Bi-JESual SPOOL Browse: REVISITED

James Lund – Computing and Information Services
Texas A&M University (TE)

Adam Nadel – JES2 Level 2 Team Lead
IBM – Poughkeepsie, NY

August 6, 2014
Session Number 16168

#SHAREorg

SHARE in Pittsburgh 2014

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:

Performance is in Internal Throughput Rate (ITR) ratio based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput improvements equivalent to the performance ratios stated here.
IBM hardware products are manufactured from new parts, or new and serviceable used parts. Regardless, our warranty terms apply.
All customer examples cited or described in this presentation are presented as illustrations of the manner in which some customers have used IBM products and the results they may have achieved. Actual environmental costs and performance characteristics will vary depending on individual customer configurations and conditions.
This publication was produced in the United States. IBM may not offer the products, services or features discussed in this document in other countries, and the information may be subject to change without notice. Consult your local IBM Business contact for information on the product or services available in your area.
All statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only.
Information about non-IBM products is obtained from the manufacturers of those products or their published announcements. IBM has not tested those products and cannot confirm the performance, compatibility, or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.
Prices subject to change without notice. Contact your IBM representative or Business Partner for the most current pricing in your geography.

Acknowledgements

Thanks to **Richard Peurifoy** (TAMU), **David Jones** (IBM JES3), and **Tom Wasik** (IBM JES2) for their valuable contributions to this revised and enhanced presentation.

About the Speakers

- James Lund
  - Texas A&M University ’87 - Computer Science
  - 30 years working with MVS technology
  - 20+ years at SHARE
  - 15+ years DPM in the JES3/EPS Project
  - JES3 Committee Lead under MVS Core Technologies Project

- Adam Nadel
  - Binghamton University ’04 - Computer Science
  - 10 years working with MVS technology
  - IBM z/OS Level 2 Support
    - JES2 (team lead), SDSF
    - z/OSMF REST jobs API (team lead), z/OSMF SDSF UI

About the Company

- **Texas A&M University** formed in 1876 as Texas' first public institution of higher learning
- 50,000 undergraduate and 8,500 graduate students
- 250 degree programs
- 10 colleges – 6th largest enrollment in nation.

- Two branch campuses (Galveston, Tx and Doha, Qatar) and overseas centers (Mexico, Costa Rica, and Italy)
- George Bush (Sr.) Presidential Library/School of Government

IBM 704

IBM 7094 Data Processing System, IBM 726 Tape

Our Environment

- Four FTEs for system and subsystems support
- IBM z10 BC
- z/OS v1.13, JES3 v1.13
- Who are our customers?
  - Budget/Payroll System (BPP)
  - System-wide Financials (FAMIS) – 20+ universities and state agencies
    - 30,000 faculty and staff

Current Conditions (Review)

A concurrent move to a new z/OS and JES is difficult!

- 13 JES local usermods
- 22 site-developed macros

- 4 local DSP Dictionary modules
- 1 local FCT module, driving 4 function modules
  - MUSAS (Wylbur) JES Spool interface
- JES3 USERMOD required to implement - IATGRPT

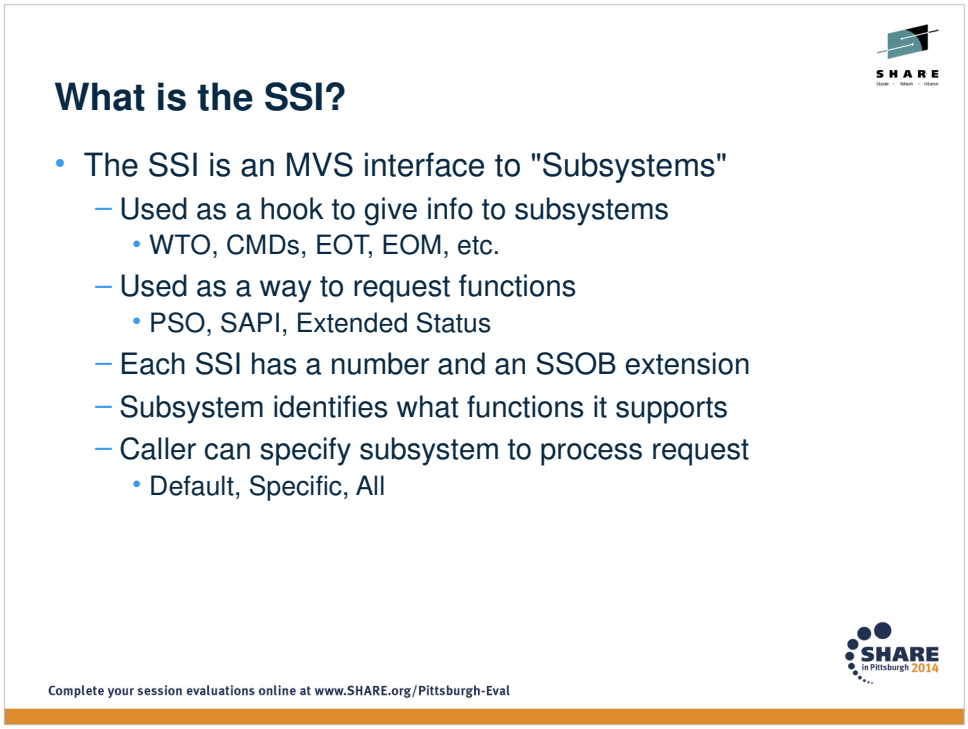
- **See SHARE 2011 Orlando Session 9716 Extended Status Spool Browse**

Local FCT

```
*-------------------------------------------------------------*
*         DEFINE LOCAL FCTS                                   *
*-------------------------------------------------------------*
TIJPFCT  IATYFCD ECFMASK=FF,ECFADD=IATGRJR,R14=IATGRJR,
               R15=IATGRJR,COND=80,DRVR=IATUMIJ,INISH=YES,
               PRTY=6,SUCC=FSFCT,NAME=TAMUIJP,PREV=GRSRFCT
```

Local DSPs

```
*-------------------------------------------------------------*
*        DEFINE LOCAL DSP ENTRIES                             *
*-------------------------------------------------------------*
OPERDSP  IATYDSD PRTY=1,XABLE=YES,MUCC=NO,DRVR=TAMDS77,NOREQ=1,
               REQ=(CNS3277),MAXCT=2
IOSCREEN IATYDSD PRTY=1,XABLE=YES,MUCC=NO,DRVR=TAMIOSC,NOREQ=1,
               REQ=(CNS3277),MAXCT=2
JOBCARD  IATYDSD PRTY=1,XABLE=YES,MUCC=NO,DRVR=TAMUTJC
WTJ      IATYDSD PRTY=1,XABLE=YES,MUCC=NO,DRVR=TAMWTJB
* SPOOLCHK IATYDSD PRTY=5,XABLE=YES,MAXCT=1,REENT=NO,DRVR=MSPOL
* JES3ARTS IATYDSD PRTY=5,REENT=YES,XABLE=YES,NOREQ=1,
*                REQ=(JS3ARTS),DRVR=U110DJ
* SAG ENTIRE SYSTEM SERVER
UQJ3     IATYDSD PRTY=10,XABLE=YES,DRVR=IATUQJ3
DMYDSP01 IATYDSD
DMYDSP02 IATYDSD
DMYDSP03 IATYDSD
```

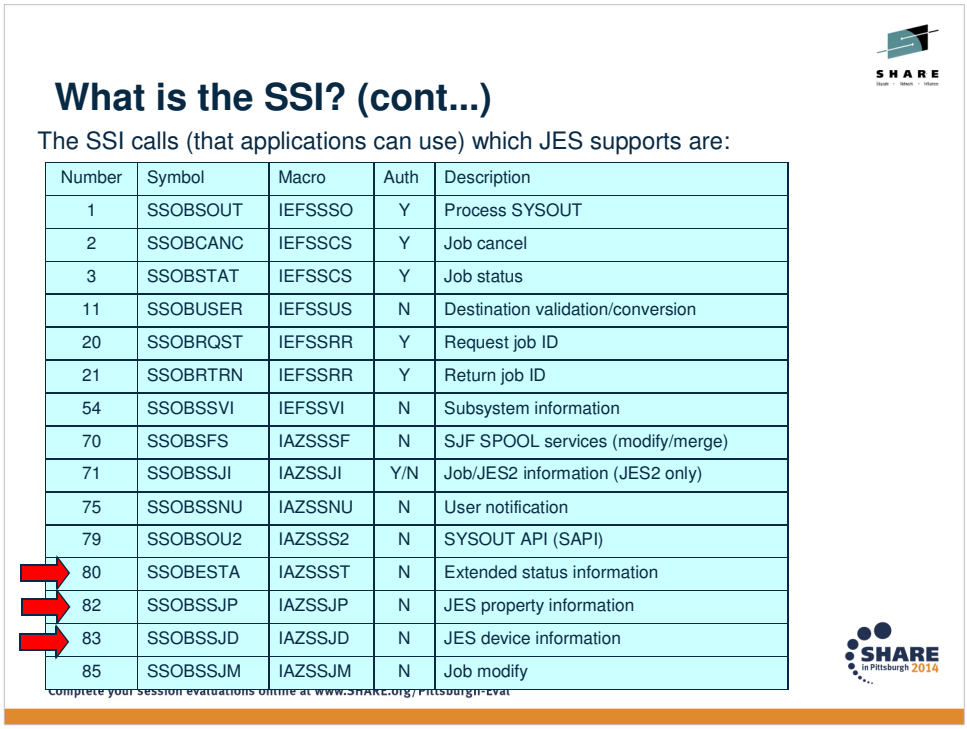


The SSI is an MVS interface to subsystems. A subsystem in this context is defined as any program that responds to SSI requests. JES2 and JES3 are two of the major users of the SSI interface. The SSI functions as both an hook that provides information to the subsystems when certain events occur, as well as a way to request information/services from a subsystems. WTO, command, End of task, End of Memory are all examples of SSIs that are invoked by MVS to tell a subsystem that something has happened. These SSIs are intended to only be issued by MVS and listened to by subsystems. PSO, SAPI, Extended Status are all examples of SSIs that are invoked by applications that are requesting services from a subsystem.

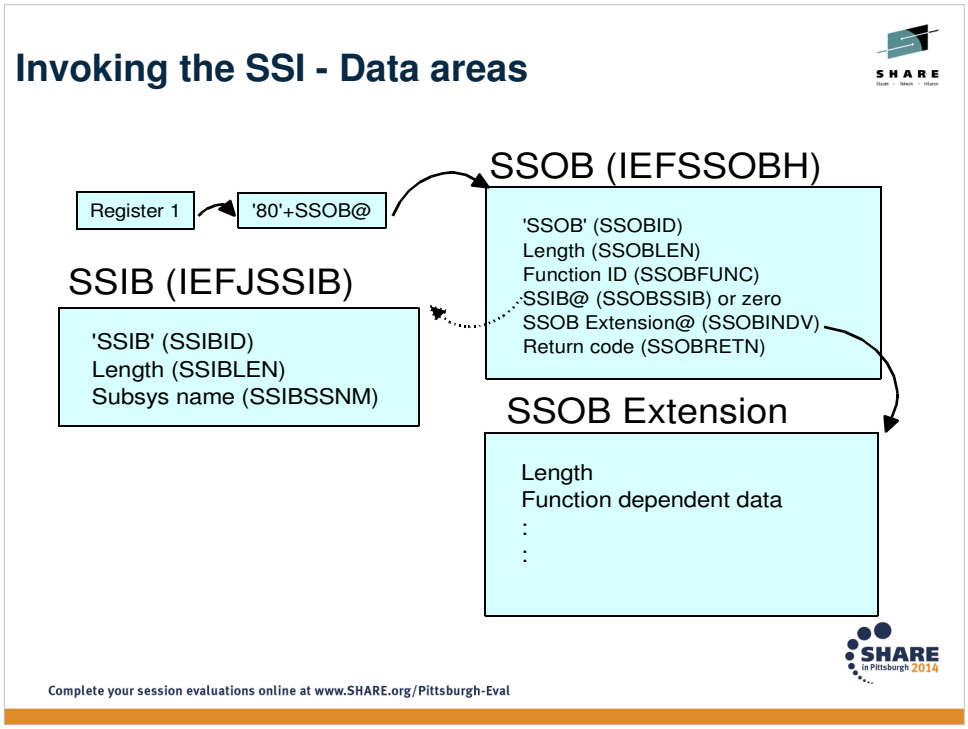
Each SSI has associated with it a number and an SSOB extension. The numbers (normally stated in decimal) ensures that the proper function is requested. The SSOB extension is where the parameters for the specific SSI are defined.

Each subsystem must identify to MVS what SSI numbers (function codes) it supports. The next chart lists the function that JES supports (for use by applications).

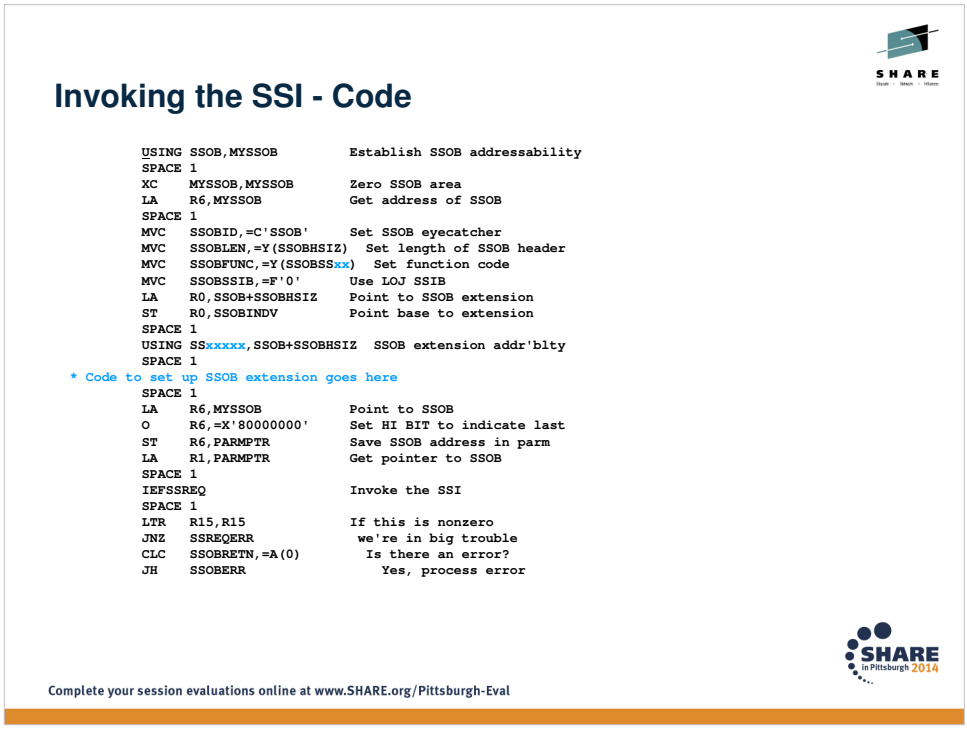
SSI calls can be directed to the default subsystem (the one the application was started under), a specific subsystem, or all subsystems. Sending a request to all subsystems is called a broadcast SSI. Only certain SSIs support being broadcast. The only JES SSI available to applications that can be broadcast is the extended status SSI.



This table lists the SSI request that are available to applications that are supported by JES2 and JES3. Newer SSIs have the higher numbers. Some of these SSIs are documented in the z/OS V1R13.0 MVS Using the Subsystem Interface book (SA22-7642-12). However, most of the newer SSIs have fairly complete documentation in their SSOB extensions (Macro column in the table). The Auth column indicates if the caller of the SSI needs to be authorized. SSI 71 (job/JES2 information SSI) is only supported by JES2. Most function of SSI 71 do not require the caller to be authorized but one function does.

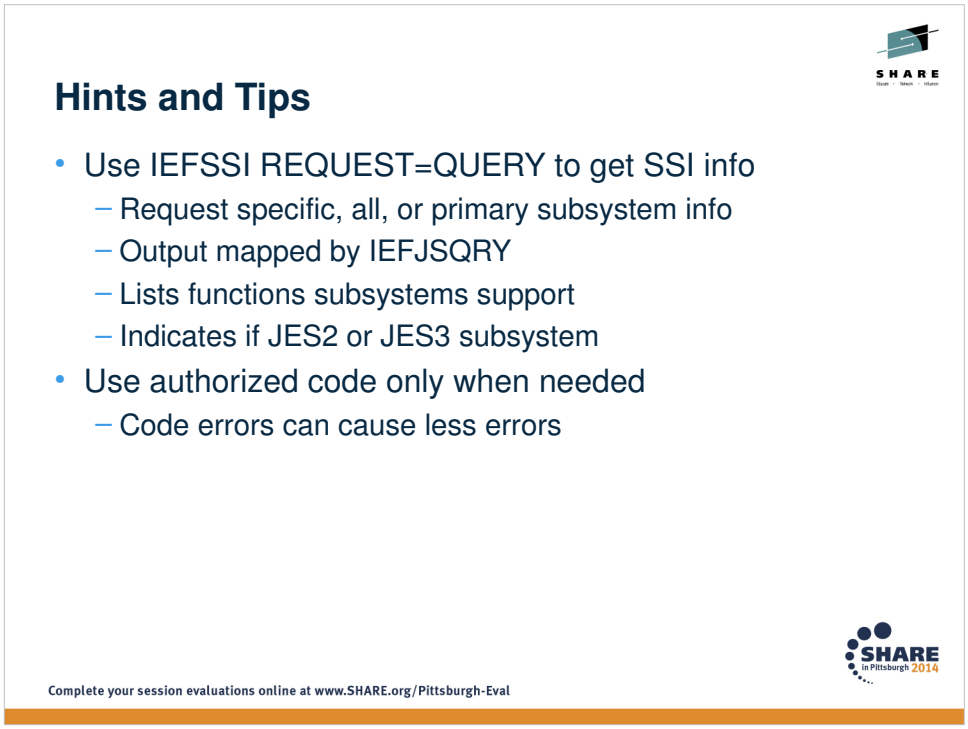


The major data areas that must be filled in to invoke the SSI are the SSOB and the SSOB extension. If you want to direct the request to a specific subsystem, then you can also pass an SSIB on the request. The SSOB extension that is used will depend on the function ID (SSI number) being used.



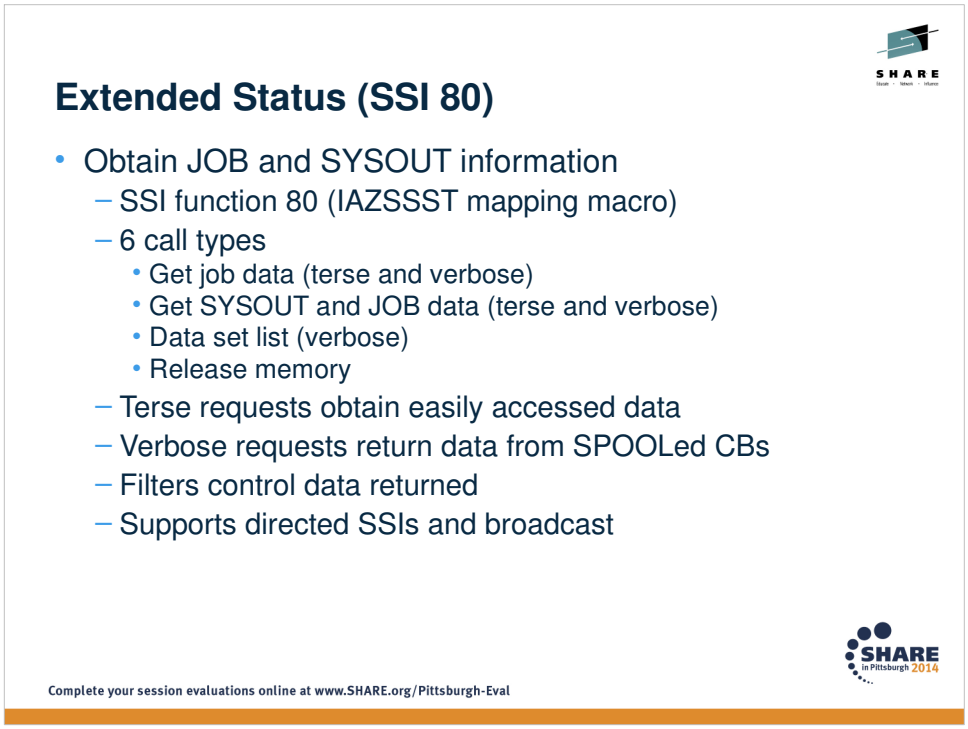
This is the basic code needed to invoke any SSI request. This code sends the request to the subsystem associated with the address space (uses the life of job - LOJ SSIB). This SSIB points to the subsystem that started the address space. If the address space was started under the master subsystem (does not have a job structure in JES or used request job id), then the request will go to the MSTR subsystem. If it was started under JES2/JES3 (has a job structure that is not from request jobid) then the request will go to the JES instance that started the address space.

Notice that after the call, there are 2 return codes being checked. The R15 value after the call to IEFSSREQ is a function independent return code defined in IEFSSOBH. These return code are often not set by the subsystem itself but rather by the IEFSSREQ logic. The SSOBRETN is a function dependent return code that is defined in the individual SSOB extensions. These are only set by the subsystems. Often there will be a third return code (or a reason code) in the SSOB extension itself to further identify the cause of an error.



When coding general interfaces into JES, it is often interesting to know if you are interfacing with JES2 or JES3. Or you may need to know what subsystems exist on your system. The easiest way to do this is to use the QUERY request on the IEFSSI macro. This can give you information on all the subsystems that are defined on your system, or information on a particular subsystem (including the primary subsystem). It returns information on which SSI function numbers are supported and whether the subsystem is JES2 or JES3.

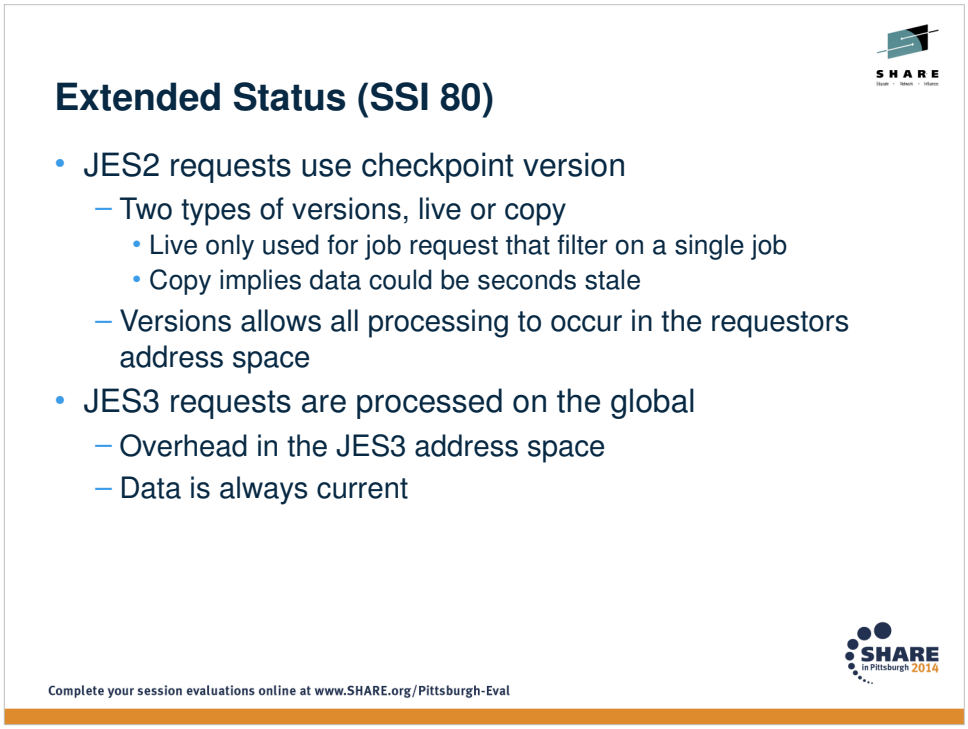
Another helpful word of advice is to avoid running authorized as much as possible. This is more for the sake of others rather than yourself. An authorized program that has an error can cause damage to the system. Unauthorized code is much less likely to mess things up outside your address space.



The extended status SSI returns information about job and SYSOUT in the JES queue. There are 6 functions supported, 5 to obtain information and one to return the storage obtained. Two of the functions, referred to as terse requests, obtain information from mostly instorage control blocks with minimal SPOOL I/O (in JES2 there is no SPOOL I/O for terse requests). The other 3 obtain information from SPOOL data areas and are referred to as verbose requests. There are terse and verbose requests for JOB data, and terse and verbose requests to get SYSOUT (and job) data, and a verbose request to get a list of all JES data sets (input and output) associated with a job. As is typical of the newer SSIs, the storage for the return data is managed by the SSI. It is obtained on functions that get data and then freed by a subsequent memory management call. You will see this on many of the SSI calls.

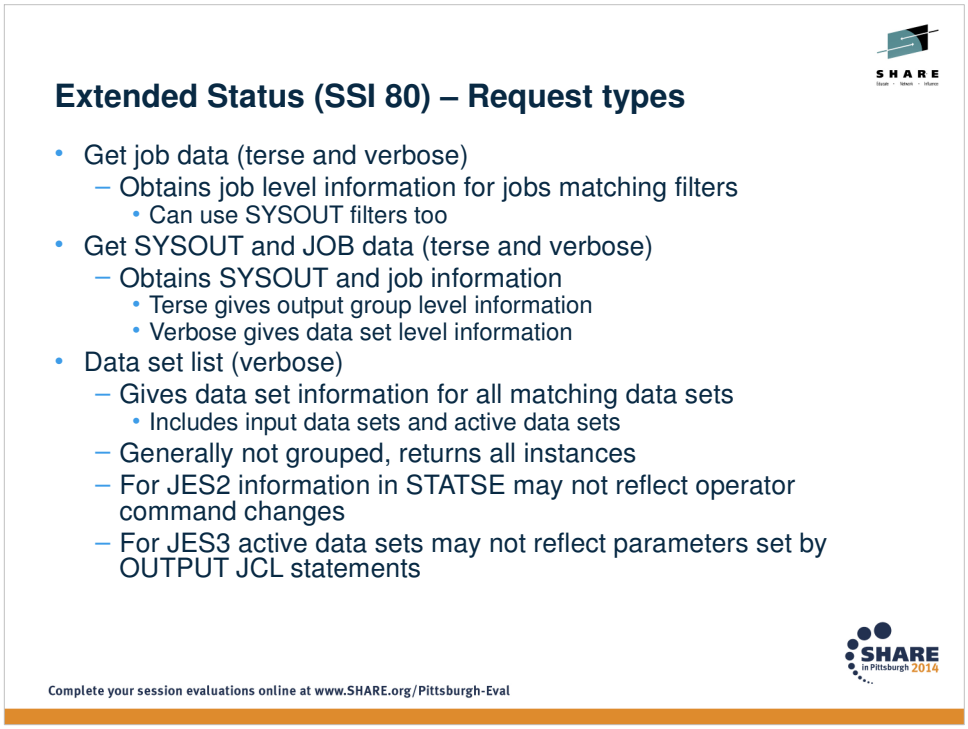
The requester can filter the data returned based on a wide range of JOB as well as SYSOUT filters.

The SSI supports both directed and broadcast requests. Directed request implies that you do not have to be running under the target subsystem to make this request. Since this SSI also supports a broadcast request, you can ask all subsystems (all JES subsystems) on a system to return data in one call.



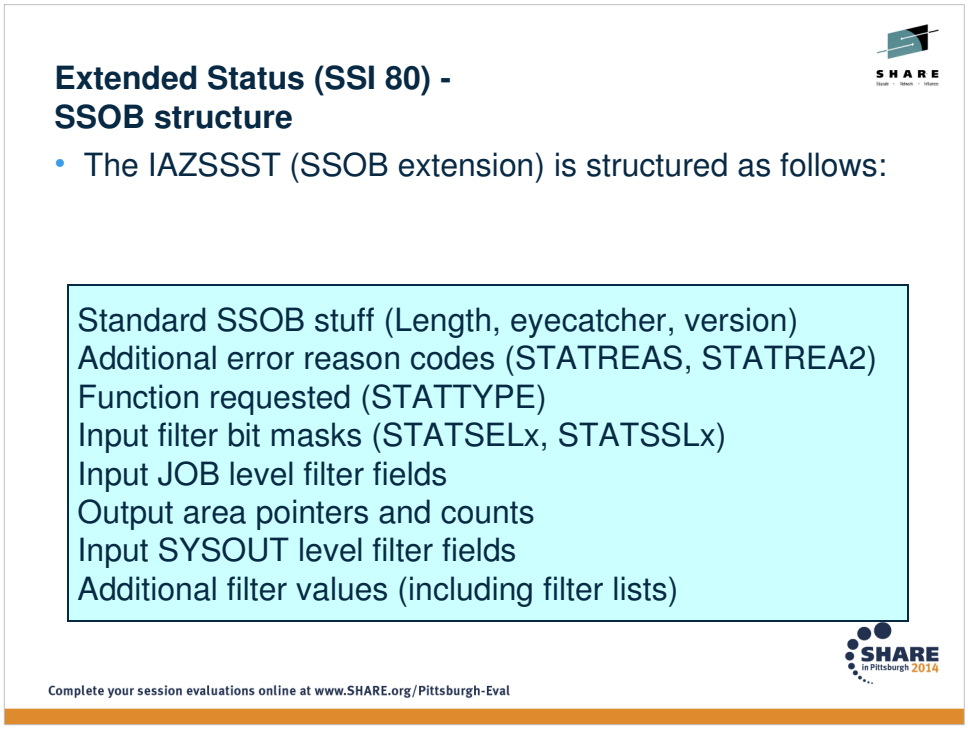
JES2 obtains the data returned on these requests from a copy of the checkpoint that lives in a data space. This can be a static point in time copy or a live copy of the checkpoint data. If a static copy is used, the data can be up to 5 seconds stale relative to what JES2 commands would indicate (this is an extreme value for an idle system and in reality it is as stale as the HOLD= value on MASDEF). Live copies have current information but are only used when a job level information is requested for a single job (not SYSOUT information). Using a version allows the request to be processed in the requestor's address space without getting the JES2 address space involved.

JES3 requests are processed on the JES3 global. This allows JES3 to provide information that is current at the time it was retrieved. However extended status request must compete with resources on the global.

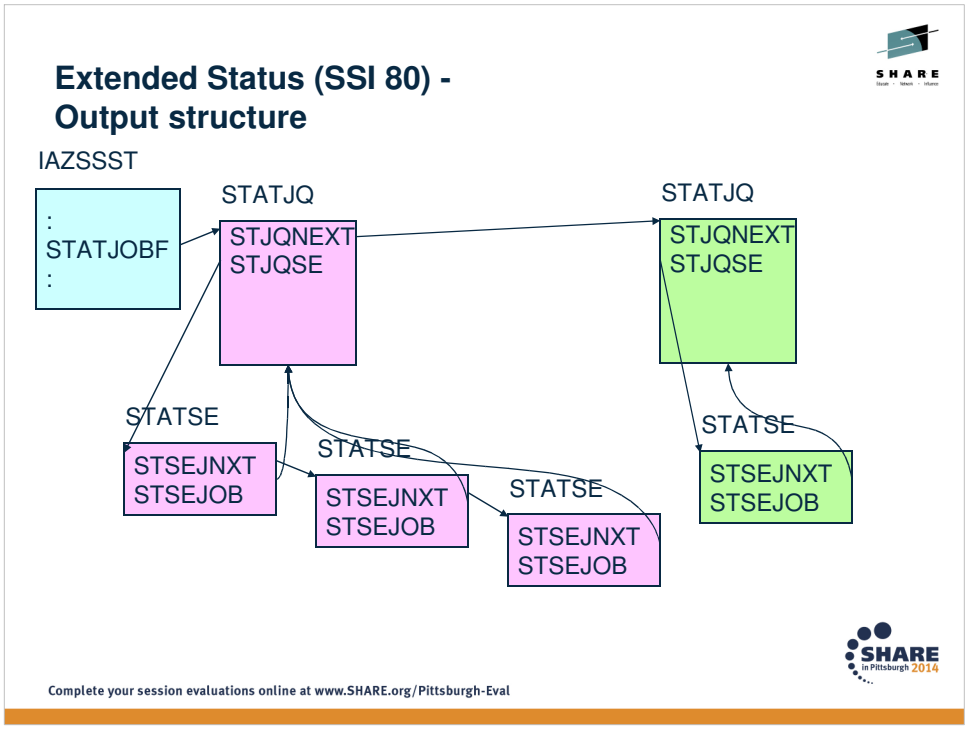


There are 3 major request types, job, SYSOUT and data set list. The job requests return information on the jobs in the system. Terse returns information that is readily available and the verbose returns more details. SYSOUT requests return information on SYSOUT groups (terse request) and include the information on the data sets in the group when a verbose request is made. The data set list request returns all the JES data sets for the job including input (instream) data sets and SYSOUT data sets that are still active (have not been through output services). The data set list function returns all instances of a data set so there may be duplicate entries for one data set, each with different characteristics. Restriction on the data set list function:

- the data returned may not reflect changes made by operator commands (JES2)
- the data returned may not reflect options set in OUTPUT statements for active data sets (JES3).

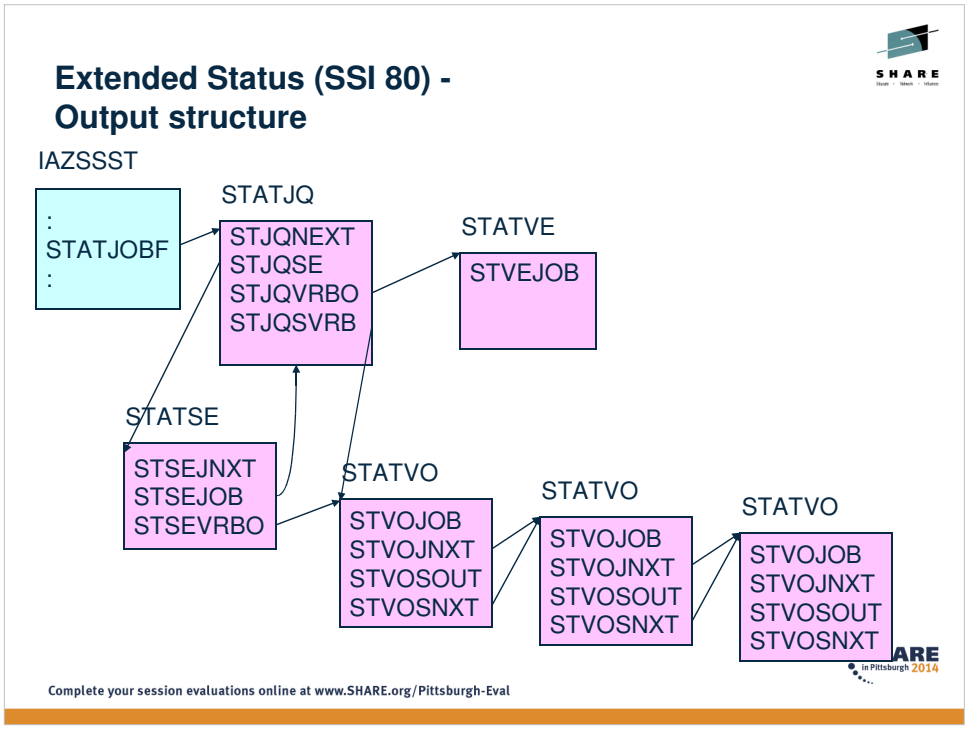


The SSOB extension is mapped by IAZSSST. The extension is made up of a number of sections, each representing a different function. Filtering is accomplished by setting a bit to activate the filter and then setting a corresponding field to the value (or a pointer to a list of values) to filter on. Lists are supported for job name, ID, class, phase, default destination and SYSOUT class and destination. Many filters support generic characters (* and ? or application specifiable). The results are returned in an output areas are chained into the SSOB extension.



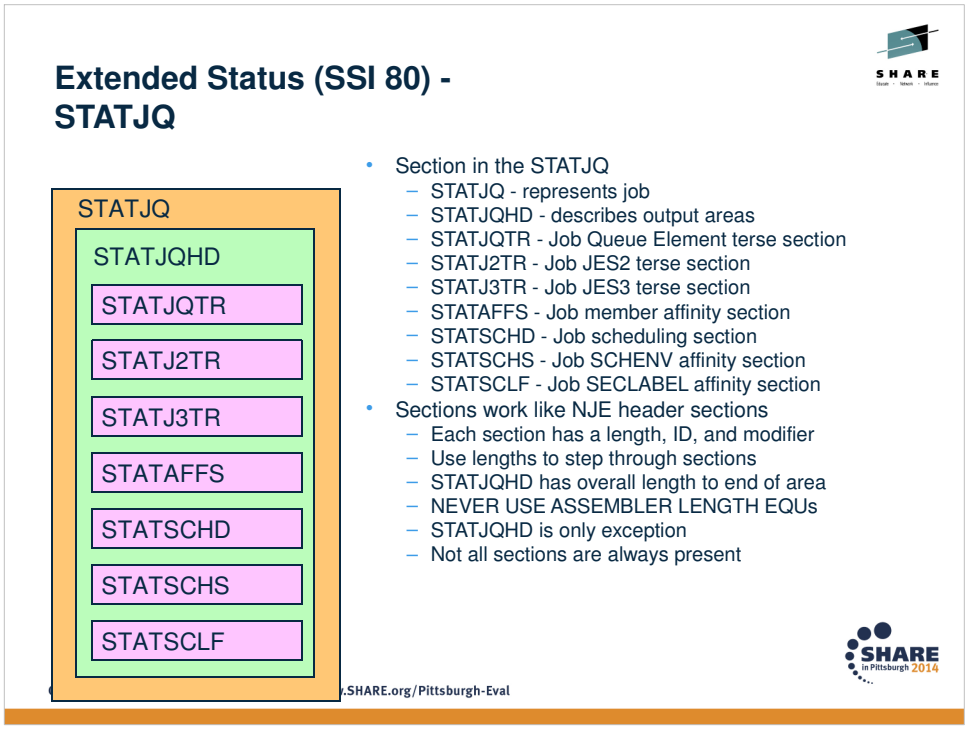
The output areas returned by extended status are pointed to by STATJOBF in the SSOB extension (IAZSSST). For terse requests, there are 2 types of output areas. STATJQs represent a job (JQE). For every job which matched the filter criteria, a STATJQ is built.

STATSE represent an output group (JOE). The STATSEs are chained out of the STATJQ (so if you ask for SYSOUT information, you will always get STATJQs too). The STATSEs point back to the STATJQs that own them.



Verbose requests return additional data areas for the job and SYSOUT. Verbose requests are limited to a single job at a time. For each STATJQ returned, a STATVE contains information that is stored in the JCT. If SYSOUT verbose data is requested, then each STATSE (JOE level data area) has 1 or more STATVOs chained to it. Each STATVO represents a data set (PDDB) that is associated with the JOE.

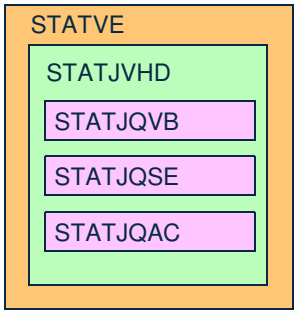
Verbose data can be requested as part of the original request or can be added to the output of an existing request by passing a STATJQ or STATSE address in STATTRSA on a subsequent request.



The STATJQ contains the terse job information and is composed of a number of sections. Each section has identifying information and a section length (the exception is the STATJQHD section). The high level DSECT (STATJQ) has the pointers to the next STATJQ, a pointer to any STATSEs (SYSOUT terse areas), a pointer to the STATVE (job verbose areas), and a pointer to any STATVO sections (SYSOUT verbose areas).

The length of the STATJQ header is stored in STJQOHDR. Add this length field to the STATJQ and you point to the STATJQHD. This is a header for the remaining fields. STHDLEN (in STATJQHD) has the overall length of the remaining areas. This length is used to determine when you have reached the end of the variable sections. You add the STATJQHD length equate (STHDSIZE) to the address of the STATJQHD to get the first variable section. Each variable section starts with a 2 byte length (STxxLEN), a 1 byte ID fields (STxxTYPE) and a 1 byte modifier (STxxMOD). When scanning for or identifying a section, ensure you check both the type AND modifier to determine what section this is. To get to the next section, add the STxxLEN field to the current section pointer. Not all sections are present for all jobs. In addition, maintenance or a new release can add new section types or modifiers to existing types. Ensure your application can handle unknown types.

Extended Status (SSI 80) - STATVE

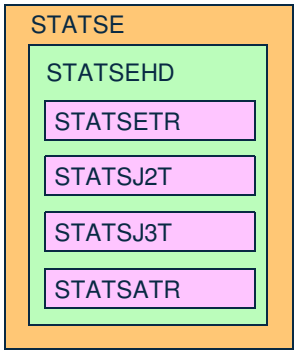


- Section in the STATVE
  - STATVE - represents job's JCT data
  - STATJVHD – describes output areas
  - STATJQVB – Job verbose section
  - STATJQSE – Job security section
  - STATJQAC – Job accounting section

The STATVE contains job verbose information and is composed of a number of sections. Each section has identifying information and a section length (the exception is the STATJVHD section). The high level DSECT (STATVE) has a pointer back to the owning STATJQ section.

The length of the STATVE header is stored in STVEOHDR. Add this length field to the STATVE and you point to the STATJVHD. This is a header for the remaining fields. STJVLEN (in STATJVHD) has the overall length of the remaining areas. This length is used to determine when you have reached the end of the variable sections. You add the STATJVHD length equate (STJVSIZE) to the address of the STATJVHD to get the first variable section. Each variable section starts with a 2 byte length (STxxLEN), a 1 byte ID fields (STxxTYPE) and a 1 byte modifier (STxxMOD). When scanning for or identifying a section, ensure you check both the type AND modifier to determine what section this is. To get to the next section, add the STxxLEN field to the current section pointer. Not all sections are present for all jobs. In addition, maintenance or a new release can add new section types or modifiers to existing types. Ensure your application can handle unknown types.

Extended Status (SSI 80) - STATSE

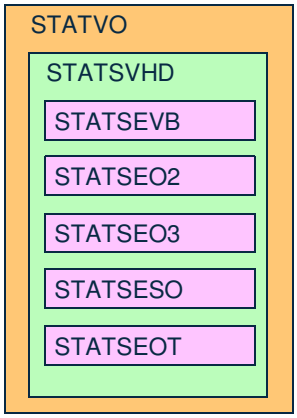


- Section in the STATSE
  - STATSE - represents SYSOUT group
  - STATSEHD - describes output areas
  - STATSETR - SYSOUT element terse section
  - STATSJ2T - SYSOUT JES2 terse section
  - STATSJ3T - SYSOUT JES3 terse section
  - STATSATR – Transaction information section

The STATSE contains SYSOUT information for a collection of data sets (JOE in JES2, OSE variable section with up to 16 data sets in JES3) and is composed of a number of sections. Each section has identifying information and a section length (the exception is the STATSEHD section). The high level DSECT (STATSE) has the pointers to the next STATSE, a pointer back to the job level STATJQ, and pointers to any STATVO sections (SYSOUT verbose areas).

The length of the STATSE header is stored in STSEOHDR. Add the length field to the STATSE and you point to the STATSEHD. This is a header for the remaining fields. STSHLEN (in STATSEHD) has the overall length of the remaining areas. This length is used to determine when you have reached the end of the variable sections. You add the STATSEHD length equate (STSHSIZE) to the address of the STATSEHD to get the first variable section. Each variable section starts with a 2 byte length (STxxLEN), a 1 byte ID fields (STxxTYPE) and a 1 byte modifier (STxxMOD). When scanning for or identifying a section, ensure you check both the type AND modifier to determine what section this is. To get to the next section, add the STxxLEN field to the current section pointer. Not all sections are present for all SYSOUT areas. In addition, maintenance or a new release can add new section types or modifiers to existing types. Ensure your application can handle unknown types.

Extended Status (SSI 80) - STATVO

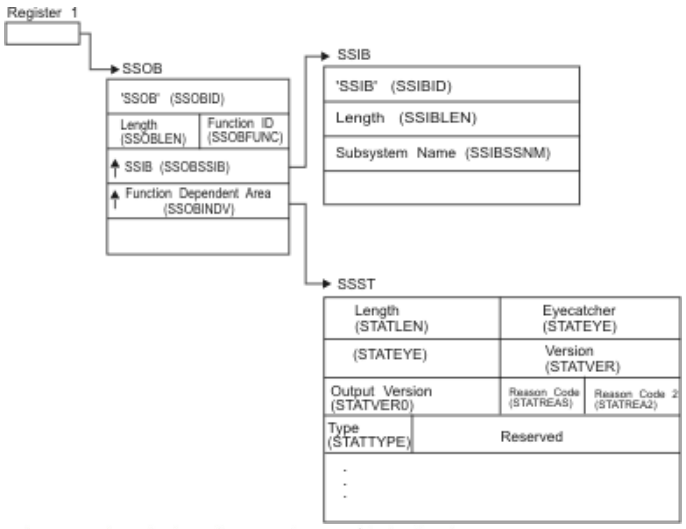


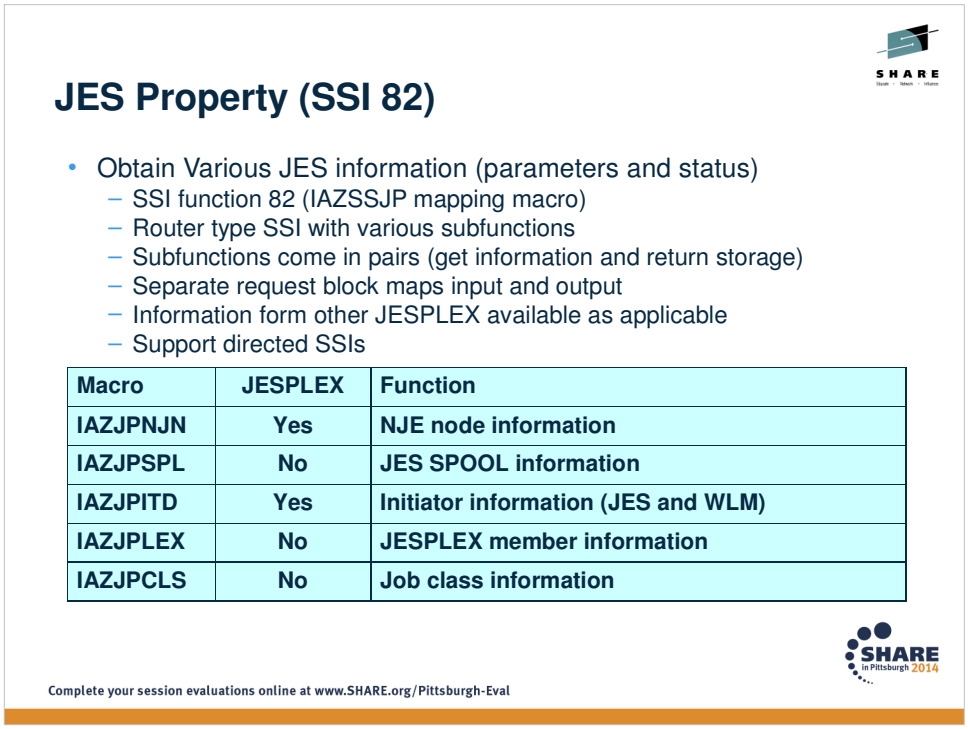
- Section in the STATVO
  - STATVO - represents SYSOUT data set (PDDB)
  - STATSVHD - describes output areas
  - STATSEVB - SYSOUT data set verbose section
  - STATSEO2 - SYSOUT data set JES2 verbose section
  - STATSEO3 - SYSOUT data set JES3 verbose section
  - STATSESO - SYSOUT data set security section
  - STATSEOT – Transaction (APPC) output section

The STATVO contains data set level SYSOUT information (JES2 PDDB) and is composed of a number of sections. Each section has identifying information and a section length (the exception is the STATSVHD section). The high level DSECT (STATVO) has a pointer back to the job level STATJQ, a pointer to the next STATVO off the STATJQ, a pointer back to the SYSOUT level STATSE, and a pointer to the next STATVO off the STATSE.

The length of the STATVO header is stored in STVOOHDR. Add the length field to the STATVO and you point to the STATSVHD. This is a header for the remaining fields. STSVLEN (in STATSVHD) has the overall length of the remaining areas. This length is used to determine when you have reached the end of the variable sections. You add the STATSVHD length equate (STSVSIZE) to the address of the STATSVHD to get the first variable section. Each variable section starts with a 2 byte length (STxxLEN), a 1 byte ID fields (STxxTYPE) and a 1 byte modifier (STxxMOD). When scanning for or identifying a section, ensure you check both the type AND modifier to determine what section this is. To get to the next section, add the STxxLEN field to the current section pointer. Not all sections are present for all SYSOUT areas. In addition, maintenance or a new release can add new section types or modifiers to existing types. Ensure your application can handle unknown types.

Environment for SSI 80 Call





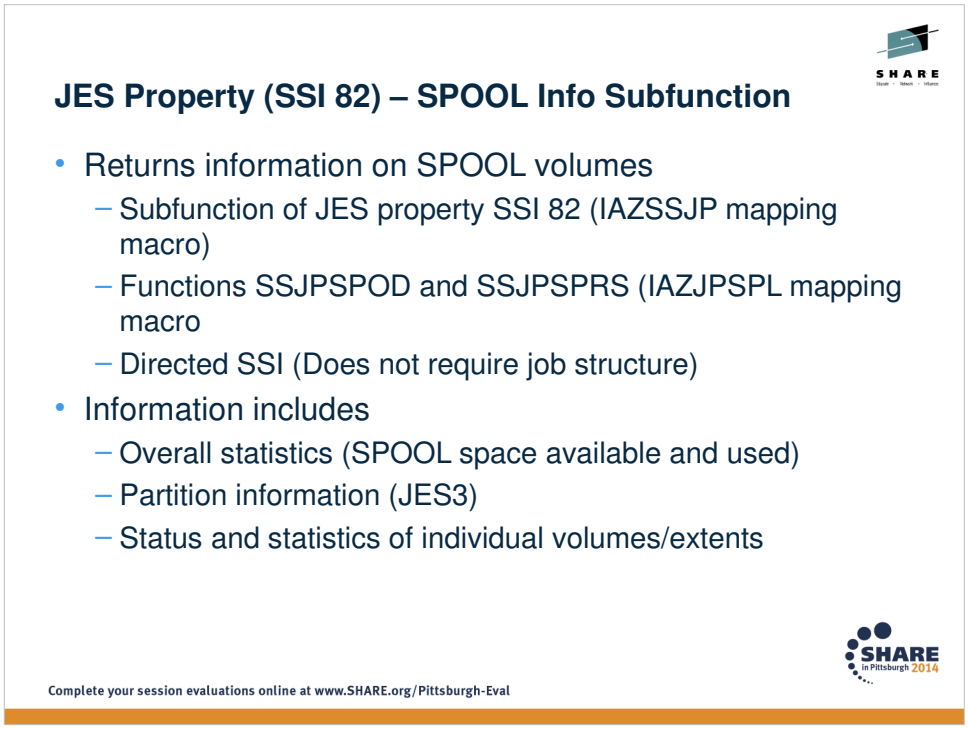
The JES property SSI is a router SSI that returns information on various JES parameters. It is intended that this information be available in a JES independent manner when possible. The SSOB extension for this SSI is IAZSSJP. There are 5 types of information that can be obtained each having a pair of function codes, one to get information and one to return the storage from a prior request. There are different mapping macros for each type of information that can be obtained. In the cases where it applies, it is possible to obtain the information from the perspective of another member of the JESPLEX.

JES Property (SSI 82) - SSOB structure

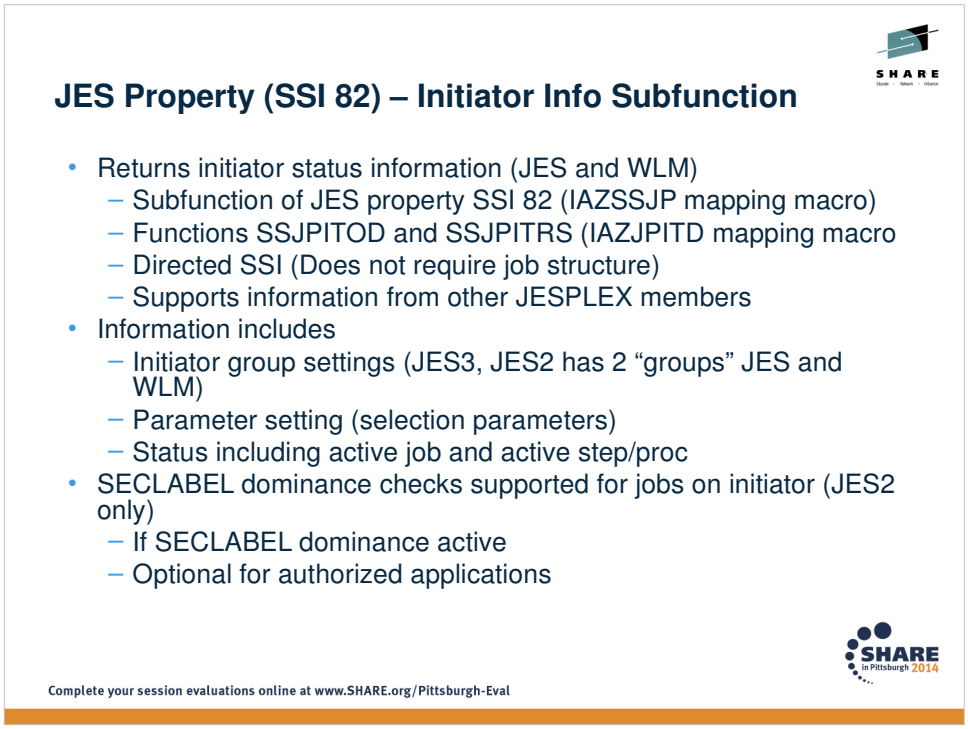
- The IAZSSJP (SSOB extension) is structured as follows:

Standard SSOB stuff (Length, eyecatcher, version)
Function being requested
SSJPRETN – router and subfunction return code
Pointer to function dependent data area

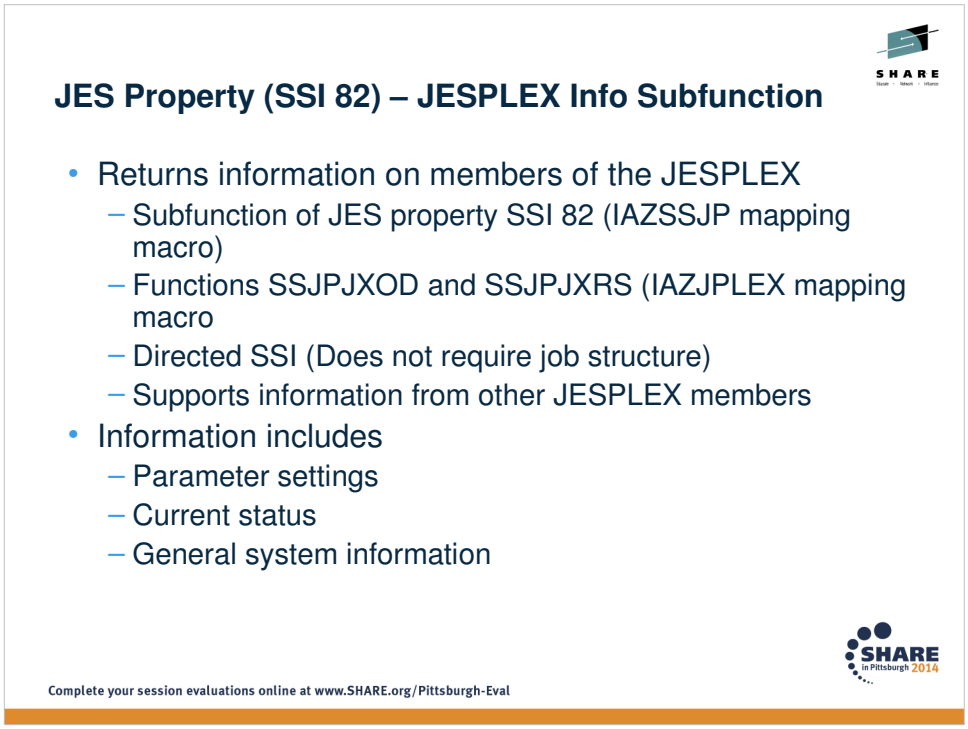
The SSOB extension is mapped by IAZSSJP. The extension is essentially a standard extension with a function request byte, data area pointer, and extended return code. It is the function depended area that has most of the interesting information



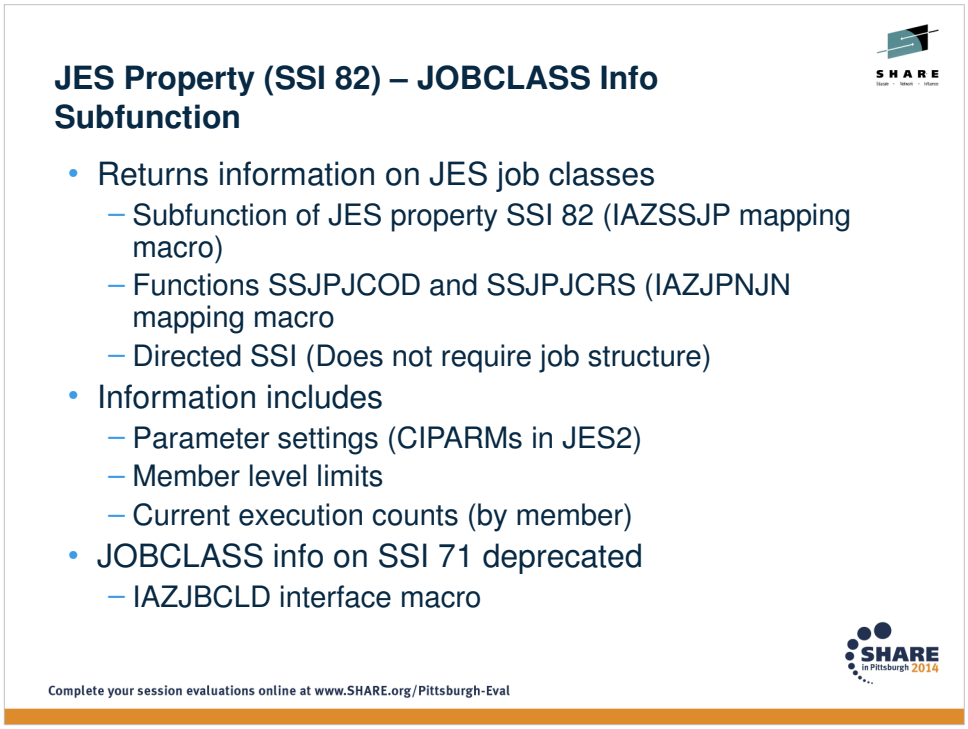
The SPOOL subfunction of the JES property SSI returns information on overall SPOOL space and individual volumes defined to JES. Information is JESPLEX in nature since SPOOL space is defined to the JESPLEX. Information is broken down at the JEXPLEX level, the SPOOL partition level (JES3) and the individual extent/volume level. Even though JES2 does not support SPOOL partitions, the output is presented as if all JES2 SPOOL space was in a single partition. This simplifies processing the output of this request.



The initiator subfunction of the JES property SSI returns information on initiators defined and active in the JESPLEX. It returns the current settings for the initiator (selection parameters), the status of the initiator, and if present, the job currently active in the initiator. The SSI supports returning this information for the local JESPLEX member or some other member of the JESPLEX. If you are running with SECLABEL dominance active, then a dominance check is done to determine if the requester can obtain information about the job executing in the initiator. This check is optional for authorized applications.

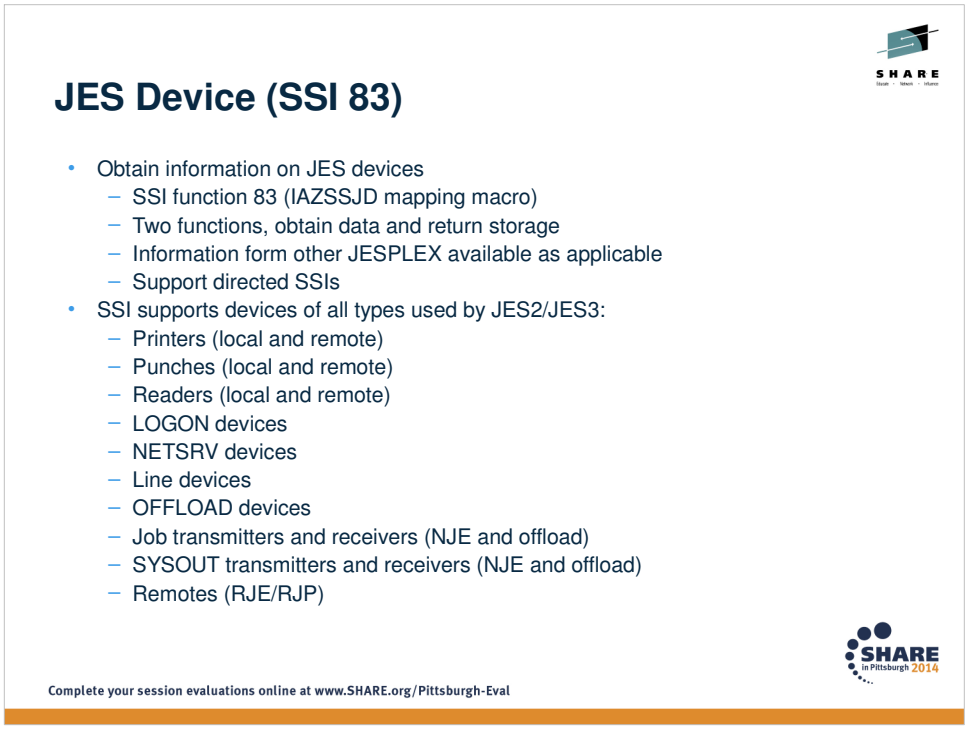


The JESPLEX subfunction of the JES property SSI returns information on member of the JESPLEX (JES2 MAS or JES3 complex). It returns the parameter settings for the member, the current member status, and general system information (product version, etc).



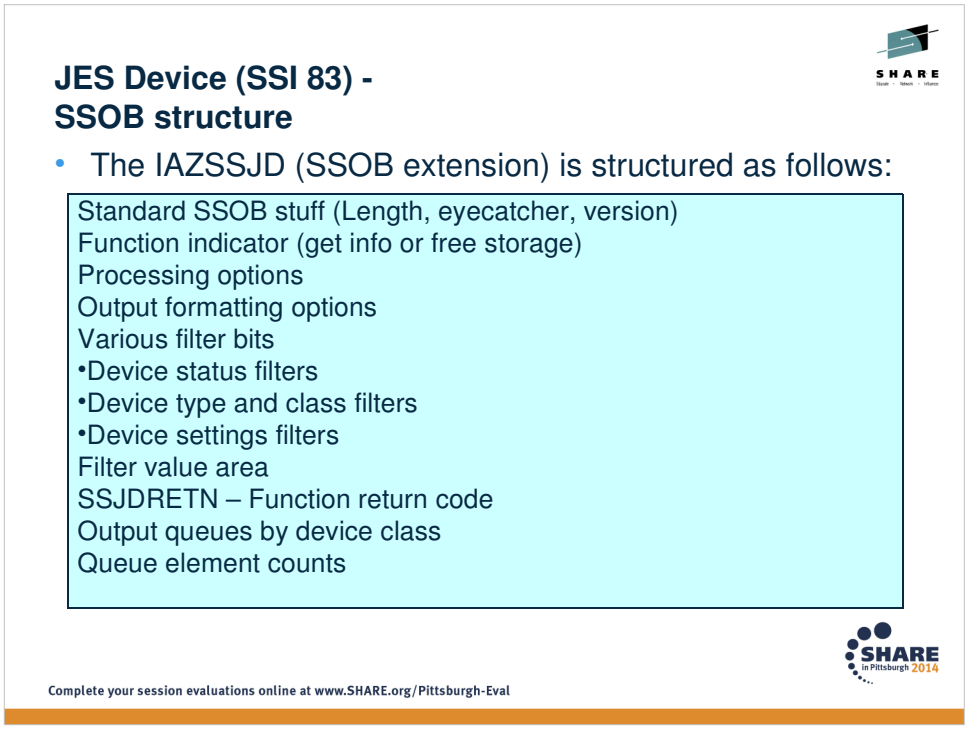
The job class subfunction of the JES property SSI returns information on job classes defined to JES. It returns the parameter setting for the class (Converter parms for JES2), the various limits for the class, and the current execution counts by member.

Note that the JES2 only job class information subfunction of the JOB/JES2 information SSI (71) has been deprecated and is no longer being enhanced. This subfunction uses the IAZJBCLD macro to request information similar to what is returned using this function. Users of IAZJBCLD should switch to using this SSI to obtain job class information,



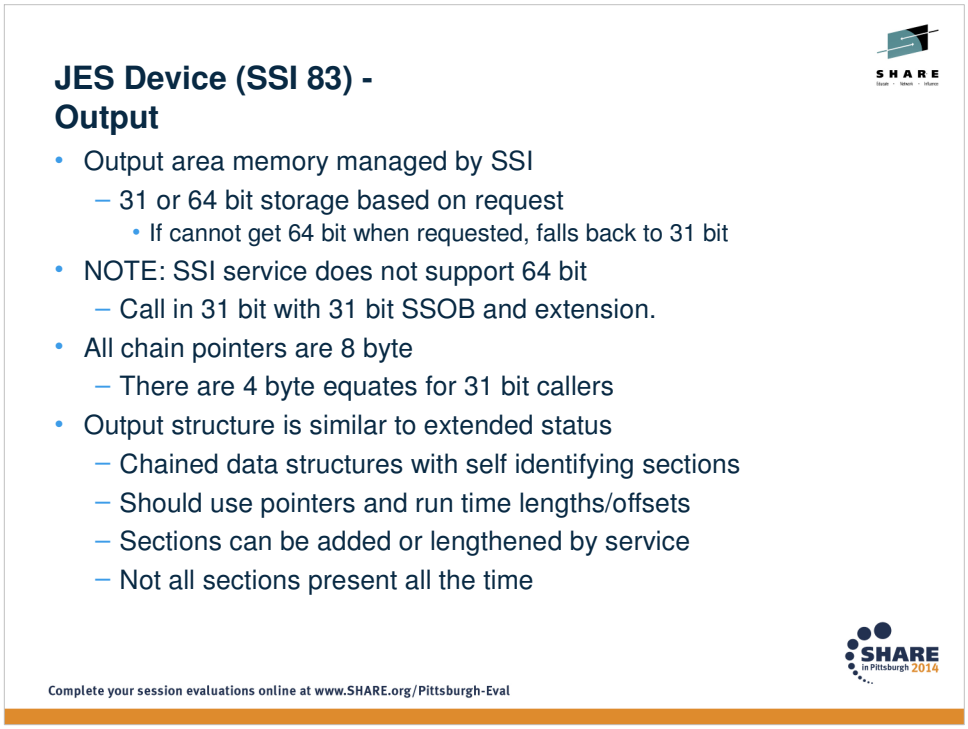
The JES device SSI returns information on the devices that JES uses. It returns the settings for the devices along with the current device status (job active on the device, etc). It is intended that this information be available in a JES independent manner when possible. The SSOB extension for this SSI is IAZSSJD. Filters control what devices information will be returned for. Output areas for ALL devices are mapped in IASSJD.

it is possible to obtain the information from the perspective of another member of the JESPLEX.



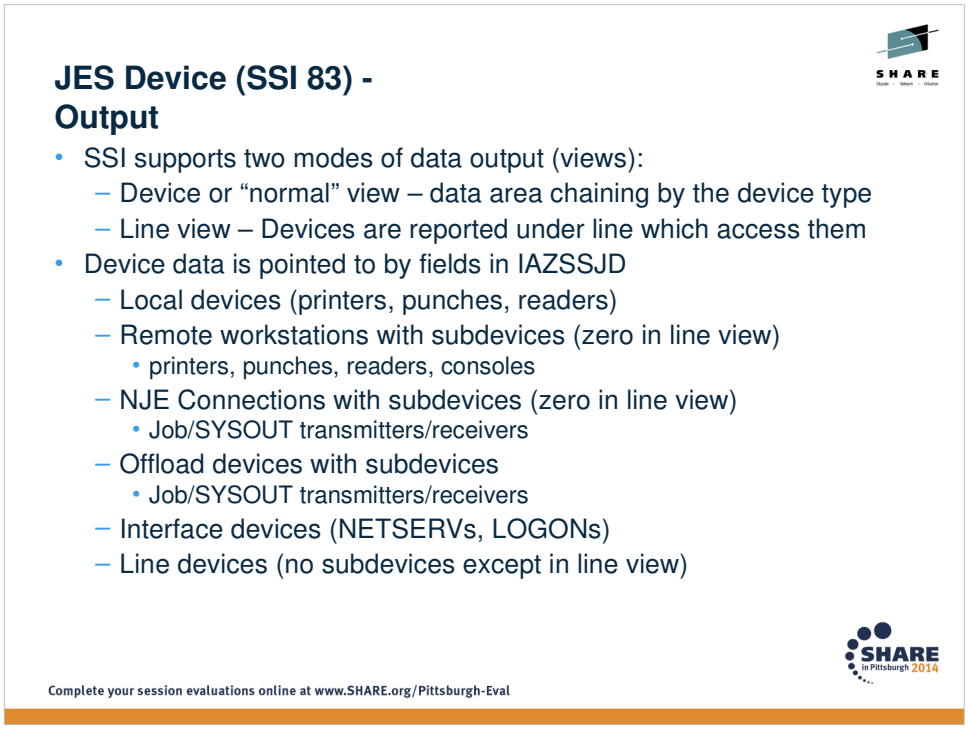
The SSOB extension is mapped by IAZSSJD. It is divided into al number of section to help understand what options are available. The start is the standard SSOB extension stuff with a function byte to indicate if this is a request to get information or return storage. This is followed by input fields used to control what gets returned and how to organize the output. The processing options indicate if the output is to be in 64 bit storage and if there is a limit to how much data is to be returned. There are filters that can select device classes (eg. local, remote, NJE) and device types (eg printers, punches, lines) to return. There are status filters for things like active vs inactive, and other general filters like systems, device settings, etc. Many filters have related value which are then listed.

The input area is followed by the output area. This includes the return code for the request and the various device queue heads and counts.



You can request output to be returned in 31 or 64 bit storage. All output pointers are 8 byte wide with 4 byte equates for 31 bit address users (8 byte fields are always valid addresses). If the SSI cannot get needed 64 bit memory, it will use 31 bit memory instead. Note that even though the output can be returned in 64 bit storage, the SSI interface does not support 64 bit callers and all input must be passed in 31 bit storage.

The output of this SSI is very similar to extended status. There are self defining sections that are chained together with the various output data. The same rules apply, always use run time lengths when available, be tolerant of unknown sections or missing sections, and things can change with service.



Output areas are chained according to the class of the device. Most devices fit into exactly one category of output. However, an application can request a special line view of the data. Under a line view, NJE and RJE devices are returned under the line that they are associated with instead of being returned under the remote and NJE connection sections. This allows an application that is building a display based on lines (BSC line in JES3 and all line types in JES2) to have an appropriate high level structure (a line) with the appropriate devices chained under them.

A Look at the TAMDS77 Screen

```
POOL= 70%  IN SYSTEM=2319  16 JUL 14.197   9:56:19 A

IDLE INITS: NRM-19 ADAS- 3 ADAM- 1 ADAL- 2 BPPS- 3 BPPM- 1 BPPL- 2
          FAMS- 3 FAMM- 1 FAML- 1 SIMS- 3 SIMM- 1 SIML- 7
NUMB   NAME     STEP  REGION CLASS    PRC                          REL  HELD
                                                          C/I       0     8
                                                        ERROR       0     0
                                                     DUP NAME       0     0
                                                        SETUP       0     0
                                                   ALLOCATION       0     0
                                                    VOL UNAVL       0     0
                                                    VOL MOUNT       0     0
                                                         MAIN      18     0
                                                       OUTPUT     383  1910
                                                        PURGE       0     0
                                                      BACKLOG       0     0

                                        DEVICE   JOB  STATUS  LMT  FORMS
                                        XEROX1        UNAVAIL
                                        XEROX2        UNAVAIL
```

The New Technique

TAMDS77

Spool Status – SSI82

- SSJPFREQ = SSJPSPOD
- Search SSPGENI chain and calculate spool usage

Main Processor Status – SSI 82

- SSJPFREQ = SSJPJXOD
- Search JPXGENI chains for processor status

The New Technique

TAMDS77

Initiator Status – SSI 82

- SSJPFREQ = SSJPITOD
- Search ITIGDIGI chain and parse needed info

Queue Status – SSI 80

- STATTYPE = STATTERS
- Search STATJQTR chain and count based on STTRPHAZ and STTRHOLD flags

Our Lone FCT

History – Think “Early ‘70s”

- MUSAS **(Stanford Wylbur)** implemented with OS/VS2 SVS with HASP4
  - Approx. 1000+ users – students, faculty, staff, universities
- Moved to OS/VS2 MVS with JES3
- Use TSO facilities – **slow, single-threaded**
- 1st attempted fix: Create unique interface block to support multiple STATUS and OUTPUT threads – still too slow

Our Lone FCT

More History –

- 2nd attempted fix: Write SSISERV interface to JES3 for FETCH and LOCATE
  - IATUMIJ (FCT)
  - IATUMLC (LOCATE)
  - IATUMPS (FETCH, RELEASE, CANCEL, PURGE, ALTER)
  - IATUMQU (SHOW JOBS/PRINT)

Code ran for 35+ year!

Out with the Old… FCT

Wylbur LOCATE

- User issues a Wylbur “locate” job status request (name, number, general, pattern)
- JES3 queries JQE/JCT/MPC for each job and tables back to Wylbur
- Wylbur deblocks the returned output, evaluates job status and attributes, and builds response for user

Old Job Status - LOCATE

JOB 33783 TESTGRP AWAITING EXECUTION PRTY=5 SHIFT=1 GROUP IS OFF
JOB 33784 TESTCLAS AWAITING EXECUTION PRTY=5 SHIFT=1 CLASS IS OFF
JOB 33785 TESTVUN IN VOLUME UNAVAILABLE QUEUE
JOB 33786 TESTVUN AWAITING EXECUTION PRTY=5 SHIFT=1 DUPLICATE JOBNAME RUNNING
JOB 33787 TESTMNT WAITING FOR VOLUME MOUNT PRTY=5 SHIFT=1
JOB 33788 TESTALLO WAITING FOR ALLOCATION PRTY=5 SHIFT=1
JOB 33789 TESTHOLD AWAITING EXECUTION PRTY=5 SHIFT=1 IN OPERATOR HOLD
JOB 33790 TESTWTR AWAITING PRINT   CL=A F=1100 D=XEROX   L=37
JOB 33791 TESTHOUT AWAITING PRINT HOLDOUT CL=A F=1100 D=XEROX   L=39
JOB 33792 TESTOUTH AWAITING PRINT   CL=A F=1100 D=XEROX   L=37
JOB 33793 TESTCC4 AWAITING PRINT HOLDOUT CL=A F=1100 D=XEROX   L=158
                   AWAITING PRINT HOLDOUT CL=G F=1100 D=XEROX   L=258
JOB 33794 TESTABS AWAITING PRINT HOLDOUT CL=A F=1100 D=XEROX   L=75
JOB 33795 TESTABU AWAITING PRINT HOLDOUT CL=A F=1100 D=XEROX   L=160
                   AWAITING PRINT HOLDOUT CL=G F=1100 D=XEROX   L=263
JOB 33796 TESTJCLE AWAITING PRINT HOLDOUT CL=A F=1100 D=XEROX   L=28
JOB 33797 TESTEXEC EXECUTING STEP G   ON T PRTY=5 SHIFT=1

Initial Findings

- TERSE call only provides some fields we need
- Some fields are blank
- If a job is ineligible to run, get code NO INFORMATION
- MAXCC set with Cond Code, Abend code, but not on JCL errors (UPDATE: bug in my code…)
- TERSE JOB returns all jobs, including those waiting print
- TERSE OUTPUT returns *only* jobs awaiting print

New Wylbur LOCATE

```
JOB01888  TESTJCLE  Awaiting Output WTR      JCL ERR
JOB01891  TESTEXEC  Execting                on D
JOB01875  TESTGRP   No Subchain
JOB01876  TESTCLAS  No Subchain
JOB01877  TESTVUN   Unavailable Volumes
JOB01878  TESTVUN   No Subchain                    Duplicate Jobname
JOB01879  TESTMNT   Awaiting Mount
JOB01880  TESTALLO  Awaiting Allocation
JOB01881  TESTHOLD  No Subchain                    In Operator Hold
JOB01882  TESTWTR   Awaiting Output WTR      CC 0000
JOB01883  TESTHOUT  Awaiting Output WTR      CC 0000
JOB01884  TESTOUTH  Awaiting Output WTR      CC 0000   In Operator Hold
JOB01885  TESTCC4   Awaiting Output WTR      CC 0004
JOB01886  TESTABS   Awaiting Output WTR      AB S806
JOB01887  TESTABU   Awaiting Output WTR      AB U1234
```

What We Learned

- This looks promising!
- What we were able to work with is powerful and versatile
- Example program in the *SSI* manual is a good starting point
  - Might clean up the CALL section….
- STATPERF (IAZSSST) isn't set for JES3!
  - Can't measure the cost of running the SSI
- Job Owner vs Job Submitter?
- Overall, TERSE call is good enough for wildcard Wylbur LOCATE, but VERBOSE is needed for more specific

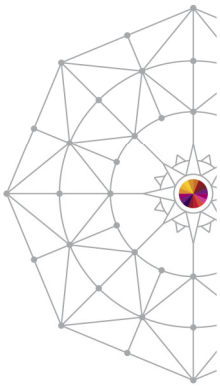
Where Do We Go From Here?

- Look forward to full filtering support in all SSIs
- Continue with JES mod elimination and MUSAS modification

Questions?

Thanks for stopping by…

james-lund@tamu.edu
anadel@us.ibm.com



#SHAREorg

SHARE in Pittsburgh 2014