LC23-0067-3
File No. S370-36

Program Product

System Programming Library: JES2 User Modifications and Macros

MVS/System Product - JES2
Version 1 Release 3.6

IBM

LC23-0067-3
File No. S370-36

Program Product

System Programming Library: JES2 User Modifications and Macros

MVS/System Product - JES2
Version 1 Release 3.6

Program Number 5740-XYS

Do not replace your existing documentation until you install the JES2 component of MVS/SP Version 1 Release 3.6.

Fourth Edition (February, 1987)

This is a major revision of, and obsoletes, LC23-0067-2. See the Summary of Amendments following the Contents for a summary of changes made to this manual. The second edition (LC23-0067-1) still applies to Version 1 Release 3.4 of System Product - JES2 5740-XYS and may now be ordered using temporary order number LQ23-0067.

This edition applies to Version 1 Release 3.6 of MVS/System Product - JES2 5740-XYS and all subsequent releases. 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, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM program product in this publication is not intended to state or imply that only IBM's program product may be used. Any functionally equivalent program may be used instead.

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

A form for 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 390, 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.

Preface

Who Should Use This Book

This book provides the information that you need to:

- Plan to modify JES2 with exits
- Implement and test the JES2 exits
- Use JES2 programmer macros

This book is intended for JES2 system programmers or for anyone responsible for installing, initializing, tuning, and modifying JES2.

Organization and Content

The "Table of Contents" listed within the Preface contains a minimal subheadings listing for each chapter. Use this listing to decide upon which chapter to reference. Prior to each chapter is a full "Chapter Table of Contents" that provides a complete listing of all heading within that chapter. Use this listing for detailed chapter content.

The organization and content of each chapter are as follows:

- Chapter 1, "Introduction," describes the processing concepts of JES2 exits and provides a framework within which the JES2 programmer can modify JES2 operation.

- Chapter 2, "Customizing JES2," describes how you should go about modifying JES2 processing. This chapter presents implementation details you should know to develop your modifications and lists the IBM-defined exits established for your use.

- Chapter 3, "IBM-Defined Exits" lists the IBM-defined exits and how to use them. The chapter describes how to choose which exits to implement, and what to consider when writing an exit routine.

- Chapter 4, "JES2 Programmer Macros," describes the JES2 programmer macro instructions that you can use when writing JES2 exit code.

- Appendix A, "Acronym List," defines those acronyms used in this book.

- Appendix B, "Hints for Coding JES2 Exit Routines," presents several hints and suggestions for writing JES2 user exits to improve code readability, debugging capability, migration ease, and reduction of coding errors.

- Appendix C, "JES2 Macro List," provides a summary listing of all JES2 macros. The list contains abbreviated descriptions of and uses for each macro instruction.

Related Publications

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

Abbreviated Title	Base Publication	Order Number
SYSTEM INSTALLATION AND INITIALIZATION		
Routing and Descriptor Codes	*OS/VS2 Message Library: Routing and Descriptor Codes*	GC38-1102
System Codes	*OS/VS2 Message Library: VS2 System Codes*	GC38-1008
MVS Initialization and Tuning	*OS/VS2 System Programming Library: Initialization and Tuning*	GC28-1029
Service Aids	*OS/VS2 System Programming Library: Service Aids*	GC28-0674
System Generation Reference	*OS/VS2 System Programming Library: System Generation Reference*	GC26-3792
System Management Facilities (SMF)	*OS/VS2 System Programming Library: System Management Facilities*	GC28-1030
Job Management	*OS/VS2 System Programming Library: Job Management*	GC28-1303
Data Management	*OS/VS2 System Programming Library: Data Management*	GC26-3830
Conversion Notebook	*MVS/SP Version 1 Conversion Notebook*	GC28-1122
JES2 Logic	*JES2 Logic*	LY24-6006
JES2 Initialization and Tuning	*System Programming Library: JES2 Initialization and Tuning*	SC23-0046
ACF/VTAM	*ACF/VTAM Version 2 Planning and Installation Reference*	SC27-0610

Abbreviated Title	Base Publication	Order Number
OPERATIONS		
JES2 Commands	*Operator's Library: JES2 Commands*	SC23-0048
System Commands	*Operator's Library: OS/VS2 MVS System Commands*	GC28-1206
Remote Terminals (JES2)	*Operator's Library: OS/VS2 Remote Terminals (JES2)*	GC38-0225
	Operator's Library: ACF/VTAM Operation	SC27-0466
USER		
JCL	*MVS/370 JCL User's Guide*	GC28-1349
	and/or *MVS/370 JCL Reference*	GC28-1350
System Messages	*OS/VS2 Message Library: System Messages*	GC38-1002
Linkage Editor and Loader	*OS/VS2 Linkage Editor and Loader*	GC26-3813
Utilities	*OS/VS2 MVS Utilities*	GC26-3902
Assembler Reference	*Assembler H Version 2 Application Programming: Language Reference*	GC26-4037
Assembler	*Assembler H Version 2 Application Programming: Guide*	SC26-4036
	TSO Extensions: Interactive Data Transmission Facility User's Guide	SC28-1104
HARDWARE		
	IBM 3704 and 3705 Communications Controllers	GC30-3008
	IBM 3800 Printing Subsystem Programmer's Guide	GC26-3846
	IBM System/3 MRJE/WS Support Reference Manual	GC21-7621
	System/370 Special Feature: Channel-to-Channel Adapter	GA22-6983
	OS/VS2 IBM 3540 Programmer's Reference	GC24-5111
GENERAL		
DP Glossary	*IBM Vocabulary for Data Processing, Telecommunications, and Office Systems*	GC20-1699

Referenced Program Products

Listed below are those program products and product numbers referred to in this book:

Program Product	Number
MVS/System Product-JES2 Version 1	5740-XYS
TSO/E Interactive Data Transmission Facility	5665-285

Contents

Figures

Summary of Amendments

Summary of Amendments
for LC23-0067-3
MVS/System Product – JES2

The following provides an overview of those sections that were revised for MVS System Product - JES2 Version 1, Release 3.6.

Technical Revisions for this Release

Documentation was added for the following:

- Greater than 99 Printer Support Enhancement, that is, printer support for up to 9999 "locally-defined" printers.
- Dynamic SPOOL Offload Data Set Allocation introduced by APAR OZ92050
- New macros -- $DCBDYN

Book Reorganization and Editorial Revisions

- A new appendix, Appendix D, "JES2 Exit Usage Limitations" is added to provide a table listing those instances when the reader/converter exits (2, 3, 4, 6, and 20) are invoked or not invoked.
- All sections include minor editorial and technical changes.

Summary of Amendments
for LC23-0067-2
MVS/System Product - JES2

The following list provides an overview of those sections of this book that were revised for MVS System Product - JES2 Version 1, Release 3.6.

Technical Revisions for this Release

Fourteen new executable macros are available and a number of existing macros have been updated to support the enhancements made to the JES2 product with this release.

- New Macros:
  - $DCTDYN - Call the Dynamic DCT Service Routine
  - $DTEDYN - Call the Dynamic DTE Service Routine
  - $DTETAB - Build and Map DTE Tables
  - $MODCHK - Call the Module Verification Routine
  - $NHDGET - Get Network Data Set Header
  - $POSTQ - Quick Post Facility
  - $SCANCOM - Call the $SCAN Facility Comment Service Routine
  - $SCAND - Call the $SCAN Facility Display Service Routine
  - $STCK - Call the $STCK Service Routine
  - $TGMSET - Call the $TGMSET Service Routine
  - $VERIFY - Call the $VERIFY Service Routine
  - $VERTAB - Build the Inline Verification Tables
  - $WSSETUP - Set Work Selection Values
  - $WSTAB - Map and Generate the Work Selection Table Entries
- Modified Macros:
  - $#ADD
  - $#GET
  - $#JOE
  - $#REM
  - $AMODE
  - $BLDTGB
  - $CKPT
  - $DCTTAB
  - $ERROR
  - $EXTP
  - $GETABLE
  - $GETMAIN
  - $JCTIO
  - $MODULE
  - $MODMAP
  - $PCETAB
  - $PURGE
  - $QGET
  - $RETURN
  - $SCAN

  - $SCANB
  - $SCANTAB
  - $STMTLOG
  - $TIDTAB
  - $TRACK
  - $WTO

- Deleted Macros

  - $#JCT

  - $BADTG

- New IBM-Defined User Exits

  One new exit is available with the support added with this release.

  - Exit 26 (Termination/Resource Release)

    During JES2 termination, Exit 26 allows the user to free resources obtained by an exit routine during previous JES2 processing. This exit can provide this function for any JES2 termination ($P JES2, initialization termination, or abend).

Book Reorganization and Editorial Revisions

- Appendix C, "Mapping Macro Dependency List" has been deleted. There is no longer a need for this material because the JES2 $MODULE macro determines what the dependencies are and automatically generates the required JES2 and MVS macros.

- Appendix C, "JES2 Macro List" has been added to summarize all the JES2 macros. This list contains abbreviated descriptions and uses of each macro instruction.

- All sections include minor editorial and technical changes.

Summary of Amendments
for LC23-0067-1
As Updated October 12, 1984
By TNL LN28-1005
MVS/System Product – JES2

Documentation for Virtual Storage Constraint Relief (VSCR), Resource Access Control Facility (RACF), a listing of JES2 macro dependencies, and minimum volume spool partitioning are added by this TNL.

- Virtual Storage Constraint Relief -- three existing macros have been modified and two new macros created to provide support for JES2 exploitation of MVS/Extended Architecture. These are:
  - Macro changes -- $PCETAB, $PGSRVC, $TRACE
  - New macros -- $AMODE, $PCEDYN
- RACF -- addition of job authorization functions controlled through new RACF options added in RACF Version 1, Release 6, causing changes to Exit 16 and two existing macros.
  - Exit changes -- Exit 16
  - Macro changes -- $MODULE, $WTO
- Minimum volume spool partitioning -- enhancements to the method of spool volume allocation causing changes to Exit 11 and Exit 12.
- Appendix C, "Mapping Macro Dependency List," is added to provide a consolidated listing of those JES2 macros that have dependencies on other JES2 and/or MVS macros.

Chapter 1: Introduction to Modifying JES2

Chapter 1. Introduction to Modifying JES2

Modifying JES2

JES2 is a general job entry subsystem of MVS and sometimes cannot satisfy all installation-specific needs at a given installation. If you modify JES2 code to accomplish your specific functions, you then are susceptible to the migration and maintenance implications that result from installing new versions of JES2. JES2 exits allow you to modify JES2 processing without directly affecting JES2 code. In this way you keep your modifications independent of JES2 code, making migration to new JES2 versions easier and making maintenance less troublesome.

Caution:

You should not view the JES2 exits defined by IBM as an unchanging interface; however, IBM will attempt to make any needed changes as transparent as possible to the user.

Only an experienced system programmer should attempt to make use of the JES2 exits. The writing of an exit routine and the installation of a new exit require a thorough knowledge of system programming, of JES2 programming conventions, and of JES2 internals. If, without having this knowledge, you attempt to write exit routines or to install new exit points, you run the risk both of seriously degrading the performance of your system and causing complete system failure.

Therefore, defining exits and writing exit routines is intended to be accomplished primarily by experienced system programmers; the reader is assumed to have an advanced knowledge of JES2 internals.

In certain areas of JES2 processing, you may still want to customize JES2 to a greater extent than that made possible by the standard options, JES2 initialization statements and their parameters, JES2 operator commands, JES2 control statements, and JES2 exits. While you are not constrained from making more extensive changes by directly altering the JES2 source code, direct alteration can eventually result in problems. For instance, you would have to adapt IBM maintenance code before applying it to your altered JES2 source code and, during migration, there would be no simple way of transferring your alterations to a new release. To avoid this class of problems, use the JES2 exit facility to its fullest extent; excessive alteration of JES2 code is not recommended.

Figure 1-1 illustrates many of those areas where you can modify JES2 processing using the JES2 exit facility.

- *Initialization Processing*

  You can modify the JES2 initialization process and incorporate your own installation-defined initialization statements in the initialization process. Also you can change JES2 control blocks prior to the end of JES2 initialization.

- *Job Input Processing*

  You can modify how JES2 scans and interprets a job's JCL and JES2 control statements. Also, you can establish a job's system affinity, execution node, and priority assignments before the job actually runs.

- *JES2-to-Operator Communications*

  You can tailor how JES2 communicates with the operator and implement additional operator communications for various installation-specific conditions. Also you can preprocess operator commands and alter, if necessary, subsequent processing.

- *Spool Processing*

  You can alter how JES2 allocates spool space for jobs.

- *JES2-SMF Processing*

  You can supply to SMF added information in SMF records.

- *Output Processing*

  You can selectively create your own unique print and punch separator pages for your installation output on a job, copy, or data set basis.

- *RJE Processing*

  You can implement additional security checks to control your RJE processing and gather statistics about signons and signoffs.

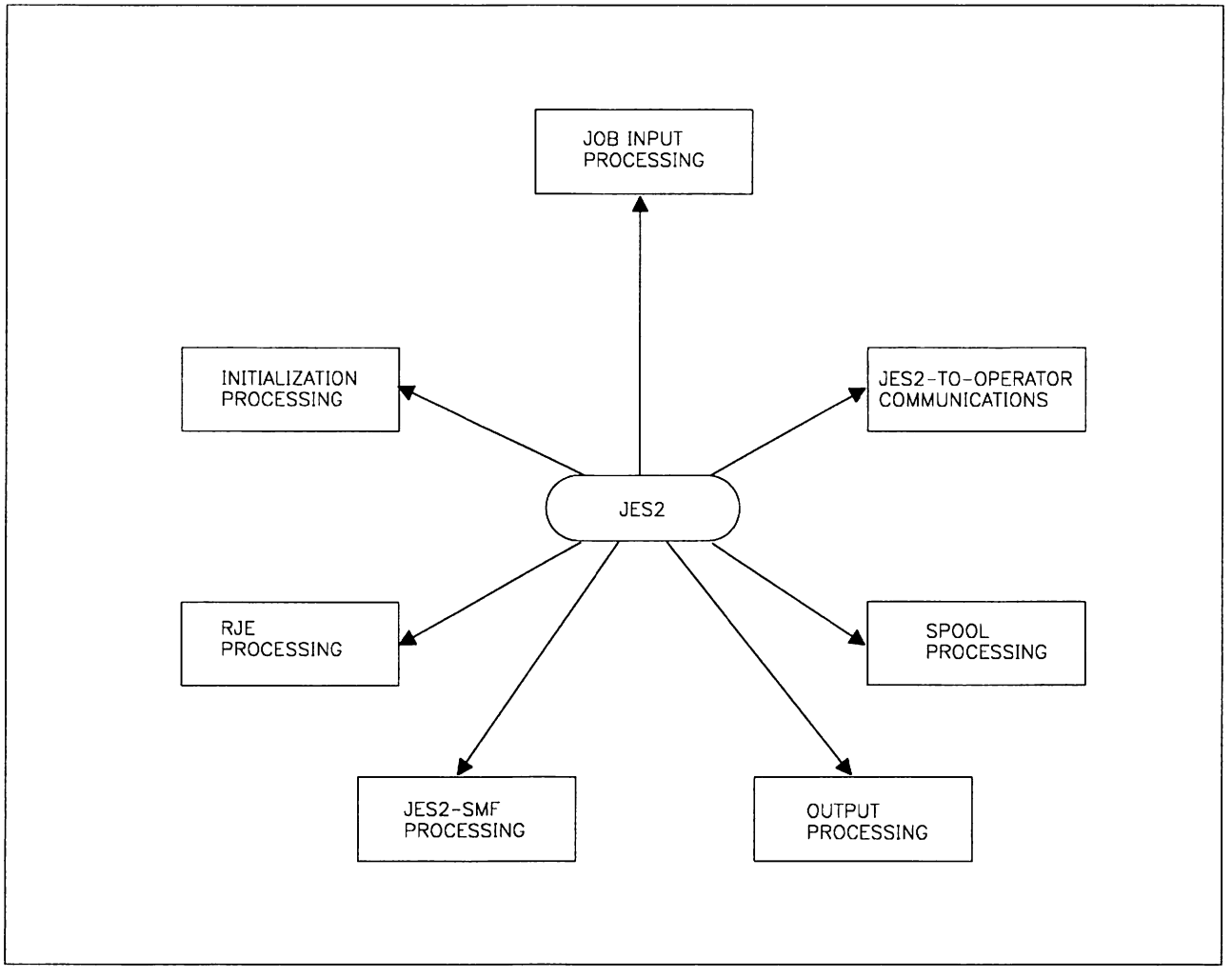


Figure 1-1. Areas of JES2 Modification

What Is a JES2 Exit?

JES2 exits provide a clean, convenient, relatively stable interface between JES2 and your user-written code. Installation-written exit routines are invoked from standard JES2 processing at various strategic locations in JES2 source code. These strategic locations in JES2 source code are called *exit points*. A JES2 exit is established by one or more exit points.

An exit point is defined by the $EXIT macro and, as illustrated in Figure 1-2, is the exact location in JES2 code where JES2 can pass control to your *exit routine* (that is, your user-written code). The JES2 exit, identified by the "exit-id code" of ppp, is defined by one exit point at label JLBL in the JES2 code. It is at JLBL in JES2 processing that JES2 passes control to your exit routine.

To use the exit facility you perform the following steps, as illustrated in Figure 1-2.

1. Package your code into one or more exit routines, identifying each exit routine with an entry point name. (In Figure 1-2 there is a series of exit routines noted as entry points X1...Xn.) Then include the exit routine in a load module. In this case LMOD is the load module containing the exit routine. You can also include the exit routine in the load module for the JES2 code if you wish instead of creating a separate load module.

2. In the JES2 initialization deck include the LOAD initialization statement, which causes your exit routine's load module to be loaded, and the EXITnnn initialization statement, which identifies your exit routines' entry point to the exit point in the JES2 code.

   The EXITppp initialization statement matches the exit point "ppp" at label JLBL for the $EXIT macro in the JES2 code. The EXITppp initialization statement identifies the label "X1" as the entry point of the exit routine for exit point "ppp." The LOAD initialization statement identifies LMOD as the load module to be loaded into storage.

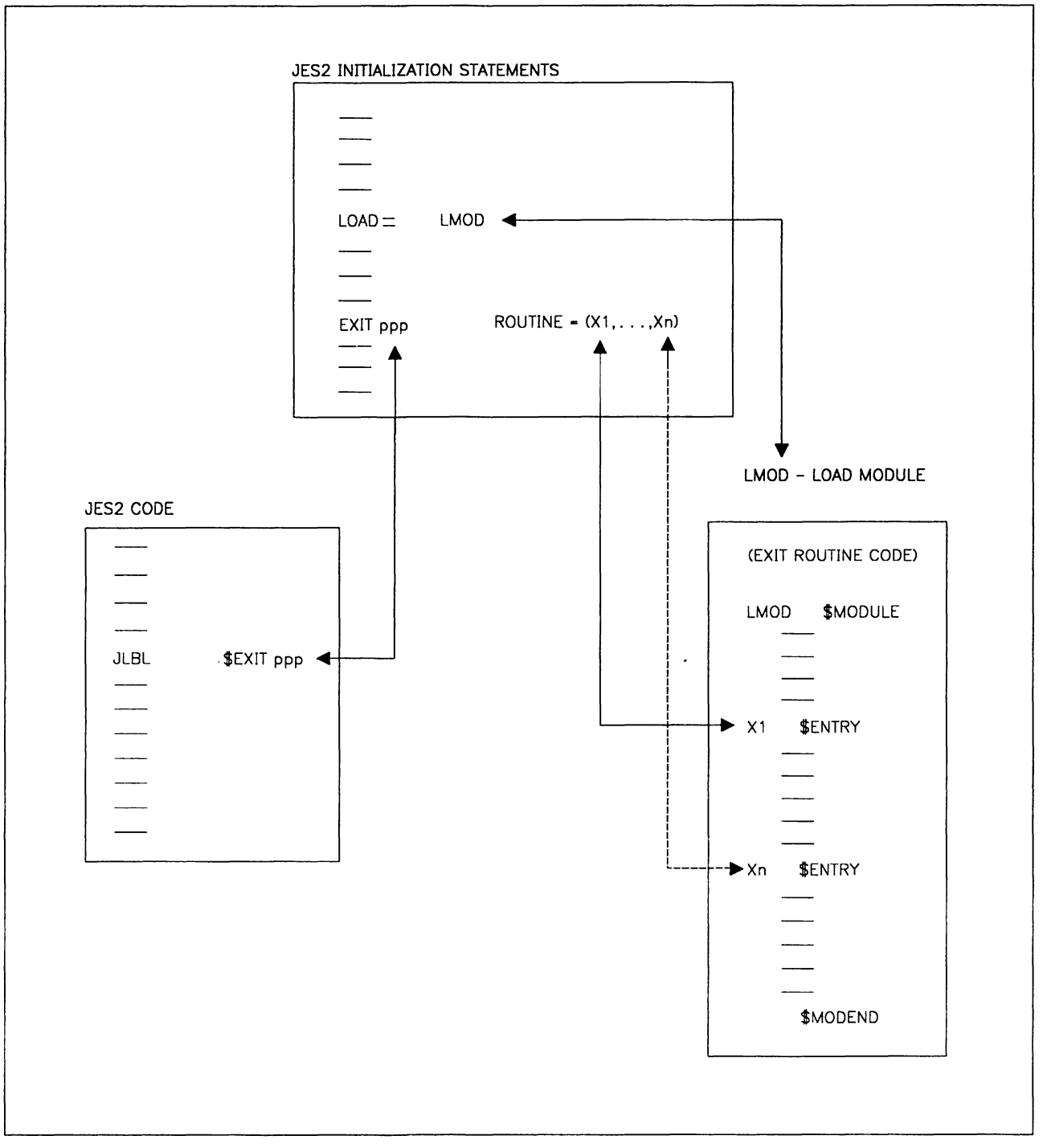


Figure 1-2. A JES2 EXIT

JES2 can have up to 256 exits, each identified by a number from 0 to 255. You specify the number on the required "exit-id code" parameter on the $EXIT macro.

This exit-id code identifies the JES2 exit. When more than one exit point is defined for a single exit, the $EXIT macros that defined the multiple exit points have unique labels but are all specified with the same exit-id code - see Figure 1-3.

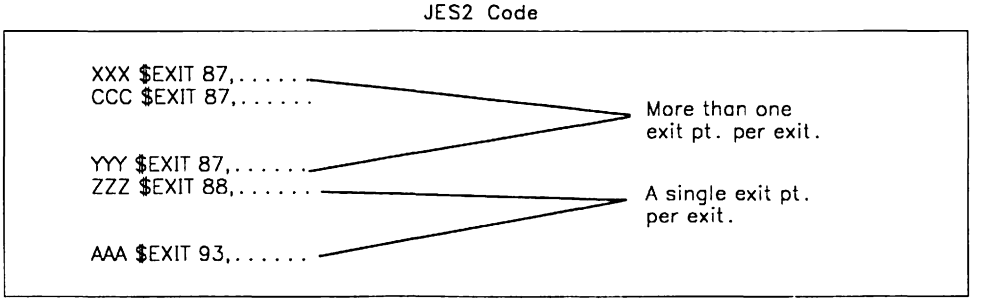


Figure 1-3. EXIT Point Variations

JES2 code includes a number of *IBM-defined exits*. That is, various exit points - via the $EXIT macro - have already been strategically placed in the JES2 code. The intended purpose of each of these exits is summarized below in the EXIT Selection Table - Figure 2-1. For these IBM-defined exits you need only write your own exit routines and incorporate them via the EXITnnn initialization statement. The selection of the point in JES2 code where the exit point should be placed has already been done for you. To ensure a proper implementation you should thoroughly understand the IBM-defined exit and its JES2 operating environment. A comprehensive description of each exit is presented in Chapter 3.

Also, the JES2 exit facility allows you to establish your own exits, should the IBM-defined exits not suffice. Exits established by you are modifications to JES2 and are called *installation-defined* exits, and you define them by placing the $EXIT macro yourself at appropriate points in the JES2 code (or in your own exit routine code). Note, however, that implementing your own exit can be considerably more difficult than writing an exit routine for an IBM-defined exit. You should realize that in establishing your own exits, you run a greater risk of disruption when installing a new version of JES2 code. The new JES2 code into which you have placed your exits may have significantly changed since your $EXIT macros were inserted. For more information, see "Establishing Installation-Defined Exits" in the next chapter.

Every exit, both IBM-defined and installation-defined, has a status of *enabled* or *disabled* which is set at initialization via the EXITnnn initialization statement and which can be dynamically altered by the $T EXIT operator command. When an exit is enabled, JES2 checks for the existence of an associated exit routine. If none are found, standard JES2 processing continues. When an exit is disabled for a particular job (by use of the job mask), it is automatically bypassed by standard JES2 processing; that is, JES2 does not pass control to an associated exit routine. However, if associated exit routines are present JES2 passes control to them. For

certain exits, called *job-related exits*, (refer to "Job-Related Exits" in Chapter 2) the status can be altered on a job-by-job basis by the action of an exit routine.

Environment

JES2 operates in four environments: JES2 main task, JES2 subtask, user address space, and functional subsystem address space. JES2 is comprised of the following four load modules HASPINIT, HASPSSSM, HASJES20, HASPFSSM. Your exit routine receives control as fully-authorized extensions of JES2, and as such receives control in one of these four environments depending on where the associated exit point is placed. JES2 main task and subtask exit points exist in the HASJES20 load module. Your exit routine has access to various control blocks and service routines to which the standard JES2 code has access at the exit point, and it runs with the same authorization as the JES2 code from which your exit routine was invoked. Exit routines invoked from the JES2 address space run in supervisor state in either the JES2 main task or JES2 subtask environment with a protect key of "1." Exit routines invoked from the user address space (SSSM) execute in the key current at the time the $EXIT macro was issued. Exit routines invoked from the functional subsystem (FSS) address space run in the FSS environment and usually run in protect key 1 (as set by the FSS). Also, exit routines invoked from the FSS address space have access to all service routines supported by HASPFSSM.

A JES2 *exit effector* provides linkage services between an exit point and exit routines. It locates and passes control to your exit routines and returns control to JES2. There are two exit effectors: one provides linkage to exit routines that run as extensions to the JES2 main task and the other provides linkage to exit routines that run as extensions to JES2 subtasks or as extensions to routines in the user address space. Exit routines that are invoked as extensions to the JES2 main task receive control in the JES2 protect key. Exit routines that are invoked as extensions to routines in the user address space receive control in the key of the current PSW.

Your exit routines can affect JES2 processing by directly manipulating JES2 data areas and by passing back return codes. You can have up to 256 individual exit routines associated with a single exit on the EXITnnn initialization statement. These *multiple exit routines* are all called consecutively in the order of their appearance on the EXITnnn initialization statement. Consider the example below:

```
EXIT175 ROUTINE= (X1, X2, X3, X4, X5, .....)
```

For exit 175, the exit routine identified by label X1 is called before the exit routine identified by X2, and so forth, until all of them (X1 through X5) are called or until one of them generates a nonzero return code, which causes the exit effector to return to the JES2 mainline after the exit point.

Note: No exit routines are ever required as part of standard JES2 processing. The JES2 exit facility is fully optional. If you have not implemented an exit--that is, if you have not written an exit routine for it, or have not included the exit routine in a load module, or have not associated the routine with the exit at initialization time--the presence of the exit point or points that establish the exit is transparent during standard JES2 processing.

Chapter 2: Customizing JES2

Chapter 2. Customizing JES2

Choosing Which Exits to Implement

When considering an alteration to a standard JES2 function, you should determine whether one of the IBM-defined exits accommodates your intended change.

Each exit has an intended purpose. If you use an IBM-defined exit for other than its intended purpose, you increase the risk of performance degradation and system failure.

The exit selection table (Figure 2-1) summarizes the available exits and their functions.

Exit Selection Table

Exit	Exit Title	Purpose	Some Specific Uses
0	PRE-INITIALIZATION	Control the initialization process	• Provide verification of JES2 initialization options, specifically $HASP426 and $HASP427 messages. • Acquire user control blocks and user work areas for use in initialization (such as the user control table (UCT)). • Provide addresses of user tables in the master control table (MCT). • Determine whether or not JES2 initialization is to continue. • Allow implementation of installation-defined initialization options and parameters.
1	JES2 PRINT/PUNCH JOB SEPARATOR	Create your own print and punch job separators and control production of standard separators.	• Selectively produce unique separators or variations on the standard separators. • Unconditionally produce standard separators. • Unconditionally suppress production of standard separators. • Selectively produce separators for particular users or particular job classes. • Provide a different separator card on a punch device. • Place the company's logo on header page. • Provide accounting information on trailer page.
2	JOB STATEMENT SCAN	Scan the complete JOB statement image and set corresponding fields in the appropriate JES2 control blocks.	• Alter JOB statement parameters including a job's class, priority, and other attributes. • Supply additional JOB statement parameters. • Selectively cancel or purge jobs. • Set the job exit mask in the JCT for subsequent exits. • Set the spool partitioning mask in the JCT. • Initialize or modify other fields in the JCT, including your own installation-defined fields. • Modify other job-related control blocks. • Build your own installation-defined job-related control blocks. • Enforce security and standards.

Figure 2-1 (Part 1 of 5). Exit Selection Table

Exit	Exit Title	Purpose	Some Specific Uses
3	JOB STATEMENT ACCOUNTING FIELD SCAN	Scan the JOB statement accounting field and set corresponding fields in the appropriate JES2 control blocks.	• Alter accounting field information. • Supply additional accounting field information. • Perform your own accounting field scan. • Process nonstandard accounting information. • Selectively cancel jobs. • Set the job exit mask in the JCT for future exits. • Initialize or modify other fields in the JCT, including your own installation-defined fields. • Pass information to subsequent exits through the JCT user fields. • Modify other job-related control blocks. • Enforce security and standards.
4	JCL AND JES2 CONTROL STATEMENT SCAN	Scan JCL (not including JOB statements).	• Alter JCL parameters. and JES2 control statements. • Supply additional JCL parameters. • Supply a JCL continuation statement. • Alter JES2 control statements. • Supply an additional JES2 control statement. • Perform your own JES2 control statement processing. • Suppress standard JES2 processing. • Process your own installation-defined JES2 control statement subparameters. • Selectively cancel or purge jobs. • Enforce security and standards.
5	JES2 COMMAND PREPROCESSOR	Process JES2 commands received by the JES2 command processor	• Alter received commands. • Alter particular fields, such as those pertaining to command authority, in the command processor work area for the PCE to affect subsequent command processing. • Perform your own command validation checking. • Process your own installation-defined commands, operands, and suboperands. • Selectively terminate command processing and notify the operator of command cancellation.
6	INTERNAL TEXT SCAN	Scan internal text after conversion from individual JCL images and after all of the internal text for a particular job has been created.	• Scan the resolved JCL, including PROCLIB expansion that will be used by the job. • Modify individual internal text images. • Enforce security and standards.
7	JCT READ/WRITE (JES2)	Receive control whenever JCT I/O is performed by the JES2 main task.	• Read or write your own installation-defined job-related control blocks to spool along with the reading and writing of the JCT.

Figure 2-1 (Part 2 of 5). Exit Selection Table

Exit	Exit Title	Purpose	Some Specific Uses
8	JCT READ/WRITE (USER)	Receive control whenever JCT I/O is performed by a JES2 subtask or by a routine running in the user address space (HASPSSSM).	• Read or write your own installation-defined job-related control blocks to spool along with the reading and writing of the JCT.
9	JOB OUTPUT OVERFLOW	Receive control whenever an executing job is producing more output than was estimated.	• Selectively allow JES2 to follow the defined output overflow error procedure. • Selectively direct JES2 to take special action for the current job only to: – Cancel the job – Cancel the job with a dump – Allow the job to continue – Extend the job's estimated output to a specific new limit – Control how often the output overflow message is displayed – Suppress the default error message
10	$WTO SCREEN	Receive control whenever JES2 is ready to queue a $WTO message.	• Scan messages. • Change the text of a message. • Alter a message's console routing. • Selectively suppress messages.
11	SPOOL PARTITIONING ALLOCATION - $TRACK	Receive control from the main task when there are no more track groups available on the spool volumes from which the current job is permitted to allocate space.	• Expand the spool partitioning mask. • Suppress spool partitioning by allowing JES2 to use the allocation default.
12	SPOOL PARTITIONING ALLOCATION - $STRAK	Receive control from JES2 subtask or user address space (HASPSSSM) when there are no more track groups available on the spool volumes from which the current job is permitted to allocate space.	• Expand the spool partitioning mask. • Suppress spool partitioning by allowing JES2 to use the allocation default.
13	TSO/E INTERACTIVE DATA TRANSMISSION FACILITY SCREENING AND NOTIFICATION	Enhance the TSO/E interactive data transmission facility by screening incoming files at the receiving network node and notify the TSO receiver that a transmitted file has arrived.	• Perform validity checking on the control information for an incoming file and on the basis of this check: – Delete the transmitted file, not allowing it to reach the intended receiver. – Reroute the transmitted file to a TSO user other than the intended receiver. – Allow the transmitted file to remain targeted for the sender's intended receiver. • Direct JES2 to notify the receiver, via a TSO SEND command, that a transmitted file has arrived. In a multi-access spool configuration specify the system on which the TSO SEND command will notify the receiver.

Figure 2-1 (Part 3 of 5). Exit Selection Table

Exit	Exit Title	Purpose	Some Specific Uses
14	JOB QUEUE WORK SELECT	Receive control to search the job queue for work.	• Use tailored search algorithms to select work from the job queue. • Selectively bypass searching the job queue for work.
15	OUTPUT DATA SET/COPY	Receive control to handle the creation of separator pages on a data set or copy basis.	• Selectively generate separator pages for each data set to be printed. • Selectively generate separator pages for each copy made of a data set. • Selectively vary the number of copies made of a data set. • Selectively pick data sets and generate separator pages for them. • Change default print translation tables.
16	NOTIFY	Receive control to examine or modify messages that are sent.	• Alter routing of the notify message. • Examine the notify message before it is sent to the receiver and make selective modifications. • Suppress sending the notify message to the receiver. • Replace the notify message before it is sent to the receiver with an entirely new one.
17	BSC RJE SIGN-ON/SIGN-OFF	Receive control to manage and monitor RJE operations for BSC.	• Selectively perform additional security checks over and above the standard password processing of the sign-on card image. • Selectively limit both the number and types of remote devices that can be on the system at any one time. • Selectively bypass security checks. • Implement installation-defined scanning of sign-on card images. • Collect statistics concerning RJE operations on the BSC line and report the results of the activity.
18	SNA RJE LOGON/LOGOFF	Receive control to manage and monitor RJE operations for SNA.	• Selectively perform additional security checks over and above the standard password processing of the logon image. • Selectively limit both the number and types of remote devices that can be on the system at any one time. • Selectively bypass security checks. • Implement installation-defined scanning of images. • Collect statistics concerning RJE operations on the line and report the results of the activity.

Figure 2-1 (Part 4 of 5). Exit Selection Table

Exit	Exit Title	Purpose	Some Specific Uses
19	INITIALIZATION STATEMENT	Receive control for each initialization statement.	• Insert installation initialization statements. • Scan an initialization statement prior to the JES2 scan and perform parameter checking. • Selectively alter values supplied on an initialization statement to meet specific installation needs. • Optionally cause JES2 to bypass a particular initialization statement. • Optionally cause JES2 to terminate.
20	END OF JOB INPUT	Alter the status of the job at the end of job input.	• Selectively assign a job's system affinity, execution node, and priority based on an installation's unique requirements and processing workload. • Based on an installation's own defined criteria, terminate a job's normal processing and selectively print or not print its output. • Provide job tracking
21	SMF RECORD	Receive control when JES2 is about to queue an SMF buffer.	• Selectively queue or not queue the SMF record for processing by SMF. • Obtain and create SMF control blocks prior to queueing. • Alter content and length of SMF control blocks prior to queueing.
22	CANCEL/STATUS	Receive control to implement an installation's own algorithms governing job selection and ownership for TSO CANCEL/STATUS.	• Allow an installation to implement its own algorithms for job queue searching and for TSO CANCEL/STATUS.
23	FSS JOB SEPARATOR	Receive control to modify the job separator page data area (JSPA) that is used by page-mode printers such as the 3800-3 to generate the job separator page for an output group	• Control what information is passed to a page-mode printer functional subsystem application (FSA) via the JSPA. • Suppress the printing of job separator pages. • Suppress the printing of the JESNEWS data set.
24	POST INITIALIZATION	Receive control to make modifications to JES2 control blocks prior to the end of JES2 initialization.	• Make final modifications to selected JES2 control blocks prior to the end of JES2 initialization. • Initialize any special installation-defined control blocks. • Terminate JES2 during the initialization process.
25	JCT READ (FSS)	Receive control whenever JCT I/O is performed by a JES2 functional subsystem address space (HASPFSSM).	• Read or write your own installation-defined job-related control blocks to spool along with the reading and writing of the JCT.
26	TERMINATION/ RESOURCE RELEASE	Free resources obtained during previous user exit routine processing during any JES2 termination	• Free resources obtained by user-exit routine processing that JES2 continues to hold following a &P JES2 command, JES2 initialization termination, or JES2 abend.

Figure 2-1 (Part 5 of 5). Exit Selection Table

You can find more detailed information concerning the IBM-defined exits in the section, "The IBM-Defined Exits," of Chapter 3. Chapter 3 contains an exit implementation table (Figure 3-1) that enables you to locate the exit in the JES2 source code itself.

If, after thoroughly examining the IBM-defined exits, you find that none of them suit your intended change, you may want to consider establishing your own optional exit, an exit point you define yourself. However, you should be aware that the installation of an optional exit can be considerably more difficult than writing an exit routine for an existing IBM-defined exit. For more information, see "Establishing Installation-Defined Exits" in this chapter.

Writing an Exit Routine

When you are planning to write a JES2 exit routine, you need to consider the environment in which the exit routine runs and other general programming considerations (such as, the programming language to use to code your exit routine, linkage conventions that are required, return codes to set, and reentrant code requirements to follow). Chapter 3 provides the specific programming considerations you need for writing exit routines for the IBM-defined exits. You should use Chapter 3 in conjunction with the information in this chapter when writing your exit routine. Should you decide to implement your own installation-defined exit in JES2, you need to investigate all the exit-specific programming considerations yourself; see "Establishing Installation-Defined Exits" later in this chapter for more information.

Language

All JES2 source code is written in basic assembler language (BAL), and your exit routines must also be written in BAL.

Operating Environment

JES2 has four distinct execution environments as illustrated in Figure 2-2: user address space, JES2 main task, JES2 subtask, and functional subsystem address space.

1. Some JES2 routines in the load module HASPSSSM (located in PLPA) run in the user's address space. This environment, which enables user programs to interface with JES2, differs greatly from the JES2 address space environment. MVS-wide system services are available to programs in this environment, but the environment is also more complex; it involves the many integrity and synchronization considerations of cross-address space communication.

2. For security reasons, the caller of an installation-defined exit in the user's address space must be either in supervisor state or be an authorized program. JES2 will terminate a calling routine with neither of these attributes (for example, HAMGET and HAMPUT) with a privileged operation exception.

3. The JES2 main task is the most common operating environment for JES2 exits. The JES2 main task and the routines that make it up are included in the primary JES2 load module HASJES20 located in the private area of the JES2 address space. These JES2 main task routines run under the control of the JES2 dispatcher (in HASPNUC). The load module, HASPINIT, which performs JES2 initialization, runs under the main task but is not controlled by the JES2 dispatcher.

4. JES2 subtasks also run in the private area of the JES2 address space but run asynchronously with the JES2 main task. Subtasks run under the control of the MVS dispatcher and their asynchronous operation enables them to perform the wait/post type processing representative of external events and devices without imposing the same wait/post operations on the JES2 main task.

5. The functional subsystem (FSS) resides in the functional subsystem address space (HASPFSSM). You must consider task interaction within the FSS. All data areas and control blocks are not accessible from the FSS. The accessible control blocks are the JIB, FSSCB, and FSACB; and available services are $SAVE and $RETURN.

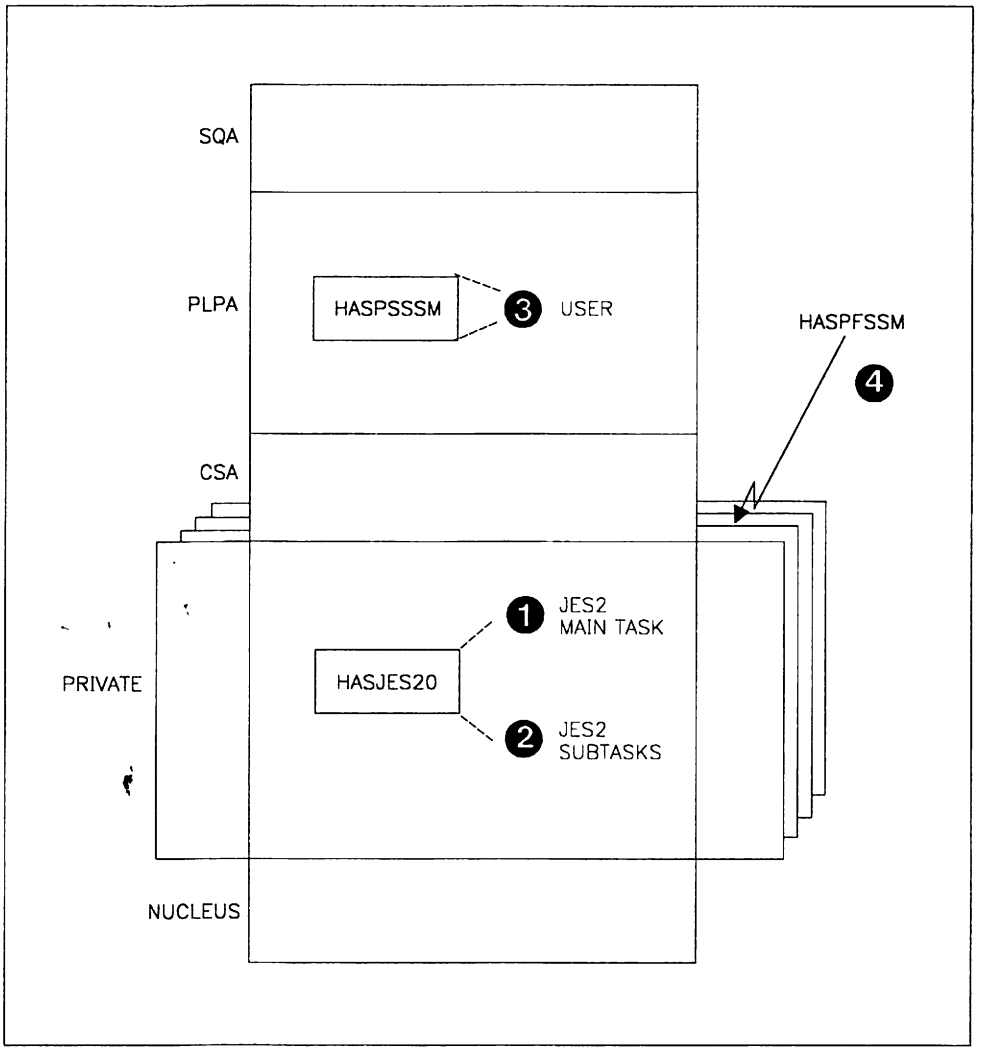


Figure 2-2. JES2 and FSS Address Spaces

When writing an exit routine, you must consider the calling JES2 environment, because your exit routine runs as an extension of that calling environment (JES2 main task, JES2 subtask, user address space, and functional subsystem). The calling environment has broad implications to your exit routine; it determines the JES2 system services available to your exit routine, the reentry considerations you should take into account, the linkage conventions that are necessary, and a number of other essential factors (such as, control block access, synchronization, recovery, and JES2 programmer macro usage). Specifically, the use of macros in exit routines is limited. Before attempting to use a particular macro in an exit routine, be certain to check the "Environment" section of each macro description in Chapter 4 to determine the environments in which the macro can be used.

Every exit is explicitly defined to JES2 as belonging to one of the four execution environments. The ENVIRON= operand of the $EXIT exit-point definition macro is specified as either "JES2," "SUBTASK," "USER," or "FSS." The default $EXIT ENVIRON= value is the same value specified for ENVIRON= on the $MODULE macro. This specification determines which of two exit effectors (the JES2 subroutines that establish the linkage between JES2 and an

exit routine) will be called when the exit is enabled. One exit effector establishes linkage to an exit routine from the JES2 main task environment; the other establishes linkage to an exit routine from either the JES2 subtask environment or the user environment. In the JES2 main task, JES2 linkage conventions (that is, $SAVE and $RETURN) are used; in the subtask and user environments, MVS linkage conventions are used.

Note: If you do not specify the execution environment on either a $EXIT or $MODULE macro, the environment defaults to that specified on the &ANVIRON global assembly macro. The &ANVIRON specification can be explicitly overridden by adding an ENVIRON= statement on the $MODULE to the exit routine.

You cannot define an exit "across" environments. That is, when an exit is required to serve the same purpose in two distinct environments, two separate exits must be defined, each with its own identification number. For example, Exit 11, an IBM-defined exit that can give you control to reset the spool partitioning mask, is defined as belonging to the JES2 main task environment. Exit 12, which serves the same functional purpose, is defined as belonging to the user environment. In implementing these exits, you must write a separate exit routine for each defined exit and adapt the routine to its calling environment.

To stress again, whether defining an exit or writing an exit routine, you must be aware of the operating environment; it influences where your exit is to be defined or what processing your exit routine can really perform. In the descriptions of the following general programming considerations for writing an exit routine, specific environmental influences are described.

Synchronization

An exit routine must use synchronization services appropriate to its calling environment.

An exit routine called from the JES2 main task must use the JES2 $WAIT macro to wait for a JES2 event, resource, or post of a MVS ECB. An exit routine called from a JES2 subtask or from the user environment must use the MVS WAIT macro to wait for a system event. An exit routine called from a functional subsystem must also use MVS WAIT; $WAIT and $POST are not valid in this environment.

A JES2 main task exit routine should *not* invoke operating system services which may wait (WAIT), either voluntarily or involuntarily. An MVS wait from a JES2 main task exit routine would stop all of the JES2 main task processors, including any devices--such as readers, printers, and remote terminals--under their control.

Reentrant Code Considerations

Reentrant code considerations are contingent on the calling environment.

An exit routine called from the JES2 main task must be reentrant in the JES2 sense. The JES2 dispatching unit, commonly called JES2 processors, running under a processor control element (PCE) perform the processing for the JES2 main task. The JES2 dispatcher controls what PCE is currently active (that is, what JES2 processor is currently running). Because a JES2 processor doesn't relinquish control to another JES2 processor involuntarily, an exit routine, invoked out of a JES2 main task processor may use a nonreentrant work area; the work area is serialized as long as the exit routine doesn't issue a $WAIT macro or until the exit routine or service called from an exit routine does issue the $WAIT macro. When the exit routine issues the $WAIT macro directly or through a called routine, control returns to the JES2 dispatcher and the serialization on the nonreentrant work area ceases. The nonreentrant work area may also be passed between exit routines, or between an exit routine and JES2, prior to a $WAIT macro call. Work areas to be used "across" a $WAIT must either be within the processor work area established as part of the processor control element (PCE) or else must be directly owned by the processor. In the same JES2 reentrant sense, an exit routine may search or manipulate a JES2 queue provided that it has ownership of the queue and doesn't issue a $WAIT macro until this action is completed.

An exit routine called from a JES2 subtask, from the user environment (HASPSSSM), or functional subsystem (HASPFSSM) must be reentrant in the MVS sense. The exit routine must be capable of taking an MVS interrupt at any point in its processing. The exit routine must be able to handle the simultaneity of execution with other subtask and user address space, or functional subsystem (FSS) routines and with the JES2 main task.

Linkage Conventions

When control is passed to an exit routine, certain general registers contain linkage information. Register 15 always contains the entry point address of the exit routine, and can be used to establish addressability for the exit routine's code. Register 14 contains the address (in the exit effector) to which the exit routine must return control. In the JES2 main task environment, register 13 always contains the address of the processor control element (PCE) of the processor that invoked the exit. In the JES2 subtask environment or the user (HASPSSSM) environment, register 13 always contains the address of an 18-word save area (as specified for the SAVAREA= operand on the $EXIT macro). In the JES2 main task and subtask environments, register 11 always contains the address of the HCT; and in the functional subsystem environment (HASPFSSM), register 11 always contains the address of the HASP functional subsystem communications table (HFCT). In the user (HASPSSSM) environment, register 11 always contains the address of the SSVT. Depending on the exit, registers 0 and 1 may be in use as parameter registers. The use of registers 2 through 10 and 12, usually used as pointer registers, is also exit-dependent.

The use of registers 0 through 15 is documented, for each IBM-defined exit, in the category REGISTER CONTENTS WHEN CONTROL IS PASSED TO THE EXIT ROUTINE in "The IBM-Defined Exits" reference section in Chapter 3.

Note that if you install an optional installation-defined exit, you are responsible for modifying JES2 code, preceding your exit, to load any parameters in registers 0 and 1 and any pointers in registers 2 through 10 and 12 that may be required by your exit routine.

For multiple exit routines, the exit effector passes registers 2 through 13 to each succeeding exit routine just as they were originally loaded by JES2 when the exit was first invoked. However, register 15 contains the entry point address of the current exit routine and, again, can be used to establish addressability for the exit routine's code. Register 14 contains the address to which the exit routine must return control. This enables you to pass the information to consecutive exit routines. For more information, see "Multiple Exit Routines In A Single Module" below.

When any exit routine receives control, it must save the caller's registers. An exit routine called from either the JES2 main task or a functional subsystem can save the caller's registers by issuing the JES2 $SAVE macro. An exit routine called from a JES2 subtask or from the user (HASPSSSM) environment must save the caller's registers, according to standard MVS linkage conventions, in the 18-word save area whose address is contained in register 13.

When any exit routine relinquishes control, it must restore the caller's registers, with the exception of registers 0, 1, and 15. An exit routine called from either the JES2 main task or functional subsystem must restore the caller's registers by issuing the JES2 $RETURN macro. An exit routine called from a JES2 subtask or from the user (HASPSSSM) environment can restore the caller's registers by reloading them from the save area whose address was originally received in register 13.

Just before returning control to JES2, an exit routine must place a return code in register 15 and must place any parameters that it intends to pass, either back to JES2 or to the next consecutive exit routine, in registers 0 and 1. If the return code is greater than zero, or if the current exit routine is the last or only exit routine associated with its exit, this return code is passed back to JES2 at the point of invocation, along with any parameters placed in registers 0 and 1. If, however, the return code is zero and the current exit routine is not the last or only exit routine associated with its exit, the exit effector passes control to the next consecutive exit routine, along with any parameters placed in registers 0 and 1. Note that in the JES2 main task environment you can use the $STORE macro prior to the $RETURN macro in order to modify the returned values of registers 0 and 1.

The specific implementation of return registers 0, 1, and 15 is documented, for each IBM-defined exit, under the category REGISTER CONTENTS WHEN CONTROL IS PASSED BACK TO JES2 in "The IBM-Defined Exits" reference section in Chapter 3. Note that if you install an optional installation-defined exit, you are responsible for modifying JES2 code, following your exit, to receive any parameters your exit routine passes back as well as any return code above four that you pass in register 15.

Received Parameters

Received parameters, passed by either JES2 or the preceding exit routine in registers 0 and 1, provide a method of passing information to an exit routine and of informing an exit routine of the current point of processing.

For any IBM-defined exit that passes parameters (to the first or only associated exit routine), the specific parameters are documented in the REGISTER CONTENTS WHEN CONTROL IS PASSED TO THE EXIT ROUTINE category of the exit's description in "IBM-Defined Exits" reference section in Chapter 3. IBM-defined Exit 6, which allows you to receive control both during and after the conversion of a job's JCL to internal text, presents a typical example. After a single JCL statement has been converted to an internal text image, Exit 6 places a zero in register 0. After all of the JCL for a particular job has been converted to internal text, Exit 6 places a 4 in register 0. Your exit routine can determine what action to take by checking this code when it first receives control.

For some exits, the parameter registers also contain pointers to control blocks, to certain control block fields, or to other parameter lists. For a discussion of an exit routine's use of control blocks, see the "Control Blocks" section below.

They are passed, as modified, from routine to routine. Note that if you install an installation-defined exit, you must ensure that JES2 passes any parameters required by your exit routine in registers 0 and 1; this may require some modification of JES2 source code.

Return Codes

A return code provides a convenient way for an exit routine to affect the course of subsequent JES2 processing.

The standard return codes are 0 and 4. If 0 is returned by an exit routine that is not the last or the only exit routine associated with its exit, the exit effector calls the next consecutive exit routine. However, a 0 returned by the last or only exit routine associated with its exit directs JES2 to proceed with standard processing. A 4 returned by any exit routine directs JES2 to proceed unconditionally with standard processing; any succeeding exit routines remain uncalled.

Note that a standard return code does not necessarily indicate that an exit routine has opted to take no action. You can write an exit routine to manipulate certain JES2 data areas and then, by generating a standard return code, direct JES2 to continue with normal processing *on the basis of this altered data.* For example, IBM-defined exits, Exit 7 and Exit 8, have no return codes greater than 4.

The definition of return codes that are greater than 4 is exit-dependent. The specific implementation of return codes greater than 4 is documented, for each IBM-defined exit, under the category RETURN CODES in "The IBM-Defined Exits" reference section of Chapter 3; a brief indication of the standard processing that results from the return of 0 or 4 is also included for each exit. Note that if you install an optional installation-defined exit, you are responsible for modifying JES2 code, following your exit, to receive and act upon any return code greater than 4 generated by your exit routine.

A return code is always a multiple of 4. If your exit routine passes a return code other than 0 or another multiple of 4 to JES2, results are unpredictable. Also, the $EXIT exit-point definition macro has a MAXRC= operand that specifies the exit's maximum acceptable return code. If your exit routine generates a return code that exceeds this specification and the exit was called from the JES2 main task, the exit effector issues the $ERROR macro. If the exit was called from a JES2 subtask, from the user (HASPSSSM), or from the functional subsystem (HASPFSSM) environment, the exit effector issues the ABEND macro.

Control Blocks

An exit routine has access to various control blocks available in the environment from which it was called.

To facilitate exit coding, IBM-defined exit routines provide in registers 0-13 pointers to control blocks currently in main storage. Register 1 can contain a pointer to a parameter list, which contains the addresses of control blocks currently in main storage. For a list of the specific pointers provided by an IBM-defined exit, see the REGISTER CONTENTS WHEN CONTROL IS PASSED TO THE EXIT ROUTINE category of the particular exit's description in "The IBM-Defined Exits" reference section of Chapter 3. Note that if you install an installation-defined exit, you have to ensure that any pointers required by your exit routine have been placed in the call registers by JES2 prior to the invocation of your exit; this may require some modification of JES2 source code.

An exit routine can access information available in control blocks. For example, IBM-defined Exit 5, which allows you to perform your own JES2 command preprocessing, passes the address of the PCE to an associated exit routine. You can write your own command validation algorithm by writing an exit routine that checks various command-information fields in the PCE.

Because an exit routine runs fully authorized, it is free to alter any field in any control block to which it has access. By altering specific fields in specific JES2 control blocks, an exit routine can pass information to JES2 and to succeeding exit routines and can thereby affect the course of subsequent JES2 processing. Note that JES2 has no protection against any change made to any control block by an exit routine. If you modify a checkpointed control block, you must ensure that it is written to the checkpoint data set either by your exit routine or by JES2. For this reason, *you should exercise extreme caution in making control block alterations.*

Except where it would seriously degrade system performance, JES2 provides a reasonable amount of space in its standard control blocks for use by your exit routines. Some storage-resident control blocks, such as PCEs and DCTs, have storage reserved for exit routine use. You can use this storage to establish your own exit-related field or fields within a standard control block or, if you require more storage, you can use four of the bytes as a pointer to a work area acquired by an exit routine using the JES2 $GETMAIN, $GETBUF, and $GETWORK macros or the MVS GETMAIN macro. Disk-resident control blocks provide considerably more space for exit routine use. For performance reasons, no checkpoint-resident control blocks reserve space for use by exit routines.

In addition to using reserved space in the standard JES2 control blocks, you can define and make use of your own installation-specific control blocks by using the JES2 exit facility. An exit routine can use the JES2 $GETMAIN, $GETBUF, and $GETWORK macros or the MVS GETMAIN macro to acquire storage and build a control block at the appropriate point in processing. For example, a job-related control block can be built by an exit routine associated with IBM-defined Exit 2. You can then use IBM-defined Exits 7 and 8 to write your installation-defined control blocks to spool and to read them from spool into main storage.

Note that if an exit routine references the symbolic name of a control block field, the DSECT for that control block must be requested in the exit routine's module at assembly time (via the $MODULE macro).

Each exit description in "The IBM-Defined Exits" reference section of Chapter 3 includes a list of DSECTs normally required at assembly.

Service Routine Usage

Many service routines available to the JES2 main task are also available to an exit routine called from the JES2 main task. You can include an executable JES2 macro instruction at any appropriate point in a JES2 main task exit routine. Not all service routines are available to the functional subsystem environment; those that can be called must be appropriate. Depending on the macro, it provides inline code expansion at assembly time or else calls a JES2 service routine, as a subroutine, in execution.

An exit routine called from a JES2 subtask or from the user (HASPSSSM) environment can make use of any JES2 service routine that can be called from its environment as well as any MVS service routine (SVC) that can be called from its environment. You can include a JES2- or MVS-executable macro instruction at any appropriate point in the subtask or user routine. Again, depending on the macro, it provides inline code expansion at assembly time or else calls a JES2 or MVS service routine, as a subroutine, in execution.

Exit Logic

Each IBM-defined exit should be used for its intended purpose. Using an exit for other than its intended purpose can increase the risk of degraded performance and system failure and may cause migration problems.

Within the scope of an exit's intended purpose, you have a wide degree of flexibility in devising exit algorithms. For example, you can base spool partitioning on a simple factor, such as job class, or on a complex comparison of several job attributes and current spool volume usage. However, you should keep in mind that as you increase an algorithm's sophistication, you also increase overhead and the risk of error. Exit-specific logic considerations are provided in the "Other Programming Considerations" category for each exit description in "The IBM-Defined Exits" reference section at the end of this chapter.

Logic considerations for installing installation-defined exits as well as for implementing them are provided in "Establishing User-Defined Exits" later in this chapter.

Note, for both IBM-defined and installation-defined exits, that the ability to associate multiple exit routines with a single exit enables you to devise modular logic segments. Each separate function to be performed after exit invocation can be isolated in its own exit routine. This can be especially useful when you need to provide alternate types of exit processing for different received parameters.

Exit-to-Exit Communication

Communication among exit routines associated with different exits must be accomplished via mutually accessible control blocks. Communication among exit routines associated with the same exit can also be accomplished via mutually accessible control blocks.

Exit-to-Operator Communication

Except for exit routines called from the HASPCOMM module of HASJES20 and exit routines called from JES2 initialization and termination, exit routines called from the JES2 main task environment can communicate with the operator via the $WTO macro. Exit routines called from the HASPCOMM module can communicate with the JES2 operator via the $CWTO macro. Exit routines called from a JES2 subtask or during JES2 initialization and termination can communicate with the operator via the $$WTO and $$WTOR macros or via the MVS WTO and WTOR macros. Exit routines called from the user (HASPSSSM) or functional subsystem (HASPFSSM) environment can communicate with the operator via the MVS WTO and WTOR macros. Note that, if a message is to be associated with jobs processed by a functional subsystem, the job id must be included with the message.

In addition, certain IBM-defined exits provide alternate methods of operator notification. Exits 2, 3, and 4 allow you to transmit an exit-generated message to the operator along with certain return codes by setting a flag in the RXITFLAG byte. Exit 5 allows you to control the standard $CRET macro "OK" message and to transmit your own exit-generated message text via the $CRET macro. Exit 9 allows you to control the standard output overflow message. Exit 10 allows you control over the text and routing of all $WTO messages. For details, refer to the individual exit descriptions in "The IBM-Defined Exits" reference section of Chapter 3.

Other Programming Considerations

The following programming considerations describe some specific requirements for coding your exit routine:

- Naming and Identifying an Exit Routine

  You must begin each exit routine with the JES2 $ENTRY macro, which you use to name the routine and to identify it to JES2. For more information, see "Packaging Exit Routines" later in this chapter.

  Note that you have flexibility in naming your exit routines, in accordance with standard labeling conventions with the exception of Exit 0 (see the description of Exit 0 in Chapter 3 for more detail).

- Exit Addressability

  The $ENTRY macro is also used to generate a USING statement for your exit routine. The BASE= operand is used to specify the register or registers which provide addressability when the exit routine gets control. However, the $ENTRY macro does not load the base register.

- Source Module Conventions

  The construction of a source module must follow certain conventions depending on how you intend to package the exit routine. Through these conventions, JES2 is able to locate both exit routines and exit points within a module.

Figure 2-3 illustrates two ways to package an exit routine:

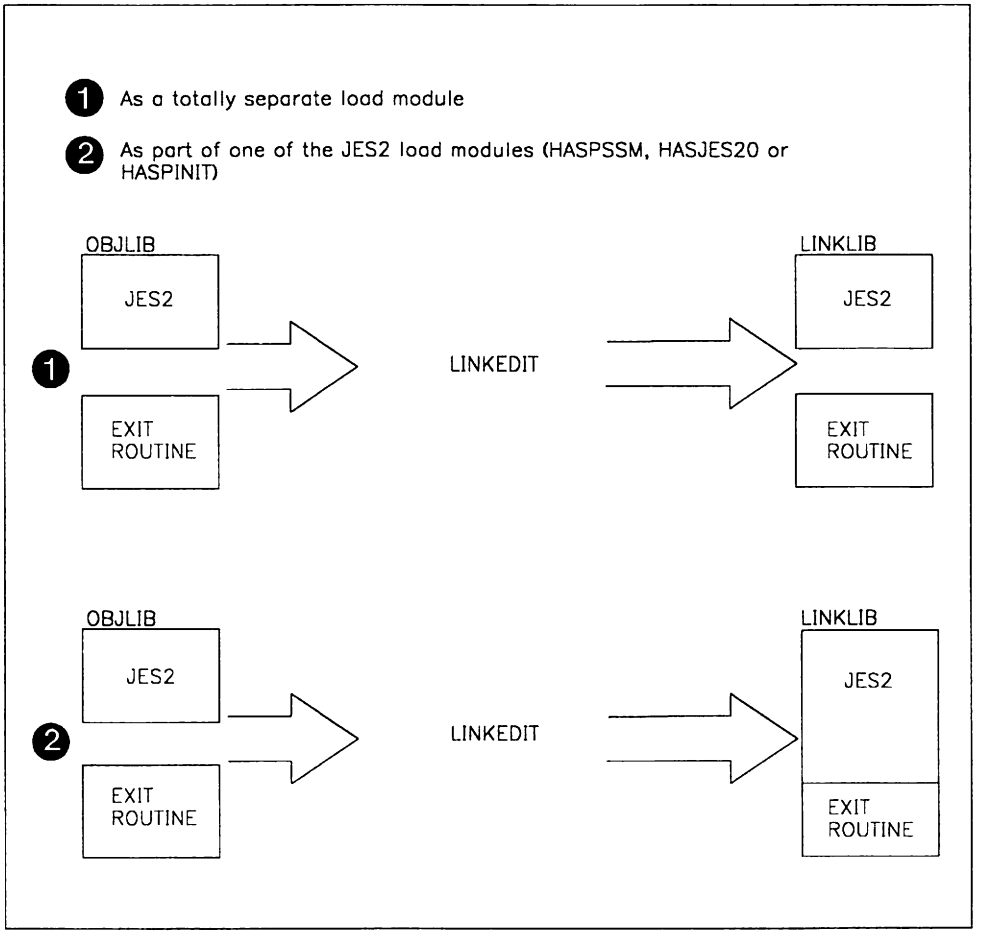


Figure 2-3. Methods of Packaging

A JES2 $MODULE macro must be the first code-generating statement (immediately preceded by COPY $HASPGBL) in a source module to be assembled and either link edited separately and loaded at initialization or a source module to be added to a standard JES2 load module. Note that the $MODULE macro call must occur prior to the first use of $ENTRY or $EXIT, and a JES2 $MODEND macro must be coded at the end of both types of source modules.

You can only code one $MODULE and one $MODEND macro in each source module. Additionally, when link editing exits into their own load modules (other than HASJES20 or HASPSSSM), each source module must be linked into its own load module.

The $MODULE macro reserves space for a module information table (MIT), which points to a MIT entry table (MITETBL); the MIT indicates which exit points are defined within the module and the MITETBL contains the names and addresses of exit routines in the module. When JES2 is initialized, the MIT and MITETBL are used to initialize the exit routines in the system. During assembly, each $ENTRY macro saves the entry point name specified in a global assembly variable and each $EXIT macro turns on a bit for that exit point in another global assembly variable. At the end of the assembly, the $MODEND macro

uses the global assembly variables to assemble the exit bit mask in the MIT and to create the MITETBL.

To locate the MITs of modules that are added to the standard JES2 load modules, JES2 uses weak external address constants. To locate the MIT's of modules that are linked in their own load modules, JES2 assumes that the MIT is located at the front of the load module to which it points. The MITETBL is located at the end of a module loaded at initialization.

Also note, for all exit routine source modules, that if an exit routine references the symbolic name of a control block field, the mapping macro for that control block must be included in the $MODULE macro list in the same source module as the exit routine at assembly time.

Furthermore, refer to Appendix C, "Hints for Coding JES2 Exit Routines" for a list of required mapping macros for individual exits. These macros are environment dependent and must be coded to prevent assembly errors and error messages.

The ENVIRON= operand of the $MODULE macro should be used to specify in which JES2 operating environment the exit routine(s) is to execute. Each exit description in the "IBM-Defined Exits" reference section in Chapter 3 includes a list of MAPPING MACROS NORMALLY REQUIRED AT ASSEMBLY.

Tracing

Minimal tracing of exit invocation can be performed automatically as part of the exit facility. For this tracing to occur, two conditions are necessary:

1. The trace ID for exit tracing (ID 13) must be enabled.
2. The EXITnnn initialization statement or the $T EXITnnn operator command must have enabled tracing. For more information, see "Tracing Status" in "Controlling Exit Status" later in this chapter.

This automatic tracing produces a limited trace entry containing such general information as exit point identification and register contents at the time of exit invocation.

In addition, to further trace execution of exit routine code, issue the standard JES2 $TRACE macro call within an exit routine. This results in a full trace record of exit routine processing.

It is recommended that you use tracing to its fullest extent only in your testing cycle, and that you limit its use in those areas of the standard processing environment--for example, in conversion processing--where it is most likely to degrade system performance.

Recovery

An exit routine *should not* depend upon JES2 for recovery. JES2 cannot anticipate the exact purpose of an exit routine and, therefore, any standard JES2 recovery that happens to be in effect when your exit routine is called is, at best, minimal for your particular needs. In other areas of processing, *no* JES2 recovery environment is in effect, and an exit routine error has the potential to cause JES2 to fail. Consequently, *you should provide your own recovery mechanisms within your exit routines.*

For all exits routines for which you provide an ESTAE routine, also be certain to add the error recovery area DSECT, $ERA, to the $MODULE macro. On return from the exit, use register 1 to point to the ERA and provide a USING statement in that recovery routine to use the ERA address.

You can make use of the standard JES2 $ESTAE recovery mechanisms in implementing your own recovery within the JES2 main task. You can make use of the MVS ESTAE recovery mechanism in implementing your own recovery in the SUBTASK, USER, or FSS environments. At minimum, a recovery mechanism should place a 0 or 4 return code in register 15. Beyond this, recovery depends on the particular purpose of an exit routine.

A Sample Exit Routine

Below is a sample JES2 user exit routine for a separator page (Exit 1). This code is a simplified example to show you how to use the JES2 macros provided with this exit facility.


```
XIT1     TITLE 'SAMPLE JES2 USER EXIT - PREAMBLE'
****** COMMENT BLOCK GOES HERE ********************************** 1
*                                                                *
*      SAMPLE JES2 SEPARATOR PAGE EXIT                           *
*                                                                *
*   PURPOSE:                                                     *
*      TO PRINT A PAGE SEPARATOR ON LOCAL AND REMOTE PRINTERS    *
*      WITH THE JOB NAME IN BLOCK LETTERS, FOLLOWED BY THE JOB   *
*      NUMBER AND SYSOUT CLASS (ALSO IN BLOCK LETTERS), AND      *
*      THEN 40 REPETITIONS OF A DUMMY DETAIL LINE.               *
*                                                                *
*   ENTRY POINT = UEXIT1                                         *
*                                                                *
*   INPUT (REGISTERS):                                           *
*      R0   =0 FOR START-OF-JOB SEPARATOR                        *
*           =4 FOR CONTINUATION SEPARATOR                        *
*           =8 FOR END-OF-JOB SEPARATOR                          *
*      R1   ADDRESS OF PRINTER OR PUNCH DCT                      *
*      R2-9 N/A                                                  *
*      R10  ADDRESS OF THE JOB'S JCT                             *
*      R11  ADDRESS OF THE JES2 HCT                              *
*      R12  N/A                                                  *
*      R13  ADDRESS OF THE PRINTER OR PUNCH PCE                  *
*      R14  RETURN ADDRESS                                       *
*      R15  ENTRY ADDRESS                                        *
*                                                                *
*   RETURN (REGISTERS):                                          *
*      R0-14 SHOULD CONTAIN THE SAME CONTENTS AS ON ENTRY        *
*      R15  CONTAINS A RETURN CODE AS FOLLOWS:                   *
*           =0  JES2 WILL PRODUCE THE STANDARD SEPARATOR         *
*           =4  SAME AS 0, BUT CALL NO OTHER EXIT ROUTINES       *
*           =8  JES2 WILL UNCONDITIONALLY NOT PRINT A SEPARATOR  *
*           =12 JES2 WILL UNCONDITIONALLY PRINT A SEPARATOR      *
*                                                                *
*  JES2 MACROS USED:                                             *
*      $ENTRY, $SAVE, $GETBUF, $SEPPDIR, $PBLOCK, $PRPUT,        *
*      $FREEBUF, $RETURN, $MODEND, $MODULE.                      *
*                                                                *
******************************************************************
        EJECT                                                    2
        COPY   $HASPGBL             COPY HASP GLOBALS            3
HASPUEX        $MODULE RPL,         REQ'D BY $BUFFER           C  4
               $BUFFER,                                        C  5
               $CAT,                                           C  6
               $DCT,                                           C  7
               $HASPEQU,            REQUIRED FOR REG CONVENTIONS C 8
               $HCT,                REQ'D BY $SAVE,$RETURN,ETC. C 9
               $JCT,                                           C 10
               $JOE,                REQ'D TO GET SYSOUT CLASS  C 11
               $JQE,                                           C 12
               $MIT,                REQ'D BY HCT               C 13
               $PCE,                REQ'D BY HCT               C 14
               $PDDB,                                          C 15
               $PPPWORK             REQ'D TO FIND JOE            16
         SPACE 1                                                17
```

[1] The line numbers here relate to the numbered notes in the discussion below.

Figure 2-4 (Part 1 of 2). Sample JES2 Separator Page Exit Routine

```
       TITLE 'SAMPLE PRINT/PUNCH SEPARATOR EXIT'                        18
             USING BFPDSECT,R3    ESTABLISH BUFFER ADDRESSABILITY       19
             USING JOE,R7         ESTABLISH JOE ADDRESSABILITY          20
             USING JCT,R10        ESTABLISH JCT ADDRESSABILITY          21
       SPACE 1                                                          22
UEXIT1       $ENTRY BASE=R12    EXIT ROUTINE ENTRY POINT               23
             $SAVE              SAVE HASPPRPU REGISTERS                24
             LR    R12,R15      SET LOCAL BASE REGISTER                25
             SLR   R15,R15      SET ZERO RETURN CODE (FOR PUN)         26
             TM    PCEID,PCEPUSID  IS DEVICE A PUNCH...               27
             BO    RETURN       RETURN IF YES                          28
      SPACE 1                                                           29
             $GETBUF TYPE=HASP,FIX=YES,WAIT=YES  GET A WORK BUFFER      30
             LR    R3,R1        SAVE ADDRESS OF BUFFER                 31
PDIR         $SEPPDIR (R3)      SEND A PDIR IN CASE A SNA RMT          32
      SPACE 1                                                           33
             LA    R0,JCTJNAME  POINT TO JOB NAME                      34
JOBNAME      $PBLOCK BUFFER=(R3),DATA=(R0),SLANT=YES PRINT JOBNAME      35
             MVC   PCEUSER0(8),$BLANKS      BLANK OUT BUFFER            36
             MVC   PCEUSER0(1),JCTJOBID     GET JOB TYPE                37
             MVC   PCEUSER0+1(4),JCTJOBID+4 AND NUMBER                  38
             L     R7,PPPWKJOE              SET WORK JOE BASE           39
             MVC   PCEUSER0+6(1),JOECURCL   GET SYSOUT CLASS            40
      SPACE 1                                                           41
             LA    R0,PCEUSER0      POINT TO BLOCK PRINT LINE          42
JOBID        $PBLOCK BUFFER=(R3),DATA=(R0),SLANT=YES  PRT JOBID/CL      43
             MVC   BUFSTART(132),=CL132'*'  START JOB JOB# JOBNAME      44
             LA    R1,BUFSTART   POINT TO PRINT LINE                    45
             LA    R0,132        LENGTH OF PRINT LINE                   46
DETAIL       $PRPUT DATA=(R1),LEN=(R0),COUNT=40,WAIT=YES PUT LINES      47
             $FREEBUF (R3)       FREE WORK BUFFER                       48
      SPACE 1                                                           49
             LA    R15,8         SET RETURN CODE                        50
RETURN       $RETURN RC=(R15)    RETURN TO HASPPRPU                     51
      SPACE 1                                                           52
             DROP  R3,R7,R12     DROP ADDRESSABILITIES                  53
             LTORG                                                      54
             $MODEND                                                    55
             END                                                        56
```

Figure 2-4 (Part 2 of 2). Sample JES2 Separator Page Exit Routine

The following notes provide line-by-line explanation, where required, for the example exit presented in Figure 2-4.

1. A comment block should go here. The block (designated by asterisks) is an example of a comment block to document the input, output, and purpose of the exit.

2. N/A

3. As in JES2 source modules, the $HASPGBL member of SYS1.HASPSRC is copied to define many symbols used by JES2 macros and user code in this exit.

4. All exit modules must start with a $MODULE macro as the first code-generating statement. This macro provides a module information table (MIT) and must come before any executable code. The label must be the same as the name of the resulting load module. If it is not the same name, you will receive the $HASP875 MODULE NAME DOES NOT MATCH ITS MIT message. However, if the routine is to be linked with HASJES20, the label must be one of the weak externals provided (that is HASPXJ01, HASPXJ02, ...). (The numbers on these label do not relate to the exit numbers or routines.) The label that you provide on the $MODULE macro will be the name in the MIT.

   $BUFFER is used by $GETBUF, $FREEBUF, and the exit code that uses a JES2 buffer as a work area.

   The JES2 DSECT mapping macros $JQE, $PCE,... are required by other JES2 macros. More DSECTs may have to be added to this list as required by user code or other JES2 macros. You can also include a request for $USERCBS that allows you to define your own DSECTs.

5. BUFFER fields are required.

6. CAT fields are required by the HCT.

7. DCT fields are required by the HCT.

8. $HASPEQU is required to establish register conventions.

9. The HCT is required by the $SAVE and $RETURN macros and also contains the $BLANKS constant referenced in line 36.

10. JCT fields are required by the class attribute table (CAT), which in turn is required by the HASP communications table (HCT).

11. The job output element (JOE) is required by this exit to determine the SYSOUT class. (JOECURCL - see code lines 40.)

12. Job queue element (JQE) fields are required by the HCT.

13. MIT fields are required by the $MODEND macro.

14. Processor control element (PCE) fields are required by the HCT.

15. PDDB fields are required by the HCT.

16. The print/punch work area defined by $PPPWORK contains PPPWKJOE, which is used to find the JOE for this work element.

17. N/A

18. N/A

19. Establish buffer addressability.

20. Establish JOE addressability.

21. Establish JCT addressability.

22. N/A

23. The first executable statement of the user exit routine must be $ENTRY. A label (for example, UEXIT1) must be provided; this is the same name that is coded on the ROUTINE= parameter on the EXITnnn initialization statement (see the last coding example at the end of this section).

24. The $SAVE macro obtains a JES2 save area and save registers 14, 15, 0, and 1 through 12. No registers are destroyed by this macro.

25. This instruction loads the base register. This is necessary because the $ENTRY sets the USING statement; it does not load the base registers.

26. This instruction clears register 15 indicating that JES2 should punch the standard separator card in the event the following branch is taken.

27. Test the PCEID to determine if this is a punch. The device type is in the PCE at label PCEID. Register 13 already points to the PCE. USING is not required because $MODULE contains a 'USING PCE,R13' statement.

28. If the device is a local punch, then return with an indication (R15=0). JES2 should produce the standard separator.

29. N/A

30. The $GETBUF macro gets a JES2 buffer required by the $SEPPDIR and $PBLOCK macros. This buffer is also used as a work area to build print lines.

31. The address of the JES2 buffer is saved for future use. (Register 1 is generally not a safe register to retain this pointer.)

32. If the printer is on an SNA remote work station, this instruction issues the $SEPPDIR macro instruction. It is treated as a no-op for local or BSC remote devices. The buffer is not a keyword parameter, but a positional parameter. Code it as shown in this example.

33. N/A

34. The job name is in the job control table (JCT) at label JCTJNAME. Register 10 already points to the JCT.

35. The $PBLOCK macro can use data directly from a JES2 control block. It needs a JES2 (HASP-type, fixed) buffer to build the print lines. The letters can be slanted (SLANT=YES) or not (the default). In addition, the name is centered as a default. To produce a blank line between these block letters and the next line of block letters, the $PRPUT macro can be used.

In the actual standard JES2 separator page, block letters are only printed if the SEPLINE or RSEPLINE parameters on the PRINTDEF initialization statement are specified as greater than 29. (SEPLINE and RSEPLINE are the JES2 initialization parameters that define the separator line counts for local and remote printers, respectively.)

36. The user fields in the printer PCE (PCEUSER0 and PCEUSER1) are eight contiguous bytes and are used here to build the 8-byte character string to be passed to the $PBLOCK macro. If these user fields are not available to this exit (that is, in use by some other modification or exit) another JES2 buffer can be acquired as a work area. $BLANKS is a double-word constant of blanks in the HCT and is used to blank out the eight bytes where the 5-character job ID and 1-character sysout class will be built.

    Note: The PCEUSERn fields reside above the 16-megabyte address in virtual storage; therefore, these fields cannot also be used as input to $PBLOCK as they are for $PRPUT.

37. The 'job type' used here is either 'J' for batch job, 'S' for started task, or 'T' for time-sharing user.

38. The job number should be left-justified to be more readable (but is not in this example.) See code at label PRINTID in the HASPPRPU JES2 source module for a detailed example. (Refer to *JES2 Logic.*)

39. To access the work JOE, get the pointer in the PPPWORK area (at label PPPWKJOE), which is an extension of the printer PCE.

40. Move the sysout class for this work JOE to the right side of the 8-byte input data area used as input to the PBLOCK routine.

41. N/A

42. N/A

43. Here, the $PBLOCK macro points to data in the same buffer that is being used as a work area by the PBLOCK service routine. The full eight bytes of the data are converted to block letters and printed. The scan does not stop with the first blank character.

    To produce a blank line after the block letters and before the first detail line below, use the $PRPUT macro.

44. This MVC is a dummy instruction where the user code would build the detail print line. For further explanation of how the standard JES2 header/trailer detail print line is built, see the code at labels PDEST and BLKSKIP in the HASPPRPU JES2 source module. (Refer to *JES2 Logic.*)

    Also required is a work area unique to this printer to keep this routine reenterable by JES2. To use the JES2 buffer acquired above (line 30) as a work area to build a print line, refer to the start of the work area (beyond the IOB) by label BUFSTART.

45. N/A

46. N/A

47. The $PRPUT macro is invoked once at this point to print 40 lines of detailed information. The data should be fixed because PRPUT uses EXCPVR. Code WAIT=YES to ensure that the I/O occurs before freeing the buffer (see line 46 below.) Unlike the PBLOCK routine, PRPUT does not need a JES2 buffer to be passed to it.

    Use R1 for the DATA register, and R0 for the LEN (length). Also note that if CC=M is not coded on this macro, NO carriage control is assumed.

    This example shows 40 lines of detail, but the standard JES2 separator actually has the number specified by either the SEPLINE or RSEPLINE parameters on the PRINTDEF statement minus 29 lines. The COUNT parameter must specify an absolute value; it cannot specify a register to hold the count.

48. $FREEBUF is used to free the JES2 buffer obtained as a work area. See line number 28 above.

49. N/A

50. Set register 15 to indicate the appropriate return code to JES2. In this example, it is set to 8 to indicate that:

    - The next exit routine (if any) will not be called.
    - JES2 is not to produce the standard separator page.

51. The $RETURN macro must include RC=(R15) to generate return codes other than 0.

52. N/A

53. N/A

54. Code the LTORG before the $MODEND macro to maintain addressability to the literal constants.

55. Code the $MODEND at the end of every JES2 module. It generates the MIT entry table (MITETBL) and the MIT exit mask.

56. N/A

    Figure 2-5 shows how the above routine can be assembled and link-edited, and how to use the load module name. The source is in SYS1.JESEXITS, and the load module is linked into SYS1.LINKLIB with the name of HASPUEX. This name must also appear on the LOAD initialization statement.


```
//ASM EXEC PGM=IEV90,PARM='OBJECT,NODECK,XREF(SHORT)'
//SYSLIB    DD DSN=SYS1.HASPSRC,DISP=SHR
//          DD DSN=SYS1.MACLIB,DISP=SHR
//          DD DSN=SYS1.AMODGEN,DISP=SHR
//SYSUT1    DD UNIT=SYSDA,SPACE=(1700,(1200,300))
//SYSPRINT  DD SYSOUT=A
//SYSIN     DD DSN=SYS1.JESEXITS(HASPUEX),DISP=SHR
//SYSLIN    DD DSN=&&OBJ,DISP=(,PASS),UNIT=SYSDA,
//             SPACE=(CYL,(1,1))
//LINK EXEC PGM=HEWL,COND=(0,LT,ASM),
//          PARM='XREF,LET,REUS'
//SYSPRINT  DD SYSOUT=A
//SYSUT1    DD UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSLMOD   DD DSN=SYS1.LINKLIB,DISP=OLD
//SYSLIN    DD DSN=&&OBJ,DISP=(OLD,DELETE)
//          DD *
 NAME HASPUEX(R)
/*
```

Figure 2-5. Example of Assembly and Link-Edit of a Installation-Written Routine

The following JES2 initialization statements can be used to load and associate Exit 1 with the above routine. **Note that the name on the LOAD statement must match the label specified on the $MODULE macro statement, and the name on the ROUTINE= parameter on the EXITnnn statement must be the same name as on the $ENTRY macro.**

```
LOAD=HASPUEX
EXIT1 ROUTINE=UEXIT1,ENABLE,TRACE=NO
```

Multiple Exit Routines In A Single Module

When developing and testing user exits, it is probably easier to keep each exit routine in its own source and load module. In this manner, the routines can be assembled, loaded, and tested independently. If there are many routines, you may want to eventually combine them into a single source and load module for easier maintenance procedures.

Figure 2-6 shows three exit routines in a single module with a general structure that you may want to follow.

```
XITS         TITLE 'SAMPLE JES2 USER EXITS - PREAMBLE'
*****************************************************************
*                                                               *
*             COMMENT BLOCK FOR MODULE GOES HERE .....          *
*                                                               *
*****************************************************************
             COPY   $HASPGBL      COPY HASP GLOBALS
HASPUEX      $MODULE  RPL,        REQ'D BY $BUFFER                C
             $BUFFER,                                             C
             $CAT,                                                C
             $DCT,                                                C
             $HASPEQU,          REQUIRED FOR REG CONVENTIONS      C
             $HCT,              REQ'D BY $SAVE,$RETURN,ETC.       C
             $JCT,                                                C
             $JOE,              REQ'D TO GET SYSOUT CLASS         C
             $JQE,                                                C
             $MIT,              REQ'D BY HCT                      C
             $PCE,              REQ'D BY HCT                      C
             $PDDB,             REQ'D BY $PPPWORK                 C
             $PPPWORK,          REQ'D TO FIND JOE                 C
             $RDRWORK
*****************************************************************
*                                                               *
*             ADDITIONAL MAPPING MACROS GO HERE                 *
*                                                               *
*****************************************************************
```

Figure 2-6 (Part 1 of 2). Example of Providing Multiple Exits within a Single Load Module

```
      TITLE 'SAMPLE SEPARATOR PAGE EXIT - ROUTINE 1'
*****************************************************************
*                                                               *
*            COMMENT BLOCK FOR EXIT 1 GOES HERE                 *
*                                                               *
*****************************************************************
XIT1RTN1     $ENTRY BASE=R12       EXIT ROUTINE ENTRY POINT
             $SAVE
             LR    R12,R15         LOAD BASE REGISTER
*****************************************************************
*                                                               *
*        USER EXIT CODE FOR EXIT 1 ROUTINE 1 GOES HERE          *
*                                                               *
*****************************************************************
             LA    R15,8             SET RETURN CODE
RETURN1      $RETURN RC=(R15)        RETURN TO HASPPRPU
             TITLE 'SAMPLE SEPARATOR PAGE EXIT - ROUTINE 2'
XIT1RTN2     $ENTRY BASE=R12         EXIT ROUTINE ENTRY POINT
             $SAVE
             LR    R12,R15           LOAD BASE REGISTER
*****************************************************************
*                                                               *
*        USER EXIT CODE FOR EXIT 1 ROUTINE 2 GOES HERE          *
*                                                               *
*****************************************************************
             LA    R15,8             SET RETURN CODE
RETURN2      $RETURN RC=(R15)        RETURN TO HASPPRPU
             LTORG
             TITLE 'JOB CARD SCAN EXIT'
*****************************************************************
*                                                               *
*         COMMENT BLOCK FOR EXIT 2 ROUTINE 1 GOES HERE          *
*                                                               *
*****************************************************************
XIT2RTN1     $ENTRY BASE=R12         EXIT ROUTINE ENTRY POINT
             $SAVE
             LR     R12,R15          LOAD BASE REGISTER
*****************************************************************
*                                                               *
*         USER EXIT CODE FOR EXIT 2 ROUTINE 1 GOES HERE         *
*                                                               *
*****************************************************************
            LA    R15,8              SET RETURN CODE
            $RETURN RC=(R15)          RETURN TO HASPRDR
            LTORG
            $MODEND
            END
```

Figure 2-6 (Part 2 of 2). Example of Providing Multiple Exits within a Single Load Module

The following JES2 initialization statements can be used to load and associate exit points 1 and 2 with the above routines.

```
LOAD=HASPUEX
EXIT1 ROUTINE=(XIT1RTN1,XIT1RTN2),ENABLE,TRACE=NO
EXIT2 ROUTINE=XIT2RTN1,ENABLE,TRACE=NO
```

Testing Your Exit Routine

To test your exit routine you need to integrate your exit routine in the system, ensure that it gets control and executes, and verify that the functions it is intended to perform are performed. Verifying that the exit routine performed its function is exit routine-dependent and unique for each exit routine; the following discussion of testing your exit routine only focuses on the following points:

- Integrating the exit routine in the system
- Passing control to the exit routine

Also, you should test and debug your exit routine by running it on a secondary JES2 first. In this way, any errors that occur do not directly affect your main JES2 production system. Once the errors in the exit routine are fixed and tested, you can then integrate it into the production JES2 system. Note that the following restrictions apply to JES2 functions when using a secondary JES2:

- Started tasks (STCs) can be directed to either a primary or secondary JES. However, following an IPL, started tasks do not complete start processing until the primary subsystem has been started and completed initialization.
- Time-sharing users (TSUs) may only interface with the primary JES2.
- The MVS I/O attention table can only be associated with the primary JES. Therefore, secondary JESs cannot receive the "unsolicited interrupt" required to support pause-mode for print and punch devices and "hot readers" (that is, readers started via the physical start button without the $S RDRn JES2 command).
- The MVS log console (SYSLOG) can only be associated with the primary JES.
- Secondary subsystems are started individually rather than automatically during IPL by a start command in the master scheduler JCL (MSTJCL) as is the primary subsystem.

Integrating the Exit in the System

Packaging the Exit

Exit routines need to be packaged into load modules before they can be loaded into the system and tested. There are two ways to package your exit routines.

Modules that contain exit routines which execute in the JES2 main task or subtask environment can be linkedited into the standard HASJES20 load module, which is loaded into the private area of the JES2 address space. Modules that contain exits in the user or functional subsystem environment can be linkedited into the standard HASPSSSM load module which must reside in SYS1.LPALIB; these exits must be loaded into common storage. **Do not linkedit multiple exit points with different environments into the same load module.** When you package your exit routines in this manner, it is required that you make use of a collection

of weak external names for the module names. These names should be the same as the label used on the $MODULE macro of your exit routine.

For HASPSSSM the "weak external names" are as follows: HASPXS00, HASPXS01, ..., HASPSX15.

For HASJES20 the "weak external names" are as follows: HASPXJ00, HASPXJ01, ..., HASPXJ31.

Alternatively, a source module containing exit routines can be assembled and link edited separately. However, you can only code one MIT and MITETBL within the load module. That is, only one $MODULE and $MODEND statement pair should appear in each exit source module, and each source module must be linkedited into its own load module. The resulting load module can then be loaded dynamically during initialization, through use of the LOAD initialization statement. For this form of packaging you do not need to use the "weak external name" for the module names of your exit routines. *NOTE:* If your exit routine is a separate load module and executes in the USER or FSS environment, you must ensure that it resides in LPA.

You may choose to use one of these packaging techniques exclusively, or you may choose to use both methods in combination, assembling and link editing some routines into the standard JES2 load modules and assembling and link editing others separately and then loading them at initialization. *Creating separate load modules for your exit routines is recommended.* JES2 never makes unconditional direct references to external addresses or entry points in installation-written code. The association between exit routines and JES2 source code is resolved during initialization.

Figure 2-7 illustrates a separately linkedited load module for an exit routine and the MIT and MITETBL structure associated with it. JES2 initialization uses this load module and the information in the MIT and MITETBL to initialize the exit routine in the system. The next topic describes this initialization process.

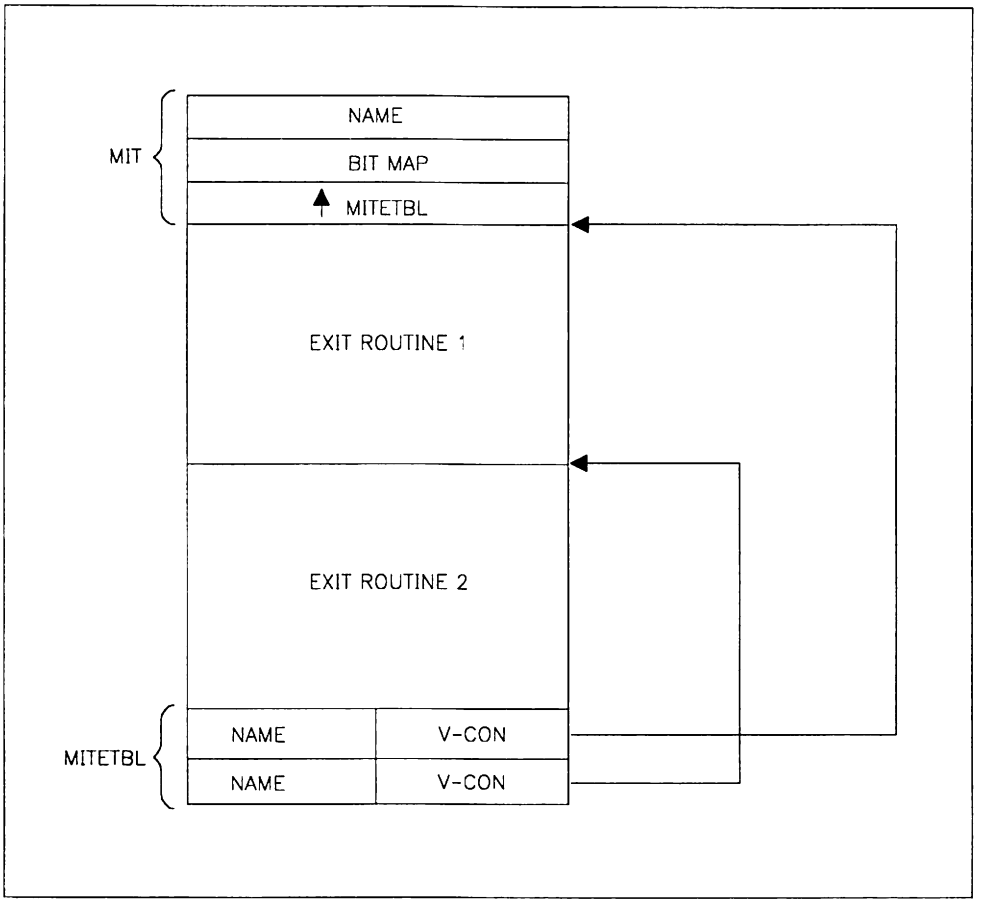


Figure 2-7. Exit Routines Load Module

Initializing the Exit in the System

Initializing an exit and its exit routines involve the use of the following two JES2 initialization statements:

- LOAD

  Use the *LOAD initialization statement* to load the modules containing your exit routines. The only operand of the LOAD initialization statement specifies the name of the module to be loaded as defined on the NAME control statement for the linkage editor. The module must be named according to MVS naming conventions. Be certain that the name you give to the load module (LOAD=) is the same as the label you coded on the $MODULE statement. Exit routines to be called from the user environment must be loaded into LPA (PLPA, MLPA, or FLPA) during IPL. Exit routines to be called from the JES2 main task and subtask environments should be loaded in the private area of the JES2 address space.

- EXITnnn

  Use the *EXITnnn initialization statement* to associate one or more exit routines with an exit.

  Replace *nnn*, the exit number, with the corresponding exit identification number specified on the $EXIT macro or macros that define the exit point or points that establish the exit. EXITnnn can then be followed by from 1 to 255 exit routine names, as specified for the $ENTRY macro or macros that identify the corresponding exit routines. The JES2 exit effector calls multiple exit routines in the sequence of their specification on the EXITnnn statement. If you specify more than one EXITnnn statement with the same identification number, JES2 honors the last statement it encounters during initialization.

  Figure 2-9 illustrates the completed control block structure after two separate load modules (the first containing three routines and the second containing two routines) have been initialized in the system. Note also that one of the routines (XITCOMON) is shared by both Exit1 and Exit2.

NOTE: The LOAD and EXITnnn initialization statements are not positional and do not have to be specified in any required order. Figure 2-8 illustrates the results of this initialization process.

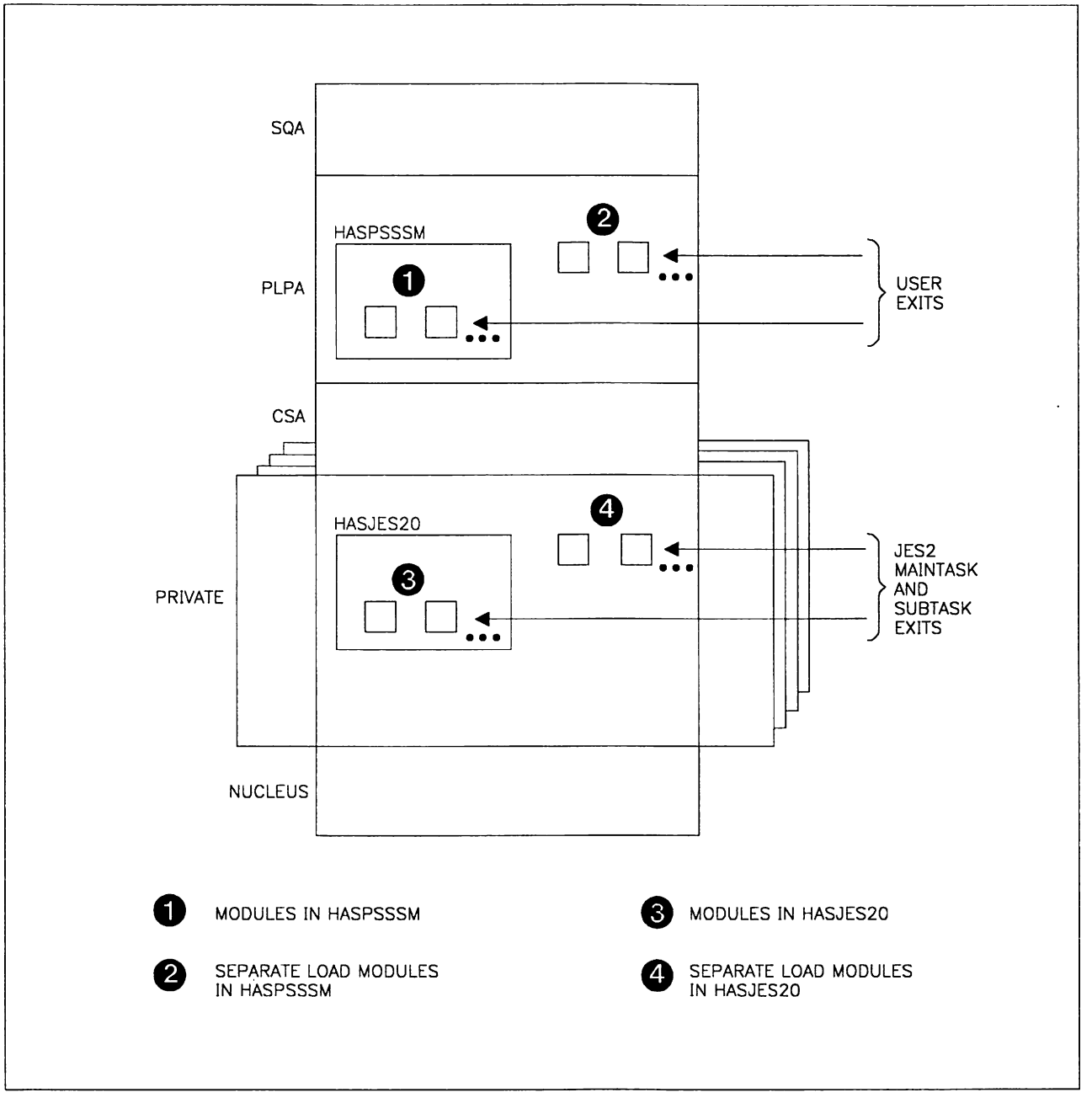


Figure 2-8. Exit Placement

During initialization, JES2 uses the EXITnnn statements to build the exit information table (XIT) and the exit routine table (XRT), both located in CSA. The XIT contains an entry for each exit. It also contains an index into the XRT, which contains an entry for each exit routine name and address.

During initialization statement processing, JES2 puts all the information available from the EXITnnn statements into the XIT and XRT. It also resolves addresses for XRT entries using the MITs of any modules that are already loaded. For each LOAD statement encountered and processed, routine addresses in the dynamically loaded module's MIT are propagated to the XRT for routine names already present. Installation-defined exit points in the module are propagated to the XIT. JES2 builds the load module table (LMT) that contains the names of all load modules and their MIT addresses. At the end of initialization statement processing, any routine names that remain outstanding in the XRT, without resolved addresses, are brought to the operator's attention. When this process is complete, the exit effectors can use the XIT and the XRT to provide linkage from JES2 to exit routines. Note that, because of the manner in which initialization statements are processed, there is no order dependency between EXITnnn statements and LOAD parameters.

Figure 2-9 illustrates the completed control block structure after two separate load modules (the first containing three routines and the second containing two routines) have been initialized in the system. Note also that one of the routines (XITCOMON) is shared by both Exit1 and Exit2.

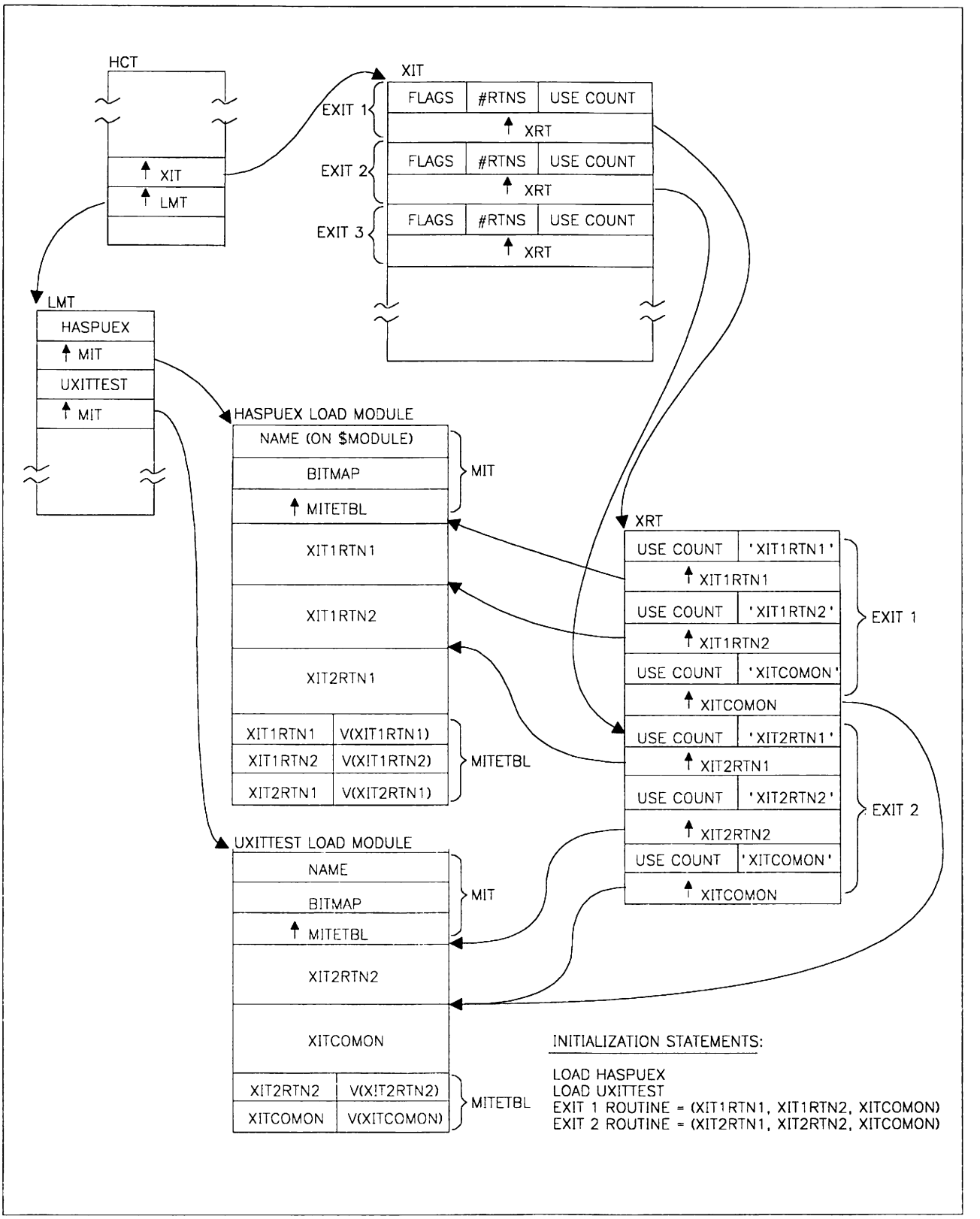


Figure 2-9. Load Module Initialization

Passing Control to Exit Routines

Every exit has a status of *enabled* or *disabled*. If an exit is enabled, JES2 calls its associated exit routine(s) whenever one of the exit's exit points is encountered in processing JES2 code. (Note: The TYPE=TEST form of the $EXIT macro is an exception; a TEST-type exit point occurs prior to a TYPE=ENTER exit point to allow JES2 to determine whether the exit is implemented and enabled. If the exit is not both implemented and enabled, JES2 saves processing time by bypassing the call to the exit effector when it encounters the ENTER-type exit point.) When an exit is disabled, its exit points are transparent in the course of JES2 processing and JES2 does not call the exit's associated exit routine(s).

An exit's status is first set at initialization. You can specify either ENABLE or DISABLE on the EXITnnn initialization statement. If you leave the status of the exit unspecified, ENABLE is the default.

An exit's status can then be dynamically controlled by the operator, using the $T EXITnnn command. Again, the operator has the option of identifying any exit by number, including a string of exits, a range of exits, or all exits, and specifying either ENABLE or DISABLE. The operator can display an exit's status by identifying the exit by number on the $D EXITnnn command.

When you suspect that an exit routine associated with a particular exit is causing an error, a simple way of isolating the problem is to disable the exit, via an operator command ($T EXITnnn), to determine if the error still occurs when the exit routine is not allowed to execute. You can also enable tracing as a debugging aid.

An exit can also be dynamically controlled on a job-related basis, using the exit facility.

Job-Related Exits

Certain exits are identified as *job-related exits*. For each individual job, exit routine action can determine whether or not a job-related exit will be taken. For these exits, the JOBMASK parameter is specified on the $EXIT macro or macros defining their exit point or points. JOBMASK is specified with the address of the *job exit mask*, a 256-bit mask in the job control table (JCT), of which each bit, except for bit 0, corresponds to an exit identification number; bit 1 corresponds to Exit 1, bit 2 to Exit 2, and so on. Initially, when the JCT is created, all the bits in the job exit mask are set to one.

For a job-related exit, the status of its corresponding bit in the job-exit mask becomes an additional factor in determining its exit status. If an exit has been enabled in the standard way, by either the EXITnnn initialization statement or the $T EXITnnn command, and its corresponding bit in the job exit mask is set to one, the exit has a status of enabled and the exit effector calls its associated exit routine(s). If, however, the exit has been enabled in the standard way but its corresponding bit in the job exit mask is set to zero, the exit has a status of disabled and the exit effector does not call its associated exit routine(s) for that particular job. If the exit has been disabled in the standard way, the status of its corresponding bit in the job exit mask is not taken into account; the exit remains disabled. Note that if JOBMASK is not specified on the $EXIT macro, or if the

JCT is not in storage, the job exit mask can have no effect on the status of an exit.

Bits in the job exit mask can be manipulated by an exit routine on a job-by-job basis. The recommended IBM-defined exit for setting the job exit mask is Exit 2. Exit 2 is, in most cases, the first exit to be taken for a job, and provides access to most of the job's attributes specified in its JCL and placed in its JCT. For more information, see the description of Exit 2 in "The IBM-Defined Exits" reference section in Chapter 3.

For each exit description in "The IBM-Defined Exits," the JOB EXIT MASK category lists the exit as either job-related or not job-related. Note that Exits 11 and 12 present special cases.

Tracing Status

You can also control the status of exit invocation tracing.

Initially, for the tracing to occur automatically, two conditions are necessary:

1. The trace ID for exit tracing (ID 13) must be enabled.
2. The TRACE= operand of the EXITnnn initialization statement must be specified as, or allowed to default to, TRACE=YES.

If one of these conditions is absent, tracing does not occur.

The status of exit tracing can then be dynamically controlled by the operator, using the $T EXITnnn command. The operator has the option of identifying any exit by number, including a string of exits, a range of exits, or all exits, and specifying either TRACE=YES (or TRACE=Y) or TRACE=NO (or TRACE=N). The operator can display the status of exit tracing by identifying the exit by number on the $D EXITnnn command.

The status of exit tracing cannot be controlled on a job-related basis.

Establishing Installation-Defined Exits

JES2 can contain up to 256 exits. IBM has defined some of these. If none of the IBM-defined exits is suited to a particular modification you would like to make, you can consider installing an optional installation-defined exit.

In general, establishing your own exit is much more difficult than writing an exit routine for an existing IBM-defined exit; it requires a thorough knowledge of the area of processing in which you would like your exit to occur. You should attempt to place a installation-defined exit in a stable area of processing; the risk of error increases with the complexity of the JES2 code in which you place the exit. If possible, you should use your exit in replacing a JES2 function that is already isolated. As an example, IBM-defined Exit 3 allows you to provide an exit routine to completely replace the standard HASPRSCN accounting field scan routine.

You must consider whether the exit will require a single exit point or more than one. You can determine this based on the requirements of your intended modification and on the structure of the IBM code in the area of processing that you intend to modify. You must also consider whether or not the function you wish to modify is contained within a single JES2 execution environment. If it occurs in a second environment, you may have to install a second exit as well.

Once you have determined the exact point of processing at which an exit point must occur, use the $EXIT macro to define it.

First, you should specify the positional ID parameter with the exit's identification number. It is recommended that you begin numbering installation-defined exits with 255 and work down. (If additional IBM-defined exits are added later, your exit numbers will not conflict with the new IBM-defined exit numbers.)

You must define the exit's environment to JES2 using the ENVIRON= operand or let the environment default to what was specified as ENVIRON= on the module's $MODULE macro. This is specified as either JES2, SUBTASK, USER, or FSS. If you use more than one exit point to establish an exit, the environment specification must be the same on each $EXIT macro. The same exit cannot be established in more than one execution environment.

The exit effector for the subtask and user environments assumes MVS linkage conventions when register 13 contains the address of an available 18-word save area. However, because of MVS linkage conventions you may need to add code allowing the $EXIT macro to acquire and dispose of a standard 18-word save area for use by the subtask and user environment exit effector.

If the exit is to be job-related, specify the address of the job exit mask for the JOBMASK= operand. Note that if the JCT is not in storage you will have to point to a copy of the job exit mask.

Specify the SAVAREA= operand only if the exit occurs in a JES2 subtask, in the user (HASPSSSM) environment, or functional subsystem (HASPFSSM) environment. If you specify the SAVAREA= operand and ENVIRON=JES2 as well, an error message is generated. For subtask and user exits, SAVAREA= can be specified with the address of a save area that the exit effector will pass to an associated exit routine in register 13. If you don't specify SAVAREA= for a subtask or user exit, the exit effector will acquire the necessary area via a GETMAIN macro whenever the exit is invoked.

Note: There is a distinction between the two save areas, the save area pointed to by register 13 and that pointed to by the SAVAREA= keyword. Register 13, at the $EXIT point, contains the address of a save area into which the exit effector stores the current registers. SAVAREA= provides a save area (typically provided via an MVS GETMAIN) into which the exit routine stores the exit effector's registers.

Use the TYPE= operand to specify the mode of $EXIT macro operation. To avoid special processing overhead, you can define a TYPE=TEST $EXIT macro at some location shortly prior to a TYPE=ENTER $EXIT macro in JES2 code.

A TEST-type $EXIT macro tests the status of the exit and sets a condition code (not a return code):

cc=0 No exit routines are to be called
cc=1 Call exit routines, without tracing
cc=2 Call exit routines, with tracing

When JES2 encounters the TYPE=ENTER $EXIT macro, it does not have to retest the exit's status; it simply checks the condition code and either bypasses the exit point or calls the exit effector, with or without tracing. Note that a TYPE=TEST $EXIT macro and a TYPE=ENTER $EXIT macro must always be used together. If you omit the TYPE= parameter, the resulting exit point causes JES2 to both determine the status of the exit and then, depending on the status, either to bypass the exit point or to call the exit effector.

Use the AUTOTR= operand to specify that automatic exit effector tracing should (AUTOTR=YES) or should not (AUTOTR=NO) occur. If AUTOTR=NO is specified, the exit effectors set the condition code on return from the exit to indicate the tracing requirement:

0 No tracing is required
1 Tracing is required

For more information on exit effector tracing, see "Tracing" in "Writing an Exit Routine" and "Tracing Status" in "Controlling Exit Status" earlier in this chapter.

Along with inserting the $EXIT macro in JES2 source code, you may have to modify the code prior to the exit point to pass parameters and pointers to the exit routines, and you may have to modify the code following the exit point to receive exit-generated parameters and to receive any return code greater than 4. For more information, see "Linkage Conventions," "Received Parameters," and "Return Codes" in "Writing an Exit Routine" earlier in this chapter.

Note: When using the $EXIT macro, you may need to include additional control block DSECT mappings in that module. If, for example, the module you are modifying did not previously require the mapping provided by the $EXITPL and $XIT macros, but these are required to map the exit parameter list and exit information table (XIT), you must add these to the $MODULE macro coded at the beginning of the module.

Defining JES2 Tables

Modifying JES2 by Defining User Tables

You should attempt to isolate as many of your installation-specific modifications as possible within user modules. Doing so can speed your migration should you install a new level of JES2, ease maintenance without having to rework your own code, and more easily isolate problems. JES2 provides a method of tailoring many areas of its processing through the use of tables and table pairs.

JES2 provides user fields in a number of the commonly modified control blocks. For example, the UCT (user communication table) is effectively an extension of the HCT (HASP communication table). Other extension points are the tables that define processor control elements (PCEs), daughter task elements (DTEs), trace ID tables (TIDTABs), and the scan facility. These "extensions" should not reside within JES2 inline code, but rather in a user module or dynamic storage area accessed through user exits. By taking advantage of user exits and JES2 macros, you can build fields to point to your own user tables to override the default JES2 tables. This eliminates the need to directly modify JES2 control blocks or copy JES2 code into user modules.

As noted above, a number of JES2 processing areas use tables; that is, a JES2 table contains the elements of a particular area of processing. You can tailor certain JES2 functions by extending these tables. JES2 provides two tables, **a table pair**; one table (the **JES2 table**) provides the default processing specifications. If you do not extend this table, JES2 will remain unmodified. However, if you choose to fill in the **user table** of the table pair you will be adding new function or overriding those JES2 specifications with what you have provided in the user table. This is so because JES2 searches the user table before the JES2 table whether or not JES2 has defined a particular specification. For example, JES2 has specified that the minimum length of the RANGE parameter on the PRINTERnn initialization statement must be 5 (that is, all five characters must be coded). You can change this requirement by overriding the JES2 table by coding your own RANGE table entry. If you prefer the minimum number of characters to be 3, you could then code either RAN, RANG, or RANGE when specifying this parameter.

The macros required to build these tables are documented in Chapter 4. These macros call the required service routines to modify JES2 processing. Because JES2 provides a JES2 table (the table you would be overriding), you are automatically provided an example of how such a table is coded. (Refer to "General Table Coding Conventions," below, for further coding information.)

Table Linkage

To implement user extensions, you must create a user control table (UCT). To do this, you must either:

1. Link-edit the module containing the UCT with HASJES20 by using the reserved (weak external) names provided by JES2

   or

2. Use Exit 0 to place the address of the UCT into the HCT

Figure 2-10 depicts the two linkage procedures.

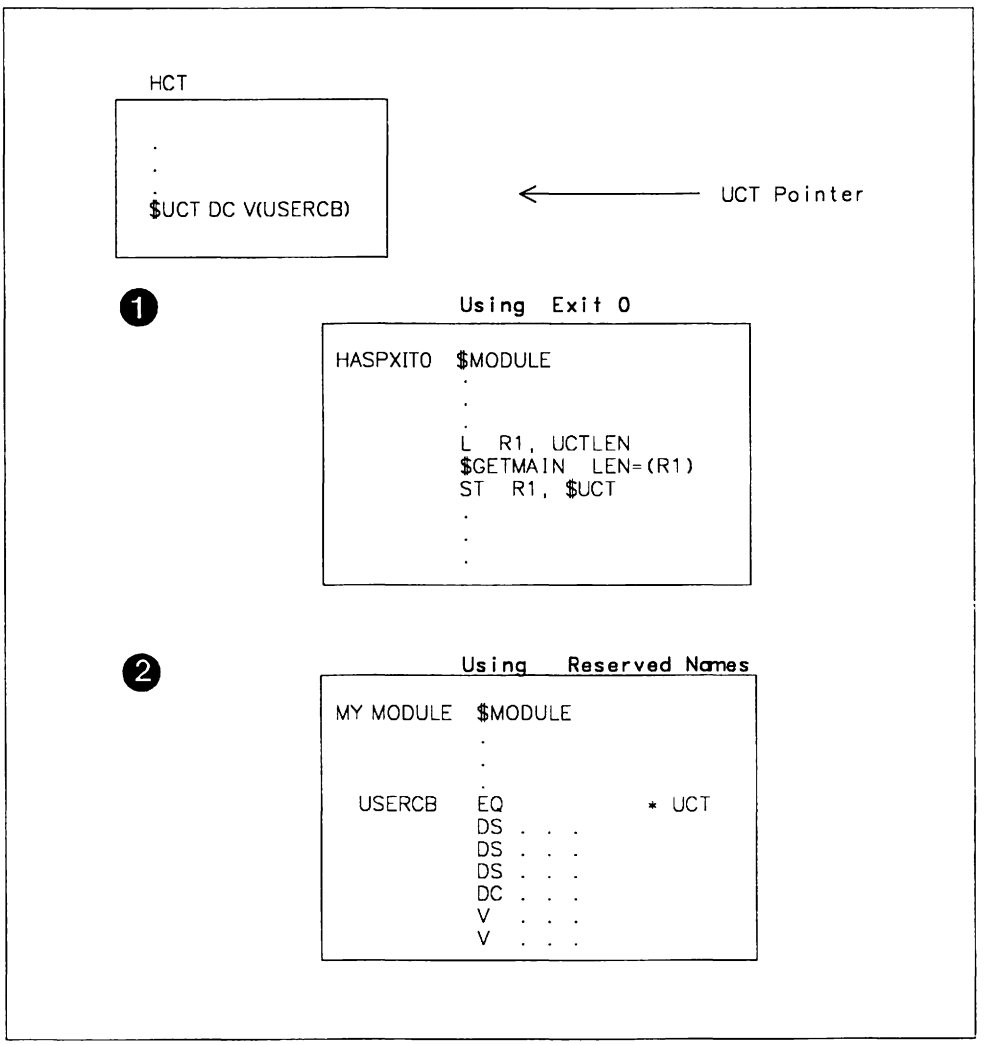


Figure 2-10. Linking User Tables to HASP Tables to Extend/Modify JES2 Function

As noted above you can provide linkage to the MCT through two methods. This is required in order that the address you provide can be resolved. Either is adequate and the choice is yours to make. The two methods and some advantages of each are:

- Use JES2 initialization Exit 0 to store your table addresses in the MCT.

  This method requires more code than the method below, but results in a later binding, that is, at JES2 initialization rather than link-edit time. Furthermore, Exit 0 is refreshable over a JES2 hot start without changing the HASJES20 load module.

- Link-edit your tables to JES2 using weak external references.

  This technique requires less code than the first method, but it requires that you modify the JCLIN logic in the system modification program (SMP) by adding an INCLUDE statement for your user module. Following such a change to either JES2 or a user module, you **must** again link-edit the entire load module.

Master Control Table

The **master control table** (MCT) is a list of all JES2 table pairs. The MCT is pointed to by the $MCT field in the HCT and points to the JES2-defined and user-defined tables. Refer to Figure 2-11 below. This example shows the MCT with its pointers (using weak external VCONs) to the PCE, DTE, OPT, MPS, and SCAN tables.

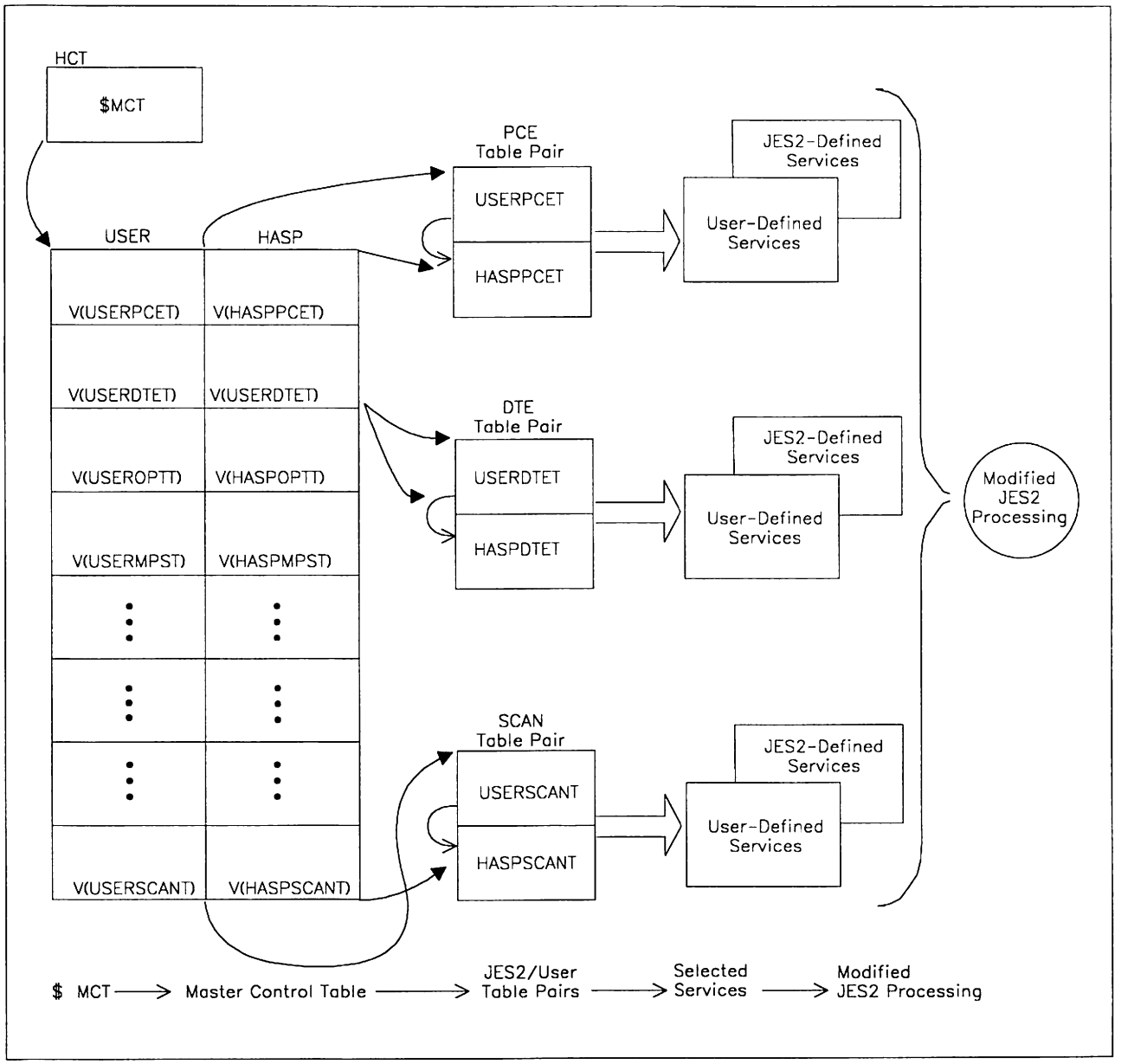


Figure 2-11. Master Control Table and Table Pairs

General Table Coding Conventions

All tables that you may build already have a JES2 counterpart available as an example in the JES2 code. The table type is defined by the corresponding macro, that is, use $PCETAB to build a PCE table, $SCANTAB to build a scan table. Each JES2 table begins with a TABLE=HASP specification. To code a user table, begin with a $xxxTAB TABLE=USER specification. Subsequent lines are added to specify the statement, command, or processor that you are defining, such as the PCE name and the module containing the processor's code for a PCE table entry or the valid parameter length and range in a SCAN table entry. Each table is then ended by coding a $xxxTAB TABLE=END statement.

Overview of Modifiable Tables

Each entry in the MCT points to a pair of tables (the JES2 table and the user table). A brief description of the tables that may be accessed and overridden follow. Each is discussed in greater detail later in this section.

$SCAN Tables

Scan tables are used to process initialization statements, their parameters and subparameters, and commands. Scanning can be recursive until all levels of an initialization statement or command have been scanned for syntax requirements, valid range, etc. The $SCAN facility is detailed below.

PCE Tables

PCEs define the names, descriptions, IDs, and other characteristics of JES2 processors. User processors (PCEs) can be accessed by using the $PCETAB macro and linking it to the USERPCET entry in the MCT. (Refer to "Table Linkage" below for a description of the recommended linkage procedures.) These tables reside in HASPTABS and are accessed by the $PCETAB macro by using the TABLE=USER specification. This facility is further described below.

TID Tables

Trace IDs can be mapped and generated by use of the $TIDTAB macro. These tables are defined for formatting entries in the JES2 and user trace tables.

OPT Tables

The JES2 initialization OPT (options table) defines the JES2 options, COLD | WARM, FORMAT | NOFMT, REQ | NOREQ, CONSOLE, LIST | NOLIST, LOG | NOLOG, PRMCKPT | ALTCKPT, and $P JES2. You can define others and disable those previously defined by JES2 by modifying this table. (Refer to *JES2 Initialization and Tuning* for a description and use of these options.) These tables reside in HASPSTAB and are pointed to and accessed by use of the $SCAN facility. Use $SCANTAB and the related $SCAN macros. This facility is described in detail below.

MPS Tables

The main parameter statements (JES2 initialization statements and their parameters) are defined in the MPS table. (Refer to *JES2 Initialization and Tuning* for a description and use of these initialization statements.) These tables reside in HASPSTAB and are pointed to and accessed by use of the $SCAN facility. Use $SCANTAB and the related $SCAN macros. This facility is described in detail below.

DTE Tables

Daughter task element (DTE) tables can be mapped and generated by using the $DTETAB macro and by using the $DTEDYN macro to facilitate subtask attaches and detaches for the JES2 main task. These tables reside in HASPTABS.

WSTAB Tables

The work selection criteria provided by JES2 to allow printers, punches, and the offload transmitters and receivers to selectively choose work are present in the work selection tables (WSTABs). If you find the JES2-defined work selection criteria inadequate to meet your needs you can define more criteria through use of the work selection facility. These tables reside in HASPTABS and are pointed to and accessed by using $WSTAB macro.

JES2 $SCAN Facility

JES2 provides a service facility for scanning parameter statement input (initialization statement and operator commands) called $SCAN. It is a general facility that defines a general grammar for the input statements to be processed, allows for definition of the allowed input via tables, and provides for special processing via exit routines called during the scan.

$SCAN is basically designed to perform most of the scanning required for processing the JES2 initialization statements, with the remaining processing for those statements being done by the exits from $SCAN, and to allow the use of multiple tables to define the allowed parameter input. $SCAN can scan various input structures, including those that require recursive calls to $SCAN itself. At each level of recursion, $SCAN can use two tables of specifications that define the allowed input at that level.

JES2 has implemented the scanning of its initialization options and its initialization statements using $SCAN and a series of these **table pairs**. A JES2-defined table has been built as the second table of each pair, and an installation table can be defined as the first table to add to or modify the specifications in the JES2 table. $SCAN can be useful, as well, in implementing other types of statements within routines called from the $EXIT facility, such as installation-defined operator commands or JECL statements.

$SCAN Macros

Five macros are provided to aid your use of the $SCAN facility. These macros, $SCAN, $SCANB, $SCANCOM, $SCAND, and $SCANTAB are fully documented in Chapter 4, "JES2 Programmer Macros." It is important that you review that material and understand the interrelationships of these macros before attempting to implement any use of the $SCAN facility.

A brief overview of these five macros and their individual functions follows:

$SCAN: This macro is used to scan initialization and control statements and some JES2 commands and place the specified keyword values in associated control blocks. Also, $SCAN can be used to display keyword values. A pair of scan tables, which are built by the $SCANTAB macro, the parameter or command input, and a display area and routine (these are used for diagnostic information in SET and DISPLAY calls) are passed to $SCAN as parameters.

$SCANTAB: To use the $SCAN facility, you must first create scan tables. The $SCANTAB macro is used to create tables to define permissible input and syntax for installation-defined statements, and options. Examples of scan table definition are presented in the example code presented under "Examples of $SCAN Tables" below.

$SCANB: Use $SCANB to create a backup copy of the control block fields in a storage area prior to altering them during $SCAN facility execution. This backup copy protects control block fields that $SCAN will change, and if an error occurs during the scan, all backup fields are restored to their original values. The interface between $SCAN and these exit routines is through the scan work area (SCWA), the address of which is passed in register 1 and which is mapped by the $SCANWA macro. ($SCANWA is not available for your use but accessed during the expansion of the $SCAN macro.) $SCANB is only used by installations from within a pre-scan or post-scan exit routine after it receives control from $SCAN.

$SCAND: This macro allows you to add text to a $SCAN display line during either pre-scan or post-scan exit routine processing when these exits are called by the $SCAN. $SCAND is used when $SCAN=DISPLAY or $SCAN=SETDISP is specified and is only valid within an associated $SCAN exit routine called by $SCAN or from within $SCAN itself.

$SCANCOM: The $SCANCOM macro allows JES2 to scan a text string and locate the first non-blank, non-comment character. The address of this character is returned to the calling module (typically $SCAN) to allow that caller to ignore the comment text. This scanning is provided for both initialization statements and commands. Comments on both initialization statements and commands must be bound by the beginning (/*) and ending (*/) delimiters. If a delimiter is missing, JES2 returns a note to that effect and passes a return code noting the invalid comment.

$SCAN-Related Control Blocks

There are several control blocks related to $SCAN. First, the facility recognizes a set of "primitive" control blocks specified in the scan table entries. They are the HCT, the current PCE, DCTs, and the user control table (UCT). The UCT is not generated or specifically used by JES2, but rather is an optional user control block pointed to by the $UCT field of the HCT. Installations requiring a central control block for use in exit routines or user modifications should generate a UCT and use it as their central main task control block rather than adding new fields to the HCT.

Additionally, scan table entries can indicate the control block from the previous "level" of scanning or a temporary control block should be used. A subsequent search for the actual required control block via control block chains and subscript indexing can also be indicated by the table entries.

Another important control block for the JES2 uses of $SCAN is the JES2 master control table (MCT). The MCT is pointed to by field $MCT in the HCT and it contains the addresses of many JES2 statically-defined tables and related routines. Importantly, the MCT contains the doublewords containing addresses of the scan table pairs. The MCT is assembled into module HASPTABS in load module HASJES20.

The scan table entries themselves form control blocks which are mapped by the $SCANTAB macro. Also, during a scan, a work area is used by $SCAN and passed to pre-scan and post-scan exit routines. The scan work areas are allocated via $GETWORK and mapped by the $SCANWA macro.

Implementing $SCAN Tables

The $SCANTAB macro should be used to generate a scan table. Your installation can define, for example, a table that describes the keywords and input allowed on a JES2 control statement and use $SCAN and that table from a JES2 HASPRDR Exit 4 to implement your own JES2 job control statements.

As mentioned, the JES2 initialization options, initialization parameter statements, and some operator commands are now implemented using $SCAN. (Refer to *JES2 Commands* for a list of these commands.) Scan tables in the HASPSTAB module make up the JES2 half of the table pairs that define those options and parameters. Your installation can define its own scan tables to add to or replace any or all of the JES2 scan table entries.

Each of the pairs of table addresses used in the implementation of the initialization statements is defined in the MCT in HASPTABS. Your installation can locate the MCT while running in a JES2 initialization exit (for example, Exit 0) and store the addresses of its tables in the MCTxxxTU fields of the MCT, or you can point to your installation-defined UCT that contains a pair of table addresses.

Optionally, the linkage editor can be used to define these table addresses to JES2 by linkediting them (and possibly the UCT) into HASJES20. This is possible because the MCTxxxTU and $UCT fields are defined as the weak external

symbol names. The installation must name its scan tables with those specific names (listed in Figure 2-12)
in order to use this method of providing user scan tables for the initialization statements.

The installation tables can provide entries that define new initialization options or statements, new parameters on JES2-defined statements, or new commands. Since the installation table is searched first, JES2-defined entries can be functionally replaced as well. The following scan tables are involved with the JES2 initialization statements. In each case,the 'xxx' described indicates the MCT labels for the table address pair (MCTxxxTP), the installation table address (MCTxxxTU), and the JES2 table address (MCTxxxTH), as well as the installation table WXTRN (USERxxxT) used in the MCTxxxTU fields (the UCT WXTRN is USERCT).

OPT Initialization options
MPS Main Initialization parameter statements

Master Control Table Name	Parameters on Initialization Statement
APL	APPL
BAD	BADTRACK
BUF	BUFDEF
CAT	JOBCLASS
CKT	CKPTDEF
CND	CONDEF
COM	COMPACT
CON	CONNECT
DES	DESTID
ELC	ESTLNCT
EBY	ESTBYTE
EPG	ESTPAGE
EPN	ESTPUN
ETM	ESTIME
FSS	FSSDEF
INR	INTRDR
JOB	JOBDEF
JPY	JOBPRTY
LNE	LINEnnnn
LOG	LOGONn
MAS	MASDEF
NET	NETACCT
NJE	NJEDEF
NOD	NODEnnnn
OFF	OFFn.DV
OFL	OFFLOADn
OJM	OFFn.JR*
OJR	OFFn.JR
OJT	OFFn.JT
OPY	OUTPRTY
OSM	OFFn.SR*
OSR	OFFn.SR
OST	OFFn.ST
OUT	OUTDEF
PAR	PITDEF
PCD	PCEDEF
PIT	INITnnnn
PRT	PRINTERnn
PTD	PRINTDEF
PUD	PUNCHDEF
PUN	PUNCHnn
RCV	RECVOPTS
RDR	READERnn
RDV	RnnnnDVx
RMT	RMTnnnn
RPR	RnnnnPRx
RPU	RnnnnPUx
RRD	RnnnnRDx
SCT	OUTCLASS
SMF	SMFDEF
SPD	SPOOLDEF
STC	STCCLASS
TPD	TPDEF
TRC	TRACE
TSU	TSUCLASS
XIT	EXITnnn

* Indicates that this MCT name is used only to scan the subparameters contained on the MOD= parameter of this statement.

Figure 2-12. JES2 Reserved Master Control Table Names

Examples of $SCAN Tables

The three examples in Figure 2-13 show

(A) how to add a new simple initialization statement

(B) how to replace a parameter statement specification on the PRINTERnn statement

(C) how to define a new initialization statement to provide an installation-specific function with a single parameter.

All the specifications that can be made in scan table entries are described with the $SCANTAB macro in Chapter 4.

Most of the capabilities of the $SCANTAB specifications and the $SCAN facility are illustrated by reviewing the JES2 tables in HASPSTAB that define the initialization statements.

The addresses of these tables can be specified to JES2 by including the tables in an assembly module linkedited into HASJES20 or by having an Exit 0 or Exit 19 routine locate the tables and save their addresses in the MCT.

```
(A)
*********************************************************************
*                                                                   *
*    USER TABLE FOR MAIN INITIALIZATION STATEMENTS                  *
*                                                                   *
*    'USERCAN'  - USER DEFINED STATEMENT THAT SHOULD EFFECT THE     *
*                 SAME ACTION AS JES2 'CANCEL'                      *
*                                                                   *
*    'HASPSSSM' - USER DEFINED REPLACEMENT STATEMENT FOR THE        *
*                 JES2 HASPSSSM STATEMENT  - ALLOWS THE INPUT TO    *
*                 BE ONLY 3-5 CHARACTERS LONG RATHER THAN 1-8.     *
*                                                                   *
*********************************************************************
         SPACE 1
USERMPST $SCANTAB TABLE=USER
         $SCANTAB NAME=HASPSSSM,FIELD=SSSMNAM,CB=HCT,CONV=CHARJANS, C
               RANGE=(3,5),CALLERS=($SCIRPL,$SCIRPLC,$SCDCMDS),     C
               MSGID=827
         $SCANTAB NAME=USERCAN,FIELD=CIRFLAG1,CB=PCE,CONV=FLAG,      C
               VALUE=(,CIRF1CAN,FF),CALLERS=$SCIRPLC
         $SCANTAB TABLE=END
```

Figure 2-13 (Part 1 of 3). Three Examples of $SCANTAB Tables

B

```
*****************************************************************
*                                                               *
*    USER TABLE FOR PRINTER INITIALIZATION STATEMENTS           *
*                                                               *
*    'USERCLS'  - USER DEFINED STATEMENT THAT SHOULD EFFECT THE *
*                 SAME ACTION AS JES2 'CLASS', EXCEPT ONLY      *
*                 FOR CLASSES A-Z.                              *
*                                                               *
*    'FORMS'    - USER DEFINED REPLACEMENT KEYWORD FOR THE      *
*                 JES2 FORMS PARAMETER - ALLOWS THE INPUT TO    *
*                 BE ONLY 3-5 CHARACTERS LONG RATHER THAN 1-8.  *
*                                                               *
*****************************************************************
         SPACE 1
USERPRTT $SCANTAB TABLE=USER
         $SCANTAB NAME=FORMS,CB=PARENT,DSECT=DCTDSECT,MINLEN=1,     C
               FIELD=DCTFORMS,CONV=CHARAN,RANGE=(3,5)
         $SCANTAB NAME=USERCLS,CB=PARENT,DSECT=DCTDSECT,CONV=CHARA, C
               FIELD=(DCTCLASS,PITCLLEN),RANGE=(1,PITCLLEN-1)
         $SCANTAB TABLE=END
```

Figure 2-13 (Part 2 of 3). Three Examples of $SCANTAB Tables

```
C
*****************************************************************
*                                                               *
*    USER TABLE TO DEFINE A NEW INITIALIZATION STATEMENT        *
*    TO DEFINE THE MAXIMUM NUMBER OF VALID USERS                *
*                                                               *
*    'USERDEF'  - USER DEFINED STATEMENT TO SPECIFY THE         *
*                 NUMBER OF USERS                               *
*                                                               *
*                 NUM  - PARAMETER ON THE 'USERDEF'             *
*                        STATEMENT TO DEFINE THE MAXIMUM        *
*                        NUMBER OF DEFINED USERS                *
*                                                               *
*****************************************************************
         SPACE 1
UDEFTAB  $SCANTAB  TABLE=USER
         $SCANTAB  NAME=USERDEF,CB=UCT,CONV=SUBSCAN,                C
               SCANTAB=(UCTSCANT,UCT),
               TABLE=END
USUBSCAN $SCANTAB  TABLE=USER
         $SCANTAB  NAME=NUM,CB=UCT,MINLEN=3,DSECT=UCT,FIELD=$UCTNUM, C
               CONV=NUM,RANGE=(0,9)
         $SCANTAB  TABLE=END
         SPACE 1
         UCT
         .
         .
         .
         USERCB    EQU
         UCTSCANT  DC  V(USUBSCAN)
         $UCTNUM   DS  F
         .
         .
         .
```

Figure 2-13 (Part 3 of 3). Three Examples of $SCANTAB Tables

The above examples are only examples of defining and modifying initialization statements by use of the $SCAN facility. The source module must, of course, include all standard JES2 statements, such as: $MODULE and $MODEND.

Implementing PCE Tables

Processor control element (PCE) tables are used by the $PCEDYN service to dynamically add or delete JES2 PCEs that define PCE types and their attributes. PCE tables are generated in the HASPTABS module by coding the $PCETAB macro. This macro maps and generates table entries. Use the TABLE=HASP or TABLE=USER keyword specification to specify the type of table. If you do not code a label for this line of code the label defaults to HASPPCET for HASP tables and USERPCET for a USER table. If this keyword is coded, no further keywords should be coded; and if coded, JES2 ignores them. Only use the TABLE= keyword to begin the table. The only other notation that can be made with the TABLE= keyword is the addition of NOENTRY to the TABLE=USER specification. The NOENTRY operand instructs JES2 not to generate an ENTRY statement for this label.

The succeeding lines coded for the macro specify the module, PCE address, PCE work area, etc. by using the remaining optional macro keywords. Finally, the table is ended by coding the TABLE=END specification. No further keywords or operands can be coded on this line. Figure 2-14 presents two examples of PCE tables.

```
UTABLE1   $PCETAB   TABLE=USER,NOENTRY                BEGIN USER TABLE1
          $PCETAB   NAME=PCETYPEA,MODULE=HARRYS,                        C
                WORKLEN=256
          $PCETAB   TABLE=END                       END USER TABLE1
HTABLE1   $PCETAB   TABLE=HASP                      BEGIN HASP TABLE1
          $PCETAB   NAME=PCETYPEB,MODULE=LLOYDS,
                WORKLEN=128                                             C
          $PCETAB   TABLE=END                       END HASP TABLE1
```

Figure 2-14. Two Examples of $PCETAB Tables

Relate the PCE tables to JES2 by one of the two following methods:

- locate the master control table (MCT) while running an initialization exit (Exit 0) and store the address of its table in the MCTPCETU field
- define the table with the reserved name USERPCET and linkedit it with HASPJES20

Implementing Trace Tables

JES2 uses trace tables to define both JES2-defined and user-defined trace IDs. These tables are used during JES2 initialization processing of the trace control bytes in the subsystem vector table (SVT) and by the trace log processor to provide formatting information when constructing trace table entries. The $TIDTAB tables located in HASPTABS define each identifier, its associated name and formatting subroutine. New entries are generated using the $TIDTAB macro as described below.

Use TABLE=HASP or TABLE=USER on the $TIDTAB (trace ID table) macro to define either a JES2- or user-defined trace table, respectively. This macro further defines the name for the trace output, a trace identifier, and an output formatting routine. Currently, there are 16 trace identifiers defined by IBM. User-defined trace tables should be assigned identifiers beginning with 255 and decreasing. Initial trace option values are defined on the TRACE initialization statement and modifiable by operator command. (Refer to *Initialization and Tuning* for the definition of the trace identifiers and the TRACE statements.) Specific JES2 activities are traced as defined by the $TRACE macro; this macro is also required to allocate a JES2 trace table entry. (Refer to the $TRACE macro in Chapter 4 for further information on this macro.)

Trace tables are always ended with a TABLE=END entry. When the TABLE= keyword is used (to begin or end a table), no other keywords are allowed. An example of a user-defined table follows:

```
TID255    $TIDTAB  TABLE=USER                BEGIN USERTAB255
          $TIDTAB  NAME=$USERTRC,ID=255,                          C
                FORMAT=TROUT255
          $TIDTAB  TABLE=END                 END USERTAB255
```

The above example defines a user trace table named $USERTRC with associated identifier of 255 and using the TROUT255 subroutine to format the trace output.

Relate the trace id tables to JES2 by one of the two following methods:

- locate the master control table (MCT) while running an initialization exit (Exit 0) and store the address of its table in the MCTTIDTU field
- define the table with the reserved name USERTIDT and linkedit it with HASPJES20

Implementing DTE Tables

JES2 uses daughter task element (DTE) tables to define both the JES2-defined and user-defined subtask management information. These tables are used when calling service routines in HASPDYN to attach ($DTEDYNA) and detach ($DTEDYND) subtasks. By using DTE tables, JES2 provides a centralized subtask management facility through a general subtask ATTACH/DETACH service. This facility is available for your use by creating your own DTE tables.

Use the $DTETAB macro to define general subtask characteristics. This macro allows you to create subtasks without modifying inline JES2 code. These subtasks can then be automatically attached during JES2 initialization processing.

At JES2 termination, subtasks, that were previously attached are automatically detached through a $DTEDYN macro call.

Subtask Management

The JES2 subtask management function is centralized though a general subtask ATTACH/DETACH service. This service provides MVS ATTACH/DETACH processing, centralized subtask management, error checking, and messages to assist code debugging. You can access and use these subtask management services without the need of modifying JES2 code in support of user subtasks.

Daughter Task Element (DTE) Structure

The daughter task element structure is comprised of an MVS-style save area, a common DTE control block foundation and an optional variable length work area. The DTEs are on a double headed and double threaded chain. The HCT contains $DTEORG which is the origin of the DTE chain, and $DTELAST, the last DTE on the chain. The DTEs themselves contain pointers DTENEXT and DTEPREV. Within the chain, the DTEs are grouped by subtask type with HCT pointers into the DTENEXT chain.

In addition to the chain fields, the DTE foundation contains three ECB pairs used for subtask/main task communication. The DTE contains a pointer to the subtask task control block (TCB) (the TCB also points back to the DTE). The DTE foundation contains an error recovery area (ERA) and a general parameter list area, and if there is a one-to-one relationship between the subtask and a PCE, the DTE foundation contains a PCE pointer.

$DTEDYN Services

The generalized subtask management service is called $DTEDYN and invoked through the $DTEDYN macro. The service, which resides in HASPDYN, contains two entry points. Those entry points are $DTEDYNA for ATTACH and $DTEDYND for DETACH. These are two different service routines called by one macro. The $DTETAB tables are used to define general subtask characteristics such as whether the subtask is attached during JES2 initialization, whether a unique subpool 0 is required, and whether the subtask can be forced down (that is, abnormally terminated through the specification of STAE=YES on the MVS DETACH macro) during JES2 termination.

You can use the $DTETAB tables to create your own subtasks without modifying JES2 code. Subtasks can be automatically attached during JES2 initialization, from a user-defined exit routine or through a user-defined processor. Subtasks are able to communicate with a user-defined processor using JES2 XECBs or the generalized dispatcher queues. Subtasks can be detached by a user-defined processor, exit routine, or automatically detached during JES2 termination. Also, catastrophic errors ($DOx) are issued by $DTEDYN to catch user logic and table errors early during testing.

Generalized JES2 Dispatcher Support

The JES2 dispatcher is completely generalized, thereby making it easy for you to add processor (PCE) resource queues without source code modification.

There are 64 general resource queues, of which JES2 uses the first 23. The resource queue heads are in their own area pointed to by the $DRQUES field of the HCT. The HCT also contains the event control fields and the $READY queue.

You can issue your own $WAITs and $POSTs, using equated symbols (or hard-coded values, that is, $WAIT 54) in the same manner as JES2. You can also use $$POST to post JES2 main task resources from subtasks or other address spaces. The cross-system operand (MASPOST=) on the $POST macro is also supported in this generalized scheme.

Chapter 3: IBM-Defined Exits

Chapter 3. IBM-Defined Exits

This is a reference section. It provides the information you need in writing exit routines for the IBM-defined exits.

The exits are described in the order of their identification numbers, the *ID* numbers assigned to them on their respective $EXIT macros. Each exit description begins with a discussion of its recommended use, followed by a breakdown of environmental considerations, linkage conventions, and other programming considerations specific to the particular exit being described. (Note: For convenience, except where single or multiple exit routines are mentioned specifically--as in the distinction between return code 0 and return code 4--the following descriptions imply either one or more exit routines by the inclusive term "exit routine." For example, "your exit routine may replace the standard routine" should be read to imply "your exit routine or exit routines may replace the standard routine.") Figure 3-1 summarizes for each exit the exit points in JES2 where your exit routine can get control.

Exit Implementation Table

The following table is a reference to the various exit points that are defined for the IBM-defined exits and the JES2 environment in which the exit point is taken. Use this table as an aid in implementing your exit routines.

Exit	Exit Title	Containing CSECT	Exit Point(s)	Environment
0	PRE-INITIALIZATION	HASPIRMA	NEXIT0	Main Task (Initialization) Job Exit Mask - N/A
1	PRINT/PUNCH SEPARATOR	HASPPRPU	PEXITSC PEXITTR	Main Task Job Exit Mask - A
2	JOB STATEMENT SCAN	HASPRDR	RXITJBCD RXITJBCC	Main Task Job Exit Mask - A
3	JOB STATEMENT ACCOUNTING FIELD SCAN	HASPRDR	RXITACC	Main Task Job Exit Mask - A
4	JCL AND JES2 CONTROL STATEMENT SCAN	HASPRDR	RXITCCA RXITCCB RXITCCC	Main Task Job Exit Mask - A
5	JES2 COMMAND PREPROCESSOR	HASPCOMM	COMMEXIT	Main Task Job Exit Mask - N/A
6	INTERNAL TEXT SCAN	HOSCNVT subtask of HASPCNVT	XCSTCUET XCSTMUEE XCSTCUEE	Subtask Job Exit Mask - A
7	JCT READ/WRITE (JES2)	HASPNUC	JIOEXIT	Main Task Job Exit Mask - A
8	JCT READ/WRITE (USER)	HASPSSSM	HXJCT01 HXJCT02 HXJCT03 HXJCT04	User address space Job Exit Mask - A
9	JOB OUTPUT OVERFLOW	HASPAM	SVCOUTX	User address space Job Exit Mask - A
10	$WTO SCREEN	HASPCON	WTOEXIT	Main Task Job Exit Mask - N/A
11	SPOOL PARTITIONING ALLOCATION -- $TRACK	HASPTRAK	$TRACKX	Main Task Job Exit Mask - A
12	SPOOL PARTITIONING ALLOCATION -- $STRAK	HASPSSSM	$STRAKX	User address space Job Exit Mask - A
13	TSO/E INTERACTIVE DATA TRANSMISSION FACILITY SCREENING AND NOTIFICATION	HASPNET	MAILXIT	Main Task Job Exit Mask - N/A
14	JOB QUEUE WORK SELECT	HASPNUC	QVALID	Main Task Job Exit Mask - N/A
15	OUTPUT DATA SET/COPY SEPARATORS	HASPPRPU	PEXT15D PEXT15O	Main Task Job Exit Mask - A
16	NOTIFY	HASPHOPE	OPNEXIT	Main Task Job Exit Mask - A

Figure 3-1 (Part 1 of 2). Exit Implementation Table

Exit	Exit Title	Containing CSECT	Exit Point	Environment
17	BSC RJE SIGN-ON/SIGN-OFF	HASPBSC	MSOXITA MSOXITB MDSXITA	Main Task Job Exit Mask - N/A
18	SNA RJE	HASPSNA	MICEXIT MALGXIT MSNALXIT MSNAXT2	Main Task Job Exit Mask - N/A
19	INITIALIZATION STATEMENT	HASPIRPL	NPLEXIT	Main Task (Initialization) Job Exit Mask - N/A
20	END OF JOB INPUT	HASPRDR	REXITA	Main Task Job Exit Mask - A
21	SMF RECORD	HASPNUC	SMFEXIT	Main Task Job Exit Mask - N/A
22	CANCEL/STATUS	HASPXEQ	ZTCSEXIT YTCSEXIT	Main Task Job Exit Mask - N/A
23	JOB SEPARATOR PROCESSING (JSPA)	HASPFSSM	JSPAXIT	Functional Subsystem Job Exit Mask - A
24	POST INITIALIZATION	HASPIRA	NEXIT24	Main Task (Initialization) Job Exit Mask - N/A
25	JCT READ I/O (FSS)	HASPFSSM	FGDSX25	Functional Subsystem Job Exit Mask - A
26	TERMINATION/RESOURCE RELEASE	HASPTERM	EX26	Main Task (Termination) Job Exit Mask - N/A

Figure 3-1 (Part 2 of 2). Exit Implementation Table

Exit 0: Pre-Initialization

FUNCTION:

This exit allows you to control the start of the initialization process through various means, such as:

- Processing JES2 initialization options, specifically the JES2 cataloged procedure parameter field and/or the replies to the $HASP426 and $HASP427 WTORs. The options can optionally be altered or bypassed.
- Acquiring installation control blocks and installation work areas for later initialization
- Providing user fields and addresses of installation-defined tables in the MCT. The table pointers in the master control table (MCT) allow your installation to extend JES2 processing of user tables to define JES2 initialization to extend or tailor certain table-driven JES2 functions. Define user table pointers in the MCT as MCTstmTU, where 'stm' is the JES2 initialization statement that you are replacing. Refer to "Defining JES2 Tables" in Chapter 2 for a list of the MCT names.
- Determining whether or not JES2 initialization is to proceed.

ENVIRONMENT: JES2 main task (Initialization) - JES2 dispatcher disabled

POINT OF PROCESSING:

This exit is taken in the initialization routine that processes the initialization options (IROPTS, in module HASPIRMA). The initialization options are taken from the parameter field specified via the JES2 procedure or START command, or are requested from the operator via the $HASP426 WTOR message if necessary. The point of processing for this exit is just before parsing and analyzing the options and setting appropriate flags. Exit 0 may be called a multiple number of times, because new options may be requested repetitively via the $HASP427 WTOR message until valid options are specified or the exit directs JES2 to bypass the options analysis.

The exit control blocks and the exit effector are not initialized at this point in IROPTS when Exit 0 gets control. Therefore, the normal JES2 exit facility initialization parameters cannot be used. IROPTS searches for module HASPXIT0 in the HASPINIT load module and then, if necessary, in the HASJES20 load module. The name HASPXIT0 is defined as a weak external reference (WXTRN) in both load modules. If HASPXIT0 is not found via this search, an MVS LOAD is issued for a separate load module named HASPXIT0. If HASPXIT0 is found through these means, a temporary XIT and XRT are built for the exit facility and the $EXIT macro. The HASPXIT0 module's MIT is searched for all entry point names of the form 'EXIT0nnn' and the entry point names found and the associated addresses are placed in the temporary XRT in the order they are found.

If HASPXIT0 is found during JES2 initialization, an entry for that module is placed in the exit facility LMT as if a LOAD initialization statement had been

processed for it and the module is not deleted. Therefore other exit routines (e.g., for Exits 19 and 24) and installation-defined tables (e.g., initialization statement $SCANTAB tables) can be assembled in the same module with the Exit 0 routines without having them deleted by JES2 after initialization completes. Note, however, that HASPXIT0 will be deleted from storage with HASPINIT if HASPXIT0 is linkedited with the HASPINIT load module.

REGISTER CONTENTS WHEN CONTROL IS PASSED TO THE EXIT ROUTINE:

R0 A code indicating where the initialization options were specified

- **0** Options passed are from the EXEC card, the PARM field
- **4** Options passed are from the $HASP426 message WTOR reply
- **8** Options passed are from a $HASP427 message WTOR reply

R1 Address of a 2-word parameter list with the following structure:

Word 1 (+0) address of the initialization options string
Word 2 (+4) length of the initialization options string

R2-R10 N/A

R11 Address of HCT

R12 N/A

R13 Address of initialization PCE - the PCE work area for this PCE is the common initialization routine work area, mapped by the $CIRWORK macro.

R14 Return address

R15 Entry address

REGISTER CONTENTS WHEN CONTROL IS PASSED BACK TO JES2:

R0-R1 N/A
R15 A return code

RETURN CODES:

0 Tells JES2 that if there are any additional exit routines associated with this exit, call the next consecutive exit routine. If there are no additional exit routines associated with this exit, continue with normal IROPTS processing..

4 Tells JES2 to ignore any additional exit routines associated with this exit and to continue with normal IROPTS processing.

8 Tells JES2 to bypass processing of the options string and assume the current values for the JES2 initialization options flags are correct.

12 Tells JES2 to terminate processing. This results in the $HASP864 error message to the operator.

JOB EXIT MASK: This exit point is not subject to job exit mask suppression.

MAPPING MACROS NORMALLY REQUIRED FOR THIS EXIT AT ASSEMBLY: $HASPEQU, $HCT, $MIT, $PCE, $CIRWORK

OTHER PROGRAMMING CONSIDERATIONS:

1. Tracing for this exit is disabled because of its sequence in the initialization process.

2. JES2 does not have a recovery environment established at the processing point for Exit 0 (the JES2 ESTAE will process termination but not recover).

3. Because Exit 0 is called early in JES2 initialization, some main task services may not be functional and most control blocks and interfaces are not yet established. The JES2 dispatcher is not yet functional, so MVS protocol should be used in Exit 0 routines (WAIT rather than $WAIT, ESTAE rather than $ESTAE, etc.).

4. If Exit 0 returns a return code of 12, IROPTS issues message $HASP864 indicating that Exit 0 terminated initialization. IROPTS then returns to the IRLOOP with return code 8, indicating that the $HASP428 message should be issued before final termination.

5. The initialization options string passed to Exit 0 is first 'folded', that is all the characters are 'folded' up to their capitalized versions.

6. The processing that JES2 does for the initialization options string after calling Exit 0 is performed using the JES2 $SCAN facility and a table that defines the options input allowed and how to process it. The table is actually composed of two tables, an installation-defined table followed by a JES2-defined table.

   By specifying installation-defined tables, an installation can implement its own initialization options or replace the JES2 definition for existing options. Thus this function can be accomplished without implementing Exit 0, or in conjunction with an implementation of Exit 0. Also, the $SCAN facility itself can be used from an Exit 0 exit routine to process initialization options.

Exit 1: Print/Punch Separators

FUNCTION:

This exit allows you to provide an exit routine to produce your own print/punch separators and to control production of standard print/punch separators.

When using this exit to create your own separators, you can devise entirely unique separators or you can create variations on the standard separators. When using this exit to control the production of standard separators, you can unconditionally suppress production of standard separators, you can direct JES2 to unconditionally produce standard separators, or you can allow JES2 to produce any standard separators that are in effect. Whether or not standard separators are in effect for any particular device is determined by how the initialization statement and operator command separator options have been defined at your installation at any given time; these options are described in the "Other Programming Considerations" below.

For punch devices, JES2 provides the option of producing start-of-job header cards and trailer cards. For printers, JES2 provides the option of producing start-of-job header pages, continuation-of-job header pages, and trailer pages. Continuation-of-job header pages are produced at each output data set group (represented by a work JOE) within a job and for the continuation of a data set group if printing has been interrupted. In each of these instances, whether or not the standard separator is in effect, when this exit is implemented and enabled your exit routine receives control. Therefore, you have the ability to control the production of separators on a job-by-job basis and, for printers/punches on a data set group basis.

Each time your exit routine is called, you can direct JES2:

- To produce only your own separator (unconditionally suppressing production of the standard separator)
- To produce only the standard separator, if it happens to be in effect (without producing your own separator)
- To produce the standard separator unconditionally
- To produce your own separator followed by the standard separator, if it happens to be in effect (for example, your own start-of-job header page followed by the standard start-of-job header page)
- To produce your own separator and then to produce the standard separator unconditionally
- To produce no separator (by not producing your own separator and by suppressing production of the standard separator)

Two exit points establish this exit. The first, at label PEXITSC, provides for insertion of a start-of-job header page/card or a job continuation header page exit

routine. The second, at label PEXITTR, provides for insertion of a trailer page/card exit routine.

ENVIRONMENT: JES2 main task

POINT OF PROCESSING:

This exit is taken from the HASPPRPU module in the JES2 main task. Both exit points occur prior to the check for standard separator pages.

Exit point PEXITSC occurs in the PTESTSEP area of the PHEADER (produce job header) subroutine, prior to the call of PUNCHSEP, if a standard separator card is to be produced, or PRINTSEP, if a standard separator page is to be produced.

Exit point PEXITTR occurs in the PPDONE (print/punch processor termination) routine, prior to the call of PRINTTR, if a standard trailer page is to be produced.

REGISTER CONTENTS WHEN CONTROL IS PASSED TO THE EXIT ROUTINE:

R0 A code indicating whether a header page or card, continuation page, or trailer page is being processed

- **0** Indicates a start-of-job page or card
- **4** Indicates a continuation-of-job page
- **8** Indicates a trailer page or card

R1 Address of the printer or punch DCT

R2-R9 N/A

R10 Address of the JCT

R11 Address of the HCT

R12 N/A

R13 Address of the PCE

R14 Return address

R15 Entry address

REGISTER CONTENTS WHEN CONTROL IS PASSED BACK TO JES2:

R0-R1 N/A
R15 A return code

RETURN CODES:

0 Tells JES2 that if there are any additional exit routines associated with this exit, call the next consecutive exit routine. If there are no additional exit routines associated with this exit, produce any installation-defined separator that has been created and, if standard separator production is in effect for the printer or punch, produce a standard separator.

4 Tells JES2 to ignore any additional exit routines associated with this exit, to produce any installation-defined separator that has been created and, if standard separator production is in effect for the printer or card punch, produce a standard separator.

8 Tells JES2 to produce any installation separator that has been created and to unconditionally suppress production of the standard separator.

12 Tells JES2 to produce any installation separator that has been created and to unconditionally (that is, even if the printer has been set to S=N) produce the standard separator (following production of any user separator).

JOB EXIT MASK: This exit is subject to job exit mask suppression.

MAPPING MACROS NORMALLY REQUIRED FOR THIS EXIT AT ASSEMBLY: $DCT, $HASPEQU, $HCT, $JCT, $MIT, $PCE

OTHER PROGRAMMING CONSIDERATIONS:

1. You cannot use this exit to modify the standard separator routines directly. If you intend to produce a modified version of a standard separator, your exit routine must replace the standard separator routine entirely and is responsible for producing the standard separator elements that you want to retain as well as your new or modified separator elements.

2. This exit is available to provide a user-written separator page for local or RJE printers only. There is no separator page for JES2 or user-supplied networking output. If separator pages are required for networking output jobs, they must be supplied (through use of this exit) at the destination node when printed

3. The DCTPPSWS flag in the DCT indicates whether or not standard separators are to be produced for a particular device.

4. For each device, initialization statements first determine whether or not standard separators are in effect--that is, whether or not, without the intercession of an exit routine, JES2 would normally produce or suppress standard separators.

   For a local printer, the NOSEP parameter of the PRTnnnn statement specifies that separator pages are not to be produced, and the SEP parameter, the default, specifies that separator pages are to be produced. However, even if SEP is specified, if the local printer separator page line count parameter, SEPLINE parameter on the PRINTDEF statement, is set to zero, no separator pages are produced.

   For a remote printer, the NOSEP parameter of the Rnnnn.PRm statement specifies that separator pages are not to be produced, and the SEP parameter, the default, specifies that separator pages are to be produced. However, even if SEP is specified, if the remote printer separator page line count parameter, RSEPLINE parameter on the PRINTDEF statement, is set to zero, no separator pages are produced.

   For a local card punch, the NOSEP parameter of the PUNnn statement specifies that separator cards are not to be produced, and the SEP parameter, the default, specifies that separator cards are to be produced.

For a remote card punch, the NOSEP parameter of the Rnnnn.PUm statement specifies that separator cards are not to be produced, and the SEP parameter, the default, specifies that separator cards are to be produced.

After JES2 has been started, the operator can use the S option of the $T PRT or $T PUN command to change the status of any printer or card punch. For any device, the operator issues the $T command with S=Y to specify that standard separators are to be produced and with S=N to specify that standard separators are not to be produced.

5. Use the *$PRPUT service routine* to produce any new separators created by your exit routine. $PRPUT accesses the same routines used by HASPPRPU to produce output. It performs all necessary interfacing with the low-level CCW building and I/O routines in HASPPRPU. $PRPUT constructs a CCW for each data record to be output. It then calls PPUT to put the CCW into an output buffer. Note that the data area supplied to $PRPUT must be fixed. Use $GETBUF to supply this routine with buffers and $FREEBUF to release them after your separator has been created. Buffers supplied to $PRPUT must be HASP-type buffers.

6. When using $PRPUT with WAIT=NO, I/O does not occur synchronously; the device does not physically process the data areas until either a $PRPUT macro specified with WAIT=YES is issued or the CCW area is exhausted. Therefore, do not reuse data areas until after you issue $PRPUT with WAIT=YES specified.

7. Use the *$PBLOCK service routine* to create block letters on any new separator page created by your exit routine. Note that, for a local printer, if the $PRIDCT separator page line count parameter is specified as less than 30 and, for a remote printer, if the $TPIDCT separator page line count parameter is specified as less than 30, no block letters can be produced. Use $GETBUF to supply this routine with buffers and $FREEBUF to release them after your separator has been created. Buffers supplied to $PBLOCK must be HASP-type buffers rather than PP-type buffers.

8. If the spooling capabilities of a remote SNA device (such as the 3790) are in use, use the *$SEPPDIR service routine* to send a peripheral data information record (PDIR) to the device. Use the $GETBUF macro to supply this routine with buffers and the $FREEBUF macro to release them after your separator has been created. Buffers supplied to $SEPPDIR must be HASP-type buffers.

9. If a hardware error or intervention situation interrupts $PRPUT processing, Exit 1 will lose control. Whatever resources (for example, JES2 buffers or MVS virtual storage) obtained by this exit routine are not unallocated by JES2 at this time. You can prevent this situation from occurring if your exit routine saves the address of the resources in a PCE field such as PCEUSER0 and checks for an address(es) upon entry to the exit routine. Previously acquired resources can thereby be reused. Before returning to JES2, this routine should release these resources and the pointer field(s) used should be zeroed.

10. To avoid feeding blank pages through your printer, be certain to include a page eject statement in your exit routine following the trailer separator page. This is required by some printers because the printer is not repositioned to the "top of forms" after printing the trailer page.

11. If JES2 abnormally terminates (ABENDS) when Exit 1 is enabled, be certain to check the following:

    - $PRPUT must be printing from a fixed area of storage, that is, an area obtained with $GETBUF FIX=YES.
    - $PRPUT is coded with WAIT=YES to ensure that the data is actually printed prior to freeing the buffer.
    - The returned buffer is not being used.
    - The I/O processing does not place data beyond the buffer limit.
    - The exit is JES2 reentrant.

12. *Note on recovery:* There is no JES2 recovery in effect. As with every exit, you should supply your own recovery within your exit routine.

Exit 2: JOB Statement Scan

FUNCTION:

This exit allows you to provide an exit routine for scanning the complete JOB statement image, and for setting the corresponding fields in the appropriate JES2 control blocks. If this exit is implemented and enabled, it is taken whenever JES2 encounters a JOB statement or a JOB continuation statement.

You can also use your exit routine to interpret JOB statement input and, on the basis of this interpretation, to decide whether to cancel the job, to purge it from the system, or to allow it to continue processing normally. Your routine can also alter JOB statement parameters and supply additional JOB statement parameters (to include accounting information). If necessary, when supplying expanded JOB statement data, your routine can pass an additional JOB continuation statement image back to JES2.

You can use this exit to directly alter the JCT. Since, if enabled, this is usually the first exit taken for a job, you would use this exit if you wanted to control (enable or disable) other exits on a job-by-job basis by setting the job exit mask in the JCT. For more information, see "Job-Related Exits" in Chapter 2. If you want to implement spool partitioning, you can use this exit to set the spool partitioning mask in the JCT. For more information, see the description of spool partitioning in the description of Exit 11 below.

If this exit isn't taken, JES2 continues with the standard HASPRDR JOB statement scan processing.

This exit is established by two exit points. The first, at label RXITJBCD, gives control to your routine when JES2 encounters a JOB statement. The second, at label RXITJBCC, gives control to your routine when JES2 encounters a JOB continuation statement.

ENVIRONMENT: JES2 main task

POINT OF PROCESSING:

This exit is taken from the JES2 main task, from the HASPRJCS subroutine of HASPRDR during JOB statement processing.

Exit point RXITJBCD occurs after the JCT and IOT have been obtained and initialized for a new job but before HASPRDR performs its standard JOB statement scan and before any spool space has been assigned.

In the case of JOB continuation statements, exit point RXITJBCC occurs in subroutine RCONTNUE.

Note: Refer to Appendix D, "JES2 Exit Usage Limitations" for a listing of specific instances when this exit will be invoked or not invoked.

REGISTER CONTENTS WHEN CONTROL IS PASSED TO THE EXIT ROUTINE:

R0 A code indicating the type of JOB statement being scanned

- **0** indicates an initial JOB statement image
- **4** indicates a subsequent JOB continuation statement

R1 Address of a 3-word parameter list with the following structure:

- **Word 1** (+0) points to the JOB statement image buffer
- **Word 2** (+4) points to the exit flag byte, RDWFLAGX, in the PCE
- **Word 3** (+8) points to the JCTXWRK field in the JCT

R2-R9 N/A

R10 Address of the JCT

R11 Address of the HCT

R12 N/A

R13 Address of the PCE

R14 Return address

R15 Entry address

REGISTER CONTENTS WHEN CONTROL IS PASSED BACK TO JES2:

R0-R1 N/A
R15 A return code

RETURN CODES:

0 Tells JES2 that if there are any additional exit routines associated with this exit, call the next consecutive exit routine. If there are no additional exit routines associated with this exit, continue with normal HASPRDR processing.

4 Tells JES2 to ignore any additional exit routines associated with this exit and to continue with normal HASPRDR processing.

8 Tells JES2 to cancel the job; output (the incomplete JCL images listing) is produced.

12 Tells JES2 to purge the job; no output is produced.

JOB EXIT MASK: This exit is subject to job exit mask suppression. Note, however, that because this exit is used to set the job exit mask, it can only disable itself for a second invocation, for a JOB continuation statement.

MAPPING MACROS NORMALLY REQUIRED FOR THIS EXIT AT ASSEMBLY: $PCE, $RDRWORK, $JCT, $HCT, $BUFFER, $MIT, $HASPEQU, RPL

OTHER PROGRAMMING CONSIDERATIONS:

1. This exit is suitable for setting default values in the JCT because the JOB or JOB continuation statements have not been scanned. However, exercise caution in using this exit to alter the JCT. Because JCL processing is not complete at this point, subsequent JCL processing can potentially override certain fields in the JCT that you may intend to set from this exit. Unless

you are certain that further JCL processing will not override the JCT fields you intend to set from this exit, consider setting those fields from Exit 3, which occurs later in the HASPRDR processing routine. However, because this is usually the first exit taken for a job, this is the *optimum* exit for certain JCL alterations--notably, for setting the job exit mask (JCTXMASK) and the spool partitioning mask (JCTSAMSK).

2. An 80-byte work area in the JCT, JCTXWRK, is available for use by your routine. If your routine requires additional work space, use the GETMAIN macro to obtain storage (and the FREEMAIN macro to return it to the system when your routine has completed).

3. When passing a return code of 0 or 4, your exit routine can expand the JOB statement to include additional input by returning an additional statement image. JES2 receives this continuation statement as the next statement to be read and processed by HASPRDR. To return this job continuation statement to JES2, (1) move a comma into the last byte of the current job statement image following standard JCL syntax, and (2) move the statement image to the JCTXWRK field and set RDWXXSNC bit in the RDWFLAGX byte to one. If, however, an additional statement (not a JOB statement image) is to be returned to JES2, only step 2 should be followed, and do not add a comma to the current job statement image.

   RDWFLAGX has the following structure:

   RDWXJCL (X'01') when on indicates that JES2 has detected a JCL statement.

   RDWXJECL (X'02') when on indicates that JES2 has detected a JES2 control statement.

   RDWXJOBC (X'04') when on indicates that JES2 has detected a JOB statement.

   RDWXCONT (X'08') when on indicates that JES2 has detected a DD DATA or DD * continuation statement.

   RDWXXSNC (X'10') when on indicates that an installation exit has supplied the next card image.

   RDWXXSEM (X'20') when on indicates that an installation exit has supplied an error message.

4. When passing a return code of 8, your exit routine can pass an installation-defined error message to JES2 to be added to the JCL data set rather than the standard error message. To send an error message, generate the message text in your exit routine, move it to JCTXWRK, and set the RDWXXSEM bit in RDWFLAGX to one.

5. *Note on recovery:* $ESTAE recovery is in effect. The RDRRCV0 recovery routine will attempt to recover from program check errors, including program check errors in the exit routine itself. However, as with every exit, your exit routine for this exit *should not* depend on JES2 for recovery. JES2 cannot anticipate the exact purpose of your exit routine and can therefore provide no more than minimal recovery. Provide your own recovery within your exit routine.

6. For certain jobs, such as SYSLOG and $TRCELOG, this exit is not taken.

Exit 3: JOB Statement Accounting Field Scan

FUNCTION:

This exit allows you to provide an exit routine for scanning the JOB statement accounting field and for setting the corresponding fields in the appropriate JES2 control blocks.

You can use your exit routine to interpret the variables in the accounting field and, on the basis of this interpretation, to decide whether or not to cancel the job. Use Exit 2 to alter the accounting information and supply new accounting information at the time the entire JOB statement is first scanned. Use this exit to record alterations to the accounting field; they will not appear on the user's output but are reflected in the JCT and when the SMF type 6 record is written.

This exit is associated with the existing HASPRSCN accounting field scan subroutine. You can write your exit routine as a replacement for HASPRSCN or you can use a return code to direct HASPRJCS to call HASPRSCN after your exit routine has executed. In either case, when this exit is implemented and enabled, JES2 treats your exit routine as the functional equivalent of HASPRSCN. The specification of the ACCTFLD parameter on the JOBDEF initialization statement, which normally determines whether JES2 is to call HASPRSCN, becomes an additional factor in determining whether your exit routine is to be called. The exit is taken only if the ACCTFLD= parameter on the JOBDEF initialization statement is specified as either REQUIRED or OPTIONAL. The exit is not taken if ACCTFLD=IGNORE is specified. When it is called, your exit routine--rather than When it is called, your exit routine--rather than the ACCTFLD parameter--determines whether HASPRSCN is to be executed as an additional scan of the accounting field. For a complete explanation of how the ACCTFLD parameter is specified, refer to *JES2 Initialization and Tuning*. The relationship of HASPRSCN to this exit is described in greater detail in the "Other Programming Considerations" below.

This exit is established by a single exit point, at label RXITACC.

ENVIRONMENT: JES2 main task

Note: Refer to Appendix D, "JES2 Exit Usage Limitations" for a listing of specific instances when this exit will be invoked or not invoked.

POINT OF PROCESSING:

This exit is taken from the JES2 main task, from the HASPRJCS JOB statement scan subroutine of HASPRDR. The exit occurs after JES2 has scanned the entire JOB statement, but prior to the execution of the HASPRSCN accounting field scan subroutine, if HASPRSCN is to be called. The JCT has been initialized with the JES2 and installation defaults; in addition, those fields of the JCT that correspond to JOB statement parameters other than accounting field parameters have been set. The JCTWORK field of the JCT contains the accounting field image.

REGISTER CONTENTS WHEN CONTROL IS PASSED TO THE EXIT ROUTINE:

R0 The length of the accounting field, in bytes; if no accounting field has been specified, this length is zero

R1 Address of a 3-fullword parameter list

Word 1 (+0) points to the accounting field (JCTWORK in the JCT)
Word 2 (+4) points to the exit flag byte, RDWFLAGX in the PCE
Word 3 (+8) points to the JCTXWRK field in the JCT

R2-R9 N/A

R10 Address of the JCT

R11 Address of the HCT

R12 N/A

R13 Address of the HASPRDR PCE

R14 Return address

R15 Entry address

REGISTER CONTENTS WHEN CONTROL IS PASSED BACK TO JES2:

R0-R1 N/A
R15 A return code

RETURN CODES:

0 Tells JES2 that if there are any additional exit routines associated with this exit, call the next consecutive exit routine. If there are no additional exit routines associated with this exit, use the current setting of the ACCTFLD parameter on the JOBDEF statement to determine whether or not to execute the HASPRSCN subroutine.

4 Tells JES2 to ignore any other exit routines associated with this exit and to use the current setting of the ACCTFLD parameter on the JOBDEF statement to determine whether or not to execute HASPRSCN.

8 Tells JES2 to suppress execution of HASPRSCN and to complete HASPRJCS processing.

12 Tells JES2 to cancel the job because an illegal accounting field has been detected. Tells JES2 to suppress execution of HASPRSCN and to queue the job for output; output (the incomplete JCL images listing) is produced.

JOB EXIT MASK: This exit is subject to job exit mask suppression.

MAPPING MACROS NORMALLY REQUIRED FOR THIS EXIT AT ASSEMBLY: $PCE, $RDRWORK, $JCT, $HCT, $MIT, $BUFFER, $HASPEQU, RPL

OTHER PROGRAMMING CONSIDERATIONS:

1. The accounting field resides in a 144-byte work area, JCTWORK, in the JCT. The address of JCTWORK is in the first word of the 3-word parameter list whose address is passed to the exit routine in R1.

2. If you need to verify the existence of a JOB rather than a started task (STC) or TSO logon, this can be done by comparing the JCTJOBID field to a "J." The presence of a "J" indicates the existence of a JOB.

3. The ACCTFLD parameter on the JOBDEF statement indicates whether JES2 should scan the accounting field of a JOB statement. For further details concerning the use of the ACCTFLD parameter, refer to *JES2 Initialization and Tuning*.

   If the ACCTFLD parameter indicates that the scan should be performed, and if this exit is implemented and enabled, then HASPRJCS calls your exit routine to perform the scan. If your exit routine passes a return code of 0 or 4 to JES2, then HASPRJCS calls the existing HASPRSCN accounting field scan subroutine after your routine has executed. Note that if both routines are to be called, your routine should not duplicate HASPRSCN processing. For example, your routine should not set the fields in the JCT that are set by HASPRSCN. However, if your routine passes a return code of 8 or 12 to JES2, it causes JES2 to suppress execution of HASPRSCN. If the ACCTFLD parameter indicates that the scan should be performed but this exit is disabled, then only HASPRSCN is called; your exit routine is not called and is not given the opportunity to allow or suppress HASPRSCN execution. If the ACCTFLD parameter indicates that a scan should not be performed, your exit routine is not called, even if this exit is enabled, and execution of HASPRSCN is also suppressed.

4. The ACCTFLD parameter on the JOBDEF statement indicates whether JES2 should cancel a job if the accounting field on the JOB statement is invalid or if a JCL syntax error has been detected during HASPRJCS processing. Note that your exit routine can affect this termination processing. For example, ACCTFLD=REQUIRED indicates that JES2 should scan the accounting field, that the job should be canceled if the accounting field is invalid, and that the job should be canceled if a JCL syntax error has been found. If you pass a return code of 8 to JES2, HASPRSCN is not called and therefore cannot terminate a job with an invalid accounting field, even though ACCTFLD=REQUIRED. Also note that HASPRSCN scans the JCTWORK field of the JCT. Therefore, if your routine alters this field, you affect HASPRSCN processing.

5. The specification of the ACCTFLD parameter is stored in the HCT, in field $RJOBOPT. If your exit routine is meant to completely replace HASPRSCN, you may want to access this field for use by your algorithm.

6. In general, use this exit, rather than Exit 2, to alter the JCT directly. If you use Exit 2 to alter the JCT, subsequent processing might override your changes. The job exit mask and the spool partitioning mask are exceptions. See note 2 of Exit 2 for more information.

7. An 80-byte work area in the JCT, at label JCTXWRK, is available for use by your routine. If your routine requires additional work space, use the GETMAIN macro to obtain storage (and the FREEMAIN macro to return it to the system when your routine has completed).

8. When passing a return code of 8, your exit routine can pass an installation-defined error message to JES2 to be added to the JCL data set rather than the standard error message. To send an error message, generate the message text in your exit routine, move it to JCTXWRK, and set the RDWXXSEM bit in RDWFLAGX to one.

   RDWFLAGX has the following structure:

RDWXJCL	(X'01') when on indicates that JES2 has detected a JCL statement
RDWXJECL	(X'02') when on indicates that JES2 has detected a JES2 control statement
RDWXJOBC	(X'04') when on indicates that JES2 has detected a JOB statement
RDWXCONT	(X'08') when on indicates that JES2 has detected a DD DATA or DD * continuation statement
RDWXXSNC	(X'10') when on indicates that an installation exit has supplied the next card image
RDWXXSEM	(X'20') when on indicates that an installation exit has supplied an error message

9. If there is no accounting field on a JOB statement, the length passed by JES2 to the exit routine in R0 is zero. Your exit routine should take this possibility into account.

10. If you intend to use this exit to process nonstandard accounting field parameters, you should either suppress subsequent execution of HASPRSCN or you should code your exit routine to delete nonstandard parameters before passing control to HASPRSCN. If you do neither, that is, if you allow HASPRSCN to receive the nonstandard parameters, it might cancel the job because of an illegal accounting field (depending on how the ACCTFLD parameter on the JOBDEF statement is specified).

    If you change the length of the accounting field, you must reload the address of the last character (or terminator) into field RDWSAVE1.

11. There are two job class fields in the JCT. JES2 uses one to change the job's execution class and the other to retain the job's original job class. JES2 also uses these two fields to set the job class in the JQE and transfers the other to the job management record. (JMR) for use in SMF processing. If you intend to use your exit routine to change the job class, your exit routine should take both fields into account.

12. *Note on recovery:* $ESTAE recovery is in effect. The RDRRCV0 recovery routine will attempt to recover from program check errors, including program check errors in the exit routine. However, as with every exit, your exit routine for this exit *should not* depend on JES2 for recovery. JES2 cannot anticipate the exact purpose of your exit routine and can therefore provide no more than minimal recovery. You should provide your own recovery within your exit routine.

Exit 4: JCL and JES2 Control Statement Scan

FUNCTION:

This exit allows you to provide an exit routine for scanning JCL and JES2 control statements. If this exit is implemented and enabled, it is taken whenever JES2 encounters a JCL or JES2 control statement. (Note: JOB statements and internal reader control statements such as /*DEL are not included in the scan.)

For JCL statements, your exit routine can interpret JCL parameters and, on the basis of this interpretation, decide whether JES2 should cancel the job, purge the job, or allow the job to continue normally. Your routine can also alter JCL parameters and supply additional JCL parameters. If necessary, in supplying expanded JCL data, your routine can pass a JCL continuation statement back to JES2. It can also pass back a new JCL statement, such as a new DD statement.

For JES2 control statements, your routine can interpret the JES2 control parameters and subparameters and, on the basis of this interpretation, decide whether JES2 should cancel the job, purge the job, or allow the job to continue normally. For any JES2 control statement, you can write your exit routine as a replacement for the standard HASPRCCS control statement routine, suppressing execution of the standard JES2 scan, or you can perform your own (partial) processing and then allow JES2 to execute the standard HASPRCCS control statement routine. In addition, your routine can alter a JES2 control statement and then pass the modified statement back to JES2 for standard HASPRCCS processing, or your routine can pass an entirely new JES2 control statement back to JES2, to be read (and processed) as the next incoming statement by HASPRDR.

This exit also allows you to process your own installation-specific JES2 control statements or to implement new, installation-specific subparameters for existing JES2 control statements.

Three exit points establish this exit. The first, at label RXITCCA, gives control to your exit routine when JES2 detects a JES2 control statement or JCL statement within a job. The second, at label RXITCCB, gives control to your exit routine when JES2 detects a JES2 control statement or JCL statement outside of a job. The third, at label RXITCCC, gives control to your exit routine when JES2 detects a JCL DD * or DD DATA continuation statement.

ENVIRONMENT: JES2 main task

Note: Refer to Appendix D, "JES2 Exit Usage Limitations" for a listing of specific instances when this exit will be invoked or not invoked.

POINT OF PROCESSING:

This exit is taken from HASPRDR in the JES2 main task. The exit occurs in HASPRDR's main processing loop.

Exit point RXITCCA processing JES2 control statements occurs after HASPRDR has encountered an apparent JES2 control statement or JCL statement within a job and prior to control statement processing.

Exit point RXITCCB occurs after HASPRDR has encountered an apparent JES2 control statement or JCL statement outside a job and prior to control statement processing.

Exit point RXITCCC, for processing JCL DD DATA or DD * continuation statements, occurs after the RCONTNUE subroutine has detected a continuation statement and just before RCONTNUE returns control to the calling routine.

REGISTER CONTENTS WHEN CONTROL IS PASSED TO THE EXIT ROUTINE:

R0 A code indicating whether a JES2 control or JCL control statement is being processed

- **0** indicates a JES2 control statement
- **4** indicates a JCL statement

R1 Pointer to a 3-word parameter list with the following structure:

- **Word 1 (+0)** address of the control statement image buffer
- **Word 2 (+4)** address of the exit flag byte, RDWFLAGX, in the PCE
- **Word 3 (+8)** address of the JCTXWRK field in the JCT

R2-R9 N/A

R10 Address of the JCT

R11 Address of the HCT

R12 N/A

R13 Address of the PCE

R14 Return address

R15 Entry address

REGISTER CONTENTS WHEN CONTROL IS PASSED BACK TO JES2:

R0-R1 N/A
R15 A return code

RETURN CODES:

0 Tells JES2 that if there are any additional exit routines associated with this exit, call the next consecutive exit routine. If there are no additional exit routines associated with this exit, perform standard HASPRDR processing.

4 Tells JES2 to ignore any other exit routines associated with this exit and to perform standard HASPRDR processing.

8 For JES2 control statements, tells JES2 not to perform standard HASPRCCS processing; instead, immediately convert the statement to a command (//*) with the null-on-input flag set to one and write the statement to the JCL data set. For JCL statements, tells JES2 to perform standard HASPRDR processing.

12 Tells JES2 to cancel the job because an illegal control statement has been detected; output (the incomplete JCL images listing) is produced.

16 Tells JES2 to purge the job because an illegal control statement has been detected; no output is produced.

JOB EXIT MASK: This exit is subject to job exit mask suppression.

MAPPING MACROS NORMALLY REQUIRED FOR THIS EXIT AT ASSEMBLY: $HCT, $JCT, $MIT, $PCE, $RDRWORK, $BUFFER, $HASPEQU, RPL

OTHER PROGRAMMING CONSIDERATIONS:

1. This exit is taken once for each control statement (with the exception of JOB statements and internal reader control statements) encountered by JES2. A code in R0 indicates whether the current statement is a JCL statement or a JES2 control statement. Your exit routine gets control for //* comment, /* (generated), and /* PRIORITY JES2 control statements. Your exit routine gets control for DD * and DD DATA JCL statements.

2. During HASPRDR processing, JES2 writes the JCL records to a JCL data set. If an error occurs during HASPRDR processing, it is the JCL data set that is printed when the job goes through output processing. If the job is successfully processed by HASPRDR, the JCL data set is the input for the converter. The converter produces a JCL images data set, which is the data set that is printed when the job goes to output processing after being successfully processed by HASPRDR. Normally, when HASPRDR receives a JES2 control statement HASPRDR first writes the statement to the JCL data set with the null-on-input flag set to one. Because the statement is null-on-input it is passed over by the converter; however, because the null-on-output flag has not been set to one, the statement appears in user's copy of the JCL data set if the job bypasses conversion because an error was detected in HASPRDR processing. Next, HASPRDR calls one of the specific HASPRCCS control statement processing routines to perform the function requested by the JES2 control statement. When the standard routine has completed execution, HASPRDR converts the JES2 control statement to a comment, sets its null-on-output flag to one, and writes it to the JCL data set. Because the statement is null-on-output, it does not appear in the user's copy of the JCL data set; however, because the null-on-input flag has not been set to one, the JES2 control statement is read (as a comment) by the converter. This flagging scheme enables the user to see each JES2 control statement as it was received by HASPRDR if the job does not go to the converter, and enables the converter to see each JES2 control statement as a comment, which will appear in the users output when the JCL images data set is printed. (Note: The /*$ command and the /*PRIORITY statements are exceptions.)

   When this exit is implemented and enabled, it receives control after the initial copy of the JES2 control statement, with the null-on-input flag set to one, has been written to the JCL data set. Therefore, unless your exit routine passes a

return code of 16, which purges the job with no resulting output, the user sees the JES2 control statement on his output, just as it was received by HASPRDR, if the job goes directly to output phase (bypassing converter), otherwise, the user will see it as a comment in the JCL images file.

If you pass a standard return code (of 0 or 4) from your exit routine, the standard HASPRCCS routine executes. After the HASPRCCS routine is finished, HASPRDR writes the statement to the JCL data set as a comment with the null-on-output set to one. This form of the statement will be seen by the converter, and the converter will place it in the JCL images file.

If you pass a return code of 8, standard HASPRCCS processing is suppressed and HASPRDR immediately converts the statement to a comment with the null-on-output flag set, and writes the statement to the JCL data set. This form of the statement will be seen by the converter, and will be seen by the user as a comment in the JCL images file when it is printed. Note that, on the basis of the JCL images output data set he sees, the user has no indication that his statement was not processed normally.

If you pass a return code of 12, normal HASPRDR processing is halted, the statement is not processed by HASPRCCS, nor is it written to the JCL data set as a comment. The user sees the original statement on his output, as the last statement in the JCL data set, indicating that this JES2 control statement caused the job to be cancelled.

Finally, return code 16 also halts normal HASPRDR execution, but with no resulting output; the user has no indication of why his job failed.

3. When passing a return code of 0, 4, or 8, you may supply an additional statement image to be read and processed as the next statement in the input stream. For JCL statements, this can be a continuation statement or a new statement. For JES2 control statements, this must always be a new statement. To return this additional statement to JES2, move the statement image to the JCTXWRK field and set the RDWXXSNC bit in the RDWFLAGX byte to one.

   RDWFLAX has the following structure:

   RDWXJCL (X'01') when on indicates that JES2 has detected a JCL statement.

   RDWXJECL (X'02') when on indicates that JES2 has detected a JES2 control statement.

   RDWXJOBC (X'04') when on indicates that JES2 has detected a JOB statement.

   RDWXCONT (X'08') when on indicates that JES2 has detected a JCL DD DATA or DD * continuation statement.

   RDWXXSNC (X'10') when on indicates that an installation exit has supplied the next card image.

   RDWXXSEM (X'20') when on indicates that an installation exit has supplied an error message.

4. To entirely replace standard HASPRCCS processing for a particular JES2 control statement, write your routine as a replacement version of the standard HASPRCCS routine and then pass a return code of 8 back to JES2 to suppress standard processing. Note that your routine becomes responsible for

duplicating any HASPRCCS function you wish to retain. If you merely want to supplement standard HASPRCCS processing, you can write your exit routine to perform the additional function and then, by passing a return code of 0 or 4, direct JES2 to execute the standard HASPRCCS routine.

5. To nullify a JES2 control statement, pass a return code of 8 to JES2 without using your exit routine to perform the function requested by the statement. Note that, on the basis of the JCL images output data set, the user is not informed that the statement was nullified.

6. To modify a JES2 control statement, also use return code 8. Place the altered statement in JCTXWRK and set RDWXXSNC to one. If HASPRDR processing is successful, the user will see in the output of the JCL images file the original statement (as a comment statement), and the altered statement (also as a comment statement). Note, that if you modify a JES2 control statement and then pass a return code of 0 or 4, JES2 performs normal HASPRDR (HASPRCCS) processing, and the modified version of the statement will appear on the user's output in the JCL images file, but the original statement will not appear unless you go directly to output phase (bypassing the converter); then, the user will see the original statement when the JCL data set is printed.

7. Also use return code 8 in processing your own installation-specific JES2 control statements. Write your exit routine to perform the function requested by the statement and then pass return code 8 to JES2 to suppress standard processing and thereby prevent JES2 from detecting the statement as "illegal."

8. To process your own installation-specific JES2 control statement subparameters, you should generally write your exit routine to replace standard HASPRCCS processing entirely. That is, write your exit routine to perform the function(s) requested by the standard parameters and subparameters as well as those requested by any unique installation-defined subparameters on a statement. Then, from your exit pass a return code of 8 back to JES2. In general, because the parameters and subparameters on a JES2 control statement are interdependent, you will be limited to this method. However, if you have defined an installation-specific subparameter which can be processed independently of the rest of the control statement on which it appears, you can write your exit routine to process this subparameter alone, then to delete it, and then to pass a return code of 0 or 4 to JES2. JES2 can then process the remainder of the statement as a standard JES2 control statement.

9. When passing a return code of 12 or 16, it is also possible for your exit routine to pass an error message to JES2 for display at the operator's console. To send an error message, generate the message text in your exit routine, move it to JCTXWRK, and set the RDWXXSEM bit in RDWFLAGX to one.

10. If you intend to use this exit to affect the JCT, your exit routine must ensure the existence of the JCT upon receiving control. If the JCT has not been created when your exit routine receives control, the pointer to JCTXWRK, the third word of the 3-word parameter list whose address is passed to your exit routine in R1, is zero. For example, when your exit routine receives

control for a /*PRIORITY statement, the JCT doesn't exist yet. In this case, your routine must store any data to be placed in the JCT until JES2 creates the JCT.

11. Your exit routine does not have access to the previous control card image. You should take this into account when devising your algorithm.

12. An 80-byte work area, JCTXWRK, is available for use by your exit routine. If your routine requires additional work space, use the GETMAIN macro to obtain storage (and the FREEMAIN macro to return it to the system when your routine has completed).

13. *Note on recovery:* $ESTAE recovery is in effect. The RDRRCV0 recovery routine will attempt to recover from program check errors, including program check errors in the exit routine itself. However, as with every exit, your exit routine *should not* depend on JES2 for recovery. JES2 cannot anticipate the exact purpose of your exit routine and can therefore provide no more than minimal recovery. You should provide your own recovery within your exit routine.

Exit 5: JES2 Command Preprocessor

FUNCTION:

This exit allows you to provide an exit routine to preprocess JES2 commands received by the JES2 command processor. If this exit is implemented and enabled, it receives control every time HASPCOMM receives a JES2 command.

You can use your exit routine to perform your own command validation and, based on the checking performed by your validation algorithm, decide whether JES2 should terminate processing for the command or allow normal HASPCOMM processing to continue. If you use your exit routine to terminate processing for a command, the command subprocessor is bypassed and the requested action is not taken.

This exit also enables you to implement your own installation-specific JES2 command operands and suboperands, as well as nonstandard JES2 commands unique to your installation. Your exit routine must process nonstandard, installation-specific operands, suboperands, and commands itself, and then suppress standard HASPCOMM processing. Nonstandard command processing is considered in greater detail in the "Other Programming Considerations" below.

When suppressing standard HASPCOMM processing, you have the option of directing JES2 to send the standard "OK" return message to the operator, sending your own exit-generated message to the operator, or of suppressing standard HASPCOMM processing without operator notification.

This exit is effected by a single exit point, at label COMMEXIT.

ENVIRONMENT: JES2 main task

POINT OF PROCESSING:

This exit is taken from the JES2 main task, from the HASPCOME command edit routine of HASPCOMM. The exit point occurs after the command has been edited but before lookup in the command selection tables (COMFASTR and COMTAB), before console authority checking, and before the call to the command subprocessor.

REGISTER CONTENTS WHEN CONTROL IS PASSED TO THE EXIT ROUTINE:

R0-R4	N/A
R5	Pointer to the address of the current operand[1]
R6	Increment value of 4[1]
R7	Pointer to the address of the last operand[1]
R8-R10	N/A
R11	Address of the HCT
R12	N/A
R13	Address of the HASPCOMM PCE
R14	Return address
R15	Entry address

REGISTER CONTENTS WHEN CONTROL IS PASSED BACK TO JES2:

R0 If an exit-generated message is to be passed, this register contains the length of the message; otherwise, it's not applicable.

R1 N/A

R15 A return code

RETURN CODES:

0 Tells JES2 that if there are any additional exit routines associated with this exit, execute the next consecutive exit routine. If there are no other exit routines associated with this exit, continue with normal command processing.

4 Tells JES2 to ignore any other exit routines associated with this exit point and to continue with normal command processing.

8 Tells JES2 to terminate standard processing for the command and to issue the $CRET macro to return control to the main command processor; the command subprocessors are bypassed.

12 Tells JES2 to terminate standard processing for the command and to issue the $CRET macro, specifying the standard $HASP000 "OK" message, to return control to the main command processor. The "OK" message is issued and the command subprocessors are bypassed.

16 Tells JES2 to terminate standard processing for the command and to issue the $CRET macro, specifying a message generated by your exit routine, to return control to the main command processor. The exit-generated message is issued and the command subprocessors are bypassed.

JOB EXIT MASK: This exit is not subject to job exit mask suppression.

MAPPING MACROS NORMALLY REQUIRED FOR THIS EXIT AT ASSEMBLY: $HASPEQU, $HCT, $MIT, $PCE, $COMWORK

OTHER PROGRAMMING CONSIDERATIONS:

1. For a multiple command, this exit is taken once for each command verb.

2. To preprocess a standard JES2 command, a typical exit routine would perform some type of validation checking. This validation checking would determine whether JES2 should terminate command processing or allow standard command processing to continue. You can base a validation algorithm on various factors. The fields of the command processor work area

---

[1] Refer to "Other Programming Considerations" below for use of these registers in a networking environment.

of the PCE contain extensive command-related information that can be used in validation checking. Note, however, that even if your exit routine validates a command, it is still possible for JES2 to reject the command as a result of its standard validation checking.

3. In processing your own installation-specific JES2 commands, your exit routine should perform its own validation checking to replace the functions normally performed by HASPCOME. Your routine should validate the command verb, contained in the COMVERB field of the PCE's command processor work area, with the equivalent of the command table lookup performed by HASPCOME. This check should determine whether the command has a valid installation-specific command verb and what action your exit routine should take based on the verb. Your routine should also perform console authority checking by testing the COMAUTH field, of the PCE's command processor
work area, which contains the command's restriction bits. COMAUTH has the following structure:

   COMS (X'01') when on indicates that the command should be rejected unless authorized for the system.

   COMD (X'02') when on indicates that the command should be rejected unless authorized for the device.

   COMJ (X'04') when on indicates that the command should be rejected unless authorized for the job.

   COMR (X'08') when on indicates that the command should be rejected if it was entered from a remote work station.

   If your routine validates the command, it can then perform the requested function, serving as the equivalent to a standard command subprocessor. If, however, your routine determines that the command is invalid, it must terminate processing for the command internally before returning control to JES2. Then, it should pass a return code (of 8, 12, or 16) to terminate standard HASPCOMM processing, with or without an accompanying message to the operator.

4. In general, to process nonstandard operands and suboperands, you must write your exit routine to replace standard JES2 processing entirely. That is, your exit routine must process both the nonstandard operands or suboperands as well as the standard portion of the command, by performing the function of the standard command subprocessor. This is because, in general, the command verb and the accompanying operands and suboperands are interdependent; the operands and suboperands modify the action of the command verb and cannot be processed independently.

5. When passing a return code of 16 and issuing an exit-generated message to the operator, move the text of the message to the COMMAND field of the command processor work area in the PCE. Place the length of the message in R0. Also, be certain to issue the $STORE (R0) macro after loading the message length in R0 but prior to issuing the $RETURN macro because $RETURN macro destroys the contents on register 0. (When passing a return code of 12, to cause JES2 to issue the standard "OK" return message, you do not have to supply the message length in R0.)

6. *Note on recovery:* $ESTAE recovery is *not* in effect while an exit routine associated with this exit is being processed. However, you can implement $ESTAE recovery within your routine. As with all exits, you are responsible for your own recovery within your exit routine, whether you choose to implement $ESTAE recovery or other recovery procedures.

7. Use the $CWTO macro instruction in this exit to communicate to the operator. If you use the $CWTO macro, you must do all the processing required by the specified command within your exit routine and provide a return code indicating that JES2 should bypass any further processing of the specified command.

8. When this exit routine operates in a networking environment, your exit must check the contents of the COMINCON field of the PCE pointed to by register 13. If the X'80' bit is on, the current command is in subsystem interface (SSI) format, and registers 5, 6, and 7 do not contain pertinent information. The SSI format command is located at label COSICMDA in this PCE; the mapping is located in HASPDOC.

9. L=cca processing takes place outside Exit 5. Therefore, if you require this function, you must provide your own L=cca processing for user commands.

Exit 6: Internal Text Scan

FUNCTION:

This exit allows you to provide an exit routine for scanning internal text. If this exit is implemented and enabled, it is taken once after each internal text statement has been converted from a JCL statement and once after all of the JCL for a particular job has been converted to internal text.

You can use your exit routine to interpret an internal text image and, on the basis of this interpretation, decide whether JES2 should either cancel the job or allow it to continue with normal execution. Your routine can also modify any internal text image. In addition, after all of the JCL for a particular job has been converted to internal text, this exit again allows you to direct JES2 either to cancel the job or to allow it to continue with normal execution.

Two exit points effect this exit. The first, at label XCSTCUEE, makes it possible for your routine to scan individual internal text images. The second, at label XCSTMUEE, makes it possible for your routine to determine whether or not a job is to continue after all of its JCL has been converted to internal text.

JCL internal text is represented by 'keys' that identify the various JCL parameters. These keys are documented in the JES2 assembly, HASPDOC, which calls macros IEFVKEYS and IEFTXTFT, which are distributed in SYS1.AMODGEN. IEFVKEYS contains the definition of the values for each key, and IEFTXTFT contains the definition of the format of the internal text.

ENVIRONMENT: JES2 subtask

Note: Refer to Appendix D, "JES2 Exit Usage Limitations" for a listing of specific instances when this exit will be invoked or not invoked.

POINT OF PROCESSING:

This exit is taken from HOSCNVT, the JCL conversion processor subtask, from within HASPCNVT.

The first exit point, at label XCSTCUEE, occurs after the OS/VS converter has converted a single JCL statement into its internal text image. The XTXTEXIT routine, which performs standard modifications to internal text images created from certain EXEC statements and from subsystem data set DD statements, has executed. (Note: XTXTEXIT is a standard JES2 exit routine, not to be confused with an installation exit.) All of the standard modifications that JES2 will make to the internal text image are complete when the exit receives control.

The second exit point, at label XCSTMUEE, occurs after all of the JCL for a particular job has been converted to internal text. It occurs at the return from the link to the converter, at label XCNVCNV, and before JES2 creates the scheduler work area (SWA) control blocks.

REGISTER CONTENTS WHEN CONTROL IS PASSED TO THE EXIT ROUTINE:

R0 A code indicating the status of conversion processing

- **0** Indicates that a JCL statement has been converted to an internal text image
- **4** Indicates that all of the JCL for a particular job has been converted to internal text

R1 Address of a 4-word parameter list

- **Word 1 (+0)** address of a 16-byte work area available to the installation.
- **Word 2 (+4)** if the code passed in R0 is 0, this word points to the address of the last single internal text image converted from a JCL statement. If the code passed in R0 is 4, this word contains the address of the converter's return code.
- **Word 3 (+8)** Address of the DTE
- **Word 4 (+12)** Address of the JCT

R2-R10 N/A

R11 Address of the HCT

R12 N/A

R13 Address of an 18-word OS-style save area

R14 Return address

R15 Entry address

REGISTER CONTENTS WHEN CONTROL IS PASSED BACK TO JES2:

R0 A code indicating the status of conversion processing

- **0** Indicates that a JCL statement has been converted to an internal text image
- **4** Indicates that all of the JCL for a particular job has been converted to internal text

R1 Address of a 4-word parameter list

- **Word 1 (+0)** address of a 16-byte work area available to the installation.
- **Word 2 (+4)** if the code passed in R0 is 0, this word points to the address of the last single internal text image converted from a JCL statement. If the code passed in R0 is 4, this word points to the address of the converter's return code.
- **Word 3 (+8)** Address of the DTE
- **Word 4 (+12)** Address of the JCT

R2-R10 N/A

R11 Address of the HCT

R12 N/A

R13 Address of an 18-word OS-style save area

R14 A return address

R15 A return code

RETURN CODES:

0 Tells JES2 that if there are any additional exit routines associated with this exit, execute the next consecutive exit routine. If there are no more exit routines associated with this exit point, continue with normal JES2 processing. If the exit routine was called from exit point XCSTCUEE, normal processing is the conversion of the next JCL statement to an internal text image. If the exit routine was called from exit point XCSTMUEE, normal processing is to queue the job for execution.

4 Tells JES2 to ignore any additional exit routines associated with this exit for this (internal text) statement and continue with normal processing. If the exit routine was called from exit point XCSTCUEE, normal JES2 processing is the conversion of the next JCL statement to an internal text image. If the exit routine was called from exit point XCSTMUEE, normal JES2 processing is to queue the job for execution.

8 Tells JES2 to bypass execution and cancel the job; the job is queued for output rather than for execution.

JOB EXIT MASK: This exit point is subject to job exit mask suppression.

MAPPING MACROS NORMALLY REQUIRED FOR THIS EXIT AT ASSEMBLY: $HASPEQU, $HCT, $MIT, $XIT, TEXT, KEYS

OTHER PROGRAMMING CONSIDERATIONS:

1. When internal text is created, the user's JCL has been merged with the expanded JCL from PROCLIB, and all substitutions for symbolic parameters have been made. Therefore, this exit gives you the opportunity to scan and modify the completely-resolved JCL to be used by a particular job.

2. Use extreme caution in modifying internal text. If any of your modifications cause a job to fail (because of an interpreter error), there will be no correlation of the error with the resulting abend on the user's output. In fact, because the output JCL images data set is created prior to the conversion of the JCL to internal text, none of the modifications you make in this exit will ever appear in your output. If you suspect that an exit routine associated with this exit is causing a problem, the most expedient method of debugging is to disable the exit to determine whether the problem still occurs when your exit routine is not executed. Then, if the problem seems to be within your exit routine, you can test the routine by turning on the tracing facility. The trace record serves as a valuable debugging aid because it contains two copies of each internal text image, one before the call to your exit routine and one after the call to your exit routine. However, **do not** turn on tracing in your normal production environment or you will seriously degrade the performance of your system.

3. Note that your routine must take into account the code passed to it by JES2 in R0. Your routine can modify internal text only when this code is 0; any modifications must be made to a single internal text image immediately after its conversion. When the code in R0 is 0, you can also extract information from an internal text image for use by your routine. For instance, your routine can use parameters from an internal text image to set fields in a control block written at your installation. When this code is 4, your routine can determine whether or not the job should be allowed to continue; however, no internal text modifications can be made and no scanning can be performed at this time.

One possibility is to code the first part of your routine, the part that receives control when the code in R0 is 0, to keep certain counters--for instance, the number of DD cards received. Then, when the JCL for the entire job has been processed, the second part of your routine, the part that receives control when the code in R0 is 4, can determine whether or not to allow the job to continue based on the contents of these counters.

4. If you decide to cancel the job, your routine is responsible for issuing any error messages to the operator and to the user.

5. JES2 does not create the scheduler work area (SWA) control blocks until all the JCL for a particular job has been converted to internal text. JES2 sets certain fields in these control blocks from corresponding internal text parameters. Therefore, by using this exit to alter internal text images, you can affect the contents of the SWA control blocks.

6. A 16-byte work area available to the installation, whose address is given as the first word of the 2-word parameter list pointed to by R1 on entry, is available for use by your exit routine.

7. The MVS converter provides an internal text buffer that is 8192 decimal (2000 hex) bytes long.

8. *Note on recovery:* No recovery is in effect when this exit is taken. As with every exit, you should provide your own recovery within your exit routine.

Exit 7: JCT Read/Write (JES2)

FUNCTION:

This exit allows you to provide an exit routine to receive control whenever JCT I/O is performed by the JES2 main task. That is, if this exit is enabled, your routine receives control before the JCT is written out to spool and after the JCT is read into storage. (Note: Whenever JCT I/O is performed by a JES2 subtask or by a routine running in the user address space, HASPSSSM, Exit 8 provides the function of this exit; in the HASPFSSM address space, Exit 25 provides this function.)

You can use this exit to perform I/O for any installation-specific control blocks you may have created.

This exit is established by an exit point at label JIOEXIT in HASPNUC.

ENVIRONMENT: JES2 main task

POINT OF PROCESSING:

This exit is taken from the JES2 main task in the HASPNUC module. Exit point JIOEXIT occurs in the $JCTIOR routine (HASPNUC), just after the JCT is read from or just before the JCT is written out to spool.

REGISTER CONTENTS WHEN CONTROL IS PASSED TO THE EXIT ROUTINE:

R0	A code passed to your routine by JES2 **0** Indicates that the JCT has been read from spool **4** Indicates that the JCT will be written to spool
R1	Address of the buffer that contains the JCT
R2-R10	N/A
R11	Address of the HCT
R12	N/A
R13	Address of the PCE
R14	The return address
R15	The entry address

REGISTER CONTENTS WHEN CONTROL IS PASSED BACK TO JES2:

R0-R1	N/A
R15	A return code

RETURN CODES:

0	Tells JES2 that if there are any additional exit routines associated with this exit, call the next consecutive exit routine. If there are no other exit routines associated with this exit, continue with normal processing, which is determined by the particular exit point from which the exit was called.
4	Tells JES2 that even if there are additional exit routines associated with this exit, ignore them; continue with normal processing, which is determined by the particular exit point from which the exit routine was called.

JOB EXIT MASK: This exit point is subject to job exit mask suppression.

MAPPING MACROS NORMALLY REQUIRED FOR THIS EXIT AT ASSEMBLY: $HASPEQU, $JCT, $MIT, $PCE

OTHER PROGRAMMING CONSIDERATIONS:

1. For JCT I/O, the JES2 input/output processor, $EXCP, sets the following condition codes:

   1 Indicates that the I/O operation was unsuccessful
   3 Indicates that the I/O operation was successful

2. You can use this exit to determine the queue on which a job resides at any point of processing at which JCT I/O is performed for the JES2 main task. First, your exit routine must use the pointer in the JCTJQE field of the JCT to locate the JQE. Then, after accessing the JQE, your routine must locate the JQETYPE field. JQETYPE can then be tested to determine on which queue, out of nine possible queues, the current job resides. The following table lists the nine possible queues along with their corresponding hexadecimal representations in JQETYPE:

$XEQ	X'40'
$INPUT	X'20'
$XMIT	X'10'
$RECEIVE	X'04'
$OUTPUT	X'02'
$HARDCOPY	X'01'
$PURGE	X'00'
$FREE	X'FF'

3. Use the PCEID field to determine which processor is reading or writing the JCT; this avoids unnecessary processing.

4. *Note on recovery:* No recovery is in effect when this exit is taken. As with every exit, you should provide your own recovery within your exit routine.

Exit 8: JCT Read/Write (USER)

FUNCTION:

This exit allows you to provide an exit routine to receive control whenever a JES2 subtask or a routine running in the user address space (HASPSSSM) performs JCT I/O. That is, your routine receives control just before the JCT is written out to spool and just after the JCT is read into storage. (Note: Whenever JCT I/O is performed by the JES2 main task, Exit 7 serves the purpose of this exit.)

You can use this exit to perform I/O for any installation-specific control blocks you may have created.

Four exit points effect this exit. The first, at label HXJCT01, can give control to your exit routine just after JES2 reads the JCT into storage for a job requested by the MVS initiator and selected by HASPXEQ. The second exit point, at label HXJCT02, can give control to your exit routine just before JES2 writes the JCT to spool for a job whose termination has been requested by the MVS initiator. The third exit point, at label HXJCT03, can give control to your exit routine just before JES2 writes the JCT to spool after the control block checkpoint routines in HASPSSSM have processed it. The fourth exit point, at label HXJCT04, can give control to your exit routine at SYSOUT data set allocation, if JES2 had to alter the JCT, just before JES2 writes the JCT out to spool.

ENVIRONMENT: User address space

POINT OF PROCESSING:

This exit is taken from the user address space (HASPSSSM).

Exit point HXJCT01, for receiving control after the JCT has been read into storage, occurs in the SSI HOSJBSL job selection routine, after HOSJBSL has reacquired control from the execution processor (HASPXEQ), has verified that a job has been found for execution, and has successfully read in the JCT.

Exit point HXJCT02, for receiving control before the JCT is to be written to spool, occurs in the SSI HOSTERM job termination routine, after HOSTERM reacquires control from the execution processor (HASPXEQ). The exact point of processing may vary slightly within HOSTERM, depending on the functioning HOSTERM performs for a particular caller.

Exit point HXJCT03, for receiving control before the JCT is to be written to spool, occurs in the control block checkpoint routine HCBCK in HASPSSSM. The checkpointing can occur at various points in HASPSSSM processing, whenever the control block checkpoint routine is called. The JCT is checkpointed only if it has been changed.

Exit point HXJCT04, for receiving control before the JCT is to be written to spool, occurs in the subroutine for allocating SYSOUT data sets (HALO) of the HOSALLOC allocation routine. It is taken only if the HALCRDSN subroutine of HASALLOC had to update the JCT, making it necessary for HALO to checkpoint the JCT after completing SYSOUT data set allocation.

REGISTER CONTENTS WHEN CONTROL IS PASSED TO THE EXIT ROUTINE:

R0	A code passed to your routine by JES2 **0** Indicates that the JCT has been read from spool **4** Indicates that the JCT will be written to spool
R1	Address of the JCT
R2-R10	N/A
R11	Address of the SVT
R12	N/A
R13	Address of an OS-style save area
R14	Return address
R15	Entry address

REGISTER CONTENTS WHEN CONTROL IS PASSED BACK TO JES2:

R0-R1	N/A
R15	A return code

RETURN CODES:

0 Tells JES2 that if there are any additional exit routines associated with this exit, call the next consecutive exit routine. If there are no other exit routines associated with this exit, continue with normal processing, which is determined by the particular exit point from which the exit routine was called.

4 Tells JES2 that even if there are additional exit routines associated with this exit, ignore them; continue with normal processing, which is determined by the particular exit point from which the exit routine was called.

JOB EXIT MASK: This exit is subject to job exit mask suppression.

MAPPING MACROS NORMALLY REQUIRED FOR THIS EXIT AT ASSEMBLY: $HASPEQU, $JCT, $MIT

OTHER PROGRAMMING CONSIDERATIONS:

1. Be sure your exit routines reside in common storage.

2. *Note on recovery:* No recovery is in effect when this exit is taken. As with every exit, you should provide your own recovery within your exit routine.

Exit 9: Job Output Overflow

FUNCTION:

This exit allows you to provide an exit routine to receive control when JES2 detects a job output overflow error. If this exit is implemented and enabled, it is invoked whenever the current count of output records, pages, or bytes exceeds the output estimate specified for the current job on the JES2 /*JOBPARM control statement or through JES2 initialization defaults for lines (ESTLNCT), cards (ESTPUN), pages (ESTPAGE), or bytes (ESTBYTE) if the /*JOBPARM control statement is not used. (Note: This exit is not taken for job OUTLIM overflow, for which a standard SMF exit already exists.)

When your exit routine receives control, you have several defined options from which to choose in directing JES2 to handle the job output overflow for the current job. You can direct JES2 to take action based on the standard value specified on the excessive estimate output option given on the ESTLNCT, ESTPAGE, ESTPUN, and ESTBYTE statements. Be aware, when blank truncation is turned off, estimated lines (ESTLNCT) and pages (ESTPAGE) will not change; only estimated bytes (ESTBYTE) will reflect the change. Therefore, be certain to consider the value specified for ESTBYTE because the byte count will increase if truncation is turned off. You can also direct JES2 to base its action, for the current job only, on the "excessive output option" value supplied by your exit routine. If the "excessive output option" is specified as 0, the job is allowed to continue. If the "excessive output option" is specified as 1, the job is canceled without a dump. If the "excessive output option" is specified as 2, the job is canceled with a dump. For more information, see the description of the the "excessive output option" statement in *JES2 Initialization and Tuning*.

In addition, you can use your exit routine to control the default error message ($HASP375) that JES2 normally sends to the operator when a job has exceeded its output estimate. You can suppress the standard error message or you can allow JES2 to send it. You can also control the interval, in excess print lines or in excess punch cards, at which the message is reissued to the operator. You can allow the message to be sent at the interval specified at your installation for the "message interval for exceeding estimated output" initialization statement, or you can direct JES2 to send the message at the interval specified by your exit routine for the current job. For more information, see *JES2 Initialization and Tuning*.

This exit is effected by a single exit point SVCOUTX in routine HEXTCALL.

ENVIRONMENT: User address space

POINT OF PROCESSING:

This exit is taken from the user address space (HASPSSSM), from the SVCHAM portion (SVC111) of the HASP access method (HASPAM). The exit is taken during SVC 111 put processing, whenever the current output record count exceeds the output estimate.

REGISTER CONTENTS WHEN CONTROL IS PASSED TO THE EXIT ROUTINE:

R0 N/A

R1 Address of the 7-word parameter list, having the following structure:
+0 JCTLINES or JCTPUNCH value (in actual lines)
+4 JCTPAGES value
+8 JCTBYTES value
+12 User's increment for records
+16 User's increment for pages
+20 User's increment for bytes
+24 Output Overflow Flag

A fullword with the flag in the high-order byte. The flag settings are:

bit 0	= 0	(Cards have not exceeded the limit)
	= 1	(Cards have exceeded the estimate)
bit 1	= 0	(Lines have not exceeded the limit)
	= 1	(Lines have exceeded the estimate)
bit 2	= 0	(Pages have not exceeded the limit)
	= 1	(Pages have exceeded the estimate)
bit 3	= 0	(Bytes have not exceeded the limit)
	= 1	(Bytes have exceeded the estimate)
bits 4-31		Not Applicable

R2-R6 N/A
R7 Address of the JCT
R8-R10 N/A
R11 Address of the SVT
R12 N/A
R13 OS-style 18-word save area
R14 Return address
R15 Entry address

REGISTER CONTENTS WHEN CONTROL IS PASSED BACK TO JES2:

R0 If the return code (passed in R15) is not 8, the contents of this register are ignored. However, if the return code is 8, this register contains return processing flags, as follows:

bit 0	= 0	Tells JES2 to take action based on the value currently specified for the "excessive output option" at your installation.
	= 1	Tells JES2 to take action based on the value specified for the "excessive output option" in bits 24-31 of this register.
bit 1	= 0	Tells JES2 to use the output overflow increment specified on the INT= parameter of the ESTLNCT, ESTPUN, ESTPAGE, or ESTBYTE initialization statements
	= 1	Tells JES2 to use the output overflow increment supplied by the exit routine in the parameter list.
bit 2	= 0	Tells JES2 to send the default error message ($HASP375) to the operator.
	= 1	Tells JES2 to suppress the default error message.
bits 3-23		N/A
bits 24-31		Tells JES2 the execution option for this exit call. If this byte is zero then continue processing. If this byte is equal to 1 then ABEND (722) without a dump. If this byte is equal to 2 then ABEND (722) with a dump.

R15 A return code.

RETURN CODES:

0 Tells JES2 that, if there are any additional exit routines associated with this exit, it is to execute the next consecutive exit routine. If there are no additional exit routines, JES2 is to perform standard job output overflow processing based on the INT= and OPT= parameters on the ESTLNCT, ESTPUN, ESTPAGE, and ESTBYTE initialization statements.

4 Tells JES2 to ignore any additional exit routines associated with this exit and to perform standard job output overflow processing based on the values supplied for the ESTLNCT, ESTPUN, ESTPAGE, and ESTBYTE initialization statements at your installation.

8 Tells JES2 to take action based on the return processing flags in R0; see the description of the contents of R0 above.

JOB EXIT MASK: This exit is subject to job exit mask suppression.

MAPPING MACROS NORMALLY REQUIRED FOR THIS EXIT AT ASSEMBLY: $HASPEQU, $JCT, $MIT, $SVT

OTHER PROGRAMMING CONSIDERATIONS:

1. If the job output overflow error occurs when output is being produced for the JES2 job log, for the JES2 messages file, or for the JCL images file, the exit is taken but the return codes and return processing flags are ignored.

2. This exit is not taken for job OUTLIM overflow. For more information on standard OUTLIM processing, see the SMF IEFUSO exit in *System Management Facilities (SMF)*.

3. Be sure your exit routines reside the link pack area (LPA).

4. *Note on recovery:* There is no JES2 recovery in effect. As with every exit, you should supply your own recovery within your exit routine.

Exit 10: $WTO Screen

FUNCTION:

This exit allows you to provide an exit routine to receive control every time that JES2 is ready to queue a $WTO message for transmission. If this exit is implemented and enabled, it receives control for all messages destined for remote stations and for other systems, as well as for all messages with a destination of local.

However, this exit does **not** receive control for messages generated by HASPSSSM or HASPFSSM, the subsystem interface and functional subsystem modules.

You can use your exit routine to interrogate the message's console message buffer (CMB) and, on the basis of this interrogation, direct JES2 either to cancel the message or to queue it for normal transmission. You can also use your exit routine to change the text of the message or to alter its console routing.

This exit is effected by a single exit point, at label WTOEXIT.

ENVIRONMENT: JES2 main task

POINT OF PROCESSING:

This exit is taken from the JES2 main task, from the HASPWQUE (special purpose CMB queueing) routine of the HASPCON (console support services) module, for all JES2 main task $WTO messages. The exit occurs at the beginning of HASPWQUE, after the $WTOR routine has processed the $WTO macro and before HASPWQUE queues the CMB containing the message for transmission. If, by passing a return code of 0 or 4, your routine allows the message to continue, control returns to HASPWQUE, which then queues the message for transmission. If, however, your exit routine cancels the message by passing a return code of 8, the transmission queueing performed by HASPWQUE is bypassed and JES2 gives control to $FRECMBR, the $FRECMB service routine.

REGISTER CONTENTS WHEN CONTROL IS PASSED TO THE EXIT ROUTINE:

R0	N/A
R1	Address of the CMB
R2-R10	N/A
R11	Address of the HCT
R12	N/A
R13	Address of the PCE
R14	Return address
R15	Entry address

REGISTER CONTENTS WHEN CONTROL IS PASSED BACK TO JES2:

R0	N/A
R1	Address of the CMB
R15	A return code

RETURN CODES:

0 Tells JES2 that if there are any more exit routines associated with this exit, execute the next consecutive exit routine. If there are no more exit routines associated with this exit, continue with normal processing by queueing the CMB for transmission.

4 Tells JES2 to ignore any additional exit routines associated with this exit and to continue with normal processing by queueing the CMB for transmission.

8 Tells JES2 to discard the message by freeing the CMB; the message is not queued for transmission.

JOB EXIT MASK: This exit is not subject to job exit mask suppression.

MAPPING MACROS NORMALLY REQUIRED FOR THIS EXIT AT ASSEMBLY: $CMB, $HASPEQU, $HCT, $MIT, $PCE

OTHER PROGRAMMING CONSIDERATIONS:

1. This exit is taken only for $WTOs issued from the JES2 main task. This exit will not be taken for $WTOs issued from HASPSSSM, HASPFSSM, JES2 subtask, or any other $WTOs issued from outside the JES2 main task environment.

2. To cancel a message, pass a return code of 8 to JES2. This return code directs JES2 to bypass the HASPWQUE routine, which normally queues the CMB for the console service processor, and to give control directly to the $FRECMBR routine, which then discards the message by freeing its CMB.

3. To change the text of a message, your routine must access either the CMBTEXT field or the CMBJOBN field. If the message does not contain the job's name and number, the message text starts in CMBJOBN. The length of the message is always in the CMBML field. Your routine can either retrieve the existing message text and modify it or else generate a completely new message and then write the new or modified message over the original message. If the new or modified message is longer or shorter than the original message, your routine should alter the CMBML field accordingly. After altering the text of the message, pass a return code of 0 or 4 to direct JES2 to queue the CMB for transmission. JES2 will then transmit the new or modified message.

4. To alter a message's console routing, your routine should first test the flag byte CMBFLAG to determine whether the CMBFLAGW, CMBFLAGT, and CMBFLAGU flags are off. If these three flags are off, the CMBROUT field contains the MVS console routings. After altering CMBROUT, pass a return code of 0 or 4 to direct JES2 to queue the CMB for transmission. JES2 will base its console routing on the new contents of CMBROUT.

5. If MESSAGE ID=NO, this exit still sees the message (as specified on the MSGID parameter of the CONDEF statement).

6. *Note on recovery:* There is no JES2 recovery in effect. As with every exit, you should supply your own recovery within your exit routine.

Exit 11: Spool Partitioning Allocation ($TRACK)

FUNCTION:

This exit allows you to provide an exit routine to receive control from the JES2 main task when no more track groups are available on the spool volumes from which the current job is permitted to allocate space. (Note: In the JES2 subtask and user environments, Exit 12 serves the purpose of this exit. If you intend to use this exit, you should use Exit 12 as well. The description of Exit 12 presents the special considerations that apply to it.)

Standard JES2 processing allows all jobs to allocate track groups from all available spool volumes. However, using the exit facility and/or the FENCE parameter on the SPOOLDEF initialization statement, you can implement spool partitioning--that is, you can limit the spool volumes from which a particular job is permitted to allocate track groups. If you require allocation to a minimum number of spool volumes for a job, the JES2 FENCE parameter can be used. If you intend to implement more complicated spool partitioning (for example, by specific spool volume) you will need to code an Exit 11 routine.

To implement spool partitioning, using Exits 11 and 12, the 32-byte spool partitioning mask (the JCTSAMSK) can be modified as specified by these exit routines. The first $SPOLNUM bits in the JCTSAMSK correspond to the DASDs. Those bits, when turned on in the JCTSAMSK, indicate the volumes from which a job may allocate space. The exact correspondence between bits and spool volumes is described in "Other Programming Considerations" below. A job is permitted to allocate track groups from those volumes whose corresponding bits in the spool partitioning work area are set to one.

Initially, when a job is started, JCTSAMSK is set to all zeros. If you haven't implemented spool partitioning and have specified the FENCE parameter as FENCE=NO, JES2 automatically sets JCTSAMSK to all ones the first time that it issues the $TRACK macro (from HASPRDR). When JCTSAMSK is set to all ones, the current job can allocate track groups from all available spool volumes and the existence of JCTSAMSK is transparent; standard JES2 track group allocation is in effect. However, as a replacement for the allocation default, you can provide your own exit routine to expand the spool partitioning mask to include only those particular spool volumes from which you intend to allow the current job to allocate space. You set the bits corresponding to the designated spool volumes to one, and you allow the bits corresponding to the other spool volumes to remain zeros. Note, the exit routine sets bits in a spools-allowed mask work area that has been previously initialized to the IOTSAMSK field. JES2 then updates the JCTSAMSK to reflect your changes. Your routine *must* turn on at least one bit in the mask work area. If it does not, and the FENCE parameter is set to NO, JES2 responds by providing the default, setting JCTSAMSK to all ones and allowing the job to allocate track groups from all available spool volumes.

As an alternative to setting the partitioning mask for the first time from this exit, you can set the partitioning mask when the JCT is initialized, from Exit 2 (Job Statement Scan). The relative merits of setting the mask for the first time from Exit 2 or this exit are discussed in "Other Programming Considerations" below. Note that Exit 3 cannot be used to set the spool partitioning mask because a

$TRACK has already occurred prior to calling this exit. If you want to provide spool partitioning for most jobs, such that they will only allocate space from a minimum number of spool volumes, use the FENCE parameter; specify FENCE=YES. When FENCE=YES is specified, the JCTSAMSK is not set to all ones (the default), rather only one bit is set on, and jobs are assigned evenly across all spool volumes.

Once the spool partitioning mask has been set, whether from this exit or from another exit, this exit--if it has been implemented--receives control only when no more track groups are available on the spool volumes designated for the job in its mask. If only one volume has been designated initially for a particular job but the job never requires more space than happens to remain available on its single designated volume, the spool partitioning mask never has to be expanded and the exit is not invoked. Again, when and if your routine does receive control, it *must* allocate at least one additional spool volume. And, again, if it does not expand the mask at all, JES2 responds by expanding JCTSAMSK to all ones if the FENCE parameter is specified as FENCE=NO or adding one more volume to JCTSAMSK if FENCE=YES.

This exit is effected by a single exit point, at label $TRACKX.

ENVIRONMENT: JES2 main task

POINT OF PROCESSING:

This exit is taken from the JES2 main task, from the $TRACK subroutine in HASPTRAK, when there is no space available on the spool volumes from which the current job is permitted to allocate space. If the job is permitted to allocate space from any spool volume, that is, if the spool partitioning mask is set to all ones, this exit is not invoked; it is only invoked when spool partitioning is in effect.

When $TRACK is called, it first checks the TGB for an available track group from an eligible spool volume. If a track group is available, $TRACK allocates it to the job. If there is no available track group on an eligible spool volume represented in the current TGB, $TRACK checks the bit map in SVTMTSPL to determine whether there are any track groups not represented in the current TGB that are available on any of the eligible spool volumes. If there are additional track groups available, $TRACK posts ($POST) the checkpoint processor to replenish the TGB via the KBLOB routine. However, if there are no more track groups available on any of the eligible spool volumes, $TRACK must determine whether spool partitioning is in effect. If no spool partitioning is in effect, that is, if the mask is set to all ones, then $TRACK waits ($WAIT) for the JES2 dispatcher to free at least one of the track groups currently in use. If spool partitioning is in effect, $TRACK invokes the exit to expand the copy of the spool partitioning mask. When the exit routine returns control, $TRACK checks to see if the mask has been expanded. If not, $TRACK expands the spool partitioning mask to all ones, if FENCE=NO or adds only one bit to the mask if FENCE=YES. If the mask has been expanded, $TRACK places the spool partitioning work area (that is, the updated version of the spool partitioning mask) in the JCTSAMSK. $TRACK then restarts its allocation cycle by checking the new (replenished) TGB for an available track group.

Note that this exit is not invoked when there is still space available on the job's designated spool volumes but the particular track groups from the designated spool volumes represented in the TGB have been allocated. That is, the exit is not invoked merely when the TGB has to be replenished before additional track groups can be allocated to the job from its designated spool volumes; instead, JES2 automatically replenishes the TGB, prior to and transparent to this exit. This distinction is described in greater detail in the "Other Programming Considerations" provided below.

REGISTER CONTENTS WHEN CONTROL IS PASSED TO THE EXIT ROUTINE:

R0 N/A
R1 Address of the 3-word parameter list, having the following structure:

word 1 (+0) Address of the IOT

word 2 (+4) Address of the JCT (if available); otherwise 0

word 3 (+8) Address of a 32-byte spool partitioning mask work area, initially set to the IOTSAMSK field

R2-R10 N/A
R11 Address of the HCT
R12 N/A
R13 Address of the PCE
R14 Return address
R15 Entry address

REGISTER CONTENTS WHEN CONTROL IS PASSED BACK TO JES2:

R0-R14 Unchanged

R15 A return code

RETURN CODES:

0 Tells JES2 that if there are any additional exit routines associated with this exit, execute the next consecutive exit routine. If there are no additional exit routines associated with this exit point, this return code tells JES2 to set the spool partitioning mask as indicated by the FENCE parameter on the SPOOLDEF initialization statement setting and to reissue the $TRACK request.

4 Tells JES2 that even if there are additional exit routines associated with this exit, ignore them; instead, set the spool partitioning mask as indicated by the FENCE parameter on the SPOOLDEF initialization statement setting and reissue the $TRACK request.

8 Tells JES2 that an updated version of the spool partitioning mask--with at least one additional bit turned on--has been passed to JES2 in the spool mask work area and will now determine subsequent spool allocation. It also tells JES2 to reissue the $TRACK request.

JOB EXIT MASK: This exit is subject to job exit mask suppression.

MAPPING MACROS NORMALLY REQUIRED FOR THIS EXIT AT ASSEMBLY:

MAPPING MACROS NORMALLY REQUIRED FOR THIS EXIT AT ASSEMBLY: $BUFFER, $DAS, $HASPEQU, $HCT, $IOT, $JCT, $MIT, $PCE, $SCAT, $SVT, $TAB, $XECB, RPL

OTHER PROGRAMMING CONSIDERATIONS:

1. When your exit routine receives control, it *must* expand the spool partitioning mask to include at least one additional spool volume. If your routine passes a return code of 8 to JES2 but hasn't actually expanded the mask via the new mask returned in the spool mask work area, JES2 automatically sets the mask to all ones or to one bit, as indicated by the FENCE parameter specification. You can also intentionally direct JES2 to expand the allocation mask by passing the standard return code of 0 or 4 from your exit routine.

2. You may implement spool partitioning from Exit 2 or from this exit. In general, you would set JCTSAMSK from Exit 2 if you intended to base your allocation algorithm on JOB statement parameters and from this exit if you intended to base your allocation algorithm on spool volume usage, for example, if you want to always allocate track groups from the least full spool volume. Note that the algorithms mentioned here are only suggestions; you may wish to use other criteria, or some combination of criteria, in implementing your own spool volume allocation algorithm.

   If you set the mask in Exit 2, this exit receives control only when it becomes necessary to expand the mask. It is also possible to set the mask in Exit 2 and yet not write a mask-expansion routine for this exit. In this case, if the spool volumes originally designated for the job runs out of space, JES2 provides the default and the job can then allocate track groups from any available spool volume. Or, you might choose to use this exit to both set the mask and then, if necessary, expand it. You have considerable flexibility in implementing spool partitioning and adapting it to the specific needs of your system.

3. If you intend to set JCTSAMSK from Exit 2, note that, under exceptional circumstances, JES2 may have to issue $TRACK before the entire JOB statement or accounting field has been scanned. For example, if the JOB statement is exceptionally long and spread out over a number of continuation statements, and if, at the same time, the JES2 buffer size is small, JES2 may have to call the $TRACK routine before finishing with the JOB statement. Under these circumstances, if you have written a mask expansion routine for this exit, it gets control prior to the exit routine which would have set JCTSAMSK from Exit 2.

4. If the IOTJCT field of the IOT contains the address of the JCT, the JCT is available. Otherwise, the IOTJCT field contains a 0 and the JCT is not available. For example, the JCT is unavailable when JES2 is acquiring space for the spooled remote messages or multi-access spool messages, and when JES2 is acquiring a record for the IOT for the JESNEWS data set. If you intend to base your allocation algorithm on values contained in fields of the JCT, you must take into account the fact that the JCT is sometimes unavailable and write a section of your exit routine to take control in these instances.

5. Note that, if implemented, this exit is *not* invoked every time the $TRACK routine is called. While track groups remain available on the spool volumes designated for the job by the spool volume partitioning mask, the exit is not invoked. It is invoked only when there is no space available on the spool

volumes designated for the job by the current setting of the spool partitioning mask. If all of the track groups from the designated spool volumes represented in the TGB have been allocated but the spool volumes themselves still contain unallocated track groups, this exit is *not* invoked. Instead, when a processor makes the $TRACK request, JES2 automatically determines that the TGB needs to be replenished. The executing processor then posts ($POST) the checkpoint processor, and issues the $WAIT macro. The checkpoint processor calls the KBLOB routine to replenish the TGB with unallocated track groups (from all available spool volumes, which include the job's designated spool volumes). When the TGB is replenished, the checkpoint processor $POSTs the JES2 main task processor that originally issued the $TRACK request. When this processor regains control, the $TRACK request can then be completed, using track groups from the job's designated spool volumes.

6. You *must* specify the TGBENUM parameter on the SPOOLDEF statement to specify the size of the TGB with a number large enough to accommodate all the spool volumes that could be available to JES2 at any one time. If, after you've implemented spool partitioning, the number you specify for this parameter is too low, results are unpredictable. Potentially, you could degrade JES2 performance.

   When replenishing the TGB with unallocated track groups, the standard KBLOB routine algorithm provides track groups equally from across all available spool volumes. That is, KBLOB provides a track group (or track group cell) from each available volume. If this does not fill the TGB, it again provides a track group from each available volume, repeating this process until the TGB is filled. The result--provided that the TGBENUM parameter on the SPOOLDEF statement has been specified correctly--is that all available spool volumes are represented by track groups in a single replenishment of the TGB. If at least one of the spool volumes from which the current job is allowed to allocate space is available, a track group from that volume is available to the job in the TGB. Otherwise, if none of the spool volumes designated in the job's spool partitioning mask are available, results depend on whether or not a mask-expansion routine has been implemented for this exit. If this exit is implemented and enabled, it is invoked to expand the mask, after which JES2 can call the $TRACK routine again. If this exit has not been implemented, JES2 responds by expanding the mask to all ones.

   However, in the case where the number specified for the TGBENUM parameter on the SPOOLDEF statement is too small, every available spool volume *cannot* be represented by at least one track group in a single replenishment of the TGB. For example, if the size of the TGB is specified as six track groups but there are eight available spool volumes, only six of the eight can be represented in any single replenishment of the TGB. The result is that if JES2 issues the $TRACK macro for a job that is limited by its spool partitioning mask to the use of a spool volume which is *not* represented in the current TGB, JES2 is put into a loop. The loop occurs although the spool volume was available. The exit routine, which is invoked to expand the mask when all of the volumes it represents are no longer available, is not invoked because at least one of the volumes it designates *is* still available. Therefore, because it cannot find a track group designated for the current job in the

current TGB, JES2 posts ($POST) the checkpoint processor to have it replenish the TGB. However, when the checkpoint processor takes control, it discovers that the TGB does not have to be replenished because none of its track groups have been allocated. The checkpoint processor then returns control to JES2, which reissues the $TRACK request and again, finding no track groups from the designated volumes for the current job available in the current TGB, posts ($POST) the checkpoint processor. This looping cycle continues until JES2 times out and fails. For this reason, if you intend to implement spool partitioning, it is essential that you specify TGBENUM correctly.

7. Spool partitioning, as implemented using the JES2 exit facility, is not fully effective in that you cannot absolutely limit a particular job to particular spool volumes. You cannot wait for space on an unavailable volume to become available. Once the volume or volumes to which you have limited a job are full, you must either expand the partitioning mask yourself in your exit routine or else allow JES2 to expand it to all ones. This fact, however, does not negate the usefulness of the spool partitioning that you can implement. In having the ability to limit a job to particular spool volumes *until* they become full, you can still exercise a considerable degree of control over volume usage, control that you would not have in any degree if you allowed your system to default to standard track group allocation.

8. *Note on recovery:* The $TRACK routine can be called from almost any stage in JES2 processing, with significant variations in standard JES2 recovery mechanisms from call to call. For example, HASPRDR provides more extensive recovery than HASPXEQ. Therefore, when $TRACK is called from HASPRDR, an error in your exit routine may cause only the current job to fail; however, when $TRACK is called from HASPXEQ, an error in your exit routine may cause JES2 itself to fail. As with every exit, your exit routine *should not* depend on JES2 for recovery. JES2 cannot anticipate the exact purpose of your exit routine, and therefore any standard JES2 recovery that happens to be in effect is, in general, minimal. You should provide your own recovery within your exit routine.

Exit 12: Spool Partitioning Allocation ($STRAK)

FUNCTION:

This exit allows you to provide an exit routine to receive control from a JES2 subtask or a routine executing in the user address space (HASPSSSM) when no more track groups are available on the spool volumes from which the current job is permitted to allocate space. (Note: In the JES2 main task, Exit 11 serves the purpose of this exit. Like Exit 11, this exit is used in implementing spool partitioning. For an initial description of spool partitioning and how to implement it in your system, you should read the description of Exit 11, above, prior to reading this one. You should not implement Exit 11 or Exit 12 without implementing its companion exit; therefore, this description concerns itself primarily with how Exit 12 relates to and differs from Exit 11, and the special considerations that apply to Exit 12.)

Unlike Exit 11, this exit is not invoked to set the spool partitioning mask for the first time. The mask is set in Exit 2 or in Exit 11. For any job, the $TRACK routine is always called to allocate track groups for the first time from HASPRDR or HASPNET, which are part of the JES2 main task. This exit is called from the $STRAK routine, which allocates track groups in the subtask and user environments, rather than $TRACK, which allocates track groups in the JES2 main task.

This exit, similar to Exit 11, if implemented, receives control only when no more track groups are available on the spool volumes designated for the job in its mask. And, just as with Exit 11, once your exit routine receives control it *must* allocate at least one additional spool volume. If it does not expand the mask at all, JES2 responds by expanding JCTSAMSK to all ones if the FENCE parameter on the SPOOLDEF initialization statement is specifies as NO, or by adding one bit to the mask if FENCE=YES.

This exit is effected by a single exit point, at label $STRAKX.

ENVIRONMENT: User address space

POINT OF PROCESSING:

This exit is taken from $STRAK (HASPSSSM), when there is no space available on the spool volumes from which the current job is permitted to allocate space. If the job is permitted to allocate space from any spool volume, that is, if the spool partitioning mask is set to all ones, this exit is not invoked; it is only invoked when spool partitioning is in effect.

When $STRAK is called, it first checks the TGB for an available track group from an eligible spool volume. If a track group is available, $STRAK allocates it to the job. If there is no available track group on an eligible spool volume represented in the current TGB, $STRAK checks the bit map in SVTMTSPL to determine whether there are any track groups not represented in the current TGB that are available on any of the eligible spool volumes. If there are additional track groups available, $STRAK posts ($$POST) the checkpoint processor to replenish the TGB via the KBLOB routine. However, if there are no more track groups available on any of the eligible spool volumes, $STRAK must determine

whether spool partitioning is in effect. If no spool partitioning is in effect, that is, if the mask is set to all ones, then $STRAK waits (WAIT) for the JES2 dispatcher to free at least one of the track groups currently in use. If spool partitioning is in effect, $STRAK invokes the exit to expand the spool partitioning mask; once the mask has been expanded, $STRAK restarts its allocation cycle.

As with Exit 11, this exit is not invoked when there is still space available on the job's designated spool volumes and the current TGB has to be replenished. The distinction between the invocation of this exit to expand the spool partitioning mask and JES2's automatic replenishment of the TGB is described in greater detail in the "Other Programming Considerations" below.

REGISTER CONTENTS WHEN CONTROL IS PASSED TO THE EXIT ROUTINE:

R0 N/A

R1 Address of the 3-word parameter list, having the following structure:

- **word 1 (+0)** Address of the IOT
- **word 2 (+4)** Address of the JCT (if available); otherwise 0
- **word 3 (+8)** Address of a 32-byte spool partitioning mask work area, initially set to the IOTSAMSK field

R2-R10 N/A
R11 Address of the SVT
R12 N/A
R13 Address of an OS-style save area
R14 Return address
R15 Entry address

REGISTER CONTENTS WHEN CONTROL IS PASSED BACK TO JES2:

R0-R14 Unchanged

R15 A return code

RETURN CODES:

0 Tells JES2 that if there are any additional exit routines associated with this exit, execute the next consecutive exit routine. If there are no additional exit routines associated with this exit point, this return code tells JES2 to set the spool partitioning mask as indicated by the FENCE parameter on the SPOOLDEF initialization statement setting and to reissue the $STRAK request.

4 Tells JES2 that even if there are additional exit routines associated with this exit, ignore them; instead, set the spool partitioning mask as indicated by the FENCE parameter on the SPOOLDEF initialization statement setting and reissue the $STRAK request.

8 Tells JES2 that an updated version of the spool partitioning mask--with at least one additional bit turned on--has been passed to JES2 in the spool mask work area and will now determine subsequent spool allocation. This return code also tells JES2 to return the $STRAK request.

JOB EXIT MASK: This exit is subject to job exit mask suppression.

MAPPING MACROS NORMALLY REQUIRED FOR THIS EXIT AT ASSEMBLY: $IOT, $DAS, $JCT, $MIT, $TAB, $CSA, $BUFFER, $HASPEQU, $SVT, RPL

OTHER PROGRAMMING CONSIDERATIONS:

1. From Exit 11, with the substitution of $STRAK for $TRACK, several of the "Other Programming Considerations" apply directly to this exit; these are: 1, 7, 8, and 9. In addition, the considerations that follow apply specifically to this exit.

2. In general, except for environmental considerations and the possible need to set the JCTSAMSK initially, the exit routine that you provide for this exit should duplicate the algorithm that you provide for Exit 11. If you fail to provide a similar exit routine for this exit, you can seriously degrade your implementation of spool partitioning. For example, if the volumes designated for use by a particular job happen to run out of space while the job is being processed in a subtask environment and if you haven't implemented this exit, JES2 automatically expands the mask as indicated by the FENCE parameter setting, in spite of whatever allocation algorithm you have implemented for Exit 11. Now, not only is the job able to allocate track groups from more volumes from the subtask environment, but the JCTSAMSK field in the job's JCT is permanently expanded.

3. If the IOTJCT field of the IOT contains the address of the JCT, the JCT is available. Otherwise, the IOTJCT field contains a 0 and the JCT is not available. If you intend to base your allocation algorithm on values contained in fields of the JCT, you must take into account the fact that the JCT is sometimes unavailable and write a section of your exit routine to take control in these instances.

4. Note that, if implemented, this exit is *not* invoked every time the $STRAK routine is called. As long as track groups remain available on the spool volumes designated for the job by the spool volume partitioning mask, the exit is not invoked. It is invoked only when there is no space available on the spool volumes designated for the job by the current setting of the spool partitioning mask. If all of the track groups from the designated spool volumes represented in the TGB have been allocated but the spool volumes themselves still contain unallocated track groups, this exit is *not* invoked. Instead, when the subtask or user routine makes the $STRAK request, JES2 automatically determines that the TGB needs to be replenished. The $STRAK routine then posts ($$POST) the checkpoint processor and issues the $WAIT macro; the subtask or user routine remains idle until the TGB is replenished. The checkpoint processor calls the KBLOB routine to replenish the TGB with unallocated track groups (from all available spool volumes, which include the job's designated spool volumes). When the TGB is replenished, the checkpoint processor posts ($$POST) the subtask or routine that originally issued the $STRAK request. When the subtask or user routine regains control, the $STRAK request can then be completed, using track groups from the job's designated spool volumes.

5. Each spool volume is represented by a direct access spool entry (DAS). There are $SPOLNUM DASes as defined by the SPOOLNUM parameter on the SPOOLDEF initialization statement. The first $SPOLNUM bits in the JCTSAMSK have a one-to-one correspondence with the DASes. Although there are always $SPOLNUM DASes, not all may be in use at any one time. The DAS entries are contiguous in storage with copies both in the JES2

private area and CSA. Exit 12 can only access the DAS copies in CSA. The CSA copies of the DAS are pointed to by SVTDASA (and the first DAS is pointed to by SVTDAS1). Each DAS in CSA is DASSIZC in length.

DASVOLID contains the 5- to 6-character volume ID. The total number of track groups in a spool volume is contained in the field DASNOTGE. DASTGALC contains the number of track groups allocated by this volume. DASMTCSZ is the size of TGSIZE for this volume. DASALLOC in DASFLAG is used to indicate whether any unused track groups from this volume will be put in the TGB for allocation. DASEXSTS indicates whether this DAS is currently being used to represent a volume. Other flags in DASFLAG give more status information on the volume. DASMASK gives the bit setting for this volume that matches the bit for this volume to be used with JCTSAMSK.

In the $TRACK exit, the DASs can be looped through using the DASTRAKQ. This queue contains offsets of the next DAS in the queue. This is anchored out of the $HCT by field $DASTRKQ. This permits the user to just scan DASs that are represented in the track group map (in the order they are in the track group map). This would ensure only defined volumes are scanned (except a volume just beginning the start process may not be on the queue yet). The $STRAK exit cannot use this queue since they do not have access to the header. They must loop through every DAS. Reaching a DAS that is not being used does not mean the remaining DASs are not used. The empty DAS could have represented a volume that is now drained.

6. Be sure your exit routines reside in common storage.

Exit 13: TSO/E Interactive Data Transmission Facility Screening and Notification

FUNCTION:

This exit allows you to provide an exit routine for enhancing the functions of the TSO/E interactive data transmission facility. (Note: For a description of this facility, see *TSO/E Interactive Data Transmission Facility User's Guide*. The facility is part of Program Product 5665-285.) You can use this exit to screen incoming files as they arrive at the receiver's network node and to notify the receiver that a transmitted file has arrived. If this exit is implemented and enabled, it is taken every time the JES2 network SYSOUT receiver reads and processes the network data set header (NDH) of a file transmitted via the TSO/E interactive data transmission facility.

To screen an incoming file, write an exit routine to perform a validity check on the file's control information contained in the network job header (NJH), the network data set header (NDH), and the peripheral data definition block (PDDB). On the basis of this check, the exit routine can decide to delete the file, to route it to another user, or to allow it to remain targeted for the TSO receiver requested by the sender. Provided that the sender is identified in the NJHGUSID field of the NJH, JES2 sends the following message back to the sender after deleting the file:

```
$HASP548  MAIL TO nodename/userid DELETED, INVALID USERID
```

The message identifies the intended receiver as *userid* and the intended receiver's node as *nodename*. When deleting a transmitted file, the exit routine can also modify the message text ("DELETED, INVALID USERID") or replace the message text with a text of its own. Use an exit-generated text to provide the sender with the specific reason for the deletion.

To direct JES2 to notify the TSO receiver that a transmitted file has arrived, write an exit routine to pass a return code of 8 to JES2. As with screening, you can write an exit routine to examine control information and, on that basis, notify selected receivers. To notify every receiver unconditionally, write an exit routine to generate a return code of 8 whenever it is entered--that is, every time a file arrives via the TSO/E interactive data transmission facility. When passed a return code of 8 from this exit, JES2 invokes the TSO SEND function to issue the following message to the receiver:

```
$HASP549  MAIL FROM nodename/userid RECORDS nnn
```

The message identifies the sender as *userid*, indicates the system of origin as *nodename*, and indicates the number of records in the file by *nnn*. When ready, the TSO user can now issue the TSO/E RECEIVE command to accept the file. If the receiving node is a multi-access spool configuration, you can use your exit routine to specify the system on which the TSO/E SEND command will notify the receiver.

Note that if the exit routine deletes a transmitted file--even when the exit routine has generated a return code of 8--JES2 automatically suppresses the $HASP549 notification message. If the exit routine routes a file to an alternate receiver (that

is, a TSO user other than the sender's intended receiver) and generates a return code of 8, $HASP549 is sent to the alternate receiver; the original receiver is not notified.

This exit is established by a single exit point, at label NSRNMXIT.

ENVIRONMENT: JES2 main task

POINT OF PROCESSING:

This exit is taken from the JES2 main task, from the network SYSOUT receiver. Exit point MAILXIT occurs in the NSRDSH area of HASPNET, the SYSOUT receiver main entry point. It occurs just after the network SYSOUT receiver has read and processed the network data set header (NDH) of a file transmitted via the TSO/E interactive data transmission facility. JES2 has built a peripheral data definition block (PDDB) from the information in the NDH but has not yet written the PDDB to spool.

REGISTER CONTENTS WHEN CONTROL IS PASSED TO THE EXIT ROUTINE:

R0 N/A
R1 Pointer to a 5-word parameter list with the following structure:

- **Word 1 (+0)** address of the network job header (NJH)
- **Word 2 (+4)** address of the network data set header (NDH)
- **Word 3 (+8)** address of the peripheral data definition block (PDDB) built for the received file
- **Word 4 (+C)** address of a 1-byte binary field containing the sysid value for the multi-access spool member on which the intended receiver is currently logged on; if the intended receiver is not currently logged on, this field contains a zero
- **Word 5 (+10)** address of a 70-byte message text area for $HASP548; JES2 has initialized this area to 'DELETED, INVALID USERID'

R2-R9 N/A
R10 Address of the JCT
R11 Address of the HCT
R12 N/A
R13 Address of the PCE
R14 Return address
R15 Entry address

REGISTER CONTENTS WHEN CONTROL IS PASSED BACK TO JES2:

R0-R1 N/A
R15 A return code

RETURN CODES:

0 Tells JES2 that if there are any additional exit routines associated with this exit, call the next consecutive exit routine; if there are no additional exit routines associated with this exit, continue with normal network SYSOUT receiver processing. Note, however, that if the exit routine has set to one the PDB1NSOT bit in the PDBFLAG1 byte of the PDDB, normal processing is to delete the file. If the exit routine has altered the PDBWTRID field of the PDDB, normal processing is to route the file to a user other than the sender's intended receiver.

4 Tells JES2 to ignore any additional exit routines associated with this exit and to continue with normal network SYSOUT receiver processing. Note, however, that if the exit routine has set to one the PDB1NSOT bit in the PDBFLAG1 byte of the PDDB, normal processing is to delete the file. If the exit routine has altered the PDBWTRID field of the PDDB, normal processing is to route the file to a user other than the sender's intended receiver.

8 Tells JES2 to issue the $HASP549 notification message to the intended receiver of the transmitted file. Note, however, that if the exit routine has set to one the PDB1NSOT bit in the PDBFLAG1 byte of the PDDB, JES2 ignores this return code and suppresses the $HASP549 message. If the exit routine has altered the PDBWTRID field of the PDDB, JES2 routes the $HASP549 message to the user now indicated by the contents of PDBWTRID; the sender's intended receiver does *not* receive this notification message.

JOB EXIT MASK: This exit is not subject to job exit mask suppression.

MAPPING MACROS NORMALLY REQUIRED FOR THIS EXIT AT ASSEMBLY: $HASPEQU, $HCT, $JCT, $MIT, $NHD, $PCE, $PDDB

OTHER PROGRAMMING CONSIDERATIONS:

1. This exit will not be taken if a file is received from another member of the same multi-access spool configuration. If you require notification between different members of the same multi-access spool configuration, use the TSO/E TRANSMIT exit (INMXZ01).

2. When using the exit to screen transmissions, you can code your exit routine to examine any of the control characteristics of an incoming file. By making use of the received parameter list, the exit routine can interrogate any fields in the NJH, the NDH, or the PDDB. You can devise a fairly simple algorithm, with the validity check based on a single factor. For example, the exit routine can check the intended receiver's userid in the PDBWTRID field of the PDDB and compare it to a list of authorized receivers in an installation-written control block. Alternately, you can devise a more sophisticated algorithm, basing the validity check on the comparison of several factors. For example, you can limit a sender to certain authorized receivers, or you can limit a receiver to certain authorized senders.

3. In using the exit to initiate receiver notification, you can also code your exit routine to examine any of the control characteristics of an incoming file. A selective notification algorithm would perform comparison checks similar to those performed by a screening algorithm, and, as with a screening algorithm, can be as simple or as sophisticated as you choose to write it. To implement universal--as opposed to selective--notification, write an exit routine that generates a return code of 8 whenever it is entered. This exit, if implemented and enabled, is taken whenever a file arrives via the TSO/E interactive data transmission facility; *all* receivers are notified. However, the intended receiver is notified only if logged on or if the exit has been coded such that the desired SYSID value is moved to the address given as the fourth word of the received parameter list.

4. To delete an incoming file, set to one the PDN1NSOT bit in the PDBFLAG1 byte of the PDDB. Note that even if the exit routine generates a return code of 8, setting the PDN1NSOT bit to one causes JES2 to suppress the $HASP549 notification message.

5. When deleting a file, the exit routine can alter the standard text of the $HASP548 message by moving the replacement text to the 70-byte area whose address is the fifth word of the received parameter list. You may want to code your exit routine with several alternate message texts, each corresponding to the particular condition that caused the exit routine to delete the file.

6. To reroute an incoming file to a user other than the intended receiver, replace the contents of the PDBWTRID field of the PDDB with the userid of the alternate receiver. The alternate userid must be defined at the network node that has received the file. Note that if the exit routine alters the PDBWTRID field and also sets to one the PD1NSOT bit in the PDBFLAG1 byte, the file is deleted rather than rerouted to the alternate receiver.

7. When you write an exit routine that reroutes a file to an alternate receiver and, at the same time, generates a return code of 8, JES2 issues the $HASP549 message to the alternate receiver and *not* to the receiver intended by the sender.

8. If a TSO user submits a job to a JES2 job entry network and specifies the NOTIFY= option on the JOB statement, when the job's system output reaches its ultimate destination the SYSOUT receiver issues a notification message to the TSO user. This message ($HASP546 SYSTEM OUTPUT RECEIVED AT *nodename*) indicates that the job's SYSOUT was received and at which node it was received. NOTIFY= is automatically specified for transmissions that enter the network via the TSO/E interactive data transmission facility. Since NOTIFY= is automatically specified and since JES2 views these transmissions as SYSOUT data sets, JES2 automatically routes the $HASP546 notification message to the TSO sender whenever a transmitted file arrives at this destination node. However, because a number of factors can influence the transmission of the file between its arrival at the destination node and its actual receipt by a TSO end user, users at your installation should be informed that receipt of the $HASP546 message in no way confirms successful transmission to the intended TSO receiver. Users should also be informed that the $HASP546 message is not controlled by the NOTIFY option on the TSO/E TRANSMIT command; JES2 sends this message regardless of whether or not return notification has been requested.

9. To specify the particular system in a multi-access spool configuration on which the TSO/E SEND function will perform receiver notification, code the exit routine to move the desired sysid value to the address given as the fourth word of the received parameter list. The field at this address is one byte in length and contains the sysid of the system in the multi-access spool configuration to which the intended receiver is currently logged on; if the intended receiver is not currently logged on, this field contains a zero. You may want to code your exit routine to interrogate the SYSID value byte before modifying it; if this field contains a zero (indicating that the intended receiver is not logged on), JES2 sends no message. However, if you code your exit routine to set a value in the SYSID field and if the user has previously logged off, the $HASP549 message will be sent to the SYS1.BRODCAST data set.

10. Unless you have written an exit routine to perform receiver validation, JES2 does not check to ensure that a received file is destined for a valid TSO userid. Files that arrive with invalid userids may accumulate on receiving spools. Therefore, you should periodically use the JES2 $D F command to examine receiving spools at your installation. The $D F command enables you to display the names of all files that have entered the system via the TSO/E interactive data transmission facility, along with their target userids (external writer names). If necessary, you can then use the TSO/E RECEIVE command to clear any accumulated files from receiving spools. Specifying the target userid (external writer name) of a particular file on the RECEIVE command allows you to examine the contents of the file and then either to delete it or to reroute it to an alternate (valid) TSO userid.

11. Two JES2 initialization statements can affect the functioning of the TSO/E interactive data transmission facility. Unless your installation uses the TSO/E OUTLIM parameter, of the TSO/E macro, INMXP, to control the size of transmission files, the JES2 ESTPUN statement should be specified with a value sufficient in size to accommodate transmission files. Setting the NUM= parameter on the ESTPUN statement too low will cause the TRANSMIT command processor to terminate abnormally (D37 ABEND) when a transmission file exceeds the ESTPUN limit. Also, specifying the JES2 TSUCLASS initialization statement with the NOOUTPUT option prevents the TRANSMIT command from functioning. Therefore, you should omit the NOOUTPUT option if you intend to allow users at your installation to transmit files via the TSO/E interactive data transmission facility.

12. *Note on recovery*: No recovery is in effect when this exit is taken. As with every exit, you should provide your own recovery within your exit routine.

Figure 3-2 This is a sample JES2 Exit which exists for the sole purpose of setting the return code to 8 so that JES2 will notify users of incoming files from the TSO/E Interactive Data Transmission Facility.

```
UEXIT13  TITLE 'HASP TSO/E NOTIFY EXIT'
         COPY  $HASPGBL             COPY HASPGBL PARAMETERS
HASPU13 $MODULE $BUFFER,                                              C
                $JCT,                                                 C
                $JQE,                                                 C
                $PCE,                                                 C
                $CAT,                                                 C
                $HCT,                                                 C
                $MIT,                                                 C
                $HASPEQU
UEXIT13 $ENTRY BASE=R12
        $SAVE
         LR    R12,R15             LOAD BASE REGISTER
        $RETURN RC=8               RETURN TO CALLER
        $MODEND
         END ,
```

Figure 3-2. Example EXIT13 Routine

Exit 14: Job Queue Work Select - $QGET

FUNCTION:

This exit allows you to provide an exit routine that incorporates your own search algorithms for finding work on the job queue. You use your exit routine to search for an appropriate JQE on the job queue and to indicate when normal JES2 JQE processing should resume.

ENVIRONMENT: JES2 main task

This exit is associated with the $QGET routine, in HASPNUC, which is entered to acquire control of a job queue element (JQE).

The $QGET routine scans the appropriate queue 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, and has independent mode set in agreement with the current mode of the selecting system.

This exit is established by a single exit point at label QVALID.

POINT OF PROCESSING:

This exit is taken from the JES2 main task, from the $QGET routine of HASPNUC, after $QGET first obtains control of the shared queues and verifies that the system is not draining but before it selects a JQE from the appropriate queue. The exit point is before QNEXT in HASPNUC.

REGISTER CONTENTS WHEN CONTROL IS PASSED TO THE EXIT ROUTINE:

R0 N/A

R1 Pointer to a QGET parameter list having the following structure:

- **+0 (word 1)** Address of the node table
- **+4 (word 2)** Address of control block
  - PIT -- if INWS
  - DCT -- if OJTWS or OJTWSC
- **+8 (word 3)** Address of class list (if applicable)
- **+12 (word 4)** Address of the JQE
- **+16 (word 5)** each byte is set as follows:
  - **+16** Length of the class list
  - **+17** Queue type (refer to the $QGET macro description for a list of these) This byte is set to '00' for queue types INWS, OJTWSC, and OJTWS. Byte 18 (the type flag) is used to differentiate between these three queue types.
  - **+18** Work selection type flag
  - **+19** Reserved

R2-R10 N/A

R11 Address of the HCT

R12 N/A

R13 Address of the PCE

R14 The return address

R15 The entry address

REGISTER CONTENTS WHEN CONTROL IS PASSED BACK TO JES2:

R0 N/A
R1 Address of a QGET parameter list having the following structure:

+0 (word 1)	Address of the node table
+4 (word 2)	Address of the control block
+8	Address of the class list
+12 (word 4)	Address of the JQE
+16 (word 5)	each byte is set as follows:

+16	Length of the class list
+17	Queue type (refer to the $QGET macro description for a list of these) This byte is set to '00' for queue types INWS, OJTWSC, and OJTWS. Byte 18 (the type flag) is used to differentiate between these three queue types.
+18	Work selection type flag
+19	Reserved

R2-R14 N/A
R15 A return code

RETURN CODES:

0 Tells JES2 that if there are any additional exit routines associated with this exit, call the next consecutive exit routine. If there are no additional exit routines associated with this exit continue normal queue scan processing.

4 Tells JES2 to ignore any other exit routines associated with this exit and to continue normal queue scan processing.

8 Tells JES2 to bypass normal queue scan processing because a JQE was found by the exit routine. The address of the JQE the exit routine found is provided in the fourth word of the QGET parameter list (the address of which is returned in register 1). Continue JES2 processing at label QGOT in HASPNUC.

12 Tells JES2 to bypass normal processing because a JQE was not found. Continue JES2 processing at level QEXIT in HASPNUC.

JOB EXIT MASK: This exit is not subject to job exit mask suppression.

MAPPING MACROS NORMALLY REQUIRED FOR THIS EXIT AT ASSEMBLY: $HASPEQU, $HCT, $JQE, $MIT, $PCE

OTHER PROGRAMMING CONSIDERATIONS:

1. The $QSUSE control of the checkpoint record is not maintained if your exit routine issues a $WAIT or invokes a service that issues a $WAIT. You should ensure in your exit routine that you retain control of the checkpoint record before returning to JES2.

2. You must ensure that the spool volume that contains the JQE you pass back to JES2 is online. Also, the JQE cannot be busy, held, or on an inappropriate queue (such as the hardcopy queue).

Exit 15: Output Data Set/Copy Select

FUNCTION:

This exit allows you to create separator pages for each data set printed or for each copy requested of a data set.

The data set separator page exit point allows the exit routine to place a separator page between data sets. This is similar to the function provided by Exit1, separator page exit.

You could also use your exit routine to reset the addresses of the PRTRANS table and the CCW translate tables. The default addresses for both the PRTRANS table and the CCW command code translate tables are set in the parameter list before your exit routine is called. You can change the defaults by changing the parameter list to point to your own PRTRANS table and to point to your own CCW command code translate tables.

The PRTRANS table translates user data and changes each line to be printed on a local 1403 or remote printer. The default PRTRANS table changes lowercase letters to uppercase and any characters that are invalid on a specific universal character set (UCS) to blanks. The CCW table translates user-specified channel commands into installation-defined channel commands. **Caution: Translation of initialization, diagnostic, or control CCWs may cause unpredictable results.**

ENVIRONMENT: JES2 main task

POINT OF PROCESSING:

This exit is taken from the JES2 main task in HASPPRPU at exit points PEXT15O and PEXT15D. The exit is taken once for each output data set where the PDDB matches the job output element (JOE) and once for each copy of the data set.

Exit point *PEXT15O* occurs where the final PDDB checking is performed.

Exit point *PEXT15D* occurs for creating a data set separator page.

REGISTER CONTENTS WHEN CONTROL IS PASSED TO THE EXIT ROUTINE:

R0 A code indicating whether the exit call is for a copy of an output data set or whether the exit call is for a data set.

- **0** indicates the exit call is for a data set select (PEXT15O)
- **4** indicates the exit call is for a copy of a data set (PEXT15D)

R1 Address of a 10-word parameter list, having the following structure:

Word 1 (+0)	original number of copies of the data set to be printed (this number does not reflect changes made by this exit)
Word 2 (+4)	address of the work JOE
Word 3 (+8)	address of the JCT
Word 4 (+12)	address of the PDDB
Word 5 (+16)	address of the DCT
Word 6 (+20)	number of copies currently printed
Word 7 (+24)	print translate table address
Word 8 (+28)	CCW translate table address
Word 9 (+32)	copy group address
Word 10 (+36)	current copy group count

R2-R10 N/A

R11 Address of the HCT

R12 N/A

R13 Address of the PCE

R14 Return address

R15 Entry address

REGISTER CONTENTS WHEN CONTROL IS PASSED BACK TO JES2:

R0-R1 N/A

R15 A return code

RETURN CODES:

0 Tells JES2 that if there are any additional exit routines associated with this exit, call the next consecutive exit routine. If there are no additional exit routines associated with exit continue normal JES2 processing.

4 Tells JES2 to ignore any other exit routines associated with this exit and to continue normal JES2 processing.

8 Tells JES2 not to select this PDDB (applies for R0=0 only).

JOB EXIT MASK: This exit is subject to job exit mask suppression.

MAPPING MACROS NORMALLY REQUIRED FOR THIS EXIT AT ASSEMBLY: $DCT, $HASPEQU, $HCT, $JCT, $JOE, $MIT, $PDDB, $PCE

OTHER PROGRAMMING CONSIDERATIONS:

1. The following fields can be changed via the parameter list

   a. Copies to be printed (255 maximum).
   b. Pointer to translate table
   c. CCW translate table

2. Separator pages should *not* be produced from the data set select call (R0=0), because printer setup processing has not occurred yet.

3. The data set copy count and copy group count cannot be changed on the separator page call to Exit 15 (R0=4) because setup processing has already occurred; rather, these changes should be made on the data set select call to Exit 15, word 1 (R0=0).

4. Separator pages produced by this exit will be affected by the data set copy group count. The copy group count is sent to the 3800 printer prior to the call to Exit 15. The printer will repeat all pages, including separator pages, as specified by the copy group count.

5. If the spooling capabilities of a remote SNA device (such as the 3790) are operating, use the *$SEPPDIR service routine* to send a peripheral data information record (PDIR) to the device. Use the $GETBUF to supply this routine with buffers and the $FREEBUF macro to release them after your separator has been created. Buffers supplied to $SEPPDIR must be HASP-type buffers.

Exit 16: Notify

FUNCTION:

This exit allows you to change notify message routing and to examine and modify $WTO messages before they are sent to the TSO user.

Use your exit routine and the CMB to access the intended message, change it inplace, or replace it with a new message.

This exit is established by the exit point at label *OPNEXIT* in HASPHOPE.

ENVIRONMENT: JES2 main task

POINT OF PROCESSING:

This exit is taken from the output processor in HASPHOPE before sending the $WTO notify message.

REGISTER CONTENTS WHEN CONTROL IS PASSED TO THE EXIT ROUTINE:

R0 A code indicating if this is the first or succeeding $HASP165 (JOB nnnnn ENDED -- reason text) message

- **0** Indicates that this is the first (and possibly only) message indicating the end of the job
- **4** Indicates that this is not the first message for this job going through the output processor

R1 Address of a 3-word parameter list with the following structure:

Word 1 (+0) address of the message that is to be sent
Word 2 (+4) address of the $WTO parameter list
Word 3 (+8) address of the JCT

R2-R10 N/A

R11 Address of the HCT

R12 N/A

R13 Address of the output processor PCE

R14 Return address

R15 Entry address

REGISTER CONTENTS WHEN CONTROL IS PASSED BACK TO JES2:

R0 N/A

R1 Address of the 3-word parameter list

R15 A return code

RETURN CODES:

0	Tells JES2 that if there are any additional exit routines associated with this exit, call the next consecutive exit routine. If there are no additional exit routines associated with exit continue normal notify processing.
4	Tells JES2 to ignore any other exit routines associated with this exit and to continue normal notify processing.
8	Tells JES2 not to issue the notify $WTO.

JOB EXIT MASK: This exit is subject to job exit mask suppression.

MAPPING MACROS NORMALLY REQUIRED FOR THIS EXIT AT ASSEMBLY: $CMB, $JCT, $HASPEQU, $HCT, $MIT, $PCE

OTHER PROGRAMMING CONSIDERATIONS:

1. The CMB maps the $WTO parameter list. You map the parameter list by performing a USING on CMBWTOPL.

2. CMBML in the $WTO parameter list is the length of the message that is intended to be sent. Whether your exit routine changes the messages in place or replaces it, you must update CMBML with the length of the new message. The intended message can be changed in place for up to a length of 86 bytes.

3. To change the node where the notify message is to be sent, move correct node number NITNUM (of the NIT) to CMBTONOD.

4. To change the TSO user that the notify message is to go to store the TSO user id (7-character id) in CMB user.

5. On return from the exit, JES2 uses the address of the message in the first word of the parameter list.

6. For a return of 8 from your exit routine, JES2 resumes processing at OPNOTX in HASPPRPU.

Exit 17: BSC RJE SIGNON/SIGNOFF

FUNCTION:

This exit allows you to exercise more control over your BSC RJE remote devices. With this exit you can implement exit routines to:

- Selectively perform additional security checks over and above the standard password processing of the sign-on card image.
- Selectively limit both the number and types of remote devices that can be on the system at any one time.
- Selectively bypass security checks.
- Implement installation-defined scanning of sign-on card images.
- Collect statistics concerning RJE operations on the BSC line and report the results of the activity.

ENVIRONMENT: JES2 main task

POINT OF PROCESSING:

This exit is taken from the JES2 main task, during BSC RJE sign-on and sign-off processing of HASPBSC. Three exit points are defined; two sign-on exit points for performing additional security or checks and one sign-off exit point for gathering statistics about terminal usage.

The sign-on exit points are in the MSIGNON routines of HASPBSC. One exit point MSOXITA is before sign-on and password processing; the other MSOXITB is after sign-on and password processing.

The exit point MSOXITA before sign-on and password processing allows your exit routine to scan the incoming sign-on card. Your exit routine may also bypass both the JES2 syntax checking of the sign-on and the remote and line password parameters on the sign-on card or just bypass only the sign-on syntax checking. The exit point MSOXITB after sign-on and password processing allows your exit routine to provide additional setup of the remote terminal environment.

HASPBSC calls the sign-off exit point MDSXITA after writing the disconnect message at label MDSWTO.

REGISTER CONTENTS WHEN CONTROL IS PASSED TO THE EXIT ROUTINE:

R0 Indicates whether sign-on or sign-off processing is in effect. The following values apply:

- **0** indicates a sign-on before sign-on parameters are processed.
- **4** indicates a sign-on after the sign-on parameters have been processed.
- **8** indicates sign-off processing.

R1 Address of a 5-word parameter list, having the following structure:

- **Word 1 (+0)** address of the remote attribute table (RAT) (for R0=0 only) address of the RAT entry (for R0=4 or 8)
- **Word 2 (+4)** address of the line DCT
- **Word 3 (+8)** zero (reserved for SNA)
- **Word 4 (+12)** address of the card image (for R0=0 only)

  Otherwise not applicable
- **Word 5 (+16)** length of the card image for R0=0 only) Otherwise not applicable

  (The length is always 80.)

R2-R10 N/A

R11 Address of the HCT

R12 N/A

R13 Address of the line manager or remote reader PCE

R14 Return address

R15 Entry address

REGISTER CONTENTS WHEN CONTROL IS PASSED BACK TO JES2:

R0 Address of the remote's RAT entry when the return code in R15 is 12 or 16 and the sign-on indication in R0 is "0"

Otherwise not applicable

R1 N/A

R15 A return code

RETURN CODES:

0 Tells JES2 that if there are any additional exit routines associated with this exit, call the next consecutive exit routine. If there are no additional exit routines associated with this exit continue normal sign-on/sign-off processing continues.

4 Tells JES2 to ignore any other exit routines associated with this exit and to continue normal sign-on/sign-off processing.

8 Tells JES2 to terminate normal sign-on processing.

12 Tells JES2 to bypass syntax but not password processing.

16 Tells JES2 to bypass both syntax and password processing.

Note: RC 8, 12, and 16 are only valid for the exit when called from label MSOXITA (that is, the first call to the exit, R0=0).

JOB EXIT MASK: This exit is not subject to job exit mask suppression.

MAPPING MACROS NORMALLY REQUIRED FOR THIS EXIT AT ASSEMBLY: $DCT, $HASPEQU, $HCT, $MIT, $PCE, $RAT

OTHER PROGRAMMING CONSIDERATIONS:

1. For exit point MSOXITA (R0=0) your exit routine has the option to return a return code that allows the user to specify that the sign-on should be rejected. A return code of 12 or 16 indicates that normal HASPBSC sign-on processing can be bypassed. In this case your user exit routine is responsible for performing all the necessary syntax processing that HASPBSC does and for returning a valid RAT entry pointer in R0.

2. For the sign-off exit point your exit routine should return a return code of 0 or 4 so that normal processing can continue.

3. To define and implement an installation-defined remote name, change the remote name to a standard JES2 remote name on the sign-on card and return with a return code of 0, or supply a valid RAT pointer (valid for the installation-defined remote name) and return with or return code of 12 or 16.

4. Your user exit routine should not issue a $WAIT or invoke a service routine that issues a $WAIT.

5. The password on a sign-on card begins in column 24. (For the syntax of the sign-on card see *JES2 Initialization and Tuning.*)

6. The $RETURN macro destroys the contents of register 0. Therefore, if you return the RAT address in R0, be certain to have provided a $STORE R0 instruction prior to the $RETURN to place the contents of R0 in the current save area prior to return to JES2.

Exit 18: SNA RJE LOGON/LOGOFF

FUNCTION:

This exit allows you to exercise more control over your SNA RJE remote devices. With this exit you can implement exit routines to:

- Selectively perform additional security checks over and above the standard password processing of the sign-on card image.
- Selectively limit both the number and types of remote devices that can be on the system at any one time.
- Selectively bypass security checks.
- Implement installation-defined scanning of sign-on card images.
- Collect statistics concerning RJE operations on the SNA line and report the results of the activity.

ENVIRONMENT: JES2 main task

POINT OF PROCESSING:

This exit is taken from the JES2 main task during the SNA RJE logon and logoff processing of HASPSNA. Three exit points are defined for logon processing:

- At exit point MSNALXIT for a normal logon during REQ END processing after label MSNALPAR, your exit routine can be invoked to:
  - continue normal logon processing.
  - terminate normal logon processing.
  - perform password checking but not syntax checking.
  - bypass syntax and password checking.
- At exit point MSNALXT2 your exit can get control when the remote terminal is logged on.
- Just before checkpointing the remote autologon at exit point MALGXIT, your exit can control autologon for the remote terminal.

One exit point (MICEXIT) is defined for logoff processing. This exit point is after label MICEDMSG in the session control subroutines of HASPSNA before the remote logoff message is issued. You can use this exit point for gathering statistics and reporting remote device activity.

REGISTER CONTENTS WHEN CONTROL IS PASSED TO THE EXIT ROUTINE:

R0 A logon or logoff indication having the following meanings:

- **0** indicates syntax processing for a normal logon
- **4** indicates logon processing for a normal logon after logon parameters have been processed
- **8** indicates logoff processing
- **12** indicates autologon processing

R1 Address of a 5-word parameter list having the following structure:

Word 1 (+0) address of the remote attribute table (RAT) when R0 indicates a normal logon process of "0"

address of a RAT entry when R0 indicates other than a normal logon process (i.e., R0 contains a value of 4, 8, or 12).

Word 2 (+4)

- 0 during syntax processing (that is, R0 = 0)
- address of the line DCT after logon is complete (that is, R0 ≠ 0)

Word 3 (+8) address of the ICE

Word 4 (+12) address of the bind user data when R0 indicates normal logon processing (that is, R0 = 0).

Word 5 (+16) length of the bind user data when R0 indicates normal logon processing (that is, R0 = 0).

R2-R10 N/A

R11 Address of the HCT

R12 N/A

R13 Address of the line manage PCE

R14 Return address

R15 Entry address

REGISTER CONTENTS WHEN CONTROL IS PASSED BACK TO JES2:

R0 Address of the RAT entry when R15 contains a return code of 12 or 16 and the logon indication in R0 is 0.

Otherwise a portion of register 0 is ignored.

R1 N/A

R15 A return code

RETURN CODES:

0	Tells JES2 that if there are any additional exit routines associated with this exit, call the next consecutive exit routine. If there are no additional exit routines associated with this exit continue normal logon/logoff processing.
4	Tells JES2 to ignore any other exit routines associated with this exit and to continue normal logon/logoff processing.
8	Tells JES2 to terminate normal logon processing (R0=0 or 12 only).
12	Tells JES2 to perform password checking but bypass logon syntax processing (R0=0 only).
16	Tells JES2 to bypass logon password and syntax processing (R0=0 only).

JOB EXIT MASK: This exit is not subject to job exit mask suppression.

MAPPING MACROS NORMALLY REQUIRED FOR THIS EXIT AT ASSEMBLY: $DCT, $HASPEQU, $HCT, $ICE, $MIT, $PCE, $RAT

OTHER PROGRAMMING CONSIDERATIONS:

1. In the case of logoff processing, JES2 does not expect a return code from your exit routine. Normal logoff processing proceeds.

2. Your user exit routine should not issue a $WAIT or use a service routine that issues a $WAIT.

3. To define and implement a installation-defined remote name, change the remote name to a standard JES2 remote name on the remote logon card and return with a return code of 0, or supply a valid RAT pointer (valid for the installation-defined remote name) and return with a return code of 12 or 16.

Exit 19: Initialization Statement

FUNCTION:

This exit allows you to process each JES2 initialization statement before JES2 processes the statement. You can use your exit routine to do any of the following functions:

- check or analyze each initialization statement.
- alter values supplied on an initialization statement.
- implement your own initialization statements.
- tailor the initialization statement stream to provide for specific requirements of this start of JES2 (e.g., add or delete parameters based on the period within administrative cycles or the operator shift).

Your exit routine can modify, replace, delete, or insert statements in the initialization statement stream. Also, Exit 19 can optionally indicate to JES2 that this initialization of JES2 should be terminated and JES2 should exit the system rather than continue.

ENVIRONMENT: JES2 main task (Initialization) - JES2 dispatcher disabled

POINT OF PROCESSING:

This exit is taken during JES2 initialization from the initialization routine (IR) that processes parameter input (IRPL) in HASPIRPL. The exit point is at label NPLEXIT. IRPL is called out of the initialization routine processing loop (IRLOOP) in HASPIRA before most other IRs have been called. Previously executed IRs have processed the initialization options, analyzed the SSI status, and allocated a series of temporary and permanent control blocks. Exit 0 routines, called during initialization options processing, may have allocated installation control blocks that may be used now by Exit 19 routines.

HASPIRPL opens the initialization parameter data set (HASPPARM) and then begins a loop; get an initialization statement from HASPPARM or the operator console or a previous insertion by Exit 19, pass it to Exit 19, log the statement, process the statement using the $SCAN facility if Exit 19 has not indicated it should be deleted. When all input is exhausted, IRPL closes the parameter and log data sets.

REGISTER CONTENTS WHEN CONTROL IS PASSED TO THE EXIT ROUTINE:

R0 An indication of how the initialization input was supplied. The following values in R0 are possible:

- **0** input came from the HASPARM parameter library file
- **4** input came from the console
- **8** input came from a previous insertion by an Exit 19 routine.

R1 A 4-word parameter list having the following structure

- **Word 1 (+0)** address of the initialization statement about to be processed. You can modify the statement or replace the statement by altering this field.
- **Word 2 (+4)** length of the complete initialization statement passed. If you alter the passed statement or replace it, you should reset this field to the correct new statement length.
- **Word 3 (+8)** a word that can be used by Exit 19 to specify the address of an initialization statement you want to insert at the next possible statement insertion point. JES2 will log an information diagnostic indicating the statement was inserted by Exit 19.
- **Word 4 (+12)** length of the initialization statement pointed to by word 3.

R2-R10 N/A

R11 Address of the HCT

R12 N/A

R13 Address of initialization PCE - the PCE work area for this PCE is the common initialization routine work area, mapped by the $CIRWORK macro.

R14 Return address

R15 Entry address

REGISTER CONTENTS WHEN CONTROL IS PASSED BACK TO JES2:

R0-R1 N/A

R15 A return code

RETURN CODES:

0 Tells JES2 that if there are any additional exit routines associated with this exit, call the next consecutive exit routine. If there are no additional exit routines associated with this exit continue normal initialization statement processing. The exit routines might have changed or replaced the initialization statement passed.

4 As for return code 0, except JES2 should ignore any other exit routines associated with this exit.

8 Tells JES2 to bypass this initialization statement and continue with the next statement. JES2 will log the statement and a diagnostic information message indicating it was bypassed by Exit 19.

12 Tells JES2 to terminate all initialization processing and exit the system. HASPIRPL issues message $HASP864 and returns to the IRLOOP with a return code of 8.

JOB EXIT MASK: This exit is not subject to job exit mask suppression.

MAPPING MACROS NORMALLY REQUIRED FOR THIS EXIT AT ASSEMBLY: $CIRWORK, $HASPEQU, $HCT, $MIT, $PCE

OTHER PROGRAMMING CONSIDERATIONS:

1. Your EXITnnn and LOAD initialization statements for this exit must be placed in the initialization deck ahead of those initialization statements that your exit routine is to scan. The EXITnnn statement must ENABLE the exit; the $T EXITnnn command cannot be used to ENABLE the exit at a later time since the point of processing for Exit 19 is prior to the time at which the command processor is made functional.

2. Tracing for this exit is disabled because of its sequence in the initialization process.

3. JES2 does not have a recovery environment established at the processing point for Exit 19 (the JES2 ESTAE will process termination, but not recover).

4. Because Exit 19 is called early in JES2 initialization, some main task services may not be functional and some control blocks and interfaces may not be established. The JES2 dispatcher is not yet functional, so MVS protocol should be used in Exit 19 routines (WAIT rather than $WAIT, ESTAE rather than $ESTAE, etc).

5. The CONSOLE statement simulated after all other parameter input is exhausted if the CONSOLE initialization option was specified is not presented to Exit 19 exit routines.

6. Exit 19 routines may change the initialization statement passed or replace it by changing the address and length in the exit parameter list. They may also indicate, via a return code, that JES2 should bypass processing of the statement (perhaps because the routine has processed the statement already). Note that JES2 writes the statement (and any later diagnostics) to the log data set and hardcopy console only after return from the exit. Therefore the exit routines may want to log the statement passed from JES2, for diagnostic purposes, before changing or replacing it. The $STMTLOG macro and service routine is provided to perform the logging function.

7. Independent of the actions of the exit routine that effect the status of the statement passed, a new initialization statement may be inserted into the parameter stream by the exit routine by returning a statement address and length in the exit parameter list. The inserted statement will be processed as soon as the current statement is completely processed. Note that the current statement is not completely processed until either it is bypassed by exit 19, successfully scanned and processed by JES2, or found to be in error by JES2 and the resultant operator interaction by JES2 is complete. Since the operator interaction may involve input of multiple new initialization statements from the operator, the inserted statement may not be processed until after later calls to Exit 19. Also, when there are multiple exit 19 routines, only one routine can perform a statement insertion. For that reason, Exit 19 routines should verify that the insertion statement address and length in the exit parameter list are zero before using those fields to insert a statement.

8. The processing that JES2 does for each statement after calling Exit 19 is performed using the JES2 $SCAN facility and a collection of tables. The tables define the parameter input allowed and how to process it. The scan may involve multiple levels of scanning, i.e. parameters which have sub-parameters, etc. At each level, a new table is used. Each table is actually composed of two tables, an installation-defined table followed by a JES2-defined table.

   By specifying installation-defined tables, an installation can implement its own initialization parameters on existing JES2 statements, or replace the JES2 definition for existing statements or parameters. Thus this function can be accomplished without implementing Exit 19, or in conjunction with an implementation of Exit 19. Also, the $SCAN facility itself can be used from an Exit 19 routine to process initialization statements.

Exit 20: End of Input

FUNCTION:

This exit allows you to do the following:

- Selectively assign a job's system affinity, execution node, and priority based on an installation's unique requirements and processing workload.
- Based on installation-defined criteria, terminate a job's normal processing and selectively print or not print its output.

Note: Refer to Appendix D, "JES2 Exit Usage Limitations" for a listing of specific instances when this exit will be invoked or not invoked

ENVIRONMENT: JES2 main task

POINT OF PROCESSING:

This exit is taken at exit point REXITA in the subroutine RJOBEND of HASPRDR in the JES2 main task.

REGISTER CONTENTS WHEN CONTROL IS PASSED TO THE EXIT ROUTINE:

R0	Zero (0)
R1-R9	N/A
R10	Address of the JCT
R11	Address of the HCT
R12	N/A
R13	Address of the HASPRDR PCE
R14	Return address
R15	Entry address

REGISTER CONTENTS WHEN CONTROL IS PASSED BACK TO JES2:

R0-R1	N/A
R15	A return code

RETURN CODES:

0	Tells JES2 that if there are any additional exit routines associated with this exit, call the next consecutive exit routine. If there are no additional exit routines associated with this exit continue normal processing.
4	Tells JES2 to ignore any other exit routines associated with this exit and to continue normal processing.
8	Tells JES2 to terminate normal processing and print the output.
12	tells JES2 to terminate normal processing without printing the output.

JOB EXIT MASK: This exit is subject to job exit mask suppression

MAPPING MACROS NORMALLY REQUIRED FOR THIS EXIT AT ASSEMBLY: $JCT, $HCT, $PCE, $HASPEQU, $MIT, $RDRWORK, $BUFFER, RPL

OTHER PROGRAMMING CONSIDERATIONS:

1. To change system affinity, set the RDWSIAFF field in the HASPRDR PCE work area.

2. To change the priority set JCTIPRIO in the JCT.

3. Validate all fields in the JCT prior to using them because not all fields contain valid values each time Exit 20 is invoked.

4. This exit will not be taken if the JES2 input service processor fails the job due to an invalid JES control statement or if the submittor has included a /*DEL or /*PURGE control statement.

5. *Note on recovery:* $ESTAE recovery is in effect. The RDRRCV0 recovery routine will attempt to recovery from program check errors, including program check errors in the exit routine itself. However, as with every exit, your exit routine *should not* depend on JES2 for recovery. JES2 cannot anticipate the exact purpose of your exit routine and can therefore provide no more than minimal recovery. You should provide your own recovery within your exit routine.

Exit 21: SMF Record

FUNCTION:

This exit allows you to do the following:

- Selectively queue or not queue the SMF record of JES2 control blocks for processing by SMF.
- Obtain and create SMF control blocks prior to queueing.
- Alter content and length of SMF control block prior to queueing.

ENVIRONMENT: JES2 main task

POINT OF PROCESSING:

This exit is taken in HASPNUC at exit point SMFEXIT whenever a JES2 processor queues an SMF record for eventual processing by the JES2-SMF subtask. The $QUESMFB routine in HASPNUC places a JES2-SMF buffer on the queue of busy JES2-SMF buffers. (The $SMFBUSY cell in the HCT points to the busy queue.)

REGISTER CONTENTS WHEN CONTROL IS PASSED TO THE EXIT ROUTINE:

R0	Zero (0)
R1	SMF buffer address
R2-R10	N/A
R11	Address of the HCT
R12	N/A
R13	Address of the caller's PCE
R14	Return address
R15	Entry address

REGISTER CONTENTS WHEN CONTROL IS PASSED BACK TO JES2:

R0-R1	N/A
R15	A return code

RETURN CODES:

0	Tells JES2 that if there are any additional exit routines associated with this exit, call the next consecutive exit routine. If there are no additional exit routines associated with this exit continue normal SMF queue processing.
4	Tells JES2 to ignore any other exit routines associated with this exit and to continue normal SMF queue processing.
8	Tells JES2 to terminate normal SMF queue processing.

JOB EXIT MASK: This exit is not subject to job exit mask suppression.

MAPPING MACROS NORMALLY REQUIRED FOR THIS EXIT AT ASSEMBLY: $HASPEQU, $MIT, $PCE, $SMF

OTHER PROGRAMMING CONSIDERATIONS:

1. When modifying the SMF record, your exit routine can increase the size of the SMF record up to a length of SMFLNG (bytes).

2. You can issue $GETSMFB and $QUESMFB in your exit routine.

3. The SMF record type is detected by examining the SMFHDRTY field, **not** the SMFTYPE field of the SMF DSECT.

Exit 22: Cancel/Status

FUNCTION:

This exit allows your installation to implement its own algorithms for job queue searching and for TSO CANCEL/STATUS.

ENVIRONMENT: JES2 main task

Your exit routine can perform its own search for a requested job and indicate whether or not it has found the job, or it can let JES2 perform the standard search.

POINT OF PROCESSING:

This exit is taken just prior to searching the JES2 job queue for a "status" or "cancel" request in HASPXEQ of the JES2 main task. Two exit points, ZTCSEXIT (for cancel/status) and YTCSEXIT (for multiple status overflow) in HASPXEQ exist where HASPXEQ performs the cancel and status functions for the TSO user (XPTCS).

The cancel and status functions execute when a subsystem job block (SJB) is queued (via its SJBCHN chain field) on the SVTTSCS queue in the SSVT. The cancel/status support routine in HASPSSSM performs this queueing. HASPSSSM then issues a WAIT (against SJBECB) to wait for the completion of the cancel/status processing.

REGISTER CONTENTS WHEN CONTROL IS PASSED TO THE EXIT ROUTINE:

R0 An indication that tells why the cancel/status exits are being called. The following reasons apply:

- **0** indicates a single cancel request or the first status request determined by examining the function bit (SJBTFUNC) in the SJB.
- **4** indicates a multiple status recall request.
- **8** indicates a multiple status overflow condition. The buffer that holds the status information is too small.

R1 Address of a 1-word parameter list, having the following structure:

Word 1 (+0) address of the SJB

R2-R10 N/A

R11 Address of the HCT

R12 N/A

R13 Address of the HASPXEQ PCE

R14 Return address

R15 Entry address

REGISTER CONTENTS WHEN CONTROL IS PASSED BACK TO JES2:

R0 N/A

R1 Address of the JQE for return codes of 8 and 12; otherwise not applicable

R15 A return code

RETURN CODES:

0 Tells JES2 that if there are any additional exit routines associated with this exit, call the next consecutive exit routine. If there are no additional exit routines associated with this exit, continue normal processing.

4 Tells JES2 to ignore any other exit routines associated with this exit and to continue normal processing.

8 Tells JES2 to process a single request.

12 Tells JES2 to process a multiple request.

16 Tells JES2 that the exit routine has done all the processing requested. HASPXEQ returns a code of 0 to HASPSSSM.

20 Tells JES2 that the job is not found. HASPXEQ returns a code of 4 to HASPSSSM.

24 Tells JES2 that an invalid combination was requested. HASPXEQ returns a code of 8 to HASPSSSM.

28 Tells JES2 that jobs with the same job name were found. HASPXEQ returns a code of 12 to HASPSSSM.

32 Tells JES2 that the status buffer is too small to hold all the data requested. HASPXEQ returns a code of 16 to HASPSSSM.

36 Tells JES2 that the job was not cancelled because it is on the output queue. HASPXEQ returns a code of 20 to HASPSSSM.

40 Tells JES2 that an invalid cancel request was made. HASPXEQ returns a code of 28 to HASPSSSM.

Note: RC 12 -- 40 are only valid for this exit when called from label ZTCSEXIT (that is, R0=0 or 4 only).

The returned code causes the correct message to be presented to the TSO interface. In the case of multiple status requests (RC=12), register R1 must be returned with a zero to end the processing and cause the messages to be issued.

JOB EXIT MASK: This exit is not subject to job exit mask suppression.

MAPPING MACROS NORMALLY REQUIRED FOR THIS EXIT AT ASSEMBLY: $HASPEQU, $HCT, $MIT, $PCE, $SJB

OTHER PROGRAMMING CONSIDERATIONS:

1. The return code from your exit routine will cause HASPXEQ to pass back the proper return code to HASPSSSM. HASPSSSM propagates that return code to TSO to issue the appropriate message.

2. In the case of multiple cancel status requests, (your exit routine returned a return code of 12), HASPXEQ returns a 0 return code to HASPSSSM in the subsystem job block (SSJB). HASPSSSM propagates that return code to TSO in SSOBRETN.

3. To end a multiple status request your exit routine must return a "0" JQE address in R1 and issue a return code of 12.

4. The $JCAN macro can be used in your exit routine.

5. Message IKJ56216I can be misleading. The second level message tells the user that the job queues were searched for job names consisting of the userid plus one character. You can code your exit so that the job queue is searched for all of the user's jobs.

6. First level messages such as IKJ56190I, IKJ56192I, IJK56197I, and IJK56211I can also be misleading if the exit returned a JQE address in R1 and a return code of 12. The jobname in these messages is constructed by TSO using the TSO user's userid and the last character of the job name in the JQE that was selected by this exit. Depending on the job(s) selected by the exit, the jobname(s) taken from the JQE may not begin with the userid; however, the jobid in the message(s) is correct for the job processed.

Exit 23: FSS Job Separator Page (JSPA) Processing

FUNCTION:

This exit allows you to modify the job separator page data area (JSPA) that is used by the functional subsystem application (FSA) to generate the separator pages for an output group. When JES2 assigns a data set to a functional subsystem application, a JSPA is created. The JSPA provides job- and data set-level information for the data set. This information is used by the FSA to generate job header, job trailer, and data set header pages if required for this data set. Exit 23 can be used to modify this separator page information.

Additionally, you can use this exit to suppress the assignment of a JESNEWS data set. To suppress the assignment of the JESNEWS data set, turn off the flag bit in the JOE information block (JIB) that indicates JESNEWS printing. If this exit returns a return code of 8, both the JESNEWS data set and the separator pages are suppressed.

ENVIRONMENT: Functional subsystem (HASPFSSM)

POINT OF PROCESSING:

This exit is invoked via the exit effector in HASPSSSM from the HASPFSSM module during GETDS processing. Whenever a new JIB is initialized during GETDS processing, Exit 23 is invoked in HASPFSSM. At this time, the associated JCT, IOT, and checkpoint records are read and the JSPA is built.

Refer to "Other Programming Considerations" below for further coding requirements associated with this exit.

REGISTER CONTENTS WHEN CONTROL IS PASSED TO THE EXIT ROUTINE:

R0 0

R1 Address of a 5-word parameter list, having the following structure:

word1 (+0) JSPA address
word2 (+4) JIB address
word3 (+8) FSACB address
word4 (+12) FSSCB address
word5 (+16) GETDS parameter list address (IAZFSIP)

R2-R10 N/A

R11 Address of the HFCT

R12 N/A

R13 The address of an 18-word save area where the exit routine stores the exit effector's registers

R14 Return address

R15 Entry address

REGISTER CONTENTS WHEN CONTROL IS PASSED BACK TO JES2:

R0-R1	N/A
R2-R14	Unchanged
R15	A return code

RETURN CODES:

0 Tells JES2, if additional exit routines are associated with this exit, to call the next consecutive exit routine. If no additional exit routines are associated with this exit a zero return code tells the FSA to produce any separator that has been defined by the installation based on the information contained in the JSPA.

4 Tells JES2 to ignore any additional exit routines associated with this exit. However, all other processing noted for return code 0 is accomplished.

8 Tells JES2 to unconditionally suppress production of the job separator page. The JESNEWS data set is not assigned.

12 Tells JES2 to **unconditionally** (that is, even if the printer has been set to S=N) produce any job separator page.

JOB EXIT MASK: This exit is subject to job exit mask suppression.

MAPPING MACROS NORMALLY REQUIRED FOR THIS EXIT AT ASSEMBLY: $FSACB, $FSSCB, $HASPEQU, $HCT, $HFCT, $JIB, $JSPA, ETD, FSIP

OTHER PROGRAMMING CONSIDERATIONS:

1. A save-area type control block is obtained for use as the parameter list loaded into register 1 when control is passed to the exit routine.

2. The save area whose address is loaded into register 13 when control is passed to the exit routine is obtained via the $GETBLK and passed to the effector via the SAVAREA= keyword on the $EXIT call statement. The SAVAREA is returned via $RETBLK upon return from the exit routine.

3. The assignment of the JESNEWS data set can be checked in the JOE information block (JIB). The JIBFNEWS bit can be set or reset by the exit routine; however, if a return code of 8 is returned, the JESNEWS is not assigned; this is independent of the JIBFNEWS bit setting.

4. IAZFSIP maps the GETDS parameter list.

5. IAZJSPA maps the JSPA parameter list.

6. Exit 23 routines should issue $SAVE after the $ENTRY macro and return to the exit effector using $RETURN. These routines also can call subroutines of their own which also use $SAVE/$RETURN logic.

7. This exit must be linkedited to SYS1.LPALIB. This can be done separately or with HASPSSSM. **Do not linkedit this exit to HASPFSSM.**

8. Information from the PDDBs is available through this exit. Because there is no JES2 data set separator exit for print services facility (PSF), information from the data set PDDBs must be obtained through this exit. The PDDB information can then be moved to the job separator page data area (JSPA) user field (or by providing a pointer to the GETMAINed user area) so as to then be available to the PSF exits.

Exit 24: Post Initialization

FUNCTION:

This exit allows you to make modifications to JES2 control blocks before JES2 initialization ends and to create and initialize control blocks that your installation defines for its own special purposes.

ENVIRONMENT: JES2 Main Task (Initialization) - JES2 dispatcher disabled

The following JES2 initialization steps have been performed prior to your exit routine getting control. Essentially all JES2 initialization is done, but the JES2 warm start processor has not been dispatched yet to perform it's initialization-like processing.

1. The JES2 initialization options are obtained from the operator or the PARM parameter on the EXEC statement and converted into status bits.

2. The JES2 initialization statement data set is read and processed

3. The direct-access devices are scanned, and eligible spooling volumes are identified and allocated to JES2.

4. The spooling and checkpoint data sets are examined and initialized for JES2 processing.

5. The subsystem interface control blocks are constructed and initialized.

6. The unit-record devices, remote job entry lines, and network job entry lines are scanned; eligible and specified devices are located and allocated.

7. JES2 subtasks are attached, and exit routines are located.

8. SMF processing is started by generating a type 47 SMF record.

9. The JES2 control blocks, such as the HASP communications table (HCT), the device control tables (DCT), the data control blocks (DCB), the processor control elements (PCE), the data extent blocks (DEB), and the buffers (IOB), are constructed and initialized.

POINT OF PROCESSING:

When Exit 24 is called, HASPIRA has called each JES2 initialization routine (IR) in turn to perform JES2 initialization. After all the IRs have successfully completed, HASPIRA calls the Exit 24 routine(s) before tracing the JES2 initialization and returning control to the HASJES20 load module (HASPNUC). Upon return from HASPINIT, HASPNUC deletes the HASPINIT load module (if not part of HASJES20) and passes control to the asynchronous input/output processor, $ASYNC, resulting in the dispatching of JES2 processors.

REGISTER CONTENTS WHEN CONTROL IS PASSED TO THE EXIT ROUTINE:

R0	The value "0"
R1-R10	N/A
R11	Address of the HCT
R12	N/A
R13	Address of initialization PCE - the PCE work area for this PCE is the common initialization routine work area, mapped by the $CIRWORK macro.
R14	Return address
R15	Entry address

REGISTER CONTENTS WHEN CONTROL IS PASSED BACK TO JES2:

R0-R1	N/A
R15	A return code

RETURN CODES:

0	Tells JES2 that if there are any additional exit routines associated with this exit, call the next consecutive exit routine. If there are no additional exit routines associated with this exit continue the normal initialization process.
4	Tells JES2 to ignore any other exit routines associated with this exit and to continue normal initialization processing.
8	Tells JES2 to terminate normal initialization. This results in the HASP864 error message to the operator.

JOB EXIT MASK: This exit is not subject to job exit mask suppression.

MAPPING MACROS NORMALLY REQUIRED FOR THIS EXIT AT ASSEMBLY: $CIRWORK, $HASPEQU, $HCT, $HFCT, $PCE, ETP, FSIP

OTHER PROGRAMMING CONSIDERATIONS:

1. The EXITnnn statement for Exit 24 must specify ENABLE the for the exit; the $T EXITnnn command cannot be used to ENABLE the exit at a later time since the point of processing for Exit 24 is prior to the time at which the command processor is made functional.

2. JES2 does not have a recovery environment established at the processing point for Exit 24 (the JES2 ESTAE will process termination, but not recover).

3. Because Exit 24 is called from JES2 initialization, the JES2 dispatcher is not yet functional; so MVS protocol should be used in Exit 24 routines (for example, WAIT rather than $WAIT and ESTAE rather than $ESTAE).

4. If Exit 24 returns a return code of 8, HASPIRA issues message $HASP864 INITIALIZATION TERMINATED BY USER EXIT 24. The $HASP428 message will also be issued before final termination.

5. Your exit routine can access JES2 control blocks through the HCT. Your exit routine can then access DCTs, PCEs, buffers, the UCT, etc for making modifications.

6. Your exit routine is responsible for establishing addressability to your own special control blocks. The HCT points to the optional user-defined UCT and other areas are provided in the HCT for various installation uses, identified by labels $USER1 through $USER5.

Exit 25: JCT Read (FSS)

FUNCTION:

This exit allows you to provide an exit routine to receive control whenever a JES2 functional subsystem address space (HASPFSSM) performs JCT I/O. That is, your routine receives control just after the JCT is read into storage by the HASPFSSM module which executes as part of the FSS address space. (Note: Whenever JCT I/O is performed by the JES2 main task, Exit 7 serves the purpose of this exit, and Exit 8 is used whenever a JES2 subtask or a routine running in the user address space (HASPSSSM) performs JCT I/O.)

You can use this exit to perform I/O for any installation-specific control blocks you may have created.

A single exit point effects this exit. The exit point, at label FGDSX25, can give control to your exit routine just after the FSMGETDS routine in HASPFSSM reads the JCT for the job owning the JOE from which a data set will be selected (except if cancelled on a setup request) for assignment to a functional subsystem application (FSA).

ENVIRONMENT: Functional subsystem (HASPFSSM).

POINT OF PROCESSING:

This exit is taken from the functional subsystem address space (HASPFSSM).

Exit point FGDSX25, receives control after the JCT has been read into storage, occurs during JIB initialization processing in the FSMGETDS routine of HASPFSSM if the JCT read was successful and prior to initialization of the job separator page area (IAZJSPA) with fields from the JCT. The JCT read belongs to the job owning the JOE from which data set(s) will be selected for assignment to the FSA via the functional subsystem interface (FSI) GETDS function. Exit point FGDSX25 occurs prior to the invocation of exit point 23 (JSPA modification).

REGISTER CONTENTS WHEN CONTROL IS PASSED TO THE EXIT ROUTINE:

R0	A code passed to your routine by JES2 **0** Indicates that the JCT has been read from spool **4** Indicates that the JCT will be written to spool
R1	Address of the JCT
R2-R10	N/A
R11	Address of the HFCT
R12	N/A
R13	Address of an OS-style save area
R14	Return address
R15	Entry address

REGISTER CONTENTS WHEN CONTROL IS PASSED BACK TO JES2:

R0-R1 N/A

R15 A return code

RETURN CODES:

0 Tells JES2 that if there are any additional exit routines associated with this exit, call the next consecutive exit routine. If there are no other exit routines associated with this exit, continue with normal processing, which is determined by the particular exit point from which the exit routine was called.

4 Tells JES2 that even if there are additional exit routines associated with this exit, ignore them; continue with normal processing, which is determined by the particular exit point from which the exit routine was called.

JOB EXIT MASK: This exit is subject to job exit mask suppression.

MAPPING MACROS NORMALLY REQUIRED FOR THIS EXIT AT ASSEMBLY: $HASPEQU, $JCT, $HFCT, ETP, FSIP

OTHER PROGRAMMING CONSIDERATIONS:

1. Be sure your exit routines reside in common storage.

2. The $SAVE and $RETURN services are available in the FSS environment.

3. The service routines provided in the HASPFSSM module may be used within your exit routine. In particular, the $BUFIO and $BUFCK services can be used to perform spool I/O. Also, the cell pool services, $GETBLK and $RETBLK can be used to acquire save areas and other predefined storage cells dynamically. You are responsible for returning all storage cells explicitly acquired.

4. This exit must be linkedited to SYS1.LPALIB. This can be done separately or with HASPSSSM. **Do not linkedit this exit with HASPFSSM.**

5. *Note on recovery:* No recovery is in effect when this exit is taken. As with every exit, you should provide your own recovery within your exit routine. The $ESTAE facility is inoperative within the FSS execution environment, rather the MVS ESTAE facility must be used to provide recovery. Also note that the FSS may have recovery routines in effect and that these depend on the FSS implementation.

Exit 26: Termination/Resource Release

FUNCTION:

This exit allows you to free resources obtained during previous user exit routine processing at any JES2 termination. At a JES2 termination (that is, $P JES2 command, JES2 initialization termination, or an abend), Exit 26 receives control to free whatever resources your exit routines continues to hold. To control the release of resources, this exit permits access to the termination recovery communication area (TRCA) and the HASP communications table (HCT). With such access available, your installation is provided sufficient flexibility to withdraw or free all services and resources you may have previously acquired. This exit can also be used to permit your installation to modify the termination options and edit operator responses to those options.

ENVIRONMENT: JES2 main task (Termination) -- JES2 dispatcher disabled

POINT OF PROCESSING:

This exit is taken from HASPTERM during JES2 termination processing ($HEXIT).

At a JES2 termination, the operator receives the message $HASP098 ENTER TERMINATION OPTION. Following the operator response but prior to response processing, this exit gains control. Exit processing completes, and upon return from the exit, processing continues with the scanning of the operator response to the $HASP098 message.

REGISTER CONTENTS WHEN CONTROL IS PASSED TO THE EXIT ROUTINE:

R0	A code passed to your routine by JES2 **0** Indicates that Exit 26 is invoked for the first time **4** Indicates that Exit 26 is invoked for other than the first time
R1	Address of the JES2 main task TRCA
R2-R10	N/A
R11	Address of the HCT
R12	N/A
R13	Address of the HASPTERM PCE
R14	The return address
R15	The entry address

REGISTER CONTENTS WHEN CONTROL IS PASSED BACK TO JES2:

R0	A code passed to your routine by JES2 **0** Indicates that Exit 26 is invoked for the first time **4** Indicates that Exit 26 is invoked for other than the first time
R1	Address of the JES2 main task TRCA
R2-R10	N/A
R11	Address of the HCT
R12	N/A
R13	Address of the HASPTERM PCE - (this is a special PCE located in HASPTERM)
R14	The return address
R15	A return code

RETURN CODES:

0 Tells JES2 that if there are additional exit routines associated with this exit, call the next consecutive exit routine. If there are no other exit routines associated with this exit, continue with normal processing, which is determined by the particular exit point from which the exit routine was called.

4 Tells JES2 that even if there are additional exit routines associated with this exit, ignore them; continue with normal processing, which is determined by the particular exit point from which the exit routine was called.

JOB EXIT MASK: This exit point is not subject to job exit mask suppression.

MAPPING MACROS NORMALLY REQUIRED FOR THIS EXIT AT ASSEMBLY: $ERA, $HASPEQU, $HCT, $MIT, $PCE, $SVT, $TRCA

OTHER PROGRAMMING CONSIDERATIONS:

1. Note on recovery: Exit 26 **is protected** by an ESTAE routine. If an error occurs during Exit 26 processing in your code, the ESTAE issues message $HASP082 USER EXIT 26 ABEND to the operator. The ESTAE provides an SDUMP (if possible), returns control to JES2 termination processing ($HEXIT), and proceeds with normal termination. If this ESTAE does receive control, JES2 does not permit Exit 26 to receive control again.

Chapter 4: JES2 Programmer Macros

Chapter 4. JES2 Programmer Macros

JES2 programmer macros are provided for your use in your installation-exit routines. Only an experienced JES2 system programmer should attempt to use the macros; that is you should have advanced knowledge of JES2 internals. Changes to the structure of JES2 can cause incompatible changes to existing macros.

Macro Overview

The following JES2 control service programs provide a comprehensive set of services that aid the JES2 processors in performing their respective tasks in an efficient manner without burdening the programmer with needless detail. These services are requested by the processor through the use of JES2 macro instructions. The macros should not be used in code executing outside the control of the JES2 dispatcher unless stated in the description of the individual macro instruction.

- General storage management: Provide for the acquisition and release of JES2 buffers
- Work area management services: Provide for the acquisition and return of work areas that are chained off the processor control element (PCE)
- Virtual page service macros: Provide for the acquisition and release of virtual pages
- Job queue services: Provide for the alteration of job queues
- Direct-access space services: Provide for the allocation and deallocation of JES2 direct-access storage space
- Unit services: Provide for the acquisition and release of JES2 input/output units
- Input/output services: Provide communication with operating system input/output supervisor
- Console services: Provide all communication with the operator and manipulate associated buffer resources
- Time services: Provide for the setting and interrogation of the interval timer

- Synchronization services: Provide synchronization and communication between JES2 processors, the JES2 dispatcher, and the operating system
- System management facilities services: Provide the processors with an interface to the MVS SMF routines
- Installation exit services: Provide the $EXIT macro that is used to define exit points in JES2
- Debug services: Provide facilities for aid in debugging JES2
- Error services: Provide a uniform way of processing detected errors
- Recovery processing aids: Provide macros to aid in recovery processing
- Coding aid services: Provide the JES2 programmer with coding aids not usually available in the operating system, but useful in coding JES2 routines
- Print/punch output services: Provide macros used to define separator pages and create peripheral data information records
- Job output services: Provide macros used for job output services
- Initialization services: Provide macros to generate initialization statement (and parameter) tables and checkpoint information tables
- Verify services: Provide facilities to build control block verification tables to verify spool-resident control blocks
- Table services: Provide facilities for scanning JES2 initialization statements, dynamically creating control blocks for DCTs, PCEs, and DTEs.
- Miscellaneous services: Provide miscellaneous services such as modify the current JESNEWS data set and switch addressing mode

Some of the above services are provided by inline code expansion wherever the macro instruction is used. The remaining services are provided by routines that are integral parts of the control service programs. For more information about these routines, refer to Chapter 3. Code generated wherever the macro instruction is used links to these routines. At execution time, the macro expansion passes information to the control program routine to specify the exact nature of the service to be performed. This information is broken down into parameters and, in general, is passed to the routine through general purpose registers called parameter registers.

The macro expansion can contain load instructions (LA, L, LH, etc.) that load parameters in parameter registers, and it can contain instructions (LR,...) that load parameter registers from registers loaded by the processor. The processor can also load parameters directly. Registers 1 and 0 are generally used as parameter registers.

Each parameter resulting from the expansion of a macro instruction is either an address or a value. An address parameter is a standard 31-bit address. Any

exception to this rule will be stated in the individual macro instruction description.

An address parameter is always an effective address. The control service program is never given a 16-bit or 20-bit explicit address of the form D(B) or D(X,B) and then required to form an effective address. When an effective address is to be resolved, it is formed either by the macro expansion or before the macro instruction is issued.

A value parameter is a field of data other than an address. It is of variable length and is usually in the rightmost bits of a parameter register. The value parameter always has a binary format. The leftmost unused bits in the parameter register should contain all zeros. Any exception to this rule is stated in the individual macro instruction description.

Certain value parameters can be placed in a register along with another parameter, which can either be an address or a value parameter. In this case, a value parameter is in other than the rightmost bits. Two or more parameters in the same register are called packed parameters.

Parameters are specified by operands in the macro instruction. An address parameter can result from a relocatable expression or, in certain macro instructions, from an implicit or explicit address. A value parameter can result from an absolute expression or a specific character string. Address and value parameters can both be specified by operands written as an absolute expression enclosed in parentheses. This operand form is called register notation. The value of the expression designates a register into which the specified parameter must be loaded by the processor before the macro instruction is issued. The contents of this register are then placed in a parameter register by the macro expansion.

The programmer writes an operand in a JES2 macro instruction to specify the exact nature of the service to be performed. Operands are of two types, positional operands and keyword operands.

A positional operand is written as a string of characters. This character string can be an expression, an implied or explicit address, or some special operand form allowed in a particular macro instruction. Positional operands must be written in a specific order. If a positional operand is omitted and another positional operand is written to the right of it, the comma that would normally have preceded the omitted operand must be written. This comma should be written only if followed by a positional operand; it need not be written if followed by a keyword operand or a blank.

In the following examples, EX1 has three positional operands. In EX2, the second of three positional operands is omitted but must still be delimited by commas. In EX3, the first and third operands are omitted; no comma need be written to the right of the second operand.

```
EX1       $EXAMP A,B,C
EX2       $EXAMP A,,C
EX3       $EXAMP ,B
```

A keyword operand is written as a keyword immediately followed by an equal sign and an optional value.

A keyword consists of one through seven letters and digits, the first of which must be a letter. It must be written exactly as shown in the individual macro instruction description.

An optional value is written as a character string in the same way as a positional operand.

Keyword operands can be written in any order, but they must be written to the right of any positional operands in the macro instruction.

In the following examples, EX1 shows two keyword operands. EX2 shows the keyword operands written in a different order to the right of any positional operands. In EX3, the second and third positional operands are omitted; they need not be delimited by commas, because they are not followed by any positional operands.

```
EX1       $EXAMP KW1=X,KW2=Y
EX2       $EXAMP A,B,C,KW2=Y,KW1=X
EX3       $EXAMP A,KW1=X,KW2=Y
```

Certain operands are required in a macro instruction if the macro instruction is to make a meaningful request for a service. Other operands are optional and can be omitted. Whether an operand is required or optional is indicated in the individual macro instruction description.

Basic Notation Used To Describe Macro Instructions

JES2 macro instructions are presented in this section by means of macro instruction descriptions, each of which contains an illustration of the macro instruction format. This illustration is called a format description. An example of a format description is as follows:

```
[symbol]  $EXAMP  name1-value mnemonic,name2-CODED VALUE,    c
                  KEYWD1=value mnemonic,                     c
                  KEYWD2=CODED VALUE,                        c
                  KEYWD3=(label,value)                       c
```

Operand representations in format descriptions contain the following elements:

- An operand name, which is a single mnemonic word used to refer to the operand. In the case of a keyword operand, the keyword is the name. In the case of a positional operand, the name is merely a reference. In the above format description, name1, name2, KEYWD1, and KEYWD2 are operand names.

- A value mnemonic, which is a mnemonic used to indicate how the operand should be written if it is not written as a coded value. For example, addr is a value mnemonic that specifies that an operand or optional value is to be written as either a relocatable expression or register notation.

- A coded value, which is a character string that is to be written exactly as it is shown. For example, RDR is a coded value.

- Parentheses are always required around a list of specifications or values specified for a keyword with multiple values, such as KEYWD3=. These parentheses are optional if only one value is coded or the keyword is allowed to default.

The format description also specifies when single operands and combinations of operands should be written. This information is indicated by notational elements called *metasymbols*. For example, in the preceding format description, the brackets around symbol indicate that a symbol in this field is optional.

Operand Representation

Positional operands are represented in format descriptions in one of two ways:

- By a three-part structure consisting of an operand name, a hyphen, and a value mnemonic, for example: name1-addr
- By a three-part structure consisting of an operand name, a hyphen, and a coded value, for example: name1-RDR.

Keyword operands are represented in format descriptions in one of two ways:

- By a three-part structure consisting of a keyword, an equal sign, and a value mnemonic, for example: KEYWD1=addr
- By a three-part structure consisting of a keyword, an equal sign, and a coded value, for example: KEYWD1=RDR

The most significant characteristic of an operand representation is whether a value mnemonic or coded value is used; these two cases are discussed next.

Operands with Value Mnemonics

When a keyword operand is represented by:

```
KEYWORD=value mnemonic
```

the programmer first writes the keyword and the equal sign and then a value of one of the forms specified by the value mnemonic.

When a positional operand is represented by:

```
name-value mnemonic
```

the programmer writes only a value of one of the forms specified by the value mnemonic. The operand name is merely a means of referring to the operand in the format description; the hyphen simply separates the name from the value mnemonic. Neither is written.

The following general rule applies to the interpretation of operand representations in a format description; anything shown in uppercase letters must be written exactly as shown; anything shown in lowercase letters is to be replaced with a value provided by the programmer. Thus, in the case of a keyword operand, the keyword and equal sign are written as shown, and the value mnemonic is

replaced. In the case of a positional operand, the entire representation is replaced.

The value mnemonics listed below specify most of the allowable operand forms that can be written in JES2 macro instructions. Other value mnemonics, which are rarely used, are defined in individual macro instruction descriptions.

- symbol: The operand can be written as a symbol.
- relexp: The operand can be written as a relocatable expression.
- addr: The operand can be written as (1) a relocatable expression or (2) register notation designating a register that contains an address. The designated register must be one of the registers 2 through 12, unless special notation is used.
- addrx: The operand can be written as (1) an indexed or nonindexed implied or explicit address or (2) register notation designating a register that contains an address. An explicit address must be written in the RX form of an assembler language instruction.
- adval: The operand can be written as (1) an indexed or nonindexed implied or explicit address or (2) register notation designating a register that contains a value. An explicit address must be written in the RX form of an assembler language instruction.
- absexp: The operand can be written as an absolute expression.
- value: The operand can be written as (1) an absolute expression or (2) register notation designating a register that contains a value.
- text: The operand can be written as a character constant as in a DC data definition instruction. The format description shows explicitly if the character constant is to be enclosed in apostrophes.
- code: The operand can be written as one of a large set of coded values; these values are defined in the macro instruction description.

Coded Value Operands

Operands that are not represented in format descriptions by value mnemonics are represented by one or more uppercase character strings that show exactly how the operand should be written. These character strings are called *coded values*, and the operands for which they are written are called *coded value operands.*

A coded value operand results in either a specific value parameter or a specific sequence of executable instructions.

If a positional operand can be written as any one of two or more coded values, all possible coded values are listed in a format description and are separated by vertical strokes indicating that only one of the values is to be used.

Metasymbols

Metasymbols are symbols that convey information about how to code such as whether keywords and operands are optional or required when coding macro instructions; they are never written in the coded macro. They show the programmer how and when an operand should be written. The metasymbols used in this section are:

Metasymbol Meaning and Use

{ } Braces – denote grouping of alternative operands, one of which must be selected. For example:

```
{YES}
{NO }
```

[ ] Brackets – denote optional operands. Anything enclosed within brackets can be either omitted or written once in the macro instruction. For example:

```
[USE=code]
```

In the example above, the keyword and its operand can either be written or omitted; its use is optional.

_ Underscore – denotes the JES2 default if the particular keyword is not coded. For example:

```
WAIT={YES}
     {NO }
```

In the example above, YES is the default. To override the default, you must code WAIT=NO. The WAIT= keyword is therefore optional; any keyword with a default is enclosed within brackets in the syntax diagrams throughout this chapter, similar to the next example.

Metasymbols are nested in almost any combination throughout the macro instruction descriptions that follow. Whether any set of keywords and operands are optional or required is determined by the outermost set of metasymbols. For example:

```
[ ,WAIT={YES} ]
       {NO }
```

The entire keyword/operand statement is optional, but if you do code the WAIT= keyword, the only valid options are either WAIT=YES or WAIT=NO.

Uppercase operands must be coded as written in the syntax diagrams. Also, punctuation such as commas, parentheses, and single quotes are not metasymbols; if present in the syntax diagrams they **must** be coded. Operands in lowercase are **not** to be coded as written; they denote variables that are explained in the description of the particular keyword for the macro instruction.

Special Register Notation

Many JES2 macro instruction keywords allow you to code a register as a valid specification. If you do code a register (for example, R0 or R15), be certain to enclose the symbol representing that register in parenthesis. A symbol enclosed in parenthesis is called register notation (for example, (R0) or (R1)).

If an operand of a JES2 macro instruction is written using register notation, the resulting macro expansion loads the parameter contained in the designated register into either parameter register 1 or parameter register 0.

For example, if an operand is written as (R15) and if the corresponding parameter is to be passed to the control program in register 1, the macro expansion would contain the instruction:

```
LR   R1,R15
```

Prior to macro expansion, the processor can load parameter registers; this is called **pre-loading** When preloading a parameter register, use the JES2 equated symbols for register 0 or 1 (that is, R1 or R0) to indicate to the macro which registers contain values to be passed. If you do not use the JES2 equated symbols, you will cause the generation of an extra instruction.

For example, RONE, an absolute symbol equated to R1, should not be specified on the macro statement if the register required is R1. The macro will not recognize RONE as register 1 and will attempt to load the parameter register R1 from the RONE specification with the following redundant instruction:

```
LR    R1,RONE
```

The format description shows whether special register notation can be used, and for which operands. This is demonstrated by the following example:

```
[symbol]    $EXAMP  { abc-addrx } , { def-addrx }
                    {  (R1)    }   {  (R0)    }
```

Both operands can be written in the addrx form, and therefore can be written using register notation. Ordinary register notation indicates that the parameter register should be loaded from the designated register by the macro expansion. The format description also shows that the abc operand can be written as (R1), and the def operand can be written as (R0). If either of these special notations is used, the processor must have loaded the designated parameter register before the execution of the macro instruction.

Register Transparency

In general the following registers cannot be considered transparent across a JES2 macro expansion and the associated link to the control service program:

R14
R15
R0
R1

All other registers are transparent unless specifically stated in the individual macro instruction description.

Macro Descriptions

Figure 4-1 summarizes the available JES2 programmer macros by the function they perform. Following Figure 4-1 are the individual macros descriptions, presented in alphabetical order.

Warning: A number of JES2 macros located in data set SYS1.HASPSRC are not documented in this book. These macros are not intended for general use. If you use these undocumented macros, be aware that unpredictable results may occur.

Macro Selection Table

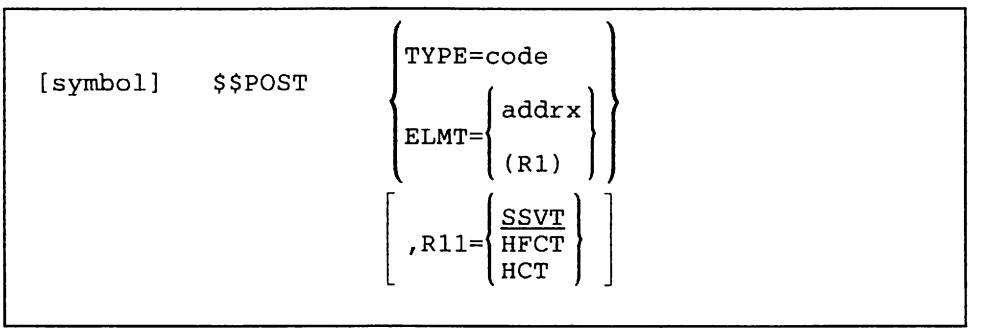
General Function	JES2 Programmer Macros
General Storage Management	$BFRBLD $FRECEL $FREEBUF $FREMAIN $GETBUF $GETCEL $GETMAIN
Work Area Management Services	$GETWORK $RETWORK
Virtual Page Services	$PGSRVC
Job Queue Services	$JCAN $QADD $QGET $QJIX $QLOC $QMOD $QPUT $QREM
Direct-Access Space Services	$BLDTGB $PURGE $TGMSET $TRACK
Unit Services	$ALLOC $FREUCBS $FREUNIT $GETUCBS $GETUNIT
Input/Output Services	$EXCP $EXTP $JCTIO
Console Services	$$WTO $$WTOR $CWTO $DOM $FRECMB $GETCMB $MID $MSG $WTO
Time Services	$STCK $STIMER $TTIMER
Synchronization Services	$ACTIVE $DORMANT $FRELOK $GETLOK $POST $POSTQ $$POST $QSUSE $RELEASE $RESERVE $WAIT $XMPOST
System Management Facility Services	$GETSMFB $QUESMFB
User Exit Services	$EXIT
Debug Services	$COUNT $SDUMP $TRACE
Error Services	$DISTERR $ERROR $IOERROR
Recovery Processing Services	$ESTAE $SETRP
Coding Aid Services	$CALL $ENTRY $MODEND $MODULE $RESTORE $RETSAVE $RETURN $SAVE $STORE $VFL
Print/Punch Output Services	$PBLOCK $PRPUT $SEPPDIR
Job Output Services	$#ADD $#ALCHK $#BLD $#CAN $#CHK $#GET $#JOE $#MOD $#PDBCAN $#PDBLOC $#POST $#PUT $#REM
Initialization Services	$STMTLOG
Table Services	$DCTTAB $DTETAB $GETABLE $PCETAB $SCANTAB $TIDTAB $VERTAB $WSTAB
Dynamic Service Access Services	$DCBDYN $DCTDYN, $DTEDYN $PCEDYN
Verify Services	$VERIFY $VERTAB
Scan Services	$SCAN $SCANB $SCANCOM $SCAND
Work Selection	$WSSETUP
Miscellaneous	$AMODE $CKPT $DEST $GETASCB

Figure 4-1. JES2 Macro Selection Table

$$POST - Post a JES2 Event Complete from Another Task

Use $$POST to post certain specific JES2 processors or resources by setting indicators that cause JES2 processors to begin executing.

Format Description



TYPE=

Specifies the resource that is to be posted. One of the following must be specified:

ABIT
Waiting for the next dispatcher cycle

ALOC
A dynamic allocation has completed

BUF
A JES2 buffer has been released

CKPTP
A checkpoint cycle has completed

CKPT
A JES2 checkpoint write has completed

CKPTW
A JES2 checkpoint should be written

CMB
A console message buffer has been released

CNVT
A converter has been released

FSS
A functional subsystem has completed FSS-level processing

HOPE
An output processor has been released

IMAGE
A UCS or FCB image has been loaded

JOE
A JOE has been released

JOT
A JES2 job output element has changed status

JOB
A JES2 job queue element has changed status

LOCK
A lock has been released

MAIN
Storage is available

PSO
A process SYSOUT request has been queued for the JES2 PSO processor(s)

PURGE
A JES2 job queue element (JQE) has been placed on the purge queue

PURGS
Purge resources from $PURGER have been released

RSV
A JES2 RESERVE has been satisfied

SMF
AN SMF buffer has been released

TRACK
A track group from the JES2 spooling data set has been released

UNIT
A device control table has been released

value
An installation-defined dispatcher resource name or number

ELMT=
Specifies the address of the element where the event indicator is to be set. Symbolic names for these indicator elements are as follows:

- SVTCOMM - Post command processor
- SVTJOB - Post execution processor
- SVTASYNC - Post asynchronous I/O processor
- SVTXSTIM - Post time excess processor
- SVTTIMER - Post timer processor
- SVTIRDR - Post all internal reader processors

- SVTTRPCE - Post trace logger
- SVTSPOOL - Post spool manager
- SVTMLLM - Post line manager

The corresponding processor control elements are posted by the JES2 dispatcher upon recognizing the post elements line in $$POST.

If register notation is used, the designated register must be loaded with the address of the element prior to executing this macro. Do not use register 2 for this address.

R11=

Specifies which of the three standard control blocks register 11 addresses as follows:

HCT

Register 11 addresses the beginning of the HASP communications table (HCT). The user must provide mapping of this table as well as the subsystem vector table (SSVT).

SSVT (default)

Register 11 addresses the beginning of the SSVT. The user must provide mapping of this table.

HFCT

Register 11 addresses the beginning of the HASP FSS communications table (HFCT).

If this operand is omitted, R11= address of SSVT is assumed.

Notes:

1. *The execution of this macro requires registers 0, 1, 2, 11, and 15.*
2. *This macro instruction should not be used when executing code that runs under control of the main JES2 task program request block. It is for use by subtasks, timer exit routines, and routines within the HASPSSSM module operating on behalf of other tasks.*
3. *Either TYPE or ELMT operands must be specified but not both.*
4. *If this macro is executed in the functional subsystem environment, the content of register 3 does not remain constant.*

Environment

- Subtask, user, and functional subsystem (HASPFSSM).
- WAIT cannot occur.

$$WTO - JES2 Subtask Write to Operator

Use $$WTO to initiate the display of an operator message from a JES2 subtask or during JES2 initialization or termination. The message is issued via an MVS execute-form WTO macro after supplying the JES2 command ID character.

$SWTO stores the message text within the message area; therefore, your program becomes non-reentrant after using this macro. Your program remains reentrant if the message area is acquired (via GETMAIN) and refreshed each time the macro is issued.

Format Description

```
[symbol] $$WTO    {message-addrx}
                  {(R1)        }
```

message
Specifies the address of a list-form MVS WTO message. If register notation is used, the address must be loaded into the designated register before execution of this macro instruction. If descriptor code 1, 2, 3, or 11 was specified, the identification number (24 bits, right-justified) is returned in register 1. Otherwise, register 1 is left unchanged.

Environment

- Subtask or main task.
- $WAIT cannot occur.

$$WTOR - JES2 Subtask Write to Operator with Reply

Use $$WTOR to initiate the display of an operator message, requiring a reply, from a JES2 subtask. The message is issued via an MVS execute-form WTOR macro instruction after supplying the JES2 command ID character.

$$WTOR stores the message text within the message area; therefore, your program becomes non-reentrant after using this macro. Your program remains reentrant if the message area is acquired (via GETMAIN) and refreshed each time the macro is issued.

Format Description

```
[symbol] $$WTOR    { message-addrx }
                   { (R1)          }
```

message
Specifies the address of a list-form MVS WTOR message. If register notation is used, the address must be loaded into the designated register before execution of this macro instruction. The identification number (24 bits, right-justified) is returned in register 1.

Notes:

1. *From JES2 subtasks, HASPINIT and HASPTERM, it is the responsibility of the issuer of this macro instruction to issue a WAIT macro instruction, the ECB of which will be posted when the operator has replied to the message.*

2. *From the main task it is the responsibility of the issuer of this macro instruction to issue a $WAIT with the XECB option.*

Environment

- Subtask.
- Main task (during JES2 initialization and termination).
- $WAIT cannot occur.

$#ADD - Add a Work/Characteristics JOE Pair to the JOT

Use $#ADD to add a job output element (JOE) to the appropriate job output table (JOT) queue and add the characteristics JOE to the characteristics queue.

Format Description

```
[symbol]   $#ADD       WORK={addrx}
                            {R0  }
                      ,CHAR={addrx}
                            {R1  }
```

WORK=
Specifies the address of a prototype work JOE that is to be added to the JOT.

CHAR=
Specifies the address of a prototype characteristics JOE that is to be merged into the characteristics queue.

Notes:

1. *The condition code upon exit from the $#ADD macro instruction is set to reflect the following status of the request:*

   RC=0
   The service was successfully performed.

   RC=4
   The JOT is full; the request must be tried again later.

2. *The queue to which the work JOE is added is determined by the current class of the JOECURCL and JOEROUT fields of the JOE or by the offload status in the JOEFLAG2 field.*

3. *If the JOECURCL of the work JOE is invalid, $#ADD terminates JES2 with a $ERROR, CATASTROPHIC ABEND J07 INVALID SYSOUT CLASS FOUND.*

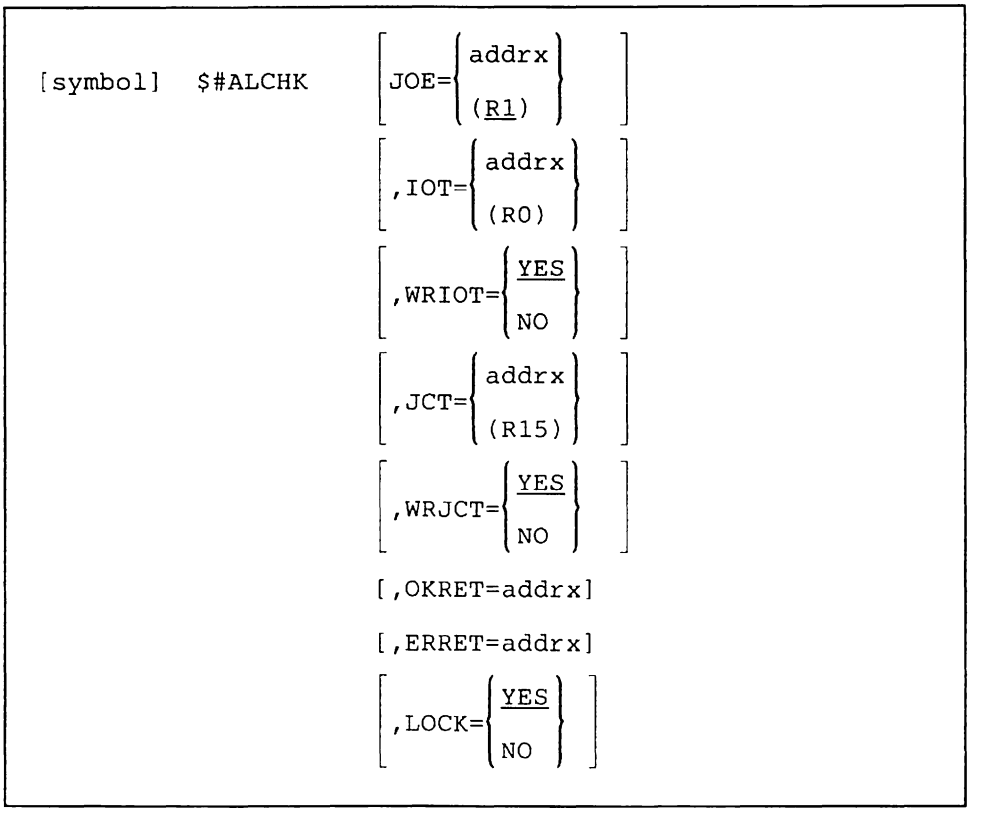
Environment

- Main task.
- $WAIT can occur.

$#ALCHK - Obtain a Spool Record for Output Checkpointing

Use $#ALCHK to obtain a spool record for output checkpointing.

Format Description



JOE=

Specifies the address of a work JOE. The spool record for this work JOE is to be obtained. If register notation is used, the designated register must contain the address of the work JOE prior to the execution of the macro. If this operand is omitted, register 1 is assumed.

IOT=

Specifies the address of the IOT that is to be used for allocating the spool. If register notation is used, the designated register must contain the address of the IOT prior to the execution of the macro. If this operand is omitted, the IOT is read from the spool. An indication is set in the generated inline parameter list whether or not the IOT was passed.

WRIOT=

Specifies whether the IOT should be written back out to the spool after $TRACK obtains the spool record. The IOT is marked as an allocation IOT (IOT1ALOC).

JCT=
Specifies the address of the JCT. If this operand is omitted, the JCT is read from the spool. If register notation is used, the designated register must contain the address of the JCT prior to the execution of the macro. An indication whether or not the JCT was passed is set in the generated inline parameter list.

WRJCT=
Specifies whether or not the JCT is to be written back to the spool. If WRJCT=YES is specified and the JQE indicates that the job is still in execution, the JCT is not written back to the spool. Otherwise, it is.

OKRET=
Specifies the address of a routine that is to receive control if the return code is zero.

ERRET=
Specifies the address of an error routine that is to receive control if the return code is not zero.

LOCK=
Specifies whether or not the job lock is to be obtained. LOCK=NO indicates that a wait will occur for IOT/JCT serialization.

Environment

- Main task.
- $WAIT can occur.

$#BLD - Format JOEs

Use $#BLD to format a pair of work and characteristics job output elements (JOEs) in the provided work area.

Format Description

```
[symbol]   $#BLD          JOES={addrx}
                               {(R1) }
                        ,PDDB={addrx}
                              {(R0) }
                        ,JQE={addrx}
                             {(R15)}
```

JOES=
Specifies the address of the work area that is to be formatted into work and characteristics JOEs. If register notation is used, the address must be loaded into the designated register before the execution of this macro instruction.

PDDB=
Specifies the address of the peripheral data definition block (PDDB) whose contents are used to format the work and characteristics JOEs. If register notation is used, the address must be loaded into the designated register before the execution of this macro instruction.

JQE=
Specifies the location of the job queue element (JQE) to which the PDDB belongs. The location is specified as the offset in bytes of the JQE from the start of the job queue. If an address is used, it specifies the address of a fullword whose two right-most bytes contains the JQE offset. If a register is used, the JQE offset must have been loaded into the designated register before the execution of this macro instruction.

Environment

- Main task.
- $WAIT can occur.

$#CAN - Cancel All Work Items Not Currently Being Processed for a Specific Job

Use $#CAN to remove from the JOT all available work items for a job. Work items removed are not processed by any output processor.

Format Description

```
[symbol]   $#CAN        JQE={addrx}
                            {(R1)}
```

JQE=
Specifies the address of the job queue element for which all JOT entries are to be purged.

Note: The specified job is purged from the system if all of its output requirements are removed and its current queue position is $HARDCPY.

Environment

- Main task.
- $WAIT can occur.

$#CHK - Process print/punch checkpoint spool I/O

Use $#CHK to process print/punch checkpoint spool I/O.

Format Description

```
[symbol]   $#CHK        TYPE={READ  }
                         {WRITE }
                       [,BUF={addrx}]
                         {(R1)  }
                       ,JOE={addrx}
                         {(R0)  }
                       [,WAIT={YES}]
                         {NO  }
                       [,DCT={PPPDADCT}]
                         {addrx        }
                       [,OKRET=addrx]
                       [,ERRET=addrx]
```

TYPE=

Specifies whether the operation is a checkpoint read or write. The read or write indication is placed in an inline parameter list (CHK1RD for read and CHK1WR for write). This operand must be specified or an error occurs at assembly time.

BUF=

Specifies the address of the checkpoint I/O buffer. If register notation is used, the designated register must contain the address of the buffer. If this operand is omitted, BUF=(R1) is assumed.

JOE=

Specifies the address of the work JOE associated with the spool I/O. If register notation is used, the designated register must contain the address of the work JOE prior to execution of the macro. This keyword is required.

WAIT=

Specifies whether or not to wait for the spool I/O to complete and whether or not to set a return code. WAIT=YES indicates to wait for the I/O to complete and to set a return code. WAIT=NO indicates to not wait for the I/O to complete and to not set a return code. If this operand is omitted, WAIT=YES is assumed.

Note: Specifying WAIT=NO nullifies the use of both the OKRET= and ERRET= keywords.

DCT=

Specifies the DCT address needed to perform the spool I/O. If this operand is omitted, PPPDADCT is used.

OKRET=

Specifies the address of a routine that is to receive control if the return code is zero. *NOTE*: Specifying WAIT=NO nullifies the use of OKRET.

ERRET=

Specifies the address of an error routine that is to receive control if the return code is not zero. *NOTE*: Specifying WAIT=NO nullifies the use of ERRET. Also, ERRET take precedence over OKRET when both operands are specified.

Environment

- Main task.
- $WAIT can occur.

$#GET - Search the JOT Class Queues for an Output Element which Matches the Requesting Specification

Use $#GET to search the JOT for output work.

Format Description

```
[symbol]    $#GET       DCT={addrx}
                        {(R1)  }
                       [,HAVE={YES}]
                        [     {NO }]
                       [,NONE=relexp]
                       [,FOUND=relexp]
                       [,CHAIN={YES}]
                        [      {NO }]
                       [,TYPE={NET}]
                        [     {WS }]
```

DCT=
Specifies the address of the JES2 device control table (DCT) for the requesting processor. The device setup fields in the DCT are used in the process of selecting work.

HAVE=
Specifies that if a selectable JOE is found it is not to be assigned to the requester (NO), or if a selectable JOE is found it is to be assigned to the requester (YES).

NONE=
Specifies a label or an address in a register to branch to if there are no selectable JOEs found.

FOUND=
Specifies a label or address in a register to branch to if a selectable JOE is found.

CHAIN=
Specifies either that all (YES) eligible job-related JOEs are to be chained to the transmitter chain and returned to the caller, or only the first (NO) eligible JOE is to be returned to the caller.

TYPE=

Specifies that for this $#GET call, either the network queue (NET) is searched or the work selection (WS) algorithm is used.

Environment

- Main task.
- $WAIT can occur.

$#JOE - Find and Validate Queue

Use the $#JOE to cause the output service processors to generate the address for the head of a specified queue. You can then reference the first JOE on the queue through the JOENEXT field. You must establish addressability to the JOT before you use this macro instruction.

Format Description

```
[symbol]    $#JOE        Q={addrx    }
                          {JOTNTWKQ }
                         ,R=Rn
                         [,INV=relexp]
                         [,DEST={LOC }]
                                {RMT }
```

Q=
Specifies the address of the storage location containing the requested SYSOUT class, or the address of the offset into the network queue (JOTNTWKQ).

R=
Specifies the register (Rn) into which the address of the desired queue head is to be loaded.

INV=
Specifies the label of the statement to which control is to be returned if the requested queue is invalid. If you omit this parameter, no check is made to ensure the validity of the queue. **Do not code this operand if you also specify Q=JOTNTWKQ.**

DEST=
Specifies the destination queue within the class specified by the Q= operand. Possible values are as follows:

LOC (default)
The local class queue.

RMT
The remote class queue. **Do not code DEST=RMT if you specified Q=JOTNTWKQ.**

Environment

- Main task.
- $WAIT cannot occur.

$#MOD - Move a Work JOE from One Queue to Another in the JOT

Use $#MOD to remove a work JOE from the queue it is currently on and to place it on the proper queue as determined by its routing (JOEROUT) or SYSOUT class (JOECURCL). $#MOD should be issued after a JOE's queue status has been changed.

Format Description

```
[symbol]    $#MOD          JOE={addrx }
                            {(R1)  }
```

JOE=
Specifies the address of the work JOE that is to be moved from one queue to another.

Environment

- Main task.
- $WAIT can occur.

$#PDBCAN - Cancel a JOE's Nonheld Data Sets

Use $#PDBCAN to mark a JOE's nonheld data sets as non-printable, so that when the held data is later released by the process SYSOUT processor (PSO), only held data sets can be gathered together under the original JOE name.

Format Description

```
[symbol]    $#PDBCAN        IOT={relexp}
                            ,PDDB={addrx}
                                  {(Rn)}
```

IOT=
Specifies the input/output table track address.

PDDB=
Specifies the address of the PDDB associated with held data sets for this job.

Environment

- Main task.
- $WAIT can occur.

$#PDBLOC - Find a PDDB by Data Set Key

Use $#PDBLOC to locate a PDDB using a specific data set key as the search argument.

Format Description

```
[symbol]    $#PDBLOC    IOT={addrx}
                         {(R1)  }
                        ,KEY={relexp}
                             {(R0)  }
```

IOT =
Specifies the address of the in storage IOT chain from which a PDDB is to be located. If register notation is used, the address of the IOT must be loaded in the designated register before the execution of the $#PDBLOC.

KEY =
Specifies the data set key search argument to be used to locate the PDDB. If register notation is used, the data set key must be loaded into the designated register before execution of $#PDBLOC.

Note: All IOTs must be in storage. The IOT whose address is supplied must be in storage.

Environment

- Main task.
- $WAIT cannot occur.

$#POST - Post Output Device Processors

Use $#POST to post device processors that are waiting for work associated with specific output devices. $#POST ensures that when a new piece of work becomes available for processing, only those processors associated with the devices eligible to select the work are posted. JES2 uses $#POST when a JOE is added or returned to the JOT. $#POST is also used 1) when a message is spooled for a remote, 2) when a node's remote or local output device becomes available for use, 3) when a console or printer is added to notify when a node, remote processor, or device becomes available for use, or 4) when a new path becomes available to a node.

Format Description

```
[symbol]    $#POST      TYPE={JOE|JQE|MSG|XMIT}
                        [,ADDR={addrx|(R1)}]
                        [,MASPOST={YES|NO}]
```

TYPE=

Specifies what type of $#POST to issue. You can specify one of four types. JOE specifies a work JOE $#POST. JQE specifies a JQE and associated work JOE(s) $#POST. MSG specifies a spooled message $#POST. XMIT specifies a SYSOUT transmitter $#POST. You must specify this operand; there is no default.

ADDR=

Specifies an address. The address depends on the TYPE selected. For TYPE=JOE, ADDR is the address of the work JOE that is to be $#POSTed. For TYPE=JQE, ADDR is the address of the JQE whose work JOE(s) are to be $#POSTed. For TYPE=MSG, ADDR is the address of the route code for the remote printers or consoles that are to be $#POSTed. For TYPE=XMIT, ADDR is the address of the line DCT associated with the SYSOUT transmitter(s) that is to be $#POSTed; if this address is specified as zero, then all SYSOUT transmitters that are waiting are $#POSTed.

MASPOST=

Specifies whether the work JOE(s) that are to be $#POSTed should have their JOE post flags reset so that the post is propagated to all members in a multi-access spool complex. MASPOST= is valid only when TYPE=JOE or TYPE=JQE is specified.

Notes:

1. *The MASPOST flag is passed in the first byte of the inline parameter list.*

2. *You need control of the checkpoint data set (obtained via $QSUSE) prior to issuing this macro.*

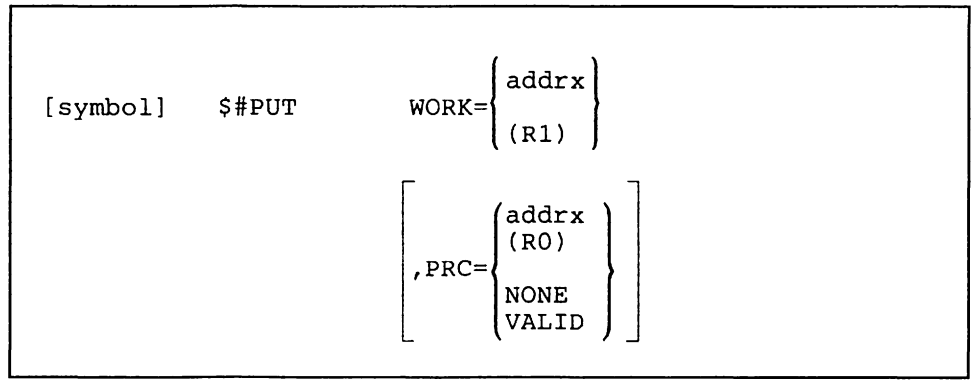
Environment

- Main task.
- $WAIT cannot occur.

$#PUT - Return an Unfinished Job Output Element (JOE) to the JOT for Later Processing

Use $#PUT in a processor to return a JOE to the JOT for later processing. Optionally, the status of the JOE is maintained for a warm start of the system or restart of the work.

Format Description



WORK=
Specifies the address of a work JOE that is to be returned to the JOT class queues for future selection.

PRC=
Specifies the address of a checkpoint buffer if the current status of the work item is to be stored. If PRC= is not specified or is specified as PRC=NONE, the work item is reset to reflect its initial entry status. If PRC=VALID is specified, no change is made to the current status of the work item.

Environment

- Main task.
- $WAIT can occur.

$#REM - Remove a Work/Characteristics JOE Pair from the JOT

Use $#REM to remove a work and characteristics JOE pair from the JOT after the output requirement they represent has been satisfied.

Format Description

```
[symbol]    $#REM    WORK=workjoe-addr
                     [,IOT=spinjoe-IOT addr]
                     [,PURGE={YES}]
                             {NO }
                     [,WAIT={YES}]
                            {NO }
```

WORK=
Specifies the address of a work JOE that is to be returned to the queue of free JOEs in the JOT. If the related characteristics JOE is not being shared by another work JOE, it is also returned to the free queue.

IOT=
Specifies the address of the spin IOT used to free the track groups used by spin data sets if the IOT is already in storage. If the IOT is not in storage (that is, not specified), it will be read.

PURGE=
Specifies whether (YES) or not (NO) to purge the track groups held for the spin IOT.

WAIT=
Specifies whether (YES) or not (NO) the $#REM service routine is permitted to $WAIT for IOT buffers if none are available. WAIT=YES is the default.

Note: The related job is purged from the system if all of its JOEs are removed and its current queue position is $HARDCPY.

Environment

- Main task.
- $WAIT can occur if WAIT=YES is specified.

$ACTIVE - Specify Processor is Active

Use $ACTIVE to indicate to JES2 that the associated processor is performing activities on behalf of the JES2 main task; this prevents JES2 from being cleanly withdrawn from the system (via $P JES2) when JES2 is processing a job or task.

Format Description

```
[symbol]    $ACTIVE        [R=Rn]
```

R=

Specifies the work register which is to be used by the $ACTIVE macro instruction. Do note enclose the register (R=) value in parenthesis. Register 1 is the default.

Notes:

1. *JES2 is considered active when the active count is greater than 0 ($DORMANT decreases the active count). When the active count is 0, JES2 issues $HASP099.*

2. *Do not use R=0.*

Environment

- Main task.
- $WAIT cannot occur.

$ALLOC - Allocate a Unit Record Device

Use $ALLOC to allocate a device to JES2.

Format Description

```
[symbol]     $ALLOC     { dct-addrx }
                        { (R1)      }
                        [,error-relexp]
```

dct
Specifies the address of the DCT to be allocated.

error
Specifies a location to which control is returned if the device (DCT) cannot be allocated. The condition code is set to reflect the allocation of the DCT as follows:

CC=0
The device could not be allocated.

CC≠0
The device was successfully allocated.

Environment

- Main task.
- $WAIT can occur.

$AMODE - Set the Addressing Mode

Use the $AMODE macro instruction to set MVS/370 (24-bit) and MVS/XA (31-bit) addressing modes. This macro can be specified when running JES2 on either an MVS/370 or MVS/XA system; however, if this macro specifies a switch to 31-bit mode when running on a MVS/370 system, no mode switch occurs and the macro instruction is ignored.

Format Description

```
[symbol]    $AMODE    { [ mode      ] }
                      { [ PUSHR=Rn  ] }
                      { [POPR=Rn]   }
                      ,R={ Rn   }
                         { R15  }
                      [,RELATED=char-string]
```

mode

Specifies the addressing mode to be used by the code that follows this macro until it is again specified. This is a positional parameter and must be specified if PUSHR= is also specified. Do not use this operand if POPR= is specified.

24

Specifies 24-bit addressing mode.

31

Specifies 31-bit addressing mode.

PUSHR=

Specifies a register to be used to store the current addressing mode. If mode is specified, this keyword is also required.

Note: Do not enclose the specified register in parenthesis.

POPR=

Specifies a register to be used to restore the previous addressing mode. The register specified here must have been previously loaded by a $AMODE mode PUSHR= instruction. Do not specify this keyword if mode and PUSHR= are specified.

Note: Do not enclose the specified register in parenthesis.

R=
Specifies a work register to be used by this macro instruction. Register 15 is the default.

Note: Do not enclose the specified register in parenthesis.

RELATED=
Specifies a character string used to self-document this macro instruction call. Any specification type valid for macro keywords can be used here. This field is useful for documenting the inline pairing of $AMODE macros.

Environment:

- Main task, subtask, user address space (HASPSSSM), and functional subsystem address space (HASPFSSM).
- waits cannot occur.

$BFRBLD - Construct a JES2 Buffer Prefix

Use $BFRBLD to construct an IOB or RPL in the front of a JES2 buffer. The IOB or RPL is used to read into or write from the data portion of the buffer.

Format Description

```
[symbol]    $BFRBLD    { buffer-addrx }
                       { (R1)         }
                       [,TYPE=type-code]
```

buffer

Specifies the address of a buffer in which the prefix (an IOB or an RPL) is to be constructed. If an address is used, it specifies a word in storage containing the buffer address.

If the notation is used, the buffer address must have been loaded into the designated register before the execution of this macro instruction.

TYPE=

Identifies the type of buffer and specifies whether an IOB or RPL is to be constructed at the beginning of the buffer, according to the type code as follows:

HASP (default)
A local buffer; an IOB is to be constructed

BSC
A TP buffer; an IOB is to be constructed

VTAM
A TP buffer; an RPL is to be constructed

PAGE
A local 4096-byte buffer; an IOB is to be constructed

PP
A local print/punch buffer; an IOB is to be constructed

Environment

- Main task.
- $WAIT cannot occur.

$BLDQC - Call the Quick Cell Build/Extend Routine.

Use $BLDQC to call the quick cell build/extend routine to build or extend a quick cell pool.

Format Description

```
[symbol]    $BLDQC    TYPE={type-code}
                        {(R0)      }
```

TYPE=

Specifies the type of quick cells to build.

type-code

specifies the type code as defined in the $QCTGEN macro for the quick-cell type to build. Quick cell types are defined as one of the following:

Quick cell Type-Code	Meaning
SAVE	Standard save area for MVS linkage conventions as described in *Supervisor Services*
JIB	JOE information block
BUF	Standard 4K buffer
RPL	Request parameter lists for GETDS processing
GETRC	Control block areas for GETRC processing

(R0)

Specifies the register that contains the type-code; if coded, be certain that the two low-order bytes of the register contain the quickcell type-code as defined in the $QCTGEN macro; the two high-order bytes must be zeroed.

The TYPE= keyword must be specified.

Environment

- Functional subsystem (HASPFSSM)
- MVS WAIT can occur

$BLDTGB - Queue TGBs to the HASPOOL Processor

Use $BLDTGB to build track group blocks (TGBs) and queue them off of the $SPOOLQ in the HCT. The TGB represents a bad track group for which the HASPSPOL processor attempts recovery.

Format Description

```
[symbol]        $BLDTGB      ADDR=addrx
                                    ,ID={TGM  }
                                        {MTTR }
```

ADDR=
Specifies the address of the track group map (TGM) that contains bad track groups or the MTTR (JES2 spool track address) of a single bad track group.

ID=
Specifies whether the ADDR= keyword specifies a TGM or MTTR.

Environment

- Main task.
- $WAIT cannot occur.

$BUFCK - Check Buffer I/O

Use $BUFCK to verify the completion of buffer I/O that was initiated by $BUFIO.

Format Description

```
[symbol]      $BUFCK    [BUF={addrx  }]
                        [    {(R1)   }]
                        [,ECB={addrx  }]
                        [     {(R0)   }]
                        [,ERRET=relexp]
```

BUF=
Specifies the address of the buffer that is to be checked for I/O completion. If register notation is used, the designated register must contain the address of the buffer prior to execution of the macro. If this keyword is omitted, BUF=(R1) is assumed.

ECB=
Specifies the address of the ECB to be used for the buffer I/O operation. If register notation is used, the designated register must contain the address of the ECB prior to the execution of the macro. If this operand is omitted, ECB=(R0) is assumed.

ERRET=
Specifies the address of the error routine that is to receive control if a nonzero return code is passed back.

Environment

- Functional subsystem (HASPFSSM)
- MVS WAIT can occur.

$BUFIO - Read or Write a Buffer

Use $BUFIO to read or write a buffer from or to spool. Note that an I/O request is initiated and control is returned without waiting for the completion of that I/O request.

Format Description

```
[symbol]   $BUFIO    TYPE={READ }
                          {WRITE}
                     [,BUF={addrx}  ]
                           {(R1) }
                     [,MTTR={addrx}  ]
                            {(R2) }
                     [,ECB={addrx}  ]
                           {(R0) }
                     [,ERRET=relexp]
```

TYPE=
Specifies whether the buffer I/O operation is to be a read or write. This operand must be specified or an error occurs at assembly time. If TYPE=WRITE is specified, BUF= must also be specified.

BUF=
Specifies the address of the buffer that is to be read or written to the spool. If you use register notation, the designated register must contain the address of the buffer before the execution of this macro. If TYPE=WRITE is specified, this keyword must also be specified. If TYPE=READ is specified and BUF= is not specified, an I/O buffer will be acquired and formatted; the address of the buffer is returned in register 1.

MTTR=
Specifies the spool track address of the record that is to be read or written. If an invalid MTTR is passed, a return code of 4 is returned. If you use register notation, the designated register must contain the address of the MTTR before the execution of this macro. If you omit this keyword, MTTR=(R2) is assumed.

ECB=
Specifies the address of the ECB to be used for this buffer I/O operation. If you use register notation, the designated register must contain the address of the ECB before the execution of this macro. If you omit this keyword, ECB=(R0) is assumed.

ERRET=
Specifies the relocatable expression that is the address of an error routine that gets control if a nonzero return code is returned as a result of the buffer I/O.

Environment

- Functional subsystem (HASPFSSM)
- $WAIT is not applicable.

$CALL - Call a Subroutine from JES2

Use $CALL to call a subroutine from a JES2 module. Note that $CALL will attempt to branch to a local routine before attempting to branch to a global one. Therefore, ensure that if you have defined two routines with the same name that your routine branches to the desired one.

Format Description

```
[symbol] $CALL   arg-addr
                 [,PARM=adval]
                 [,ERRET=addrx]
```

arg

Specifies the address of the subroutine to be called. An A-type address is generated if the argument address is not a type "U"; otherwise a V-type address is generated. If register notation is used, the designated register must contain the address of the subroutine to be called prior to executing $CALL.

PARM=

Specifies a parameter value that is to be passed to the called subroutine via register 1. Use of a register, label, or assembler literal address are the only possibilities. If register notation is used, the designated register must contain the address of the parameter value.

ERRET=

Specifies the address of an error routine that is to get control if the called subroutine passes back a nonzero return code (in register 15).

Environment

- Main task, subtask, and user address space.
- $WAIT can occur depending on the routine that is called.

$CKPT - Schedule the Checkpoint of an Element

Use $CKPT to schedule the checkpoint of an element in a JES2 checkpoint table that has been altered.

Format Description

```
[symbol]    $CKPT       ID=kit-id-code
                        [,ADDR={addrx }]
                               {(R1)  }
                        [,POST={YES}]
                               {NO }
```

ID=

Specifies the checkpoint information table (KIT) to be used. This value must be the 1- to 4-character identifier in the KIT for the element to be checkpointed. If more than 4 characters are specified, only the first 4 are used. The valid identifiers include:

DAS
The direct access control blocks.

JQE
The job queue element.

JOE
The job output table.

JIX
The job queue index table.

RSO
The remote sign-on table.

PST
The JOE flag bytes

LCK
The spool offload checkpoint element

TGM
The track group maps

ADDR=

Specifies the address of the element to be checkpointed. If this parameter is omitted, only the header of the checkpoint area that is specified is checkpointed.

POST=
Specifies whether or not a post will occur for the checkpoint.

Note: You must have control of the checkpoint data set when you issue this macro.

Environment

- Main task.
- $WAIT cannot occur.

$COUNT - Count Selected Occurrences

Use $COUNT conditionally to increase a counter every time the macro instruction is executed and to determine the number of times a particular event occurs or a particular section of code is entered. The counter is a halfword (modulo 32,768), which is located 22 bytes into the macro expansion (symbol 120); this counter is increased if DEBUG=YES was specified during JES2 initialization.

Format Description

```
[symbol]    $COUNT      [ R={ (Rn)  } ]
                          { (R1)  }
```

R=

Specifies a register to be used in performing the counting operation. If this parameter is omitted, register 1 is used.

Environment

- Main task and during JES2 initialization and termination.
- $WAIT cannot occur.

$CWTO - Command Processor Write to Operator

Use $CWTO to cause a write to operator to take place. This macro instruction returns control to the code issuing the macro. The command processor PCE must be in control when you issue this macro instruction. Note that, you cannot use $CWTO in Exit 5 if the exit routine determines that JES2 should process the command. If you use this macro in Exit 5, your routine must do all required processing within the exit. When this processing is completed, your routine must notify HASPCOMM.

Format Description

```
[symbol]    $CWTO       MSG={addr  }
                        {(R1)  }
                        {'text'}
                        ,L=value
                        ,MSGID=code
                       [,JOB={YES}]
                        [    {NO }]
                       [,TRUNC={YES}]
                        [      {NO }]
```

MSG=
Specifies either the address of the text for the message or the text itself. If you specify the text, enclose the character string in single quotes ('). If you want the text to include single quotes, code two single quotes together.

L=
Specifies the length, in bytes, of the message text. The length does not include the extra single quote coded to allow the use of a single quote within the text.

MSGID=
Specifies a 3-digit decimal number, from 001-999, to be written out with the message. You must include leading zeros.

JOB=
Specifies whether or not the WTO is job related. Code JOB= as follows:

YES
The job name and number are inserted in the message.

NO (default)
The message text remains unmodified.

TRUNC=

YES
Any multiple-line WTO is truncated. Additional $CWTO or $CRET macro executions specifying message text result in the issuance of an SVC34. The SVC34 treats the message text as a command to JES2.

NO (default)
No truncation takes place.

Environment

- Main task.
- $WAIT can occur.

$DCBDYN - Call the Dynamic DCB Service Routine

Use $DCBDYN macro as the interface to the $DCBDYN service routine to attach or detach a JES2 data control block (DCB) and data extent block (DEB) for a specified device control table (DCT).

Format Description

```
[symbol] $DCBDYN    {ATTACH }
                    {DETACH}
                   ,DCT={label}
                        {(Rn)  }
                        {(R1)  }
```

ATTACH
Requests that a DCB and/or DEB (if one is required) be dynamically created. If the DCT does not require a DCB (for example, one has already been created). JES2 takes no further action.

DETACH
Requests that the specified DCB/DEB be dynamically deleted. If the DCT does not require a DCB or if there is no DCB already attached, JES2 takes no further action.

DCT=
Specifies a label or register containing the address of the DCT to either be attached or detached.

Return Code	Meaning
0	DCB successfully ATTACHed or DETACHed as requested
4	DCB ATTACH failed (GETMAIN unsuccessful)

Environment

- Main task
- 31-bit addressing mode only
- $WAIT can occur

$DCTDYN - Call the Dynamic DCT Service Routine

Use $DCTDYN macro as the interface to the $DCTDYN service routine to attach or find a JES2 device control table (DCT). This macro passes the DCT name and subscript and type of request to the calling routine.

Format Description

```
[symbol] $DCTDYN    {ATTACH}
                    {FIND  }
                   ,NAME={label}
                         {(R1) }
                   [,NUMBER={label}]
                            {(R0) }
```

action
Specifies the action requested.

ATTACH
Requests that the specified DCT be located, or if it doesn't exist, that a new one should be created. ATTACH is only a valid specification if this macro is called by JES2.

FIND
Requests that the specified DCT be located.

NAME=
Specifies the address of an 8-byte field that contains the name of the specified DCT. NAME can be specified as a register (1 to 12) or the name of the field containing the address of the DCT name. The address is loaded into register 1.

NUMBER=
Specifies the subscript (the binary value) of the DCT. NUMBER can be a register (0, 2 to 12) or the name of a field containing the subscript. The value is loaded into register 0.

Return Code	Meaning
0	DCT successfully found for either FIND or ATTACH request
4	DCT successfully ATTACHed if ATTACH requested
8	DCT not found -- ATTACH not specified
12	DCT FIND/ATTACH not successful. The subscript specified by NUMBER= was either: not within the valid range, required and not specified, or not required and specified.
16	DCT ATTACH not successful -- error in $GETMAIN
20	DCT FIND/ATTACH not successful -- DCT table not found
24	DCT FIND/ATTACH not successful -- UCT not found

Environment

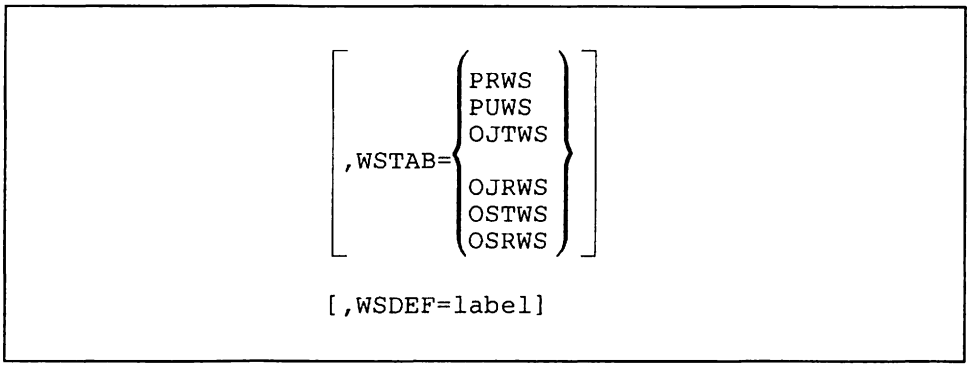
- Main task
- $WAIT cannot occur

$DCTTAB - Map DCT table entries

Use $DCTTAB to map and generate DCT table entries.

Format Description

```
[label]  $DCTTAB  TABLE={HASP  [,NOENTRY]}
                        {USER            }
                        {END             }
                ,NAME=dct-name1
                [,ALIAS=dct-name2]
                ,DESC=desc-text
                [,PCETAB=label]
                [,PCEPTR=[()]field[[{,HCT}][()]]]
                                   {,UCT}
                ,DEVTP=device
                ,SIZE=dct-size
                [,COUNT=[()]field[[{,HCT}][()]]]
                                  {,UCT}
                [,DEVID={0        }]
                        {device-id}
                [,ROUTINE=label]
                [,RANGE=nlow-nhigh]
                [,SUBTYPE={YES}]
                          {NO }
                [,PARENT=dctdev-type]
                [,SUBCHAIN=dct-field]
                [,CHAIN=[()]field[[{,HCT}][()]]]
                                  {,UCT}
                [,DISPLAY={YES}]
                          {NO }
                [,DCB={EXCP}]
                      {BSAM}
                      {NO }
```



TABLE=
Specifies the start (TABLE=HASP) and end (TABLE=END) of a DCT table. If neither this keyword nor NAME= is specified, JES2 generates the DTAB DSECT.

HASP
Specifies that this is a HASP table.

NOENTRY
Specifies that an ENTRY statement need not be generated for the label of this DCT table.

USER
Specifies that this is a USER table.

END
Specifies the end of the DCT table.

Note: If TABLE= is specified, all other keywords on this macro are ignored.

NAME=
Specifies a 1- to 8-character DCT name for this DCT type.

ALIAS=
Specifies a 1- to 8-character DCT name to be used as an alternate

DESC=
Specifies a 1- to 24-character description of this DCT type. Blanks are allowed if the text is enclosed in single quotes. This keyword is used for documentation purposes only.

PCETAB=
Specifies the label on the PCE table entry in the same assembly module that corresponds to this DCT type. This keyword causes this DCT to be defined in a one-to-one PCE-DCT correspondence.

Note: PCETAB= and PCEPTR= are mutually exclusive.

PCEPTR=
Specifies the name of a fullword field that contains the address of the PCE that handles DCTs of this type when not organized in a one-to-one PCE-DCT correspondence as specified by PCETAB=.

Note: PCETAB= and PCEPTR= are mutually exclusive.

field
Specifies an HCT field if this is a HASP table and a UCT field if this is a USER table.

HCT
Indicates an HCT field.

UCT
Indicates a UCT field.

DEVTP=
Specifies a unique value used for the 1-byte DCTDEVTP field that defines this DCT type. This is a required keyword.

WSTAB=
Specifies the type of work selection table that corresponds to this DCT.

PRWS
Indicates that this is a printer work selection table.

PUWS
Indicates that this is a punch work selection table.

OJTWS
Indicates that this is an offload job transmitter (OFFn.JT) work selection table.

OJRWS
Indicates that this is an offload job receiver (OFFn.JR) work selection table.

OSTWS
Indicates that this is an offload SYSOUT transmitter (OFFn.ST) work selection table.

OSRWS
Indicates that this is an offload SYSOUT receiver (OFFn.SR) work selection table.

Notes:

1. *This keyword is required for DCTs that support work selection.*
2. *If this keyword specification is other that those listed above, JES2 assumes that the table type is user defined.*
3. *If WSTAB is specified, you must also specify WSDEF= .*

WSDEF=
Specifies the address of the default work selection list for this device.

SIZE=
Specifies the size of this DCT type. This can be specified either as an equated symbol or computed as SIZE - DCT.

CHAIN=
Specifies the name of a fullword field from which all DCTs of this type are to be chained.

field
Specifies an HCT field if this is a HASP table and a UCT field if this is a USER table.

HCT
Indicates an HCT field.

UCT
Indicates a UCT field.

COUNT=
Specifies the name of a fullword field that contains the number of DCTs defined for this DCT type.

field
Specifies an HCT field if this is a HASP table and a UCT field if this is a USER table.

HCT
Indicates an HCT field.

UCT
Indicates a UCT field.

DEVID=
Specifies the device ID that is placed into the first byte of the DCTDEVID field. If a device does not have a device ID, this field is set to 0.

ROUTINE=
Specifies the name of the routine used to initialize the DCTs.

RANGE=
Specifies the lower (nlow) and upper (nhigh) range limits of the subscript values that are allowed for this DCT type. If this keyword is not specified, the DCTs will not contain subscripts.

SUBTYPE=
Specifies whether this DCT has other DCTs chained off it within a subchain. The default is NO.

PARENT=
Specifies the DCTDEVTP of the DCT off which this DCT is chained.

Note: If this keyword is specified, SUBCHAIN= must also be specified.

SUBCHAIN=
Specifies the name of the field in this DCT which chains the DCT off the parent DCT and any other DCT types within the subchain.

DISPLAY=
Specifies whether (YES) or not (NO) this DCT will be displayed by a $D U operator command.

YES
Indicates that this DCT is chained within the DCTPOOL chain and therefore displayed by the $D U operator command. This is the default.

NO
Indicates that this DCT is chained within the DCTPOL2 chain and therefore not displayed by the $D U operator command.

DCB=
Specifies whether either a DCB or DEB is built for this DCT.

EXCP
Indicates that both an EXCP DCB and DEB be built.

BSAM
Indicates that a BSAM DCB be built.

NO
Indicates that neither a DCB nor a DEB be built for this DCT. This is the default.

Environment

- JES2 main task or during initialization and termination
- $WAIT is not applicable -- this macro generates a DSECT or a static table entry; it does not generate executable code.

$DEST - Convert Symbolic Destination to Route Code

Use $DEST to convert a symbolic destination to a binary route code. The route code is returned in register 1.

Format Description

```
[symbol] $DEST      {dest-addrx}
                    {(R1)      }
                   ,{default-addrx}
                   ,{(R0)         }
                    [,ERR=relexp]
                    ,LEN=value
```

dest-addrx
Specifies the address of the symbolic destination for which a binary route code is obtained. If register notation is used, the destination address must be loaded into the designated register prior to the execution of this macro instruction.

default-addrx
Specifies the address of the word in storage containing, in its rightmost two bytes, the default node number used in constructing the binary route code. If register notation is used, the default is loaded into the designated register prior to the execution of this macro instruction.

ERR=
Specifies a location to which control is returned if the specified destination is invalid.

If this operand is omitted, the condition code is set to reflect the validity of the specified destination as follows:

CC=0
The destination is invalid.

CC≠0
Register 1 contains the associated route code.

LEN=
Specifies the length of the symbolic destination. If the length of the symbolic destination is less than eight bytes, the characters must be padded to eight bytes with blanks. This operand *must* be specified.

Environment

- Main task.
- $WAIT cannot occur.

$DISTERR - Indicate Disastrous Error

Use $DISTERR to indicate that a disastrous error has occurred. The macro instruction causes the message $HASP096 DISASTROUS ERROR AT SYMBOL symbol IN CSECT module to be printed out on the $ERR and $LOG consoles.

Format Description

```
[symbol] $DISTERR  [BUFFER=addrx]
                        [,JOB={addrx}]
                              {(R0) }
```

symbol
Consists of a symbol to be used to generate the error message and so that it can be referenced in the assembler cross reference for the indicated module. **This symbol must be specified.**

BUFFER=
Specifies the address of a buffer that contains information concerning the disastrous error. This buffer information is traced. If BUFFER= is omitted no disastrous error information is traced.

JOB=
Specifies the address of either the JCT or the JQE of the job being processed at the time the error occurred. If JOB= is specified, the job ID and job name are added to the start of the $HASP096 message.

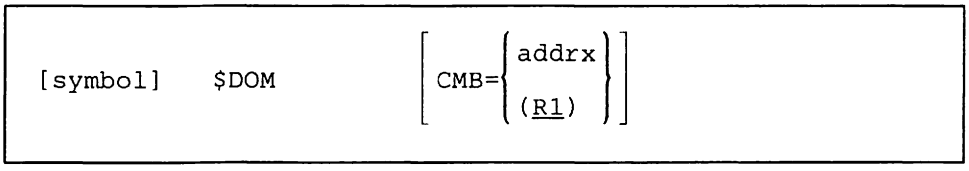
Environment

- Main task.
- $WAIT can occur.

$DOM - Delete Operator Message

Use $DOM to delete an operator message.

Format Description



CMB=
Specifies the address of the command message buffer (CMB) containing the operator message to be deleted. If register notation is used the register must contain the address of the CMB prior to executing the $DOM. If CMB= is not specified register 1 is assumed to contain the address of the CMB.

Environment

- Main task.
- $WAIT cannot occur.

$DORMANT - Specify Processor is Inactive

Use $DORMANT to indicate to the JES2 dispatcher that the associated JES2 processor has completed the processing of a job or task.

Format Description

```
[symbol]    $DORMANT      [R=R1]
```

R

Specifies the register which is to be used by the $DORMANT macro instruction. If R is omitted, register 1 is used.

Note: Do not enclose the specified register in parenthesis.

Caution: The $DORMANT macro instruction should never be used unless a corresponding $ACTIVE macro instruction has been used for the same processor.

Environment

- Main task.
- $WAIT cannot occur.

$DTEDYN - Call the Dynamic DTE Service Routines

Use the $DTEDYN macro instruction to call the dynamic DTE service routines ($DTEDYNA and $DTEDYND) located in HASPDYN that handles DTE management and subtask attaches and detaches for the JES2 main task.

Format Description

```
[symbol]    $DTEDYN    {ATTACH}
                       {DETACH}
                      [,WAIT={ECB }]
                             {XECB}
                             {NO  }
                       ,PARM=parameter
                       ,ID=subtask-id
                      [,DTE={addrx}]
                            {(R0) }
                      [,ERRET={addrx}]
                              {(Rn) }
```

ATTACH | DETACH
Specifies whether to call the $DTEDYNA (ATTACH) or $DTEDYND (DETACH) service routine.

ATTACH
Informs $DTEDYNA to obtain and initialize a new DTE and to attach the subtask for the caller. Return codes from $DTEDYNA are as follows:

RC=	Meaning
0	Processing successful
4	Processing failed. $GETWORK failed to obtain the new DTE storage.
8	Processing failed. The MVS ATTACH macro processing returned a nonzero return code (returned to the caller of $DTEDYN in register 1).

DETACH
Informs $DTEDYND to free up the DTE and to detach the subtask. Return codes from $DTEDYND are as follows:

RC=	Meaning
0	Processing successful

WAIT=
Specifies whether $DTEDYN should wait for subtask initialization and/or termination.

ATTACH Processing

ECB
Indicates that $DTEDYNA should wait for the subtask to post the initialization ECB.

XECB
Indicates that $DTEDYNA should $WAIT (XECB style) for the subtask to post the initialization ECB.

NO
Indicates that $DTEDYNA is not to wait.

DETACH Processing

ECB
Indicates that $DTEDYND should wait for MVS to post the subtask termination ECB.

XECB
Indicates that $DTEDYND should $WAIT (XECB style) for MVS to post the subtask termination ECB.

NO
Indicates that $DTEDYND is not to wait.

PARM=
Specifies a fullword parameter to be passed to the subtask during ATTACH processing. Do not use this keyword during DETACH processing.

ID=
Specifies the subtask identifier as defined in the $DTE control block. The subtask ID is passed to the $DTEDYN service in register 1. If this keyword is not specified, an assembly error will occur.

DTE=
Specifies the address of the DTE to be freed by $DTEDYND.

Note: This keyword must be specified for $DTEDYN DETACH; it is not valid for $DTEDYN ATTACH.

ERRET=
Specifies the address of an error routine that is to get control if an error occurs during dynamic DTE processing.

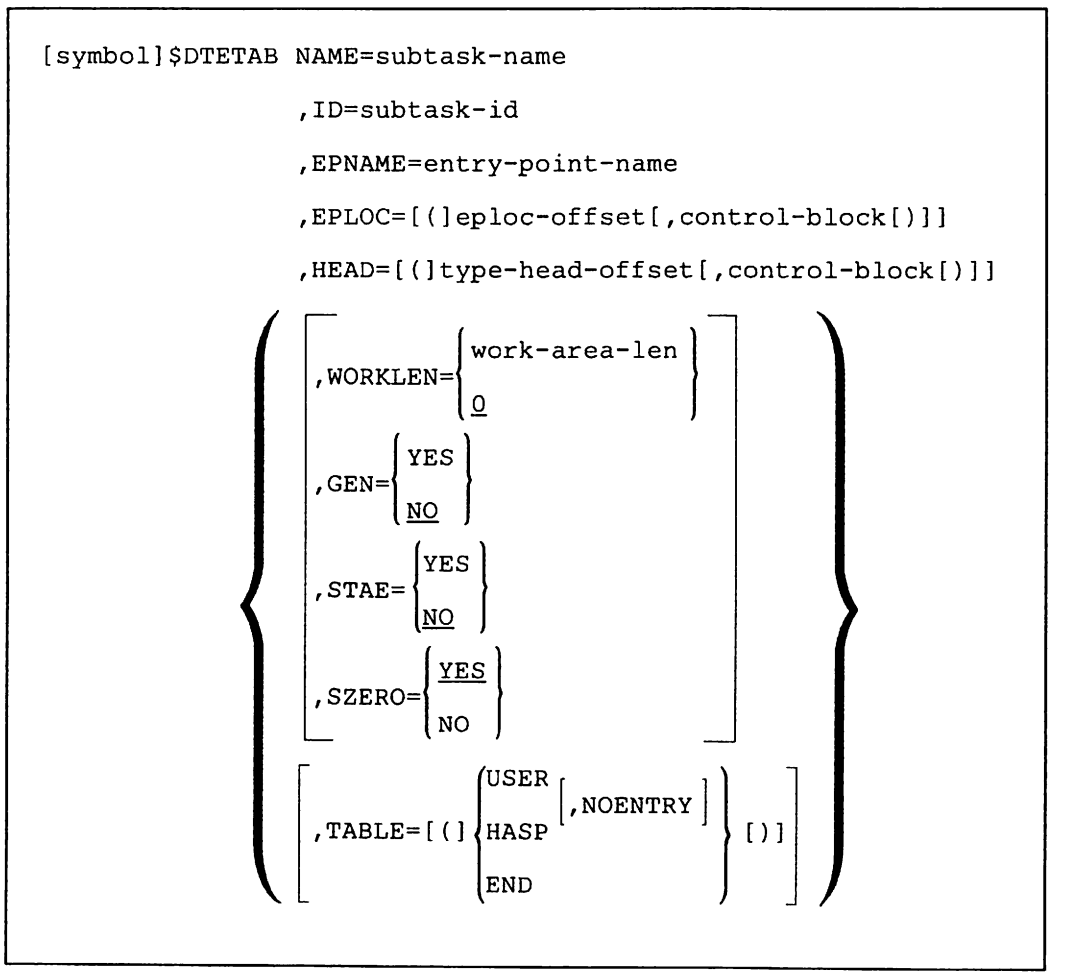
Environment

- Main task
- $WAIT can occur in JES2 main task
- MVS WAIT can occur during initialization and termination

$DTETAB - Build and Map the DTE Tables

Use the $DTETAB macro instruction to build the DTE tables accessed by the $GETABLE service.

Format Description



NAME=
Specifies the subtask name that HASP messages use to identify the subtask to the operator.

ID=
Specifies the subtask identifier used in the $DTE DSECT.

EPNAME=
Specifies the entry point name used by $DTEDYNA for the MVS IDENTIFY macro call.

EPLOC=
Specifies the offset into the specified control block and, optionally, the control block name from which the entry point address is obtained. The control block name defaults to either MODMAP (if this macro is used to

build a HASP DTE table) or UCT (if this macro is used to build a USER DTE table). If the 'control block name' is specified here it overrides either default value.

HEAD=
Specifies the offset and control block name of the subtask type chain head. The control block name defaults to either HCT (if this macro is used to build a HASP DTE table) or UCT (if this macro is used to build a USER DTE table).

WORKLEN=
Specifies the length of the subtask work area extension. If specified, this length is added to the DTE length (DTELEN) when $DTEDYNA obtains DTE storage. WORKLEN=0 (no storage) is the default.

GEN=
Specifies whether (YES) or not (NO) the subtask is ATTACHed during IRMVS processing. YES indicates that the DTE1FLAG flag, DTE1AUTO is set on to indicate subtask ATTACH. GEN=NO is the default.

STAE=
Specifies whether (YES) or not (NO) the subtask is DETACHed if STAE is specified on the DETACH macro. YES indicates that the DTE1FLAG flag, DTE1STAE is set on to indicate subtask DETACH. STAE=NO is the default.

SZERO=
Specifies whether (YES) or not (NO) the DTEFLAG1 flag, DTESUB0, is set on at the MVS ATTACH call. YES indicates that the DTEFLAG1 flag, DTE1SUB0 is set on. SZERO=YES is the default.

TABLE=
Specifies either the beginning of a USER or HASP DTE table (TABLE=USER or TABLE=HASP, respectively) or the end of a previously specified table (TABLE=END). The NOENTRY operand indicates that the ENTRY statement will not be generated for this specific DTE table. If TABLE= is specified, all other keywords on this macro are ignored.

Environment

- JES2 main task or during JES2 initialization or termination
- $WAIT is not applicable -- this macro generates a DSECT or a static table entry; it does not generate executable code.

$ENTRY - Provide Entry to JES2 Processor or Function

Use $ENTRY to identify and document an entry point to a JES2 processor function or installation exit routine. The documentation consists of the entry point name, padded with blanks to 8 characters and starting on a doubleword boundary.

Note: $ENTRY **must** be used to identify the entry point to an installation exit routine and either ENTRY=YES (the default) or CSECT=YES must be specified.

Format Description

```
[symbol]    $ENTRY    [BASE={Rn | (R1,R2,...) | (R8)}]
                      [,CSECT={YES | NO}]
                      [,ENTRY={YES | NO}]
                      [,REGUSE={(Rn) | (R15)}]
```

symbol
Specifies the entry point name. It must be provided.

BASE=
Specifies the registers that provide addressability to the entry point. If two or more registers are specified, they must be separated by commas and enclosed in parentheses. If BASE is not specified, BASE=R8 is assumed. A USING statement is generated using the specified registers.

Note: Although the $ENTRY macro is used, you must still load the base register with the address of the entry point.

ENTRY=
Specifies whether or not an ENTRY statement is to be generated. The default is YES; however, no ENTRY statement is generated if CSECT=YES was specified.

CSECT=
Specifies whether or not a CSECT statement should be generated. The default is NO. If a CSECT statement is generated, the JES2 version number is included in the documentation.

REGUSE=
Specifies the register containing the address of the entry point. This register is used as a base register in a "branch" instruction that causes processing to bypass the inline documentation. The default is register 15.

Notes:

1. *It is recommended that you have only one CSECT for each source module.*

2. *If your specify ENTRY=YES or CSECT=YES, then a new MIT entry table (MITETBL) is generated.*

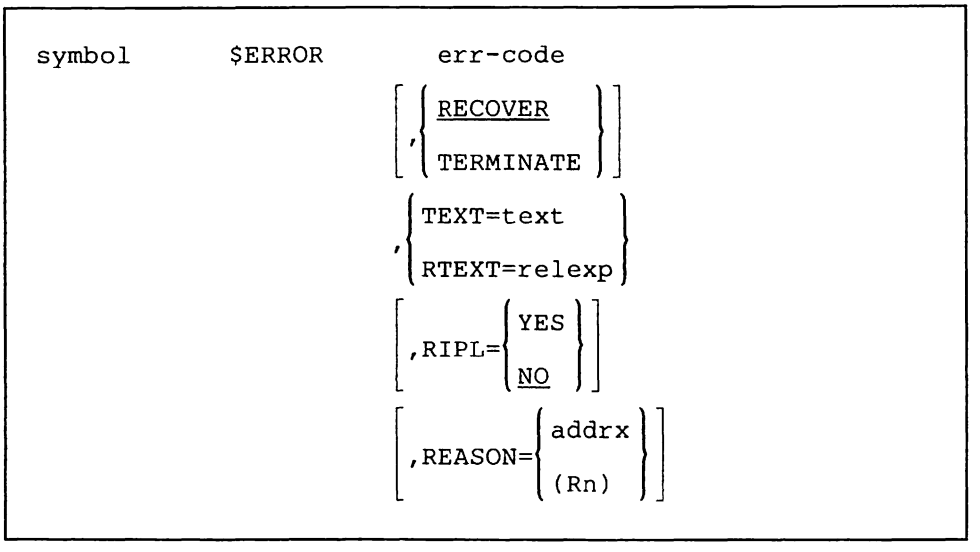
Environment

- Main task, subtask, or user address space.
- $WAIT not applicable

$ERROR - Indicate Catastrophic Error

Use $ERROR to indicate that a catastrophic error has occurred and to abend the JES2 main task. The macro instruction causes a $HASP095 message to be printed out on the console by the JES2 ESTAE routine, which receives control on the abend.

Format Description



symbol
Consists of a 1- to 4-character symbol indicating the type of error that occurred. **This symbol is required.**

err-code
Specifies the JES2 catastrophic error code. It is a 1- to 4-character symbol (usually a letter and two digits) preceded by a dollar sign ($) when printed as the error code in the message.

option-code
Specifies whether or not recovery should take place. This can be coded as follows:

RECOVER (default)
Recovery is to be attempted; that is, it is possible to recover from this error.

TERMINATE
Recovery is not to take place and an abend is to occur; that is, it is impossible to recover from this error.

TEXT=
Specifies the explicit text to be included in the catastrophic error message. A DC statement is generated for this character string. This text is used in the content of the $HASP095 message.

RTEXT=
Specifies the symbol used on another $ERROR macro instruction invocation from which the text for this catastrophic error message is to be taken. This is used when there is an existing $ERROR macro instruction invocation with the desired text.

RIPL=
Specifies whether or not the system needs an IPL to recover from this error. This can be coded as follows:

YES
The system needs an IPL.

NO (default)
The system does not need an IPL.

REASON=
Specifies the reason code that appears in the $HASP095 error message.

addrx
Indicates the address of a fullword field that contains the reason code.

Rn
Indicates a register that contains the reason code.

Environment

- Main task.
- Will ABEND with MVS system code X'02D'

$ESTAE - JES2 Error Recovery Environment

Use $ESTAE to generate the calling sequence to one of the three JES2 recovery service routines in HASPNUC for the purpose of creating, replacing or cancelling the current processor recovery element (PRE). Each PRE represents an error recovery routine that will gain control in the case of a JES2 detected error.

Use this macro to:

- Create a new JES2 error recovery environment by establishing another recovery routine
- Replace the current error recovery routine with a different routine
- Cancel the accessibility of the current error recovery routine

If you issue a $ESTAE macro instruction with a recovery address (RECADDR=) specified within code that is logically bracketed by $SAVE and $RETURN macros, the PRE created is cancelled automatically in $RETURN processing.

Format Description

```
[symbol]   $ESTAE     [action-code]
                      [ ,RECADDR={addrx }]
                                 {(R1)  }

                      [,NAME=symbol]
```

action-code
Specifies if the current PRE is to be cancelled or replaced as follows:

CANCEL
Cancel the current PRE.

REPLACE
Replace the current PRE with a new one.

If you omit this operand, a new PRE is created and stacked LIFO on the PRE stack.

RECADDR=
Specifies the address of the recovery routine to gain control if JES2 detects an error. If you specify this address as 0, recovery is suspended.

NAME=

Specifies the 8-character identifier to be associated with the PRE created when you have specified either REPLACE or nothing as the first positional operand. If you omit this parameter, the identifier will default to the label of the $ESTAE macro call. If you have no label specified, it will default to a system-generated identifier.

Notes:

1. *Should you code either CANCEL or REPLACE on the $ESTAE macro, there must be a current PRE at the current save area level or JES2 catastrophic error $ER1 is issued and JES2 terminates.*

2. *$ESTAE assumes addressability to the error recovery area (ERA) that is associated with the error that caused the recovery routine to be entered. Therefore, be certain to add the $ERA DSECT to the $MODULE macro for any routine for which you provide error recovery.*

Environment

- Main task.
- $WAIT cannot occur.

$EXCP - Execute JES2 Channel Program

Use $EXCP to initiate JES2 input/output activity.

Format Description

```
[symbol]    $EXCP      {dct-addrx}
                       {(R1)     }
                       [,TYPE=VR]
                       [,WAIT={YES}]
                              {NO }
```

dct

Specifies either the address of a pointer to a device control table (DCT) or the address of a DCT. The DCT represents a device upon which input/output activity is to be initiated. If dct is written as an address, it represents the address of a fullword which contains the address of the DCT. If dct is written using register notation (either regular or special register notation), it represents the address of the DCT. If register notation is used, the address must have been loaded into the designated register before the execution of this macro instruction.

TYPE=VR

Specifies that I/O is to be initiated through the EXCPVR macro instruction. If this parameter is omitted, EXCP is used.

WAIT=

Specifies whether the $EXCP service routine is to cause the routine issuing this macro instruction to wait ($WAIT IO) until the I/O operation has been completed (WAIT=YES), or is to return control as soon as the request has been scheduled (WAIT=NO or parameter omitted.)

If WAIT=YES is specified, the service routine exits after normal I/O completion. If any I/O error is detected, the service routine issues the $IOERROR macro instruction, which issues the JES2 I/O error message, $HASP094, then returns control to the $EXCP issuer. On return, register 1 points to the I/O buffer, and the condition code mask is set as follows:

CC=1

The I/O completed without error.

CC=4

Permanent I/O error was encountered.

Environment

- Main task.
- $WAIT can occur (if you code WAIT=YES on the macro).

$EXIT - Provide Exit Point

Use $EXIT to establish an exit point.

Format Description

```
[symbol]   $EXIT      exitid-code
                      [,ENVIRON={JES2   }]
                                {SUBTASK}
                                {USER   }
                                {FSS    }
                      [,JOBMASK=relexp]
                      [,SAVAREA=(register)]
                      [,TYPE={TEST }]
                             {ENTER}
                      [,AUTOTR={YES}]
                               {NO }
                      [,MAXRC={code}]
                              {4   }
```

symbol
Although a label for this macro instruction is not required, it is highly recommended for tracing purposes.

exitid-code
Specifies the numeric id (0-255) of this $EXIT macro.

ENVIRON=
Specifies the environment in which the exit is to be taken. This determines which of the two exit effectors are used to establish correct linkage to the user-supplied exit routine(s). If ENVIRON= is not specified the environment set in the global macro variable &ANVIRON is used. Possible environments include:

JES2
Specifies that the exit point is in the JES2 main task.

SUBTASK
Specifies that the exit point is in a subtask of the JES2 main task. Refer to the note for the SAVAREA= keyword below for information concerning save area types in the subtask environment.

USER
Specifies that the exit point is in a user address space executing code in the subsystem support module (HASPSSSM). Refer to the note for the SAVAREA= keyword below for information concerning save area types in the user environment.

FSS
Specifies that the exit point is in a functional subsystem address space executing code in the FSS support module (HASPFSSM).

JOBMASK=
Specifies the address of a 256-bit job exit mask. This mask shows the status of each of the 255 possible exit ids ('0'=inactive, '1'=active). Use this operand only if the exit point is job-related, because the mask is used to determine whether or not the exit should be taken for a given job.

SAVAREA=
Specifies the address of an area to be passed to the exit effector for use as a save area. Use this operand except when you specify ENVIRON=JES2.

Note: The save area specified by the SAVAREA= keyword is distinct from, and should not be confused with, the save area pointed to by register 13. Register 13, at the $EXIT point in the code, contains the address of the save area into which the exit effector stores the current registers. The save area specified by the SAVAREA= keyword is the save area into which the exit routine will store the exit effector's registers. or ENVIRON=USER. If you omit this operand, the exit effector obtains a save area via an MVS GETMAIN and passes its address to the exit routine(s). Register notation is required for this parameter.

TYPE=
Specifies how the exit effector is to treat this exit point.

TEST
The exit effector tests the status of the exit point, and the exit effector sets a condition code as follows:

CC=0
Either the specific job mask bit for this exit is 0 or the exit id is not enabled (that is, no exit routines are to be called).

CC=1
The exit id is enabled but tracing is disabled.

CC=2
Both the exit id and tracing are active.

ENTER
The exit routine is to be entered through the exit effector without checking the status of the exit point. A $EXIT macro instruction should be coded with TYPE=TEST to confirm exit point status before coding a $EXIT macro with TYPE=ENTER.

If this parameter is omitted, the status of the exit point is to be determined and, if the exit is enabled, the exit effector is called to invoke the appropriate user exit routine(s).

AUTOTR=
Specifies whether or not tracing for this exit point is to be automatically provided by the exit effector. Possible values are as follows:

YES (default)
This only allows tracing to take place. Tracing occurs only if trace id 13 is turned on (via a $TRACE command), and either the EXITnnn initialization statement specified AUTOTR=YES or the operator has entered a $T EXITnnn, TRACE=Y command for this exit point.

NO
No tracing takes place.

MAXRC=
Specifies the maximum acceptable return code to be set by the user exit routine(s). If this parameter is omitted, the default is 4.

Environment

- Main task, subtask, user, or FSS address space.
- $WAIT can occur if installation exit routine issues a $WAIT or invokes a routine that issues a $WAIT.

$EXTP - Initiate Remote Terminal Input/Output Operation

Use $EXTP to initiate a remote terminal or network device input/output action or operation.

Format Description

```
[symbol]    $EXTP      type-code
                       ,DCT={dct-addrx}
                             {(R1)    }
                      [,PARM={loc-addrx}]
                             {(R0)    }
```

type

Specifies the type of operation as follows:

OPEN
Initiate processing

GET
Receive one record

PUT
Send one record

CLOSE
Terminate processing

NCLOSE
Abnormally terminate processing

READ
Receive one NJE record

WRITE
Send one NJE record

DCT=

Specifies either a pointer to a DCT or the address of a DCT that represents the remote terminal device; in the case of a read or write, it represents a line DCT. If dct is written as an address, it represents the address of a fullword, which in its three rightmost bytes contains the address of the remote terminal device DCT. This word must be located on a word boundary in storage. If dct is written using register notation (either regular or special register notation), it represents the address of the remote terminal device DCT.

PARM=

If type specifies either OPEN, CLOSE, NCLOSE, READ, or WRITE, this parameter should not be specified. If type specifies GET, this parameter specifies the address of an area into which the input record will be placed. The input area must be defined large enough to contain the largest record to be received.

If the type-code is specified as PUT, this keyword specifies the address of a parameter area containing a CCW command code which contains the carriage control (or stacker select), the data length, and the starting address of the data in the following format:

AL1	CCW command word
AL3	Data length
AL4	Starting address

If register notation is used, the appropriate address must be loaded into the designated register before the execution of this macro instruction.

Upon return, the condition code is set as follows:

When type is OPEN,PUT,CLOSE,NCLOSE,READ,WRITE

CC>0
Successful completion

CC=0
Unsuccessful completion

When type is GET

CC>0
Successful GET processing

CC=0
Unsuccessful GET processing

CC>0
End-of-file received

Environment

- Main task.
- $WAIT can occur.

$FRECEL - Free Common Storage Area (CSA) Cell

Use $FRECEL to return a storage area to the cell pool.

Format Description

```
[symbol]    $FRECEL    [CELL={addrx}]
                      [     {(R1) }]
```

CELL=
Specifies the address of the storage cell to be freed. If register notation is used, the designated register must be loaded with the address of the storage cell. If this operand is omitted, CELL=(R1) is assumed.

Notes:

1. *The storage cell to be freed must contain as its first word the address of the CCE associated with the storage area. The proper form for this address is obtained by the execution of a $GETCEL macro instruction and should remain unaltered and restored prior to executing the $FRECEL macro instructions.*

2. *Although the freeing of storage cells that are obtained by $GETCEL or through supporting routines in the HASPSSSM module via the $FRECEL macro instruction is only available to routines within the JES2 main processors, other tasks may use the HASPSSSM subroutines directly without using this macro instruction.*

Environment

- Main task.
- $WAIT cannot occur.

$FRECMB - Free a Console Message Buffer

Use $FRECMB to return a console message buffer to the free queue.

Format Description

```
[symbol]    $FRECMB      CMB={addrx}
                         {(R1) }
                        [,COUNT={YES}]
                         [      {NO }]
```

CMB=
Specifies the address of the console message buffer to be placed on the free queue. If register notation is used, the address of the console message buffer must be loaded into the designated register prior to executing this macro instruction.

COUNT=
Specifies whether or not the console lockout security count contained in SVTCOMCT is to be increased by 1 as follows:

YES
The count is to be increased.

NO (default)
The count is not to be increased.

If the console lockout security count was decreased when a console message buffer was removed from the free queue, the count must be increased when the buffer is returned.

Environment

- Main task.
- $WAIT cannot occur.

$FREEBUF - Return a JES2 Buffer to the JES2 Buffer Pool

Use $FREEBUF from the JES2 main task or user environment to return a JES2 buffer to the JES2 buffer pool.

Format Description

```
[symbol] $FREEBUF    { buffer-addrx }
                     { (R1)         }
                     [,MULTIPLE]
                     [,TYPE={ PROT  }]
                            { UNPROT}
                            { HASP  }
                     [,A={ addrx }]
                         { (R1)  }
```

buffer

Specifies either a pointer to a buffer or the address of a buffer to be returned to the buffer pool as follows:

- If buffer is written as an address, then it represents the address of a full word which contains the address of the buffer to be returned in its three rightmost bytes.
- If buffer is written using register notation (either regular or special register notation), then it represents a register containing the address of the buffer to be returned.
- If register notation is used, the address must have been loaded into the designated register before the execution of this macro instruction.

MULTIPLE

Indicates that the specified buffer is the first of a chain of buffers, linked through their BUFCHAIN fields. If MULTIPLE is specified, the entire chain is returned to the buffer pool. If the parameter is omitted, only the specified buffer is returned to the pool.

TYPE=

Specifies the type of buffer that is to be returned to the JES2 buffer pool.

PROT	Indicates a protected buffer
UNPROT	indicates an unprotected buffer
HASP	Indicates a HASP buffer

A=

Specifies the starting address of the buffer to be returned. This address is loaded into register 1.

Caution: The specified buffer(s) must have been obtained by a $GETBUF macro instruction. Otherwise, the action of the macro instruction is unpredictable.

TYPE= and A= apply only to the user environment (HASPSSSM) and should be omitted in the JES2 main task environment.

Environment

- Main task or user address space (MULTIPLE is ignored in the user environment.).
- $WAIT cannot occur.

$FRELOK - Free the MVS CMS Lock, LOCAL, or JES2 Job Lock

Use $FRELOK to free the CMS lock, LOCAL, or JES2 job lock obtained via the $GETLOK macro instruction.

Format Description

```
[symbol]    $FRELOK    [TYPE={CMS                    }]
                          {LOCAL                   }
                          {[(]JOB,JQE=relexp[)]    }
```

TYPE=
Specifies the lock to be freed as follows:

CMS (default)
The cross-memory services lock is freed. All other operands are ignored.

LOCAL (valid in the HASPFSSM environment only)
The local lock is freed. All other operands are ignored.

JOB (valid in the JES2 main task only)
The JES2 job lock is freed; you *must* specify a job queue element address (JQE=).

JQE=
Specifies the address of a fullword containing the address of the JQE.

Environment

- Main task and functional subsystem (HASPFSSM).
- $WAIT can occur.

$FREMAIN - Branch-Entry FREEMAIN Services

Use $FREMAIN in both the JES2 main task and the user environment (HASPSSSM) to free an area of storage obtained via MVS GETMAIN services. $FREMAIN invokes the $GETMAIN macro with all the parameters originally specified and adds the FREMAIN=YES parameter.

Format Description

```
[symbol]    $FREMAIN    [type-code]
                        ,LV={value}
                            {(R0) }
                       {,SP=absexp }
                       {,A={addrx} }
                            {(R1) }
                       {,KEY={key-value}}
                             {1        }
                             {PSW      }
                       {,TCB={YES    }}
                             {NO     }
                             {JOBSTEP}
```

type-code

Identifies the type of GETMAIN/FREEMAIN request. You can only specify an unconditional FREEMAIN request. **Caution: If the area of storage referred to has not been obtained via MVS GETMAIN services prior to this point, your subsystem will abend.**

The way to specify requests is as follows:

R or RU or U - An unconditional FREEMAIN request.

LV=

Specifies the length of the area to be freed. This value is loaded into register 0.

SP=

Identifies the subpool number. Subpool zero is the default if no subpool is specified. You must specify this parameter if you want to free an entire subpool. (In that case, do *not* code the A= parameter.)

A=

Specifies the starting address of the storage to be freed. This address is loaded into register 1. This parameter is required unless you need to free an entire subpool, in which case you specify the SP= parameter and not the A= parameter. This keyword applies to the JES2 main task environment.

For the user (HASPSSSM) environment, specifies the address of the TCB associated with the storage to be freed; this parameter is required for this environment.

KEY=
Specifies the key of the storage you want to free. The default is "1." This keyword applies to the user environment (HASPSSSM). Specifying "PSW" is valid for only the user environment.

TCB=
Specifies that the address specified by A= is the TCB address (TCB=YES). Specifies that the address specified by A= is the starting address of the storage to be freed (TCB=NO). The default is TCB=NO. Specifying "JOBSTEP" for the job step TCB is valid for only the user environment.

Environment

- Main task or user address space.
- $WAIT cannot occur.

$FREQC - Free Quick Cell

Use $FREQC to free the storage obtained for the quick cells.

Format Description

```
[symbol]   $FREQC   TYPE={type-code}
                    {(R0)       }
                   [,NUM={nnn  }]
                   [     {(R1) }]
                   [,STACK={YES}]
                   [       {NO }]
                   [,CHAIN={addrx}]
                   [       {(R2) }]
```

TYPE=
Specifies the type of quick cells that are to be freed.

type-code
Specifies the type code as defined in the $QCTGEN macro for the quick cell to be freed.

(R0)
Specifies that register 0 contains the quick cell type-code as defined in the $QCTGEN macro. The two low-order bytes of the register must contain the type-code and the two high-order bytes must be zeroed.

Note: This keyword must be specified or an error will occur at assembly time.

NUM=
Specifies the number of quick cells to be returned to the quick cell pool.

nnn
Specifies the number (1-255) of quick cells. The value assigned to NUM= must not exceed the specification of QCTLIMIT; exceeding this value causes an execution error (F01 FSI catastrophic error), and the $HASP750 message is issued. The default is 1.

(Rn)
Specifies the register in which the number of quick cells is held.

STACK=

Specifies whether the quick cells should be dechained or chained together. The chaining field offset is specified in the QCT.

YES

Specifies that the quick cells specified by this macro are taken off a stack identified by the QCT for the specified type-code.

NO

Specifies that the quick cells specified by this macro are chained together and can be freed one at a time while while progressing through the chain.

CHAIN=

Specifies the register that contains the address of the first cell of the chain of quick cells that are to be freed. Register 0 and 1 cannot be used. This keyword must be coded if STACK=NO is coded or allowed to default.

Environment

- Functional subsystem (HASPFSSM)
- $WAIT not applicable.

$FREUCBS - Free UCB Parameter List Storage

Use $FREUCBS to free UCB parameter list storage

Format Description

```
[symbol] $FREUCBS   UPLADDR={parmlist-addrx}
                            {(R1)          }
```

UPLADDR=
Specifies the address of the UCB parameter list that is to be freed. If register notation is used, the register must be initialized with the UCB parameter list address prior to the execution of the macro.

Environment

- Main task.
- $WAIT cannot occur.

$FREUNIT - Release a Unit Device Control Table (DCT)

Use $FREUNIT to release a device control table (DCT).

Format Description

```
[symbol] $FREUNIT    { dct-addrx }
                     { (R1)      }
```

dct

Specifies either a pointer to a DCT or the address of a DCT to be released. If dct is written as an address, then it represents the address of a full word, which in its three rightmost bytes contains the address of the DCT to be released. If dct is written using register notation (either regular or special register notation), then it represents the address of the DCT to be released. If register notation is used, the address must be loaded into the designated register before the execution of this macro instruction.

Notes:

1. *The execution of this macro instruction may cause a $WTO macro instruction to be executed.*

2. *When a device that was allocated by MVS allocation facilities goes into the DRAINED status, the device is deallocated. To use the device, it must be first started by the operator using the $S device command and the device must be obtained via the $GETUNIT macro instruction. Otherwise, the system replies device unavailable.*

Caution: The specified DCT must have been obtained by a $GETUNIT macro instruction.The action of the macro instruction is unpredictable in other cases.

Environment

- Main task.
- $WAIT can occur.

$FSILINK - Link the Functional Subsystem Interface.

Use $FSILINK to provide base register setup for the major control blocks required by the JES2 functional subsystem interface (FSI) service routines. Specify this macro at the entry point of an FSI service routine. $FSILINK sets up registers to point to the functional subsystem control block and functional subsystem application control block.

Format Description

```
symbol      $FSILINK    REQUEST= function-name
                        [,ERRET={addrx }]
                                {(Rn) }
                        [,TRACEID={14}]
                                  {15}
```

symbol
A symbol **must** be specified on this macro instruction.

REQUEST=
Specifies the function id of this FSI service. The function id specified is compared to that passed in the FSI parameter list at the time of the FSI call. If they do not match, a return code of 4 is placed in register 15 and, if specified by the ERRET= keyword, a branch is taken to an error routine.

ERRET=
Specifies the label or a register containing the address in which to branch if an invalid function id was specified on the REQUEST= keyword.

TRACEID=
Specifies whether trace id 14 (trace GETDS, RELDS, SEND) or trace id 15 (trace GETREC, FREEREC, CHKPT) is to be used for tracing. This macro supports these two trace ids only.

Environment

- Functional subsystem (HASPFSSM)
- MVS WAIT can occur.

$GETABLE - Get HASP/USER Table Entries

Use the $GETABLE macro to return table entries of the HASP/USER table pairs.

Format Description

```
[symbol]    $GETABLE    TABLE={PCE }
                           {DCT }
                           {TID }
                           {DTE }
                          {[[,ID={(R0)   ]]}
                          {  {CURRENT}  }}
                          {  {adval  }  }}
                          {[,LOOP=symbol]}
                          [,ERRET={relexp}]
                                  {(Rn)  }
```

TABLE=
Specifies the type of table pairs to be used.

PCE - indicates the PCE table ($PCETAB).
TID - indicates the trace id table ($TIDTAB).
DCT - indicates the DCT table ($DCTTAB).
DTE - indicates the DTE table ($DTETAB).

ID=
Specifies the table entry id associated with the table entries to be returned to the table pairs.

CURRENT indicates that the id for the current environment is to be used - that is, PCEID (for TABLE=PCE) or TTEID (for TABLE=TID), DCTDEVID (for TABLE=DCT), or DTESTID (for TABLE=DTE). If register notation is used, the designated register must contain the table entry id prior to executing this macro.

LOOP=
Specifies a label that serves as the terminating point of the loop that is the table entries.

If LOOP= is omitted, a single table entry lockup is performed.

ERRET=
Specifies the address of the error routine that is to receive control if the table entry is invalid or the end of the table(s) is reached.

Notes:

1. *ID= and/or LOOP= must be specified.*
2. *You must preserve register 0 before executing $GETABLE in a loop.*

Environment

- Main task and during JES2 initialization and termination.
- $WAIT cannot occur.

$GETASCB - Retrieve the Primary, Secondary, or Home ASCB

Use the $GETASCB macro instruction for a specific asid to place the primary, secondary, or home ASCB address into a register.

Format Description

```
[symbol]    $GETASCB   [ASCBREG={(Rn)  }]
                       [          (R1)]
                       [,WORKREG={(Rn)  }]
                       [         (R15)]
                       [,TYPE={PRIMARY  }]
                       [      SECONDARY]
                       [      HOME     ]
                       [,ASID={addrx}]
                       [      (Rn)  ]
```

ASCBREG=
Specifies the register where the ASCB address is to be stored. Register 1 is the default.

WORKREG=
Specifies a work register that can be used in processing your request. Register 15 is the default.

TYPE=
Specifies the type of ASCB that is to be located.

PRIMARY (the default)
indicates the primary address space's ASCB.

SECONDARY
indicates the secondary address space's ASCB.

HOME
indicates the home address space's ASCB.

ASID=
Specifies the address of the storage location containing the ASID for the address space whose ASCB address is to be returned or specifies the register containing the ASID.

Note: The TYPE= operand is ignored if ASID= is specified.

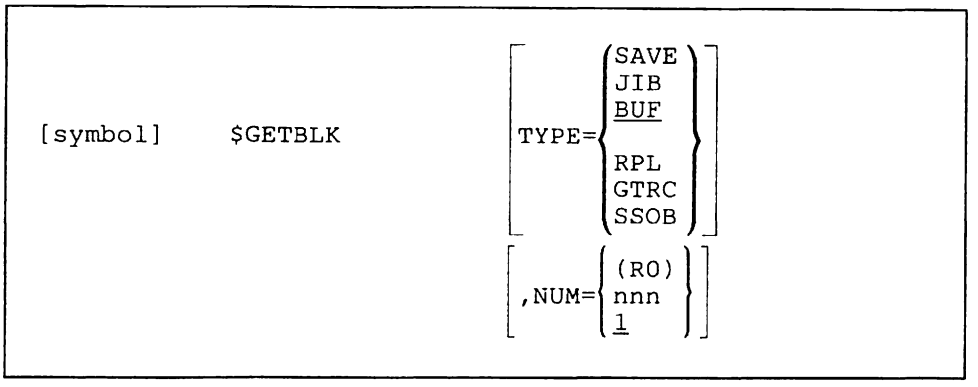
Environment

- Main task, subtask, or user address space.
- $WAIT cannot occur.

$GETBLK - Get a Storage Cell from a Free Cell Pool

Use $GETBLK to obtain a specified number of predefined storage cells from one of several free cell pools.

Format Description



TYPE=

Specifies the type of storage cell that is to be obtained and from which cell pool the cell is to be obtained. The following storage cell types may be coded:

Cell Type	Meaning
SAVE	An MVS-type save area
JIB	A JOE information block
BUF	An I/O buffer of 4K bytes
RPL	A request parameter list control block chain
GTRC	A GETREC chain control block
SSOB	A subsystem options block

NUM=

Specifies the number of storage cells that are to be obtained. Specify this number (nnn) as a valid number not greater than that specified on the $QCTGEN macro LIMIT= keyword or place the number in a register (Rn). If STACK=YES was coded on the $GETQC macro, the cells specified by this macro are chained in a stack. If NUM= is specified as greater than 1, the blocks will be chained using the chain field specified in the QCT for that storage type.

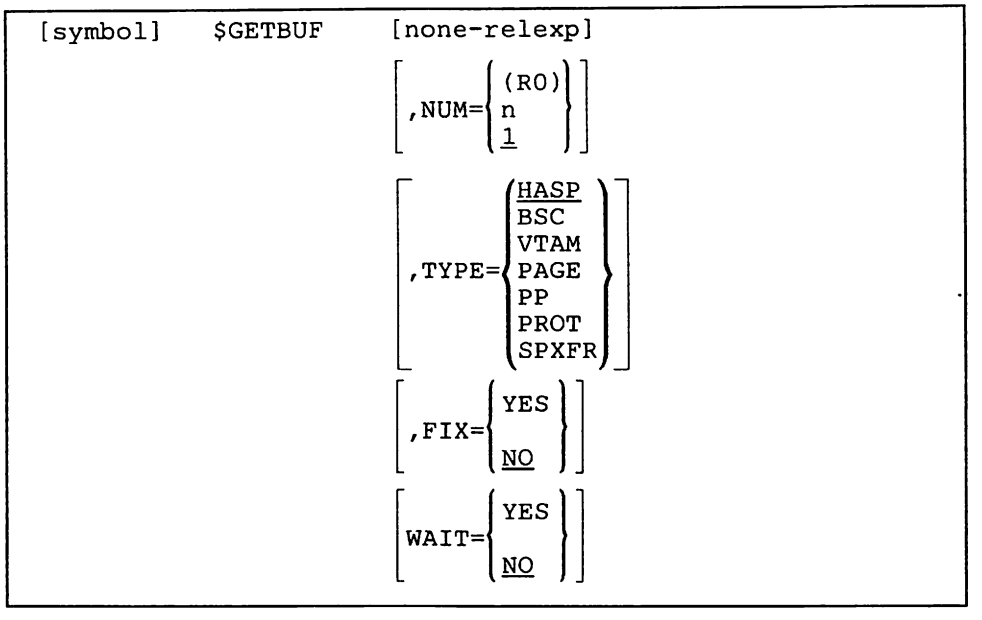
Environment

- Functional subsystem (HASPFSSM)
- MVS WAIT can occur.

$GETBUF - Acquire a Buffer from a JES2 Buffer Pool

Use $GETBUF in the JES2 main task or user environments to obtain a buffer from a buffer pool and return the address of this buffer in register 1.

Format Description



none

Specifies a location to which control is returned if there are no buffers available. (If WAIT=YES is specified, this operand is ignored.)

If this operand is omitted, the condition code is set to reflect the availability of a buffer as follows:

CC=0
No buffer is available.

CC≠0
R1 contains the address of the buffer.

NUM=

Specifies the number of buffers or a register containing the number of buffers to be obtained. (This keyword is ignored if this macro instruction is issued from the user environment.) One buffer is the default.

TYPE=

Specifies the type of buffer to be obtained, and whether the buffer is to contain an IOB or an RPL, by type code as follows:

HASP (default)
A local buffer in which an input/output buffer (IOB) is to be constructed

BSC
A teleprocessing (TP) buffer in which an IOB is to be constructed

VTAM
A TP buffer in which a request parameter list (RPL) is to be constructed

PAGE
A local 4096-byte buffer in which an IOB is to be constructed

PP
A local print/punch buffer in which an IOB is to be constructed

PROT
A protected buffer in which an IOB is to be constructed. This keyword value only applies when you use this macro in the user (HASPSSSM) environment.

SPXFR
A local spool offload buffer.

Note: You must specify WAIT=YES if TYPE=SPXFR is specified.

FIX=

Specifies whether the buffer is to be page-fixed (YES); if FIX=NO is specified or this parameter is omitted, the page containing the buffer is not fixed.

WAIT=

Specifies whether the $GETBUF service routine is to cause the routine issuing the macro to wait ($WAIT BUF) until buffers are available (WAIT=YES), or whether control is to be returned immediately (WAIT=NO or parameter omitted) if buffers are not available. If TYPE=SPXR is specified, you must also specify WAIT=YES

Note: TYPE= is the only keyword applicable in the user environment (HASPSSSM).

Caution: The JES2 buffer that is obtained by using the $GETBUF macro contains a prefix area that must not be altered. This prefix area is used by the $FREEBUF macro; unpredictable results may occur if the prefix area is altered.

Environment

- JES2 or user environment.
- $WAIT can occur if you specify WAIT=YES on the macro.

$GETCEL - Acquire a Common Storage Area Cell

Use $GETCEL to obtain a storage area for use in communicating information between the JES2 main task and the HASPSSSM module when user job and task ownership is required.

Format Description

```
[symbol]    $GETCEL     SJB={addrx}
                        ,TCB={addrx}
                        ,SIZE={value}
                        [,NONE=relexp]
```

SJB=

Specifies the primary ownership of the storage cell. The specification is a subsystem job block (SJB) address; therefore, the storage block is freed automatically when the job terminates. The user should ensure that the processor has control of the SJB prior to getting the storage cell and working with the control block.

If register notation is used, the designated register must contain the address of the SJB.

TCB=

Specifies the secondary ownership of the storage cell. This specification should either be 0 or the address of a task control block (TCB) associated with the SJB specified in the SJB operand. If a TCB address is specified, the storage cell automatically is freed upon termination of the job or task; therefore, the processor must ensure that logic between the processor and the end of task exit of the HASPSSSM module does not permit freeing of storage while the processor is working with the cell.

If register notation is used, the designated register must be loaded with the address of the TCB or set to 0 prior to execution of this macro instruction.

SIZE=

Specifies the storage size in bytes of the cell. The value specified must be between 1 and 32767, inclusive. If register notation is used, the designated register must be loaded with the value prior to execution of this macro instruction.

NONE=
Specifies the address of a routine that is to be given control if the storage cell was not obtained.

Notes:

1. *Upon obtaining a storage cell, the first word of the storage block contains the address of the controlling cell central element (CCE). This work must be left within the storage area if the $FRECEL macro instruction or HASPSSSM explicit cell freeing subroutine is to be used to free the storage.*

2. *Although the use of this macro instruction for obtaining storage cells is available only to routines within the JES2 main task processors, other tasks may use the HASPSSSM subroutines directly without using this macro instruction.*

3. *Registers 0, 1, 14, and 15 are used during the execution of this macro instruction.* **Caution: If the routines and macro instructions provided for control of storage cells are not used or the subsystem job block (SJB) and task interlocking mechanisms are not strictly followed, results are unpredictable and generally disastrous in nature.**

Environment

- Main task.
- $WAIT can occur.

$GETCMB - Get Console Message Buffers

Use $GETCMB to obtain one or more console message buffers from the free queue and return the address of the first buffer in register 1.

Format Description

```
[symbol]    $GETCMB    [NUMCMB={value|(R1)|1}]
                       [,COUNT={value|(R0)|0}]
                       [,SPECCT={YES|NO}]
                       [,WAIT={YES|NO}]
```

NUMCMB=

Specifies the number of console message buffers required. If register notation is used, the number of required console messages must be loaded into the designated register prior to the execution of this macro instruction.

If this operand is omitted, NUMCMB=1 is assumed.

COUNT=

Specifies the amount that the console lockout security count in SVTCOMCT should be decreased as a result of a successful $GETCMB execution. If the count goes to 0 (SPECCT=NO) or below an installation-specified minimum (SPECCT=YES), the $GETCMB request is rejected.

If register notation is used, the count must be loaded into the designated register prior to execution of this macro instruction.

If this operand is omitted, COUNT=0 is assumed.

SPECCT=

Specifies whether the console lockout security count in SVTCOMCT should go to 0 or the count should go below a system-specified minimum before rejecting the $GETCMB request. Specifications are as follows:

YES

Reject if the count goes below the system-specified minimum.

NO (default)
Reject if the count goes to 0

If this operand is omitted, SPECCT=NO is assumed.

WAIT=
Specifies the action to be taken in the event insufficient console message buffers (CMBs) are available to satisfy the request, as follows:

YES
Control is not returned until the request is satisfied.

NO (default)
An immediate return is made. The condition code on return has the following meaning:

CC=0
The request was rejected.

CC≠0
All console message buffers requested are chained together with register 1 pointing to the first and the chain field of the last buffer set to 0.

Notes:

1. *When console message buffers are obtained for the purpose of queuing the command to the command processor input queue, the COUNT specification should be the same as NUMCMB.*

2. *When console message buffers are obtained for the purpose of issuing a $WTO with the CLASS=$DOMACT operand specification, the COUNT specification should be the same as NUMCMB, and SPECCT=YES must be specified.*

3. *When console message buffers are obtained without specifying a corresponding count, the processor must not issue a $WTO without providing the CMB=YES parameter either directly or using a facility that does issue the $WTO. The processor also must not wait ($WAIT) for a resource owned by another processor that issues a $WTO or unconditionally attempts to get console message buffers. If these rules are violated, the system experiences serious console message buffer lockout problems.*

Environment

- Main task.
- $WAIT can occur (if you specify WAIT=YES on the macro).

$GETLOK - Acquire the MVS CMS, LOCAL, or JES2 Job Lock

Use $GETLOK to acquire either the MVS CMS, LOCAL, or JES2 job lock depending on the type of lock requested and the environment from which it is requested.

Also, use $GETLOK to obtain the cross-memory services (CMS) lock to serialize the JES2 main task with other routines executing on behalf of other tasks in the system. The CMS lock is required when modifying certain operating system control blocks and, in some cases, when interfacing with the HASPSSSM module. After obtaining the CMS lock, the user should not execute any code that allows the execution of any SVC instructions until after first freeing the lock via $FRELOK macro instruction.

Obtaining the local lock serializes the use of resources such as queues and control blocks among several tasks running within the same address space. The functional subsystem interface (FSI) service routines running in the functional subsystem address space require the local lock for serialization of queues, buffer pools, and control blocks such that many separate functional subsystem application (FSA) tasks can use these resources.

Obtaining the JES2 job lock prevents job queue elements (JQEs) from being changed by any code except the issuer of the job lock.

Note: The reason for executing the $GETLOK macro instruction should be fully documented in your code.

Format Description

```
[symbol]    $GETLOK    [TYPE={CMS                          }]
                       {LOCAL                         }
                       {[()]JOB,JQE=relexp[()]}
                       [WAIT={YES}]
                             {NO }
```

TYPE=

Specifies the lock to be obtained as follows:

CMS (default)
The cross-memory lock is to be obtained. All other operands are ignored.

LOCAL
The MVS local lock is to be obtained. All other operands are ignored.

JOB
The JES2 job lock is to be obtained. In this case a job queue element address (JQE=) must be specified.

JQE=

Specifies the address of a fullword containing the address of the specified JQE in its three right-most bytes. JQE= must be specified for TYPE=JOB.

WAIT=

Specifies whether or not to $WAIT for the JOB lock to be obtained. This keyword only applies to a TYPE=JOB request; otherwise, it is ignored. WAIT=YES is the default.

Environment

- Main task and functional subsystem (HASPFSSM).
- $WAIT can occur (if you specify WAIT=YES on the macro).

$GETMAIN - Branch-Entry GETMAIN Services

Use $GETMAIN in the JES2 main task or user environments to obtain an area of storage from MVS GETMAIN/FREEMAIN services or to free an area of storage obtained by this method.

Format Description

```
[symbol]   $GETMAIN   [type-code]
                      [,LV={value}]
                           {(R0)}
                      [,A={addrx}]
                          {(R1)}
                      [,SP={absexp}]
                           {0}
                      [,BNDRY={PAGE}]
                              {DBLWD}
                      [,FREMAIN={YES}]
                                {NO}
                      [,KEY={TCB}]
                            {PSW}
                            {1}
                      [,TCB={YES}]
                            {NO}
                            {JOBSTEP}
                      [,LOC={BELOW}]
                            {ANY}
```

type-code

Identifies the type of GETMAIN/FREEMAIN request. The types of requests are defined as follows:

Type	Meaning
R	An unconditional GETMAIN or FREEMAIN request
C	A conditional GETMAIN request that returns a condition code in register 15. (CC = 0 indicates a valid GETMAIN, CC > 0 indicates otherwise.)
RC	A conditional GETMAIN request that returns a condition code in register 15. (CC = 0 indicates a valid GETMAIN, CC > 0 indicates otherwise.)

U An unconditional FREEMAIN request. If type-code is not specified on this macro instruction, this is the default.

RU An unconditional FREEMAIN request. In the user's environment, this is the default.

BC A conditional GETMAIN request for a buffer that returns a condition code in register 15. (CC=0 indicates a valid GETMAIN, CC>0 indicates otherwise.) **BC can be used only in the user's environment.**

BU An unconditional FREEMAIN request for a buffer. **BU can be used only in the user's environment.**

LV=

Specifies the length of the area to be obtained or freed. This value is loaded into register 0.

A=

Specifies the address of the storage area to be freed. This keyword is only valid if FREMAIN=YES is also specified.

SP=

Identifies the subpool number. Subpool zero is the default if no subpool is specified. This parameter *must* be specified if you want to free an entire subpool. (In that case, do *not* code the A= parameter.)

BNDRY=

Specifies that the storage requested be located on either a page or doubleword boundary. This keyword is ignored if the type-code you specify indicates a FREEMAIN request.

PAGE
Indicates that the storage be located on a 4096-byte (page) boundary.

DBLWD
Indicates that the storage be located on a doubleword boundary.

FREMAIN= (applies only to the user environment)

Specifies whether this request is to free (FREMAIN=YES) or get (FREMAIN=NO) storage. Getting storage (FREMAIN=NO) is the default. FREMAIN can only be used in the user's environment.

KEY= (applies only to the user environment)

Specifies the key of the storage that is either to be acquired or freed. If the keyword is omitted, KEY=1 is used.

TCB indicates to use the TCBPFK key of the current TCB.

PSW indicates to use the current PSW storage key.

TCB= (applies only to the user environment)
Specifies what TCB protect key to use for this GETMAIN or FREEMAIN.

YES - indicates the current TCB.

NO (the default) - indicates that a TCB protect is not being used.

JOBSTEP - indicates the jobstep TCB.

Notes:

1. *You set the global assembly variable &ANVIRON=USER prior to executing the $GETMAIN macro in order to select the user environment.*

2. *You set the global assembly variable &ANVIRON=JES2 prior to executing the $GETMAIN macro in order to select the JES2 main task environment.*

Environment

- Main task or user address space.
- $WAIT can occur for the main task environment.
- $WAIT cannot occur for the user environment.

$GETQC - Call the Quick Cell Get Routine.

Use $GETQC to obtain storage cells from a previously-built quick cell pool. This macro instruction assists the efficient management of the work areas and buffer storage requirements for the JES2 functional subsystem support routines (HASPFSSM). The dynamic quick cell feature provides a high-performance method for allocation and deallocation of fixed-length cells of storage. These quick cells (that is, quickly obtainable blocks of storage) are defined, created, and controlled in the quick cell control table (QCT) for use by the functional subsystem support routines (HASPFSSM). Cell types include save areas, I/O buffers, and JOE information blocks (JIBs).

Format Description

```
[symbol]      $GETQC      TYPE={type-code}
                          {(R0)       }
                         [,NUM={nnn  }]
                          [    {(Rn) }]
                          [    {1    }]
                         [,STACK={YES}]
                          [      {NO }]
```

TYPE=

Specifies the type as defined in the $QCTGEN macro for the quick cell to be obtained.

type-code

Specifies that the type-code and all equated symbols for each type-code are defined in the $QCTGEN macro.

(R0)

Specifies that the register contains the quick cell type-code as defined in the $QCTGEN macro, the two low-order bytes must contain the type-code and the two high-order bytes must be zeroed.

Note: The TYPE= keyword must be specified.

NUM=

Specifies the number of quick cells to get from the quick cell pool. The value assigned to NUM= must not exceed the specification of QCTLIMIT; exceeding this value will cause an error condition.

nnn

Specifies the number (1-255) of quick cells. The default value is 1.

(Rn)
Specifies that register notation is used. The register specified contains the number of quick cells.

STACK=
Specifies whether the quick cells should be pushed onto a stack or chained together. The chaining field offset is specified in the QCT.

YES
Specifies that the quick cells specified by this macro are pushed onto a stack identified by the QCT.

NO
Specifies that the quick cells specified by this macro are chained together and the address of the head of the chain is passed back to the caller.

Note: Register 11 must contain the address of the HASP function control table (HFCT) prior to executing $GETQC.

Environment

- Functional subsystem (HASPFSSM).
- MVS WAIT can occur.

$GETSMFB - Acquire a JES2 SMF Buffer from the JES2 SMF Buffer Pool

Use $GETSMFB to obtain a buffer from the JES2 SMF buffer pool, clear the buffer contents to binary zeroes, and return the address of this buffer in register 1. The macro returns condition code 0 and a 0 in register 1 if no buffers were available and WAIT=NO was specified in the macro.

Format Description

```
[symbol]    $GETSMFB    [ WAIT={ YES }  ]
                           { NO  }
```

WAIT=

Specifies the action to be taken in the event no JES2 SMF buffers are available as follows:

YES

Control is not returned to the caller until a JES2 SMF buffer has become available.

NO

An immediate return is made. If no buffers are available, register 1 contains a 0 upon return to the calling routine. The condition code is nonzero if a buffer is available or 0 if no buffers are available.

Environment

- Main task.
- SWAIT can occur (if you specify WAIT=YES on the macro).

$GETUCBS - Obtain a UCB Address

Use $GETUCBS to obtain a single UCB address or a number of UCB addresses.

Format Description

```
[symbol]    $GETUCBS    CLASS={(R0)}
                           {(Rn)}
                          [,CONT={YES}]
                                 {NO }
```

CLASS =
Specifies the device class of the UCB or UCBs that are requested.

CONT =
Specifies whether a single UCB is to be obtained (CONT = NO) or whether multiple UCB addresses are to be obtained (CONT = YES). CONT = NO is the default. **Register 1 must be preserved in the loop when CONT = YES.**

Environment

- Main task or during JES2 initialization and termination.
- $WAIT cannot occur.

$GETUNIT - Acquire a Unit Device Control Table (DCT)

Use $GETUNIT to assign a device control table (DCT) to a specific device.

Format Description

```
[symbol]    $GETUNIT    {dct-addrx}
                        {(R1)    }
                        [,not-avail-relexp]
```

dct

Specifies either a pointer to a DCT or the address of a DCT to be obtained. If dct is written as an address, then it represents the address of a full word containing the address of the DCT to be obtained. If dct is written using register notation (either regular or special register notation), then it represents the address of the DCT to be obtained. If register notation is used, the address must be loaded into the designated register before the execution of the macro instruction. **DCT address must be specified.**

not-avail

Specifies a location to which control is returned if the specified DCT is not available. If this operand is omitted, the condition code is set to reflect the availability of a DCT as follows:

CC=0
The DCT is not available.

CC≠0
R1 contains the address of the available DCT.

Environment

- Main task.
- $WAIT cannot occur.

$GETWORK - Obtain a Work Area

Use $GETWORK to obtain a work area in subpool 1 in the JES2 address space. If the size of the requested area is appropriate, storage is allocated from JES2-maintained storage pools.

Format Description

```
[symbol]    $GETWORK    WORDS={number}
                        {(Rn)  }
                        ,USE=code
                        {[,ERRET={label}]}
                        {        {(Rn) }}
                        {[,WAIT={YES}]  }
                        {       {NO }   }
```

WORDS=
Specifies the size of the work area in full words or a register that contains the size of the work area in full words.

USE=
Specifies the 4-character identifier to be placed in the first four bytes of the work area.

The address returned in register 1 will be four bytes into the actual work area (bypassing the prefix). You do not have to adjust this value to point past the identifier.

JES2 issues catastrophic error $GW1 if the size requested on the WORDS= operand is greater than the largest size work area supported.

ERRET=
Specifies the label or register (R2-R12) that indicates where to branch if $GETWORK cannot successfully allocate required storage. If ERRET= is not specified or if the allocate fails for any reason, $GETWORK issues the catastrophic ABEND, GW1. If ERRET is specified, $GETWORK issues a catastrophic abend for internal errors only, for other errors, (e.g., a GETMAIN failure) register 15 returns a return code of 4. This keyword is not valid if WAIT= YES is also specified.

WAIT
Specifies whether (YES) or not (NO) the $GETWORK service routine is permitted to $WAIT for storage. This keyword is not valid if ERRET= is also specified.

LOC=

Specifies the location of the virtual and real storage to be allocated.

BELOW

Indicates that the storage is to be allocated below the 16-megabyte line.

ANY

Indicates that the storage can either be located above or below the 16-megabyte line.

Environment

- Main task.
- $WAIT can occur if WAIT=YES is specified.

$IOERROR - Log Input/Output Error

Use $IOERROR to log an input/output error on the operator's console.

Format Description

```
[symbol]    $IOERROR    { buffer-addrx }
                        { (R1)         }
```

buffer

Specifies either a pointer to a JES2 buffer or the address of the buffer that has been associated with a JES2 input/output error.

If buffer is written as an address, it represents the address of a fullword that contains the address of the buffer in error in its 3 rightmost bytes. If buffer is written using register notation (either regular or special register notation), it represents the address of the buffer in error. If register 1 is used, the address must be loaded into the register before the execution of the macro instruction.

Environment

- Main task.
- $WAIT can occur.

$JCAN - Cancel Job

Use $JCAN to prepare the job represented by the specified job queue element for the cancellation of its normal execution, cancellation of its normal execution as well as output, or cancellation upon completion of its current activity.

Format Description

```
[symbol]    $JCAN       [ JQE={ addrx  } ]
                        [     { (R1)   } ]
                        [ ,ENTER={ YES } ]
                        [        { NO  } ]
                        [ ,TYPE={ CANXEQ } ]
                        [       { CANALL } ]
                        [       { STOP   } ]
                        [,NOTJOB=relexp]
                        [,NOP=relexp]
                        [,OK=relexp]
                        [ ,DUMP={ YES } ]
                        [       { NO  } ]
                        [,NOJOE=relexp]
                        [,OFFLINE=relexp]
```

JQE=

Specifies the address of the job queue element that represents the job to be cancelled. If register notation is used,[the address must be loaded into the designated register before the execution of this macro instruction unless ENTER=NO is specified.

ENTER=

Specifies whether or not actual entry to the job cancel service routine is to be affected. If this operand is omitted or YES is specified, the routine is entered. If the specification is NO, the execution of this macro instruction is for the purpose of setting parameter values in register 0 based upon the specifications in the TYPE and DUMP operands. Operands other than ENTER, TYPE, and DUMP should be omitted when ENTER=NO is specified, and the value of register 0 must not be altered until after a subsequent $JCAN macro with ENTER=YES specified or omitted.

TYPE=

Specifies the action to be taken. If CANXEQ is specified, a normal batch job, which is in the system queues prior to execution or in execution, is queued for output. A request for a job in $OUTPUT is considered a no operation, and control is given to the location specified by the NOP

operand. If the job is a started task control (STC) or time-sharing user (TSU) job prior to being executed or in execution, the request is rejected, and control is given to the location specified by the NOTJOB operand. If CANALL is specified, the job is cancelled from its current activity and queued for purge. If the job is an STC or TSU job prior to or in execution, the request is rejected, and control is given to the location specified by the NOTJOB operand. If STOP is specified, the action is the same as for CANALL except that the job's current activity is not deleted. If this operand is omitted, register 0 must have been set by a previous execution of a $JCAN macro instruction specifying ENTER=NO.

Notes:

1. *If the job queue element is currently owned by a processor, queuing to $OUTPUT or $PURGE is delayed until the next $QMOD, $QPUT, or $QADD macro instruction execution.*

2. *The CANXEQ function may be negated by the execution processor if a reenqueue function is requested.*

3. *The CANXEQ function results in cancellation of output if a previous request has been made using the STOP function request.*

NOTJOB=
Specifies the location to be given control if the job to be cancelled is a STC or TSU job in the system prior to or in execution.

NOP=
Specifies the location to be given control if TYPE=CANXEQ is specified and the job has passed the execution phase. If this operand is omitted, control is given to the location specified by the OK operand.

OK=
Specifies the location to be given control if the execution of the request is successful. If this operand is omitted, control is given to the location following the macro instruction if the request is successful.

DUMP=
Specifies whether or not the system is to attempt a storage dump of the specified job whose execution is being canceled. If the specification is DUMP=YES and TYPE=CANXEQ or TYPE=CANALL is specified, and if the job is in execution and is not an STC or TSU job, the system attempts to dump the job in a manner compatible with the MVS CANCEL jobname, DUMP command.

NOJOE=
Specifies the location to be given control if the specified number of JOEs were found.

OFFLINE=
Specifies the location to be given control if the job's spool(s) is offline.

Note: OK=, NOJOE=, OFFLINE= must all be specified in order for any one of them to be recognized.

Environment

- Main task.
- $WAIT can occur.

$JCTIO - Call the $JCTIOR Routine

Use the $JCTIO macro instruction to call the $JCTIOR routine in HASPNUC to read or write a JCT from spool.

Format Description

```
[symbol]   $JCTIO   TYPE={READ }
                    {WRITE}
                   ,JQE={field-name}
                        {(Rn)      }
                  [,JCTBUF={field-name}]
                           {(Rn)      }
                  [,FREE={YES}]
                         {NO }
                  [,ERRET={field-name}]
                          {(Rn)      }
```

TYPE=
Specifies the type (READ or WRITE) of I/O operation requested by this macro.

JQE=
Specifies a register or the name of a field that contains the address of the JQE for the job whose JCT is required to be read or written.

JCTBUF=
Specifies a register or the name of a field that contains the address of the buffer into which the JCT is to be read or from which the JCT is to be written.

Note: This keyword is required if TYPE=WRITE is specified.

FREE=
Specifies whether (YES) or not (NO) to free the JCT buffer following the I/O operation. If TYPE=READ is specified, this keyword is ignored.

ERRET=
Specifies the name of the field that contains the address to which JES2 branches if an I/O error is detected or an invalid JCT is read.

Environment

- JES2 main task
- $WAIT can occur

$MID - Assign JES2 Message Identification

Use $MID to set the global variable[symbol] &MID to an EBCDIC character string so that, when the variable[symbol] is coded as the first portion of the message text field of an operating system WTO macro instruction, the correct message identification is displayed with the message. This macro instruction should be coded directly prior to the WTO macro instruction.

Format Description

```
[symbol]  $MID  id-value
```

id-value
Specifies the numeric 3-digit message identification of the message appearing in the succeeding WTO macro expansion.

Note: Coding should not depend upon the exact length or format of the character string assigned to the &MID variable symbol.

Environment

- Main task, subtask, or user address space.
- $WAIT cannot occur.

$MODCHK - Call the MODCHECK Verification Routine

Use the $MODCHK macro instruction to call the MODCHECK verification routine located in the HASPNUC load module. The MODCHECK routine can test:

1. if the module resides below the 16-megabyte line
2. if the module resides in common storage
3. if the module assembled at the same version as the JES2 nucleus and with the correct level of macros
4. if the module name matches that specified in the MIT (MITNAME)
5. if the exit point can be propagated from the MIT to the exit information table (XIT)
6. if the XIT entry address can be resolved in the exit routine table (XRT).

Specifically, this macro is useful to guarantee that you have not inadvertently attempted to mix MVS versions and that all modules are assembled at the same system product (SP) level. This early verification and notification prevents an attempt by JES2 to load an incorrect module and eventually terminate. An unsuccessful verification causes JES2 to issue message $HASP875 with a specific reason text.

Format Description

```
[symbol] $MODCHK NAME={'xxxxxxxx'}
                      {field-label}
                      {(R1)       }

                  {[,ADDR={field-name}]}
                  {      {(R0)      }  }
                  {[,LOAD={YES}]      }
                  {       {NO }       }

                        {RMODE24  }
                        {,COMMON  }
                        {,MIT     }
                ,TEST=[(]{,VERSION  }[)]
                        {,NAME    }
                        {,EXITPTS }
                        {,EXITRTNS}

                  [,ERRET={field-name}]
                          {(Rn)      }

                  [,MESSAGE={YES}]
                            {NO }
```

NAME=

Specifies the name of the module to be verified by MODCHECK. Specify a 1- to 8-character module name, a label referencing the beginning of the module, or a register containing the address of the module. This is a required keyword.

ADDR=

Specifies the address of this module by either a field name or a register containing the module address. If LOAD=YES is specified, this parameter must not also be coded.

LOAD=

Specifies that the module should (LOAD=YES) be loaded prior to verification checking. If LOAD=NO is specified, you must specify ADDR=. If LOAD=YES is specified, do not also code the ADDR= keyword.

TEST=

Specifies the tests for which this module is to be verified. If you specify more than one test type, enclose the list in parenthesis and separate each by a comma, for example, TEST=(NAME,RMODE24,VERSION)

Test Type	Meaning
RMODE24	Tests that the module resides below the 16-megabyte line
COMMON	Tests that the module resides in common storage
MIT	Tests that the module is large enough to contain the MIT, the MIT entry table pointer points to a valid field within the module, and that the MIT is located at the beginning of the module. This test is only valid if LOAD=YES is specified.
VERSION	Tests that the version of JES2 and this module are at the same level and that all macros contained in the module are assembled at the correct level of JES2.
NAME	Tests that the NAME= keyword specifies the same name as the MITNAME specified in the MIT.
EXITPTS	Test if the exit point can be propagated from the MIT to the XIT.
EXITRTNS	Test if the XIT entry address can be resolved in the XRT.

ERRET=

Specifies the address of an error routine that is to get control if the module is not successfully verified by MODCHECK. Specify the address by either a field name or a register containing the address of this module. An error message can be returned based if MESSAGE=YES is also coded.

MESSAGE=

Specifies whether (YES) or not (NO) JES2 will issue message $HASP875 with an appropriate text if an error is detected by MODCHECK.

Return Code	Meaning
0	Test(s) and load (if requested) successful
4	Load failed
8	Verification test(s) failed -- the previously loaded module is deleted

Environment

- JES2 main task
- MVS WAIT may occur

$MODEND - Generate End of Module

Use $MODEND to generate the MIT entry table (MITETBL) to fill in the 256-bit mask field in the MIT according to what exits are defined within the module, and to calculate the length for the CSECT created by $MODULE.

This macro instruction must be coded at the end of every module, with no exceptions.

Format Description

```
[symbol]   $MODEND
```

Environment

- Main task, subtask, or user address space.
- $WAIT is not applicable.

$MODULE - Provide a Module Information Table

Use $MODULE to generate a module information table (MIT) for the given module to establish the values of assembler global variables, and to generate DSECTs and EQUs for the assembly. It must be coded once in a module and must appear immediately after a COPY of $HASPGBL and before any other code. **There are no JES2 modules that are exceptions to this rule.**

Every time a $ENTRY macro instruction is invoked to establish a new entry point, another entry is added to the MIT entry table (MITETBL). Also, within the MIT there is a 256-bit mask used to indicate which exit points are defined within the given module.

A number of JES2 mapping macros require that you also request one or more JES2 or MVS macros to provide necessary mapping support. If you do not explicitly include the required macro(s), JES2 automatically generates them and issues a warning note during module assembly. For example, the JES2 macro $PSO requires that you code the JES2 macro, $PDDB, and the MVS macro SSOB. But if you only included $PSO, JES2 will generate both the missing JES2 and MVS macros.

Format Description

```
[symbol]    $MODULE     [NOTICE={NONE       }]
                        [       {value      }]
                        [       {(value,value)}]

                        [,SYSP=(print,gen,data,listmvs,listjes)]

                        [,TITLE=module title]

                        [          {JES2   }]
                        [,ENVIRON={USER    }]
                        [          {SUBTASK}]
                        [          {FSS    }]

                        [,ENTRIES=(name1,name2,...)]

                        [,dsectname,dsectname,...]
```

symbol
Specifies the module name to appear in all cross reference lists and to be the name of the CSECT stated by the $MODULE macro.

NOTICE=
Specifies whether or not the IBM copyright notice is to be generated and what release applies to this module. The JES2 release that created and last modified the module can be specified. Or a list of the creating release

followed by the last modifying release can be specified. The value can be coded as follows:

NONE (the default)
No copyright notice is generated.

SP130
Indicates that this module was created or last updated by SP1.3.0 (SP1.2.0).

SP133
Indicates that this module was created or last updated by SP1.3.3.

SP134
Indicates that this module was created or last updated by SP1.3.4.

SP136
Indicates that this module was created or last updated by SP1.3.6.

SP215
Indicates that this module was last created or updated by SP2.1.5.

SYSP=(print,gen,data,listmvs,listjes)
Specifies the parameter defaults for global assembly variables.

Note: These parameters can be overridden by using the assembler 'SYSPARM' option.

print
Specifies the JES2 default print option for certain expansions with JES2 module DSECTs. This specification is used in an assembler PRINT instruction and must be either GEN or NOGEN. The default is NOGEN.

gen
Specifies a default specification for the printing non-mapping macro expansions in a JES2 assembly. This specification is used in an assembler PRINT instruction and must be either CBREQ, GEN, or NOGEN.

data
Specifies a default specification for the printing of data constants in a JES2 assembly. This specification is used in an assembler PRINT instruction and must be either DATA or NODATA.

listmvs
Specifies a default specification for the printing of the MVS control block DSECTs. This specification must be either GEN or NOGEN.

listjes
Specifies a default specification for the printing of JES2 control block DSECTs. This specification must be either GEN or NOGEN.

TITLE=
Specifies a character string title for this module.

ENVIRON=
Specifies to what environment this module is to apply

JES2 (default)
indicates the JES2 main task environment

SUBTASK
indicates the JES2 subtask environment

USER
indicates the user (HASPSSSM) environment

FSS
indicates the functional subsystem (HASPFSSM) environment

You specify the ENVIRON= keyword to set the &ANVIRON global assembly variable, which is used by other macros; this specification will be the default for the ENVIRON= keyword for other macros.

ENTRIES=
Specifies additional entry point names in your module that are not defined by the $ENTRY macros. These additional entry names are added to the MIT entry table (MITETBL).

dsectname,dsectname,...
Identifies the MVS and JES2 DSECT mappings that are required to assemble this module. $MODULE expands the requested mappings using the appropriate MVS and JES2 macros. The dsect names can be specified in any order. They will be expanded in the proper order that provides for the limitations of the assemblers.

Each dsectname value can be specified as one of the DSECTIDs shown in the table below, or as (dsectid,genid). genid can either be specified as GEN or NOGEN and is used to override the "listmvs" and "listjes" values (for the particular macro) specified via the SYSP operand.

DSECTID	Macro(s)	Description of Code Generated
ACB	IFGACB	MVS access method control block DSECT
ACBXL	IFGEXLST	MVS access method control block exit list DSECT
ACEE	IHAACEE	MVS accessor environmental element control block DSECT
ASCB	IHAASCB	MVS address space control block DSECT
ASVT	IHAASVT	MVS address space vector table DSECT
ASXB	IHAASXB	MVS address space extension block DSECT
ATB	IECDATB	MVS attention table DSECT
BASEA	IEEBASEA	MVS master scheduler resident data area DSECT
CDE	IHACDE	MVS contents directory entry DSECT
CMAUP	IEFCMAUP	MVS common authorization check parameter list
CSCB	IEECHAIN	MVS command scheduling control block DSECT
CVT	CVT	MVS communication vector table DSECT
DCB	DCBD	MVS data control block DSECT
DEB	IEZDEB	MVS data extent block DSECT
DYN	IEFZB4D0	MVS dynamic allocation (SVC 99) parameter list
DYN	IEFZB4D2	MVS dynamic allocation (SVC 99) text unit keys
ECB	IHAECB	MVS event control block DSECT
ETD	IHAETD	MVS cross-memory entry table description DSECT
EWA	EWAMAP	MVS IOS/ERP error work area DSECT
FDA	IKJEFFDF	MVS DAIRFAIL parameter list DSECT
FSCT	IAZFSCT	MVS functional subsystem control table DSECT
FSIP	IAZFSIP	MVS functional subsystem interface parameter DSECT
FSVT	IAZFSVT	MVS functional subsystem vector table DSECT
GDA	IAHGDA	MVS global data area DSECT
IDX	IAZIDX	MVS print/punch SYSOUT record index DSECT
IOCM	IECDIOCM	MVS I/O supervisor communication area DSECT
IOSB	IECDIOSB	MVS I/O supervisor block DSECT
JESCT	IEFJESCT	MVS job entry subsystem control table DSECT
JFCB	IEFJFCBN	MVS job file control block DSECT
JFCB	IEFJFCBE	MVS job file control block 3800 extension DSECT
JSCB	IEZJSCB	MVS job step control block DSECT
JSPA	IAZJSPA	MVS FSI job separator page area DSECT
KEYS	IEFVKEYS	MVS internal text key definitions
LCT	IEFALLCT	MVS linkage control table DSECT
MGCR	IEZMGCR	MVS MGCR (SVC 34) parameter list DSECT
NEL	IEFNEL	MVS JCL converter/interpreter entry list DSECT
NIB	ISTDNIB	MVS VTAM node initialization block DSECT
ORE	IHAORE	MVS operator reply element DSECT
PDS	IHAPDS	MVS partitioned data set directory entry DSECT
PPL	IECDPPL	MVS protected step control block DSECT
PSA	IHAPSA	MVS prefixed save area DSECT
PSCB	IKJPSCB	MVS protected step control block DSECT
RB	IKJRB	MVS request block DSECT
RESPA	IAZRESPA	MVS FSI order response area DSECT
RMR	IDARMRCD	MVS VSAM record management return codes
RPL	IECRPL	MVS ACB request parameter list DSECT
RQE	IECDRQE	MVS EXCP request queue element DSECT
SDWA	IHASDWA	MVS system diagnostic work area DSECT
SIOT	IEFASIOT	MVS step I/O table DSECT
SJDLP	IEFSJDLP	MVS SJF delete SWB parameter list DSECT
SJEXP	IEFSJEXP	MVS SJF extract parameter list DSECT
SJFNP	IEFSJFNP	MVS SJF find SWB chain parameter list DSECT
SJGEP	IEFSJGEP	MVS SJF get SWB parameter list DSECT
SJPRFX	IEFSJPFX	MVS SJF NJE SWB prefix parameter list DSECT
SJPUP	IEFSJPUP	MVS SJF put SWB parameter list DSECT
SJREP	IEFSJREP	MVS SJF retrieve SWB parameter list DSECT
SJRUP	IEFSJRUP	MVS SJF update SWB parameter list DSECT
SMCA	IEESMCA	MVS SMF control table DSECT
SPP	IHASPP	MVS SETPRT parameter list DSECT
SRB	IHASRB	MVS service request block DSECT
SSCT	IEFJSCVT	MVS subsystem communications vector table DSECT
SSIB	IEFJSSIB	MVS subsystem identification block DSECT
SSOB	IEFJSSOB	MVS subsystem options block DSECT

DSECTID	Macro(s)	Description of Code Generated
TCB	IKJTCB	MVS task control block DSECT
TCT	IEFTCT	MVS SMF timing control table DSECT
TEXT	IEFTXTFT	MVS internal text format DSECT
TIOT	IEFTIOT1	MVS task I/O table DSECT
UCB	IEFUCBOB	MVS unit control block DSECT
UCM	IEECUCM	MVS unit control module DSECT
UCSIT	IGGUCSIT	MVS UCS image table DSECT
WPL	IEZWPL	MVS WTO/WTOR/MLWTO/WTP parameter list DSECT
WQE	IHAWQE	MVS WTO queue element DSECT
XTLST	IHAXTLST	MVS extent list DSECT
$ACT	$ACT	HASP automatic command table DSECT
$APT	$APT	HASP NJE/SNA application table DSECT
$BFD	$BFD	HASP generalized subsystem dataset buffer DSECT
$BFW	$BFW	HASP 3800 buffer work area DSECT
$BPM	$BPM	HASP buffer pool map DSECT
$BTG	$BTG	HASP badtrack group element DSECT
$BUFFER	$BUFFER	HASP I/O buffer DSECT
$CAT	$CAT	HASP class attribute table DSECT
$CCA	$CCA	HASP cell control area DSECT
$CCE	$CCE	HASP cell control element DSECT
$CCW	$CCW	HASP channel command word definitions
$CHK	$CHK	HASP (MVS) FSI checkpoint record DSECT
$CIRWORK	$CIRWORK	HASP common initialization routine PCE work area
$CK	$CK	HASP checkpoint block DSECT
$CKPWORK	$CKPWORK	HASP checkpoint processor PCE work area DSECT
$CMB	$CMB	HASP console message buffer DSECT
$CNVWORK	$CNVWORK	HASP conversion processor PCE work area DSECT
$COMWORK	$COMWORK	HASP command processor PCE work area DSECT
$CPT	$CPT	HASP compaction table DSECT
$CSA	$CSA	HASP console service area DSECT
$CWA	$CWA	HASP MCS console work area DSECT
$DAIR	$DAIR	HASP DAIRFAIL parameter list DSECT
$DAS	$DAS	HASP direct access spool data set DSECT
$DCT	$DCT	HASP device control table DSECT
$DCTTAB	$DCTTAB	HASP DCT table ($GETABLE) DSECT
$DSSCB	$DSSCB	HASP data set services DSECT
$DTE	$DTE	HASP daughter task element DSECT
$DTEACCT	$DTEACCT	HASP account (SMF) subtask DTE work area DSECT
$DTECNV	$DTECNV	HASP converter subtask DTE work area DSECT
$DTEIMG	$DTEIMG	HASP image subtask DTE work area DSECT
$DTEOFF	$DTEOFF	HASP spool offload subtask DTE work area DSECT
$DTESPL	$DTESPL	HASP spool subtask DTE work area DSECT
$DTETAB	$DTETAB	HASP DTE table ($GETABLE) DSECT
$DTEVTAM	$DTEVTAM	HASP VTAM subtask DTE work area DSECT
$DVA	$DVA	HASP device type parameter area DSECT
$ERA	$ERA	HASP error recovery area DSECT
$ERPL	$ERPL	HASP $ERROR parameter list DSECT
$EST	$EST	HASP estimated counts DSECT
$EXITPL	$EXITPL	HASP $EXIT parameter list DSECT
$FMH	$FMH	HASP SNA function management header DSECT
$FSACB	$FSACB	HASP functional subsystem application control block DSECT
$FSAXB	$FSAXB	HASP functional subsystem application extension DSECT
$FSIEQU	$FSIEQU	HASP FSI equates
$FSSCB	$FSSCB	HASP functional subsystem control block DSECT
$FSSWORK	$FSSWORK	HASP functional subsystem support PCE work area DSECT
$FSSXB	$FSSXB	HASP functional subsystem control block extension DSECT
$GCB	$GCB	HASP GETREC chain control block DSECT
$HASPEQU	$HASPEQU	HASP equates (general register and bit definitions)
$HCT	$HCT	HASP communication table DSECT
$HDP	$HDP	HASP Control block header DSECT
$HFCT	$HFCT	HASP FSS communication table DSECT
$ICE	$ICE	HASP SNA interface control element DSECT
$IOT	$IOT	HASP I/O table DSECT

DSECTID	Macro(s)	Description of Code Generated
$JAW	$JAW	JES2 authorization work area
$JCT	$JCT	HASP job control table DSECT
$JDR	$JDR	HASP job disposition request DSECT
$JDRWORK	$JDRWORK	HASP JDR processor PCE work area DSECT
$JFL	$JFL	HASP JCL facility list DSECTS
$JFW	$JFW	HASP JCL facility work area DSECT
$JIB	$JIB	HASP JOE information block DSECT
$JOE	$JOE	HASP job output element DSECT
$JOT	$JOT	HASP job output table DSECT
$JQE	$JQE	HASP job queue element DSECT
$KEYLIST	$KEYLIST	HASP SWB keylist table entry DSECT
$KIT	$KIT	HASP checkpoint information table DSECT
$LCK	$LCK	HASP spool offload checkpoint element DSECT
$LGRR	$LGRR	HASP LOGREC record SDWAVRA DSECT
$LMT	$LMT	HASP load module table DSECT
$LRC	$LRC	HASP logical record DSECT
$MCODE	$MCODE	HASP BSC code table DSECT
$MCT	$MCT	HASP master control table DSECT
$MIT	$MIT	HASP module information table DSECT
$MITETBL	$MITETBL	HASP module information table entry table DSECT
$MLMWORK	$MLMWORK	HASP line manager processor PCE work area DSECT
$MODMAP	$MODMAP	HASP module map directory DSECT
$NAT	$NAT	HASP network nodes attached table
$NCC	$NCC	HASP network connection control DSECT
$NETACCT	$NETACCT	HASP network account table format and DSECT
$NHD	$NHD	HASP network header/trailer DSECTS
$NIT	$NIT	HASP network information table DSECT
$NJTWORK	$NJTWORK	HASP network job transmitter processor PCE work area DSECT
$NMR	$NMR	HASP network communication message record DSECT
$NPMWORK	$NPMWORK	HASP network path manager processor PCE work area DSECT
$NSP	$NSP	HASP network services procedure RU-cleanup unit DSECT
$NSRWORK	$NSRWORK	HASP network SYSOUT receiver processor PCE work area DSECT
$NSTWORK	$NSTWORK	HASP network SYSOUT transmitter processor PCE work area DSECT
$OCR	$OCR	HASP output control record DSECT
$OCT	$OCT	HASP output control table DSECT
$OUTWORK	$OUTWORK	HASP output processor PCE work area DSECT
$PCE	$PCE	HASP processor control element DSECT
$PCETAB	$PCETAB	HASP PCE table ($GETABLE) DSECT
$PCIE	$PCIE	HASP program controlled interrupt element DSECT
$PDDB	$PDDB	HASP peripheral data definition block DSECT
$PIT	$PIT	HASP partitioned information table DSECT
$PPPWORK	$PPPWORK	HASP print/punch processor PCE work area DSECT
$PQE	$PQE	HASP 3800 page queue entry DSECT
$PQH	$PQH	HASP 3800 pending page queue header DSECT
$PRE	$PRE	HASP processor recovery element DSECT
$PRGWORK	$PRGWORK	HASP purge processor PCE work area DSECT
$PRMD	$PRMD	HASP process mode table entry DSECT
$PSO	$PSO	HASP process SYSOUT work area DSECT
$PSOWORK	$PSOWORK	HASP PSO processor PCE work area DSECT
$QCT	$QCT	HASP quick cell control table DSECT
$QGET	$QGET	HASP $QGET parameter list DSECT
$QSE	$QSE	HASP shared queue control element DSECT
$RAT	$RAT	HASP remote attribute table DSECT
$RCPWORK	$RCPWORK	HASP remote console processor PCE work area DSECT
$RDRWORK	$RDRWORK	HASP reader services PCE work area DSECT
$RDT	$RDT	HASP remote destination table DSECT
$RESWORK	$RESWORK	HASP resource manager PCE work area DSECT
$RVSTACK	$RVSTACK	HASP error recovery work stack DSECT
$RWL	$RWL	HASP remote work look-up table
$SCANTAB	$SCANTAB	HASP SCAN table ($SCAN) DSECT
$SCANWA	$SCANWA	HASP $SCAN facility work area DSECT
$SCAT	$SCAT	HASP SYSOUT class attribute table DSECT
$SCR	$SCR	HASP spool control record DSECT

DSECTID	Macro(s)	Description of Code Generated
$SDB	$SDB	HASP subsystem data set block DSECT
$SJB	$SJB	HASP subsystem job block DSECT
$SJXB	$SJXB	HASP subsystem job extension block DSECT
$SMF	$SMF	HASP SMF buffer DSECT
$SPMWORK	$SPMWORK	HASP spool manager processor PCE work area DSECT
$SVT	$SVT	HASP (MVS) subsystem vector table DSECT
$SWBIT	$SWBIT	HASP SWB information table DSECT
$TAB	$TAB	HASP track allocation block DSECT
$TGB	$TGB	HASP allocation track group block DSECT
$TIDTAB	$TIDTAB	HASP trace ID table ($GETABLE) DSECT
$TLGWORK	$TLGWORK	HASP trace log processor PCE work area DSECT
$TQE	$TQE	HASP timer queue element FORMAT
$TRCA	$TRCA	HASP termination/recovery control area DSECT
$TRP	$TRP	HASP $TRACE parameter list DSECT
$TTE	$TTE	HASP trace table entry DSECT
$UPL	$UPL	HASP UCB parameter list DSECT
$VERTAB	$VERTAB	HASP $VERIFY table entry DSECT
$WARMWRK	$WARMWRK	HASP warm start processor PCE work area DSECT
$WORK	$WORK	HASP $GETWORK/$RETWORK general work area DSECT
$WSTAB	$WSTAB	HASP work selection tables DSECT
$XECB	$XECB	HASP extended ECB element DSECT
$XEQWORK	$XEQWORK	HASP execution processor PCE work area DSECT
$XFMWORK	$XFMWORK	HASP XFR I/O manager processor PCE work area DSECT
$XIT	$XIT	HASP exit information table DSECT
$XRT	$XRT	HASP exit routine table DSECT

Environment

- Main task, subtask, user address space or functional subsystem (HASPFSSM).
- MVS WAIT and $WAIT are not applicable.

$MSG - Write to Operator Message Area

Use $MSG to generate a message text area to be referenced by the message operand of the $WTO macro instruction.

Format Description

```
[symbol]    $MSG        id-value
                        [,'message-text'[,SYMB=symbol-name]]
                        [,JOB={YES}]
                              {NO }
```

id

Specifies the numeric 3-digit message identification that is to be displayed with the message text when the MSGID parameter on the CONDEF initialization statement is set to YES.

message

Specifies the character string enclosed within single quotes that is to be displayed as the informational portion of the message. If the purpose of this macro instruction is to generate only the message identification, this operand should be omitted.

SYMB=

Specifies the symbol-name that is to be assigned to the message text portion of the area generated by the macro instruction. If this operand is specified, the message operand must be specified. This symbol may be used to modify variable portions of the message text before executing the corresponding $WTO macro instruction. It must not be referred to directly by the $WTO macro instruction; the symbol assigned to the beginning of the area must be used for this purpose.

JOB=

Specifies whether or not the user, at $WTO macro execution time, has placed the 18-byte job identification information into the beginning of the text portion using the symbol as specified by the SYMB operand. The format of the job identification is as follows:

Byte	Content
0-7	Job identification (JOBnnnn, STCnnnn, or TSUnnnn)
8	Blank
9-16	Job name
17	Blank

Specifications for the JOB operand are as follows:

YES
The user places job information into the message text portion of the area prior to executing a $WTO macro instruction.

NO (default)
The user does not place job information into the message area but can require the $WTO macro instruction to extract job information from the job control table and append the information to the console message buffer copy of the message during $WTO macro execution time.

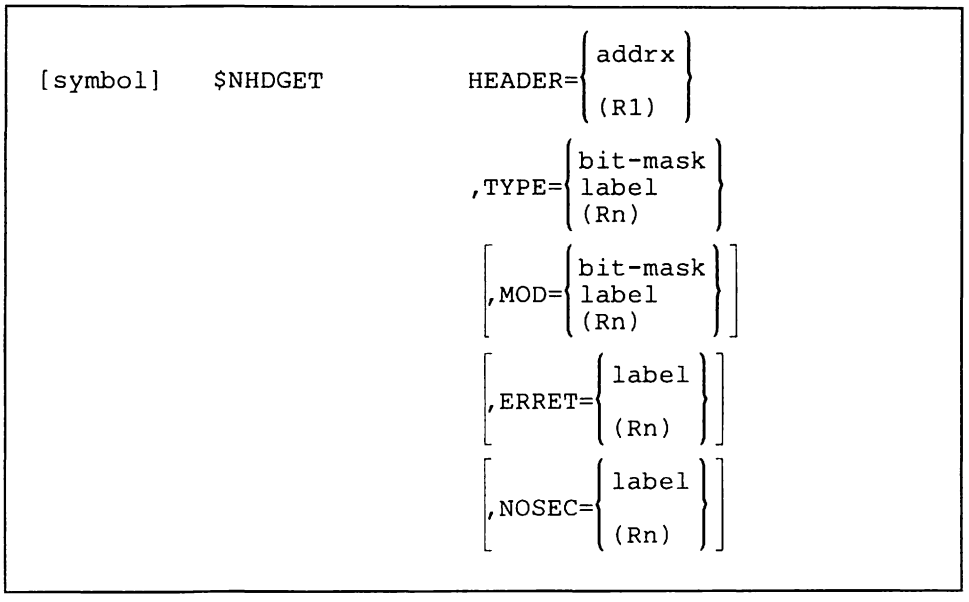
Environment

- Main task.
- $WAIT cannot occur.

$NHDGET - Get the Network Header Section

Use $NHDGET to search the network job header, job trailer, or data set header to locate a specified section of a control block. If the control block section is located, the address is returned in register 1. This address can then be used to access the control block section if you need to modify that header or trailer information.

Format Description



HEADER=
Specifies the address of the control block within the section indicated by the TYPE= keyword is to be located.

TYPE=
Specifies an 8-bit mask which indicates the type of section to be located. Valid types and their corresponding masks are defined in the $NHD macro (the job header DSECT). TYPE= can be specified as either a register whose low-order byte contains the address of the type mask, or TYPE= can be specified as a 1-byte label.

MOD=
Specifies an 8-bit mask which indicates the value of the modifier field of the control block section to be located. The valid bit mask settings are also located in the $NHD macro.

ERRET=
Specifies a label or register of an error routine which receives control if the specified header type is invalid.

NOSEC=
Specifies a label or register of an error routine which receives control if the specified header type is not found.

Notes:

1. *Register 1 contains the address of the control block section if it is located.*

2. *Registers 0, 1, and 15 are used by this macro; do not use them.*

Environment

- Main task
- $WAIT cannot occur

$PATCHSP - Generate Patch Space

Use $PATCHSP to cause a specified number of bytes of patch space to be generated. This patch space is divided into halfwords and listed in the assembly in such a way that both the assembly location (for REP and AMASPZAP patch statements) and the base displacement (in the form BDDD) are printed for each halfword.

Format Description

```
[symbol]   $PATCHSP   length-number
                      [,DSECT={YES}]
                              {NO }
```

length
Specifies the length of the patch space in bytes.

DSECT=
Specifies how the specified patch space will be generated.

YES
Indicates that the specified patch space is generated with all binary zeroes.

NO
Indicates that the specified patch space is generated with halfwords of S-type ADCONS (that is, S(*)).

Caution: Local addressability is required for this macro instruction to assemble correctly.

Environment

- Main task, subtask, user address space, or functional subsystem (HASPFSSM).
- MVS WAIT and $WAIT are not applicable.

$PBLOCK - Block Letter Services

Use $PBLOCK to create block letters from the job name in the job control table (JCT), and from the job type and job number in the processor control element (PCE).

In a user exit routine associated with the separator page JES2 exit point in the HASPPRPU module, use $PBLOCK to create block letters from 1 to 8 characters as specified in the DATA= parameter.

Format Description

```
[symbol]   $PBLOCK     DATA={addrx }
                         {(R0)  }
                      ,BUFFER={addrx }
                              {(R1)  }
                     [,SLANT={YES}]
                             {NO }
                     [,CENTER={YES}]
                              {NO }
```

DATA=
Specifies a register or name of a field containing the address, in register format, of the data to be used to create the block letters. The length of the data must be no greater than 8 characters. If you specify less than 8 characters, this field must include trailing blanks; this macro always ends its scan for characters after getting 8 characters.

BUFFER=
Specifies a register or name of a field containing the address of a HASP output buffer. This buffer area is the I/o data area; therefore, if $GETBUF is used to obtain this area, be certain to add FIX=YES to that macro statement.

SLANT=
Specifies whether or not the block letters should be slanted as follows:

YES
The job name and number are to be slanted.

NO (default)
Everything is to remain unslanted.

CENTER=
Specifies how the separator page block letters are placed on the page as follows:

YES (default)
The block letters are to be centered. However, if the field containing the character string is filled with trailing zeroes rather than blanks, the block letters will not be centered.

NO
The block letters are to be left-justified.

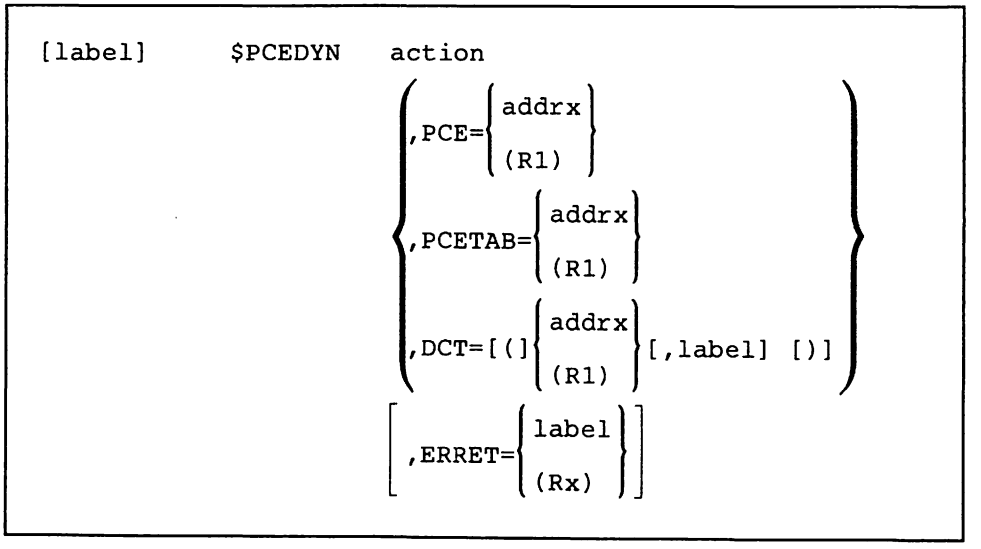
Environment

- Main task.
- $WAIT can occur.

$PCEDYN - Attach or Delete a JES2 PCE

This macro provides the interface to those services that perform all generating (ATTACHing) and deletion (DETACHing) of JES2 processors (PCEs).

Format Description



action

Specifies the type of action requested.

ATTACH
Generate a new PCE(s). These PCEs are dispatched based on the DISPTCH keyword on the associated $PCETAB entries.

DETACH
Dequeue and delete a PCE(s).

DETACHTEST
Determine if a PCE(s) can be detached at this time.

PCE=

Specifies the address of a PCE. This address can be either a register (1-12) or the name of a field containing the PCE address. R1 is loaded with the value.

If Action is:	This keyword indicates:
ATTACH	Specifies an existing PCE of a PCE type (PCEID value) for which another PCE (1) will be generated.
DETACH	Specifies a PCE to detach.
DETACHTEST	Specifies to test a PCE and determine if this PCE can be deleted (that is, determine if any resources are still outstanding).

PCETAB=

Specifies the address of a PCE table entry for a PCE type. This address can be either a register (1-12) or the name of a field containing the entry address.

If Action is:	This keyword indicates:
ATTACH	Specifies a table entry for a PCE type that is not one-to-one with a DCT type. A PCE of this type is to be generated.
DETACH	PCETAB= cannot be specified for DETACH. $PCEDYN cannot detach an arbitrary PCE of a PCE type.
DETACHTEST	PCETAB= cannot be specified for DETACH. $PCEDYN cannot detach an arbitrary PCE of a PCE type.

DCT=

Specifies the address of a device control table (DCT). This address can be either a register (1-12) or the name of a field containing the DCT address. R1 is loaded with the value.

Additionally, to indicate a DCT chain field that should be used to attach or detach PCEs for the DCT chain starting with the DCT specified, use a second positional parameter. When a DCT chain is to be processed, $PCEDYN will attach or detach PCEs, connect or disconnect DCTs and their 'managing' PCEs, or do nothing as indicated by the associated DCT and PCE table entries (for example, a DCT not owned or managed by a specific processor).

Notes:

1. *If a DCT chain is specified, no PCE is attached for DCTs for which DCTPCE is already nonzero.*

2. *If any PCE ATTACH fails for a DCT in the chain, the entire DCT chain is processed as if DETACH had been requested.*

ERRET=

Defines an error routine location (label or register) to branch to if R15 is not zero on return from ATTACH.

Caution: Before using this macro, it may be useful to review the table structures involved, i.e. the PCE table and the DCT table in HASPTABS (macros $PCETAB and $DCTTAB). Also, the mapping macros for the PCE ($PCE) and DCT ($DCT) provide further understanding of these control blocks.

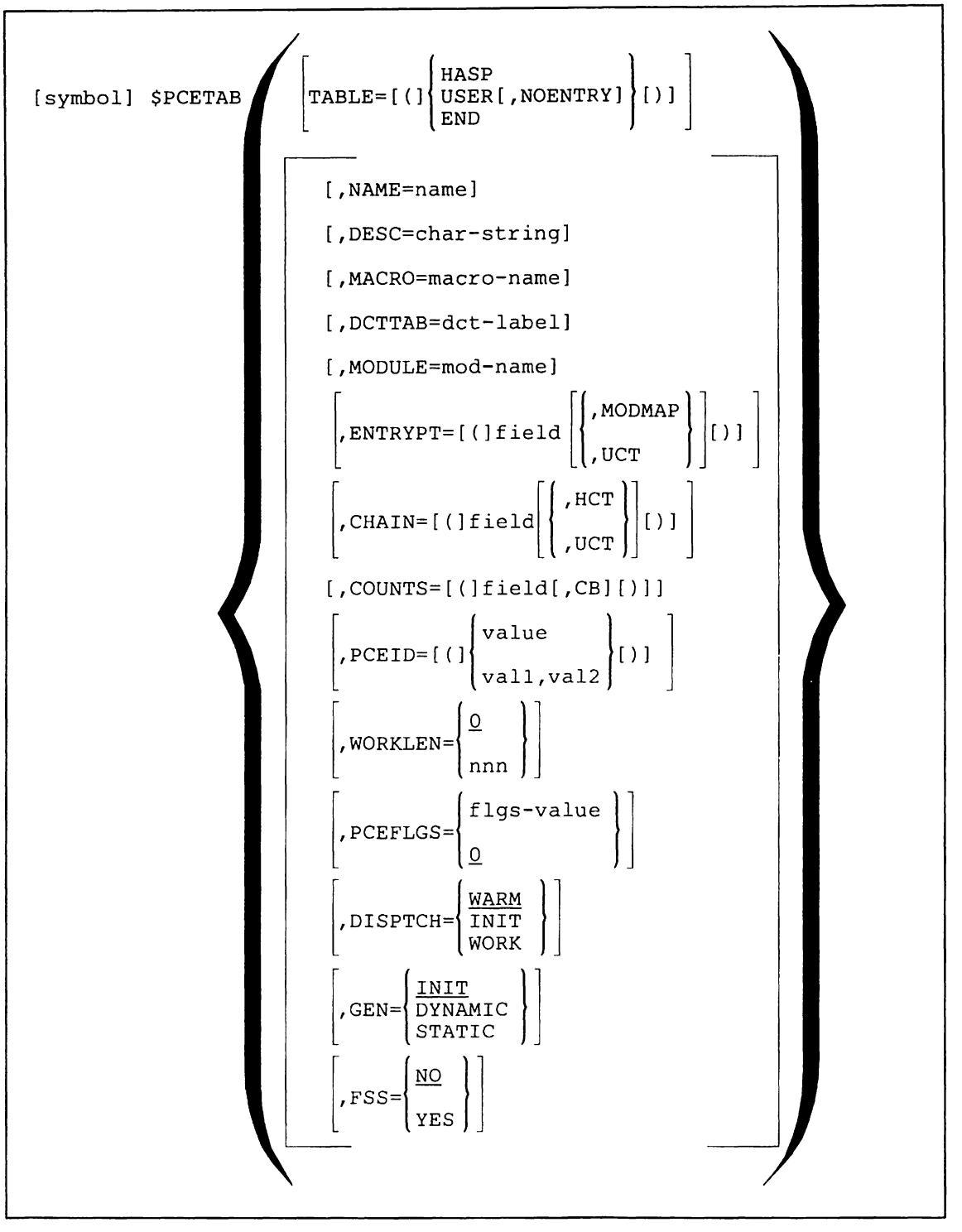
Environment

- JES2 Main task.
- $WAITs cannot occur.

$PCETAB - Generate or Map PCE Table Entries

Use $PCETAB to map and generate PCE table entries.

Format Description



TABLE=
Specifies the first entry in the HASP or user PCE table. If TABLE= is specified, all other operands are ignored. TABLE=END specifies the end of a PCE table; this must be coded at the end of all PCE tables.

HASP
Specifies the first entry in a HASP table.

USER
Specifies the first entry in a user PCE table.

(USER,NOENTRY)
Specifies that the ENTRY statement normally generated for the user PCE table is suppressed.

END
Specifies the end of the defined USER or HASP table.

NAME=
Specifies a 1- to 8-character name for the PCE type.

DESC=
Specifies a 1- to 24-character description of the PCE type. The word processor is appended to the end of this description.

MACRO=
Specifies a 1- to 8-character macro name for the macro that maps the PCE work area for this PCE type. This keyword is for documentation only.

DCTTAB=
Specifies the label provided on the DCT table entry that corresponds to the DCTs that have a one to one correspondence with the PCE type (if any).

MODULE=
Specifies a 1- to 8-character module name that contains all or most of the processor's code. This keyword is for documentation only.

ENTRYPT=
Specifies the name of a fullword field containing the processor entry point address.

field
Specifies a MODMAP field if this is a HASP table or a UCT field if this is a user table.

(field,MODMAP)
Explicitly specifies a MODMAP field.

(field,UCT)
Explicitly specifies a UCT field.

CHAIN=

Specifies the name of a fullword field that can be used to point to the first PCE of this type within the entire PCE chain that is anchored in $PCEORG in the PCE. The $PCEDYN service maintains this field.

field
Specifies an HCT field.

(field,HCT)
Explicitly specifies an HCT field.

(field,UCT)
Explicitly specifies a UCT field.

COUNTS=

Specifies the name of a fullword field that contains two halfword counts for this PCE type. The first count is the count of defined PCEs, and must be set during JES2 initialization (prior to invoking Exit 24). The second count is the count of allocated PCEs; this count is maintained by the $PCEDYN service.

field
Specifies the name of the field used by this keyword.

field,HCT
Explicitly specifies an HCT field.

field,UCT
Explicitly specifies a UCT field.

PCEID=

Specifies the values for the PCEID field. The second byte of the PCEID in user table entries should start at 255 and decrease. The two-byte PCEID must uniquely define a PCE type.

value
Specifies the PCEID as the 2-byte value defined by a DC of AL1(0,value).

(val1,val2)
Specifies the PCEID as the 2-byte value defined by a DC of AL1(val1,val2).

WORKLEN=

Specifies the length of the PCE work area for this PCE type. This value defaults to 0, and is typically specified using an equate in a PCE work area mapping macro.

PCEFLGS=

Specifies the flags to place in the PCEFLAGS field during initialization. The flag values are defined with the PCEFLAGS field in the PCE mapping macro. The default is 0.

DISPTCH=
Specifies the initial dispatching status for PCEs of this type once they are created.

WARM
Specifies that PCEs are dispatched immediately if initialization and warm-start processing have completed, otherwise, the PCEs are $WAITed on HOLD. At the end of warm-start processing, all PCEs are POSTed for HOLD.

INIT
Specifies that PCEs are made ready immediately then dispatched at the completion of initialization processing (at the same time that warm start processing begins).

WORK
Specifies that PCEs are $WAITed on work until $POSTed by later processing.

GEN=
Specifies when this specific PCE type is to be generated by the $PCEDYN macro.

INIT
Specifies that PCEs are to be generated during JES2 initialization processing (that is, after most initialization, but prior to calling Exit 24).

DYNAMIC
Specifies that PCEs are to be generated by using the $PCEDYN service after initialization.

STATIC
Specifies that this type PCE should not be generated.

Note: This specification is only used for the HASP initialization PCE.

FSS=
Specifies whether this PCE type is permitted (YES) or not (NO) to run in functional subsystem (FSS) mode. If FSS=YES is specified, then the larger of the JES2-mode PCE work area or the FSS-mode work area is used. The default is NO.

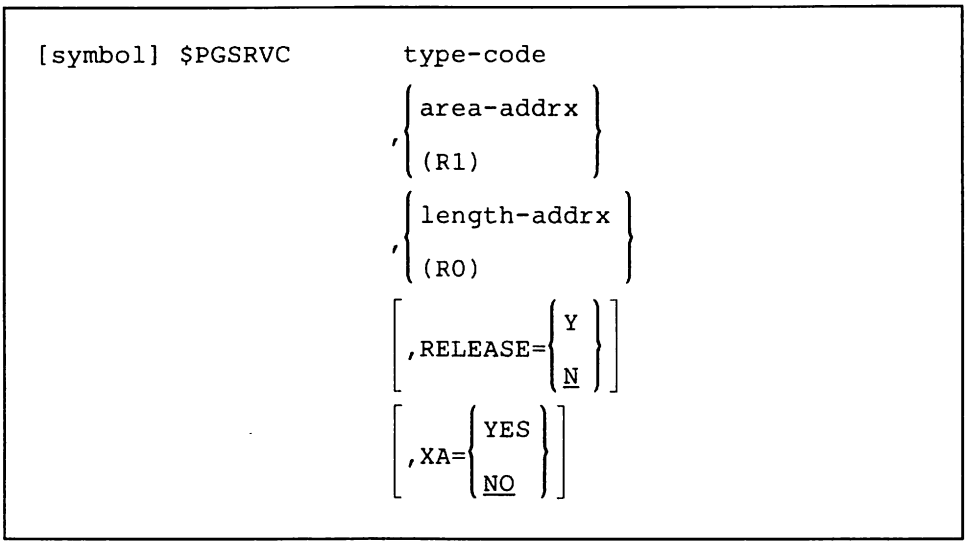
Environment

- Main task or during JES2 initialization or termination.
- $WAIT is not applicable -- this macro generates a DSECT or a static table entry; it does not generate executable code.

$PGSRVC - Perform a Virtual Page Service

Use $PGSRVC to page-fix, page-free, or page-release an area of JES2 storage through a branch entry to the MVS virtual storage manager.

Format Description



type-code
Specifies the type of function to be performed:

FIX
Page-fix a JES2 storage area (uses $PGFIX service routine)

FREE
Page-free a previously fixed JES2 storage area (uses $PGFREE service routine)

RLSE
Page-release a JES2 storage area (uses $PGRLSE service routine)

area
Specifies the starting address of the area of storage. If an address is specified, it must be the address of a word in storage containing the address of the area. If register notation is used, the area address must have been loaded into the designated register before the execution of this macro instruction.

length
Specifies the length, in bytes, of the storage area. If an address is used, it specifies a word in storage containing the area length. If register notation is used, the area length must have been loaded into the designated register before execution of this macro instruction.

RELEASE=
Specifies whether the storage is released or not during a fix or free operation. Use this parameter to prevent unnecessary page-ins and page-outs.

Y
Indicates that the storage is released before fixing or freeing. Only specify YES if you have no need for the current storage contents.

N
Indicates that the storage is not released before fixing or freeing.

XA=
Specifies whether it is permissible (YES) or not (NO) for the storage requested by this macro instruction to reside above the 16-megabyte line, that is, in 31-bit addressing mode.

Note: This keyword only has meaning for a JES2 Version 1 system; if it is specified for a JES2 Version 2 system, it is ignored.

Notes:

1. *Paging is done synchronously; that is, on return from $PGSRVC the paging action is complete, and no other JES2 processor receives control during this processing.*

2. *For page-free requests, the page is made pageable only when the number of page-free requests specifying the page equals the number of page-fix requests for the page. If the designated area is not page-fixed, the area is unaffected by the execution of this macro instruction.*

3. *For page-release operations, if the area does not encompass one or more complete pages, the area is unaffected by the execution of this macro instruction.*

4. *Register 15 is used to pass control to the specified service routine.*

5. *Refer to Supervisor for further information concerning virtual page services.*

Environment

- Main task.
- $WAIT cannot occur.

$POST - Post a JES2 Event Complete

Use $POST to indicate that one or more JES2 resources should be posted by turning off specified inhibitors in the $HASPECF field of the HASP communications table (HCT). Use $POST also to post a specific PCE that a JES2 event has occurred; if all inhibitors are reset by the action, the PCE is requeued to the JES2 dispatcher's $READY queue. Inhibitors turned off in the $HASPECF field cause requeuing of all PCEs on the resource wait queues by the dispatcher.

Format Description

```
[symbol] $POST      {pce-addrx }
                    {(R1)      }
                    {$HASPECF  }

                    ,event/resource

                    [,TYPE={SET  }]
                           {RESET}
                           {TEST }

                    [,MASPOST={YES}]
                              {NO }
```

pce

Specifies the specific processor control element (PCE) that is to be posted or specifies that the $HASPECF field within the HASP communication table (HCT) is to be posted. If register 1 is used, register 1 must refer to a PCE and must be loaded with the address of the PCE prior to executing the macro instruction. This is a positional operand and must be specified first.

event/resource

Specifies one or more events/resources that are to be posted. This operand must be consistent with the allowable events acceptable for the first operand as follows.

- If PCE was specified, the following JES2 events can be be specified:

  Event

  null

  The specified PCE is made ready for dispatching if it has no wait flags on (a $WAIT with INHIBIT=NO is issued).

  FORCE

  The specified PCE is made ready for dispatching regardless of its wait flags.

HOLD
An operator has entered a $S command.

IO
An input/output operation has completed (logically).

OPER
An operator has activated a processor. (There are no posts ($POST) with OPER in the distributed system.)

POST
An MVS POST of an ECB has been performed.

WORK
Work is available for the specified processor.

- If $HASPECF was specified, the following JES2 resource can be specified:

Resource

ABIT
Waiting for the next dispatcher cycle

ALOC
A dynamic allocation has completed

BUF
A JES2 buffer has been released

CKPTP
A checkpoint cycle has completed

CKPT
A JES2 checkpoint write has completed

CKPTW
A JES2 checkpoint should be written

CMB
A console message buffer has been released

CNVT
A converter has been released

FSS
A functional subsystem has completed FSS-level processing

HOPE
An output processor has been released

IMAGE
A UCS or FCB image has been loaded

JOE
A JOE has been released

JOT
A JES2 job output element has changed status

JOB
A JES2 job queue element has changed status

LOCK
A lock has been released

MAIN
Storage is available

PSO
A process SYSOUT request has been queued for the JES2 PSO processor(s)

PURGE
A JES2 job queue element (JQE) has been placed on the purge queue

PURGS
Purge resources from $PURGER have been released

RSV
A JES2 RESERVE has been satisfied

SMF
AN SMF buffer has been released

TRACK
A track group from the JES2 spooling data set has been released

UNIT
A device control table has been released

TYPE=
Specifies the type of action for a $POST of a resource. This keyword is ignored for PCE $POSTs of events, FORCE, or null.

SET
Indicates that the resource should be POSTed (that is, the resource flag(s) set on)

RESET
Indicates that the resource should be unPOSTed (that is, the resource flag(s) set off)

TEST
Indicates that the resource should be tested to determined if it was $POSTed.

MASPOST=
Specifies whether (YES) or not (NO) the resource $POST is to be propagated to all members of the multi-access spool complex. This keyword is only valid for resources within the JES2 main task.

Environment

- Main task.
- $WAIT cannot occur.

$POSTQ - Quick Post Facility

Use the $POSTQ macro to quick post an ECB (event control block). This macro produces the necessary inline code to either quick post (that is, bypass the POST routine) an ECB and/or issue an MVS POST if the specified ECB is currently waiting on an event.

Format Description

```
[symbol] $POSTQ  ECB={label }
                     {(RO)  }
                 [,CODE={0     }]
                        {absexp}
                 [,SVC={YES}]
                       {NO }
```

ECB=

Specifies the address of the ECB to be quick posted. If the ECB is currently waiting and you also specify SVC=YES on this macro, JES2 then requests that an MVS POST of the ECB be issued. If this keyword is not specified, an assembly error will occur.

CODE=

Specifies a 30-bit post code to be quick posted into the ECB. The default is 0.

SVC=

Specifies whether (YES) or not (NO) an MVS POST should be issued if the ECB is currently waiting (that is, the ECB wait bit is on). If SVC=NO is specified, no MVS POST is issued and a condition code is returned to the caller to signify whether the quick post was successful (CC=0) or unsuccessful (CC=1). The default is YES.

Environment

- JES2 main task, subtask, user address space, and HASPFSSM address space.
- $WAIT cannot occur

$PRPUT - Create Separator Pages

Use $PRPUT to create user-defined separator page(s). The created separator page can replace or add to the standard separator page. The separator page JES2 Exit 1 is in module HASPPRPU and Exit 23 is in module HASPFSSP.

Format Description

```
[symbol]   $PRPUT      DATA={addrx }
                            {(R1) }
                      ,LEN={addrx }
                           {(R0) }
                      [,COUNT={absval}]
                              {1     }
                      [,WAIT={YES}]
                             {NO }
                      [,CC=m]
                      [,OPTCD=J]
```

DATA=
Specifies, a register or name of a field containing the address of the data to be printed or punched. The address of this data must not be a 31-bit address. If you do specify a 31-bit address, unpredictable results may occur. The data pointed to by this register must be a fixed-data field because this data area is the I/O data area. Therefore, if $GETBUF is used to obtain this area, be certain to add FIX=YES to that macro statement.

LEN=
Specifies a register or name of the field containing the length of the fixed-data field, including any carriage control and 3800 table reference characters (TRC) if present.

COUNT=
Specifies the number of times the data is to be produced. Default is no repetitions.

WAIT=
Specifies whether or not to wait until I/O has completed as follows:

YES
Wait for I/O completion.

NO (default)
Do not wait for I/O to complete.

CC=m
Specifies that machine carriage control is desired. If this parameter is omitted, no carriage control is assumed.

OPTCD=J
Specifies that the 3800 table reference character (TRC) is present in the data. If this parameter is omitted, 3800 TRC is not assumed to be present.

Environment

- Main task.
- $WAIT can occur if WAIT=YES is specified.

$PURGE - Return Direct-Access Space

Use $PURGE to return the direct-access space that was allocated for a given job or data set.

Format Description

```
[symbol]    $PURGE        IOT=addrx
```

IOT=
Specifies the address of the primary allocation IOT from which track group allocation elements (TGAEs) are to be returned. Secondary allocation IOTs, if any, are processed by the $PURGER routine in $PURGER in HASPTRAK. This address is passed in register 1.

Environment

- Main task.
- $WAIT can occur.

$QADD - Add Job Queue Element to the JES2 Job Queue

Use $QADD to add an element to the JES2 job queue, placing it in the specified logical queue. The address of the job queue element in which the element image has been placed is returned in register 1 if the element is successfully added.

Format Description

```
[symbol]    $QADD     {element-addrx}
                      {(R1)        }
                     ,{queue-value }
                      {(R0)        }
                      [,full relexp]
```

element
Specifies the address of an element image which is to be added to the JES2 job queue. If register notation is used, the address must be loaded into the designated register before the execution of this macro instruction.

queue
Specifies the logical queue in which the job queue element is to be placed. This value must always be one of the eight logical queue types. If register notation is used, one of these values must be loaded into the designated register before the execution of this macro instruction.

The queue type specifications may be ignored if the job queue element has been flagged for cancellation. The resulting logical queue is as follows:

- JQE1OCAN bit on and JQE1PURG bit off. Any $QADD with a $XEQ or $XMIT specification is altered to $OUTPUT.
- JQE1OCAN bit off and JQE1PURG bit on. Any $QADD with JQEJOECT and JQEHLDCT fields 0 is altered to $PURGE. Any $QADD with a JQEJOECT or JQEHLDCT field nonzero is altered to $HARDCPY.

full
Specifies a location to which control is returned if the JES2 job queue is full. If this operand is omitted, the condition code is set to reflect the status of the JES2 job queue as follows:

CC=0
The queue is full and the element cannot be accepted.

CC≠0
The element was successfully added to the queue.

Environment

- Main task.
- $WAIT can occur.

$QCTGEN - Define a Quick Cell Control Table

Use $QCTGEN to define the attributes of a quick cell type in a quick cell control table (QCT). Note that **all** the keywords on this macro are required.

Format Description

```
[symbol]  $QCTGEN   NAME=cellname
                   ,SIZE=nnn
                   ,NOFFS=nnn
                   ,COFFS=nnn
                   ,INIT=nnn
                   ,EXT=nnn
                   ,SP=nnn
                   ,LIMIT=nnn
```

NAME=
Specifies (in EBCDIC) the name of the quick cell control table.

SIZE=
Specifies the size (in bytes) of an individual quick cell.

NOFFS=
Specifies the offset of the NAME field in the quick cell.

COFFS=
Specifies the offset of the CHAIN field in the quick cell.

INIT=
Specifies the number (0-32767) of quick cells created in the initial quick cell pool.

EXT=
Specifies the number (0-32767) of quick cells in a quick cell pool extension.

SP=
Specifies the storage subpool number in which the quick cell pool resides.

LIMIT=
Specifies the maximum number (0-32767) of quick cells to GET/FREE at any one time.

Environment

- Functional subsystem (HASPFSSM)
- MVS WAIT and $WAIT not applicable.

$QGET - Obtain Job Queue Element from the JES2 Job Queue

Use $QGET to obtain a job queue element from the specified logical queue of the JES2 job queue and return the address of this element in register 1.

Format Description

```
[symbol] $QGET          queue-type
                       [,TYPE=class-list-prefix]
                       [,FOUND=relexp]
                       [,NONE=relexp]
                       [,CB={addrx }]
                            {(R0)  }
                       [,MF={EX}]
                            {L }
                       [,NODETBL={addrx }]
                                 {(R0)  }
```

queue type
Specifies the logical queue from which the job queue element is to be obtained. This queue type is indicated in the inline parameter list generated by the macro expansion. Valid queue types and their meanings are:

$DUMMY
Reserved queue.

$FREE
Indicates that the JQE is to be obtained from the JES2 free queue.

$HARDCPY
Indicates that the JQE is to be obtained from the JES2 hardcopy queue.

$INPUT
Indicates that the JQE is to be obtained from the JES2 input queue.

$INWS
Indicates a QGET call for initiators.

$OJTWS
Indicates that the JQE work selection algorithms are used to select an eligible job for this transmitter.

$OJTWSC
Indicates that the JQE work selection algorithms are used to select an eligible job for this transmitter and that the conversion queue ($XEQ) is scanned for work.

$OUTPUT
Indicates that the JQE is to be obtained from the JES2 output queue.

$PURGE
Indicates that the JQE is to be obtained from the JES2 purge queue.

$RECEIVE
Indicates that the JQE is to be obtained from the JES2 SYSOUT receive queue.

$SETUP
Indicates that the JQE is to be obtained from the JES2 setup queue.

$XEQ
Indicates that the JQE is to be obtained from the execution queue.

$XEQCLAS
Indicates that the JQE is to be obtained from the JES2 execution class queue.

$XMIT
Indicates that the JQE is to be obtained from the JES2 transmit queue.

Note: Although $INWS, $OJTWS, and $OJTWSC are not actual queue types, they can be used to indicate work selection for offload job transmitters or a call for initiators.

TYPE=
Specifies the prefix of the class list used if the queue type is $INWS, $OJTWS, or $OJTWSC. The value specified must be a valid control block DSECT name, for example, DCT or PIT. This value is also used to determine the offset of the class list field. If TYPE= is not specified, the class list defaults to DCTCLASS for offload job transmitters and PITCLASS for initiators.

FOUND=
Specifies a label or address in a register to which JES2 branches if a selectable JQE is found.

NONE=
Specifies a label or address in a register to which JES2 branches if no selectable JQE is found.

CB=
Specifies the address of a control block which is to be used for work selection or an initiator call. This keyword is only valid if either the queue type is specified as $INWS, $OJTWS or $OJTWSC.

MF=

Specifies the required form of this macro.

L

Indicates the list form of the macro.

EX

Indicates the executable form of the macro.

NODETBL=

Specifies the location of the MDCTNODS field in the job transmitter's line device control table (DCT). If the queue type specification is not $XMIT, this keyword should not be used. If register notation is used, the address must have been loaded into the designated register before execution of this macro instruction.

Environment

- Main task.
- $WAIT can occur.

$QJIX - Add a Job Queue Element (JQE) to the Job Index Table (JIX)

Use $QJIX to get a job number for a specified JQE and to add that JQE to the job index table (JIX).

Format Description

```
[symbol]   $QJIX        JQE={addrx }
                            {(R1) }
```

JQE=
Specifies the address of the JQE to be added to the JIX. If you use register notation, this address must be loaded into the specified register prior to the execution of this macro instruction.

Environment

- Main task.
- $WAIT can occur.

$QLOC - Locate Job Queue Element for Specific Job

Use $QLOC to locate the job queue element associated with the job specified by a job number and return the address of this element in register 1.

Format Description

```
[symbol] $QLOC      { jobno-adval }
                    { (R1)        }
                    [,none-relexp]
```

jobno
Specifies the binary job number associated with the job for which the job queue element is being searched. If an address is used, it specifies the address of a halfword that contains the binary job number. This halfword must be located on a halfword boundary. If register notation is used, the binary job number must be loaded into the designated register before the execution of this macro instruction.

none
Specifies a location to which control is returned if the specified job number is not locatable in the JES2 job queue.

If this operand is omitted, the condition code is set to reflect the status of register 1 as follows:

CC=0
The specified job is not locatable.

CC≠0
The specified job is locatable, and R1 contains the address of the associated job queue element.

Environment

- Main task.
- $WAIT can occur.

$QMOD - Modify Job Queue Element in the JES2 Job Queue

Use $QMOD to remove a modified job queue element from the JES2 job queue and place it back on the queue in the specified logical queue in accordance with the priority of the job queue element.

Format Description

```
[symbol] $QMOD    {element-addrx}
                  {(R1)        }
                  {queue-value }
                  {(R0)        }
                   [,ALONE]
```

element
Specifies the address of an element that has been modified and is to be requeued in the JES2 job queue. If register 1 is used, the address must be loaded into register 1 before execution of this macro instruction.

queue
Specifies the logical queue in which the job queue element is to be placed. This value must always be one of the eight logical queue types. If register 0 is used, one of these values must be loaded into register 0 before the execution of this macro instruction.

The queue type operands may be ignored if the job queue element has been flagged for cancellation. The resulting logical queue is as follows:

- JQE1OCAN bit on and JQE1PURG bit off. Any $QMOD with a $XEQ specification is altered to $OUTPUT.
- JQE1OCAN bit off and JQE1PURG bit on. Any $QMOD with a JQEJOECT and JQEHLDCT field 0 is altered to $PURGE. Any $QMOD with a JQEJOECT or JQEHLDCT field nonzero is altered to $HARDCPY.

Caution: If the processor issuing the $QMOD does not have exclusive ownership of the JQE via $QSUSE, the results of the $QMOD macro instruction are unpredictable. One way to guarantee exclusive ownership is to obtain the JQE via a $QGET or $QADD macro instruction or with the $GETLOK macro instruction.

ALONE
Indicates that the busy flags associated with the moved element are to remain unchanged. If ALONE is not specified, the busy flags associated with the moved element are turned off.

Once the queues have been obtained, all modifications must be made to the JES2 job queues before a $WAIT macro can be issued. Issuing a $WAIT macro implies that the processor no longer requires the queues.

Environment

- Main task.
- $WAIT can occur.

$QPUT - Return Job Queue Element to the JES2 Job Queue

Use $QPUT to return a job queue element to the JES2 job queue, placing it in the specified logical queue.

Format Description

```
[symbol]   $QPUT     { element-addrx }
                     { (R1)          }
                     { queue-value   }
                     { (R0)          }
```

element
Specifies the address of an element which is to be returned to the JES2 job queue. If register 1 is used, the address must be loaded into register 1 before the execution of this macro instruction.

queue
Specifies the logical queue in which the job queue element is to be placed. This value must always be one of the eight logical queue types. If register 0 is used, one of these values must be loaded into register 0 before the execution of this macro instruction. If $XEQ is specified and the execution node is not equal to the local node, the queue type is altered to $XMIT.

The queue type specifications may be ignored if the job queue element has been flagged for cancellation. The resulting logical queue is as follows:

- JQE10CAN bit on and JQE1PURG bit off. Any $QPUT with a $XEQ or $XMIT specification is altered to $OUTPUT.
- JQE10CAN bit off and JQE1PURG bit on. Any $QPUT with a JQEJOECT and JQEHLDCT fields 0 is altered to $PURGE. Any $QPUT with a JQEJOECT or JQEHLDCT field nonzero is altered to $HARDCPY.

Caution: The specified job queue element must be previously obtained with a $QGET or $QADD macro instruction or the action of the $QPUT macro instruction is unpredictable.

Environment

- Main task.
- $WAIT can occur.

$QREM - Remove Job Queue Element from the JES2 Job Queue

Use $QREM to remove a specified job queue element from the JES2 job queue.

Format Description

```
[symbol]   $QREM      { element-addrx }
                      { (R1)          }
```

element
Specifies the address of an element that is to be removed from the JES2 job queue. If register notation is used, the address must be loaded into the designated register before the execution of this macro instruction.

Caution: The specified job queue element must have been previously obtained with a $QGET or $QADD macro instruction or the action of the $QREM macro instruction is unpredictable.

Environment

- Main task.
- $WAIT can occur.

$QSUSE - Synchronize to Use Shared Queues

Any JES2 processor that begins an access to any information in the checkpoint records (which are shared if the system is a multi-access spool environment) must execute the $QSUSE macro instruction prior to such access in order to update checkpoint records.

The contents of the checkpoint records include: shared queue control elements (QSEs), checkpointed HCT variables (beginning at $SAVEBEG, including job queue headers), remote message spooling queues, remote sign-on table, master track group map, job queue, and job output table (JOT).

Format Description

```
[symbol]    $QSUSE        [ TYPE={ WAIT } ]
                            { TEST }
```

TYPE=

Execution of the macro tests the $QSONDA bit and $CKPTACT bit in the $STATUS field of the HCT. Updating any checkpointed information is permitted only if both bits are 0.

If TYPE=WAIT is coded or defaulted, the calling processor waits ($WAIT CKPT) access to update the checkpoint records is permitted. The checkpoint processor is activated, if necessary, and it later posts ($POST) all other processors forced to wait ($WAIT CKPT).

If TYPE=TEST is coded, an immediate return to the caller occurs, with a zero condition code if updating is permitted.

The permission to update is granted by execution of this macro expires as soon as the processor executes any $WAIT macro instruction, actual or imbedded in another macro.

Environment

- Main task.
- $WAIT can occur if TYPE=WAIT is specified.

$QUESMFB - Queue a JES2 SMF Buffer on the Busy Queue

Use $QUESMFB to place a JES2 SMF buffer address on the busy queue ($SMFBUSY) and MVS post (POST) the HASPACCT subtask.

Format Description

```
[symbol]    $QUESMFB    { buffer-addr }
                        { (R1)        }
```

buffer-addr
Specifies the address of the buffer to be queued. If register notation is used, the buffer address must be loaded into the designated register prior to the execution of this macro instruction.

Caution: The JES2 SMF services should not be used if the BUFNUM parameter on the SMFDEF initialization statement is set to less than 2.

Environment

- Main task.
- $WAIT cannot occur.

$RELEASE - Release the Checkpoint Reserve

Use $RELEASE to release a checkpoint via MVS RELEASE services.

Format Description

```
[symbol]  $RELEASE
```

The shared checkpoint value is released.

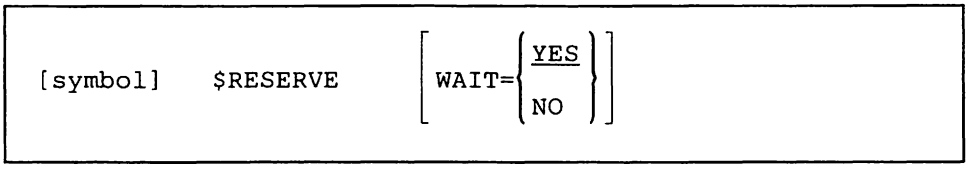
Environment

- Main task.
- $WAIT can occur.

$RESERVE - Request Checkpoint Reserve

Use $RESERVE to reserve a checkpoint via MVS RESERVE services.

Format Description



WAIT=
Specifies the action to be taken in the event that the checkpoint was not RESERVED.

YES
Control is not to be returned to the caller until the checkpoint is reserved.

NO
Control is to be returned immediately with a return code of 4 in register 15.

Environment

- Main task.
- $WAIT can occur.

$RESTORE - Restore Registers from the Current Processor Save Area

Use $RESTORE to restore one or more registers from the current processor's current save area (that is, from the save area built by the most recently issued $SAVE macro instruction).

Format Description

```
[symbol]  $RESTORE  list
```

list

Specifies a list of one or more registers, and/or groups of registers to be restored. If more than one register is being restored, the entire list must be enclosed in parentheses.

A register group is indicated by a pair of registers enclosed in parentheses. All registers, beginning with the first register specified and ending with the second register, are restored. The order of restoring a group of registers is: R14, R15, R0-R12. If the list consists of a single group, the outer (list) parentheses are not required.

Note: All registers must be specified symbolically. The accepted register symbols are: R0, R1, R2, . . ., R15.

Examples:

```
Restore register 2
  $RESTORE (R2) or
  $RESTORE R2
Restore registers 15 through 8
  $RESTORE ((R15,R8)) or
  $RESTORE (R15,R8)
Restore register 3 and register 10
  $RESTORE ((R3),(R10)) or
  $RESTORE ((R3),R10) or
  $RESTORE (R3,(R10))
Restore registers 0, 3 through 5, and 8
  $RESTORE (R8,R0,(R3,R5))
```

Note: The sublist order is unimportant.

Environment

- Main task.
- $WAIT cannot occur.

$RETBLK - Return a Storage Cell to a Free Cell Pool

Use $RETBLK to return a number of predefined storage cells to one of several previously established free cell pools.

Format Description

```
[symbol]    $RETBLK    [ TYPE={SAVE} ]
                       [      {JIB  }
                       [      {BUF  }
                       [      {     }
                       [      {RPL  }
                       [      {GTRC }
                       [      {SSOB } ]
                       [ ,ADDR={addrx} ]
                       [       {(R1)} ]
```

TYPE=

Specifies the type of storage cell to be returned to the free cell pool. Specifications are as follows:

Specification	Meaning
SAVE	An MVS-type save area
JIB	A JOE information block
BUF	A 4K I/O buffer
RPL	A request parameter list control block chain
GTRC	A GETREC chain control block
SSOB	A subsystem options block

ADDR=

Specifies the address of the first cell to be returned. If register 1 is used, it must contain the address of the first cell to be returned prior to executing this macro.

Environment

- Functional subsystem (HASPFSSM)
- MVS WAIT cannot occur

$RETSAVE - Return a JES2 Save Area

Use $RETSAVE to return the current JES2 save area to the JES2 free pool of save areas.

Format Description

```
[symbol]     $RETSAVE
```

Note: Only registers 14, 15, 0, 1, 2 are saved across this macro call. (The RETSAVA routine performs the register saving.)

Environment

- Main task.
- $WAIT cannot occur.

$RETURN - Restore Registers, Free the JES2 Save Area, and Return to the Caller

Use $RETURN to restore the caller's registers saved in the current processor save area and return that save area to the save area pool.

Note: If this macro is coded within a functional subsystem environment, $RETURN generates inline code to place the save area pointed to by register 13 onto the unused save area stack. All necessary code for standard linkage conventions is also generated.

Format Description

```
[symbol]    $RETURN    [RC={ n      }]
                       [   (R15)    ]
                       [,TRACE={YES }]
                       [       NO   ]
                       [,FSACB={address}]
                       [       (Rn)    ]
                       [,PARM={YES }]
                       [       NO  ]
```

RC=

Specifies a numeric return code to be returned in register 15. If this operand is not specified, the return code is set to 0.

If register 15 is used, the return code value must be loaded into register 15 prior to the execution of this macro instruction.

Note: The condition code is preserved across the execution of this macro instruction.

TRACE =YES

Specifies that $RETURN is traced via the JES2 event trace facility. The PCE address and the returned registers, 14, 15, 0, and 1, are traced (TRACE ID=2), and an 8-character symbol designating where the $RETURN macro was issued is also traced.

TRACE =NO (default)

Indicates that the $RETURN macro is not traced. This is the default when the TRACE parameter is not specified.

FSACB=

Used to determine whether processor tracing is on. The use of this keyword is only valid in the functional subsystem environment. If an FSACB specification is not provided, only global $RETURN tracing is done.

PARM=

Specifies whether (YES) or not (NO) $RETURN processing should skip over an inline parameter list when control is returned to the calling module from the current save area. Return is through register 14.

Note: The first byte of the parameter list must contain the length of the parameter list. (The length must also include this first byte.)

Notes:

1. *You must have the address of the HCT in register 11 prior to executing this macro.*

2. *If this macro is issued when a PRE ($ESTAE) currently exists for the save area level about to be returned, the $ESTAE is cancelled.*

3. *The use of the $RETURN macro assumes that register 11 contains either the address of the HASP communication table (HCT) for the JES2 main task environment or the HASP function communication table (HFCT) for a functional subsystem module.*

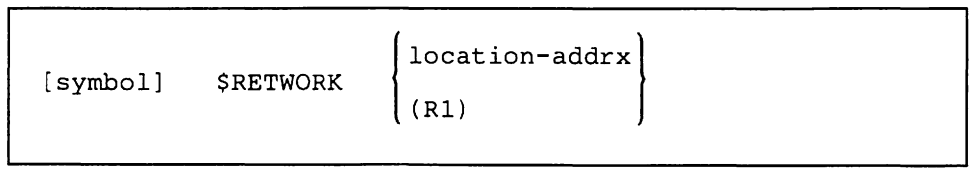
Environment

- Main task or functional subsystem (HASPFSSM).
- $WAIT cannot occur.

$RETWORK - Return a Work Area

Use $RETWORK to return a work area obtained with the $GETWORK macro instruction.

Format Description



location

Specifies the address of the work area to be returned. (This address is loaded into register 1.)

The work area to be returned must have been obtained via a previous $GETWORK macro instruction.

Environment

- Main task.
- $WAIT cannot occur.

$SAVE - Obtain JES2 Save Area and Save Registers

The $SAVE macro instruction obtains a register save area from a JES2-managed save area pool and saves registers 14, 15, and 0 through 12 in the save area. No registers are destroyed by executing this macro. If this macro is coded within a functional subsystem environment, $SAVE will save the registers in the save area pointed to by register 13. All necessary code for standard linkage conventions is also generated. For information on linkage conventions, see Chapter 2, "Linkage Conventions."

Format Description

```
[symbol]    $SAVE       [TRACE={YES}]
                        {NO }
                        [,NAME=symbol]
                        [,REGS={YES}]
                               {NO }
                        [,FSACB={address}]
                                {(Rn)  }
```

TRACE=

Specifies whether or not the $SAVE macro is traced as follows:

YES

Specifies the PCE address; the contents of registers 14, 15, 0 and 1; and an 8-character symbol designating where the $SAVE macro instruction was issued and traced.

NO (default)

Indicates that the $SAVE macro instruction is not traced. This is the default when the TRACE parameter is not specified.

NAME =symbol

Specifies the name associated with this $SAVE macro for tracing and diagnostic purposes. If NAME is not specified, the label (symbol) on the $SAVE macro is used for identification. If neither NAME or symbol is specified, the current CSECT name is used.

REGS=

Specifies whether a starting store multiple instruction (STM) and ending load multiple instruction (LM) are used. This specification is only valid in the functional subsystem environment.

FSACB=

Used to determine whether processor tracing is on. The use of this keyword is only valid in the functional subsystem environment. If an FSACB specification is not provided, only global $RETURN tracing is done.

Note: The use of the $SAVE macro assumes register 11 contains either the address of the HASP communication table (HCT) for the JES2 main task environment or the HASP function communication table (HFCT) for a functional subsystem environment.

Caution: The TRACE=YES parameter is provided so that normal JES2 operations are traced via the JES2 trace facility. Most JES2 central services specify TRACE=YES on their respective $SAVE macros. An individual routines issuing the $SAVE macro should not specify TRACE=YES if the routine is called an excessive number of times, unless the routine is considered an integral part of the normal JES2 logic flow. If a routine is traced on an infrequent basis, a trace ID can be assigned to that function so it can be traced independently.

Also, you must have stored in register 11 the address of the HCT (or the HFCT if running in an FSS environment) prior to executing this macro.

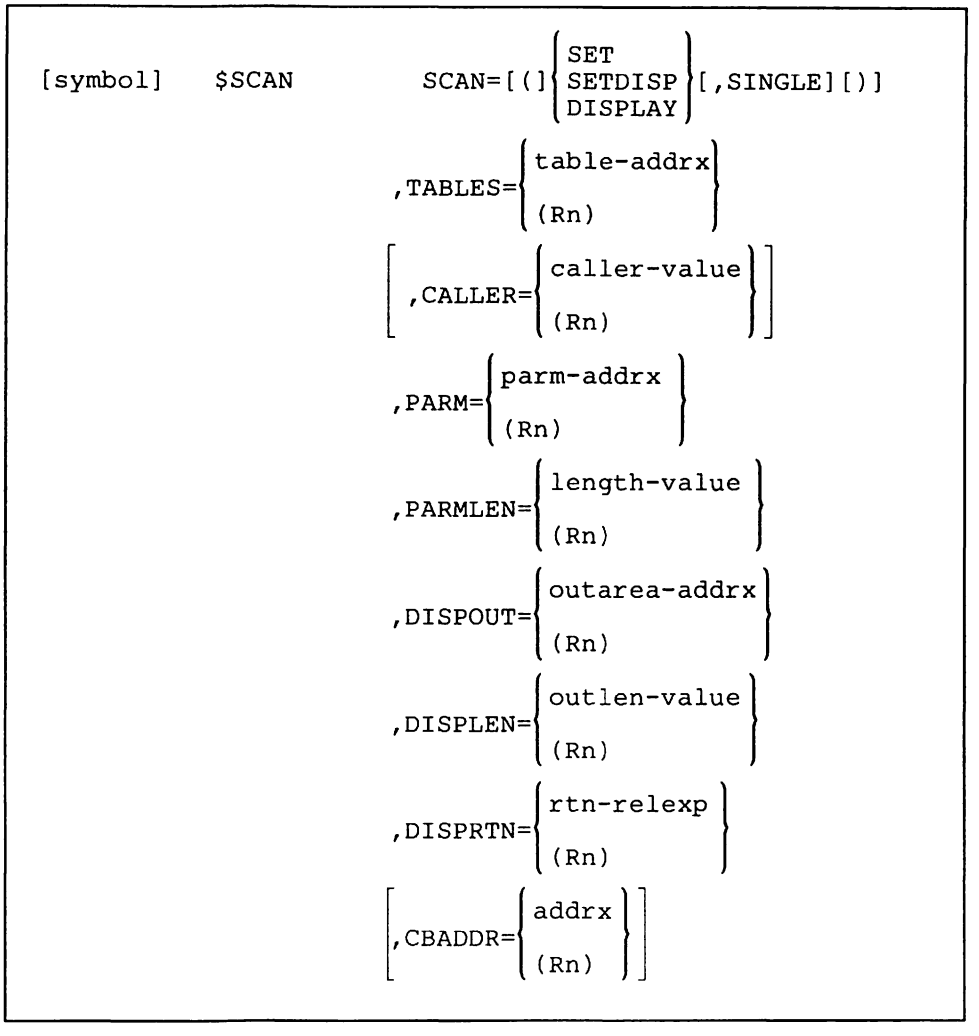
Environment

- Main task or Functional subsystem (HASPFSSM)
- $WAIT cannot occur.

$SCAN - Scan Initialization Parameters

Use $SCAN to scan parameter statements, placing specified values in associated control blocks or displaying values associated with specified keywords.

Format Description



SCAN=

Specifies either a scan of a parameter statement specifying values to be scanned and placed in control blocks, or a scan of a parameter statement requesting display of the values associated with specified keywords.

SET

indicates to scan a parameter statement specifying values to be scanned and placed in control blocks. The DISPOUT=, DISPLEN=, and DISPRTN= keywords are not required if SCAN=SET is also specified.

SETDISP
Indicates that a SET operation is to be completed immediately followed by a DISPLAY operation. The display shows the results of the SET. This is all performed within a single call to $SCAN.

DISPLAY
indicates to scan a parameter statement specifying keywords, the associated values for which should be displayed.

SINGLE
indicates to limit the scan to a single parameter keyword (and any subscanning required for that keyword).

TABLES=
Specifies the address of a two-word area containing the addresses of two scan tables (generated via the $SCANTAB macro). These scan tables are used in scanning parameter statements. One or the other, but not both, of these table addresses may be zero. If register notation is used, the register must contain the address of the two-word area prior to executing $SCAN. **This operand is required.**

CALLER=
Specifies a caller id for use during the scan. The id can be a value from 1 to 255. JES2 ids are assigned using 1 and ascending values, and ids should be assigned for installation uses of $SCAN, if required, using 255 and descending values. When CALLER is specified, only the scan table ($SCANTAB) entries having this caller id specified for their CALLER= keyword and those entries not specifying CALLER= are the entries that are used in the scan. If this operand is not specified, then all entries in the $SCANTAB tables are used when processing the request.

PARM=
Specifies the address of the parameter statement that is to be scanned. If register notation is used, the register must contain the address of the parameter prior to executing $SCAN. **This operand is required.**

PARMLEN=
Specifies the length (that is, the length plus 1) of the parameter statement specified for the PARM= operand. If register notation is used, the register must contain the parameter length value prior to executing the macro. **This operand is required.**

DISPOUT=
Specifies the address of the output area where the display lines are placed for the routine specified by DISPRTN=. **This operand is required.**

DISPLEN=
Specifies the length of the output area specified for the DISPOUT= operand. **This operand is required.**

DISPRTN=
Specifies the address of the routine to be invoked to display the scan results. The routine is passed the address of the current scan work area, which contains all pertinent information, in register 1. This routine receives control to issue diagnostic error messages. **This operand is required.**

CBADDR=
Specifies the oldest parent control block. If CBADDR= is specified, $SCAN does not search for a control block for this level of scanning. However, $SCAN does perform whatever scanning or indexing that is requested by $SCANTAB using this specified control block.

Return Codes

0
Indicates the $SCAN request executed successfully.

4
Indicates the scan found an obsolete keyword (as indicated by a $SCANTAB entry specifying OBS=YES).

8
Indicates the scan found an keyword not supported in the tables, or not supported for this caller id.

12
Indicates the scan encountered scanning errors (invalid syntax, etc.) that could not be resolved.

Environment

- Main task and during JES2 initialization.
- $WAIT cannot occur during JES2 initialization but can occur during command processing.

$SCANB - Backup Storage for a Scan

Use $SCANB to backup a copy of a storage area before it is possibly changed during execution of the $SCAN facility. $SCANB may be used only within a pre-scan or post-scan exit routine specified via the PRESCAN and PSTSCAN operands of the $SCANTAB macro.

The $SCAN facility uses $SCANB to backup all control block fields before they are changed. If, at any time during the scan, an error is found, $SCAN uses the backups created by $SCANB to restore all the changed fields to their contents prior to the start of the scan. If a $SCAN pre-scan or post-scan exit routine changes a storage area, it should first backup that area using the $SCANB macro.

Format Description

```
[symbol]    $SCANB      SCWA={addrx }
                         {(R1)  }
                       ,ADDR={addrx }
                             {(R0)  }
                     ,LENGTH={label }
                             {(Rn)  }
                    [ ,TYPE={BACKUP}]
                             {DISPLAY}
                             {ERROR  }
```

SCWA =

Specifies the address of the current scan work area, mapped by the $SCANWA macro.

ADDR =

Specifies the address of the storage area to backup before the scan possibly changes it.

LENGTH =

Specifies the length (in bytes) of the storage area indicated by the ADDR operand. If register notation is used, only registers 2 through 12 are valid.

TYPE=

Specifies that an area of storage is to be used following a SET and DISPLAY $SCAN request or if an error occurs within a $SCAN call.

DISPLAY

Indicates to save an area of storage to use to display the results of a SCAN=SET request. The value that is set is passed to $SCAN and used as input for a SCAN=DISPLAY request, for example:

```
$SCANB     SCAN=DISPLAY,ADDR=addrx,LENGTH=
```

This value must, therefore, also be scannable by $SCAN.

ERROR

Indicates to save an area of storage to use if an error is encountered during a $SCAN call. $SCAN then returns this keyword value to point to the location of the error.

BACKUP

Indicates to produce a backup copy of the storage area before it is possibly changed during the execution of $SCAN.

Environment

- Not applicable.

$SCANCOM - Call the $SCAN Facility Comment Service Routine

Use the $SCANCOM macro to search for and locate the first non-blank, non-comment character in a specified text string. This facility allows the $SCAN facility to ignore (skip over) comment text provided in both initialization statements and commands. A return code is passed in register 15. JES2 returns the address of the nonblank, non-comment character in register 1.

Format Description

```
[symbol]   $SCANCOM   TEXTBEG={label}
                              {(R1) }
                      ,TEXTEND={label}
                               {(R0) }
```

TEXTBEG=
Specifies the address of the beginning of the text that is to be scanned by the $SCAN facility.

TEXTEND=
Specifies the address of the end of the text that is to be scanned by the $SCAN facility.

Return Code	Meaning
0	Non-blank and no comments found
4	Valid comment, non-blank found
8	End of statement encountered, no non-blanks, or non-comment characters found
12	No asterisk-slash (*/), the comment ending delimiter, found and an invalid comment encountered

Environment

- JES2 main task
- $WAIT cannot occur

$SCAND - Call the $SCAN Facility Display Service Routine

Use the $SCAND macro instruction to call the display service exit routines called by $SCAN to add text to a display line being created for the SCAN=DISPLAY request. This macro instruction can only be called from a $SCAN exit routine or $SCAN itself.

Format Description

```
[symbol]   $SCAND   SCWA={addrx}
                        {(R1) }
                    ,TEXT={'text'}
                          {(R0)  }
                    ,LENGTH={label}
                            {(Rn)  }
                   [,BRKOPT={YES}]
                            {NO }
                   [,DEBLANK={YES}]
                             {NO }
```

SCWA=
Specifies the address of the current scan work area. This can either be provided as the actual address, a label, or a register (R1-R12).

TEXT=
Specifies the text (specifies in single quotes) to be added to the display line or the address of that text as specified in a register (R2-R12).

LENGTH=
Specifies the length of the text to be added. This can either be a label or a register (R2-R12). If the actual text is provided on the TEXT= keyword, the length specification on this keyword defaults to the length of that text.

BRKOPT
Specifies that the text specified by the TEXT= keyword will be separated (YES) or not (NO) from the text already passed.

DEBLANK
Specifies whether (YES) or not (NO) the blanks and X'00's are to be removed from the front and end of the text.

Environment

- JES2 main task
- $WAIT cannot occur

$SCANTAB - Scan Table

Use $SCANTAB to create scan tables to be used in conjunction with the $SCAN facility, defining the allowed input and syntax for initialization parameter statements and some operator commands. JES2 uses $SCANTAB to define the initialization parameter statements, initialization options, and selected operator commands.

$SCANTAB entries are used to define the start of a user table ($SCANTAB TABLE=USER...) or JES2 table ($SCANTAB TABLE=HASP...) and the end of a table ($SCANTAB TABLE=END). Each entry defines:

- a keyword allowed in the statement input
- how to find the correct control block and field(s) related to the keyword
- what the allowed input can be
- how to convert the input for storing into the field(s) or convert the contents of the fields for display Because $SCANTAB generates only tables, not executable code, register notation may not be used for any of the operands.

By default, the $SCANTAB macro does not expand the table entry in the assembly listing. If you require this information use either of the following methods:

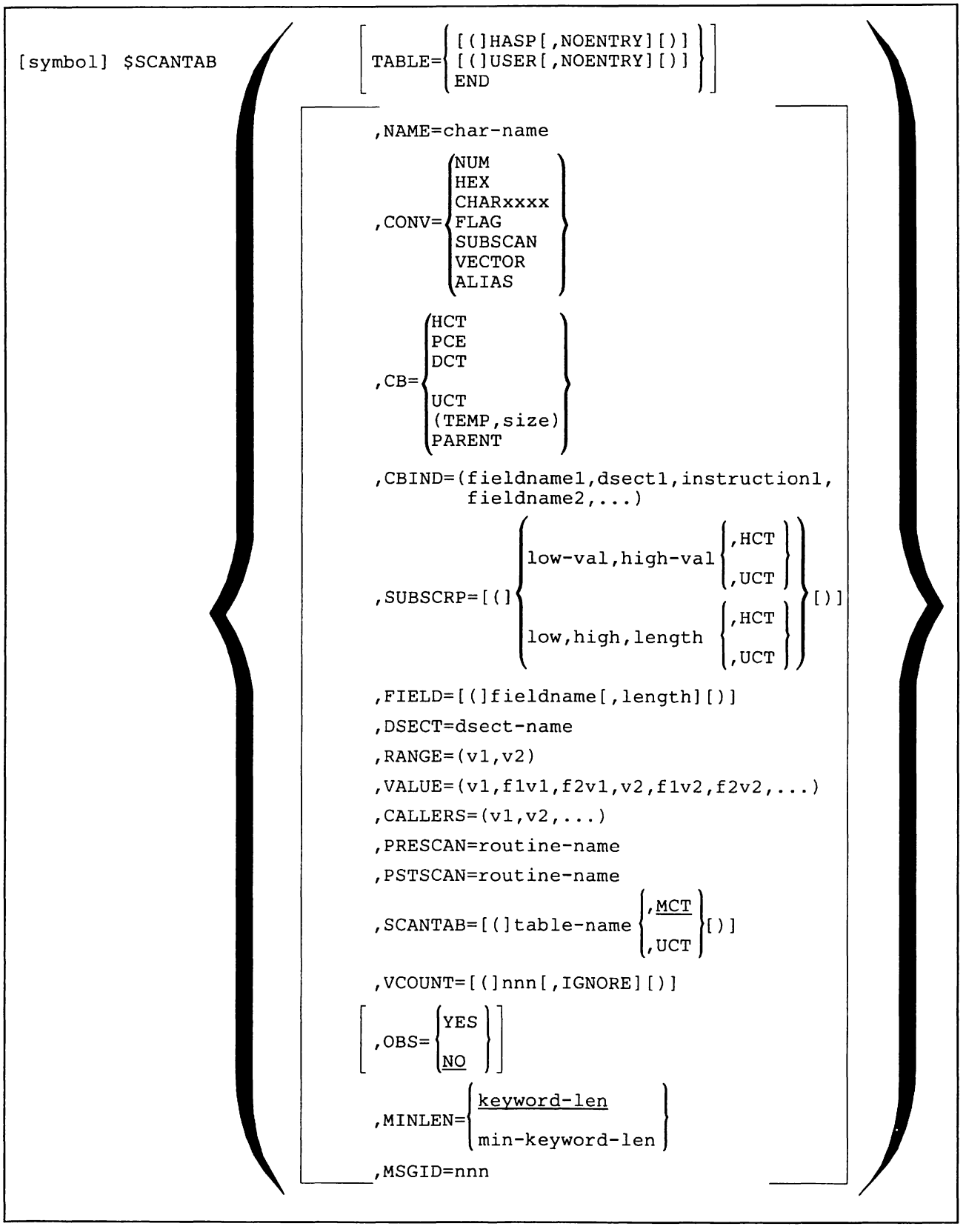
- Assemble your module with:

```
$MODULE SYSP=(PRINT,GEN,DATA,NOGEN,NOGEN)
```

- Set the SYSPARM keyword on the EXEC statement as:

```
EXEC IEV90,PARM='SYSPARM=(PRINT,,,,)'
```

Format Description



TABLE=
Specifies the start or end of a scan table.

Specify TABLE=HASP or TABLE=USER to start the corresponding table, and optionally a second parameter of NOENTRY (e.g. TABLE=(USER,NOENTRY)) to indicate no ENTRY statement need be generated for the label of the scan table.

Specify TABLE=END to terminate a scan table.

Other operands are ignored if TABLE is specified, and a label is required on the $SCANTAB macro if a table is being started. If TABLE= and NAME= are both not specified, only the mapping of an $SCANTAB entry is generated by the macro.

NAME=
Specifies a 1- to 8-character name for the scan table entry that indicates the scan keyword being defined (for example, PRINTER, CONSOLE, or JOENUM parameter on the OUTDEF statement). If TABLE= and NAME= are both not specified, only the mapping of an $SCANTAB entry is generated by the macro.

CONV=
Specifies the conversion to be done (and defines the valid input) when the keyword defined by NAME= is encountered during a scan. CONV= is required if $SCANTAB is used to generate a table entry. The following specifications are valid:

NUM
indicates the keyword represents a numeric value that is stored in binary. An optional second and third positional (e.g. CONV=(NUM,8,100)) define a multiple to round the input value to before storing and a value by which the value of the keyword is multiplied before storing.

HEX
indicates the keyword represents a hexadecimal value that is stored in binary. The optional second and third positional operands are as for CONV=NUM.

CHARxxxx
indicates the keyword represents a character value and defines the allowed character input. If the first positional for CONV= is CHAR, any characters not required for syntax within $SCAN (i.e., all except commas, parentheses, dashes, and equal signs) are valid for the value unless a specific list of characters is provided as described below. If the first positional is CHARxxxx, for a 1- to 4-character string xxxx, the input must fall within the character sets defined by the xxxx or be one of the specific list of characters that may be optionally provided as described below. Within the xxxx specification 'A' indicates the alphabetic character set A-Z, 'N' indicates numerics 0-9, and 'S' indicates the special nationals $ @ and #. Additionally, an 'F' indicates the first character of input must be alphabetic or a 'J'

indicates the first character of input must be alphabetic or special national (JCL rules), even though the remaining input may have less strict input rules.

A specific list of allowed characters may be specified as the second, third, and etc. positional operands for CONV. They may be specified as single characters, or a 2-character hex values.

FLAG
indicates the keyword represents a flag value that is stored as the setting of one or more bits within a single flag byte. The allowed values and associated bit settings are defined by VALUE=.

ALIAS
indicates the keyword is the alias of another keyword and SCANTAB= specifies the label of the scan table entry mapping that other keyword.

VECTOR
specifies the keyword represents a vector of values. Another 'level' of scan is used to process the vector of values which is specified within parentheses. Therefore each value is defined, positionally, by another scan table pointed to by SCANTAB=.

SUBSCAN
specifies the keyword requires another level of scanning to scan suboperands. Unlike the CONV=VECTOR subscanning, a completely recursive level of scanning is done, allowing suboperands of any of the CONV types specified above, not just a vector of values. SCANTAB= specifies the address of a doubleword containing the addresses of two scan tables to pass to the recursive $SCAN call.

CB=
Specifies one of the 'primitive' control blocks known by $SCAN as the control block containing the fields representing the value of the keyword, or as the starting point for a control block search for that field. If a control block is not found for a keyword during a scan, and a PRESCAN routine for the keyword does not then supply the control block address, $SCAN issues a $ERROR. CB= is required if CONV= is not specified as SUBSCAN, VECTOR, or ALIAS unless PRESCAN= is specified.

HCT
indicates the JES2 HCT control block.

PCE
indicates the current processor control element (PCE) at the time of the scan.

DCT
indicates a JES2 device control table (DCT), found by scanning all the DCTs for one whose DCTDEVN field corresponds to the NAME= specified and the device number found during the scan.

UCT
indicates the installation-defined user control table (UCT), which is pointed to by the $UCT field of the HCT.

TEMP
indicates a temporary control block should be allocated with a length defined by the second positional operand.

size
Indicates the length (in bytes) of the control block

PARENT
indicates the control block determined at the scan level 'above' this scan level should be used, i.e., when a control block is found for a CONV=SUBSCAN or CONV=VECTOR keyword, that control block is the parent control block for the resulting subscanning.

CBIND=
Specifies how to find the control block required for this keyword, if the primitive control block is not it. The search starts from the primitive control block address, and performs a series of operations of fields within each control block along the way. The fields used are defined by the first and second operands and the operation is defined by the third operand in each of a set of operand triplets defined to CBIND=. CBIND must be specified as a list consisting of a multiple of these three operands. That is, operand 1 defines a field name, operand 2 defines a dsect name for that field, and operand 3 defines the operation to be performed, with the current control block address, against the field.

The allowed operations are L (load), LA (load address), A (add), AH (add halfword), AL (add logical), S (subtract), SH (subtract halfword), and SL (subtract logical). If the operation specified is preceded by an asterisk (e.g. *LA), then any subscript indexing for the control block search is done before this CBIND operation, rather than after all CBIND operations. Subscript indexing is defined by the SUBSCRP= operand.

SUBSCRP=
Specifies an allowable subscript range for the input specifying this keyword. If SUBSCRP is specified, the allowable input forms for $SCAN are 'keyword(subscript)' and 'keywordsubscript'. SUBSCRP is specified as a list of 2, 3 or 4 values, with the first being the lowest allowed subscript value and the second being the highest allowed value. The first and second operands must be numeric values with one exception; single character alphanumeric subscripts can be used, with 'A' corresponding to value X'C1', '4' to value X'F4', etc.

The optional third value specifies an index value optionally used during the search for the control block for this keyword. After (by default) or during the CBIND processing in that search, the subscript value is used to index into the current control block to find the correct sub-block for the keyword. The lowest subscript is assumed to correspond to the 0th sub-block and the length of each sub-block is defined by the third positional value of SUBSCRP.

The optional fourth value specifies that the "high" and "low" values are to be used as offset values into the HASP (HCT) or USER (UCT) control tables.

FIELD=
Specifies the name and length of the field associated with the keyword value in the specified control block. The field must be within the DSECT specified by DSECT=, or must be an absolute offset if DSECT=0 is specified. The length is defined by the second positional parameter, and defaults to the assembler-defined length of the field label. FIELD= is required unless CONV= is SUBSCAN, VECTOR or ALIAS.

DSECT=
Specifies the DSECT name required to resolve the field specified by FIELD= in the control block found by the $SCAN search. If FIELD= is specified as an absolute offset into the control block, DSECT should be specified as DSECT=0.

RANGE=
Specifies the allowed range for the input. For keywords for which CONV is NUM, HEX, or CHARN RANGE specifies a binary range. For keywords for which CONV is CHARxxxx, for CHARxxxx not equal to CHARN, RANGE specifies a length range. RANGE and VALUE are mutually exclusive.

VALUE=
Specifies the allowed specific values a keyword may have. VALUE is used to limit input to only certain values, instead of using RANGE to limit the input to a range of values. RANGE and VALUE are mutually exclusive.

For keywords for which CONV= is not specified as FLAG, this keyword is specified as a list of allowed values. Note that the input must match the VALUE= specification exactly. For example, if the value 000293 is specified as input, it is within the allowed range for RANGE=(66,400), that is between 66 and 400, but it does not match the VALUE=(36,2,99,293,4) specification, exactly.

For CONV=FLAG keywords, VALUE is specified as a list making up a set of triplets of input, that is VALUE=(a1,b1,c1,a2,b2,c2,...). For each set of three operands, as shown, the first (a) is the allowed value the keyword may have, the second (b) is a flag byte setting to 'or' on in the FIELD if the keyword is given this value, and the third (c) is a flag bytes setting to 'and' off in the FIELD. For example, to implement a keyword with values of YES or NO, which is represented by a single flag bit setting specify the following:

```
CONV=FLAG,VALUE=(YES,YESFLAG,FF,NO,0,FF - YESFLAG)
```

If the keyword have no input value, i.e., there is value in the keyword being specified alone, VALUE should consist of one triplet with the first operand null.

CALLERS=

Specifies one or more caller ids (in a parenthesized list) for which this scan table entry is to be used. If CALLERS is not specified, the table entry is used for any $SCAN caller. This operand is useful, for example, when a scan table is to be used for multiple parameter statement purposes and not all keywords are valid in every case. Note that $SCAN supports multiple entries specified in a scan table for the same NAME= keyword with different CALLERS= specifications. Valid callers are:

Valid Callers	Identifies the:
$SCOPTS	JES2 initialization options (for example, COLD, WARM, REQ, NOREQ)
$SCIRPL	JES2 initialization statements
$SCIRPLC	console-issued commands during JES2 initialization
$SCDCMDS	display commands in HASPCOMM
SCSCMDS	set commands in HASPCOMM
SCDOCMD	short form(s) of the display commands

PRESCAN=

Specifies the name of a routine to be entered just after determining the parameter input contains this keyword and before scanning the input any further. If the routine name is undefined in the current assembly, a VCON is generated rather than an ACON. Register 1 points to the scan work area (SCWA) on entry to the routine and the routine can change the SCWA fields and use return codes to direct the actions of $SCAN. An optional second positional parameter of SET or DISPLAY on PRESCAN indicates that the PRESCAN routine should be called only for a SET or DISPLAY $SCAN call.

PSTSCAN

Specifies the name of a routine to be entered after all scanning (including possible subscanning) is done for this keyword. PSTSCAN= and the routine interface are as for PRESCAN=.

SCANTAB=

Specifies another scan table(s) or table entry required when scanning this keyword. For CONV=ALIAS, it specifies the address of another scan table entry defining the keyword of which this keyword is an alias. For CONV=VECTOR, it specifies the address of another complete scan table defining the values allowed for each element of the vector. For CON=SUBSCAN, it specifies the address of a doubleword containing the addresses of two complete scan tables, e.g. one user scan table and one JES2 scan table, to be used in the recursive $SCAN call that performs the subscanning required.

If CONV=VECTOR or CONV=SUBSCAN is specified, the pointer to the next set of scan tables is calculated as an offset into either the MCT or

UCT. This is specified by the second positional operand on this keyword. For example:

```
$SCANTAB      SCANTAB=(MCTPRTU,MCT)...
```

generates (MCTPRTU - MCT) for the offset into the $MCT of the scan table pair.

```
$SCANTAB      SCANTAB=(UCTPRTU,UCT)...
```

generates (UCTPRTU - UCT) for the offset into the $MCT of the scan table pair.

VCOUNT=

Specifies the number of vector elements this scan table entry defines. VCOUNT is ignored unless specified for an entry in a scan table specified to SCANTAB= for a CONV=VECTOR keyword. It allows a single scan entry to define multiple elements of a vector, with the associated fields for the elements being FIELD, FIELD plus the field length, FIELD plus twice the field length, etc. The default is 1.

An optional second positional parameter of IGNORE in VCOUNT indicates that null input for vector elements is allowed and the associated fields should not be changed in any way.

OBS=

Specifies whether the keyword specified for NAME= is to be considered obsolete (the default is OBS=NO). If OBS is specified as YES, $SCAN should consider it to be an error if this keyword is found during a scan, but return a less severe return code and message to the caller.

MINLEN=

Specifies the minimum character length of the keyword defined by this $SCANTAB entry that may be used to reference the keyword in parameter input. For example, if NAME=FORMS is specified, and MINLEN=2, then: FO, FOR, FORM, and FORMS are valid keyword references; F, FOX, and FORMSX are invalid. If MINLEN= is not specified, the valid keyword specification is the entire keyword; no abbreviated forms are allowed.

MSGID=

Specifies the 3-digit message ID for the $HASPnnn message identifier that is used when a SCAN=DISPLAY includes a display line in a $SCAN call. This message ID is ignored by $SCAN except at the highest level of scanning. For example, it is used for the PRINTERnn statement, but it is ignored for the FORMS= keyword on the PRINTERnn statement.

Environment

- JES2 main task or during initialization and termination.
- $WAITs is not applicable -- this macro generates a DSECT or static table entry; it does not generate executable code.

$SDUMP - Take a SDUMP of Storage

Use $SDUMP to dump the storage of selected address spaces.

Format Description

```
[symbol]    $SDUMP      [TITLE={'text'}]
                        [      {symbol}]
                        [      {(R1)  }]
                        [,ASIDLST={(Rn,Rm)  }]
                        [         {(label1) }]
                        [         {(label2) }]
                        [,ERROPT={RETURN}]
                        [        {WAIT  }]
                        [,HOME={YES}]
                        [      {NO }]
                        [,APPEND={YES}]
                        [        {NO }]
```

TITLE=
Specifies the title of the dump. You can specify the title of the dump as straight text within quotes or you can supply a symbol that identifies the beginning of the textual title or you can supply a register whose contents is the address of the textual title. If you supply a symbol or register, the symbol or register must point to a one byte length field followed by the text. If TITLE is not specified a default title for the dump is used.

ASIDLST=
Specifies a list of asids (up to two) associated with the address spaces to be dumped in addition to the home address space if HOME=YES. Label1 and label2 must define halfwords that contain the asids. Rn and Rm are two different registers that contain the asids in the right-most half of each register. The left-most half of each register must be zero.

HOME=
Specifies whether or not the home asid is dumped. HOME=YES is the default indicating that the home asid is to be dumped.

ERROPT=
Specifies the action to be taken should the dump fail.
ERROPT=RETURN indicates that when the dump fails, return to the caller should take place. ERROPT=WAIT indicates that a WTOR is to be issued to the operator and the $SDUMP processing is to wait for an appropriate reply. ERROPT=RETURN is the default.

APPEND=

Specifies whether or not the title supplied with TITLE= is to be appended to the default title. APPEND=NO is the default and indicates that the title supplied is not to be appended to the default title.

Environment

- Main task.
- MVS WAIT can occur (if ERROPT=WAIT).

$SEPPDIR - Create a User Peripheral Data Information Record (PDIR)

Use $SEPPDIR to send a PDIR to an output device immediately prior to sending a separator. The PDIR is a required control record that is sent to a SNA/RJE remote that is using its spooling capability to allow data set printing. The PDIR record is used to describe the data set (every output record, separator pages, and cards) being sent. If no separator is being sent, do not use this macro instruction. JES2 sends a PDIR preceding the print header and trailer separators. Also, JES2 sends a PDIR preceding a punch separator; no PDIR is sent following a punch file. This macro supports the separator exits (Exit 1 and Exit 15) in modules HASPPRPU. It is not used for the FSS separator exit (Exit 23) in module HASPFSSM.

Format Description

```
[symbol]    $SEPPDIR    { addrx }
                        { (R1)  }
```

The specified register contains the address of a JES2 buffer.

Environment

- Main task.
- $WAIT can occur.

$SETRP - Set Recovery Processing Options

Use $SETRP in a recovery routine to indicate how control will be received when the $RETRY routine is complete. Specifies if and how recovery is to take place.

Format Description

```
[symbol]   $SETRP   { [()]RECOVER,RESUME=resume-relexp[)] }
                    { TERMINATE                           }
                    { PERCOLATE                           }
```

RECOVER
Specifies that recovery is to take place. All functions are to resume as normal at the address specified by the RESUME= parameter.

RESUME=
Specifies where normal processing is to resume when error recovery is successful. This parameter is required when RECOVER is specified.

TERMINATE
Specifies that an abend is to take place and no recovery is to be attempted.

PERCOLATE
Specifies that this particular recovery attempt was unsuccessful but that termination is not to take place. Each of the higher level recovery routines is to be entered until either there are no more routines (in which case an abend occurs) or recovery is successful (in which case all functions resume as normal).

Note: $SETRP assumes addressability to the error recovery area (ERA) that is associated with the error that caused the recovery routine to be entered. Therefore, be certain to add the $ERA DSECT to the $MODULE macro for a routine for which you provide error recovery.

Environment

- Main task.
- $WAIT cannot occur.

$STCK - Call the $STCK Service Routine

Use the $STCK macro instruction to call the $STCK (store clock) service routine located in HASPNUC. This service routine obtains the current TOD clock and stores the current value at the location you specify.

Format Description

```
[symbol]    $STCK        ADDR={addrx }
                          {(R0)   }
```

ADDR=

Specifies the address at which the current TOD clock value is stored. This specification can be either an address or a register.

Environment

- JES2 main task
- $WAIT cannot occur

$STIMER - Set Interval Timer

Use $STIMER to set a time interval for the programmed interval timer.

Format Description

```
[symbol]    $STIMER    { loc-addrx }
                       { (R1)      }
```

loc

Specifies the address of a JES2 timer queue element (TQE). Before this macro instruction is executed, the TQE must be initialized. TQETIME must be initialized with the interval to be set in the following manner:

- If x seconds are desired, the TQETIME should be set to x.
- If y hundredth-seconds (0.01 seconds) are desired, then TQETIME should be set to the twos complement of y.

TQEPCE must be initialized with the address of the processor control element (PCE) to be posted.

If register notation is used, the address must be loaded into the designated register before the execution of this macro instruction.

Note: An unlimited number of independent $STIMER time intervals can be active at any time provided that each has been furnished with a unique JES2 timer queue element.

Environment

- Main task.
- $WAIT cannot occur.

$STMTLOG - Log an Initialization Statement

Use $STMTLOG to log initialization statements and related diagnostic information. This macro can be used by either JES2 or Exit 19 (Initialization Statement).

Format Description

```
[symbol]    $STMTLOG    [DIAG={addrx }]
                        [     {(R1)  }]
                        [     {label }]
                        [,TYPE={COMMENT}]
                        [      {WARNING}]
                        [      {ERROR  }]
                        [,MSGID={YES}]
                        [       {NO }]
                        [,LEN={field}]
                        [      {(R0) }]
```

DIAG=

Specifies the address of the diagnostic information associated with the last analyzed initialization statement or specifies the actual text of the diagnostic information when "label" is the keyword value. The message can be formatted to contain a message ID and/or message length as well as the actual text. The following table provides the required format information:

MESSAGE ID	LENGTH	DIAGNOSTIC	FORMAT*
NO	NO	Text	'DDDDDD...'
YES	NO	Text	'XXXDDD...'
NO	YES	Text	***Error***
YES	YES	Text	***Error***
NO	NO	Address	LDDDDDDD...
YES	NO	Address	XXXLDDDD...
NO	YES	Address	DDDDDDDD...
YES	YES	Address	XXXDDDDD...

* XXX = message ID
L = length
DDD = diagnostic

TYPE=

Specifies the type of diagnostic message that is to be logged for the last analyzed initialization statement.

COMMENT

Log the diagnostic information to hardcopy only.

WARNING

Log the diagnostic information to hardcopy if the source of the last analyzed initialization statement is not the console. If the source is a console also log the diagnostic information to the console.

ERROR

Log the current parameter statement to the console and hardcopy along with the diagnostic information.

MSGID=

Specifies whether (YES) or not (NO) a message ID ($HASPnnn) is included in the diagnostic text that is passed to the STMTLOG routine.

YES

Indicates that the message ID is supplied as part of the diagnostic text.

NO

Indicates that the message ID is not supplied as part of the diagnostic text.

LEN=

Specifies the address of the length (1-80 characters) of the diagnostic message. If this keyword is not specified, JES2 assumes that the length of the message is imbedded in the message.

Note: When no operands are specified, the last analyzed initialization statement is logged.

Environment

- Only Exit 19 during JES2 initialization.
- $WAIT cannot occur.

$STORE - Store Registers in the Current Processor Save Area

Use $STORE to store one or more registers in the current processor save area (that is, the most recently issued $SAVE macro instruction). The stored registers are returned to a calling routine upon execution of a $RESTORE macro instruction.

Format Description

```
[symbol]  $STORE  list
```

list

Specifies a list of one or more registers, and/or groups of registers to be stored. If more than one register is to be stored, the entire list must be enclosed in parentheses.

A register group is indicated by a pair of registers enclosed in parentheses. All registers, beginning with the first register specified and ending with the second register, are stored. The order of storing a group of registers is: R14, R15, R0-R12. If the list consists of a single group, the outer (list) parentheses are not required.

Note: All registers must be specified symbolically. The accepted register symbols are: R0, R1, R2,. . ., R15.

Examples:

```
Store register 2
  $STORE (R2) or
  $STORE R2
Store registers 15 through 8
  $STORE ((R15,R8)) or
  $STORE (R15,R8)
Store register 3 and register 10
  $STORE ((R3), (R10)) or
  $STORE ((R3),R10) or
  $STORE (R3,(R10))
Restore registers 0, 3 through 5, and 8
  $STORE (R8, R0,(R3, R5))
```

Note: The sublist order is unimportant.

Environment

- Main task.
- $WAIT cannot occur.

$TGMSET - Call the $TGMSET Service Routine

Use the $TGMSET macro instruction to generate the calling sequence to $TGMSET in HASPTRAK. $TGMSET tests and/or sets a bit in the specified track group map (TGM) and updates the DAS and $HCT field counts if COUNT=YES is specified. This macro can also be used to allocate and return track groups and mark track groups as "bad."

Format Description

```
[symbol]    $TGMSET    MTTR={mttr  }
                         {(R1)  }
                       ,ADDR={addrx }
                             {(R0)  }
                       [,SPOLMSK=addrx]
                       [,SET={ON  }]
                             {OFF }
                       [,QSUSE={YES }]
                               {NO  }
                       [,TYPE={SET    }]
                              {TEST   }
                              {TESTSET}
                       [,COUNT={YES }]
                               {NO  }
                       [,ERRET=addrx]
```

MTTR=
Specifies the MTTR of the track group for which the specified bit needs to be either tested and/or set.

ADDR=
Specifies the address of the track group map (TGM) for which the specified bit is either to be tested and/or set.

SPOLMSK=
Specifies the address of the spool mask to be used by the $TGMSET routine for validation of track group deallocation.

SET=
Specifies whether to turn the bit on (ON) or off (OFF). The default is to set the bit to 0 (OFF).

QSUSE=
Specifies whether (YES) or not (NO) it is necessary for JES2 to obtain the QSUSE (shared queue control elements) prior to setting the bit in the specified TGM and checkpointing the TGM change.

TYPE=
Specifies the type and degree of error checking that is performed before the specified bit is set in the specified TGM.

SET
Indicates that the specified bit is set in the TGM; no error checking is performed.

TEST
Indicates that the specified bit is tested as follows:

- If the bit is already set the same as specified by the SET= keyword the bit is not set; an error is returned.
- If the SPOLMSK= keyword is coded, a check is made to determine if the specified bit is on for a spool volume specified in the spool mask; if the bit is not on a spool mask volume, the bit is not set and an error is returned.
- If the bit is to be set off is currently set on in the bad track group map, the bit is not set.

Note: Specifying TEST only indicates that the three tests are performed and appropriate error returns are made; the bit tested is **not** set.

TESTSET
Indicates that the specified bit is tested as noted for TEST, above; however, if appropriate, the bit is set.

COUNT=
Specifies whether (YES) or not (NO) the corresponding count fields (DASTGALC in the DAS and $TGALLOC and $TGFREE in the HCT) are updated as specified by the SET= keyword.

ERRET=
Specifies the address of an error routine if an error is returned from the $TGMSET routine. A return code of 0 indicates that the validity test passed; 4 indicates it failed.

Environment

- JES2 main task
- $WAIT can occur if QSUSE=YES is specified

$TIDTAB - Generate the Trace ID Table DSECT

Use $TIDTAB to map and generate trace ID (TID) table entries.

Format Description

```
[symbol]     $TIDTAB    [TABLE={HASP}]
                        [      {USER}]
                        [      {END }]

                        [,NAME=char-name]

                        [,ID=value]

                        [,FORMAT=relexp]
```

TABLE=
Specifies the first entry in the HASP or user TID table or end of a TID table. If TABLE= is specified, all other operands are ignored.

NAME=
Specifies a 1-8 character name for the TID type. This name is used on the first line of output for the trace record. If NAME= is not specified, DSECT mapping is generated, an all other keywords are ignored.

ID=
Specifies a trace identifier, a value from 1-255. The HASP table currently defines identifiers from 1-16.

Note: User entries should use identifiers from 255 down to prevent overlap of trace table ID numbers.

FORMAT=
Specifies the name of a formatting routine that will format the trace records for this type. If the symbol used as the name of this formatting routine is not defined in the assembly module containing this $TIDTAB macro, then the TID table field is defined using a VCON rather than an ACON.

Note: If you omit all the operands on the $TIDTAB macro, a DSECT mapping of a TID table entry is generated; otherwise, an actual TID table entry is generated.

Environment

- Main task or during JES2 initialization and termination.
- $WAIT is not applicable -- this macro generates a DSECT or a static table entry; it does not generate executable code.

$TRACE - Trace a JES2 Activity

Use the $TRACE macro instruction to allocate a JES2 trace table entry (TTE) in an active trace table and return its address. Optionally, $TRACE initializes the TTE based upon parameters passed. The JES2 event trace facility is called to perform the TTE allocation.

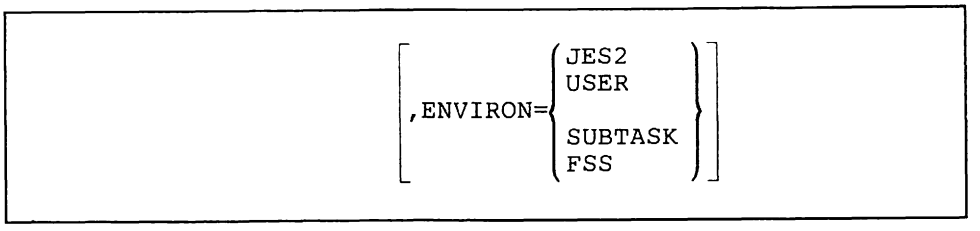
$TRACE can be specified anywhere in the JES2 system (including the HASPSSSM module) except in routines running as disabled interrupt exits (for example, an IOS appendage). R13 must point to a usable OS-style save area. Be certain to also code the $TRP macro on the $MODULE statement to provide required mapping.

On exit, register 1 contains the address of the allocated TTE. Any changes to the TTE must be accomplished prior to issuing a wait (WAIT or $WAIT, explicit or implied). A nonzero condition code on exit indicates that the TTE was successfully allocated; condition code 0 indicates unsuccessful allocation either because tracing is not activated or the individual ID is not currently being traced.

In environments other than the JES2 main task, a $TRACE RELEASE request must be made after the formatted TTE is ready. The $TRACE facility is serialized using a resource that is realized by that request.

Format Description

```
[symbol]     $TRACE      [RELEASE]
                         [,ID={id }]
                              {0 }
                         [,OFF={addrx}]
                               {(Rx) }
                         [,NAME=symbol]
                         [,LEN={(R0) }]
                               {addrx}
                               {0}
                         [,DATA={addrx}]
                                {(R1) }
                                {0}
                         [,BASE={HCT }]
                                {SSVT}
                         [,TYPE={TRUNC}]
                                {SPIN }
```



RELEASE

Specifies the release of the trace buffer. This positional operand must be used in either the user or the subtask environment after the trace table entry fields are coded.

Notes:

1. *If this operand is specified, all other operands except BASE= and ENVIRON= are ignored.*

2. *Before issuing a $TRACE RELEASE, be certain you have loaded register 0 with the value returned in register 0 from the previous $TRACE macro call.*

ID=

Specifies the ID associated with this trace entry. The ID is a value between 1 and 255. If 0 is specified, JES2 does not create a TTE but instructs the event trace facility to spin-off the current trace log data set (if logging is activated). ID=0 is specified in JES2 routines controlling trace activities and should not be specified outside of these areas.

Note: IBM trace IDs start from 1 and increase. User trace IDs should start at 255 and decrease to prevent overlap of ID numbers.

OFF=

Specifies the address that is given control if tracing is not currently being used. If register notation is used, the designated register must be previously loaded with the address.

If this operand is omitted, control is given to the location after the macro expansion with condition code 0.

NAME=symbol

Specifies the 1- to 8-character identifier to be associated with this macro call. If this operand is omitted, the label symbol used by the $TRACE macro call is used. If neither is specified, the current CSECT name is used.

LEN=

Specifies (by a valid expression or through register notation) the length of the trace table entry (TTE) to be allocated. If register notation (R2 - R12) is used, the designated register must be previously loaded with the length. The address (addrx) of either a fullword or halfword containing the length

can be used if neither register notation (Rx) nor a specified value or equated value is used for length. The maximum length is:

(PAGE x 4096) - 68

Where:

PAGE	PAGE parameter on the TRACE initialization statement
68	represents the current header lengths in JES2 trace table control blocks

DATA=
Specifies the address or a register that points to the location where the data to be logged can be found. If this keyword is specified, all activity for the new TTE is performed by the $TRACE facility. If this keyword is not specified or DATA=0 is specified and ENVIRON= is specified as USER, SUBTASK, or FSS, you must issue a $TRACE RELEASE. The returned TTE should be formatted and then a $TRACE RELEASE must be issued.

BASE=
Specifies the base address in register 11 to be used by the $TRACE macro.

HCT
Preload register 11 with the address of the HCT.

SSVT
Preload register 11 with the address of the SSVT. SSVT should be used when using the $TRACE macro instruction in the HASPSSSM environment.

TYPE=
Specifies the action to be taken if ID=0.

TRUNC
the current trace table is to be truncated, and the trace table is passed to the trace log processor (if logging is active).

SPIN
The event trace facility is to spin off the current trace log data set and truncate the current trace table, passing the table to the trace log processor (if logging is active).

ENVIRON=
Specifies the environment in which the $TRACE is issued. The environment determines where the save area is to be located. ENVIRON= defaults to the &ANVIRON global variable which is set by the ENVIRON= keyword on the $MODULE macro.

JES2
Represents the JES2 main task as the environment for tracing. BASE=HCT must be specified or allowed to default.

SUBTASK
Represents a subtask of the JES2 main task as the environment for tracing.

USER
Represents a user address space as the environment for tracing.

FSS
Represents the functional subsystem address space as the environment for tracing.

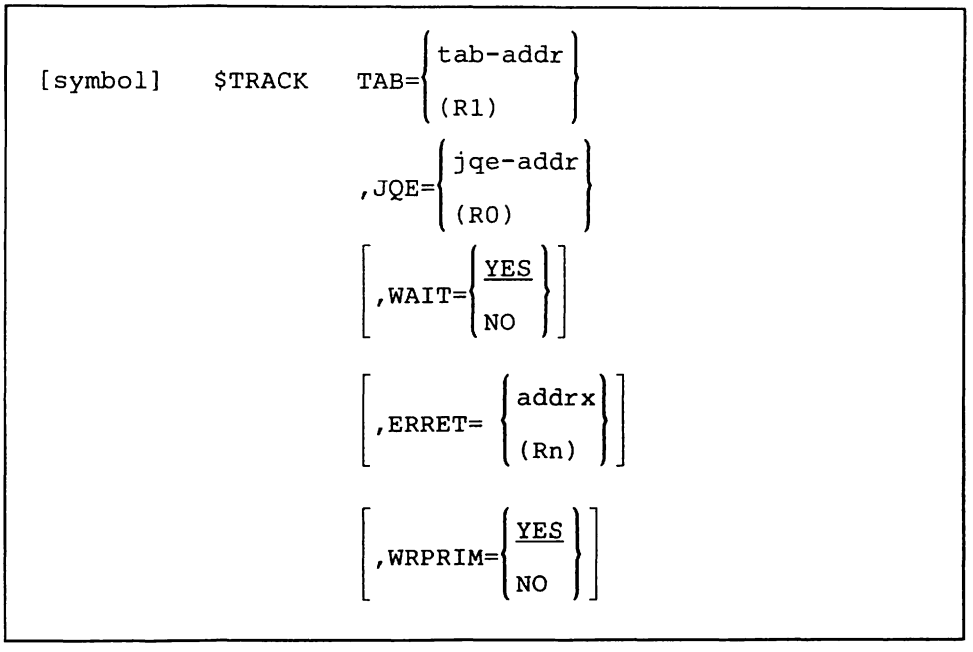
Environment

- JES2 main task, subtask, user, or functional subsystem (HASPFSSM)
- $WAIT cannot occur; however, in other than the JES2 main task environment, an MVS WAIT can occur.

$TRACK - Acquire a Direct-Access Track Address

Use $TRACK to obtain a track address on a JES2 spool volume and return this track address in register 1.

Format Description



TAB=
Specifies the address of the track allocation block (TAB).

JQE=
Specifies the address of the job queue element (JQE). This is required for the JES2 main task environment.

WAIT=
Specifies whether (YES) or not (NO) to wait if $TRACK is unable to successfully allocate a track group. If YES is specified, the service routine will issue a $WAIT TRAK if no tracks are currently available. routine will issue a $WAIT TRAK if no tracks are currently available.

ERRET=
Specifies the address (or register that contains the address) of an error routine that gets control if register 15 contains a non-zero return code from $STRAK.

Return Code	Meaning (User Address Space)
0	Allocation successful
8	Error encountered in $STRAK

Return Code	Meaning (JES2 Address Space)
0	Allocation successful within the same track group
4	Allocation successful in a different track group
8	WAIT=NO was specified -- no track group returned

WRPRIM=

Specifies whether (YES) or not (NO) to write the primary allocation IOT if a new track group is allocated. WRPRIM=YES is the default.

Return Code	Meaning
0	Allocation successful within the same track group
4	Allocation successful in a different track group
8	WAIT=NO was specified -- no track group returned

Environment

- Main task and user address space.
- $WAIT can occur.

$TTIMER - Test Interval Timer

Use $TTIMER to obtain the time remaining in the associated time interval that was previously set with $STIMER macro instruction. The value of the remaining time interval is returned in register 0 in seconds (rounded to the nearest second). The $TTIMER macro instruction can also be used to cancel the associated time interval.

Format Description

```
[symbol]      $TTIMER   { loc-addrx }
                        { (R1)    }
                        [,CANCEL]
```

loc

Specifies the address of the timer queue element. If register notation is used, the address must have been loaded into the designated register before the execution of this macro instruction.

If the timer queue element is not active or if the interval has expired before the $TTIMER macro instruction is executed, the value of the time interval returned is 0.

CANCEL

Specifies that the interval in effect should be cancelled.

If this operand is omitted, processing continues with the unexpired portion of the interval still in effect.

If the timer queue element is not active or if the interval has expired before the $TTIMER macro instruction is executed, the CANCEL operand has no effect.

Environment

- Main task.
- $WAIT cannot occur.

$VERIFY - Call the $VERIFY Service Routine

Use the $VERIFY macro instruction to call the $VERIFY service routine located in the HASPSSSM load module to validate control block contents when read in from spool.

Format Description

```
[symbol]    $VERIFY   TYPE=type-code
                      ,BUFAD={addrx}
                             {(R0) }
                     [,KEYFLD={addrx}]
                              {(R1) }
                     [,ERRET={addrx}]
```

TYPE=
Specifies the $VERIFY type-code (0-255) as mapped in $HASPEQU. This type code is required for control block verification.

BUFAD=
Specifies the address of the buffer that contains the control block to be verified.

KEYFLD=
Specifies the address of the field to be used to verify the control block.

ERRET=
Specifies the address of an error routine that is to get control if an control block error is detected or the control block is not verified

Environment

- JES2 main task, user task, or HASPFSSM address space
- No waits can occur

$VERTAB - Build the Inline Verification Tables

Use the $VERTAB macro instruction to build the inline verification tables used by the $VERIFY service routine.

Format Description

```
[symbol]   $VERTAB  TYPE=type-code
                                 ┌ JCT  ┐
                                 │ IOT  │
                                 │ OCT  │
                          ,ID=   │      │
                                 │ SWBI │
                                 │ CHK  │
                                 └ ANY  ┘
                          ,IDOFF=id-offset
                          [,KEYOFF=key-offset]
                          [,KEYLEN=key-length]
                          [,TABLE=┌ HASP ┐]
                                  └ END  ┘
```

TYPE=
Specifies the $VERIFY type-code (0-255) as mapped in $HASPEQU. This type code is required for the control block verification table that this macro is building.

ID=
Specifies the EBCDIC identifier (maximum of 4 characters) for the control block verification table that this macro builds.

IDOFF=
Specifies the offset of the control block identifier.

KEYOFF=
Specifies the offset of the key field for the control block verification field that this macro builds.

KEYLEN=
Specifies the length of the key field for the control block verification field that this macro builds.

TABLE=
Specifies the beginning (HASP) or end (END) of the verification table.

Note: The verification table is prefixed by a control block pool header.

Environment

- JES2 main task, user task, and HASPFSSM address space
- $WAIT is not applicable -- this macro generates a DSECT or a static table entry; it does not generate executable code.

$VFL - Variable Field Length Instruction Operation

Use $VFL to provide certain storage-to-storage operations where the field lengths exceed 256 bytes or where no assembler instructions exist.

Format Description

```
[symbol]      $VFL      op-code
                        ,{to-addrx }
                         {(R1)     }
                        ,{from-addrx }
                         {(R15)      }
                        ,{length-addrx }
                         {(R0)         }
```

op-code
Specifies the storage-to-storage operation as one of the following:

NC
And operation

XC
Exclusive or operation

OC
Or operation

MVC
Nondestructive overlapping move operation (see note 3)

to-addrx
Specifies the address of the first field (see note 1).

from-addrx
Specifies the address of the second field (note 2).

length-addrx
Specifies the total number of bytes in the field (see note 1).

Notes:

1. *If the length operand is written as an address, the register contains the address of a fullword which contains the address of the field (which contains the field length).*

If the length operand is written using register notation, it represents the address of the field that contains the field length. If register notation is used, the address (or field length) must be loaded into the designated register before the execution of the macro instruction.

2. *Condition codes from the execution of this macro are not usable.*

3. *When MVC is specified, a shift character long operation is performed. The number of bytes specified by length is moved from the from address to the to address. The origin and destination fields may overlap in any desired manner; the character string is moved intact without propagating the nonoverlapping portion of the fields. $VFL MVC is intended to be used in exceptional situations. For performance reasons, it should not be used where MVCL or an executed MVC would suffice.*

Environment

- Main task and subtask (where R11 = address of HCT).
- User address space (where R11 = address of SSVT)
- $WAIT cannot occur.

$WAIT - Wait for a JES2 Event

Use $WAIT to place the associated processor in a JES2 wait state and specify the event or resource for which the processor is waiting.

Optionally, use $WAIT to specify an extended ECB structure (XECB) which may be posted by OS/VS service or some other task. If the XECB has already been posted, $WAIT returns immediately to the processor; otherwise, $WAIT initializes the extended ECB and places the processor in a JES2 wait state.

Format Description

```
[symbol]  $WAIT     event/resource
                    [,INHIBIT={NO  }]
                              {YES }
                    [,XECB={addrx}]
                           {(R1) }
```

event/resource
Specifies the JES2 event or resource for which the processor is to wait as one of the following:

Event:

HOLD
Waiting for a $S operator command

IO
Waiting for the completion of an input/output operation

OPER
Waiting to be reactivated

POST
Waiting for some resource or any $POST

WORK
Waiting for more work

Resource:

ABIT
Waiting for the next dispatcher cycle

ALOC
Waiting for allocation

BUF
Waiting for a JES2 buffer

CKPTP
Waiting for a checkpoint post

CKPT
Waiting for the completion of a JES2 checkpoint

CKPTW
Waiting for checkpoint work

CMB
Waiting for a console message buffer

CNVT
Waiting for a converter

FSS
Waiting for completion of FSS-level processing

HOPE
Waiting for an output processor

IMAGE
Waiting for a UCS or FCB image load completion

JOE
Waiting for a JOE to be freed

JOT
Waiting for job output table service

JOB
Waiting for a job

LOCK
Waiting for a lock

MAIN
Waiting for storage

PSO
PSO processor waiting for work

PURGE
Purge processor is waiting for work

PURGS
Waiting for purge resources from $PURGER

RSV
Waiting for RESERVE

SMF
Waiting for an SMF buffer

TRAK
Waiting for a direct-access track address

UNIT
Waiting for a device control table

INHIBIT=
Specifies whether the processor issuing this macro instruction is to be dispatched if specifically posted ($POST).

YES
All posts ($POST) specifying this processor are ignored, except for the one indicating completion of the event specified in this macro instruction, or the one indicating the optional XECB has been POSTed.

NO
The processor issuing this macro instruction is to be dispatched if specifically posted ($POST) for any event.

XECB=
Specifies the address of an XECB. The processor issuing this macro instruction will be dispatched when the XECB is posted, or immediately resumes control if the XECB has already been posted. If register notation is used, the designated register must be loaded with the address of the XECB prior to executing this macro.

Notes:

1. *The execution of this macro instruction requires register 15; additionally, register 1 is required if the XECB option is used.*

2. *The JES2 processor is dispatched if either the JES2 event/resource is posted or the ECB in the XECB control block is posted.*

Caution:

- **If the XECB option is used, the processor is dispatched by either the JES2 event occurrence or the POST of the XECB.**

- **Upon return from $WAIT, the XECB may not be free for other uses; it may still be chained within a JES2 dispatcher queue.**

Environment

- Main task.
- $WAIT is not applicable.

$WSSETUP - Set Values Required for Work Selection

Use the $WSSETUP macro instruction to set those values that are required to support work selection.

Format Description

```
[symbol]    $WSSETUP    DEVADDR={label}
                            {(R1)  }
                        [,TYPE=cb-name      ]
                        [,VOL=dev-vol-field ]
                        [,VOLNUM=dev-vol-num]
```

DEVADDR=
Specifies the address of either a DCT or PIT. Specify this address either by a label or a register; the address is loaded in register 1.

TYPE=
Specifies the device control block name used to calculate the offset for the fields specified by the VOL= and VOLNUM= keywords. If this keyword is specified, both VOL= and VOLNUM= must also be specified.

VOL=
Specifies the device's volume field. The offset for this field is calculated using the name specified by the TYPE= keyword. If this keyword is specified, both TYPE= and VOLNUM= must also be specified.

VOLNUM=
Specifies the volume number field. The offset for this field is calculated using the name specified by the TYPE= keyword. If this keyword is specified, both TYPE= and VOL= must also be specified.

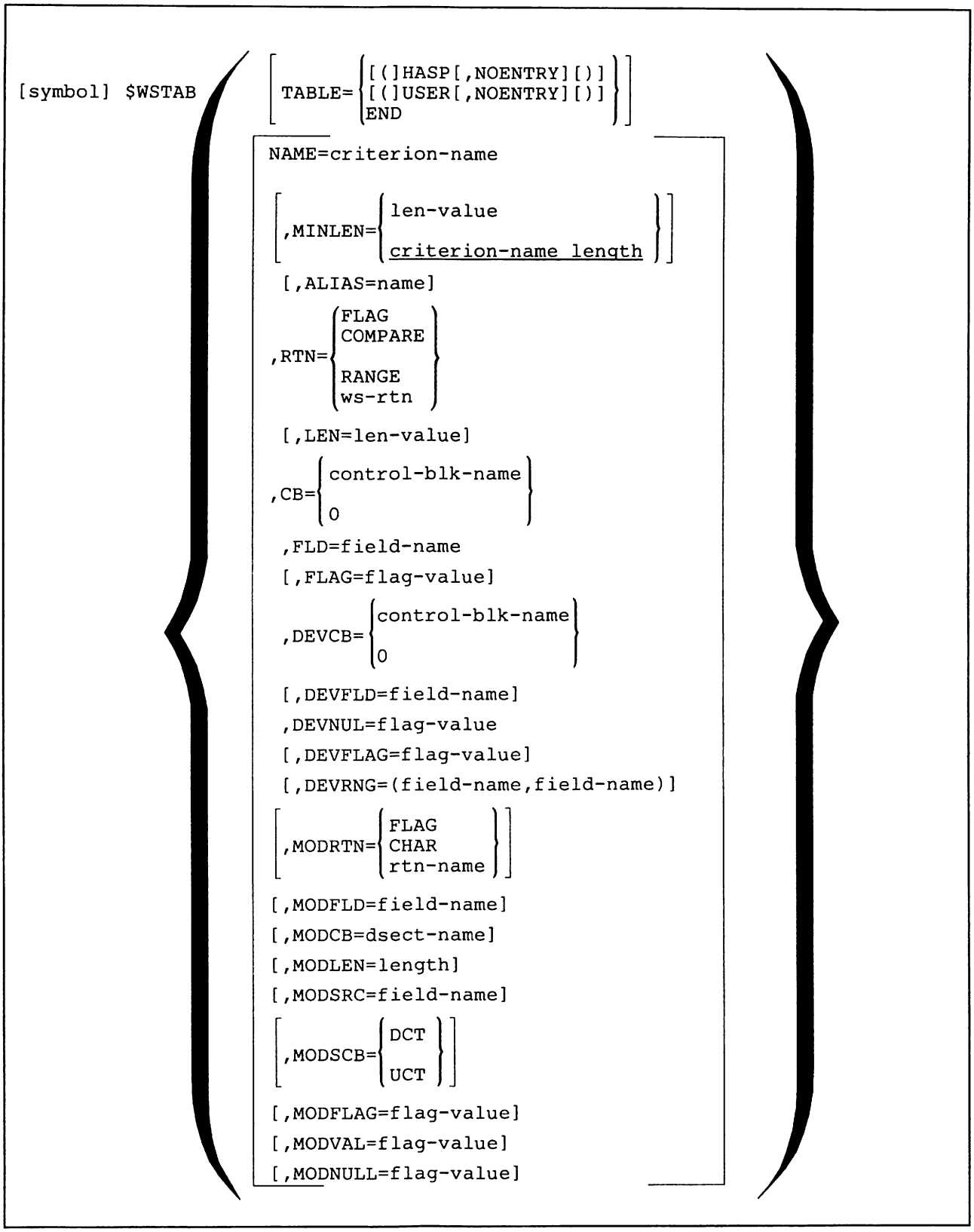
Environment

- JES2 main task
- $WAIT cannot occur

$WSTAB - Map and Generate the Work Selection Table Entries

Use the $WSTAB macro instruction to map and generate the work selection table entries.

Format Description



NAME=
Specifies a 1- to 8-character name for an individual work selection criterion.

MINLEN=
Specifies the minimum length that is acceptable for the criterion name specified on the NAME= keyword. This keyword does not also support the ALIAS= keyword. If this keyword is not specified, the length defaults to the length of the ws-name specified by the NAME= keyword.

ALIAS=
Specifies an alternate (alias) 1- to 4-byte character name for the work selection criterion. MINLEN= does not support this keyword.

RTN=
Specifies the name of the routine used to check the comparison field against the device field. This keyword is required.

FLAG
Indicates to call the general flag routine for this criterion during work selection.

COMPARE
Indicates to call the general compare routine for this criterion during work selection.

RANGE
Indicates to call the general range routine for this criterion during work selection.

ws-name
Specifies any other valid work selection routine that is to be called.

CB=
Specifies the control block name required to resolve the field specified by the FLD= keyword. This keyword is required. The valid control block names follow.

Control Block	Meaning
JQE	JQE required
WJOE	Work JOE required
CJOE	Character JOE required
HCT	HCT required
NJHO	Spool offload header section
NJHG	General section of job header required
NJH2	JES2 section of job header required
NJHU	User section of job header required
NDHG	General section of data set header required

NDHA	3800 Printer section of data set header required
NDHS	Data stream section of data set header required
NDHU	User section of data set header required
ZERO	No control block required for specified criterion. If CB=ZERO is specified, both FLD= and FLAG= keywords (if also specified) are ignored.

FLD=

Specifies the name of the work selection comparison field against which the device field is compared. If the FLAG= keyword is specified, this keyword is used as a flag byte to be compared against the flag byte setting specified by FLAG=.

FLAG=

Specifies the label of the field in the FLD= byte to be tested under mask for work selection. If FLAG= is specified, FLD= must also be specified.

LEN=

Specifies the length of a character comparison between a control block field and a device field. This keyword defaults to the length of the comparison field specified by the FLD= keyword.

TABLE=

Specifies the start or end of a work selection table.

Specify TABLE=HASP or TABLE=USER to start the corresponding table, and optionally a second parameter of NOENTRY (e.g. TABLE=(USER,NOENTRY)) to indicate no ENTRY statement need be generated for the label of the scan table.

Specify TABLE=END to terminate a scan table.

Other operands are ignored if TABLE is specified, and a label is required on the $WSTAB macro if a table is being started. If TABLE= and NAME= are both not specified, only the mapping of an $WSTAB entry is generated by the macro.

DEVCB=

Specifies the control block name required to resolve the field specified by the DEVFLD= keyword. The valid control block names follow.

Control Block	Meaning
DCT	DCT required
PIT	PIT required
HCT	HCT required
UCT	UCT required
ZERO	No control block required for specified criterion. If DEVCB=ZERO is specified, both DEVFLD= and DEVFLAG= keywords (if also specified) are ignored.

DEVFLD=
Specifies the name of the device field against which work selection comparison field (FLD=) is compared. If the DEVFLAG= keyword is specified, this keyword is used as a flag byte to be compared against the flag byte setting specified by DEVFLD=. Also, if DEVFLD= is specified, DEVRNG= cannot also be specified.

DEVFLAG=
Specifies the label in the DEVFLD= byte to be tested under mask for work selection. If DEVFLAG= is specified, DEVRNG= cannot also be specified.

DEVRNG=
Specifies the names of the upper and lower device fields against which the control block field (as specified by the CB= and FLD= keywords) is compared. If either DEVFLAG= or DEVFLD= are specified, this keyword cannot also be specified.

DEVNUL=
Specifies the value for the device flag byte that is used to determine if a null value was specified for this criterion.

MODRTN=
Specifies the name of the routine used to modify the criterion following selection by a offload receiver. This keyword is required if the criterion is to be modified.

FLAG
Indicates that the general flag routine is to be called

CHAR
Indicates that the general character routine is to be called

rtn-name
Indicates a valid modify routine that is to be called

MODFLD=
Specifies the name of the field that is to be modified when the job or SYSOUT is reloaded.

MODCB=
Specifies the DSECT name required to resolve the field specified by the MODFLD= keyword. Valid field names are:

Field Name	Meaning
NJHG	General section of the job header
NJHO	Spool offload header section
NJH2	JES2 section of the job header
NJHU	User section of the job header
NDHG	General section of the data set header
NJHA	3800 printer section of the data set header
NDHS	Data stream section of the data set header
NDHU	User section of the data set header

MODLEN=
Specifies the length of the field that is to be modified. If this length is not specified, it defaults to the length of the name specified by the MODFLD= keyword.

MODSRC=
Specifies the name of the field in the control block indicated by the MODSCB= keyword that contains the value that replaces the current value in the field specified by the MODFLD= keyword. If MODRTN= is set to either FLAG or CHAR this keyword is required.

MODSCB=
Specifies the name of the valid control block (DCT or UCT) containing the MODSRC field. This keyword is required if MODSRC= is specified.

MODFLAG=
Specifies the flag value to be set in MODFLD if MODRTN=FLAG is specified. This keyword is required if MODRTN=FLAG is specified.

MODVAL=
If MODRTN=FLAG is specified, this keyword specifies a mask that is compared against the byte specified by the MODSRC= keyword. If the flag is set, then the flag specified by the MODFLAG= keyword is turned off in the byte specified by the MODFLD= keyword. If the flag is not set, then the flag specified by the MODFLAG= keyword is turned off in the byte specified by MODFLD=. This keyword is required if MODRTN= is specified.

Note: MODVAL and MODNULL must both map to the same MODFLD= byte.

MODNULL=
If MODRTN=FLAG is specified, this keyword specifies a mask that is compared against the value specified by the MODSRC= keyword. If the null flag is set, then the device characteristic in the MOD= list has previously been set to NULL and is not modified.

Note: MODNULL and MODFLD must both map to the same MODFLD= byte.

Environment

- JES2 main task
- $WAIT cannot occur in a routine specified by either the RTN= or MODRTN= keywords

$WTO - JES2 Write to Operator

Use $WTO to initiate the display of a message intended for the operator either on one or more operating system consoles or a JES2 remote work station console or printer device.

Format Description - Standard Form

```
[symbol]    $WTO    { message-addrx }
                    { (R1)          }
                   ,{ length-absexp }
                    { (R0)          }
                   [,JOB={ YES }]
                         { NO  }
                   [,WAIT={ YES   }]
                          { NO    }
                          { relexp }
                   [,TYPE={ SVC35 }]
                          { SVC34 }
                          { WPL   }
                   [,ROUTE=code]
                   [,CLASS={ $DORMANT }]
                           { $ALWAYS  }
                           { $ACTION  }
                           { $NORMAL  }
                           { $TRIVIA  }
                   [,PRI={ $HI }]
                         { $ST }
                         { $LO }
                   [,CMB={ YES }]
                         { NO  }
                   [,RMT={ YES }]
                         { NO  }
                   [,UCM={ YES }]
                         { NO  }
```

Format Description - Execution Form

```
[symbol]      $WTO      {message-addrx}
                        {(R1)        }
                        [,{length-absexp}]
                        [,{(R0)        }]
                        [,CMB={YES}]
                        [,CMB={NO }]
                        [,WAIT=relexp]
                        [,MF={E,name }]
                        [,MF={EX,name}]
```

Format Description - List Form

```
name    $WTO          [,length-absexp]
                      [,JOB={YES}]
                      [,JOB={NO }]
                      [,WAIT={YES}]
                      [,WAIT={NO }]
                      [,TYPE={SVC35}]
                      [,TYPE={SVC34}]
                      [,TYPE={WPL  }]
                      [,ROUTE=code]
                      [,CLASS={$DORMANT}]
                      [,CLASS={$ALWAYS }]
                      [,CLASS={$ACTION }]
                      [,CLASS={$NORMAL }]
                      [,CLASS={$TRIVIA }]
                      [,PRI={$HI}]
                      [,PRI={$ST}]
                      [,PRI={$LO}]
                      [,MF={L }]
                      [,MF={LX}]
                      [,RMT={YES}]
                      [,RMT={NO }]
                      [,UCM={YES}]
                      [,UCM={NO }]
```

message
Specifies the address of a message which is to be displayed on the designated consoles or the address of a console message buffer (CMB) in which the message resides if CMB=YES is specified. If register notation is used, the address must be loaded into the designated register before the execution of this macro instruction.

length
Specifies the length of the above message. If register notation is used, the value must be loaded into the rightmost byte of the register (R0) before the execution of the macro instruction. The rest of the register must be 0 unless the message is being sent to a remote terminal (see RMT operand).

Note: When using the execute and list forms of the macro instruction as a pair, the length must be specified in one of the macro instructions but not both.

JOB=
Specifies whether the job identification and job name from the job control table (JCT) or the job queue element (JQE) are appended to the start of the messages as follows:

YES (default)
The job information is appended to the message.

NO
The job information is not appended to the message.

If this operand is omitted, JOB=YES is assumed.

Caution: If JOB=YES is specified, register 10 must be loaded with the address of either the job control table entry or the job queue element before the execution of this macro instruction, or the job information printed is unpredictable.

WAIT=
Specifies the action to be taken in the event that no console message buffers are available as follows:

YES
Return is not made until a console message buffer has become available and the message has been queued.

NO
An immediate return is made with the condition code set as follows:

CC=0
No console message buffers are available. The message was not accepted and the macro instruction must be reissued.

CC≠0
The message was accepted.

relexp
A location to which control is returned if CMB=YES is not specified and no console message buffers are available; or a location to which control is returned if CMB=YES is specified and the control of the MLWTO resource was required but not obtained.

If this operand is omitted, WAIT=YES is assumed.

Notes:

1. *The specification of WAIT=YES or WAIT=NO has no meaning when the console message buffer is provided by the user and CMB=YES is specified.*

2. *The specification of WAIT=relexp has no meaning if MF=L or LX.*

CMB=
Specifies whether or not the user of the $WTO macro instruction has provided a console message buffer for use by console services as follows:

NO (default)
A console message buffer has not been provided, and the message operand refers to message text.

YES
A console message buffer has been provided, and the message operand refers to the console message buffer containing the user's message.

If this operand is omitted, CMB=NO is assumed.

Notes:

1. *If CMB=YES is specified, the message must appear in the appropriate locations within the console message buffer as follows:*

   JOB=YES
   Message starts in CMBTEXT field

   JOB=NO
   Message starts in CMBJOBN field

2. *If CMB=YES and an address is not specified by the WAIT operand as a nonprocess exit, the user must test the condition codes following the macro instruction when multiline write-to-operator messages (MLWTO) are specified. The condition codes indicate the following:*

   CC=0
   The MLWTO was rejected, and the CMB is available to the caller.

   CC≠0
   The message has been queued for display, and the console message buffer is not available for use by the user.

3. *If CMB=YES is specified, the CMB must have been obtained by the use of the $GETCMB macro instruction which deletes the previously-obtained CMB.*

4. *The $MSG macro instruction should be used to generate the message text or, if dynamically generated, at least the message identification section.*

TYPE=

Specifies the logical meaning of the ROUTE operand as follows:

SVC35 (default)

The message is to be displayed on the dynamically designated remote work station console or the operating system consoles which have been set to receive the message category.

Note: If this option is selected, use one and only one of the following specifications: ROUTE= or UCM= or RMT=

SVC34

The message is to be given to the operating system as a command. If this option is selected, ROUTE=, UCM=, and RMT= have no meaning.

WPL

The message is to be displayed on the operating system consoles; however, the message address must be an MVS WTO parameter list (WPL). (You can generate this list by using the list form of the MVS WTO macro.) Refer to *Supervisor Macros* for a description and use of this MVS macro. Also, the specified WPL must include routing and descriptor codes.

Note: If this option is selected, CMB=NO must be specified (or defaulted), and MF=, RMT=, UCM=, and ROUTE= are ignored, if also specified.

ROUTE=

When using standard and MF=L forms, specifies the JES2 logical routings which are converted to operating system equivalent routings as follows:

Designation	Console Specified
$LOG	Hard-copy console
$ERR	Error console(s)
$UR	Unit-record operations area
$TP	Teleprocessing operations area
$TAPE	Tape operations area
$MAIN	Chief operator's area
$ALL	All of the above consoles
0	UCMID or logical console are specified dynamically.

When using the MF=LX format, the operating system console routings must be specified directly using the X'xxxx' or B'xxxxxxxxxxxxxxxx' form.

If this operand is omitted, the $LOG console is assumed unless UCM=YES or RMT=YES is specified.

Caution: The designation $ALL should not be used in conjunction with any other console but should be specified alone; otherwise, results are unpredictable.

RMT=
Specifies whether or not the display location is to be a JES2 remote work station.

NO
The destination is not a remote work station and therefore must be logical routing or UCM.

YES
The destination is a work station, and the following action is required prior to executing execution forms of the macro instruction:

- For standard or MF=E forms, the remote number is set in register 0 (R0); 0 must be set into bits 0-7, the remote number must be set into bits 8-23, and the length of the message must be set into bits 24-31. R0 must be specified in the executing macro instruction's length operand.

- For MF=EX form, the remote number must be placed in the MF=LX halfword field corresponding to CMBRMT field of the CMBDSECT.

UCM=
Specifies whether or not the display location is to be a specific operating system console identified by a unit control module (UCM) number.

NO
The destination is not a UCM console number and therefore must be logical routing or RMT.

YES
The destination is a UCM console number, and the following action is required prior to executing execution forms of the macro instruction:

- For standard or MF=E forms, the UCM console number must be set into register 0 (R0) with bits 0-15 set to zero, the UCM console id in bits 16-23, and the length in bits 24-31.

- For MF=EX form, UCM console, UCM console area, and MLWTO line type information must be placed in the MF=LX field corresponding to the CMBUCM, CMBUCMA, and CMBLMET fields of the CMBDSECT.

When using extended forms of the WTO macro instruction, the MF=LX format of macro is used to create a partial parameter list in the format of the resulting console message buffer (CMB) used to queue the message for display. This list can be mapped by the $CMB macro instruction offset so that byte 1 of the macro expansion corresponds with the CMBFLAG field. The parameter list is 14 bytes long. The fields corresponding to the

CMBTO control fields must be set into the parameter list prior to execution of the corresponding MF=EX form of the $WTO macro instruction. Normally this is done by moving in the $SYSID field of the HASP communications table (HCT). Other fields may be required depending upon options specified in the MF=LX format as follows:

ROUTE =specified
No setting required in the CMBOUT field.

UCM=YES and RMT=YES are prohibited.

RMT =YES
Remote work station number must be set in the CMBRMT byte of the CMBOUT field.

ROUTE and UCM=YES are prohibited.

UCM =YES
Operating system UCM information is required:

- UCM number is set in CMBUCM.
- UCM area for MLWTOs is set in CMBUCMA (CMBUCMA set to 0 indicates single line WTO).
- UCM.MLWTO line type is set in CMBLINET (required for CMBUCMA nonzero.)

Use of MLWTO formats of the parameter list is reserved for the command processor.

Precautions must be taken not to issue a normal $WTO by the command processor or any processor on which the command processor waits from the time the first (control) MLWTO line $WTO is issued until the last (END) MLWTO line $WTO is issued. A **violation of this rule might cause console lock out.**

CLASS=
Specifies the class of the message as one of the following:

$DOMACT
The message requires immediate action and is always written.

$ALWAYS
The message is essential and is always written.

$ACTION
The message requires eventual operator action.

$NORMAL
The message is considered important to normal computer operations.

$TRIVIA
The message is considered unimportant to normal computer operations.

If this operand is omitted, $NORMAL is assumed.

The $DOMACT specification is reserved for $WTOs issued to logical consoles. On return from $WTO processing, register 1 contains the address of the console message buffer (CMB) containing the message. The CMB is retained in the system until a corresponding $DOM is executed using the returned pointer.

PRI=
Specifies the priority of the message as one of the following:

$HI
High priority

$ST
Standard priority

$LO
Low priority

If this operand is omitted, $ST priority is assumed.

Notes:

1. *MF=L or MF=E are forms of the $WTO macro instruction that allow the predefinition of parameter lists similar to MVS supervisor services macros specified with the "execute" or "list" forms.*

2. *MF=LX or MF=EX are extended forms of MF=L or MF=E, respectively. These extended forms allow you to specify the $WTO macro instruction without using $GETCMB, formatting the CMB in detail, and using $WTO CMB=YES.*

3. *The CMBTO field in the CMB can be filled out to route messages to another node in the JES2 network or to another member in the multi-access spool JES2 complex. Field CMBUCM can be set to indicate not only the console id, but also the console area. MLWTOs can be used with explicit route and descriptor codes. To do this, you must format the console area with the CMB mapping and the MF=EX form of the $WTO macro instruction.*

4. *The MF=LX form of the $WTO macro instruction only allows explicit route codes via ROUTE=B'xxxxxxxxx...' or X'xxxx'. Descriptor codes cannot be specified using the MF=LX form of the $WTO macro.*

5. *The CMB mapping starts with the CMBFLAG field and* **not** *the CMBDSECT field.*

Environment

- Main task.
- $WAIT can occur depending on how WAIT= is specified.

$XMPOST - POST Task in Another Address Space

Use $XMPOST to POST a task in another address space.

Format Description

```
[symbol]    $XMPOST    { element-addrx }
                       { (R1)          }
```

element
Specifies either register 1 which contains the address or the address of a 3-word POST element formatted as follows:

+0
Address of ECB to be posted ($POST)

+4
Address of related ASCB

+8
Address of error return

Environment

- Main task.
- $WAIT cannot occur.

Appendix A: Acronym List

Appendix A. Acronym List

ACB	-	access control block
ACF	-	advanced communication function
APAR	-	authorized program analysis report
APT	-	application table
BAL	-	basic assembler language
BSAM	-	basic sequential access method
BSC	-	binary synchronous communication
BSCA	-	binary synchronous communication adapter
CCE	-	cell control element
CCW	-	channel command word
CLPA	-	common link pack area
CMB	-	console message buffer
CMS	-	cross memory services
CSA	-	common service area
CSECT	-	control section
CTC	-	channel-to-channel
DAS	-	direct access control blocks
DASD	-	direct access storage device
DCT	-	device control table
DTE	-	daughter task element
EBCDIC	-	extended binary-coded decimal interchange code
EM	-	end of media
EOF	-	end of file
ERA	-	error recovery area
FCB	-	forms control buffer
FSA	-	functional subsystem application
FSACB	-	functional subsystem application control block
FSS	-	functional subsystem
FSSCB	-	functional subsystem control block
FSVT	-	functional subsystem vector table
GRS	-	global resources serialization
HASP	-	Houston automatic spooling priority
HCT	-	HASP communication table
HFCT	-	HASP functional subsystem communication table
I/O	-	input/output
IOT	-	input/output table
IPL	-	initial program load
IPS	-	installation performance specification
JES2	-	job entry subsystem 2
JCL	-	job control language
JCT	-	job control table
JIB	-	JOE information block
JIX	-	job queue index
JMR	-	job management record
JOE	-	job output element
JOT	-	job output table
JQE	-	job queue element
JSPA	-	job separator page data area
LMT	-	load module table
LPA	-	link pack area
LRECL	-	logical record length
LU	-	logical unit
MCT	-	master control table
MIT	-	module information table
MLU	-	multiple logical unit
MSS	-	mass storage system
MTTR	-	module relative track/record
MVS	-	multiple virtual storage
NACT	-	network account table
NAT	-	nodes attached table
NCP	-	network control program
NCP/VS	-	network control program/VS
NDH	-	network data set header
NIT	-	nodes information table
NJE	-	network job entry
NJH	-	network job header
NPM	-	network path manager
PCE	-	processor control element
PDDB	-	peripheral data definition block
PDIR	-	peripheral data information record
PID	-	Program Information Distribution
PIT	-	partition information table
PLPA	-	pageable link pack area
PPL	-	purge parameter list
PRE	-	processor recovery element
PSO	-	process SYSOUT processor
PTF	-	program temporary fix

PU - physical unit
QCT - quick cell control table
QSE - shared queue element
RACF - Resource Access Control Facility
RAT - remote attribute table
RMT - remote
RPL - request parameter list
RPS - rotational position sensing
RTAM - remote terminal access method
RTP - remote terminal program
SAM - sequential access method
SCWA - scan work area
SDLC - synchronous data link control
SMF - system management facility
SMP - system modification program
SNA - system network architecture
SPOOL - simultaneous peripheral operations online
SRM - system resources manager
SSVT - subsystem vector table
STC - started task control
SVC - supervisor call instruction
SWA - scheduler work area
SWB - scheduler work block
SYSIN - system input
SYSOUT - system output
TGAE - track group allocation element
TGB - track group block
TOD - time-of-day
TRC - table reference character
TRCA - termination recovery communication area
TSO - time-sharing option
TSO/E - time-sharing option extension
TSU - time-sharing user
TTE - trace table entry
UCB - unit control block
UCM - unit control module
UCS - universal character set
UCT - user control table
VIO - virtual input/output
VTAM - virtual telecommunication access method
WS - work selection
WTO - write-to-operator
WTOR - write-to-operator with reply
XA - extended architecture
XIT - exit information table
XFER - transfer
XRT - exit routine table

Appendix B: Hints for Coding JES2 Exit Routines

Appendix B. Hints for Coding JES2 Exit Routines

Overview

This appendix presents several hints and suggestions for writing JES2 installation exits. Following these hints can help you in the following ways:

- Improve your code's readability and simplify debugging of your exit code.
- Ease migration to a new release or maintenance level.
- Reduce the number of errors in your exit code.

Assembler Instructions

- All USING/DROP statements should be paired. No overriding USINGs should be used except when PUSH/POP is used. This helps prevent errors caused by incorrect base registers.

- All TM (test-under-mask) instructions should use BO/BOR/BNO/BNOR/BM/BMR branch instructions rather than BZ/BZR/BNZ/BNZR branch instructions. If this technique is used, the logic of the branch instruction does not have to be modified when adding or deleting flags in the instruction mask.

- Branches to *- or *+ should not be used except in macro code. This reduces the possibility of causing errors when inserting new lines of code that change the offset of the instruction to which the code is branching.

- Branch tables should be fully coded and documented. Branches to a non-labeled line immediately after the branch table should not be used.

- To increase code readability, all branch instructions should use the extended mnemonic instructions for both RX and RR machine instruction formats.

- All flag bits in flag-byte fields should be defined by equated symbols. Explicit hexadecimal constants should not be used within instructions to represent flag-bit settings. This allows easy reference to a given flag setting. The SI format instructions TM, OI, NI, and XI should also use equated symbols. To provide easy reference, these instructions should use equated symbols for their masks.

- When the implied length of the target field cannot be used, instructions containing length fields should use equated symbols, not hard-coded lengths. Therefore, only a reassembly is necessary if the length of the field is changed.

Constants

- Rather than using literals, the HCT/HFCT/SSVT DSECTs define many constants which you should use whenever possible. The following are a few examples from the HCT:

  - $ZEROES - doubleword of binary zeroes
  - $F1 - fullword binary one
  - $H4 - halfword binary four
  - $BLANKS - doubleword of EBCDIC blanks (X'40')

DSECTs

- For ease of migration, mapping DSECTs used as templates should not be explicitly duplicated within source code. An example of this technique is the use of JES2 $PDDB macro.

- Whenever possible, the use of locally-defined DSECTs, macros, or equated symbols should be avoided. This technique helps to avoid future migration problems.

Registers

- Equated symbols for general purpose registers 0 to 15 (R0-R15) should be used.

- The general-purpose register equates used throughout JES2 are as follows:

R0	Parameter passing
R1	Parameter passing
R11	HCT addressability (JES2 main task)
R11	HCT addressability (JES2 subtasks)
R11	HFCT addressability (HASPFSSM)
R11	SSVT addressability (HASPSSSM)
R12	Local addressability if $SAVE/$RETURN
R13	PCE addressability (JES2 main task)
R13	Save area address (HASPFSSM)
R14	Return address
R15	Entry address/return code

Miscellaneous

- Returned information used for routines and subroutines should use return codes, not condition codes. All return codes should be passed in register 15.

- Except in critical performance areas, the use of dynamic work areas rather than PCE work areas (for example, using $GETCMB to obtain a message building work area) is recommended. Dynamic work areas should be used to

prevent unnecessary wasted storage caused by defining many unique PCE work area fields.

- The inclusive OR instruction (OC) should not be used to test whether a field is zero or non-zero. The OC can cause unnecessary page-outs, thus incurring needless system overhead. Rather, the CLC (compare logical) instruction can be used to compare the field with an appropriate constant (for example, $ZEROES).

- All code should be documented clearly and concisely. A good rule is to document every line of code. In addition, block comments should be used to document every module, routine, and subroutine. These comments should include detailed information on the function of the routine, register values required on entry and exit, register usage within the routine, and possible return codes.

Required Mapping Macros

Depending on the environment in which an exit executes, you will need to provide the appropriate set of mapping macros to map storage areas. Below, listed by environment, are the standard mapping macros required in order that your exit routine will assemble properly. You should also note that individual exits also require other specific mapping macros. These are listed under the "Mapping Macros Normally Required for This Exit At Assembly" heading provided for each exit.

Note: The addition of $MODULE in each exit will caused JES2 to "pull in" required mapping macros. However, all macros should be explicitly coded to prevent the return of MNOTEs and the possibility of assembly errors. Be certain your exit routines conform to JES2 coding conventions. This will allow easier diagnosis if an error should occur.

JES2 Main Task Environment Exits

0 - 5
7
10 - 11
13 - 22
24
26

Assuming you minimally code the following for each exit.

START
$MODULE
$ENTRY
$SAVE
$RETURN
$MODEND
END

Required Macros:

$HASPEQU (required for standard JES2 equates)
$HCT (required by $SAVE / $RETURN)
$MIT (required by $MODULE)
$PCE (required by $MODULE)

JES2 User or Subtask Task Environment Exits

6
8 - 9
12

Assuming you minimally code the following for each exit.

```
START
$MODULE
$ENTRY
STM R14,R12,R12(R13)
LM R14,R12,R12(R13)
$MODEND
END
```

Required Macros:

$HASPEQU (required for standard JES2 equates)
$MIT (required by $MODULE)

Functional Subsystem Address Space Environment Exits

23
25

Assuming you minimally code the following for each exit.

```
START
$MODULE
$ENTRY
$SAVE
$RETURN
$MODEND
END
```

Required Macros:

$HASPEQU (required for standard JES2 equates)
$HFCT (required to support $SAVE / $RETURN)
ETD (required to support $HFCT)
FSIP (required to support $HFCT)

Appendix C: JES2 Macro List

Appendix C. JES2 Macro List

Macro	Title	Use
$$POST	Post a JES2 Event Complete from Another Task	Post specific JES2 processors or resources
$$WTO	JES2 Subtask Write to Operator	Initiate the display of an operator message from a JES2 subtask or at JES2 initialization termination
$$WTOR	JES2 Subtask Write to Operator with Reply	Initiate the display of an operator message, requiring a reply, from a JES2 subtask
$#ADD	Add a Work/Characteristics JOE Pair to the JOT	Add a JOE to the appropriate JOT queue and add the characteristics JOE to the JOE queue
$#ALCHK	Obtain a Spool Record for Output Checkpointing	Obtain a spool record for output checkpointing
$#BLD	Format JOEs	Format a pair of work and characteristic JOEs in the provided work area
$#CAN	Cancel All Work Items Not Currently Being Processed for a Specific Job	Remove from the JOT all available work items for a job
$#CHK	Process Print/Punch Checkpoint Spool I/O	Process print/punch checkpoint spool I/O
$#GET	Search the JOT Class Queues for an Output Element that Matches the Requesting Specification	Search the JOT for output work
$#JOE	Find and Validate Queue	Cause the output service processors to generate the address for the head of specific queues
$#MOD	Move a Work JOE from One Queue to Another in the JOT	Remove a work JOE from the queue it is on and place it on the proper queue as determined by its routing, SYSOUT class, or offload status

Figure C-1 (Part 1 of 8). JES2 Macro List

Macro	Title	Use
$#PDBCAN	Cancel a JOE's Nonheld Data Sets	Mark a JOE's nonheld data sets for subsequent gathering for printing
$#PDBLOC	Find a PDDB by Data Set Key	Locate a PDDB using a specific data set key as the search argument
$#POST	Post Output Device Processors	Post device processors that are waiting for work associated with specific output devices
$#PUT	Return an Unfinished JOE to the JOT for Later Processing	Return a JOE to the JOT for later processing
$#REM	Remove a Work Characteristics JOE Pair from the JOT	Remove a work and characteristics JOE pair from the JOT after the output has been completed
$ACTIVE	Specify Processor is Active	Indicate to JES2 that the associated processor is performing work for the JES2 main task
$ALLOC	Allocate a Unit Record Device	Allocate a device to JES2
$AMODE	Set the Addressing Mode	Set the appropriate addressing mode for either MVS/370 or MVS/XA
$BFRBLD	Construct a JES2 Buffer Prefix	Construct an IOB or RPL in the form of a JES2 buffer
$BLDQC	Call the Quick Cell Build/Extend Routine	Call the quick cell build/extend routine to build or extend a quick cell pool
$BLDTGB	Queue TGBs to the HASPOOL Processor	Build TGBs and queue them off of the $SPOOLQ in the HCT
$BUFCK	Check Buffer I/O	Verify the completion of buffer I/O that was initiated by $BUFIO
$BUFIO	Read or Write a Buffer	Read or write a buffer from or to spool
$CALL	Call a Subroutine from JES2	Call a subroutine from a JES2 module
$CKPT	Schedule the Checkpoint of an Element	Schedule the checkpoint of an element in a JES2 checkpoint table that has been altered
$COUNT	Count Selected Occurrences	Increase a counter to determine the number of times a particular piece of code is entered

Figure C-1 (Part 2 of 8). JES2 Macro List

Macro	Title	Use
$CWTO	Command Processor Write to Operator	Cause a write to operator
$DCBDYN	Call the Dynamic DCB Service Routine	Provide the interface to the $DCBDYN service to attach or detach a JES2 data control block and/or data extent block.
$DCTDYN	Call the Dynamic DCT Service Routine	Provide the interface to the $DCTDYN service to attach or find a JES2 device control table.
$DCTTAB	Map DCT Table Entries	Map and generate DCT table entries
$DEST	Convert Symbolic Destination to a Route Code	Convert a symbolic destination to a binary route code
$DISTERR	Indicate Disastrous Error	Indicate that a disastrous error occurred
$DOM	Delete Operator Message	Delete an operator message
$DORMANT	Specify Processor is Inactive	Indicate to the JES2 dispatcher that the associated JES2 processor has completed job or task processing
$DTEDYN	Call the Dynamic DTE Service Routine	Call the daughter task element (DTE) dynamic service routine to provide ATTACH/DETACH services for user-defined subtasks.
$DTETAB	Build the DTE tables	Build the DTE tables
$ENTRY	Provide Entry to JES2 Processor or Function	Identify and document an entry point to a JES2 processor function or installation exit routine
$ERROR	Indicate Catastrophic Error	Indicate that a catastrophic error has occurred and to abend the JES2 main task
$ESTAE	JES2 Error Recovery Environment	Generate the calling sequence to a recovery service routine in HASPNUC
$EXCP	Execute JES2 Channel Program	Initiate JES2 input/output activity
$EXIT	Provide Exit Point	Establish an exit point
$EXTP	Initiate Remote Terminal Input/Output Operation	Initiate a remote terminal or network device input/output action or operation
$FRECEL	Free Common Storage Cell	Return a storage area to the cell pool

Figure C-1 (Part 3 of 8). JES2 Macro List

Macro	Title	Use
$FRECMB	Free a Console Message Buffer	Return a console message buffer to the free queue
$FREEBUF	Return a JES2 Buffer to the JES2 Buffer Pool	Return a JES2 buffer to the JES2 buffer pool
$FRELOK	Free the MVS CMS Lock, LOCAL, or JES2 Job Lock	Free the CMS lock, LOCAL, or JES2 job lock obtained by $GETLOK
$FREMAIN	Branch-Entry FREEMAIN Services	Free an area of storage obtained via MVS GETMAIN services.
$FREQC	Free Quick Cell	Free the storage obtained for the quick cells
$FREUCBS	Free UCB Parameter List Storage	Free UCB parameter list storage
$FREUNIT	Release a Unit Device Control Table	Release a DCT
$FSILINK	Link the Functional Subsystem Interface	Provide the base register setup for the functional subsystem
$GETABLE	Get HASP/USER Table Entries	Return table entries for the HASP/USER table pairs
$GETASCB	Retrieve the Primary, Secondary, or Home ASCB	Access the ASID to place the primary, secondary, or home ASCB address into a register
$GETBLK	Get a Storage Cell from a Free Cell Pool	Obtain a specified number of predefined storage cells from one of several cell pools
$GETBUF	Acquire a Buffer from a JES2 Buffer Pool	Obtain a buffer from a buffer pool and return the address
$GETCEL	Acquire a Common Storage Area Cell	Obtain a storage area for use in communicating information between the JES2 main task and HASPSSSM
$GETCMB	Get Console Message Buffers	Obtain console message buffers from the free queue
$GETLOK	Acquire the MVS CMS, LOCAL, or JES2 Job Lock	Obtain the MVS CMS, LOCAL, or JES2 job lock as required
$GETMAIN	Branch-Entry GETMAIN Services	Obtain or free an area of storage from MVS GETMAIN/FREEMAIN services
$GETQC	Call the Quick Cell Get Routine	Obtain storage cells from the previously built quick cell pool
$GETSMFB	Acquire a JES2 SMF Buffer from the JES2 SMF Buffer Pool	Obtain a buffer from the JES2 SMF buffer pool

Figure C-1 (Part 4 of 8). JES2 Macro List

Macro	Title	Use
$GETUCBS	Obtain a UCB Address	Obtain UCB addresses
$GETUNIT	Acquire a Unit Device Control Table (DCT)	Assign a unit device control table to a specific device
$GETWORK	Obtain a Work Area	Obtain a work area chained off the current PCE
$IOERROR	Log Input/Output Error	Log an input/output error on the operator's console
$JCAN	Cancel Job	Prepare a specific job for current processing cancellation
$JCTIO	Call the JCTIOR Routine	Call the JCTIOR routine in HASPNUC to read or write a JCT from spool.
$MID	Assign JES2 Message Identification	Set the global variable symbol &MID to an EBCDIC character string
$MODCHK	Call the MODCHK Verification Routine	Verify module residence, level, and name.
$MODEND	Generate End of Module	Generate the MIT entry table to fill the mask field in the MIT
$MODULE	Provide a Module Information Table	Generate a module information table to establish assembler global variable values, DSECTs, and EQUs
$MSG	Write to Operator Message Area	Generate a message text area to be used by $WTO
$NHDGET	Get Network Header Section	Search through the network job header, job trailer, or data set header to locate a specified section of a control block.
$PATCHSP	Generate Patch Space	Generate a specified number of bytes of patch space
$PBLOCK	Block Letter Services	Create block letters from the JCT and PCE for separator pages
$PCEDYN	Attach or Delete a JES2 PCE	Provide interface to those services that perform generating and deleting of JES2 processors
$PCETAB	Generate or Map PCE Table Entries	Map and generate PCE table entries
$PGSRVC	Perform a Virtual Page Service	Page-fix, page-free, or page-release an area of JES2 storage
$POST	Post a JES2 Event Complete	Indicate that a JES2 resource should be posted using the HCT

Figure C-1 (Part 5 of 8). JES2 Macro List

Macro	Title	Use
$POSTQ	Quick Post Facility	Quick post an ECB or issue an MVS POST
$PRPUT	Create Separator Pages	Create user-defined separator pages
$PURGE	Return Direct-Access Space	Return the direct access space that was allocated for a job or data set
$QADD	Add Job Queue Element to the JES2 Job Queue	Add an element to the JES2 job queue and place it into a specified logical queue
$QCTGEN	Define a Quick Cell Control Table	Define the attribute of a quick cell control table
$QGET	Obtain Job Queue Element from the JES2 Job Queue	Obtain a job queue element from a specified logical queue of the JES2 job queue
$QJIX	Add a Job Queue Element to the Job Index Table (JIX)	Get a job number for a specified JQE and add the JQE to the JIX
$QLOC	Locate Job Queue Element for Specific Job	Locate the JQE associated with the specified job
$QMOD	Modify Job Queue Element in the JES2 Job Queue	Remove a modified JQE from the JES2 job queue and return it to the specified logical queue
$QPUT	Return Job Queue Element to the JES2 Job Queue	Return a JQE to the JES2 job queue and place it in the specified logical queue
$QREM	Remove Job Queue Element from the JES2 Job Queue	Remove a specified JQE from the JES2 job queue
$QSUSE	Synchronize to Use Shared Queues	Update the checkpoint records to access any checkpoint information
$QUESMFB	Queue a JES2 Buffer on the Busy Queue	Place a JES2 SMF buffer address on the busy queue and MVS POST the HASPACCT subtask
$RELEASE	Release the Checkpoint Reserve	Release the checkpoint through MVS RELEASE services
$RESERVE	Request Checkpoint Reserve	Reserve a checkpoint through MVS RESERVE services
$RESTORE	Restore Registers from the Current Processor Save Area	Restore registers from the current processor's save area
$RETBLK	Return a Storage Cell to a Free Cell Pool	Return predefined storage cells to previously defined free cell pools
$RETSAVE	Return a JES2 Save Area	Return the current JES2 save area to the JES2 free pool of save areas

Figure C-1 (Part 6 of 8). JES2 Macro List

Macro	Title	Use
$RETURN	Restore Registers, Free the JES2 Save Area, and Return to the Caller	Restore the caller's registers saved in the current processor save area and return the save area
$RETWORK	Return a Work Area	Return a work area obtained with the $GETWORK macro
$SAVE	Obtain JES2 Save Area and Save Registers	Obtain a register save area from the JES2 save area pool and save registers 14, 15, and 0 through 12.
$SCAN	Scan Initialization Parameters and Operator Commands	Scan JES2 parameters and option statements and operator commands and place values in or create control blocks, or display keyword values
$SCANB	Backup Storage for a Scan	Backup a copy of a storage before it is changed by $SCAN
$SCANCOM	Call the $SCAN Facility Comment Service Routine	Use the $SCAN comment service facility to skip over imbedded comments in both initialization statements and commands.
$SCAND	Call the $SCAN Facility Display Service Routines	Add text to a display line created by the $SCAN macro
$SCANTAB	Scan Table	Create scan tables used by $SCAN to define syntax for initialization statements and operator commands
$SDUMP	Take an SDUMP of Storage	Dump the storage of selected address spaces
$SEPPDIR	Create a User Peripheral Information Record (PDIR)	Send a PDIR to remote SNA devices that require a PDIR to use their spooling capabilities
$SETRP	Set Recovery Processing Options	Indicate where control is passed following a $RETRY
$STCK	Call the $STCK Service Routine	Call the $STCK (store clock) service routine
$STIMER	Set Interval Timer	Set a time interval for the programmed interval timer
$STMTLOG	Log an Initialization Statement	Log initialization statements and related diagnostics
$STORE	Store Registers in the Current Processor Save Area	Store registers in the current PSA created by $SAVE
$TIDTAB	Generate the Trace ID Table CSECT	Generate and map trace ID (TID) table entries
$TGMSET	Call the $TGMSET Service Routine	Generate the calling sequence to $TGMSET

Figure C-1 (Part 7 of 8). JES2 Macro List

Macro	Title	Use
$TRACE	Trace a JES2 Activity	Allocate a JES2 trace table entry (TTE) in a active trace table
$TRACK	Acquire a Direct Access Track Address	Obtain a track address on a JES2 spool volume
$TTIMER	Test Interval Timer	Obtain the time remaining in the associated time interval set by $STIMER
$VERIFY	Call the $VERIFY Service Routine	Call the $VERIFY service routine to validate spool-resident control blocks
$VERTAB	Build the Inline Verification Tables	Build the inline verification tables used by the $VERIFY service routine
$VFL	Variable Field Length Instruction Operation	Provide storage-to-storage operations when the field length exceeds 256 bytes or there are no associated assembler instructions
$WAIT	Wait for a JES2 Event	Place the associated processor in a JES2 wait state and specify the specific event
$WSSETUP	Set Values Required for Work Selection	Set specific values that are required to support work selection
$WSTAB	Map and Generate the Work Selection Table Entries	Map and generate the work selection table entries
$WTO	JES2 Write to Operator	Initiate the display of a message for the operator on a specified device
$XMPOST	POST Task in Another Address Space	POST a task in another address space

Figure C-1 (Part 8 of 8). JES2 Macro List

Appendix D: JES2 Exit Usage Limitations

Appendix D. JES2 Exit Usage Limitations

The following table notes those instances when reader and converter exits (Exits 2, 3, 4, 6, and 20) are invoked or not invoked. Be certain to consider this information when attempting to implement these exits.

Source of Job	Exits Taken For: Input Service 2	3	4	20	Converter 6
Job from local reader	Y	Y[1]	Y	Y	Y
Job from remote reader	Y	Y[1]	Y	Y	Y
SYSOUT data from spool offload	N	N	N	N	N
TSO session logon (TSU)	Y	Y[1]	Y	Y	Y
TSO submitted job	Y	Y[1]	Y	Y	Y
Started task	Y	Y[1]	Y	Y	Y
Job with /*ROUTE XEQ	Y	Y[1]	Y	Y	N
Job following /*XMIT JECL	N	N	N	N	N
Job from NJE job receiver:					
Job for this node	Y	Y[1]	Y	Y	Y
Store and forward	N	N	N	Y	N
Job from NJE SYSOUT receiver:					
Job for this node	N	N	N	N	N
Store and forward	N	N	N	N	N
Job internally generated by JES2 (SYSLOG-RMTMSG)	N	N	N	N	N
Spool offload job receiver[2]	Y	Y[1]	Y	Y	Y
Spool offload SYSOUT receiver	N	N	N	N	N
XBM invocation	Y	Y[1]	Y	Y	Y
XBM joblet	Y	Y[1]	N	N	N
Special Case JCL and JECL					
JCL from cataloged procedure	-	-	N	-	Y
//* COMMENT cards	-	-	N	-	-
/*PRIORITY statements	-	-	Y	-	-
/*$command statements[3]	-	-	-	Y	-
/* end of SYSIN data	-	-	N	-	-
// null statements	-	-	N	-	-
Generated DD* statement	-	-	Y	-	-
/* with invalid verb	-	-	Y	N	-
// with invalid verb	-	-	Y	Y	-
/*EOF internal reader	-	-	N	-	-
/*DEL internal reader[4]	-	-	N	N	N
/*PURGE internal reader[4]	-	-	N	N	N
/*SCAN internal reader	-	-	N	-	N

Y Exit is invoked
N Exit is not invoked
- Not applicable

Notes:

1 Exit 3 is taken only if ACCTFLD=REQUIRED or OPTIONAL is specified on the JOBDEF initialization statement. Exit 3 will be taken even if there is no accounting information provided on the JOB statement.

2 This may be the second (or more) pass through these exits for this job.

3 Commands must be outside of a job; they will invoke Exit 4 but will not have a JCT (R10=0).

4 DEL and PURGE INTRDR control statements causes a job to bypass Exit 20.

Index

Special Characters

A

B

C

D

E

F

G

H

K

L

M

N

Q

R

S

T

U

V

W

X

System Programming
Library: JES2
User Modifications
and Macros

READER'S
COMMENT
FORM

LC23-0067-3

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.)

System Programming Library: JES2 User Modifications and Macros

LC23-0067-3

S370-36

Reader's Comment Form

Cut or Fold Along Line

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 390
Poughkeepsie, New York 12602

Fold and tape | Please Do Not Staple | Fold and tape

Printed in U.S.A.

IBM®

LC23-0067-3

S370-36

Printed in U.S.A.

LC23-0067-03