#SHAREorg

JES2 Bootcamp – Part 2 of 2
Setup, Operations, and Customization

Tom Wasik
IBM Rochester, MN
wasik@us.ibm.com
Wednesday 11:00 AM
Session Number 15326

Mainframe 50 April 7th 1964 - April 7th 2014
http://www.ibm.com/mainframe50/

SHARE in Anaheim

Trademarks

The following are trademarks of the International Business Machines Corporation in the United States and/or other countries.

- IBM®
- MVS™
- Redbooks®
- RETAIN®
- z/OS®
- zSeries®

The following are trademarks or registered trademarks of other companies.

- Java and all Java-related trademarks and logos are trademarks of Sun Microsystems, Inc., in the United States and other countries.
- Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.
- Microsoft, Windows and Windows NT are registered trademarks of Microsoft Corporation.
- UNIX is a registered trademark of The Open Group in the United States and other countries.
- SET and Secure Electronic Transaction are trademarks owned by SET Secure Electronic Transaction LLC.
- All other products may be trademarks or registered trademarks of their respective companies.

Notes:

What do you do if JES2 is not working?

- Issue the command $JD STATUS
- JES2 monitor detects abnormal conditions
  - $JD STATUS reports those conditions
- Normal output:

```
 $jdstatus
          $HASP9120 D STATUS
$HASP9121 NO OUTSTANDING ALERTS
$HASP9150 NO JES2 NOTICES
```

- Abnormal output

```
 $jdstatus
          $HASP9120 D STATUS
$HASP9121 NO OUTSTANDING ALERTS
$HASP9150 JES2 NOTICES
$HASP9158 JES2 PROCESSING STOPPED, $S NEEDED
$HASP9161 NOT ALL SPOOL VOLUMES ARE AVAILABLE
```

Setting up JES2

- JES2 is a subsystem and must be defined to MVS
- Use the IEFSSN member to define JES2
  - `SUBSYS SUBNAME(JES2) PRIMARY(YES) START(YES)`
    - `JES2,,,PRIMARY` – old format
- This defines the primary subsystem
  - You must define a primary subsystem
    - It must be either a JES2 or JES3 subsystem
    - But the subsystem name can be whatever you want
      - *I have seen HASP and JES9*
  - There are other subsystem besides JES
- Name corresponds to a PROC or address space name (more later)
- The START attribute is a choice
  - Should MVS start during IPL or should a command start it
    - Command can be in COMMNDxx parmlib member
    - COMMNDxx allow for specification of overriding PARM=

Setting up JES2 – Just one?

- You can run more than one JES2s on an MVS image
  - Called poly-jes
  - Each has an IEFSSN statement with a unique name
    - SUBSYS SUBNAME(JESA)
- Why more than one JES2?
  - Second JES2 can be for testing
    - Verifying initialization deck
  - Dedicated to some special processing
    - SAPI, offload, etc
  - Offloading work from main JES
- Secondary JES can be in same MAS as primary or some other MAS

Setting up JES2 – JES2 PROC

- Wide variety of options for the JES2 PROC
- Generally simpler is better (less that can go wrong)

```
//JES2     PROC
//IEFPROC  EXEC PGM=HASJES20,TIME=1440,REGION=0M
```

  - That is about as simple as it gets
- Other common things in the JES2 PROC
  - STEPLIB DD for where to load JES2 code
  - HASPPARM DD for where to read the initialization statements
  - PROCLIB concatenations for use by conversions
  - HASPLIST DD for init deck output listing
- Good practice to have symbolics for data set named in PROC
  - In case something gets deleted by mistake

Setting up JES2 – JES2 PROC

- A more typical JES2 PROC

```
//JES2    PROC MEMBER=JES2Z7TS,STLIB=SHASLNKE,SYS=SYS1,
//   CONNECT=CONNECTS,NODE=NODES
//IEFPROC EXEC PGM=HASJES20,TIME=1440,REGION=0M,PARM=(WARM,NOREQ)
//STEPLIB   DD DSN=&SYS..&STLIB,DISP=SHR             <- Where JES2 is loaded from
//PROC00    DD DSN=SYS1.PROCLIB,DISP=SHR             <- One of the PROCLIB concatenations
//          DD DSN=SYS1.COB140.PROCLIB,DISP=SHR
//          DD DSN=SYS1.PPROD390.PROCLIB,DISP=SHR
//*         DD DSN=SYS1.VTAM.PROCLIB,DISP=SHR
//          DD DSN=SYS1.USER.PROCLIB,DISP=SHR
//          DD DSN=SYS1.CICS.PROCLIB,DISP=SHR
//          DD DSN=SYS1.MQM.PROCLIB,DISP=SHR
//PROC01    DD DSN=TSO.PROCLIB,DISP=SHR              <- Second PROCLIB concatenations
//          DD DSN=TSO.USER.PROCLIB,DISP=SHR
//HASPPARM  DD DSN=SYS1.JESPARM(&MEMBER),DISP=SHR    <- JES2 initialization decks (normal)
//          DD DSN=SYS1.JESNODE(&CONNECT),DISP=SHR
//          DD DSN=SYS1.JESNODE(&NODE),DISP=SHR
//HASPLIST  DD DDNAME=IEFRDER                        <- Listing data set
//*      HASPPARM=JES2ALT TO FIND ALTERNATE INIT DECK
//JES2ALT   DD DSN=SYS1.PARMLIB(JESALTXA),DISP=SHR   <- JES2 initialization decks (alternate)
//          DD DSN=SYS1.JESNODE(&CONNECT),DISP=SHR
//          DD DSN=SYS1.JESNODE(&NODE),DISP=SHR
```

Setting up JES2 – JES2 PROC recomendations

- PROCLIBs are the most common problem starting JES2
  - Data set gets deleted and JES2 will not start
  - Also the easiest to fix (put data set back)
- Use PROCLIB initialization statements to define PROCLIB
  - Errors in the initialization deck do not stop JES2 from starting

- Minimize the number of DDs for the initialization deck
  - Use INCLUDE statements for read additional init decks
  - Use default parmlib instead of HASPPARM DD

- Depending on installation policies, consider access JES2 load module via a STEPLIB
  - Can make switching to an alternative code level easier

JES2 statement format

- JES2 uses a common parser for
  - Initialization statements
  - Most operator commands
  - Command responses and messages
- Once you understand the syntax, you can figure out most commands and initialization statements
  - General init statement format - object keyword=value
  - General command format is - $verb object,keyword=value
  - Display is - $HASPxxx object keyword=value
- One important difference (quirk)
  - Commands have a comma after the object
  - Init statements have a space after the object

JES2 statement format example INTRDR

- Here is an example using the INTRDR object
  - Initialization deck

```
D INTRDR
$HASP838 INTRDR
   AUTH=(DEVICE=NO,JOB=YES,SYSTEM=NO),BATCH=YES,
$HASP838
   CLASS=A,HOLD=NO,HONORLIM=NO,PRTYINC=0,
$HASP838         PRTYLIM=15,TRACE=NO
INTRDR TRACE=YES
```

  - Operator command

```
$D INTRDR
$T INTRDR,TRACE=YES
```

Statement format – Space rules

Initialization statements can have space between items

- `INTRDRTRACE=NO` – error `INTRDR TRACE=NO` – OK
- `INTRDR TRACE = NO` – OK
- `DINTRDR` – error `D INTRDR` – OK
- `D INTRDRTRACE` – error `D INTRDR TRACE` – OK

Commands ignore all spaces

`$TINTRDR,TRACE=YES` – OK

`$ T I N T R D R , T R A C E = Y E S` – OK

Statement format – Comment rules

- Comments are enclosed in /* */
  - Init decks allow /* to end of line as a comment
- Generally anywhere there can be a space, you can put a comment
- In init decks, you can replace a required space

```
D/*comment*/INTRDR
INTRDR TRACE/*comment*/=/*comment*/YES
```

- In commands

```
$T /*c*/ INT/*c*/rdr,tr/*c*/ace=yes
```

- Flexibility exists but do what makes sense

Statement format – Subscripts

- Some objects represent a group of things to manipulate
  - JOBCLASS(*x*) or PRINTER(*x*)
- The individual object is identified by a subscript
- Some subscripts are numbers, others characters
  - Some can be both – NODE(1) or NODE(ANAHEIM)
- Variants on how to specify subscripts
  - Range – NODE(1-5) or NODE(5-1)
  - Generics – NODE(ANA*) or all – NODE(*)
  - List – NODE(1,2,ANAHEIM)
- Placing subscript in quotes is an absolute comparison string
  - $D J('?') displays jobs with a job name of ?
- Subscript can be omitted for displays (implies (*))
- Not all commands or all objects support generics
  - $ADD SPOOL(*) – Not allowed

Statement format – limiting displays

- Do a display and only provide object

```
D INTRDR
```

  - All keywords display
- Do a display and list keyword only

```
D INTRDR CLASS,TRACE
```

  - Only listed keywords display (in order listed)

```
$HASP838 INTRDR  CLASS=A,TRACE=NO
```

- This is useful if you want to know only specific information
- Also useful if you plan to parse the response of the command
  - Simplifies response based automation

Statement format – Filtering

- Filtering allow you to select which object to work on
    - Used with subscripts to control what is processed
- Most keywords can be used as filters
    - If it is a display action or non-settable keyword format is
        - Keyword op value – eg STATUS=ACTIVE or SLEVEL>0
            - *Op can be =, <, >, <=, >=, <>, or combined with ! or ¬*
    - Alternate format is to precede keyword with a /
        - Always a filter with a /, even for settable keywords
        - Eg /TRACE=NO or /DEST=LOCAL
- Filtering can be used on commands and init statements

Statement format – putting it all together

- Combining subscripts, filters, and set
  - `$TO JQ(*),/DEST=LOCAL,DEST=N2`
    - Reroute the output of all output destined locally to node 2
  - `$T PROCLIB(*),DD(*)=(/DSN=TEST.PROCLIB,DSN=TEST2.PROCLIB)`
    - Change the data set names in all PROCLIB DDs that specify TEST.PROCLIB to TEST2.PROCLIB
  - `$T PROCLIB(*),DD(*)=(/DSN=TEST.PROCLIB),DD(99)=(DSN=TEST3.PROCLIB)`
    - Add (or set) the 99th DD of any PROCLIB that has TEST.PROCLIB to TEST3.PROCLIB
    - Note PROCLIB DD subscripts specifications above the last get compressed down
      - *So if no PROCLIBs have 99 DDs, this add TEST3.PROCLIB*
  - `$TO JQ(*),/FORMS=3HOLE,CLASS=3`
    - Change all output that has a forms value of 3HOLE to have a SYSOUT class of 3
  - `$TJOBCLASS(A-Z),/ACTIVE=YES,SYSSYM=ALLOW`
    - Change any alphabetic class (including STC and TSU) that specify ACTIVE=YES to have SYSSYM=ALLOW

Statement format – putting it all together

- Beware some subtleties
  - Subscript of (*) implies all existing vs (1-*) implies all possible
    - PRINTER(*) init statement refers to all existing printers
    - PRINTER(1-*) init statement creates all printers from 1-32767
      - *PRINTER(*) FSS=PSF vs PRINTER(1-*) FSS=PSF*
      - ** affects all exist printers and 1-* defined 32767 printers with FSS=PSF*
  - Names that are numbers (NODE and MEMBER are init statements)
    - NODE(1) NAME=2 and NODE(2) NAME=ANAHEIM
    - MEMBER(3) NAME=12 and MEMBER(12) NAME=DSNY
    - Numbers match in preference to names
      - *MEMBER(12) is number 12 (DSNY) not number 3*
      - *MEMBER('12') is number 3 (with name 12)*
      - *NODE(2) is ANAHEIM, NODE('2') is same as NODE(1)*

The JES2 initialization deck

- Contains all the statements needed to set up JES2
  - Supports system symbols substitution
- Can be a DD in the JES2 PROC
  - Default is HASPPROC or PARM=(HASPPARM=ddname)
- Can be a member of PARMLIB concatenation
  - Default is **HAS*jesx*** where *jesx* is the subsystem name
    - Default is only used if no HASPPARM DD in PROC
  - Specify specific as PARM=(MEMBER=*parmmemb*)
- Which you use is more an installation preference
  - PARMLIB is more reliable but there may be an access issue
    - But nice to have all initialization parms in one place
  - JES specific data set is easier manage
    - Keep JES separate from rest of MVS

The JES2 initialization deck

- Regardless of where you load, organization is key
- Initialization statements can be grouped by purpose/scope
  - Common initialization statements to set up JES2
    - Checkpoint, SPOOL, MAS wide resources
  - Local definition statements
    - Member name, local resource limits, PROCLIBs, initiators, etc
  - Device definitions
    - Printers, FSSs, RJE, Lines, Logons, Netservs
  - NJE definitions
    - Nodes and connect statements, ownnode, etc
- Exact organization depends on personal preferences

The JES2 initialization deck

- The INCLUDE statement brings in new initialization decks
  - **INCLUDE DSNAME=**dsn(member) for including from a PDS
  - **INCLUDE MEMBER=**member includes another member from current data set (or parmlib if that is current)
    - NOT from current DD
  - **INCLUDE PARMLIB_MEMBER=***member* includes from logical PARMLIB data set
- For MAS wide parameters consider specific names
  - MEMBER=NJEDEFS
- For local definitions consider symbolic names
  - MEMBER=LCL&SYSNAME(1:4)
- Two philosophies for organization
  - Start with member specific and INCLUDE global members
  - Start with global members and INCLUDE member specific

The JES2 initialization deck

- Use the $D INITINFO to see what was used (new in 2.1)

```
$HASP825 INITINFO  --- Command used to start JES2
$HASP825           S JES2,PARM=(WARM,NOREQ)
$HASP825           --- HASPPARM data sets read
$HASP825           DSN=SYS1.PARMLIB(SPOOLZ21),VOLSER=J2SHR2,
$HASP825           CARDS=458,
$HASP825           DSN=SYS1.PARMLIB(DYEXIT21),CARDS=122,
$HASP825           DSN=CONSOLE,CARDS=1,
$HASP825           DSN=SYS1.PARMLIB(NULL),VOLSER=J2SHR2,CARDS=1
$HASP825           --- STEPLIB Concatenation
$HASP825           DSN=ZOS21.LINKLIB,VOLSER=J2COM1,
$HASP825           DSN=NULL.JES2000.LINKLPA,VOLSER=J2SPA1,
$HASP825           DSN=SYS1.SRVLIB.JES2000.LINKLPA,
$HASP825           VOLSER=J2SPA1,
$HASP825           DSN=SYS2.LINKLIB,VOLSER=ZDR21B,
$HASP825           DSN=SYS1.MIGLIB,VOLSER=ZDR21B
```

Initialization deck error processing

- Initialization statement errors result in a WTOR
  - Intent is to allow correction of the problem
    - Can type correct statement in at the consol

```
$HASP003 RC=(01),JOBCLASS(A)  - INVALID SYNTAX
32 $HASP469 REPLY PARAMETER STATEMENT, CANCEL, OR END
```

  - In this case I coded `jobclass(a),SYSSYM=ALLOW`
  - Common problem of a comma after the operand
- Another class of error is logical errors
  - OFF1.JT CLASS=(ABC) and no class ABC

```
 $HASP539 CLASS=ABC on OFF1.JT is not a defined job class or job class
group
*39 $HASP441 REPLY 'Y' TO CONTINUE INITIALIZATION OR 'N' TO TERMINATE IN
  RESPONSE TO MESSAGE HASP539
```

  - Generally discovered after initialization deck is processed

Initialization deck error processing

- Should initialization WTOR caused by errors be automated?
  - They can delay start of JES2 and delay recovery
- But what if JES came up with the error setting?
  - Could error require another IPL to correct?
  - Would it be easier to fix now?
- Most things in JES2 can be updated by command
  - Would that be the right way to correct the problem?
- If reply is automated, would anyone notice the problem?

Initialization deck error processing

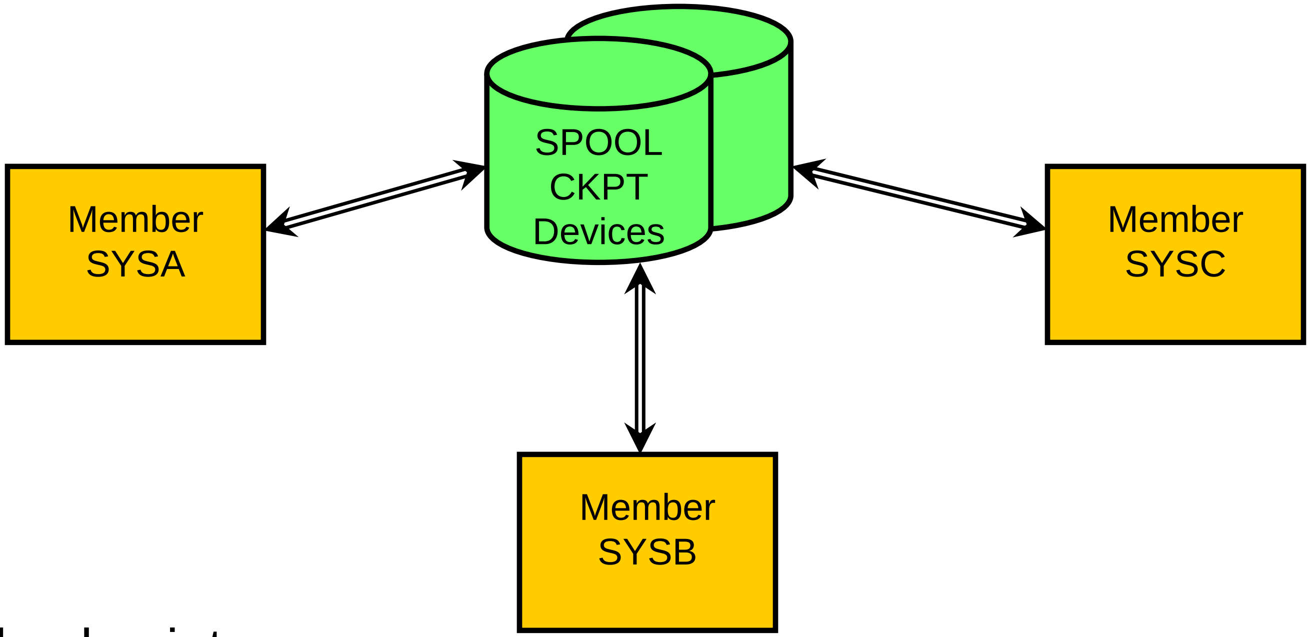
- Errors cause entire statement to be backed out
  - Not just bad setting of a keyword
  - Applies to initialization statements and command
- Consider placing fewer keywords on init statements

```
CKPTDEF   CKPT1=(DSN=SYS1.JESCKPT1,VOL=SPOOL1,INUSE=YES),
          CKPT2=(DSN=SYS1.JESCKPT2,VOL=SPOOL1,INUSE=YES),
          NEWCKPT1=(DSN=SYS2.JESCKPT1,VOL=SPOOL1),
          NEWCKPT2=(DSN=SYS2.JESCKPT2,VOL=SPOOL1),
          MODE=DUAL,DUPLEX=ON,LOGSIZE=4,VERSIONS=NUMBER=2
```

versus

```
CKPTDEF   CKPT1=(DSN=SYS1.JESCKPT1,VOL=SPOOL1,INUSE=YES)
CKPTDEF   CKPT2=(DSN=SYS1.JESCKPT2,VOL=SPOOL1,INUSE=YES)
CKPTDEF   NEWCKPT1=(DSN=SYS2.JESCKPT1,VOL=SPOOL1)
CKPTDEF   NEWCKPT2=(DSN=SYS2.JESCKPT2,VOL=SPOOL1)
CKPTDEF   MODE=DUAL,DUPLEX=ON,LOGSIZE=4,VERSIONS=NUMBER=2
```

JES2 MAS



Up to 2 checkpoints

Up to 253 spool volumes

Up to 32 JES2 images

JES2 Checkpoint

- A MAS is a set of JES2 images that share CKPT and spool
- Up to 2 checkpoints can exist per MAS
  - Recommend always have 2 checkpoint
  - One checkpoint is OK for short error recovery scenarios
- Checkpoint can be on DASD or in a CF structure
  - One checkpoint (CKPT2) should always be on DASD
- Two modes of checkpoint operation – DUAL and DUPLEX
  - Dual (DASD checkpoint only) trade more CPU for less I/O volume
    - Not a huge advantage on modern DASD

JES2 SPOOL

- Up to 253 spool volumes supported
  - 1 single extent spool data set per volume
  - Maximum data set size 1M tracks
  - Can be in cylinder managed storage
- Basic unit of spool allocation is the TRACK GROUP
  - 1-20 tracks per track group
  - Size can differ from one volume to another
- Each member performs allocations for job/processes on that member
  - Cache of 255 entries for when checkpoint is not owned
- Spools can be added (started) or deleted (drained) via operator command
- Spools can be migrated (merged) with the migrate command

JES2 SPOOL

- SPOOL affinity limits what SPOOLs a member can allocate SPOOL from
  - Can associate volumes with systems running a particular workload
  - Guideline rather than a hard limit
- SPOOL fencing can limit how many volumes a job can be spread over
  - Can limit impact of loss of SPOOL volume
  - Has negative performance implications
  - Specify the number of volumes that can be used

JES2 SPOOL I/O

- Each address space does EXCP(VR) to read/write SPOOL
  - Each JES data set does its own EXCP(VR)
  - Writes to SYSOUT done asynchronously
  - I/Os can be done in parallel with PAV
    - Even within a single address space
- Uses 31 bit private buffers in application address space
- Supports 31 bit callers (ACB/RPL interface)
- Allocations can use XTIOT
  - Set NON_VSAM_XTIOT=NO|YES in DEVSUP*xx*

Key init parms – CKPTDEF

- CKPTDEF defined what data sets to use for CKPT
- CKPT can be on DASD or in a CF structure
- There are 2 checkpoints for redundancy, CKPT1 and CKPT2
- CKPT1 on DASD good for smaller MAS (and GDPS)
- CKPT1 on CF for larger MAS (>3 members)
- One caution with CKPT on CF, JES2 uses CF as storage not as cache
  - Data on CF is not completely backed up except on CKPT2
  - If CF data is lost (CF is restarted), some state info may be lost
  - Move data off CF when CF is being restarted for maintenance
  - This is why CKPT2 should always be on DASD

Key init parms – CKPTDEF

- Checkpoint reconfiguration is used to move CKPT data sets
  - Can move checkpoint to any new location via operator command
  - Can deal with checkpoint harware errors automatically
    - Need to set up NEWCKPTn data sets for this to happen
    - Should set OPVERIFY=NO to ensure automatic recovery
- It is very important to understand how to deal with reconfiguration
  - Customers have had unnecessary outage because they did not understand reconfiguration
  - See SHARE CKPT reconfiguration presentation for more info

  ftp://service.boulder.ibm.com/s390/jes2/Share103/s2662.pdf

Key init parms – SPOOLDEF

- Important on cold start to point to correct spool volumes
- Two methods to locate spool (as of 1.13)
  - Based on a fixed data set name and volser prefix
    - SPOOLDEF DSN= and PREFIX=
  - Based on SPOOL initialization statements
    - Limited by SPOOLDEF DSNMASK= and PREFIX with generics
- Warm starts (as of 1.13 and compat APAR) use data set names and volumes from the checkpoint
  - Makes SPOOLDEF less important except for cold start

Key init parms – SPOOLDEF

- Also controls
  - Fencing – how spread out jobs are
  - Number of spool volumes supported
  - Track cell size – only set on a cold start
  - Track group size – set when volume starts
  - Buffer size – usually 3992 unless some special reason
  - Track group map size – number of tracks groups of spool
    - This is a limited MAS wide resource that needs to be monitored
    - More about monitoring resources later
- Spool presentation: Defining, Managing, and Updating

https://share.confex.com/share/118/webprogram/Handout/Session10844/JES2%20SPOOL.pdf

Key init parms – MASDEF

- Defines JES2 member name
  - Default is 4 character SMF id for system
- Defines XCF group name
  - Defaults to NJE node name
  - MUST be unique name XCF
- Controls auto reset for members
  - AUTOEMEM and RESTART parameters
  - Should be set to AUTOEMEM=ON and RESTART=YES
- Defines CKPT tuning parameters HOLD and DORMANY
  - Critical for tuning JES2 checkpoint process
  - See JES2 performance considerations for more info

  http://www.share.org/p/do/sd/sid=8717&type=0

Key init parms – Resource limits

- There are 2 types of resources in the initialization deck
  - MAS wide resources
    - Can cause MAS delays if resources are exhaused
    - BERTs, JOEs, JQEs, job numbers, track groups
  - Local resources
    - Can cause local delays if resources are exhausted
    - Message buffers, CKVRs, buffers, extended buffers, BSC buffers, SNA buffers, SMF buffers, trace buffers, header buffers, SNA session buffers, command buffers

Key init parms – Resource limits

- Resource limits exist for a reason
  - MAS level limits are related to data areas in the CKPT
    - Number is limited by size of the data set
    - Grossly over defining can impact performance
    - But having a lot extra is not bad
    - Making CKPT data sets extra large is a good idea
      - *2x needed size is not unreasonable*
  - Local limits prevent runaway processes
    - Runaways are limited by resources defined
    - One runaway process does not stop other things JES does

Key init parms – Resource limits

- HASP050 message warns of resource shortages
  - Each resource has a warning threshold
  - When exceeded, get HASP050 message
- Do not set warn too low or HASP050 gets ignored
- Do not set warn too high or no time to react to shortage

Key init parms – Resource limits

- Ignore HASP050 messages at your own peril
  - Can result in many wasted hours
  - Check old SYSLOGs for HASP050 messages
  - Did operations do anything or just ignore them?
  - Is warning level appropriate? Is limit appropriate?
- JES2 monitor records history of resource usage
  - $JD HISTORY displays results
  - Also in SDSF RM panel
- See SHARE presentation "How to Recover from JES2 Imminent Disasters"

http://www.share.org/p/do/sd/sid=1898&type=0

Don’t Run Out of BERTs

•Don’t run out of BERTs

Dynamic PROCLIB

- Use instead of PROCxx in JES2 PROC
  - Define PROCs in the JES2 init deck
  - From example JES2 PROC

```
//PROC00     DD DSN=SYS1.PROCLIB,DISP=SHR
//           DD DSN=SYS1.COB140.PROCLIB,DISP=SHR
//           DD DSN=SYS1.PPROD390.PROCLIB,DISP=SHR
//           DD DSN=SYS1.USER.PROCLIB,DISP=SHR
//           DD DSN=SYS1.CICS.PROCLIB,DISP=SHR
//           DD DSN=SYS1.MQM.PROCLIB,DISP=SHR
```

  - Init deck format

```
PROCLIB(PROC00) DD1=(DSN=SYS1.PROCLIB),
                DD2=(DSN=SYS1.COB140.PROCLIB),
                DD3=(DSN=SYS1.PPROD390.PROCLIB),
                DD4=(DSN=SYS1.USER.PROCLIB),
                DD5=(DSN=SYS1.CICS.PROCLIB),
                DD6=(DSN=SYS1.MQM.PROCLIB),
                UNCONDITIONAL
```

Dynamic PROCLIB

- If a DD in PROCLIB(PROC00) fails allocation, initialization continues
- UNCONDITIONAL keyword allocates what it can
    - Others are skipped
- Can updated concatenation list with $T command
    - Even if in initialization deck (small change in 2.1)
- Can add and delete PROCLIB with $ADD and $DEL PROCLIB commands
- Display is with $D PROCLIB command

DSI or to NODSI for PROCLIB

- By default, JES2 (HASJES20) is defined with NODSI in the PPT
  - This implies that DSN ENQs are not held for data set defined with DDs in the JES2 PROC
  - As a result, data sets can be deleted while JES2 is using them
- DSI/NODSI does not affect dynamic allocation
  - Controlled by bit set at allocation time
  - JES2 always set NODSI for all dynamic allocations
- You can override the PPT using SCHED*xx*

```
PPT PGMNAME(HASJES20)      /* PROGRAM NAME APSPPIEP              */
    KEY(1)                 /* PROTECTION KEY 1                   */
    NOSWAP                 /* NON-SWAPPABLE                      */
    NOCANCEL               /* JOB IS CANCELLABLE                 */
    PRIV                   /* NOT PRIVILEGED                     */
    SYST                   /* SYSTEM TASK, NOT TIMED             */
    DSI                    /* REQUIRES NO DATASET PROTECTION     */
    NOPASS                 /* REQUIRES PASSWORD PROTECTION       */
    AFF(NONE)              /* THIS PROGRAM REQUIRES NO AFFINITY  */
    NOPREF                 /* PRIVATE AREA SHORT-TERM FIXED PAGES*/
                           /* NEED NOT BE ASSIGNED TO PREFERRED  */
                           /* STORAGE FRAMES.                    */
```

To DSI or to NODSI

- So which to use?
  - Customers report success with DSI set
  - Prevented accidental deletion of PROCLIB DSN defined in JES2 PROC
  - Special consideration to rename data sets with ENQ active
- JES2 PROCLIB support
  - Dynamic PROCLIB, JES2 honors PPT DSI setting
    - Specifies DSI if PPT for HASJES20 has DSI

NJE definitions

- NJE transmits JOBs and SYSOUT between peers
- Protocol that sits on top of BSC, SNA, and TCP/IP
- Must define NJE nodes (a MAS is a node)
- Connections between nodes (optional)
- Devices to establish connections between nodes
  - Transmitters and receivers (job and SYSOUT)
  - Lines
  - Servers (NETSERVs or LOGONs)
- All can be set up in the initialization deck or via operator commands
- Way too much to get into in this presentation
- See the ABCs of NJE SHARE presentation

http://www.share.org/p/do/sd/sid=8718&type=0

Job classes

- Job classes serve two purposes
    - Help to divide work to make it easier to manage
        - Long vs short, I/O vs CPU intensive, group/department/application "owned" class
        - Give priority to certain classes of work
            - *System programmer class so you do not have to wait to run jobs?*
    - Limit access to or assign certain attributes
        - Bypass label processing (BLP)
        - ENQ downgrade capability (new in 2.1)
        - Journaling
- How many do I need?
    - One is too few, 500 probably too many
    - Depends on how finely you need to control jobs and properties

Job classes

- What about service class?
  - A job cannot specify its service class
    - Installation policy assigns it based on various criteria
  - Job class is one criteria for assigning service class
  - Service class is how jobs are assigned to WLM initiators
    - Service class defined how jobs are ordered (scheduled) to initiators
      - *Ordered base on time job was first made available to execute*
  - However, some attributes are hard to derive when assigning service class
    - Just how long will job run?

Job classes

- Use scheduling environment (SCHENV) to identify jobs resource needs
  - Abstract resources
    - Need to run with certain DB2 or other application
    - Need to run off shift
  - Real resources
    - Tapes, DASD, etc
  - Criteria used to select service class
  - Not good for “how many” or “how much” type screnarios
    - Need 10 tape drives
    - Going to run a long time

Job classes
Criteria to select service class

- The following is input to WLM classify for batch jobs
  - Owning user id
  - Job class
  - Job name
  - Accounting string
  - Performance group
  - Priority (current)
  - Scheduling environment
- Question to think about is what these tell you about your jobs?
  - And is that knowledge going to change? (is it easy to manage)
  - Owning user id can indicate department/division
    - But is it an individual or department
    - If an individual, what if they change jobs?
    - If department, what if there is a breakup or combining of departments?

Job classes

- So what should I do? How do I control my jobs?
  - History can dictate this
  - Certain things easier to do one way or another
    - Understand the strength and weakness of jobclass vs service class
  - WLM recommends NOT having too many service classes
    - How many is too many?
  - Exits are often used to help manage these problems
    - This includes vendor products to help assign service class, job class, and SCHENV
  - And will you run WLM or JES mode initiators?
    - Comments you will hear about this
      - *WLM mode takes away my control over jobs*
      - *JES mode means more work to manage initiator*

Job class vs service class Strengths and weakness

- Want to control how many job from department x run at a time
  - JOBCLASS execution limits by system or by MAS
- Want to manage jobs by run time (wall clock or CPU?)
  - Can I derive this information from classify criteria?
  - Can I manage this by SCHENV?
  - Or is it easier to use job classes? (short, medium, long, epoch)
- Want to run this job where I have the xyz license
  - Scheduling environment resource
- Can only run 2 job at a time that use product xys (license again)
  - JOBLCLASS execution limits
- Certain old applications require SWA below the line
  - JOBCLASS for SWA placement policy
- Need to protect who can use BLP
  - JOBCLASS SAF/RACF security and BLP property (2.1 feature)

Job class one vs multi character (new in z/OS 2.1)

- Use only 4 or 5 job classes
  - Stick with single character
  - Consider making unused job classes ACTIVE=NO
    - Prevents junk jobs sitting, waiting to execute
- Use more that 10 or 20
  - Do you have a document explaining what each class means?
  - When was the last time it was updated?
  - Pure usability says you should consider multi character
- Use more than 20?
  - First do you REALLY need that many?
    - History driving you? We always added a job class for new workload
    - Are they really all used? Application that need job class went away
      - *Just because jobs are running in a class does not mean they should*
  - Again usability starts becoming more important than history

SYSOUT routing

- Some SYSOUT attributes correspond to a physical thing
  - FORMs, UCS, etc
- Others are abstract scheduling attributes
  - Class, destination, PRMODE, writer
- Have to understand how SYSOUT is processed
  - Printers select criteria specified on WS=
  - SAPI devices select using application defined work selection
  - External writers are a more traditional way of selecting work
- There are multiple ways to get things done
  - Most exist for traditional reason
    - That's how the JCL was set up when the new printers arrived in 1985

SYSOUT routing

- Some general rules of thumb
  - Selection should include at least one of
    - CLASS, DESTINATION and OUTDISP
  - This ensures better performance
- SYSOUT is queued based on SYSOUT class
  - Within each class is 3 queue heads based on destination
    - Local – Destination of local or U*xxxx* (U1234)
    - Remote – Destination of Rxxxx (R1234)
    - User – Destination of *xyz* (not local or remote)
- Try to use as many queues as practical to reduce long queue size
- SAPI performance selecting from these queues is better in z/OS 2.1
- See JES2 performance consideration SHARE presentation

http://www.share.org/p/do/sd/sid=8717&type=0

Useful operator commands

- Clean up old spool files:

```
$PO JOBQ,/Q=S,/Days>4  /*Class S output older than 4 days */
$P JQ, /DAYS>7         /* Purge all Jobs older than 7 days */
$T A,I=86400,'$PJQ,/DAYS > 7'     /* Use Automatic Commands*/
```

- Keep Lines started & Nodes connected:

```
$T A SLNE, I=3599,'$S LINE(2-27)'  /* Start all SNA Lines */
$T A SNL2, I=3600,'$SN,LINE2,N=WSCNEXT‘
```

- Alternative to keep lines started and nodes connected
  - `RESTART=(YES,10)` On LINEs, LOGONs and NETSERVs
  - CONNECT=YES On NODEs, APPLs, and SOCKETs
  - CONNECT=YES On NJEDEF

Security (SAF/RACF)

- JES2 interfaces with SAF/RACF to provide security
- JES2 directly used classes
  - JESJOBS – protects access and modifications to jobs
  - JESSPOOL – protects access to SYSOUT and instream data sets on SPOOL
  - WRITER – protects access to NJE destinations and FSS/local printers and punches
  - NODE – control data from other nodes to NJE nodes
  - OPERCMDS – access to JES2 commands

Security (SAF/RACF)

- JES2 also gets involved with verifying batch jobs
  - Passes information like user and password to SAF/RACF
  - Gets back token that represents/protect the job
- Verification gets other classes involved
  - JESINPUT, SECLABEL, PROPCNTL, SURROGAT, etc
  - These are checks made in the security product itself
- See JES2 Security SHARE pitch

http://www.share.org/p/do/sd/sid=7213&type=0

Security (SAF/RACF)

- Don’t forget to use security to protect JES2
  - Restrict access to the spool and checkpoint data sets
    - Generally ONLY JES2 needs access to these data sets
    - There may be a vendor product or two that still read SPOOL
      - *Not a good idea in these days*
  - Protect PROCLIB data sets from unintended delete or update
    - JES2 only needs read access to PROCLIBs
    - Jobs do not need read access to PROCLIBs to use them
  - Protect your initialization decks
    - Very limited people need update access to these data set
- Reality is that even with restricted access, it is an authorized user that deletes something they should not
  - Be paranoid when deleting any data sets
    - The job you save may be your own

JES2 Exits

- In JES2 exits are considered extensions to the product
  - This historically is what customers wanted
  - Give exit control at a point in processing and let it do what it wants
  - No limits on what an exit can do
- This, combined with availability of source, leads to
  - Exits that can do amazing functions
  - Exits that are very sensitive to JES2 internal data structures
  - Exits that need to be updated every release to continue working
- Because of this, JES2 requires JES2 exits be re-assembled for every release
  - When fields change in nature, they are renamed to trigger assembly errors in exits
  - Assembly errors are GOOD because they tell you a change is needed
- Having said this, reality is
  - Most exits recompile and work with the new release
  - JES2 development focuses on NOT breaking exits
  - Newer exits have formal parameters to reduce dependence on internal structures

JES2 Exits

- Each exit point in JES2 has an exit number
- Exit code must be is placed in a JES2 format module
  - Has a $MIT and $MITABLE
  - Loaded by the JES2 LOADMOD statement
- An exit routine must be identified with a $ENTRY macro
- Each exit number can have multiple exit routines associated with it
  - Specify on the ROUTINES= keyword of the EXIT statement
  - Called in the order listed
- Exit return code are defined as
  - 0 – exit successful, continue calling routines
  - 4 – exit successful, stop calling exit routines
  - >4 – exit failed, stop calling exit routines

JES2 Table Pairs

- Not all JES2 customizations are done in exits
- Table pairs extend/customize JES2 processing
  - Pair is a misnomer – Support allows more than 2 tables
    - JES2 and User table (origin of pair concept)
    - PLUS dynamic tables – unlimited number of additional tables
- Table pairs can be used to
  - Add installation commands or keywords to command
  - Override default processing for a command keyword
  - Replace the text in messages issued by $BLDMSG
  - Add structures like PCEs, DTEs, PC routines, etc
  - Extend data area via $BERTTABs
- There are 2 major classes of table
  - Commands and their operands ($SCANTABs)
  - Other structures supported by $GETABLE
    - PCE TABLE ($PCETAB)
    - DCT TABLE ($DCTTAB)
    - DTE TABLE ($DTETAB)
    - TRACE ID TABLE ($TIDTAB)
    - PC Routine table ($PCTAB)
    - BERT table ($BERTTAB)
    - Work selection ($WSTAB)

SDSF

- SDSF provides many controls for JES2

DA	Active users	INIT	Initiators
I	Input queue	PR	Printers
O	Output queue	PUN	Punches
H	Held output queue	RDR	Readers
ST	Status of jobs	LINE	Lines
		NODE	Nodes
LOG	System log	SO	Spool offload
SR	System requests	SP	Spool volumes
MAS	Members in the MAS		
JC	Job classes	RM	Resource monitor
SE	Scheduling environments		
RES	WLM resources	ULOG	User session log
ENC	Enclaves		
PS	Processes		

- Only items you have access to are listed
  - So get access if you are missing something
- Not just for looking at jobs on the queues
  - Real time management of various JES2 function

What do you do if JES2 up but not working?

- Issue the command $JD STATUS
- JES2 monitor detects abnormal conditions
  - $JD STATUS reports those conditions
- Normal output:

```
 $jdstatus
          $HASP9120 D STATUS
$HASP9121 NO OUTSTANDING ALERTS
$HASP9150 NO JES2 NOTICES
```

- Abnormal output

```
 $jdstatus
          $HASP9120 D STATUS
$HASP9121 NO OUTSTANDING ALERTS
$HASP9150 JES2 NOTICES
$HASP9158 JES2 PROCESSING STOPPED, $S NEEDED
$HASP9161 NOT ALL SPOOL VOLUMES ARE AVAILABLE
```

Questions?

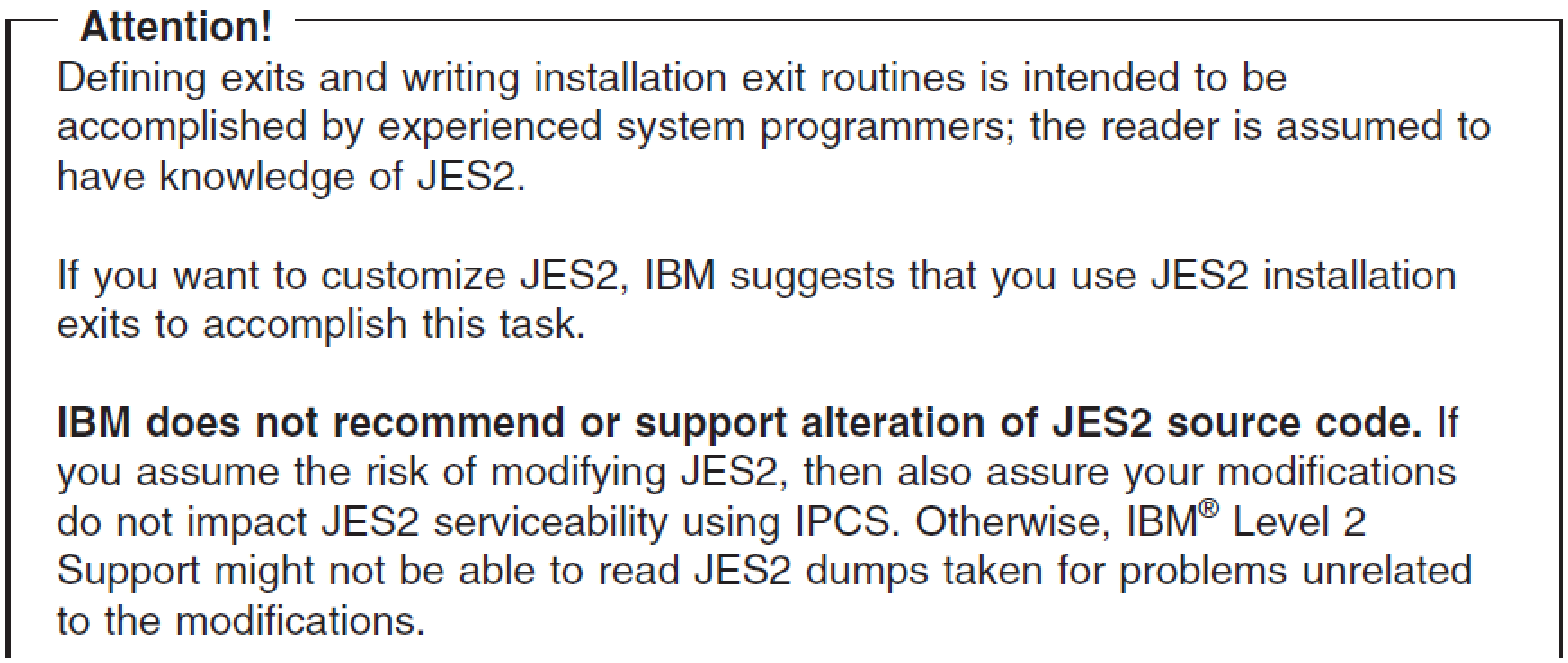
Questions?

Session 15326

Mainframe 50 April 7th 1964 - April 7th 2014
http://www.ibm.com/mainframe50/

JES2 Exits

- From JES2 exits chapter 1 page 1



Attention!

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 suggests that you use JES2 installation exits to accomplish this task.

IBM does not recommend or support alteration of JES2 source code. If you assume the risk of modifying JES2, then also assure your modifications do not impact JES2 serviceability using IPCS. Otherwise, IBM® Level 2 Support might not be able to read JES2 dumps taken for problems unrelated to the modifications.

JES2 Exits

- In JES2 exits are considered extensions to the product
  - This historically is what customers wanted
  - Give exit control at a point in processing and let it do what it wants
  - No limits on what an exit can do
- This, combined with availability of source leads to
  - Exits that can do amazing functions (eg what some vendors do)
  - Exits that are very sensitive to JES2 internal data structures
  - Exits that need to be updated every release to continue working
- Because of this, JES2 requires JES2 exits be re-assembled for every release
  - When fields change in nature, they are renamed to trigger assembly errors in exits
  - Assembly errors are GOOD because they tell you a change is needed
- Having said this, reality is
  - Most exits recompile and work with the new release
  - JES2 development focuses on NOT breaking exits
  - Newer exits have formal parameters to reduce dependence on internal structures

JES2 Exits

- Each exit point in JES2 has an exit number
- Exit code must be is placed in a JES2 format module
  - Has a $MIT and $MITABLE
  - Loaded by the JES2 LOADMOD statement
- An exit routine must be identified with a $ENTRY macro
- Each exit number can have multiple exit routines associated with it
  - Specify on the ROUTINES= keyword of the EXIT statement
  - Called in the order listed
- Exit return code are defined as
  - 0 – exit successful, continue calling routines
  - 4 – exit successful, stop calling exit routines
  - >4 – exit failed, stop calling exit routines

JES2 Exits - Environments

- All code in JES2 has an associated assembly environment
- Environment defines what services/macros are available
  - Some macro expand differently base on environment
- Environment also defines register contents
- Exit environments
  - JES2 - JES2 main task
  - SUBTASK - JES2 subtask
  - USER - JES2 USER environment
  - FSS - Functional subsystem environment
- Set on $MODULE ENVIRON= or $ENVIRON macro
- If you get this wrong, EXIT will probably ABEND
  - Wild branch is a typical problem

JES2 Exits - Environments

- JES2 Main Task environment– ENVIRON=JES2
- JES2 address space, JES2 Main task
  - MVS WAITs vs JES2 $WAIT is exit dependent
  - 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?

JES2 Exits - Environments

- Avoid MVS WAITs in JES2 environment!
  - Use $WAIT instead
    - Caution, some exits do not even allow $WAIT!
  - Use JES2 equivalent services that do not MVS WAIT
    - $WTO, etc.
  - If you must call a service which can MVS WAIT
    - Use the $SUBIT service to send request to a subtask
      - *PCE $WAITs while subtask does the MVS WAIT.*
  - Exceptions - JES2 Initialization/Termination (Exits 0, 19, 24, 26)
    - JES2 dispatcher not enabled, so CAN NOT $WAIT
    - ENVIRON=(JES2,INIT) can be used in 0, 19, 24
    - ENVIRON=(JES2,TERM) can be used in 26

JES2 Exits - Environments

- JES2 Subtask environment – ENVIRON=SUBTASK
- 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

JES2 Exits - Environments

- JES2 USER environment – ENVIRON=USER
- 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 address space is down!

JES2 Exits - Environments

- JES2 USER Environment – ENVIRON=(USER,ANY)
- Any address space, any task
  - R11 = HCCT
  - R13 = Available save area or PCE
- Same as ENVIRON=USER except:
  - $SAVE/$RETURN services based on caller
    - PCE caller use JES2 main task logic
      - *No BAKR's (so $WAIT can be done if main task)*
    - Not PCE use USER environment logic
- Useful for routines that could be called in the user environment OR under main task
  - Detect run-time environment by checking
    - PSAAOLD, PSATOLD for JES2 Main task
    - R13 eyecatcher for ‘PCE’

JES2 Exits - Environments

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

JES2 Exits – basic macros

- The structure of an exit (without code or comments)

```
         COPY   $HASPGBL
         $MODULE ENVIRON=xxxx

ROUTINE1 $ENTRY SAVE=YES
         $ESTAE <- not required but recommended
... Insert code here
         $RETURN

         $MODEND
```

JES2 Exits – $XPL

- Many exit (all newer ones) pass an XPL
  - Generally in register 1 but older exits may be in R0 or R2
- XPL has a header to describe the exit being called

```
XPLID    Eye catcher
XPLLEVEL Version number for base section
XPLXITID Exit id number
XPLEXLEV Version number for specific exit
         XnnnVERN is the equate)
XPLIND   Indicator byte – Field for why called
XPLCOND  Condition byte – Bits for conditions
XPLRESP  Response byte  - Bits for exit response
         (Modifiable by Exit routine)
XPLSIZE  Size of parameter list including
         the base section
```

JES2 Exits – $XPL

- Header followed by exit specific fields
  - Label format is XnnnDATA where nnn is the exit number
    - Eg X001JCT
- Equate over XPLIND, XPLCOND, and XPLRESP
  - Eg X001IND, X001COND and X001RESP
- Note that XPLRESP may start off with bits sets
- XPL is more formal way to interface to JES2
  - When possible set XPL field instead of control block field
- JES2 Exits manual describes interface to each exit
  - This includes fields defined in the XPL

JES2 Exits – Control blocks

- JES2 data areas can be expanded by exits
  - Some data areas have user fields built in
    - $USER1-5 in HCT data area
    - CCTUSER1-4 in HCCT data area
    - SJBUSER in SJB data area
    - JCTUSER1-F in JCT data area
  - Other can be extended using services
    - $JCTXxxx service to extend the SPOOLed JCT data area
    - $BERTTAB tables to extend certain checkpointed data areas
  - Some services support use of named tokens
    - $SCANTABs, $PCETAB, etc

JES2 Exits – Control blocks

- Some control blocks are intended just for exit use
  - UCT – pointed by $UCT field in HCT
  - CUCT – pointed by CCTCUCT field in HCCT
  - UPADDR – pointed to by $UPADDR field in HCT
  - DUCT – pointed to by name/token pair (HOME level)
  - DCCT – pointed to by name/token pair (SUBSYS level)
  - Etc (see JES2 Exits manual for more information)
- User related data areas can be defined in $USERCBS
  - Macro supported by $MODULE to define DSECTs

JES2 Exits – Table Pairs

- Not all JES2 customizations are done in exits
- Table pairs extend/customize JES2 processing
  - Pair is a misnomer – Support allows more than 2 tables
    - JES2 and User table (origin of pair concept)
    - PLUS dynamic tables – unlimited number of additional tables
- Table pairs can be used to
  - Add installation commands or keywords to command
  - Override default processing for a command keyword
  - Replace the text in messages issued by $BLDMSG
  - Add structures like PCEs, DTEs, PC routines, etc
  - Extend data area via $BERTTABs

JES2 Exits – Table Pairs

- The MCT is the home for tables that have pairs
- There are 2 major classes of table
  - Commands and their operands ($SCANTABs)
  - Other structures supported by $GETABLE
    - PCE TABLE ($PCETAB)
    - DCT TABLE ($DCTTAB)
    - DTE TABLE ($DTETAB)
    - TRACE ID TABLE ($TIDTAB)
    - PC Routine table ($PCTAB)
    - BERT table ($BERTTAB)
    - Work selection ($WSTAB)
- These can be customized to some extent by installations

JES2 Exits – Table Pairs - $SCANTAB

- Probably the most common to modify
  - Used for commands, initialization statements, messages
- Commands and init statement syntax is in levels

  $T CKPTDEF,CKPT1=(DSN=SYS1.HASPCKPT)
- Components of this command are
  - Verb – $T
  - Object – CKPTDEF
  - Keyword level 1 – CKPT1
  - Keyword level 2 – DSN
- The verb is not controlled by table pairs
- The object and keywords are controlled by tables

JES2 Exits – Table Pairs - $SCANTAB

- Extend or modify a $SCANTAB table use table name
  - Main operand table (CKPTDEF level) – MCTMPSTP
    - Just have to know this one
  - First level keywords (CKPT1 level) – MCTCKTTP
    - SCANTAB= value from CKPTDEF $SCANTAB
  - Second level keywords (DSN level) – MCTKPNTP
    - SCANTAB= value from CKPT1 $SCANTAB
- Add or modify using table name in a table of $SCANTABs

```
USERCKPT $SCANTAB TABLE=(DYNAMIC,MCTCKTTP)
         $SCANTAB NAME=MAXCKPT_4K,CB=HCT,
          CBIND=($CKW,HCT,L),DSECT=CKW,
          FIELD=CKWMAXRC,RANGE=(1,X'FFFFFFFF'),CONV=NUM,
          CALLERS=($SCDCMDS)
         $SCANTAB TABLE=END
```

- This adds a MAXCKPT= keyword to $D CKPTDEF
  - Displays size of the CKPT when all keywords are set to max
    - JQENUM, JOENUM, etc.

JES2 Exits – Table Pairs - $SCANTAB

- To use this extension, add it to a module and load it
- Basic structure is similar to a basic exit

```
          COPY   $HASPGBL
         $MODULE ENVIRON=xxxx


         $SCANTAB  …


         $MODEND
```

- The writing of the $SCANTAB is beyond this presentation
- See http://www.share.org/d/do/7211 for more details

JES2 Exits – Table Pairs - $SCANTAB

- Example from HASX00A sample

```
MAINPARM $SCANTAB TABLE=(DYNAMIC,MCTMPSTP)
         $SCANTAB NAME=TESTDEF,CONV=SUBSCAN,MINLEN=4,
               CMDRDIR=DEF,SCANTAB=(X0TBTSTP,ADDR),MSGID=949,
               CALLERS=($SCIRPL,$SCIRPLC)  INIT STMT
        $SCANTAB TABLE=END
X0TBTSTP $PAIR ,
        $SCANTAB NAME=IRPLERRS,FIELD=CIRFLAG2,CB=PCE,CONV=FLAG,
               VALUE=(YES,0,FF-CIRF2SSE,NO,CIRF2SSE,FF),MINLEN=4,
               CALLERS=($SCIRPL,$SCIRPLC)
        $SCANTAB TABLE=END
```

- Creates new initialization statement TESTDEF
- Creates new keyword on statement IRPLERRS=YES|NO
  - If IRPLERRS=NO, then set a bit CIRF2SSE
  - Bit suppresses WTOR when an initialization error is detected
- Who said sample exits are not useful?

JES2 Exits – Table Pairs – Other tables

- Some messages in JES2 are generated using $SCANTAB
- These also have a table pair for the FULL message
  - Not for keywords or pieces of messages
- This messages are build using the $BLDMSG service
- Can be used to issue new message or replace JES2's
  - Table name is MCTMGTP
- Caution when replacing JES2 processing
  - Can be confusing or could miss service update
- Basic technique
  - Start with JES2 text from HASPMSG or HASCBLDM
  - Copy to your load module and modify $SCANTABs
  - Assemble and load updated message

JES2 Exits – Table Pairs – Other tables

- PCEs/DTEs are another good candidate
  - Can create your own PCE to perform functions
  - Can add a new subtask for special processing
- No need to know table name for these

```
$PCETAB TABLE=DYNAMIC
$DTETAB TABLE=DYNAMIC
```

- Do not recommend overriding JES2 tables
  - Though possible to do
- Need code to actually start the PCE or SUBTASK
  - $PCEDYN and $DTEDYN ATTACH macros should be used
  - Can be placed in $$$$LOAD routines

JES2 Exits – Load and delete routines

- Some load modules need initialization routines
  - Need to set up storage, start PCEs/DTEs, etc
- One solution is to use an initialization exit to do function
  - Exit 24 often used, but can also use exit 0 or 19
- Routine $$$$LOAD get called when module is loaded
  - Must be defined with a $ENTRY
  - Must be in first CSECT in the load module
- Routine runs in the JES2 main task after module is loaded
  - Better because it is called for dynamic load of module
    - $ADD LOAD command

JES2 Exits – Load and delete routines

- Similar to initialization, some modules need cleanup
  - Delete data areas or stop processes that they run
- Again, can be done in termination exit
  - Exit 26 is standard for this
- Routine $$$$DEL get called when module is deleted
  - Must be defined with a $ENTRY
  - Must be in first CSECT in the load module
- Routine runs in the JES2 main task as part of delete
  - Can prevent or delay module deletion
  - Better than exit since it is called for dynamic delete of module
    - $DEL LOAD command

JES2 Exits – Dynamic exits

- The LOADMOD initialization statements loads modules
  - Special case processing exists for exit 0
- The $$$$LOAD routine is called as part of loading module
- Modules can be refreshed by the $T LOADMOD command
  - Assuming they support this function
  - $$$$LOAD from new module is called
- Alternatively, exits can be loaded by $ADD LOADMOD
  - This dynamically brings in a new load module
  - Can be used for temporary functions
    - One time cleanup or setup processing
    - Diagnostic routines
    - Trying new exits in a test LPAR

JES2 Exits – Dynamic exits

- At JES2 termination exits are deleted
  - CSA/LPA exits are explicitly deleted and $$$$DEL called
  - Private exits do not call $$$$DEL routines
- Are also deleted as part of refresh processing
  - Assuming the module supports refresh
  - $$$$DEL from old module called as part of delete
- Can also delete modules that are no longer needed
  - $DEL LOADMOD command deleted module
  - Logically disconnected and eventually deleted
  - $$$$DEL called as part of process

JES2 Exits – Dynamic exits

- Not all exits can be dynamic
  - Too dependent on data areas built at initialization
  - Function too critical to run without
- Dynamic function can be disabled on $MODULE
  - DYNAMIC=NO keyword prevents dynamic processing
- Other considerations exist
  - See http://www.share.org/d/do/5781 for more information

JES2 Exits

- There is a wealth of resources on JES2 exits
  - IBM publication like
    - JES2 Exits manual
    - JES2 Macros
    - JES2 Data areas
  - Sample exits shipped in SYS1.SHASSAMP
  - SHARE presentation
  - IBM education assistant

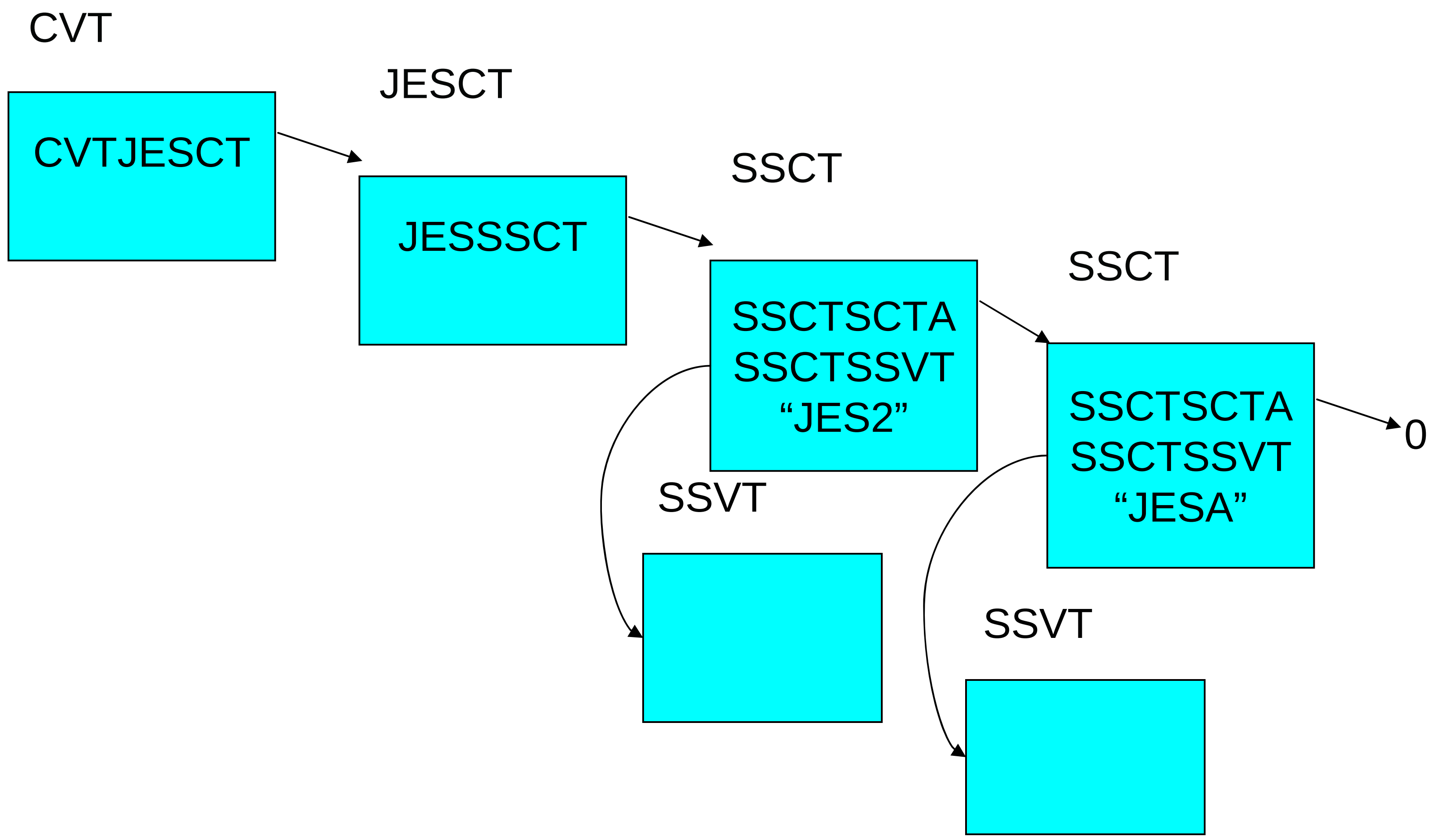
http://publib.boulder.ibm.com/infocenter/ieduasst/stgv1r0/index.jsp
http://publib.boulder.ibm.com/infocenter/ieduasst/stgv1r0/topic/com.ibm.iea.zos/zos/1.7/JobEntrySS/JES2_Exits_Overview.pdf

  - Google searches on JES2 exits
    - Most references were found on google

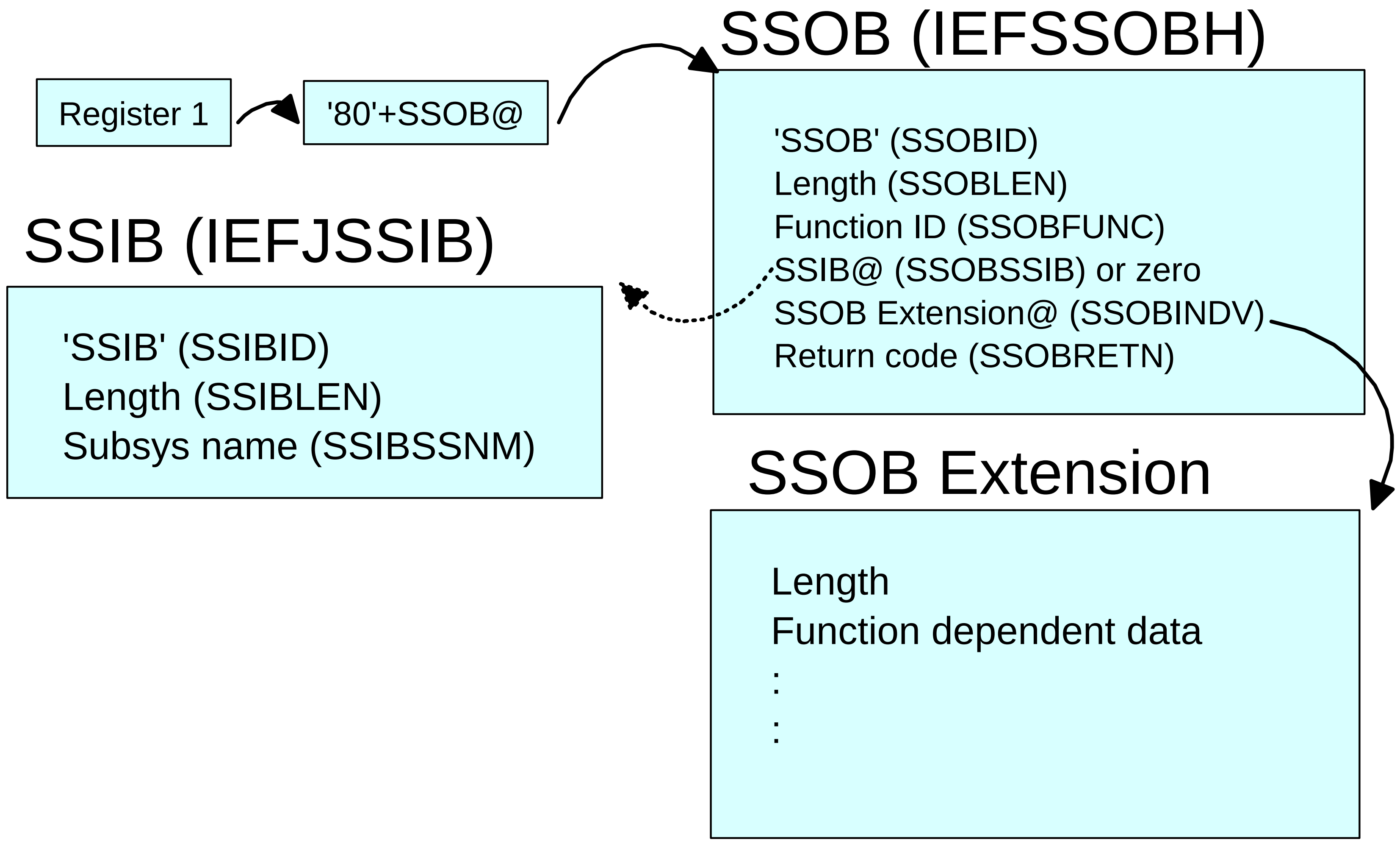
What is the SSI

- The SSI is an MVS interface to "Subsystems"
  - Used as a hook to give info to subsystems
  - WTO, CMDs, EOT, EOM, etc.
  - Used as a way to request functions
    - PSO, SAPI, Extended Status
  - Each SSI has a number and an SSOB extension
  - Subsystem identifies what it supports
  - Caller can specify subsys to process request
    - Default, Specific, All

Basic data structure



Invoking the SSI - Data areas



Register 1

'80'+SSOB@

SSOB (IEFSSOBH)

'SSOB' (SSOBID)
Length (SSOBLEN)
Function ID (SSOBFUNC)
SSIB@ (SSOBSSIB) or zero
SSOB Extension@ (SSOBINDV)
Return code (SSOBRETN)

SSIB (IEFJSSIB)

'SSIB' (SSIBID)
Length (SSIBLEN)
Subsys name (SSIBSSNM)

SSOB Extension

Length
Function dependent data
:
:

Invoking the SSI - Code

```
         USING SSOB,MYSSOB         Establish SSOB addressability
         SPACE 1
         XC    MYSSOB,MYSSOB       Zero SSOB area
         LA    R6,MYSSOB           Get address of SSOB
         SPACE 1
         MVC   SSOBID,=C'SSOB'     Set SSOB eyecatcher
         MVC   SSOBLEN,=Y(SSOBHSIZ)  Set length of SSOB header
         MVC   SSOBFUNC,=Y(SSOBSSxx)  Set function code
         MVC   SSOBSSIB,=F'0'      Use LOJ SSIB
         LA    R0,SSOB+SSOBHSIZ    Point to SSOB extension
         ST    R0,SSOBINDV         Point base to extension
         SPACE 1
         USING SSxxxxx,SSOB+SSOBHSIZ  SSOB extension addr'blty
         SPACE 1
* Code to set up SSOB extension goes here
         SPACE 1
         LA    R6,MYSSOB           Point to SSOB
         O     R6,=X'80000000'     Set HI BIT to indicate last
         ST    R6,PARMPTR          Save SSOB address in parm
         LA    R1,PARMPTR          Get pointer to SSOB
         SPACE 1
         IEFSSREQ                  Invoke the SSI
         SPACE 1
         LTR   R15,R15             If this is nonzero
         JNZ   SSREQERR             we're in big trouble
         CLC   SSOBRETN,=A(0)        Is there an error?
         JH    SSOBERR                 Yes, process error
```

What is the SSI (cont...)

The SSI calls (that applications can use) which JES supports are:

Number	Symbol	Macro	Description
1	SSOBSOUT	IEFSSSO	Process SYSOUT
2	SSOBCANC	IEFSSCS	Job cancel
3	SSOBSTAT	IEFSSCS	Job status
11	SSOBUSER	IEFSSUS	Destination validation
20	SSOBRQST	IEFSSRR	Request job ID
21	SSOBRTRN	IEFSSRR	Return job ID
54	SSOBSSVI	IEFSSVI	Subsystem information
70	SSOBSFS	IAZSSSF	SJF spool services
71	SSOBSSJI	IAZSSJI	Job information (Much of this SSI has been depricated)
75	SSOBSSNU	IAZSSNU	User notification
79	SSOBSOU2	IAZSSS2	SYSOUT API (SAPI)
80	SSOBESTA	IAZSSST	Enhanced status information
82	SSOBSSJP	IAZSSJP	JES properties
83	SSOBSSJD	IAZSSJD	JES device information
85	SSOBSSJM	IAZSSJM	Job modify