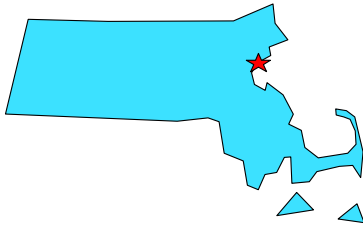


JES2 Exits Overview

Job Related Exits

Exit Environments



SHARE 105, Session 2662

Thursday, August 25, 2005

Permission is granted to SHARE Inc. to publish this presentation in the SHARE proceedings. IBM retains its right to distribute copies of this presentation to whomever it chooses.

Chip Wood
JES2 Design/Development/Service
Poughkeepsie, NY



chipwood@us.ibm.com

Agenda



- Planning and researching your exit thoroughly
- Exit points
- Coding your exit
- Loading and enabling your exit
- Debugging your exit

Disclaimers



- From JES2 Installation Exits, page 1:

Caution!

Defining exits and writing installation exit routines is intended to be accomplished by experienced system programmers; the reader is assumed to have knowledge of JES2.

If you want to customize JES2, IBM recommends that you use JES2 installation exits to accomplish the task. **IBM does not recommend or support alteration of JES2 source code.**

- From JES2 Diagnosis:

CAUTION: IBM does not recommend or support modifications to JES2 source code. If you assume the risk of modifying JES2, then also assure that your modifications do not impact JES2 serviceability using IPCS. Otherwise, LEVEL2 may not be able to read JES2 dumps for problems unrelated to the modifications.

Make sure you need it



- Consider the following:
 - Do you have time, ability, and experience:
 - ▶ To write the exit?
 - ▶ To \$upport the exit?
 - Is the cost involved worth it?
 - Are there alternatives?
 - ▶ Can you use standard features/automation?
 - ▶ Can you use a manual procedure?
 - ▶ Has someone else solved the same problem (vendor product, mods tape, ...)?
 - ▶ Can you use table pairs instead?
 - ▶ Can you write an application using a supported interface (e.g. Extended status, SAPI)?

Exits vs. inline mods



- Exits are typically easier to maintain
 - Defined interface
 - Separate module, no interference from service updates
 - **++HOLD** if service changes something we anticipate an exit using
- Exits have limitations
 - Fixed points in processing
 - If there's no exit where you need it
 - ▶ Consider placing \$EXIT inline rather than extensive mods inline
 - ▶ Start with \$EXIT 255 and work your way down

Research your exit



- Choose the correct exit(s) for your task
- For example:
 - Set job attributes at input time:
 - ▶ Use exit 2 to set default attributes which can be overridden later
 - ▶ Use exit 20 to force attributes
 - Set job attributes at converter time
 - ▶ Use exit 6 to scan C/I text
 - ▶ Use exit 44 to store data in the JQE
- Many tasks require a combination of exits
 - Sample exit HASXJECL has a good example.

Research your exit



- Sources of information
 - [JES2 Exits](#)
 - Other JES2 Manuals
 - WSC Technical Bulletins and Flashes
 - SHARE Presentations
 - ▶ [How to Write an Error-free Exit](#) - Bob Jenkins, Summer 1994
 - ▶ [JES2 Job Related Exits](#) - John Hutchinson, Summer 2000
 - ▶ [Introduction to JES2 Table Pairs](#) - Chip Wood, Session 2665, Summer 2001
 - JES2 sample exits from **SYS1.SHASSAMP**
 - Exits from a "mods tape"
 - JES2 source code

Research your exit

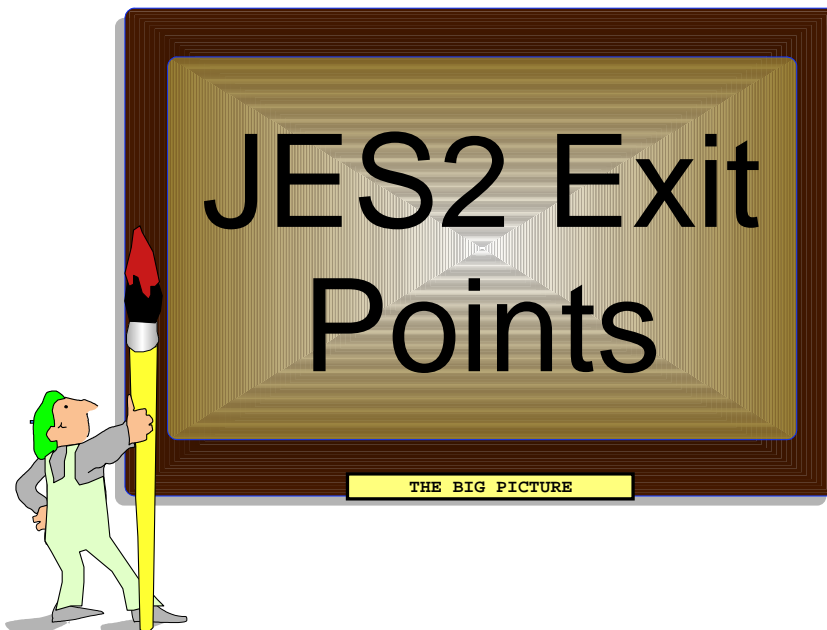


- JES2 Exits are system extensions
 - **Supervisor state, Key 0 or Key 1!**
 - Can do as much damage as an inline mod!
- Before attempting to write an exit, understand:
 - JES2 exit environments
 - JES2 and MVS serialization
 - What control blocks are available (and when)
 - What point in processing exit gets control
 - What fields are set when you get control
 - What fields you can change
 - Which exits can \$WAIT
 - Which macros can \$WAIT
 - Which macros can be used in which exits
 - etc.

Plan for Migration



- Consult [z/OS Migration](#)
 - Exits may need extensive rework
 - Exits may no longer be needed
- Use field names rather than hard-coded offsets
 - Offsets may change
 - If field usage changes, IBM will *usually* change the name (and your exit will not assemble)
- Consider: Can JES2 with and without your changes coexist in the same MAS?
 - With exits enabled?
 - With exits disabled (control block changes only)?
 - If they can't coexist, is a cold start required?
 - Does your data survive a spool offload/reload?

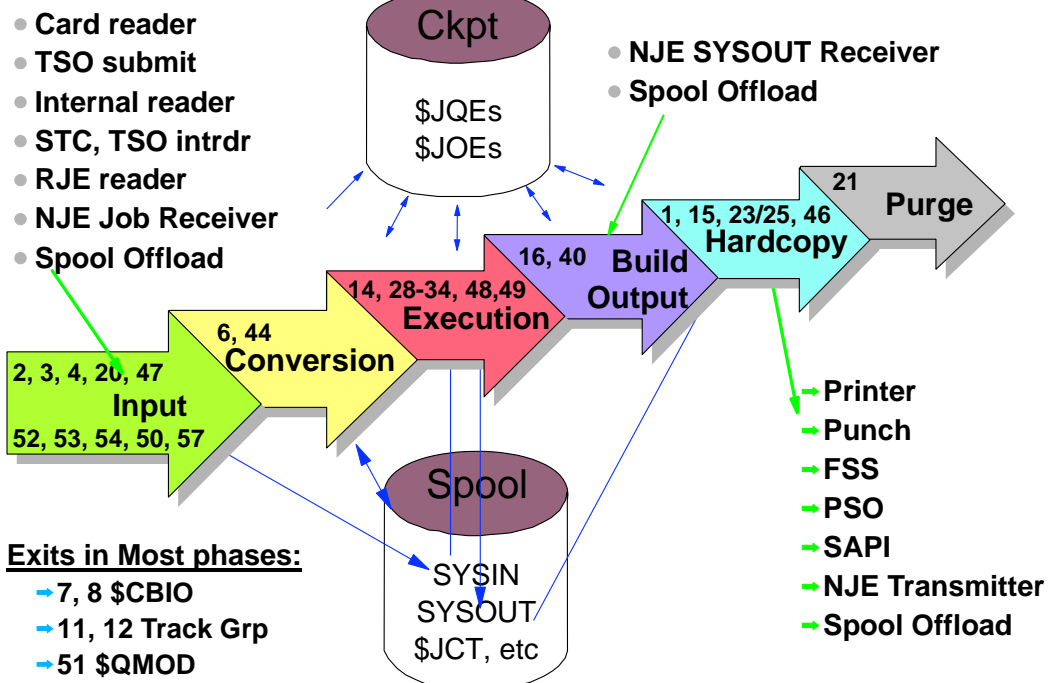


Exits and environments



Exit	Environment	Description	Exit	Environment	Description
0	JES2	Initialization	29	USER	End of Memory
1	JES2	Separator Page	30	USER	OPEN
2	JES2	Job Card	31	USER	Allocation
3	JES2	Accounting	32	USER	Job Select
4	JES2	JECL Card	33	USER	CLOSE
5	JES2	Command	34	USER	Unallocation
6	SUBTASK	Converter	35	USER	End of Task
7	JES2	\$CBIO	36	USER	Pre-SAF
8	USER	\$CBIO	37	USER	Post-SAF
9	USER	Excession	38	JES2	TSO Receive SAF
10	JES2	\$WTO	39	JES2	NSR SAF
11	JES2	\$TRACK	40	JES2	Modify SYSOUT
12	USER	\$STRAK	41	USER	Generic Grouping
13	USER	Netmail (obsolete)	42	USER	Notify SSI
14	JES2	\$QGET	43	USER	APPC
15	JES2	DS Separator	44	JES2	Converter
16	JES2	Notify	45	USER	Pre-SJF
17	JES2	BSC Signon	46	JES2	NJE Hdr XMIT
18	JES2	SNA Logon	47	JES2	NJE Hdr Recv
19	JES2	Init statement	48	USER	Late Unallocation
20	JES2	End of input	49	JES2	QGOT
21	JES2	SMF	50	USER	End of Input
22	JES2	Cancel/Status	51	JES2	QMODO
23	FSS	JSPA Separator	52	USER	Job Card
24	JES2	Post-initialization	53	USER	Accounting
25	FSS	JCT I/O	54	USER	JECL Card
26	JES2	Termination	55	USER	NSR SAF
27	JES2	PCE Attach	56	USER	NJE Hdr XMIT
28	USER	Job Termination	57	USER	NJE Hdr Recv
29	USER	End of Memory			

The big picture



Job Input Processing



HASPRDR	Exit Seq.	Sample Exits
Get NJH {if NJE}	47	HASX47A, HASXJECL
Read Job Stmt	2	HASX02A
\$TRACK	11	
\$CBIO JCT/IOT	7	HASX07A
Accounting Field	3	HASX03A
JCL and JECL	4	HASX04A, HASXJECL
Get NJT {if NJE}	47	
SAF VERIFYX plus 6 system DS	36/37	HASX36A, HASX37A
End of INPUT	20	HASXDYNT
\$CBIO JCT/IOT	7	
\$QMOD		

= Exit may be called multiple times



Job Input Processing - z/OS 1.7

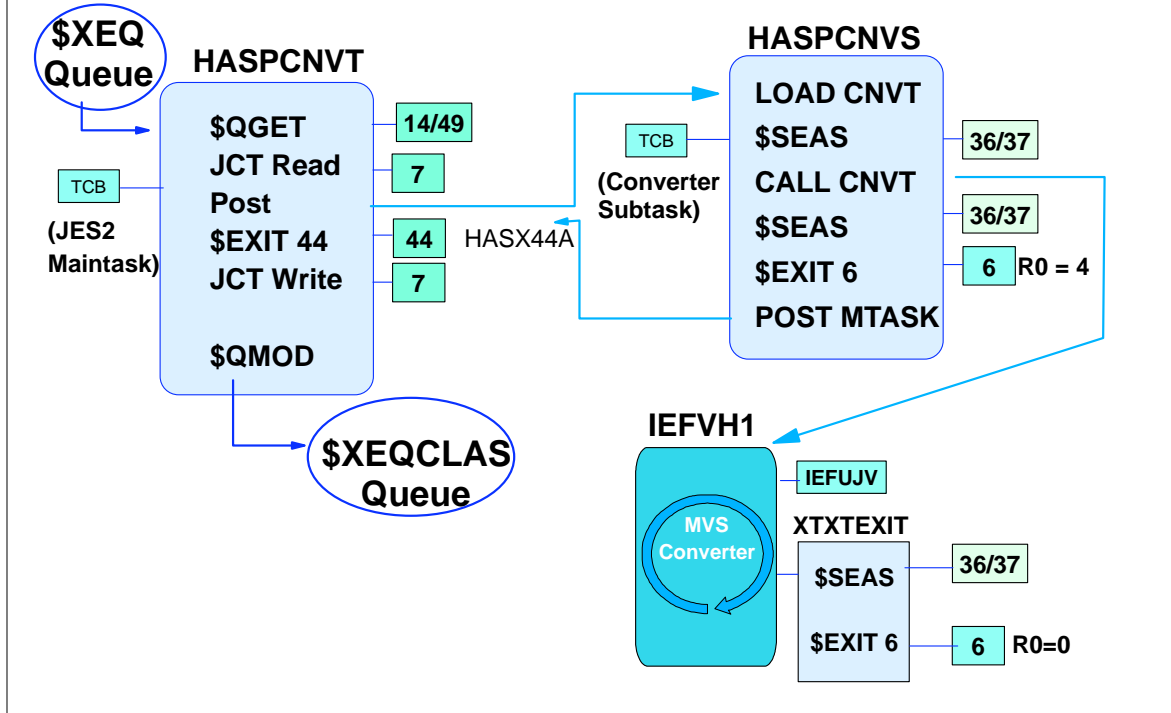


HASCSRIP	Exit Seq.	Sample Exits
Get NJH {if NJE}	57	HASX57A, HASXJECL
Read Job Stmt	52	HASX52A
\$TRACK	12	
\$CBIO JCT/IOT	8	
Accounting Field	53	HASX53A
JCL and JECL	54	HASX54A, HASXJECL
Get NJT {if NJE}	57	
SAF VERIFYX plus 6 system DS	36/37	HASX36A, HASX37A
End of INPUT	50	HASXDYNT
\$CBIO JCT/IOT	8	
\$QMOD	51	

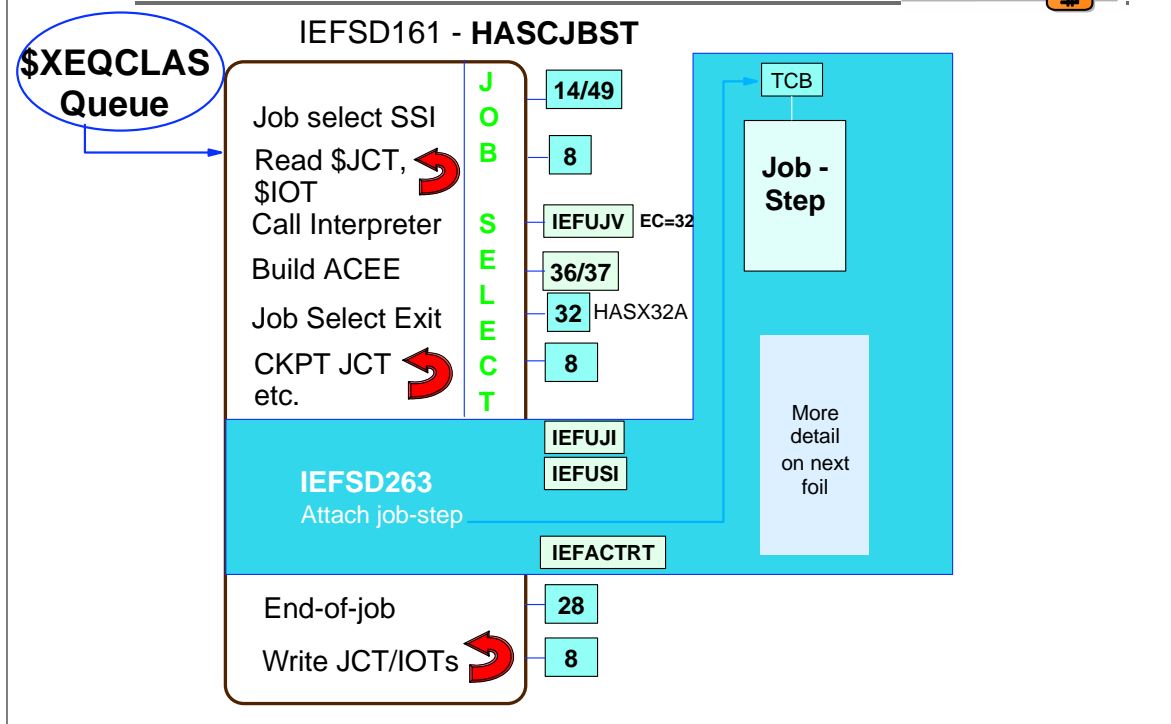
= Exit may be called multiple times



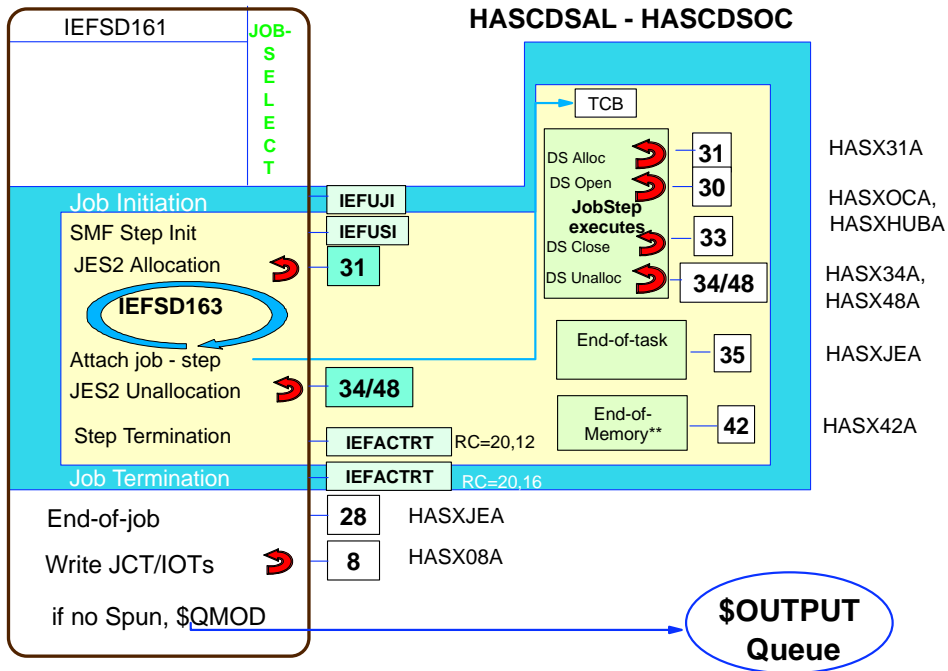
Conversion Processing



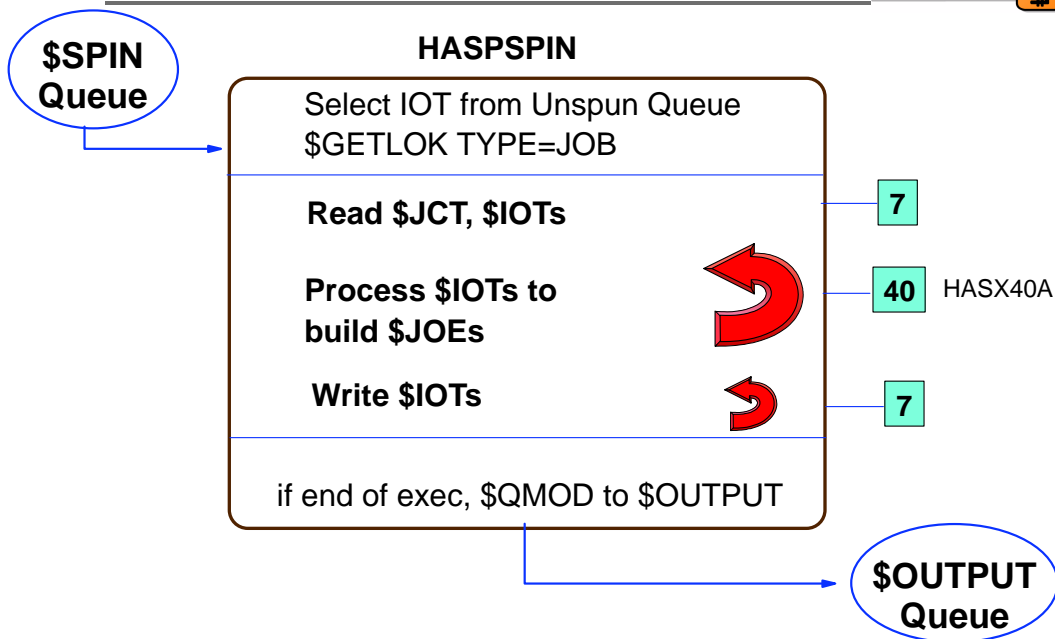
Execution, Part 1

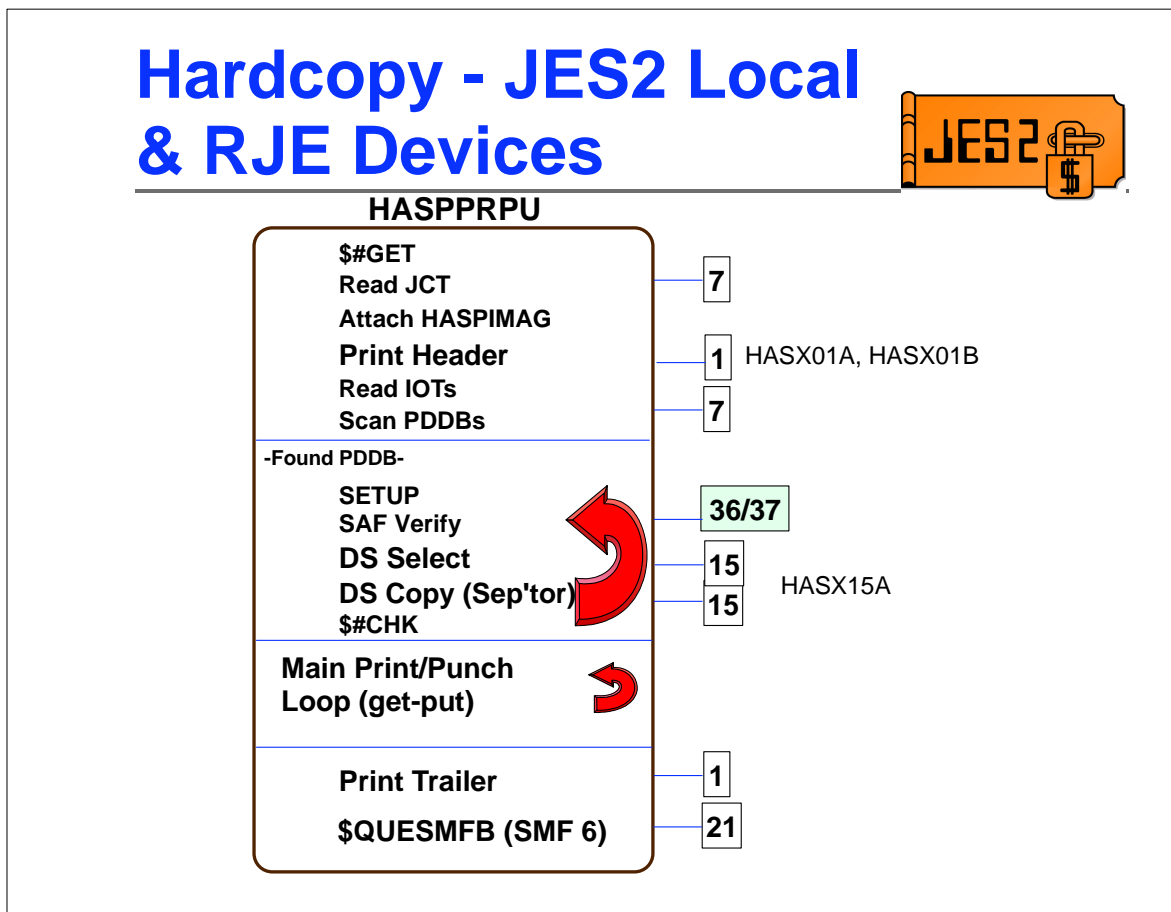
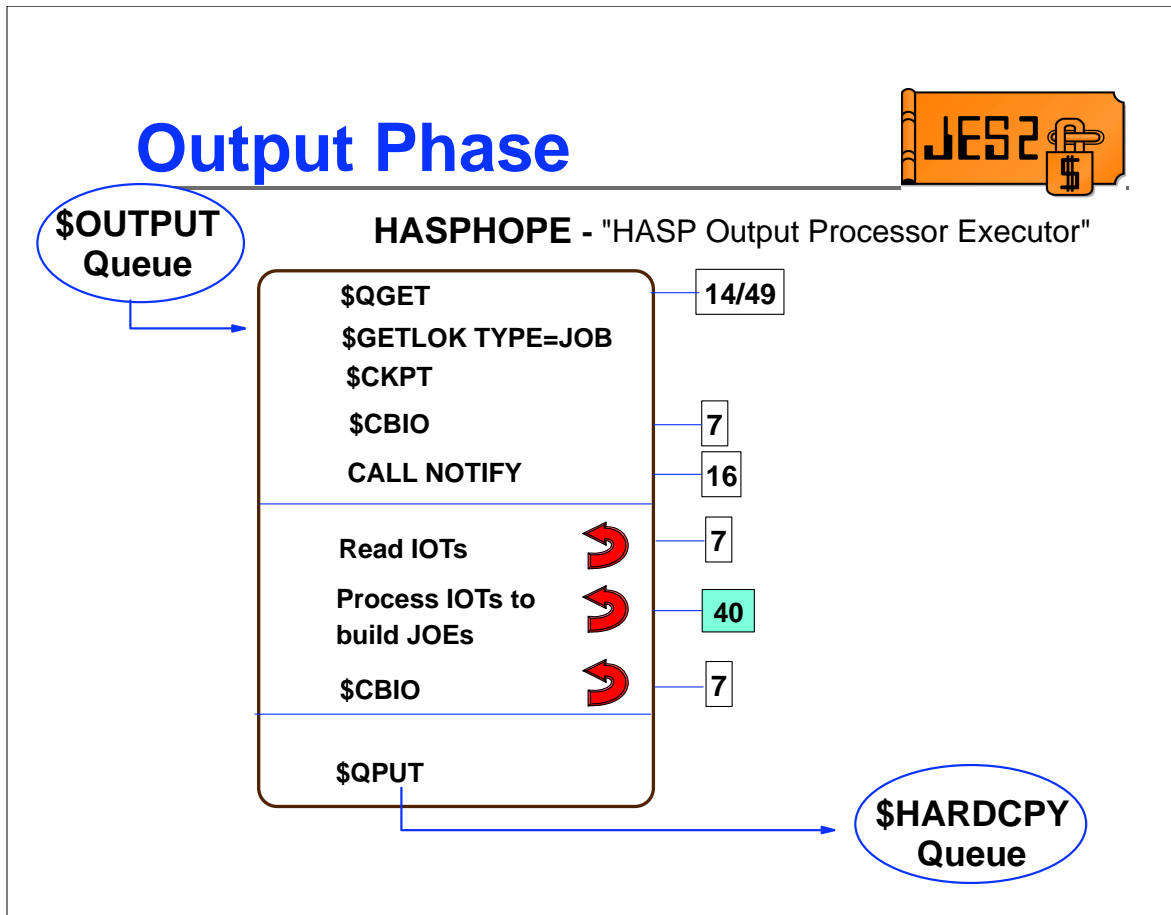


Execution, Part 2



Output Spin Phase





Hardcopy - FSS Devices (PSF printers)



PSF Addr.
Space

TCB

HASPFSSM

```

FSMGETDS
  Scan JIB queue
  (look for eligible JIB)
$CBIO (Read JCT)
  Initialize JSPA
$CBIO (read IOT)

$CALL PRTAUTH (JNEWS)
  More JSPA initl.

$CALL FINDP (PDDB)
$CALL PRTAUTH
$CALL DSOPEN (OPEN)
$CALL SWBRD (Read SWB)
GET (Initiate DS Input)
$CALL FINDP
  (NEXT PDDB)
$RETURN (to FSS)
    
```



- 8
- 25
- 8
- 36/37
- 23 HASX23A
- 36/37

PSF Exits:
APSUX01-APSUX07

NJE & Spool Offload Transmitters



HASPNST

```

Initialization
$#GET
$CBIO JCT/IOT
Xmit NJE JOB Header
-----
$SEAS Auth Chk
Xmit NJE DS Hdr
Xmit Data
-----
Xmit NJE JOB Trailer
$#REM
SAF Verify
$QUESMFB - write SMF
  type 24 or 57 record
    
```

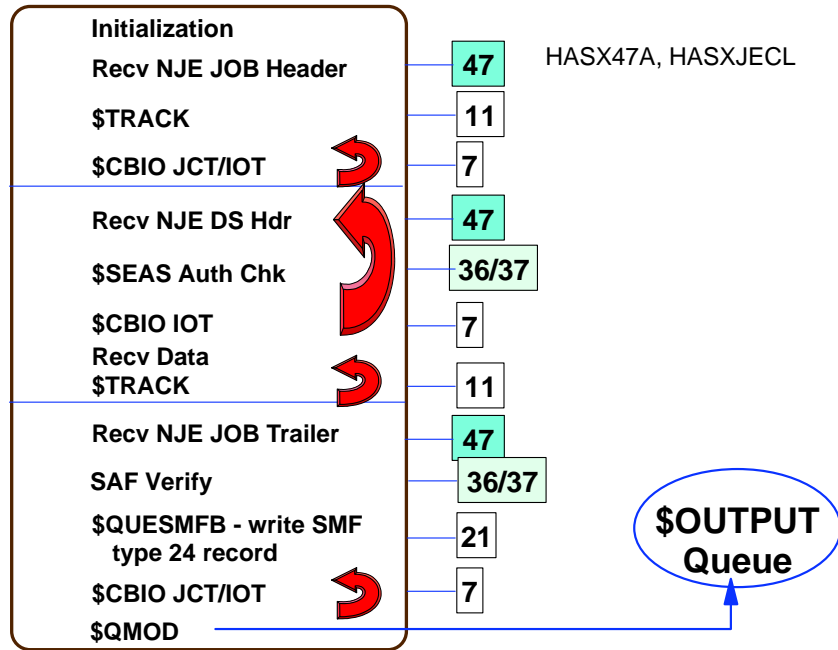


- 7
- 46 HASX46A, HASXJECL
- 36/37
- 46
- 46
- 46
- 36/37
- 21

NJE & Spool Offload Receivers



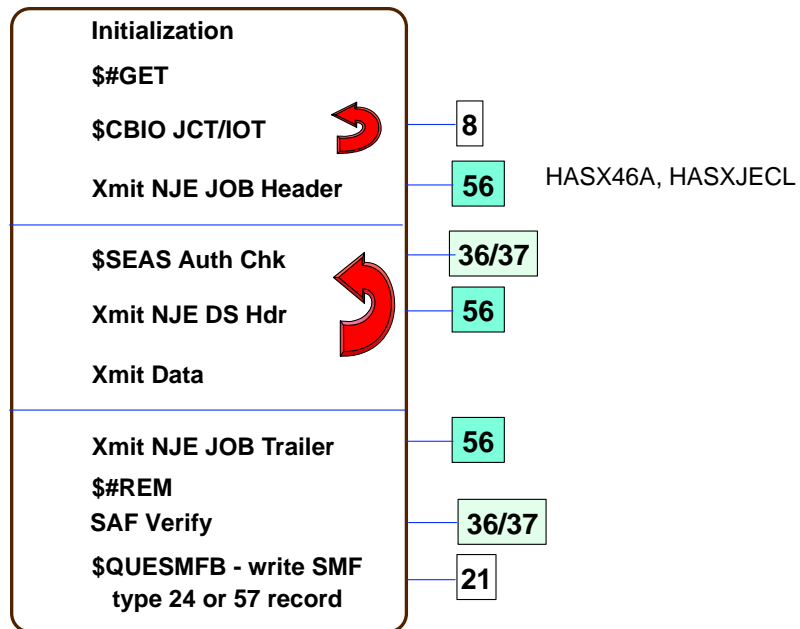
HASPNSR



NJE Transmitters - z/OS 1.7 (NJE/TCP)



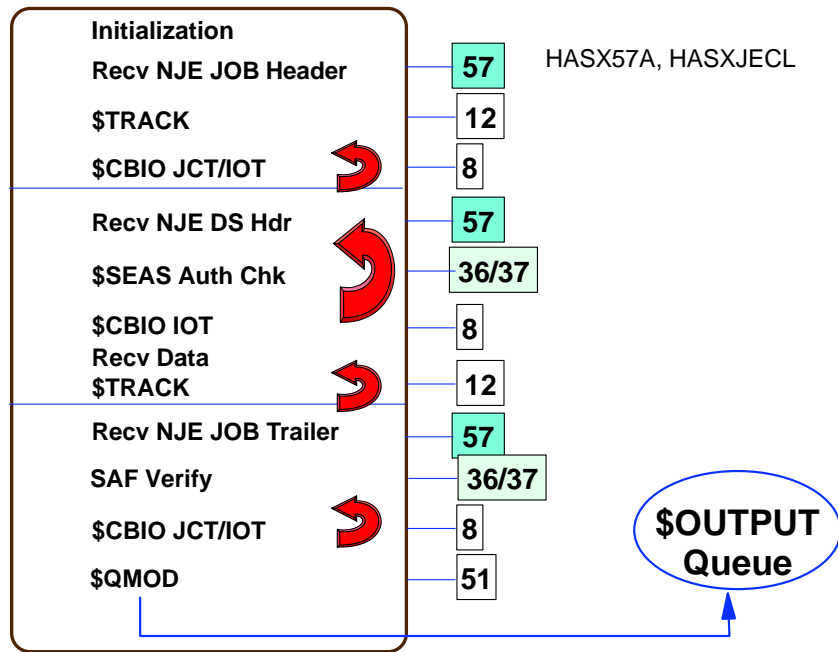
HASCNJST



NJE Receivers - z/OS 1.7 (NJE/TCP)



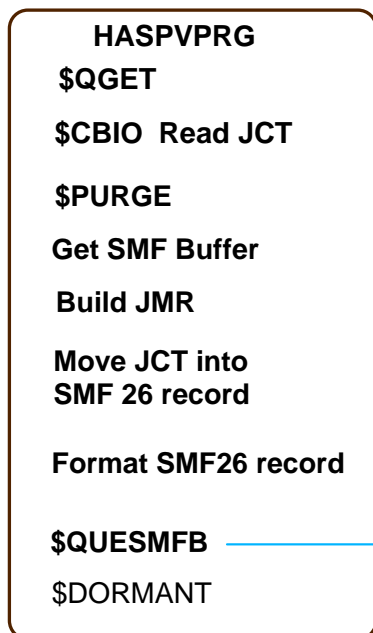
HASCNJSR



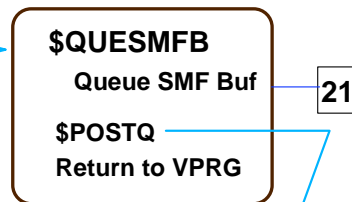
Purge Phase



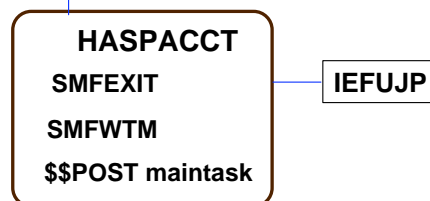
HASPTRAK



HASPNUC



HASPMISC

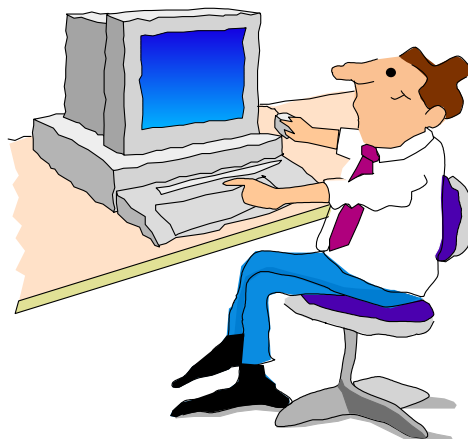


Job-related Control Blocks



- **Checkpointed Control Blocks**
 - **\$JQE** - Job Queue Elements - access w/\$DOGJQE
 - **\$JOE** - Job Output Elements - access w/\$#GET, \$#....
- **Spool Resident Control Blocks**
 - **\$JCT** - Job Control Table (Addr.usually passed in \$XPL)
 - **\$PDDB** - Peripheral Data Definition Block (1/Data Set)
 - **\$IOT** - I/O Table (contains PDDBs for a Job)
 - NJE Job & Data Set hdrs (only created if transmitted)
- Generally passed to exits in parameter list (XPL)

Coding Exits



Basic Constructs



- Code JES2 exits in assembler
- Some basic constructs are common to all exits

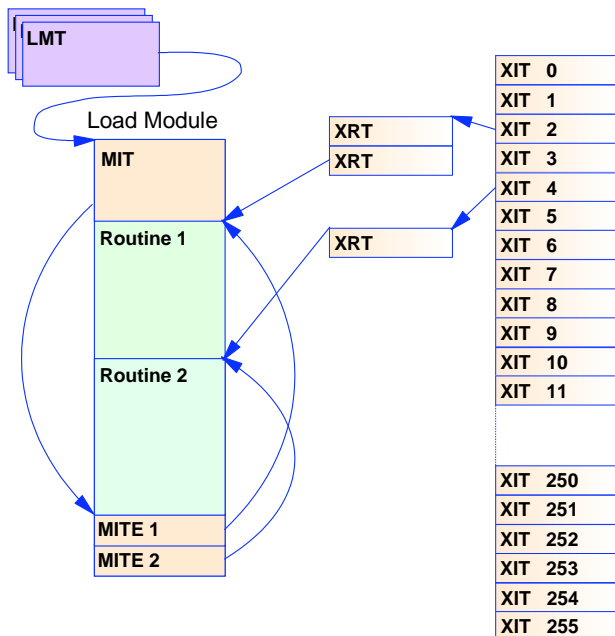
```

COPY $HASPGBL
$MODULE

ROUTINE1 $ENTRY
$SAVE
$ESTAE
...
$RETURN

$MODEND
    
```

JES2 Exit Data Structures



MIT

- Module Information Table
- Created by \$MODULE

MITE

- Module Information Table Entry
- Created by \$MODEND
- One MITE per \$ENTRY point

LMT

- Load Module Table
- Created by LOAD initialization statement
- Pointer and attributes of load module

XIT

- Exit Information Table
- Information about exit

XRT

- Exit Routine Table
- List of routines called by exit
 - Name and Address

\$HASPGBL



■ COPY \$HASPGBL

- Includes and initializes assembler variables that are required by later JES2 macros

- **&J2VERSN SETC 'z/OS 1.5'**
- **&J2PLVL SETA 35**

```

        AIF (&J2PLVL LT 35).SKIPL
        <code that only runs on JES2 z/OS 1.5 and up>
        .SKIPL ANOP

```

- **&J2SLVL SETA 0**

- Includes assembler GBLx statements for other variables that may be referenced in open code

- **GBLC &ANVIRON**
- **GBLC &J2SECTN**

\$MODULE



■ \$MODULE:

- Defines what JES2 environment the exit will run in
 - Affects register conventions
 - Affects what you're allowed to do
- Defines what mappings are to be included
 - Includes all JES2 mapping macros
 - Includes many MVS mapping macros
 - Automatically resolves mapping dependencies
- Defines RMODE, reentrancy
- Defines default print options and TITLE
- Creates MIT (Module Information Table)
 - Contains information validated when module is **LOAD**ed

Environments



- **\$MODULE ENVIRON=** Keyword
 - Assembly environment (**&ANVIRON**)
- **If you get this wrong, the exit will not work!!**
- Helps catch coding errors
 - MNOTE from unsupported macro services (\$CKPT, etc).
- **ENVIRON=** should be one of:
 - **JES2** - JES2 main task
 - **SUBTASK** - JES2 subtask
 - **USER** - JES2 USER environment
 - **FSS** - Functional subsystem environment

ENVIRON=JES2



- **ENVIRON=JES2** - JES2 Main Task
- JES2 address space, JES2 Main task
- **R11** = HCT address
- **R13** = PCE address
- JES2 Main Task Serialized
 - Only one PCE runs at a time
- Routine should reside in private storage
 - **LOAD(routine) STORAGE=PVT**
- Routine may reside in common storage
 - Why use up CSA unnecessarily?
 - Can't refresh routine on a hot start

ENVIRON=JES2



- Avoid MVS waits!
 - Use **\$WAIT** instead
 - **Note that some exits do not allow \$WAIT!**
 - Use JES2 equivalent services that do not MVS WAIT
 - **\$WTO**, etc.
 - If calling service which can MVS WAIT, consider using **\$SUBIT** to \$WAIT the PCE while a JES2 subtask does the MVS WAIT.
 - Exceptions - JES2 Initialization/Termination (Exits 0, 19, 24, 26)
 - JES2 dispatcher not enabled, so DON'T \$WAIT
 - **ENVIRON=(JES2,INIT)** can be used in 0, 19, 24
 - **ENVIRON=(JES2,TERM)** can be used in 26

ENVIRON=SUBTASK



- **ENVIRON=SUBTASK** - JES2 Subtask environment
- JES2 address space, non-main task
 - MVS WAIT can be done
 - \$WAIT not allowed
- **R11** = HCT
- **R13** = DTE
- Multi-tasking considerations apply!
 - Multiple subtasks can simultaneously be in subtask
 - Reentrancy is very important
- Routine should reside in private storage
- Routine may reside in common storage

ENVIRON=USER



- **ENVIRON=USER** - JES2 USER Environment
- Any address space, any task
- **R11** = HCCT
- **R13** = Available save area
- Multi-tasking considerations:
 - Multiple tasks in multiple address spaces may be in exit simultaneously
 - Reentrancy very important
- Routine **MUST** reside in common storage
 - **LOAD(routine) STORAGE=CSA**
 - **LOAD(routine) STORAGE=LPA**
- Exits may get control when JES2 is down!

ENVIRON=(USER,ANY)



- **ENVIRON=(USER,ANY)** - JES2 USER Environment
 - z/OS 1.7 only
- Any address space, any task
- **R11** = HCCT
- **R13** = Available save area
- Same as ENVIRON=USER except:
 - \$SAVE/\$RETURN services use JES2 main task logic
 - No BAKR's (so \$WAIT can be done if main task)
- Useful for routines that could be called in the user environment OR under main task
 - Detect run-time environment by checking PSAAOLD and PSATOLD for JES2 Main task

ENVIRON=FSS



- **ENVIRON=FSS** - Functional subsystem environment
- FSS address space
- **R11** = HFCT
- **R13** = Save area
- Routine **MUST** reside in common storage

\$ENTRY



- **\$ENTRY** - Generates entry point
 - Entry point defined (and MITE)
 - Eyecatcher ('**\$\$\$\$routine-name**')
 - Using on register(s) specified for **BASE=**
 - Optionally, sets base registers (**SETBASE=YES**)
 - Optionally, generates \$SAVE (**SAVE=YES**)
 - Trace \$SAVE via **SAVE=(YES,TRACE)**
 - Can specify **CSECT=YES** to put each entry point in a separate CSECT
 - Data isolation
- Alternatively, specify **ENTRIES=(routine, routine)** on **\$MODULE**
 - Generates ENTRY and MITE only

\$SAVE/\$RETURN



■ \$SAVE

- Saves registers on entry
- Different expansions, depending on environment!
 - \$HASP095 CODE=\$S02

■ \$RETURN

- Returns to caller
- RC= - specifies a return code
 - Return code can be coded explicitly (RC=0)
 - Return code can be in register (RC=(R15))

\$ENTRY



■ Example \$ENTRY expansion

```

MYRTN  $ENTRY BASE=(R12,R8),SAVE=(YES,TRACE)
+      ENTRY MYRTN                    <--- Entry Statement
+MYRTN DS 0D                          <--- Entry Point
+      USING MYRTN,R12,R8             <--- Using for BASE=
+      J $EE2185
+      DC C'$$$$'                    <--- Eyecatcher
+      DC CL8'MYRTN'
+$EE2185 STM R14,R12,12(R13) <--- $SAVE
+      L R15,$PADDR
+      L R15,P@GETSAVE-PADDR(R15)
+      BASR R14,R15
+      J $CAL2192
+      DC AL1($SAVTRC,0)
+      DC CL8'MYRTN'
+$CAL2192 DS 0H
+      LM R14,R12,12(R13)
+      LR R12,R15                    <--- Set base registers
+      LR R8,R12
+      AHI R8,4096

```

\$MODEND



- Defines end of module
- Generates MITEs for each \$ENTRY
 - Names and entry points of all routines defined via \$ENTRY
 - LOAD init statement can identify all routines defined in module for inclusion via EXITnn ROUTINES=
 - Entry points that LOAD uses

```

COPY $HASPGBL
MYMOD $MODULE ENVIRON=JES2
MYRTN $ENTRY BASE=(R12),SAVE=YES
$RETURN RC=0
...
$MODEND
+ ...
+ DC CL8 'MYRTN' <--- MITE for MYRTN
+ DC A(MYRTN),AL1(0,MITENVJ,0,0)
+ DC AL2(0),AL1(0,0)
+ ...

```

Packaging



- Package exits for one function together
 - For example, **HASXJECL**
 - ▶ \$EXIT 1 to print data on separator page
 - ▶ \$EXIT 4,54 to obtain separator page data from JCL
 - ▶ \$EXIT 46,47,56,57 to transmit separator page data across NJE
- Package unrelated exits separately
 - For example
 - ▶ \$EXIT 5 to create 2 separate commands
 - ▶ Different routine for each
 - ▶ Separate load modules

\$ENVIRON



- **\$ENVIRON** - changes the current assembly environment (&ANVIRON)
- For example
 - Need an exit 6 and an exit 44 to accomplish a function in the converter
 - Code like this:

```

$MODULE ENVIRON=JES2
EXIT44  $ENTRY
...
$ENVIRON SET,ENVIRON=SUBTASK
EXIT6   $ENTRY
...
$MODEND

```

- **BE CAREFUL!!!** Remember that if **ANY** routine is **ENVIRON=USER** or **ENVIRON=FSS**, it needs to be in common storage!!!

\$ENVIRON



- Use to set environment before calling a routine that expects another environment
 - Used by sample exits 2, 3, 4, 20, 46 and 47
 - For example:

```

$MODULE ENVIRON=JES2
EXIT20  $ENTRY
...
$ENVIRON PUSH,ENVIRON=(USER,ANY),SETR11=YES
$CALL  EXIT50
$ENVIRON POP,SETR11=YES
...
$RETURN
$ENVIRON SET,ENVIRON=(USER,ANY)
EXIT50  $ENTRY
...
$MODEND

```

- **SETR11=YES** - sets register 11 based on old and new environment

Extending data areas



- **"Unsupported"** way - modify mapping macro
 - IBM does not recommend expanding control blocks
 - Consider:
 - ▶ Effects of enabling and disabling exits
 - ▶ Effects on checkpointed or spool control blocks (**COLD** start implications!)
 - ▶ Which base JES2 modules require reassembly
 - Beware of using "reserved" fields
 - ▶ IBM may reclaim them later
 - ▶ There may be an intended purpose for reserved space in a particular place (e.g. job number)
 - Keep IPCS formatters in synch
 - May interfere with LEVEL2 resolution of unrelated problems for dumps sent to IBM

Extending data areas



- **"Supported"** ways
 - **\$UCT** control block
 - ▶ Obtain in exit 0, store address in \$UCT field
 - ▶ Can contain table pairs, pointers to other CBs
 - **\$USER_n, PCEUSER_n, DCTUSER_n, CCTUSER_n** fields
 - ▶ Can be used as pointers to your own area if not enough space
 - \$JCTX services (JCT)
 - ▶ **\$JCTXADD, \$JCTXGET, \$JCTXEXP, \$JCTXREM**
 - **\$BERTTABs** (JQE and CAT)
 - ▶ \$BERTTAB CBOFF=*
 - ▶ \$DOGBERT ACTION=GETOFFSET

Enabling your exit



- **LOADmod** initialization statement
 - Loads your load module
 - STEPLIB, LNKLST, or LPA
 - **LOADmod(name) STORAGE=**
 - **PVT** - Module is loaded into JES2 private storage
 - **CSA** - Module is loaded into common storage
 - **LPA** - Version of module in LPA used
 - ENVIRON=USER or FSS cannot go in private
- **\$D LOADMOD(name)** displays
 - **ADDRESS** and **LENGTH**
 - **ROUTINES** and **TABLES**
 - **STORAGE** and **RMODE**

Enabling your exit



- **EXIT(nnn)** statement
 - **ROUTINES=(routine-name,routine-name)**
 - Lists all routines to be called by exit
 - **ENABLE/DISABLE**
 - Whether exit is initially enabled or disabled
 - Default: ENABLE if you specified routines
- **\$TEXITnn,ENABLE** or **\$TEXITnn,DISABLE**
 - Enable/disable exits
 - Exit level only (not individual routines)
 - Consider implementing router exits to disable individual functions

Router exits



- Old way
 - **EXIT(5) ROUTINES=(rtn1,rtn2,rtn3)**
 - **\$TEXT5,DISABLE** disables all 3 routines
- New way
 - **EXIT(5) ROUTINES=(router1,router2,router3)**
 - **EXIT(253) ROUTINES=(rtn1)**
 - **EXIT(254) ROUTINES=(rtn2)**
 - **EXIT(255) ROUTINES=(rtn3)**

 - Routine **router1** consists of standard linkage and invokes **\$EXIT 253** with appropriate parameters
 - Routines **router2** and **router3** do likewise

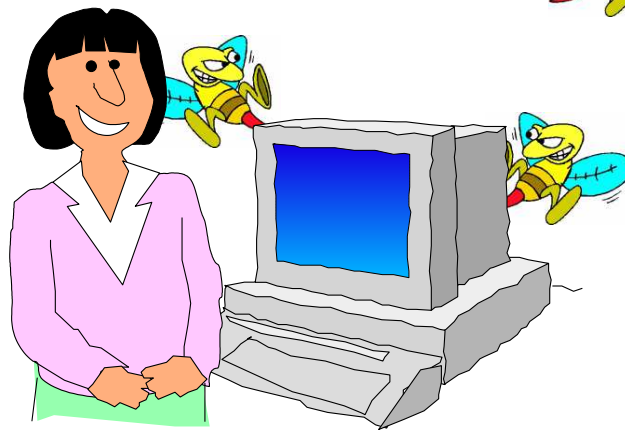
 - **\$T EXIT253,DISABLE** disables only routine **rtn1**

Enabling Exit 0



- Enabling exit 0 is a special case
 - Called **BEFORE** any EXIT statements can be processed
 - Load module name must be **HASPXIT0**
 - All routines must begin with the characters **EXIT0**
 - Routines are called in the order found in MITE
 - **\$DEXIT(0)** displays routines that were called by exit

Debugging



Debugging aids



■ \$TRACE 13

- Traces every exit point on entry and exit

```
17.43.12.79 ID = 13 $EXIT STC00009 STCINRDR 0667B0D0
# 4: ENVIRON= JES2 LABEL= RXITCCA PRE INVOCATION
R0-R7 = 00000004 0667B200 0667B428 00006000 ...
R8-R15 = 00000000 00000000 06955000 00007000 ...
```

```
17.43.12.79 ID = 13 $EXIT STC00009 STCINRDR 0667B0D0
# 4: ENVIRON= JES2 LABEL= RXITCCA POST INVOCATION
LAST ROUTINE CALLED = DIAGX04 R15-R1 = 00000000 00000004 ...
```

```
17.43.12.81 ID = 13 $EXIT ASID 0017
# 36: ENVIRON= USER LABEL=EXIT6RTN PRE INVOCATION
R0-R7 = 00000000 001344E0 83DA33A4A 03A97810 ...
R8-R15 = 001340D0 00000000 03C490000 00006000 ...
```

- Was the input what was expected?
- Are multiple callers in exit simultaneously?

Debugging aids



- \$SAVE/\$RETURN \$TRACE ids
 - 1 and 2 - JES2 and FSS environments
 - 11 and 12 - Only \$SAVE/\$RETURN calls for specific PCE types
 - 18 and 19 - USER and SUBTASK environments

```

12.59.59.08 ID = 1 $SAVE COMM 066690E8 PRKEYWRD
R14-R1 = 800F338E 000F4E70 00000000 7F6F100C
12.59.59.08 ID = 2 $RETURN COMM 066690E8 PRKEYWRD
R14-R1 = 800F338E 00000000 00000000 7F6F100C

14.07.06.28 ID = 18 $SAVE ASID 012E USERSUB
R14-R1 = 85EE8770 05F03C70 7F706AD8 7F6FC58C
14.07.06.29 ID = 19 $RETURN ASID 012E USERSUB
R14-R1 = 85EE8770 00000000 7F706AD8 7F6FC58C
    
```

- What services were called by exit?
- What point in processing was exit called?
- What was happening at same time as exit?

Debugging aids



■ \$HASP088 message

```

$HASP088 ----- JES2 ABEND ANALYSIS -----
$HASP088 FMID = HJE7703 LOAD MODULE = HASJES20
$HASP088 SUBSYS = JES2 OS 2.10
$HASP088 DATE = 2001.179 TIME = 15.14.02
$HASP088 DESC = PROTECTION EXCEPTION
$HASP088 MODULE MODULE OFFSET SERVICE ROUTINE EXIT
$HASP088 NAME BASE + OF CALL LEVEL CALLED ##
$HASP088 -----
$HASP088 *UNKNOWN 00000004 + 000000 UNKNOWN *ABEND S0C1 7
$HASP088 HASTDIAG 00189000 + 009124 NONE XIT7MSG 7
$HASP088 HASPNUC 00007000 + 00742C NONE DIAGX07 7
$HASP088 HASPNUC 00007000 + 0079F8 NONE $JESEFF -
$HASP088 HASPNUC 00007000 + 0076B0 NONE CBMWRTN
$HASP088 HASPRDR 000DC000 + 00430C OW44439 $CBIOM
$HASP088 HASPRDR 000DC000 + 007DCE OW44439 RWRTJOB
$HASP088 HASPRDR 000DC000 + 000C94 OW44439 HASPRJCS
$HASP088 PSW = 071C0600 00000004 ILC = 2 IC = 01
$HASP088 ASID = 0017 (HOME) 0017 (PRIM) 0017 (SCND)
$HASP088 PCE = STCINRDR (066790D0) STC00007 DEALLOC
$HASP088 R0 = 06618284 06953000 002921B0 00000007
$HASP088 R4 = 06679400 06953000 06953000 06953000
$HASP088 R8 = 800E030C 000E3A48 06618284 00007000
$HASP088 R12 = 001994A0 066790D0 001995B0 00000000
$HASP088 -----
    
```

Debugging aids



■ Finding your exit in storage

- On a live system:
 - \$D MODULE command

```

$dmodule(hastcdia),long
$HASP468 MODULE(HASTCDIA) ADDRESS=00B3B000,ASSEMBLY=(01/23/01,
$HASP468          14.25),ENVIRON=USER,EXITPTS=(),
$HASP468          FMID=HJE7703,IBMJES2=SAMPLE,
$HASP468          LASTAPAR=NONE,LASTPTF=NONE,
$HASP468          LENGTH=0010D0,LOADMOD=HASTCDIA,
$HASP468          MACLEVEL=6,
$HASP468          ROUTINES=(DIAGXITC=00B3B050,
$HASP468          DIAG3637=00B3C0D0,RMTWAIT=00B3C790),
$HASP468          SPLEVEL=CHECK,
$HASP468          VERSION=OS 2.10,UVERSION=

```

- In a dump
 - \$MIT formatting
 - \$XIT/\$XRT formatting

Debugging aids



- **SLIP** in routine to verify flow
- Obtain dumps at strategic points in your exit
 - **SLIP** dump
 - Zap bad opcode (X'0000')
 - **\$ERROR**
 - **\$DISTERR**
- Use IPCS to:
 - Verify register contents
 - Verify data structures
 - Look for possible overlays (PCE analysis, etc.)
 - Look for possible storage leaks (\$GETWORK analysis)

Summary



- Research your exit thoroughly
 - Investigate alternatives
 - Understand the costs involved
- Understand JES2
 - Different JES2 environments
 - Point in processing of exit
 - What exit can and cannot do
- Test your exit thoroughly
 - Understand all possible inputs and outputs
- Protect your production system
 - Build in recovery
 - Know effects of disabling/re-enabling exit