, JES2 purges the job or data set without printing any output and sends an error message to the sending node.

When building job header records to send a job or SYSOUT data sets to another node, JES2 extracts the user id, password, group id, and SECLABEL from the job's security token and places it in the header.

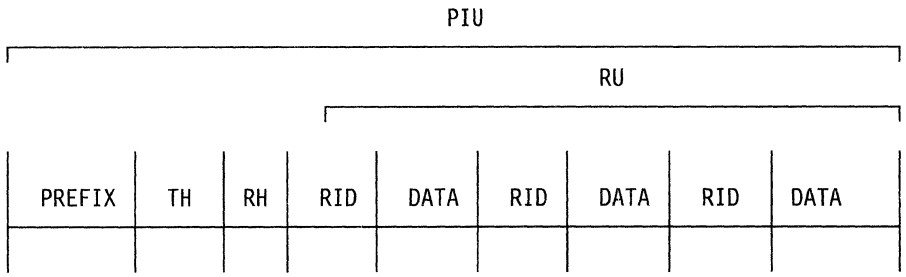
Record Identifiers

Record identifiers (RIDs) are 3-byte headers, which are required on every logical record sent and received by SNA NJE. The RID identifies what type of information is to be found in the associated record.

An RU consists of as many record identifiers and corresponding logical records as can fit in the specified RU. No logical record may include more than 256 bytes; therefore, the maximum logical record length is 256 bytes for the logical record and 3 bytes for the record identifier. Although these logical records may be compacted which normally reduces the length, RU sizes are large enough to hold several logical records. See Figure A-1 for a picture of an RID within an SNA transmission unit.

JES2 identifies 3 types of records in the RID: the network topology record, the stream control record, and the data record. Network topology information is sent within a session to build the NJE topology control blocks, thereby allowing alternate path, multiple trunk, and mixed network capabilities. Stream control records define and control the streams on which the data records flow. In other words, before a job can be transmitted from one node to another, a job stream must be started at both sides of the session. Once the stream control records have established the job stream successfully, data records (the job information itself) may flow on that stream.

Currently, JES2 sends only one type of record in an RU. However, the architecture of SNA NJE has specifically allowed for the possibility of multiplexing record types within an RU.



```
PIU  = SNA path information unit
TH   = SNA transmission header (26 bytes)
RH   = SNA request header (3 bytes)
RU   = SNA request unit (maximum 65,535 bytes)
RID  = SNA NJE record identifier (3 bytes)
DATA = Maximum 256 bytes
```

Figure A-1. SNA Transmission Unit

Data Compression and Compaction

JES2 has the capability for compressing 2 or more repetitions of the character "blank" into a 2-byte sequence, and for compressing 3 or more repetitions of any other (non-blank) character into a 3-byte sequence. This standard compression capability is supported by all SNA NJE applications.

JES2 also supports an optional capability called compaction for all SNA NJE communication. Compaction is a scheme by which some 8-bit data characters, called master characters, may be represented by 4-bit codes for transmission when they are adjacent in the data. Another group of characters, called non-master characters, can occur adjacent to master characters and be represented in 8-bit form without disturbing the compaction mode. All other characters, called non-compactable characters (the least frequent ones in the data), are represented in their true 8-bit forms and are transmitted uncompacted. The master and non-master compaction characters are defined by user-specified compaction tables which, are transmitted by SNA NJE in FM headers.

In SNA NJE, compression and compaction is performed on an RU basis, not on a record basis. Therefore, everything from the beginning to the end of the RU, without regard to record identifiers, is compressed or compacted. Only one compaction table can be in effect at a time for the sending application.

The first byte of each RU must be a string control byte (SCB). The SCB contains a count which locates the next SCB or completes the particular SCB code definition.

The one-byte SCB consists of a 2-bit field describing the mode of compression or compaction in effect, followed by a 6-bit field that provides the count of the number of characters described by this SCB. The format of the SCB is:

tt=cccccc

tt=00 No compressed characters; the count field defines the number of unique characters following.

tt=01 Compact code; the count field indicates the number of bytes between this SCB and the next.

tt=10 Repeated blanks; the count field indicates the number of blanks to appear in the expanded data.

tt=11 Repeated non-blank character; the count field indicates the number of times the non-blank character is to be repeated in the expanded data.

In all cases, a count of 0 is a reserved value and must not be used. Valid counts are in the range 1-63.

In compression, repeated blanks are compressed only when 2 or more occur; repeated non-blank characters are compressed only when 3 or more occur. When both compression and compaction are used, as in SNA NJE, the trade-offs used in selecting and changing SCB modes are significantly more complex and cannot be adequately described here. The processes of compaction and compression deal with the data simply as a series of bytes without consideration for the information they may represent (RID, data, etc.).

Record Identifier (RID) Format

The following is a description of the 3-byte RID format.

RIDRCB (record control byte) - byte 0

Binary	**Hex**	**Record Type**	**Meaning**
1000 0000	80	Stream control	Hold stream (RIDSRCB=SCB of stream)
1001 0000	90	Stream control	Request to allocate JOB/SYSOUT stream (RIDSRCB=RCB of stream)
1010 0000	A0	Stream control	Permission to allocate stream granted (RIDSRCB=RCB of stream)
1011 0000	B0	Stream control	Permission to allocate stream denied-or-receiver cancel (RIDSRCB=RCB of stream)
1100 0000	C0	Stream control	Acknowledge end-of-transmission RIDSRCB=RCB of stream)
1101 0000	D0	Stream control	Release stream (RIDSRCB=RCB of stream)
1111 0000	F0	Network topology	The RID is followed by an NJE network topology record

liii 1000	98-F8	Data	The RID is followed by an NJE job stream data record (iii identifies the particular job stream)
liii 1001	99-F9	Data	The RID is followed by an NJE SYSOUT record (iii identifies the particular SYSOUT stream)
1001 1010	9A	Data	The RID is followed by a modal message record, which in turn contains an NJE command

or console message

RIDSRCB: Subrecord Control Byte - Byte 1

The meaning of this byte depends on the value of the record control byte (byte 0).

RIDRCB **RIDSRCB**

80 RCB of JOB/SYSOUT stream to be held (X'FF' indicates all streams to be held.)

90 RCB of JOB/SYSOUT stream to be allocated

A0 RCB of JOB/SYSOUT stream for which permission to allocate has been granted

B0 RCB of JOB/SYSOUT stream to be cancelled or for which permission to allocate has been denied

C0 RCB of JOB/SYSOUT stream for which end-of-transmission has been acknowledged

D0 RCB of JOB/SYSOUT stream to be released (X'FF' indicates all streams to be released)

F0 EBCDIC character identifying type of NJE topology record as follows:

C'I' = Initial network sign-on
C'J' = Response to initial network sign-on
C'K' = Reset network sign-on
C'L' = Accept (concurrence) network sign-on
C'M' = Add network connection
C'N' = Delete network connection

98 Transmission end information as follows:

0000 0000 - Normal job stream end-of-transmission

0100 0000 - Job stream cancelled by transmitter

98-F8 NJE job control information as follows:

1000 0000 - Normal input record
1100 0000 - Job header
1110 0000 - Data set header
1101 0000 - Job trailer
1111 0000 - Data set trailer (not used)

99-F9 Transmission end information as follows:

0000 0000 - Normal SYSOUT stream end-of-transmission
0100 0000 - SYSOUT stream cancelled by transmitter

NJE SYSOUT control information as follows:

10cc 0000 - Carriage control type as follows:

1000 0000	- No carriage control
1001 0000	- Machine carriage control
1010 0000	- ASA carriage control
1011 0000	- Reserved for future use

11cc 0000 - Control record as follows:

1100 0000	- Job header
1110 0000	- Data set header
1101 0000	- Job trailer
1111 0000	- Data set trailer (not used)

1000 SS00 Spanned record control as follows:

1000 0000	- Normal record (not spanned)
1000 1000	- First segment of spanned record
1000 0100	- Middle segment of spanned record
1000 1100	- Last segment of spanned record

9A 1000 0000 - SRCB always X'80'

RIDRLEN: Record Length-Byte 2

The meaning of this byte also depends on the value of the record control byte (byte 0).

RIDRCB	RIDRLEN
80	N/A (set to zero)
90	N/A (set to zero)
A0	N/A (set to zero)
B0	N/A (set to zero)
C0	N/A (set to zero)
D0	N/A (set to zero)
F0	Length of the network topology record following the RID plus 3 (length of the RID). This is so defined to make SNA NJE topology records, which are processed by the network path manager without use of the $EXTP GET macro instruction, compatible with BSC NJE topology records.
98-F8	Length-1 of the data record following the RID (used during $EXTP GET processing to determine the number of bytes beyond the RID to be decompressed/decompacted into the user-supplied area to satisfy the request for a record).
99-F9	Same as for 98-F8.
9A	Length-1 of the nodal message record (NMR) following the RID.

Appendix B. External Writer

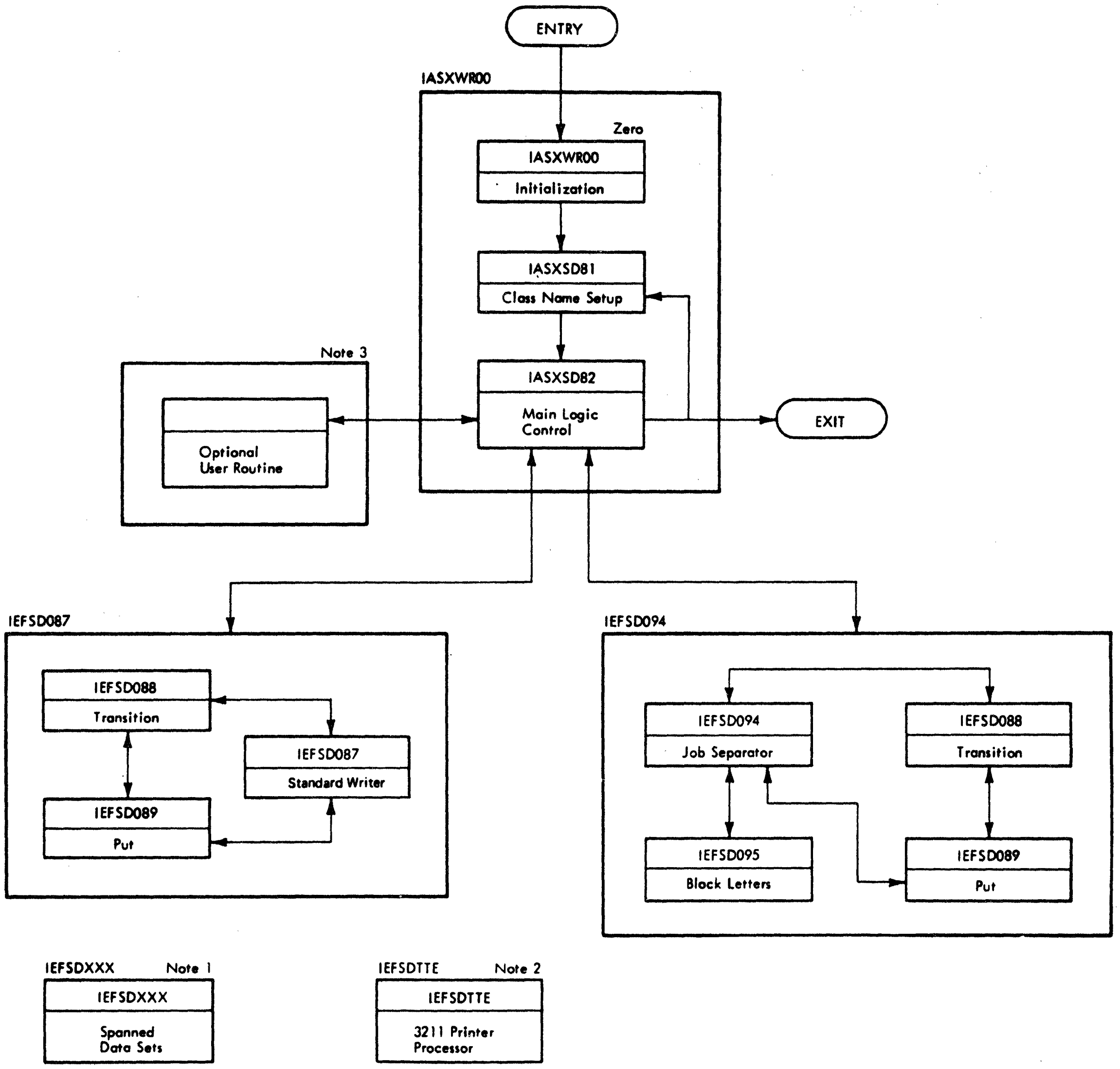
There are a maximum of 10 modules that make up the external writer:

IASXWR00
IASXSD81
IASXSD82
IEFSD087
IEFSD088
IEFSD089
IEFSD094
IEFSD095
IEFSDTTE
IEFSDXXX

These modules are used to create five load modules:

IASXWR00
IEFSD087
IEFSD094
IEFSDTTE
IEFSDXXX

Refer to Figure B-1 for a load module map of the external writer.



Notes:

1. Loaded by either IASXSD82 or IEFSD087; used as a subroutine by IEFSD087, IEFSD089, and IEFSD094.

2. Linked to from IASXSD82, IEFSD087, and IEFSD094.

3. IASXS82A loads either IEFSD087 or the optional user routine.

Figure B-1. Load Module Map of External Writer

The function of the external writer, once it has been started by an operator command, is to access data sets from the primary job entry subsystem, dynamically allocate these data sets, read them, write them to an output device, and dynamically deallocate them. When there are no subsystem data sets eligible in terms of the writer selection criteria, the external writer waits on two event control blocks (ECBs), one a subsystem ECB and another a command scheduling control block (CSCB) ECB, which gets posted whenever an operator stops or modifies the external writer.

The purpose of the external writer is twofold. One is to provide a means of writing SYSOUT data sets to any device that can be accessed by QSAM, such as a tape or a direct-access device. The second is to allow two user interfaces for SYSOUT data set. The user can write a job separator routine and name that routine to the external writer via the PARM field on the EXECUTE statement of the writer procedure. The IBM-supplied separator routine is IEFSD094. The second user interface is a data set processing subtask that is loaded by the external writer main logic control program whenever a data set is to be written to the output device. The IBM-supplied default data set writer module is IEFSD087.

The IBM-supplied writer procedure, XWTR, is set up to write class A SYSOUT data sets to tape. The program name on the EXEC statement is IASXWR00. This external writer control module, IASXWR00, must be named on any procedure that invokes the external writer. This control module loads a user-written data set writer, if requested. The user specifies a user-written writer name on a SYSOUT DD statement. If there is no name on the SYSOUT DD statement, the external writer loads the IBM-supplied data set writer routine, IEFSD087.

The external writer processes all data sets that are received from the job entry subsystem in the same manner. It is unaware whether the data set is a JES2 job log data set, a system message block (SMB) data set or a user-written data set.

IASXSD81

IASXSD81, the class name setup routine, initializes the external writer based upon the writer procedure parameter area and the parameters in the start command. The address of the communication's ECB in the CSCB is saved in the PARLIST for future reference. If the parameters are all valid, the output data set is opened, and control is passed to IASXSD82. If errors are detected, an error message is issued, and the external writer stops.

IASXSD81 also validates modify commands processed by the external writer and modifies the writer selection criteria.

IASXSD82

The main logic control module of the external writer, IASXSD82, accesses data set names from the primary job entry subsystem and loads the data set writer to read these input data sets and to write this data to the output device. If the external writer was started without any class list being specified, IASXSD82 immediately issues a waiting for work WTO and waits for the stop/modify CSCB ECB to be posted. If the writer was started with a class list as selection criteria, the IEFSSREQ macro is issued to request a data set name that is eligible for the writer to process. If there are no data sets eligible according to the writer selection criteria, IASXSD82 issues a waiting for work WTO and then waits on both a stop/modify CSCB ECB and a subsystem ECB, which is posted when new data sets are added to the JES2 output queue. When the writer is posted, it either performs the action requested by the command or requests another data set from JES2.

If a data set name is received from JES2, IASXSD82 checks to see if a writer name was specified on the SYSOUT DD statement by the user; if so, the name is verified. Then the data set name is used to dynamically allocate the input data set. The job separator routine is linked to when necessary. The job file control block (JFCB) for the input data set is read and a cancellable step type CSCB is created for the life of this data set processing so that the operator can cancel the writing of this data set. A subtask, IASXS82A, is attached. IASXS82A exists as an entry point in IASXSD82 to load IEFSD087 or a user-written data set writer. Once the data set writer finishes processing the data set, the IASXS82A subtask purges any outstanding I/O of the subtask, deletes the data set writer, and returns to IASXSD82. IASXSD82 then removes the cancel CSCB from the chain, dynamically deallocates the input data set, and restores any outstanding I/O to itself.

To ensure that it will generate a type 6 SMF record, IASXSD82 uses an eight-character job id to test for a change in job numbers. This id is located in field SMF6JBID of the routing section of the record.

If no modify or stop commands were issued by the operator, IASXSD82 starts the main loop again by requesting more work from JES2. If the stop command was issued, IASXSD82 returns to JES2 one final time to indicate that processing is complete and the external writer gives up control. If a modify command was issued by the operator, IASXSD82 branches to IASXSD81 to process the modify command. Then, IASXSD82 goes back to the main loop to call JES2 with the new selection criteria.

IASXWR00

IASXWR00 is the external writer initialization routine that is entered when the writer is started. IASXWR00 initializes an internal control block, PARLIST, used by the external writer and then exits to IASXSD81.

IEFSD087

IEFSD087, the data set processing subtask, issues an OPEN (J type) for the input data set received from JES2, reads the input data set, and uses the modules IEFSD088 and IEFSD089 to write data from the input data set to the output data set (device). If the output device is a 3211 Printer, IEFSD087 uses IEFSDTTE. If applicable, IEFSD087 uses module IEFSDXXX. After the entire input data set has been read, IEFSD087 closes the input data set and returns control to IASXSD82.

IEFSD088

The transition routine, IEFSD088, is used by IEFSD087 and IEFSD094 to handle American National Standard and MCH control character differences and conversion. It uses the put routine, IEFSD089, to write records to the output device, when necessary.

IEFSD089

The put routine, IEFSD089, is entered from IEFSD087, IEFSD088, or IEFSD094 to write records to the output device. IEFSD089 calls IEFSDXXX, if necessary and does any control character manipulation caused by differences in control characters between the input and the output data sets.

IEFSD094

The IBM-supplied job separator control routine, IEFSD094, uses the modules IEFSD095, IEFSD088, IEFSD089, and optionally IEFSDTTE to create the separator pages or cards necessary to separate jobs.

IEFSD095

The block letter routine IEFSD095, generates the records that make up the block letters and numbers.

IEFSDTTE

The 3211 processor routine, IEFSDTTE, is entered only if the output device is a 3211 Printer and some change to the universal character set/forms control buffer (UCS/FCB) environment has been predetermined. When the input request differs from the present output, the input request is transferred from the input JFCB or from PARLIST to a SETPRT list, and the SETPRT macro instruction is issued to cause data management to reload the UCS/FCB buffers. IEFSDTTE is used by IASXSD82, IEFSD087, and IEFSD094.

IEFSDXXX

The spanned data sets routine, IEFSDXXX, is used by IEFSD089 and IEFSD087 when the output data set consists of variable-length spanned (VS) records. IEFSDXXX moves records (or segments) from the input buffers (or work area) to the output buffers.

Appendix C. SSVT, $HCCT, and $CADDR

The subsystem vector table (SSVT), and the $HCCT and $CADDR control blocks contain the control information JES2 uses to communicate between processing programs in a user address and the JES2 mainline code (HASJES20) in the JES2 address space. Information is broken down as follows:

SSVT
Function support routine matrix and entry addresses

$HCCT
Contains communication control fields, $$POST elements, job service queue heads, and miscellaneous queue heads.

$CADDR
Pointers to JES2 service routines

Function Support Routine Entry Addresses

The $SENTRY macro contains the addresses of these routines. The $SVTC macro contains the supported function matrix.

SSI Function Number	Label	Module	Function
1	SSISOUT	HASCSIRQ	Process SYSOUT
2	SSICSCAN	HASCSIRQ	TSO Cancel Command
3	SSICSTAT	HASCSIRQ	TSO Status Command
4	SSIETEOT	HASCJBTR	End of Task
5	SSIJSEL	HASCJBST	Job Selection
6	SSIALOC	HASCDSAL	Allocation
7	SSIALUNA	HASCDSAL	Unallocation
8	SSIENEOM	HASCJBTR	End of Memory
9	SSIWTA	HASCSIRQ	Write to Operator (WTO)
10	SSICMD	HASCSIRQ	Command Processing
11	SSIUSUSE	HASCSIRQ	Verify User
12	SSIJTERM	HASCJBST	Job Termination
13	SSIRQRNQ	HASCJBST	Job Re-enqueue
16	SSIDAOPN	HASCDSOC	SYSIN/SYSOUT Open
17	SSIDACLO	HASCDSOC	SYSIN/SYSOUT Close
18	SSIDACKP	HASCDSOC	Checkpoint
19	SSIDARES	HASCDSOC	Restart
20	SSIRRREQ	HASCJBST	Request Job ID
21	SSIRRRET	HASCJBST	Return Job ID
53	SSIFSCNT	HASCSIRQ	Connect/disconnect FSI

Communication Control Fields

Label	Meaning
CCTHCT	Address of HASP Communication Table
CCTHTCBA	JES2 Main Task TCB Address
CCTSSCT	Address of SSCT
CCTHAVT	Address of HASP Address Space Vector Table
CCTPIDLE	Number of Idle PITs
CCTHASP	HASP Condition
CCTSTUS	Subsystem Status
CCTTSLOK	TSU Abend Interlock Flag
CCTCOMCH	HASP Command Character
CCTSID	System ID

$$POST Elements

Label	Meaning
CCTECF	Event Control Field. Indicates what JES2 main task resource has been freed by another task.
CCTCOMM	Command Processor.
CCTJOB	Execution Processor.
CCTASYNC	Asynchronous I/O Processor.
CCTXSTIM	Time Excess Processor.
CCTTIMER	Timer Processor.
CCTTRPCE	Trace Logger Processor.
CCTSPOOL	Spool Manager Processor.
CCTMLLM	Line Manager Processor.
CCTOFFM	Spool Offload Processor.
CCTCKPTP	Checkpoint Processor.

Job Service Queue Heads

Label	Meaning
CCTJLOCK	Job service queues lock header
CCTJPCLS	Subsystem job blocks (SJBs) waiting selection of job by class
CCTJPXBM	SJBs waiting selection of job for batch monitor
CCTJPNUM	SJBs waiting selection of job by number
CCTJXCLS	SJBs representing jobs selected by class
CCTJXNUM	SJBs representing jobs selected by number
CCTJTERM	SJBs waiting for termination
CCTJRENQ	SJBs waiting for job re-enqueue

Miscellaneous Queue Heads

Label	Meaning
CCPOSTE	8-Byte Field. Address of JES2 ECB ($HASPECB) and JES2 ASCB. Used by $$POST for XMPOST.
CCTSPIOT	IOTs representing spin data sets
CCTTSCS	SJBs with status or cancel requests pending
CCTPSOQ	SJBs with process SYSOUT requests
CCTFIFOQ	IOTs in FIFO order representing spin/hold requests
CCTPRGQ	Purge PSO queue
CCTPSOFF	PSO FIFO queue
CCTIOERR	Spool processor I/O error queue
CCTFSSCB	FSSCB queue

Appendix D. JES2 Acronyms

ACB	access control block
ACB	access method control block
ACE	automatic command element
ACT	automatic command table
ADS	abort device select
API	application program interface
APPL	application
APT	application table
APW	application work area
ARR	address recall register
ASID	address space identifier
AX	authorization index
BCB	block control byte
BDS	begin destination select
BFD	generalized subsystem data set buffer
BPM	buffer pool map
BSC	binary synchronous control
BSCA	binary synchronous communication adapter
BTG	bad track group
CAT	class attribute table
CCA	cell control area
CCE	cell control element
CCT	control record TCT
CCW	channel command word
CDS	continue destination select
CES	connection event sequence
CHK	checkpoint control block
CID	communication identifier
CIRWORK	common initialization routine PCE work area
CKB	checkpoint block
CKD	count-key-data
CMB	console message buffer
CMS	cross memory services
CPT	compaction table
CSA	common storage area
CSCB	command scheduling control block
CTC	channel-to-channel adapter
CVT	communications vector table
DAS	direct access spool control block
DCB	data control block
DCT	device control table
DEB	data extent block
DECB	data event control block
DOE	dump offload element
DOM	delete operator message
DTE	daughter task element
EB	end bracket
ECB	event control block
EDS	end data set
EDS	end device select
ERA	error recovery area
EST	estimated count table
ETB	end of transmission block
FB	function block
FCB	forms control buffer
FCS	function control sequence
FIFO	first-in-first-out
FM	function management
FSA	functional subsystem application
FSACB	functional subsystem application control block
FSAXB	functional subsystem application extension block
FSCT	functional subsystem control table
FSI	functional subsystem interface
FSS	functional subsystem
FSSCB	functional subsystem control block
FSVT	functional subsystem vector table
FSSWORK	functional subsystem PCE work area
GCB	GETREC control block
HAM	HASP access method
HASB	HASP address space block
HASPGBL	HASP global macro variables
HASPGEN	HASP generation values
HAVT	HASP address space vector table
HCA	hopper control area
HCCT	HASP common communications table
HCT	HASP communications table
HDB	HASP data block
HFCT	HASP functional subsystem communication table
IAR	instruction address register
ICE	interface control element
IDAL	indirect address list
IOB	input/output buffer
IOS	input/output supervisor
IOT	input/output table
IPL	initial program load
IRDCT	internal reader device control table
JAW	JES2 authorization work area
JFCB	job file control block
JIB	JOE information block
JIX	job queue index
JMR	job management record
JOE	job output element
JOT	job output table
JQE	job queue element
JSCB	job step control block
KEYLIST	keyword - SWB id table
KIT	checkpoint information table

LGRR	LOGREC record
LIFO	last-in-first-out
LMT	load module table
LRC	logical record control block
MCH	machine check handler
MCS	multiple console support
MCT	master control table
MDR	3800 maintenance data record
MFCU	multi-function card unit
MGCR	MGCR parameter list
MIT	module information table
MLWTO	multiline write-to-operator
MTTR	module relative track/record
NAT	nodes attached table
NCC	network connection control
NDH	network data set header
NEL	interpreter entry list
NIT	node information table
NJH	network job header
NJR	network job route receiver
NJT	network job trailer
NMR	nodal message record
NSR	network SYSOUT receiver
NST	network SYSOUT transmitter
NTQ	NAT temporary queue element
OCR	output control record
OCT	output control table
ODS	one-chain destination select
ORE	operator reply element
PBF	protected buffer
PC	program call
PCE	processor control element
PCI	program-controlled interrupt
PCIE	program-controlled interrupt element
PDDB	peripheral data definition block
PDIR	peripheral data information record
PIT	partition information table
PQE	page queue entry
PQEC	checkpoint PQE
PQED	data set PQE
PQEJ	job start PQE
PQES	SMF type 6 PQE
PPPWORK	HASPPRPU PCE data area
PRE	processor recovery element
PSCB	protected step control block
PSO	process - SYSOUT
PSV	processor save area
PSW	program status word
PT	program transfer
QCE	queue control element
QCT	quick cell control table
QSE	shared queue element
RAT	remote attribute table
RBA	relative byte address
RCB	record control byte
RCP	remote console processor
RCT	printer TCT
RDS	resume device select
RDT	remote destination table
RID	record identifier
RJE	remote job entry
RJO	relative job offset
RPL	request parameter list
RPS	rotational position sensing
RQR	request recovery request
RTAM	remote terminal access method
RTM	recovery termination manager
RTP	remote terminal program
RTR	ready to receive
RU	SNA request units
RWT	remote work table
SCA	synchronous communication adapter
SCAT	SYSOUT class attribute table
SCB	string control byte
SCR	spool control record
SCS	standard character string
SCWA	scan work area
SDB	subsystem data set block
SDLC	synchronous data link control
SDS	suspend device selection
SJB	subsystem job block
SJXB	subsystem job block extension
SNA	system network architecture
SPL	dynamic spool allocation
SRB	service request block
SRCB	subrecord control byte
SSCS	cancel status portion of SSOB
SSCT	subsystem communications vector table
SSDA	for OPEN/CLOSE and checkpoint/restart
SSI	subsystem interface
SSIB	subsystem identification block
SSJS	SSOB extension for JOB select
SSOB	subsystem options block
SSVT	subsystem vector table
STAB	scan table entry
STC	started task control
STQE	STIMER queue element
SVF	set vertical format
SWA	scheduler work area
SWB	scheduler work block
SWBIT	SWB information table
TAB	track allocation block
TAT	temporary nodes attached table
TCB	task control block
TCT	total control table
TGAE	track group allocation entry
TGB	track group block
TGM	track group map
TIC	transfer-in-control
TID	trace id table

TOD	time-of-day
TP	teleprocessing
TQE	timer queue element
TRC	table reference character for 3800
TRCA	termination recovery communication area
TRE	TCB recovery element
TSO	time-sharing option
TSU	time-sharing user
TTE	track table entry
UBF	unprotected buffer
UCB	unit control block
UCM	unit control module
UCMID	unit control module identifier
UCSB	universal character set buffer
UCT	punch TCT
UFCB	unit function control block
UPL	UCB parameter list
VRA	SDWA variable recording area
VS	variable-length spanned records
VTAM	virtual telecommunication access method
WCT	console TCT
WS	work selection
WQE	WTO queue element
XBM	execution batch monitor
XECB	extended ECB
XIT	exit information table
XRT	exit routine table

Index

MVS/ESA
JES2 Logic
MVS/System Product:
JES2 Version 3

READER'S
COMMENT
FORM

LY28-1006-2

This manual is part of a library that serves as a reference source for systems analysts, programmers, and operators of IBM systems. You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Note: ***Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.***

Possible topics for comment are:

Clarity Accuracy Completeness Organization Coding Retrieval Legibility

If you wish a reply, give your name, company, mailing address, and date:

____

____

____

____

What is your occupation? ______________________________

How do you use this publication? ______________________________

Number of latest Newsletter associated with this publication: ______________

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments or you may mail directly to the address in the Edition Notice on the back of the title page.)

MVS/ESA JES2 Logic MVS/System Product: JES2 Version 3

LY28-1006-2 S370-36

cut along this line

Reader's Comment Form

Fold and Tape Please Do Not Staple Fold and Tape

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 40 ARMONK, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE

International Business Machines Corporation
Department D58, Building 921-2
PO Box 950
Poughkeepsie, New York 12602-9935

Fold and Tape Please Do Not Staple Fold and Tape

Printed in U.S.A.

IBM®

IBM®

Program Number
5685-001

File Number
S370-36

LY28-1006-2

Printed in U.S.A.