Using the New JES2/JES3 SSIs in z/OS 1.13

SHARE Orlando, Session 09762
Friday, August 12, 2011

Tom Wasik - wasik@us.ibm.com
David Jones - davidjon@us.ibm.com
JES Design/Development/Service
Rochester, NY

This presentation is a technical description of how to use the SubSystem Interface (SSI) to access information stored in JES2 and JES3. In it, we will discusses the JES SSIs as they exist in z/OS 1.13 and how to use the newer interfaces with some assembler coding examples. It is intended for programmers that may wish to exploit the interface or to understand what is possible using these interfaces.

The major change in z/OS 1.13 is the completion of the 2 new SSIs, JES properties and JES devices. But there are also some changes from earlier releases that will be discussed and some SSIs that are new to JES3.

There are examples programs for many of these SSIs that are attached to this PDFs as .TXT files in the SHARE proceedings. If your want the latest version of this presentation and the samples, send me a note at wasik@us.ibm.com and I will send you the latest version.

Trademarks

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

IBM®
MVS
JES2
JES3
RACF®
z/OS®
zSeries®

* Registered trademarks of IBM Corporation

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.

The required trademark page.....

Overview

- **This presentation will cover the following:**
  - What is the SSI and how is it used
  - JES property SSI request
  - JES device SSI request
  - Extended status SSI request
  - SAPI SSI requests
  - SWB modify and merge (read) SSI
- **Note: Sample programs are attached to this PDF**
  - View->Show/hide->Navigation Panes->Attachments

The goal of this presentation is to give you an idea of what is possible using the various SSIs that JES supports. It will concentrate on the JES side of the interface and not on how to present or externalize the data. We will start with the fundamentals of the SSI, and then go over some specific SSIs concentrating on changes made in z/OS 1.11 through z/OS 1.13.

There are examples programs for many of these SSIs that are attached to this PDFs as .TXT files in the SHARE proceedings. If your want the latest version of this presentation and the samples, send me a note at wasik@us.ibm.com and I will send you the latest version.

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 functions it supports
  - Caller can specify subsystem to process request
    - Default, Specific, All

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 JES2 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 JES2 SSI available to applications that can be broadcast is the extended status SSI.

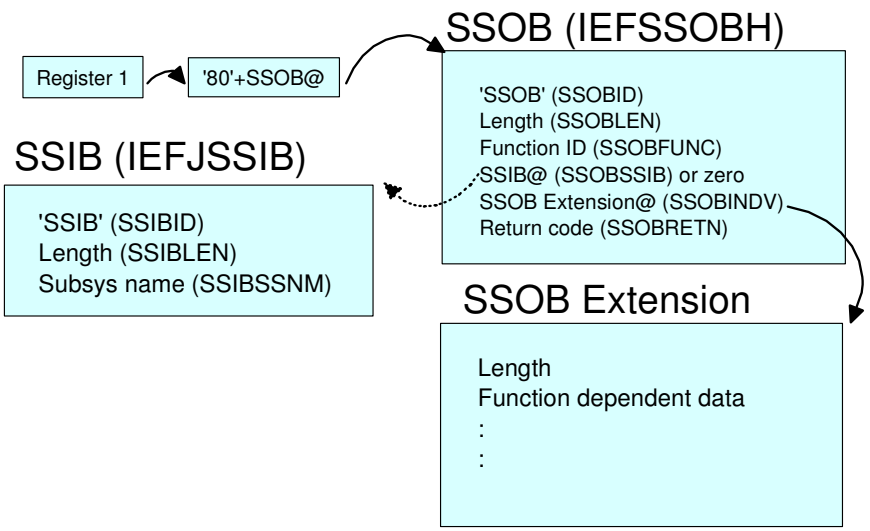
What is the SSI? (cont...)

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

Number	Symbol	Macro	Auth	Description
1	SSOBSOUT	IEFSSSO	Y	Process SYSOUT
2	SSOBCANC	IEFSSCS	Y	Job cancel
3	SSOBSTAT	IEFSSCS	Y	Job status
11	SSOBUSER	IEFSSUS	N	Destination validation/conversion
20	SSOBRQST	IEFSSRR	Y	Request job ID
21	SSOBRTRN	IEFSSRR	Y	Return job ID
54	SSOBSSVI	IEFSSVI	N	Subsystem information
70	SSOBSFS	IAZSSSF	N	SJF SPOOL services (modify/merge)
71	SSOBSSJI	IAZSSJI	Y/N	Job/JES2 information (JES2 only)
75	SSOBSSNU	IAZSSNU	N	User notification
79	SSOBSOU2	IAZSSS2	N	SYSOUT API (SAPI)
80	SSOBESTA	IAZSSST	N	Extended status information
82	SSOBSSJP	IAZSSJP	N	JES property information
83	SSOBSSJD	IAZSSJD	N	JES device information

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 (SA22-7642-06). 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.

Invoking the SSI - Data areas



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.

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
```

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.

Hints and Tips

- **Use IEFSSI REQUEST=QUERY to get SSI info**
  - Request specific, all, or primary subsystem info
  - Output mapped by IEFJSQRY
  - Lists functions subsystems support
  - Indicates if JES2 or JES3 subsystem
- **Use authorized code only when needed**
  - Code errors can cause less errors
- **Examples help understand input/ouptut**
  - Many examples have hex display of input/output
  - Simplistic logic that demonstrates function

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.

Finally, the examples that are provided are designed to help you understand the inputs and outputs of the SSIs. They are useful to use as a starting point as well as to run to actually look at the data returned. They are very simplistic in how they obtain their input and display their output.

JES Property SSI

- **Obtain Various JES information (parameters and status)**
  - SSI function 82 (IAZSSJP mapping macro)
  - Router type SSI with various subfunctions
  - Subfunctions come in pairs (get information and return storage)
  - Separate request block maps input and output
  - Information form other JESPLEX available as applicable
  - Support directed SSIs

Macro	JESPLEX	Function
IAZJPNJN	Yes	NJE node information
IAZJPSPL	No	JES SPOOL information
IAZJPITD	Yes	Initiator information (JES and WLM)
IAZJPLEX	No	JESPLEX member information
IAZJPCLS	No	Job class information

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 - 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

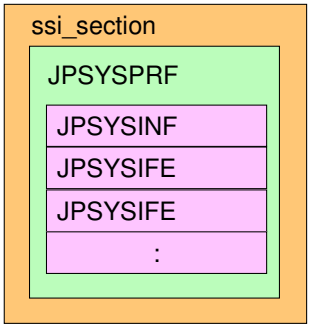
The SSOB extension is mapped by IAZSSJP. The extension is essentially a standard extension with a function request byte, data area pointer, and extended return code. It is the function depended area that has most of the interesting information

JES Property SSI – JESPLEX Member Data

- **Used in JESPLEX capable SSIs**
  - Mapped by IAZJPLXI
  - Provides information on member returning data
    - MVS name and JES2 member name
    - Subsystem name
    - Command prefix and length
    - JES2 version (character, product/service level)
    - Global/Local flags (JES3)
    - Success/error/status flags
- **Standard across multiple SSI and subfunctions**

There are a number of SSIs and subfunctions that return information from members of the JESPLEX different from the one that processed the request. Some even provide a mechanism to request information for all members of the JESPLEX. When this information is returned, it is sometimes important to understand some basic information about the JESPLEX member replying to the request. To simplify this processing, a single structure was created to return information about the member that responded to the request. This eliminates the need to do a separate JESPLEX request to get information about other member and contains status information pertaining to this specific request (such as the internal level of the data that was returned).

JES Property SSI – JESPLEX Member Data



- **Section in the IAZJPLXI**
  - ssi_section – Based on SSI returning data
  - JPSYSPRF – Describes the output areas
  - JPSYSINF – Offset, length and count of member sections that follow
  - JPSYSIFE – Array of up to 32 member information sections
- **Also used in the JES device SSI (see below)**

The JESPLEX member data will be contained in a structure that is defined for a specific SSI. This section is composed of a number of sections. Each section has identifying information and a section length (the exception is the PSYSPRF section). The high level DSECT will generally have additional pointer to other areas.

JPSYXLNG (in JPSYSPRF) 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 JPSYSPRF length equate (JPSYXSIZ) to the address of the JPSYSPRF to get the first variable section (JPSYSINF). Currently this is the only variable section. It starts with a 2 byte length (JPSYLNG), a 1 byte ID fields (JPTYSYSI) and a 1 byte modifier (JPMDSYSI). You should always verify both the type AND modifier to determine what section this is. JPSYSINF defines the offset to the member entries (from JPSYSINF), the number of entries, and the size of each entry. Your code should always use the run time lengths to access each section.

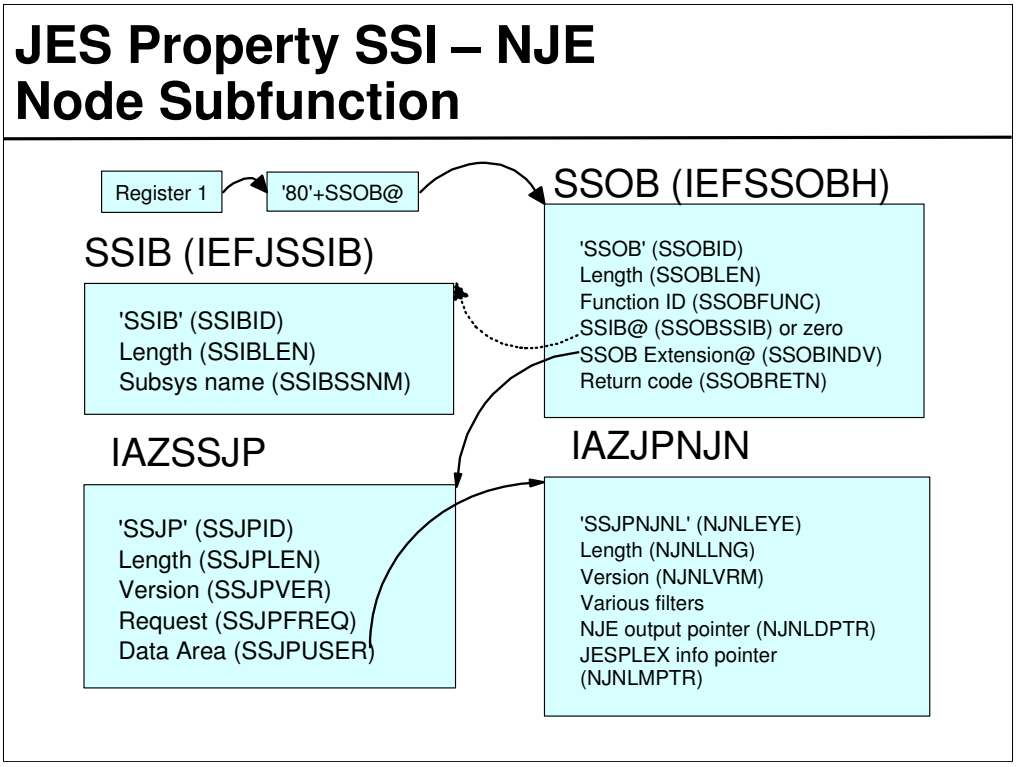
This section is also included in the output of the JES device SSI that is described later in this presentation.

JES Property SSI – NJE Node Subfunction

- **Returns information on NJE nodes**
  - Subfunction of JES property SSI 82 (IAZSSJP mapping macro)
  - Functions SSJPNJOD and SSJPNJRS (IAZJPNJN mapping macro)
  - Directed SSI (Does not require job structure)
  - Supports information from other JESPLEX members
- **Information includes parameter settings and connection status**
- **SECLABEL dominance checks supported (JES2 Only)**
  - If SECLABEL dominance active
  - Optional for authorized applications

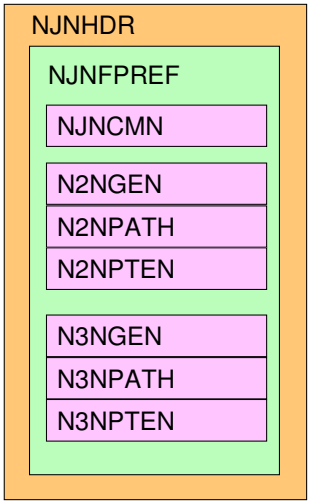
The NJE node subfunction of the JES property SSI returns information on NJE nodes defined to JES. It returns the current settings for the node and the connection status. The SSI supports returning this information for the local JESPLEX member or some other member of the JESPLEX. If you are running with SECLABEL dominance active, then a writer class check is supported to determine if the requester can obtain information about this node. This check is optional for authorized applications.

JES Property SSI – NJE Node Subfunction



This is the data areas associated with the NJE node information SSI. It is the normal data area associated with a JES property SSI request plus the extra function dependent data area (IAZJPNJN).

JES Property SSI – NJE Node Subfunction



- **Section in the NJNHDR**
  - NJNFPREF – Prefix section
  - NJNCMN – Common section
  - N2NGEN – JES2 section
    - N2NPATH – JES2 path array
    - N2NPTEN – JES2 path element
  - N3NGEN – JES3 section
    - N3NPATH – JES3 path array
    - N3NPTEN – JES3 path element
- **Section NJSHDR contains IAZJPLXI entries**

Output from the NJE Node subfunction is a chain of NJNHDRs, each representing one NJE node, and a separate chain of NJSHDRs, each representing IAZJPLXI entries for systems that responded to the request.

The NJNHDR contains the information for the node including path information indicating the adjacent note that data is sent to. It is composed of a number of sections each having identifying information and a section length (the exception is the NJNFPREF section). The high level DSECT (NJNHDR) has the pointer to the next NJNHDR.

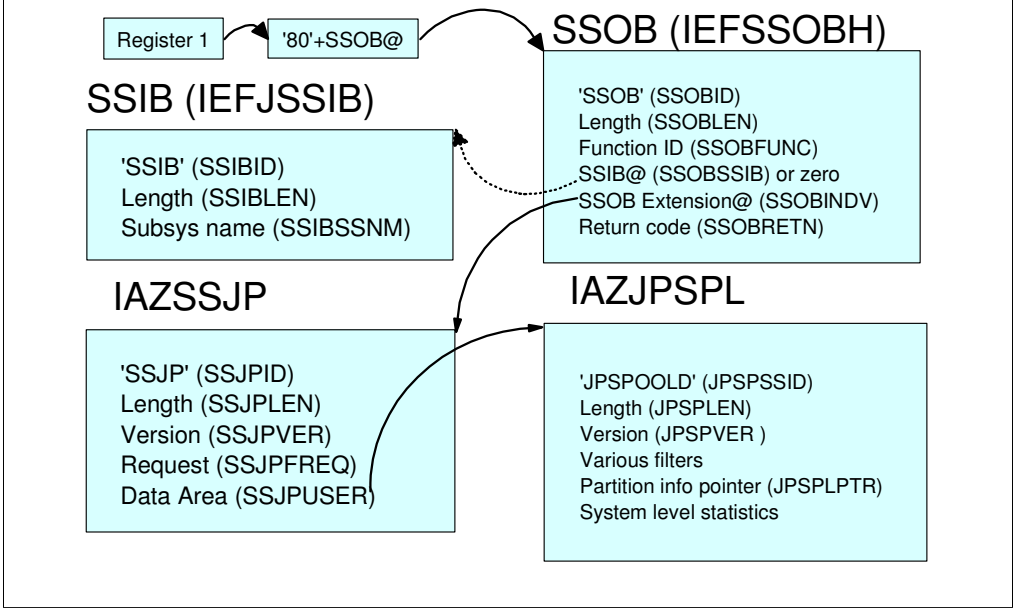
The length of the NJNHDR header is stored in NJNHOHDR. Add this length field to the NJNHDR and you point to the NJNFPREF. This is a header for the remaining fields. NJNFLNG (in NJNFPREF) has the overall length of the remaining areas. This length is used to determine when you have reached the end of the variable sections. You add the NJNFPREF length equate (NJNFSIZE) to the address of the NJNFPREF to get the first variable section. Each variable section starts with a 2 byte length, a 1 byte ID fields and a 1 byte modifier. 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 2 byte section length field to the current section pointer. Not all sections are present for all nodes. In addition, maintenance or a new release can add new section types or modifiers to existing types. Ensure your application can handle unknown types.

JES Property SSI – SPOOL Info Subfunction

- **Returns information on SPOOL volumes**
  - Subfunction of JES property SSI 82 (IAZSSJP mapping macro)
  - Functions SSJPSPOD and SSJPSPRS (IAZJPSPL mapping macro
  - Directed SSI (Does not require job structure)
- **Information includes**
  - Overall statistics (SPOOL space available and used)
  - Partition information (JES3)
  - Status and statistics of individual volumes/extents

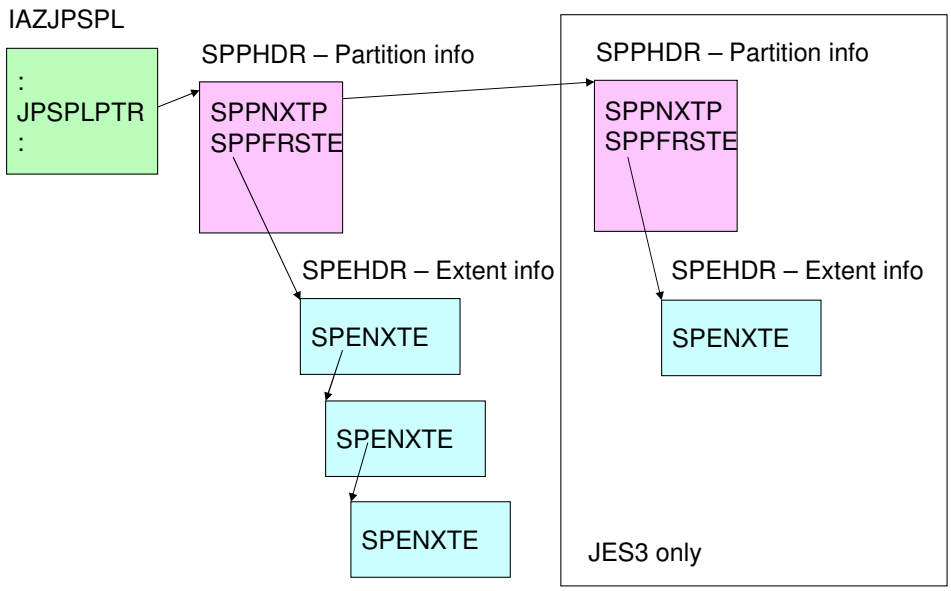
The SPOOL subfunction of the JES property SSI returns information on overall SPOOL space and individual volumes defined to JES. Information is JESPLEX in nature since SPOOL space is defined to the JESPLEX. Information is broken down at the JEXPLEX level, the SPOOL partition level (JES3) and the individual 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.

JES Property SSI – SPOOL Info Subfunction



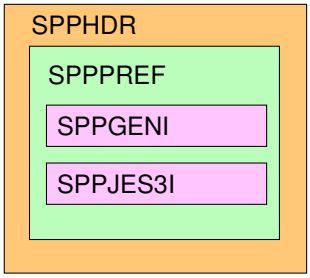
This is the data areas associated with the SPOOL information SSI. It is the normal data area associated with a JES property SSI request plus the extra function dependent data area (IAZJPSPL).

JES Property SSI – SPOOL Info Subfunction



Output for the SPOOL information SSI is returned at 2 levels. The first level is the SPOOL partition level represented by the SPPHDR area. A SPOOL partition represents a subset of SPOOL volumes defined for management purposes. JES2 has only one SPOOL partition. JES3 can define multiple partitions. The SPPHDR then points to a chain of SPEHDR areas, each representing a single SPOOL extent

JES Property SSI – SPOOL Info Subfunction

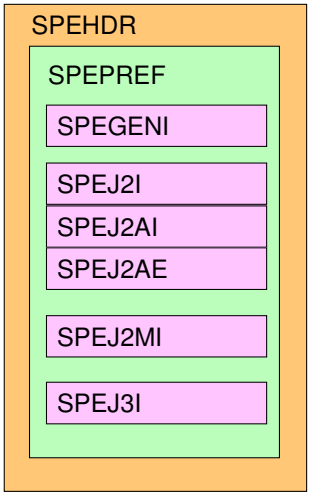


- **Section in the SPPHDR**
  - SPPPREF – Prefix section
  - SPPGENI – Common section
  - SPPJES3I – JES3 section

The SPPHDR contains the partition level information (including usage at the partition level). JES2 has only one partition and JES3 can have multiple defined. The SPPHDR is composed of a number of sections each having identifying information and a section length (the exception is the SPPPREF section). The high level DSECT (SPPHDR) has the pointer to the next SPPHDR and to the first of the chain of SPOOL extent blocks (SPEHDRs).

The length of the SPPHDR header is stored in SPPOPRF. Add this length field to the SPPHDR and you point to the SPPPREF. This is a header for the remaining fields. SPPPRLN (in SPPPREF) 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 SPPPREF length equate (SPPPRSZ) to the address of the SPPPREF to get the first variable section. Each variable section starts with a 2 byte length, a 1 byte ID fields and a 1 byte modifier. 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 2 byte section length field to the current section pointer. Not all sections are present for all nodes. In addition, maintenance or a new release can add new section types or modifiers to existing types. Ensure your application can handle unknown types.

JES Property SSI – SPOOL Info Subfunction



- **Section in the SPEHDR**
  - SPEPREF – Prefix section
  - SPEGENI – Common section
  - SPEJ2I – JES2 section
    - SPEJ2AI – Affinity array
    - SPEJ2AE – Affinity element
  - SPEJ2MI – JES2 migration section
  - SPEJ3I – JES3 section

The SPEHDR contains the SPOOL extent level information. It is composed of a number of sections each having identifying information and a section length (the exception is the SPEPREF section). The high level DSECT (SPEHDR) has the pointer to the next SPEHDR in the same partition.

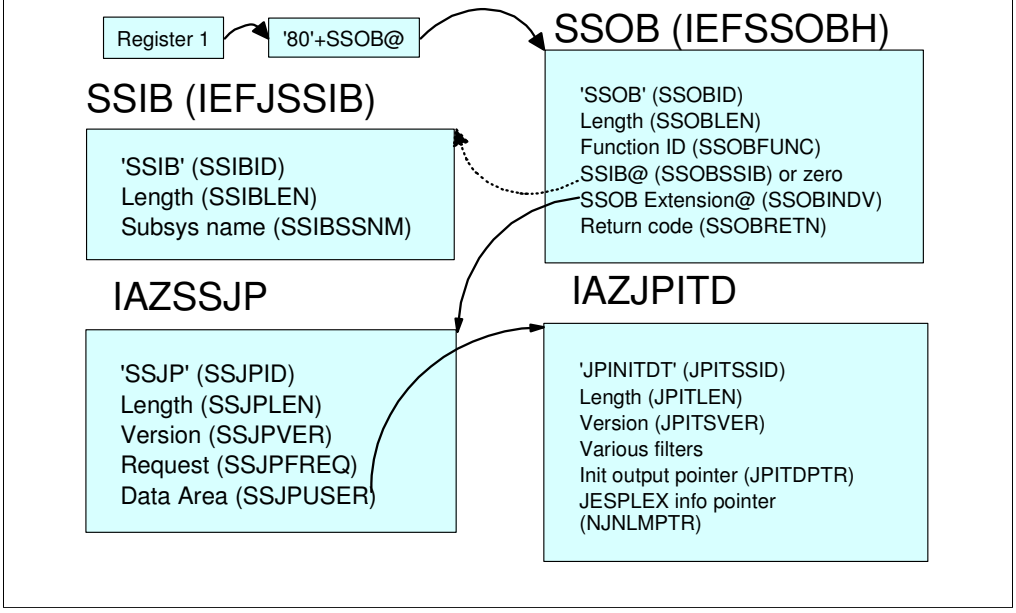
The length of the SPEHDR header is stored in SPEOPRF. Add this length field to the SPEHDR and you point to the SPEPREF. This is a header for the remaining fields. SPEPRLN (in SPEPREF) has the overall length of the remaining areas. This length is used to determine when you have reached the end of the variable sections. You add the SPEPREF length equate (SPEPRSZ) to the address of the SPEPREF to get the first variable section. Each variable section starts with a 2 byte length, a 1 byte ID fields and a 1 byte modifier. 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 2 byte section length field to the current section pointer. Not all sections are present for all nodes. In addition, maintenance or a new release can add new section types or modifiers to existing types. Ensure your application can handle unknown types.

JES Property SSI – Initiator Info Subfunction

- **Returns initiator status information (JES and WLM)**
  - Subfunction of JES property SSI 82 (IAZSSJP mapping macro)
  - Functions SSJPITOD and SSJPITRS (IAZJPITD mapping macro
  - Directed SSI (Does not require job structure)
  - Supports information from other JESPLEX members
- **Information includes**
  - Initiator group settings (JES3, JES2 has 2 “groups” JES and WLM)
  - Parameter setting (selection parameters)
  - Status including active job and active step/proc
- **SECLABEL dominance checks supported for jobs on initiator (JES2 only)**
  - If SECLABEL dominance active
  - Optional for authorized applications

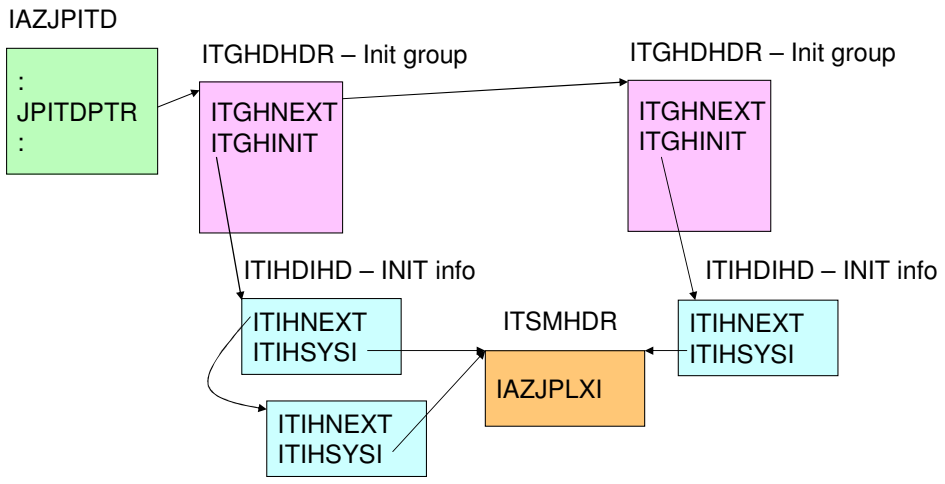
The initiator subfunction of the JES property SSI returns information on initiators defined and active in the JESPLEX. It returns the current settings for the initiator (selection parameters), the status of the initiator, and if present, the job currently active in the initiator. The SSI supports returning 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.

JES Property SSI – Initiator Info Subfunction



This is the data areas associated with the initiator information SSI. It is the normal data area associated with a JES property SSI request plus the extra function dependent data area (IAZJPITD).

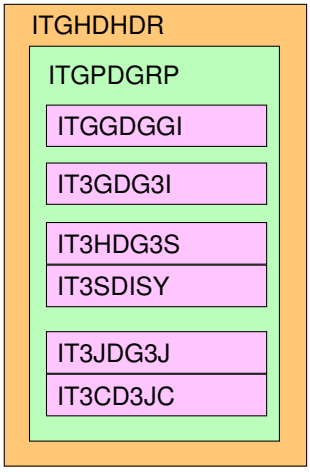
JES Property SSI – Initiator Info Subfunction



- JES2 has 2 groups 1 for WLM initiators and 1 for JES initiators
- JES3 can have multiple groups defined

Output for the initiator information SSI is returned at 2 levels. The first level is at the initiator group level represented by the ITGHDHDR area. A initiator group represents a subset of the initiators defined for management purposes. JES2 has two initiator groups, one for JES mode initiators and one for WLM mode initiiators. JES3 normally has multiple initiator groups defined. The ITGHDHDR then points to a chain of ITIHDIHD areas, each representing a single initiator. The ITGHDHDR points to the system information section (mapped by IAZJPLXI) that describes the system the initiator is running on.

JES Property SSI – Initiator Info Subfunction

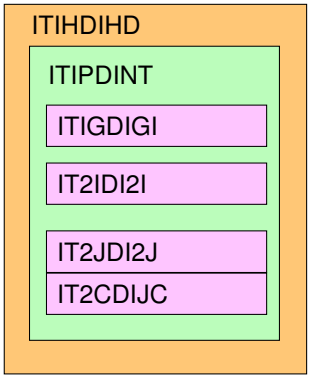


- **Section in the ITGHDHDR**
  - ITGPDGRP – Prefix section
  - ITGGDGGI – Common section
  - IT3GDG3I – JES3 section
  - IT3HDG3S – JES3 system array
    - IT3SDISY – System array entry
  - IT3JDG3J – JES3 jobclass array
    - IT3CD3JC – jobclass array entry

The ITGHDHDR contains the group level initiator information. JES2 has one group got JES mode initiators and one for WLM mode initiators. JES3 normally has multiple groups each can be either WLM or JES mode. The ITGHDHDR is composed of a number of sections each having identifying information and a section length (the exception is the ITGPDGRP section). The high level DSECT (ITGHDHDR) has the pointer to the next ITGPDHDR and to the first of the chain of initiator information blocks (ITIHDIHDs).

The length of the ITGHDHDR header is stored in ITGHOHDR. Add this length field to the ITGHDHDR and you point to the ITGPDGRP. This is a header for the remaining fields. ITGPGLEN (in ITGPDGRP) 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 ITGPDGRP length equate (ITGPSIZE) to the address of the ITGPDGRP to get the first variable section. Each variable section starts with a 2 byte length, a 1 byte ID fields and a 1 byte modifier. 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 2 byte section length field to the current section pointer. Not all sections are present for all nodes. In addition, maintenance or a new release can add new section types or modifiers to existing types. Ensure your application can handle unknown types.

JES Property SSI – Initiator Info Subfunction



- **Section in the ITIHDIHD**
  - ITIPDINT – Prefix section
  - ITIGDIGI – Common section
  - IT2IDI2I – JES2 section
  - IT2JDI2J – JES2 jobclass array
    - IT2CDIJC – Jobclass array entry

The ITIHDIHD contains the initiator level information. It is composed of a number of sections each having identifying information and a section length (the exception is the ITIPDINT section). The high level DSECT (ITIHDIHD) has the pointer to the next ITIHDIHD in the same initiator group. It also has the pointer to the ITSMHDR section that contains the member information (IAZJPLXI mapped) where the initiator is running.

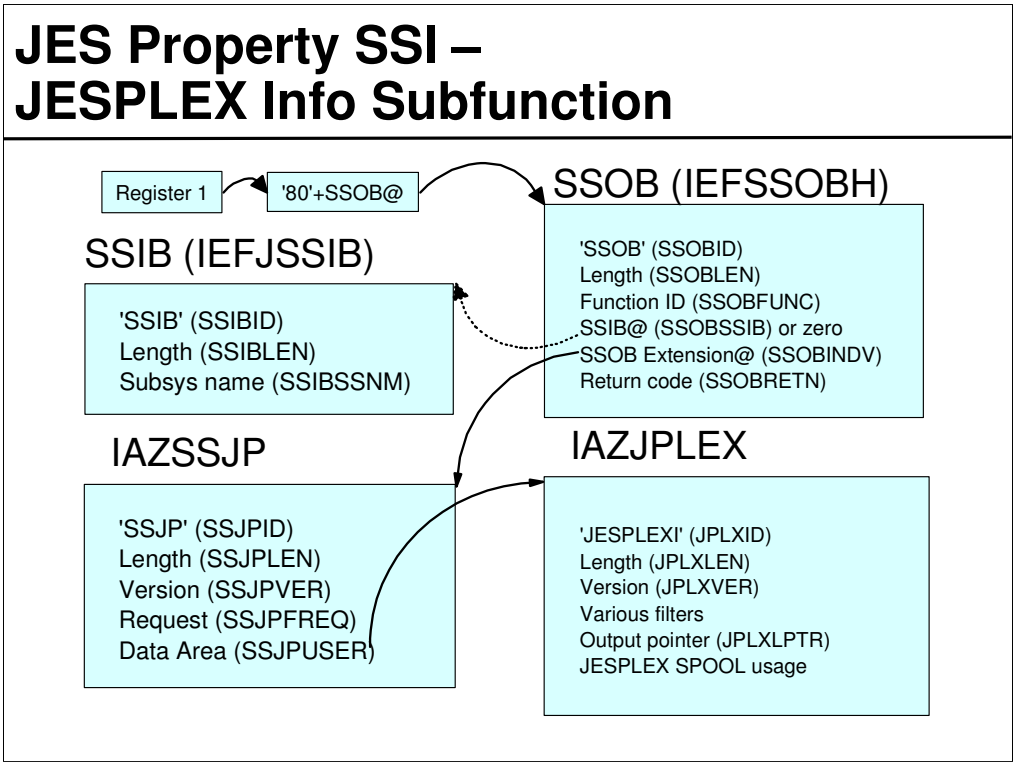
The length of the ITIHDIHD header is stored in ITIHOHDR. Add this length field to the ITIHDIHD and you point to the ITIPDINT. This is a header for the remaining fields. ITIPILEN (in ITIPDINT) has the overall length of the remaining areas. This length is used to determine when you have reached the end of the variable sections. You add the ITIPDINT length equate (ITIPSIZE) to the address of the ITIPDINT to get the first variable section. Each variable section starts with a 2 byte length, a 1 byte ID fields and a 1 byte modifier. 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 2 byte section length field to the current section pointer. Not all sections are present for all nodes. In addition, maintenance or a new release can add new section types or modifiers to existing types. Ensure your application can handle unknown types.

JES Property SSI – JESPLEX Info Subfunction

- **Returns information on members of the JESPLEX**
  - Subfunction of JES property SSI 82 (IAZSSJP mapping macro)
  - Functions SSJPJXOD and SSJPJXRS (IAZJPLEX mapping macro
  - Directed SSI (Does not require job structure)
  - Supports information from other JESPLEX members
- **Information includes**
  - Parameter settings
  - Current status
  - General system information

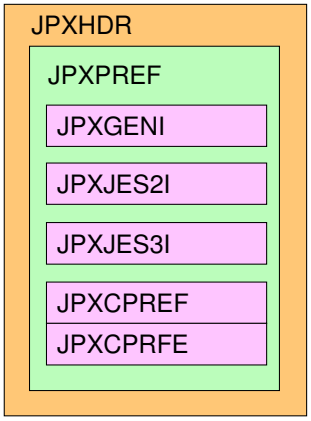
The JESPLEX subfunction of the JES property SSI returns information on member of the JESPLEX (JES2 MAS or JES3 complex). It returns the parameter settings for the member, the current member status, and general system information (product version, etc).

JES Property SSI – JESPLEX Info Subfunction



This is the data areas associated with the JESPLEX information SSI. It is the normal data area associated with a JES property SSI request plus the extra function dependent data area (IAZJPLEX).

JES Property SSI – JESPLEX Info Subfunction



- **Section in the JPXHDR**
  - JPXPREF – Prefix section
  - JPXGENI – Common section
  - JPXJES2I – JES2 section
  - JPXJES3I – JES3 section
  - JPXCPREF – Command prefix array section
    - JPXCPRFE Command prefix entry

Output from the JESPLEX information subfunction is a chain of JPXHDRs, each representing one member of your JESPLEX (JES2 MAS or JES3 Complex).

The JPXHDR contains the information for the member. It is composed of a number of sections each having identifying information and a section length (the exception is the JPXPREF section). The high level DSECT (JPXHDR) has the pointer to the next JPXHDR.

The length of the JPXHDR header is stored in JPXOPRF. Add this length field to the JPXHDR and you point to the JPXPREF. This is a header for the remaining fields. JPXPRLN (in JPXPREF) 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 JPXPREF length equate (JPXPRSZ) to the address of the JPXPREF to get the first variable section. Each variable section starts with a 2 byte length, a 1 byte ID fields and a 1 byte modifier. 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 2 byte section length field to the current section pointer. Not all sections are present for all nodes. In addition, maintenance or a new release can add new section types or modifiers to existing types. Ensure your application can handle unknown types.

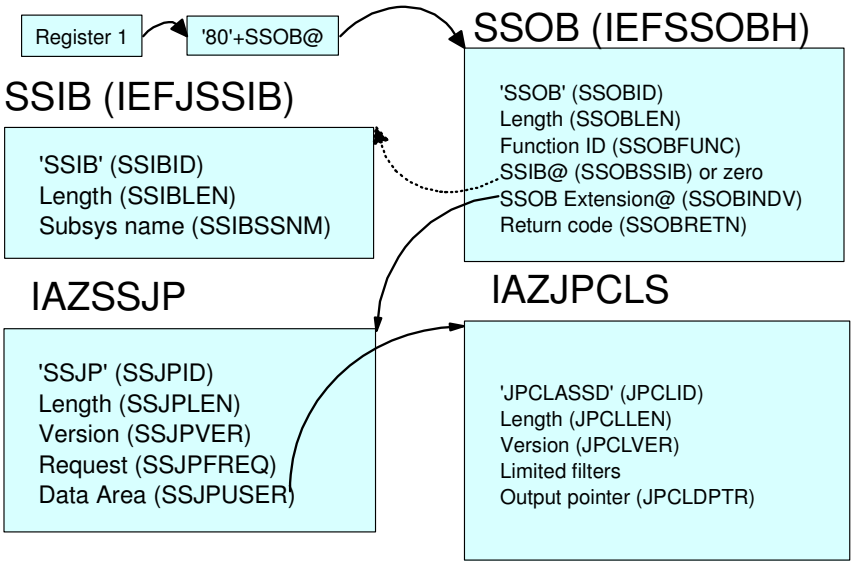
JES Property SSI – JOBCLASS Info Subfunction

- **Returns information on JES job classes**
  - Subfunction of JES property SSI 82 (IAZSSJP mapping macro)
  - Functions SSJPJCOD and SSJPJCRS (IAZJPNJN mapping macro
  - Directed SSI (Does not require job structure)
- **Information includes**
  - Parameter settings (CIPARMs in JES2)
  - Member level limits
  - Current execution counts (by member)
- **JOBCLASS info on SSI 71 deprecated**
  - IAZJBCLD interface macro

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.

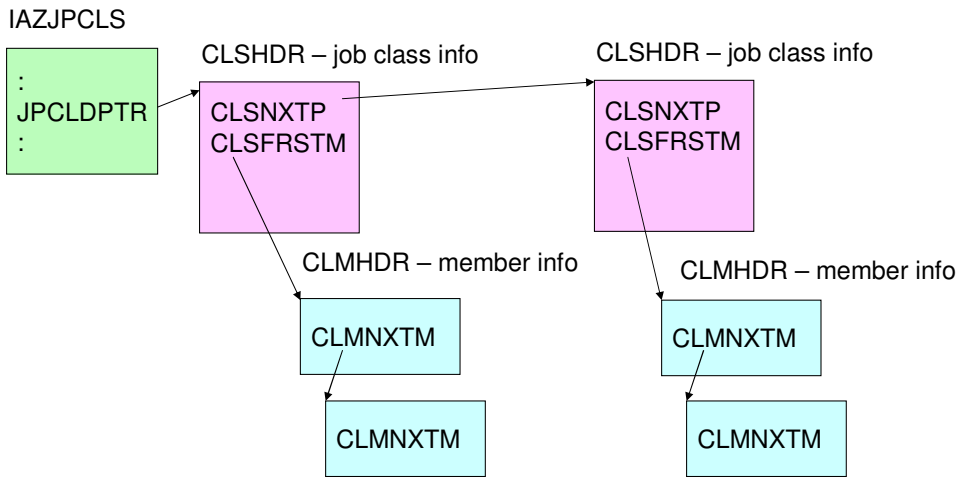
Note that the JES2 only job class information subfunction of the JOB/JES2 information SSI (71) has been deprecated and is no longer being enhanced. This subfunction uses the IAZJBCLD macro to request information similar to what is returned using this function. Users of IAZJBCLD should switch to using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

JES Property SSI – JOBCLASS Info Subfunction



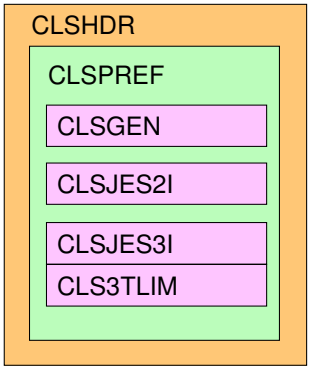
This is the data areas associated with the job class information SSI. It is the normal data area associated with a JES property SSI request plus the extra function dependent data area (IAZJPCLS).

JES Property SSI – JOBCLASS Info Subfunction



Output for the job class information SSI is returned at 2 levels. The first level is the job class level represented by the CLSHDR area. This represents the attributes of the job class. Chained to the CLSHDR is the job class member areas, one for each defined member of the JESPLEX. This defines member level settings/statistics for the job class.

JES Property SSI – JOBCLASS Info Subfunction



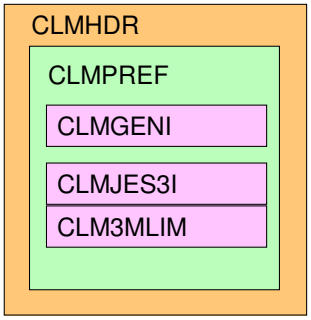
- **Section in the CLSHDR**
  - CLSPREF – Prefix section
  - CLSGEN – Common section
  - CLSJES2I – JES2 section
  - CLSJES3I – JES3 section
    - CLS3TLIM – TLIMIT array entry

Output from the job class information subfunction is a chain of CLSHDRs, each representing a job class defined to JES. The high level DSECT (CLSHDR) has the pointer to the next CLSHDR and to the first of the chain of class member blocks (CLMHDRs).

The CLSHDR contains the information for the job class. It is composed of a number of sections each having identifying information and a section length (the exception is the CLSPREF section). The high level DSECT (CLSHDR) has the pointer to the next CLSHDR.

The length of the CLSHDR header is stored in CLSOPRF. Add this length field to the CLSHDR and you point to the CLSPREF. This is a header for the remaining fields. CLSPRLN (in CLSPREF) 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 CLSPREF length equate (CLSPRSZ) to the address of the CLSPREF to get the first variable section. Each variable section starts with a 2 byte length, a 1 byte ID fields and a 1 byte modifier. 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 2 byte section length field to the current section pointer. Not all sections are present for all nodes. In addition, maintenance or a new release can add new section types or modifiers to existing types. Ensure your application can handle unknown types.

JES Property SSI – JOBCLASS Info Subfunction



- **Section in the CLMHDR**
  - CLMPREF – Prefix section
  - CLMGENI – Common section
  - CLMJES3I – JES3 MLIMIT array
    - CLM3MLIM – MLIMIT entry

The CLMHDR contains member information for the job class (eg. class active on member, member limit, execution count). There is one block for every member defined to JES. It is composed of a number of sections each having identifying information and a section length (the exception is the CLMPREF section). The high level DSECT (CLMHDR) has the pointer to the next CLMHDR.

The length of the CLMHDR header is stored in CLMOPRF. Add this length field to the CLMHDR and you point to the CLMPREF. This is a header for the remaining fields. CLMPRLN (in CLMPREF) 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 CLMPREF length equate (CLMPRSZ) to the address of the CLMPREF to get the first variable section. Each variable section starts with a 2 byte length, a 1 byte ID fields and a 1 byte modifier. 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 2 byte section length field to the current section pointer. Not all sections are present for all nodes. In addition, maintenance or a new release can add new section types or modifiers to existing types. Ensure your application can handle unknown types.

JES Device SSI

- **Obtain information on JES devices**
  - SSI function 83 (IAZSSJD mapping macro)
  - Two functions, obtain data and return storage
  - Information form other JESPLEX available as applicable
  - Support directed SSIs
- **SSI supports devices of all types used by JES2/JES3:**
  - Printers (local and remote)
  - Punches (local and remote)
  - Readers (local and remote)
  - LOGON devices
  - NETSRV devices
  - Line devices
  - OFFLOAD devices
  - Job transmitters and receivers (NJE and offload)
  - SYSOUT transmitters and receivers (NJE and offload)
  - Remotes (RJE/RJP)

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.

JES Device SSI - SSOB structure

- **The IAZSSJD (SSOB extension) is structured as follows:**

Standard SSOB stuff (Length, eyecatcher, version)
Function indicator (get info or free storage)
Processing options
Output formatting options
Various filter bits
•Device status filters
•Device type and class filters
•Device settings filters
Filter value area
SSJDRETN – Function return code
Output queues by device class
Queue element counts

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.

JES Device SSI - Input Filters

- **Filters are divided in multiple groups**
- **Filters within group are ORed**
  - e.g. device type = printer OR device type = punch
- **Filters in different groups are ANDed**
  - e.g. device type = printer AND device class = local
- **Exception – device name filter:**
  - Device is selected if it matches both type AND class filters OR it matches name filter
  - If device type and class filters are not set, device name filter determines device selection
- **Bit indicates if device name filter:**
  - Applies to device names
  - Applies to NJE connection names
- **If filter is not implemented or does not apply in the context, it is ignored.**
  - e.g. JES3-only filters have no effect on JES2

The filters help you limit the amount of data the SSI returns. Filter processing is similar to other SSI with the exception that if you request a device by name, then that device will be returned whether it matches the device type or class filters. Filters that do not apply to a specific JES (or are not implemented by the JES) are ignored.

JES Device SSI - Output

- **Output area memory managed by SSI**
  - 31 or 64 bit storage based on request
    - If cannot get 64 bit when requested, falls back to 31 bit
- **NOTE: SSI service does not support 64 bit**
  - Call in 31 bit with 31 bit SSOB and extension.
- **All chain pointers are 8 byte**
  - There are 4 byte equates for 31 bit callers
- **Output structure is similar to extended status**
  - Chained data structures with self identifying sections
  - Should use pointers and run time lengths/offsets
  - Sections can be added or lengthened by service
  - Not all sections present all the time

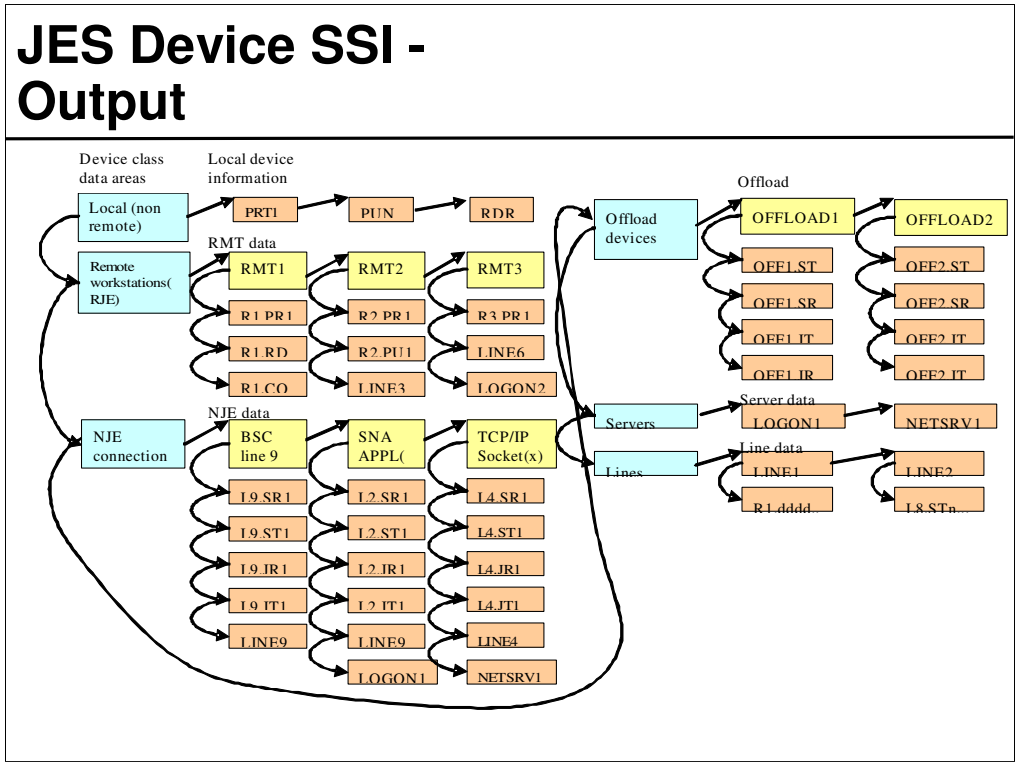
You can request output to be returned in 31 or 64 bit storage. All output pointers are 8 byte wide with 4 byte equates for 31 bit address users (8 byte fields are always valid addresses). If the SSI cannot get needed 64 bit memory, it will use 31 bit memory instead. Note that even though the output can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

JES Device SSI - Output

- **SSI supports two modes of data output (views):**
  - Device or “normal” view – data area chaining by the device type
  - Line view – Devices are reported under line which access them
- **Device data is pointed to by fields in IAZSSJD**
  - Local devices (printers, punches, readers)
  - Remote workstations with subdevices (zero in line view)
    - printers, punches, readers, consoles
  - NJE Connections with subdevices (zero in line view)
    - Job/SYSOUT transmitters/receivers
  - Offload devices with subdevices
    - Job/SYSOUT transmitters/receivers
  - Interface devices (NETSERVs, LOGONs)
  - Line devices (no subdevices except in line view)

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.

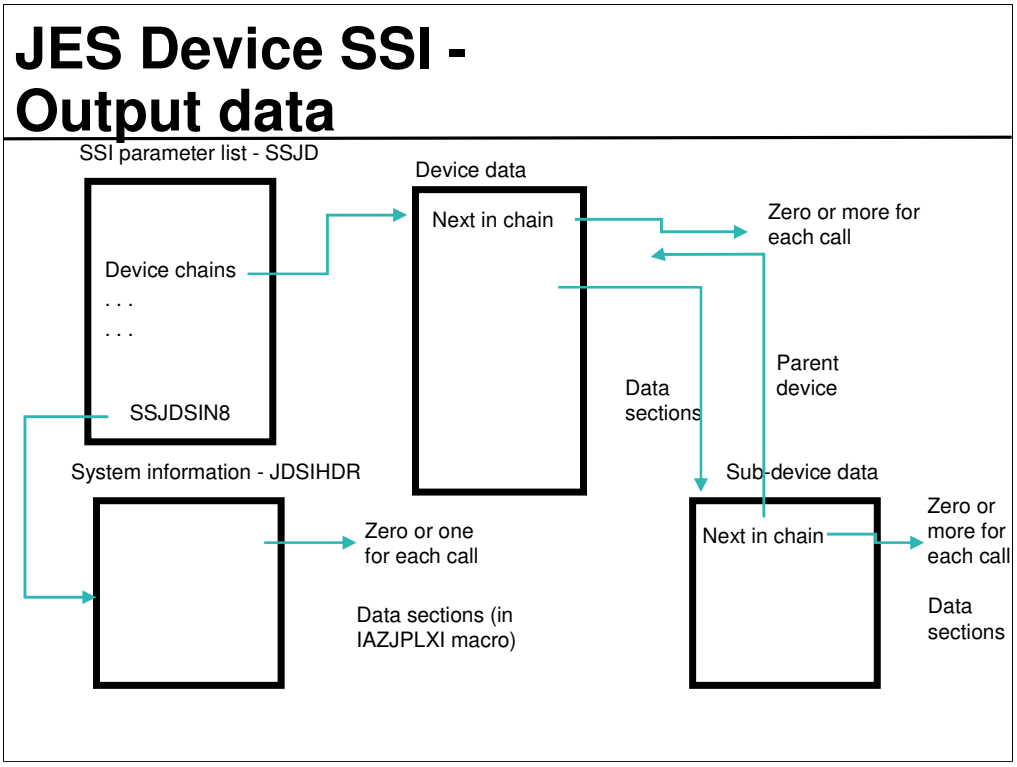


This eye chart is intended to show the levels of output that is returned. The light blue elements are the pointers in the IAZSSJD SSOB extension. The light yellow sections are real or pseudo devices that act as headers for sub devices that are chained under them.

JES Device SSI - Major DSECTs

JDGHLOGN	**– Logon device header**
JDNHNSRV	**– NETSRV device header**
JDLHLINE	**– Line device header**
JDPHPRPU	**– Printer/punch device header**
JDCHCONS	**– Console device header**
JDRHRDR	**– Reader device header**
JDOHOFLD	**– OFFLOAD device header**
JDBHJRCV	**– Job receiver device header**
JDSHSRCV	**– SYSOUT receiver device header**
JDXHJXMT	**– Job transmitter header**
JDYHSXMT	**– SYSOUT transmitter header**
JDJHNJEC	**– NJE connection header**
JDWHRMTW	**– Remote workstation data header**
JDCXPREF	**– Common prefix mapping**
JDSIHDR	**– System information header**

This is the list of the major headers that contain the chaining information and are followed by the data sections. All but the last 2 represent a (non-blue) box in the previous diagram. For more information on what is in each section, see the various DSECTs in the IAZSSJD macro.



Output data from the SSI:

-System information data area – zero or one per call to the SSI (see IAZJPLXI)

-Data areas for the devices are chained to the anchor in the parameter list

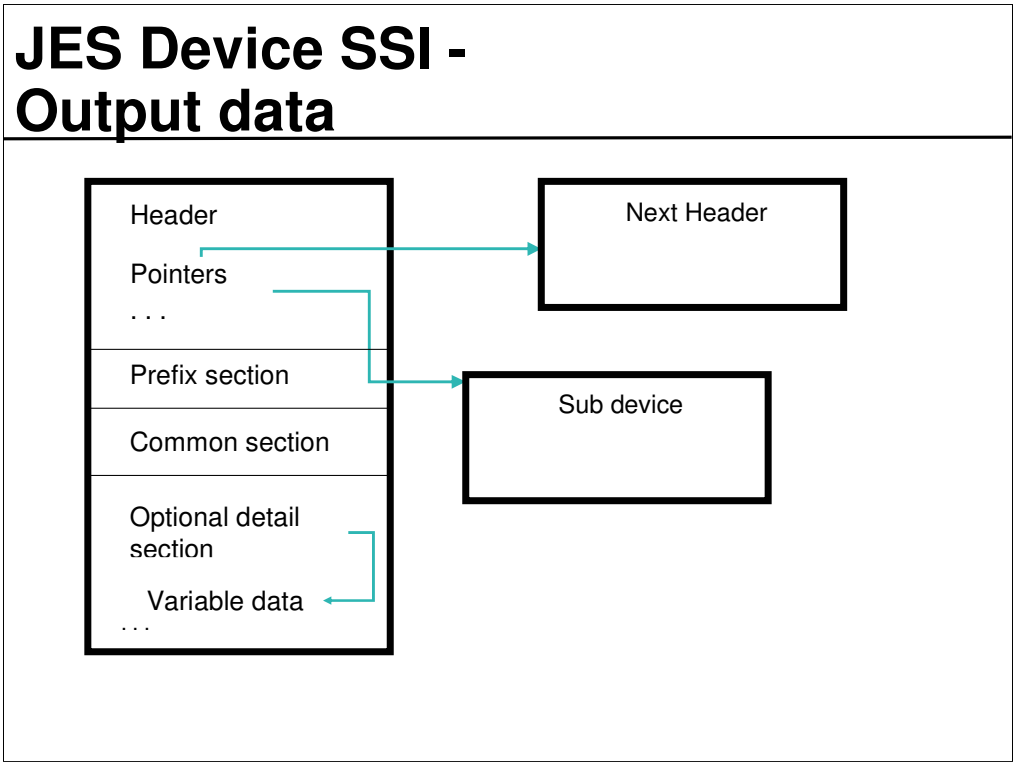
-Some devices have a chain of sub-devices

-Device hierarchy has two levels – device, sub-device.

-Sub-device must have a parent device to be returned – even if parent itself was not selected by filters

-Sub-devices have pointer to their parent device

-For convenience, each device has a pointer to the system information entry for the owning system (member)



Similar to other SSIs, all output data areas consist of a header followed by variable sections. The variable sections format is:

-First section is prefix section
-Next is common section
-Other optional sections that are created as needed (data dependent, JES2, JES3 sections)

Header contains:
-Data area identification - eye-catcher (all eye-catchers are 8 characters)
-Pointers – chaining pointers, parent pointer, system information pointer

Prefix section is the same for all device data areas (DSECT JDCXPREF) and contains:
-Data identification – data/section type and the total length of this data area (without header)

Data sections contain actual data. All data sections have standard fields in the beginning:
-Section type and subtype (type modifier) and section length
These fields are mapped by standard prefix section DSECT (JDCXPREF). Caller can navigate between sections without looking inside section. Sections do not have pointers.

Some sections have variable data – e.g. array of job names. In this case array body follows the section and array is identified within section with three fields:
-Offset to array from start of section, Length of array element, Number of array elements

Extended Status SSI

- **Obtain JOB and SYSOUT information**
  - SSI function 80 (IAZSSST mapping macro)
  - 6 call types
    - Get job data (terse and verbose)
    - Get SYSOUT and JOB data (terse and verbose)
    - Data set list (verbose)
    - Release memory
  - Terse requests obtain easily accessed data
  - Verbose requests return data from SPOOLed CBs
  - Filters control data returned
  - Supports directed SSIs and broadcast

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.

Extended Status SSI

- **JES2 requests use checkpoint version**
  - Two types of versions, live or copy
    - Live only used for job request that filter on a single job
    - Copy implies data could be seconds stale
  - Versions allows all processing to occur in the requestors address space
- **JES3 requests are processed on the global**
  - Overhead in the JES3 address space
  - Data is always current

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.

Extended Status SSI – Request types

- **Get job data (terse and verbose)**
  - Obtains job level information for jobs matching filters
    - Can use SYSOUT filters too
- **Get SYSOUT and JOB data (terse and verbose)**
  - Obtains SYSOUT and job information
    - Terse gives output group level information
    - Verbose gives data set level information
- **Data set list (verbose)**
  - Gives data set information for all matching data sets
    - Includes input data sets and active data sets
  - Generally not grouped, returns all instances
  - For JES2 information in STATSE may not reflect operator command changes
  - For JES3 active data sets may not reflect parameters set by OUTPUT JCL statements

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).

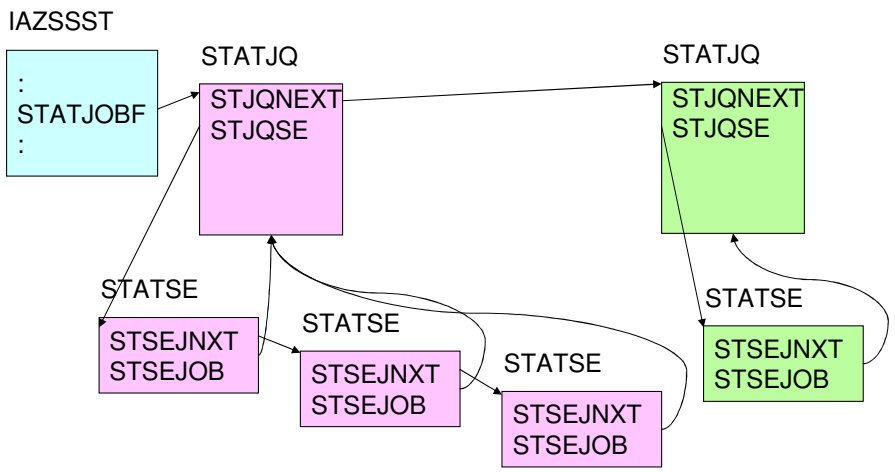
Extended Status SSI - SSOB structure

- **The IAZSSST (SSOB extension) is structured as follows:**

Standard SSOB stuff (Length, eyecatcher, version)
Additional error reason codes (STATREAS, STATREA2)
Function requested (STATTYPE)
Input filter bit masks (STATSELx, STATSSLx)
Input JOB level filter fields
Output area pointers and counts
Input SYSOUT level filter fields
Additional filter values (including filter lists)

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.

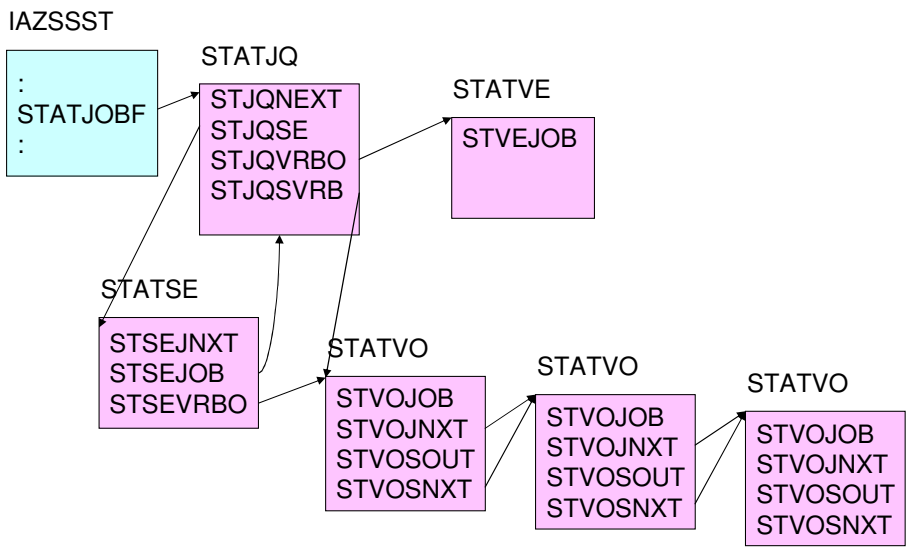
Extended Status SSI - Output structure



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.

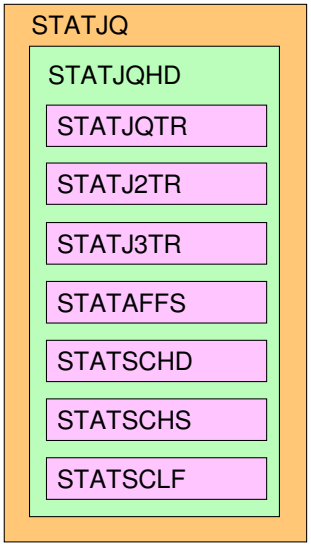
Extended Status SSI - Output structure



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.

Extended Status SSI - STATJQ

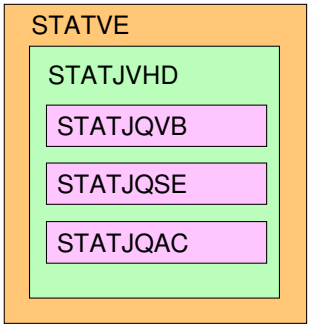


- **Section in the STATJQ**
  - STATJQ - represents job
  - STATJQHD - describes output areas
  - STATJQTR - Job Queue Element terse section
  - STATJ2TR - Job JES2 terse section
  - STATJ3TR - Job JES3 terse section
  - STATAFFS - Job member affinity section
  - STATSCHD - Job scheduling section
  - STATSCHS - Job SCHENV affinity section
  - STATSCLF - Job SECLABEL affinity section
- **Sections work like NJE header sections**
  - Each section has a length, ID, and modifier
  - Use lengths to step through sections
  - STATJQHD has overall length to end of area
  - NEVER USE ASSEMBLER LENGTH EQUs
  - STATJQHD is only exception
  - Not all sections are always present

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 - STATVE

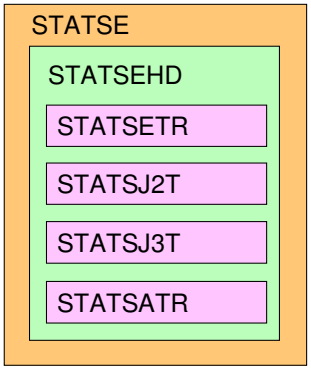


- **Section in the STATVE**
  - STATVE - represents job's JCT data
  - STATJVHD – describes output areas
  - STATJQVB – Job verbose section
  - STATJQSE – Job security section
  - STATJQAC – Job accounting section

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 - STATSE

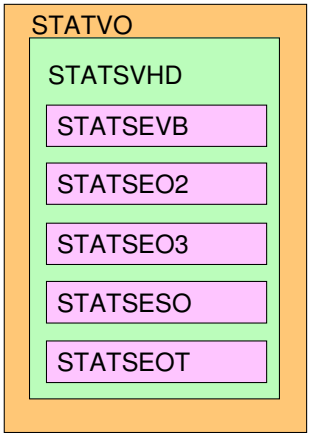


- **Section in the STATSE**
  - STATSE - represents SYSOUT group
  - STATSEHD - describes output areas
  - STATSETR - SYSOUT element terse section
  - STATSJ2T - SYSOUT JES2 terse section
  - STATSJ3T - SYSOUT JES3 terse section
  - STATSATR – Transaction information section

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 - 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.

Extended Status SSI - SOUTs (STATSE) for JES3

- **Each SYSOUT Terse Element (SOUT) returned represents a collection of output data sets for a job.**
  - For JES2 each SOUT represents a Job Output Element ($JOE) which identifies the data sets that would print between header pages.
  - For JES3 each SOUT represents an Output Service Element (OSE) variable section which identifies 1 to 16 SYSOUT data sets with the same output characteristics.
- **JES3 problem is that multiple SOUTs may be returned for all the data sets in a job with the same output characteristics which may be selected by a writer.**
  - Harder to get a complete list of data sets within the same output characteristics.

For JES3, each SOUT returned contains data obtained from Output Service Elements (OSE). Each OSE represents one or more output data sets for a job that have the same output characteristics for selection, but may not be printed together. Also, an OSE does not represent all the job's output data sets with the same output characteristics. This means there may be multiple, duplicate OSEs being used to manage all the data sets with the same output characteristics. All these duplicate OSEs are returned, each in a SOUT, for SSI 80 STATOUTT.

To obtain the complete list of SYSOUT data sets with the same output characteristics, a STATDLST request would be required for each SOUT returned along with a user having to determine which SOUTs have identical characteristics.

By contrast, with JES2 each SOUT contains data obtained from Job Output Elements (JOEs) which represent a collection of output data sets for a job that are to be printed together between separator pages. To obtain a complete list of SYSOUT data sets that would be printed together, a single STATDLST for the SOUT returned.

Extended Status SSI - WSI added for JES3

- **JES3 has added the Work Selection Identifier (WSI)**
  - A value maintained by JES3 in all OSE variable sections that have the same set of output characteristics within a job.
  - Value is maintained across JES3 restarts.
  - WSI values for one job have no relationship to WSI values for another job.
- **JES3 returns the WSI value in each SOUT.**
  - No change to the number of SOUTs returned.

The Work Selection Identifier (WSI) is a new value that was created and is maintained in the OSE variable section. The purpose of the WSI is to provide a mechanism that can be used to quickly identify the set of OSE variable sections within a job that represent a collection of SYSOUT data sets that are related based upon a specific set of output characteristics. The WSI value will be returned with SSI 80 Extended Status and can be used to consolidate SOUTs so that a single SOUT is returned for related OSE variable sections within a job.

Extended Status SSI - Using WSI

- **WSI can be used to identify the set of SOUTs returned that have the same set of output characteristics within a job.**
  - Can then use STATDLST for each SOUT to get the complete list of data sets.
- **Optionally, JES3 can consolidate the SOUTs for a job based upon the WSI value.**
  - One SOUT is returned for the collection of SOUTs with the same WSI value.
  - One STATDLST can be used to get the complete list of data sets that have the same set of output characteristics within a job.

Users can determine the set of SOUTs returned that are related by using the WI value. Or for an SSI80 request, indicate that the returned SOUTs be consolidated such that a single SOUT is returned for each WSI value.

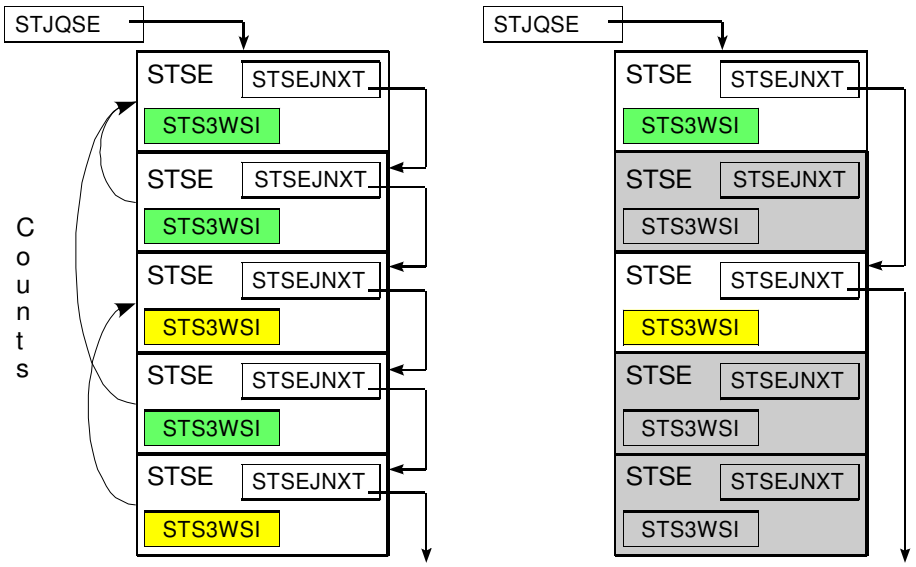
Only the SOUTs returned for the current SSI80 request are consolidated. SOUTs returned for previous SSI80 requests are not affected.

Extended Status SSI - WSI fields

- **SSI80 updated.**
  - STS3WSI = WSI returned in JES3 portion of a SOUT.
  - STAT1WSI = Input flag to consolidate and return one SOUT per WSI.
  - STS31WSI = Output flag indicating one SOUT returned for the WSI.
  - SYSOUT token for a consolidated SOUT updated to include the WSI.
    - When used for a SYSOUT verbose request or data set list request, all data sets from all OSE variable sections with the same WSI are returned.

Extended Status SSI - WSI consolidation

- Before…
- After…



With SOUT consolidation, the SOUTs are still returned in storage. However, JES3 has changed the linkage to skip SOUTs with duplicate WSI values. The counts from the skipped SOUTs are added the consolidated SOUT. The SYSOUT token for a consolidated SOUT updated to include the WSI.

Extended Status SSI - Other Considerations

- **You can set STATTERS (job info) and specify SYSOUT filters**
  - Only STATJQs for jobs that have output that matches the filters are returned (no STATSEs returned)
- **STATSEs have a JOE level token (STSTCTKN) that can be passed to SAPI to process the output**
- **STJ2SPOL can be passed to SPOOLIO SSI to read the JCT for the job**
- **Verbose calls are limited to one Job at a time**
  - Except if jobid list is used
- **STATVOs have a data set token (STVSCTKN) that can be passed to SAPI or SPOOL browse to process the data set**

All filters can be used on all call types. That implies that SYSOUT filters can be used when job level information is being requested. When this is done, only jobs that have output that matches the filter will be returned (and only STATJQs are returned).

The STATSE contains a JOE token (STSTCTKN) that can be passed on the SYSOUT API (SAPI) to select a specific output group for processing.

The STATJQ contains a SPOOL token (STJ2SPOL) and a job key (STJ2JKEY) that can be passed to the SPOOLIO SSI to read the JCT for a job.

Verbose calls can only process a single job at a time. This is to limit the impact on the system of the I/Os needed to obtain the data for the request.

The STATVO contains a data set token (STVSCTKN) that can be passed on the SYSOUT API (SAPI) to select a specific data set or on the SPOOL browse interface to allocate that specific data set. Note it is not the same as the token passed in ENF 58 and should not be used for that purpose.

Extended Status SSI - STATTRSA

- **Use STATTRSA to expand terse sections with verbose section**
  - Use terse request to locate jobs/output you need
  - Only get expensive verbose data when needed
  - Set STATTRSA to STATJQ or STATSE
  - Set request type (STATTYPE) to either
    - STATVRBO – verbose job data
    - STATOUTV – verbose SYSOUT data
  - New STATVE and STATVO sections added to chain

Verbose data provides much more information about a job and its output. However, the I/O needed to obtain the data can be expensive. If you do not need verbose data for all jobs or output elements (or if you can defer the requests until needed) then you can use the field STATTRSA to expand an existing STATJQ or STATSE. Point STATTRSA to the section for which you want verbose data, set the request type to either return verbose job data (STATVRBO) or verbose SYSOUT data (STATOUTV), and make your extended status request. The SSI will chain the appropriate data areas to the control block you passed.

Extended Status SSI - "OR" Filtering (Filter lists)

- **"OR" filtering**
  - Job name, Job ID, Job class, Job destination, Job phase, SYSOUT class, and SYSOUT destination
  - First value in existing SSST field
  - Count of additional values and pointer added to SSST
  - For example, to filter on the job classes A, B, C, or D you would set:

```
STATCLSL = C'A       ' First class to filter on
STATCLSN = F'3'        Number of additional classes
STATCLSP = A(CLASSLST) Pointer to class list
CLASSLST = CL8'B       ',CL8'C       ',CL8'D       '
                       List of 3 additional 8 byte classes
```

For performance reasons, the ability to pass multiple values for certain filters was added. These "OR" filters apply to job name, class, destination, and phase as well as SYSOUT class and destination. The first of the list of values is placed in the traditional location. Additional values are pointed to by a pointer later in the SSST. There is also a count field for the number of extra values specified.

Extended Status SSI - Job ID list

- **Job name and job ID lists are mutually exclusive**
  - Single input list pointer (STATJBNN/STATJBNP)
- **For job ids list all requested IDs in list**
  - STATJBIL and STATJBIH are not used
- **Can be used with terse or verbose requests**
  - Can get multiple jobs for verbose
- **List is ONLY JES job ids not transaction**
- **JES2 can use live version if not asking for SYSOUT information**

One of the filter list functions is the JOBID list. It is different from other lists because it is not a supplement to a single value but instead a complete replacement. The job ID list uses the same pointer and count as the job name list and thus they are mutually exclusive. It cannot be use with the low and high job ID filters STATJBIL and STATJBIH. It is permitted for both terse and verbose requests (including data set list) and is the only way to get multiple jobs on a verbose request. It is a list of JES job ids and will not be matched to transaction job ids.

If you are using JES2 and you are requesting job or data set information, then JES2 will use a live version for all the jobs requested. This is also the only way to get a list of jobs and use a live version.

Extended Status SSI - Program example

- **STATUS2 is the example program**
  - TSO command
  - "STATUS2 ?" gives a brief explanation of options
- **Output is a hex display (dump format) of**
  - SSOB and extension
  - All output areas returned (STATJQs, STATVE, STATSEs, and STATVOs)
- **Useful to see format of data returned**

The STATUS2 program is an example of how to use the extended status SSI. STATUS2 is a TSO command that takes as input the various filters that are supported by extended status and builds an IAZSSST (SSOB) to invoke the SSI. The command syntax can be obtained by issuing "STATUS2 ?".

The output of the command is very primitive. The SSOB as well as any STATJQs, STATVEs, STATSEs, and STATVOs are displayed in hex format with the EBCDIC translation (similar to what is seen in a dump). This provides an easy way to look at what is returned from the SSI call.

SJF Services SSI – SWB Modify and Merge

- **Requests JES SJF services**
  - SSI function 70 (IAZSSSF mapping macro)
  - SWBs are parameters specified on OUTPUT JCL cards
  - Multiple subfuntions supported
    - Modify (SSSFSWBM) alters existing SWBs
    - Merge (SSSFSWBF) reads SWBs
      - Merges JOE and DATA set SWBs in JES2
    - Storage return (SSSGSWBC) after done with merge data
- **Supports unauthorized callers**
  - SAF JESSPOOL check for access to data
- **Directed SSI (does not require job structure)**

JES SJF services (SSI 70) supports the SWB modify and merge (read) function. SWBs are data areas build by z/OS to represent JCL characteristics. JES uses them to contains the characteristics associated with a SYSOUT data set from the OUTPUT JCL card. This SSI supports subfunctions to alter or delete characteristic or to merge characteristics and return the SWBs to the application.

The modify function works on an output group (JOE basis) in JES2 and a SYSOUT data set instance basis in JES3..

This is an unauthorized SSI using JESSPOOL SAF checks to confirm access to the data. It also supports directed SSI requests. Callers are not required to have a job structure associated with the JES that is processing the request.

SJF Services SSI – SWB Modify

- **Modify request function (SSSFSWBM)**
- **Identify data set or group to modify**
  - JES2 must be JOE to modify
    - Group/JOE token from extended status
    - Job id, job name, and JOE id
  - JES3 must pass a data set token
    - From extended status, SAPI, or allocation
- **Pass 2 SWB lists**
  - Erase list removes selected SWBs
  - Modify lists alters values of specified SWBs
- **JES2 processing occurs in JES2 address space**
  - No direct feedback of success once queued to JES2
- **JES3 processing on global**
  - Request is synchronous
- **3 types of security checks**
  - Normal JESSPOOL check
  - SDSF like destination owner check (ISFAUTH class)
  - SECLABEL dominance check only (Authorized callers only) (JES2)

The SWB modify function (SSSFSWBM) allows an application to modify the OUTPUT JCL characteristics for SYSOUT.

In JES2, modifications must be done at the output group level (JOE level). The application specifies what JOE to modify by either passing in a JOE token (from extended status) or a job ID, jobname and JOE id. The changes made in JES2 apply to all data sets in the JOE.

In JES3, modification occurs at a data set instance. You must pass in a data set token (from extended status, SAPI or allocation) to identify what data set instance to modify.

The changes that are to be made are specified in 2 SWB lists that are passed in. The first is an erase list (a list of SWBs, by key, that are to be removed from the data set or JOE) and a modify list (a list of SWBs and their values to be used to override any SWBs with matching keys or to be added to the list of SWBs).

In JES2 the bulk of the processing for this SSI occurs in the JES2 address space asynchronous from the requesting address space. There is no feedback of whether the request did or did no work from JES2.

In JES3, the bulk of the processing for this SSI occurs on the JES3 global. This is done synchronously. The results of the processing is returned to the application.

There are 3 types of security checks for this request. The first is a standard JESSPOOL class check with an entity name of:

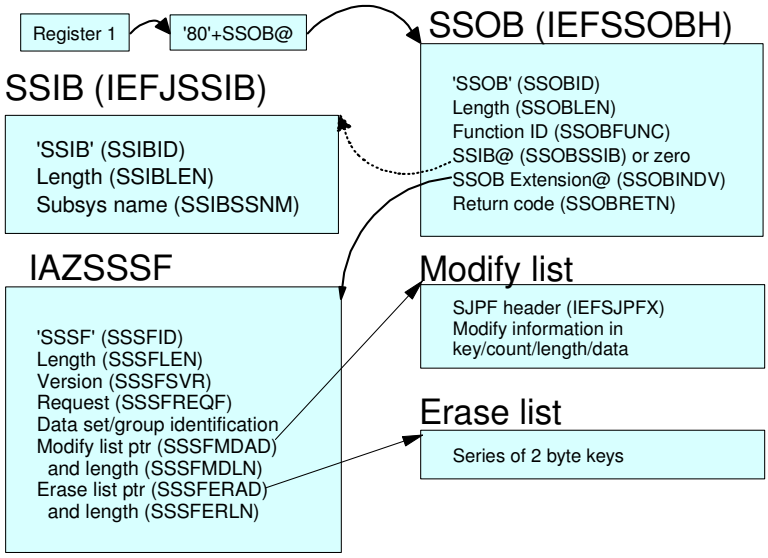
node.userid.jobname.jobid.GROUP.*groupname*

The second option is uses a profile that is compatible with the SDSF destination owner check (in the ISFAUTH class). This has an entity name of:

ISFAUTH.DEST.*destname*

Finally, a basic SECLABEL dominance check is available for authorized callers that want to do their own security checking (JES2 only)

SJF Services SSI – SWB Modify



This is the data areas associated with the SWB modify SSI. It is the normal data area associated with an SSI request. The SSOB extension points to two lists. The first is a modify list. This starts with a header mapped by IEFSJPFX. That is followed by a series of key, count, length, data TUs that represent the attributes the caller wants changed. The other list is an erase list. This is a series of 2 byte keys that the caller wants removed from the output.

In JES2, this SSI does not return any information. It just validates the input and queues it to the JES2 address space for processing.

In JES3, processing occurs on the global synchronously with this request. Success or failure is returned to the caller.

SJF Services SSI – SWB Modify

- **SWBMOD is a TSO command**
  - Parameters indicate
    - What job to modify (job name, id and output group)
    - Which subsystem to direct request to
    - List of TUs (by name) to erase
    - TUs to update and the new values
  - Output is dump format of input data
  - SWBMOD ? Gives a brief help display

SWBMOD is the sample program for the SWB modify function of the SJF services SSI. It is a TSO command with parameters to indicate what output group to modify, a list of TUs to erase from the output group, and a list of TUs to update and their values. The output of the command is a dump format display of the input parameters. No output information is formatted since there is no real output from the request. The data is displayed after the SSI call completes (and contains any return codes received). SWBMOD ? Gives a brief help display.

SJF Services SSI – SWB Merge (Read)

- **Merge (read) request function (SSSFSWBF)**
  - Reads SWBs for a data set
    - Indentified by passed data set token
  - Optionally merges in group level SWBs (JES2)
    - Identified by JOE token
  - Output is SWBTUs and optionally non-SWA SWBs
    - Same output format as SWBs from SAPI
  - Compatibility SWBs are optional
- **When done with output cleanup call needed**
  - Function SSSFSWBC frees associated storage
- **SECLABEL dominance checks performed (JES2 Only)**
  - Security is same as extended status
  - Only accessing meta data for job

The SWB merge (read) function (SSSFSWBF) allows an application to access the SWB data for a SYSOUT data set, In JES2 this includes the ability to merge the JOE level SWBs (created by SWB modify) with the data set level SWB. JES3 stores all SWBs at the data set level. The data set whose OUTPUT SWB data is to be returned is identified by a data set (or client) token. In JES2 the JOE to merge into the data set is identified by a JOE token.

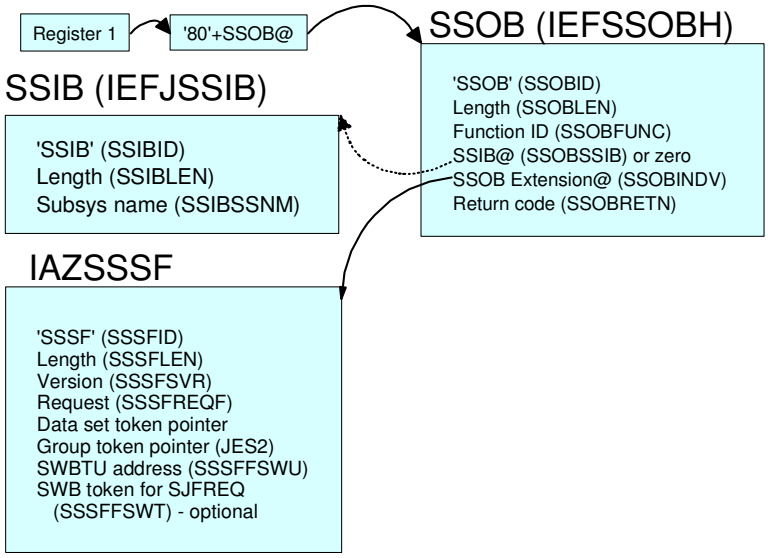
This service always output a pointer to a SWBTU list for processing. There is also an option to request non-SWA SWBs be built and a token returned. The token is then used with the SJFREQ service to access the SWBs. These are the same options available to SAPI.

The application can also request that compatibility SWBs be returned. This causes SWBs to be built for characteristics (such as class) that may be set at the DD level and not present on the OUTPUT statement.

Once you have completed processing the output of this SSI, there is a cleanup function (SSSFSWBC) that must be called to free the memory associated with the request/

Since this SSI only returns meta data (characteristics) of the output, the only security check is a SECLABEL dominance check in JES2 similar to what is done in extended status.

SJF Services SSI – SWB Merge (Read)



This is the data areas associated with the SWB merge (read) SSI. It is the normal data area associated with an SSI request. The SSOB extension points to the data set and optional JOE token (JES2) that identified the SWBs to read. The output is a pointer to a SWBTU block and a SWB token that can be used on SJFREQ requests.

Summary

- **Gone through brief overview of newer SSIs**
  - Some less new than others
- **Examples are a good starting point for how to code to interfaces**
- **Real value comes in combining multiple SSIs**
- **More features are being added all the time**
- **Using the Subsystem Interface great starting point**
  - Looking at DSECTs can fill in holes

The completes the quick tour of the newer SSI requests (and a couple older requests that may have been overlooked). The examples provided are intended as a starting point to build applications that interact with JES2. Though the examples tend to stress one interface at a time, the real value comes in combining interfaces to develop more complete applications.

These are not dead interfaces, they are constantly being updated with new features. New features are discussed at SHARE and in the Using the Subsystem Interface book. That book is a great resource for learning about the interfaces. However, looking at the DSECT that are involved in the interface often fills in some of the missing holes. SSI 71 is a great example. It is a router for many JES2 functions, not all are fully documented. Here is a complete list of the functions available and the related data areas:

```
SSJIFOBT (4)   IAZDSERV   CKPT version obtain
SSJIFREL (8)   IAZDSERV   CKPT version return
SSJIFJCO (12)  IAZJBCLD   JOBCLASS info obtain
SSJIFJCR (16)  IAZJBCLD   JOBCLASS info return
SSJISIOM (20)  IAZSPLIO   Read SPOOL block
SSJISIRS (24)  IAZSPLIO   Free SPOOL block storage
SSJICVDV (28)  IAZCVDEV   Convert device id
SSJIMNOD (32)  IAZMOND    Function Monitor info obtain data
SSJIMNRS (36)  IAZMOND    Function Monitor info return storage
```

There are lots of things you can do with the SSI, all it takes is a little exploring

Questions?

Session 09762

Backup charts

These charts discuss other SSIs/JES interfaces and may not be as current as the carts above. Also, these charts are more of a JES2 view. However, most functions work in both JES2 and JES3.

ABEND 1E0

- **Unauthorized callers could result in more ABENDs in SSI code**
  - Application developers developing new functions
- **Concern over the number of SVC dumps for application coding errors**
- **If SSI ABENDs 0C4 while accessing user passed data**
  - SSI will not take SVC dump
  - ABEND S0C4 will be changed to a S1E0
  - Recovery processing will percolate to caller
  - Caller can debug using SYSMDUMP or SYSUDUMP, etc.
- **This applies to authorized and unauthorized callers**
- **Treat 1E0 ABENDs as errors in the SSI caller (application) rather than system errors.**

One of the concerns with opening up the SSI to unauthorized callers was in the process of writing code, people make mistakes. If they pass bad data into the SSI, then the SSI may ABEND with an S0C4 causing JES2 to take an SVC dump. These application errors could flood a system with uninteresting SVC dumps. To prevent this, the JES2 recovery code converts all 0C4 ABENDs accessing user passed data to S1E0 ABENDs (a new ABEND code). JES2 recovery code does not take an SVC dump, and it percolates the error to the caller (rather than passing back a bad return code on the SSI). This change applies to all SSI callers (authorized and unauthorized). It is expected that installations will treat S1E0 ABENDs as application errors (or errors in the caller of the SSI) rather than a system error or JES2 problem. The documentation for the S1E0 ABEND is intended to make this clear to the application developer.

SYSOUT API (SAPI) SSI

- **Allows applications access to SYSOUT**
  - SYSOUT must not be active (busy)
  - Can only access non-NJE routed SYSOUT
  - Similar to PSO but with more function
- **SSI function 79 (IAZSSS2 mapping macro)**
- **3 primary functions**
  - PUT/GET - accesses individual data sets
  - Count - returns various counts from JOEs
  - Bulk Modify - Alters SYSOUT characteristics
- **Filters control what JOEs are processed**
- **Address space must be known to JES**

The SYSOUT API (SAPI) provides a rich set of services that provide information about SYSOUT on the system. It is intended as an enhancement to the PSO interface (which is no longer being enhanced). The primary function of SAPI is to act as a printer application. Since it is a printer application, it can only access non-NJE bound output that is not already busy on another device. However, it can access output that is held (JCL type hold vs non-selectable hold) as well as non-held. This is to be compatible with some PSO functions.

Note: non-selectable output can be selected by SAPI if selecting by client token (SSS2SCTK) and SSS2SBLK is set.

Other functions available via SAPI are a count request and a bulk modify request. Count requests simply look at all the JOEs that match a selection criteria and count the number of elements that match and accumulate various counts (e.g. lines, pages, etc.). This is useful if you are trying to determine how much output is waiting to be processed.

Bulk modify is similar to the PSO group request. It allows you to make changes to the characteristics of a set of JOEs with one SAPI call. With bulk modify you can alter the SYSOUT's class, destination, or release SYSOUT from held status. In addition you can delete SYSOUT.

What SYSOUT data sets to process is controlled by various filters specified in the IAZSSS2.

The SAPI SSI requires that the requesting address space have a job structure associated with the target JES2.

SYSOUT API (SAPI) SSI – SSOB (IAZSSS2)

- **IAZSSS2 identifies:**
  - Function to be performed
  - Filters to select output to be processed
  - Output information associated with request
- **One request may need multiple SSI calls**
  - A thread is a set of related SSIs
  - First call clears SSS2JEST
  - Last call sets SSS2CTRL on
  - If SSS2JEST is non-zero when done, SSS2CTRL call is needed
  - Errors can cause JES2 to terminate thread

The IAZSSS2 contains both input and output fields. The input includes the function to be performed and a set of filters to select which SYSOUT data sets are to be processed. The output information includes the job and SYSOUT characteristics, pointers needed to allocate data sets, SWB information (data on OUTPUT JCL cards), and other fields that may be needed to process the SYSOUT. Which input fields are valid and output fields that are returned are based on the function requested. PUT/GET calls return the most data, bulk modify returns just a success or failure indicator.

When a SAPI request is made, an environment is created to process this request. The assumption is that multiple requests will be made using this same environment. The set of requests that share this environment is called a thread. A new thread is started when a SAPI request is made with the field SSS2JEST set to zero. This thread is maintained until either the task that created the thread terminates, or a SAPI call is made with the bit SSS2CTRL being set on. One task can have multiple threads associated with it. Threads can be passed between tasks (however it is always owned by the creating task). Under certain error scenarios, a thread can be terminated by JES2, but normally, it takes a successful SAPI request with SSS2CTRL set. When SSS2CTRL is set, the function requested is not performed, except if a JOE is currently assigned to the thread (from a previous PUT/GET call). Then normal put processing will occur. Setting SSS2CTRL on a call with SSS2JEST set to zero does nothing.

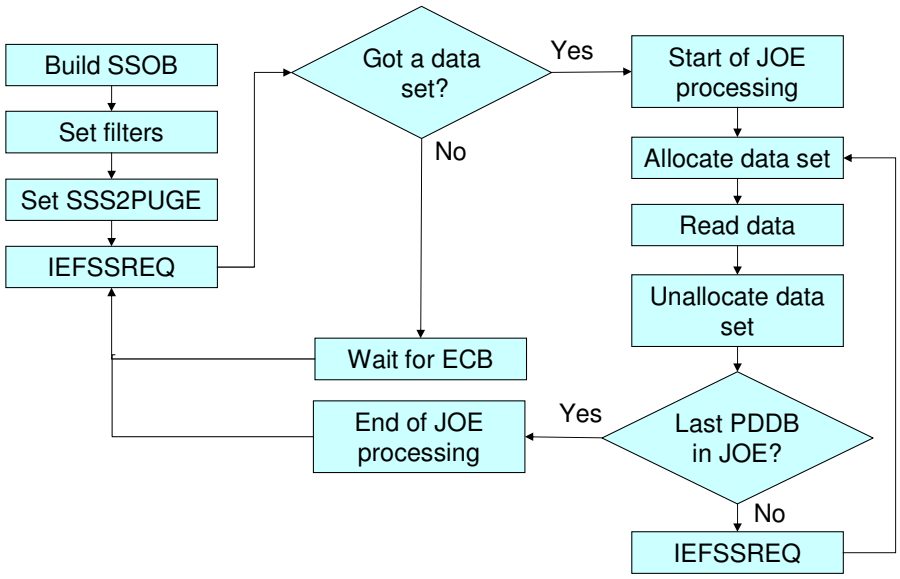
SYSOUT API (SAPI) SSI - Security

- **PUT/GET and Bulk modify calls perform JESSPOOL RACF checks**
  - One check per data set (NOT at JOE level)
  - Update or read access based on SSS2SRON
    - Update needed if characteristics change or data set deleted
  - Data sets (PDDBs) that fail check are ignored
  - Can cause one JOE to be broken into multiple
    - 5 PDDB JOE, requester cannot access 3rd PDDB
    - Bulk modify request to change class
    - Result 2 JOES, #1 has PDDBs 1, 2, 4, 5; #2 has PDDB 3.
- **No RACF checks for count request**

SAPI performs RACF checks to ensure that the requester has access to the SYSOUT. The checks are the standard JESSPOOL class checks made elsewhere in JES2. By default, update access is requested. However. If SSS2SRON is set, then only read access is requested. Update access is only needed if the data set characteristics are being changed or the data set is being deleted. The checks are made at the SYSOUT data set level (PDDB) not the JOE level. This implies that an application may be able to access some data sets in a JOE but not others. Data sets which the application does not have access to are ignored. The application has no way to know that there are data set to which it has no access. Security failures can cause a single JOE to be split into multiple JOEs. For example, if there is a 5 PDDB JOE in which the application has access to all but PDDB 3, then a bulk modify request to change the class of this data set can only effect PDDBs 1, 2, 4, and 5. So when the request completes, there will be 2 JOEs, one with PDDBs 1, 2, 4, and 5 and one with just PDDB 3.

The security checks apply to PUT/GET and bulk modify request. They do not apply to count requests.

SYSOUT API (SAPI) SSI - Typical logic (PUT/GET)



This logic shows a typical application that selects JOEs that match some criteria, process the JOEs, then wait for more JOEs to become available. As illustrated, this is an unending process. A real application would have a command to halt the process or it would exit when no work was found.

The processing starts with setting the selection criteria, indicating it is a PUT/GET calls, then invoking the SAPI SSI. If no SYSOUT is returned, then the process waits for JES2 to post when work is available. If a SYSOUT data set was returned (a JOE), then processing starts with setup for the new JOE (header processing, data set allocation, etc.). The first data set is allocated, opened, and read. Once it has been read and processed, the data set is closed and unallocated. If there are more data sets in this JOE, then another SAPI SSI call is made to dispose of the first data set and get then next data set. If the last data set for the JOE (SSS2DSL is on) was just processed, then cleanup processing for the JOE is done. Another SAPI SSI call is made and the last data set in the current JOE is disposed of and a new JOE is obtained (if one is available) and the processing continues.

SYSOUT API (SAPI) SSI – Put/Get Advanced topics

- **SYSOUT characteristics**
  - Most JOE/PDDB fields returned
  - SWBs (TUs or SWA) available
  - NJE job and data set headers
  - RACF security token
- **Disposition processing**
  - Keep or "process" the data set
  - Hold the data set (system or JCL)
  - Change attributes
  - Don't show to thread/address space again

The output area in the IAZSSS2 contains all the traditional characteristics of the SYSOUT and owning job. These are the fields obtained from the JOE, JQE, JCT, and PDDB. In addition, the output SWBs (data from OUTPUT JCL statements) and the NJE job/data set headers are also made available (the NJE headers are only available if the job or output arrived via NJE). The RACF security token associated with the output is also made available.

When the SAPI application completes processing of a SYSOUT data set, there exist a number of options (indicated in SSS2DISP) regarding its disposition. These options are:

•to "process" the data set (same as a printer does when it prints a data set)

•keep the data set (with or without a system hold)

•JCL hold the data set

•release the data set

•ask that the data set not be shown to this address space or thread again

The do not show options apply until the instance (address space or thread) terminates or until some attribute of the output is changed ($T command). Additionally, there are a number of attributes associated with the output that can be altered..

SYSOUT API (SAPI) SSI – Put/Get Advanced topics

- **Do not have to completely process JOEs**
  - Can change selection criteria mid JOE
  - Can change request type mid JOE
- **One thread can process count, bulk modify, and PUT/GET requests**
  - If selection or request changes, next call does a PUT with old rules and then new request
  - Acts like a "PUT/COUNT" or "PUT/BULK" request

When selecting JOEs for processing, you do not have to completely process the entire JOE you were handed. You can choose to process some data sets and then stop, or change the parameters (filters or call type) on the next request. For example, perhaps you want to process all output for a job once you find a SYSOUT data set that matches a particular filter. You could do a selection based on a criteria like SYSOUT class, then when a JOE is selected, change your filters to the JOBID of the output you were handed and do a COUNT request (to see how much output there is for this job), then based on the counts returned, create a regular data set large enough to handle the output, and finally start doing PUT/GET calls with the JOBID filter. When all the available data sets for the job have been processed, you could update your filter to be just a SYSOUT filter again and look for another job. Note, when you switch requests (in this case change the type to count), you must define what to do with the data set you were handed (in this example, you would indicate to keep the data set). The count request in this case is really a "PUT/COUNT" request. It PUTs the JOE you were handed and then COUNT with the new filter.

SYSOUT API (SAPI) SSI – Put/Get Advanced topics

- **All SYSOUT token types valid input for SAPI selection**
  - JOE tokens – extended status
  - Client tokens – Allocation
  - Data set tokens – SAPI or extended status
- **Terminate thread using SSS2CTRL**
  - IAZSSS2 still validated
  - Check return code because request can fail
  - When in doubt, use SSS2PUGE type (least likely to fail)
  - Thread terminated when SSS2JEST is zero

Processing using SAPI can be enhanced by using SYSOUT token to select specific SYSOUT data sets to process. SYSOUT tokens take 3 forms:

•JOE tokens represent an entire JOE and can be obtained from the extended status interface. This allows the application to use extended status to determine which JOEs to process (using whatever selection process they want) and then using SAPI to actually process the data sets.

•Client tokens are obtained at the time a dynamic allocation creates a SYSOUT data set. These tokens represent a single SPIN data set. Using client tokens, an application can directly allocate for processing a specific data set that was created earlier.

•Data set tokens are part of the SYSOUT characteristics that are returned when SAPI selects a data set for processing or on an extended status verbose SYSOUT request.
One way these tokens can be used is by selecting OUTPUT using normal SAPI selection but not actually processing the data set. The data set tokens can be saved and the data set returned with KEEP specified (and perhaps do not show to thread again). At a later time, data sets can be selected in the order that the application wants for processing. Or the data sets can be read using SPOOL browse in any order that the application wants and later deleted.

All SAPI requests create a thread. These threads must be terminated when no longer needed. Failing to terminate threads can increase system overhead. Threads are normally terminated when the creating task terminates. However, if multiple threads are being processed by one long running task, then the application must terminate the threads that are no longer needed by issuing a SAPI request with SSS2CTRL set. These calls go through normal validation processing but the only processing that occurs is the PUT portion of a PUT/GET. A thread exists if the creating task exists and SSS2JEST is non-zero. This is important because request with SSS2CTRL set can fail. When this happens, you may need to issue a request with SSS2PUGE set since it performs the least validation.

SYSOUT API (SAPI) SSI – Put/Get Advanced topics

- **What about CLONE JOEs?**
  - Multiple JOEs point to same set of PDDBs
  - Clone JOEs created when
    - /*JOBPARM COPIES=
    - $N PRTnnnn
  - Splitting PDDBs from one JOE would affect other (CLONE) JOE
  - Restriction: All data sets in a CLONE JOE MUST be process the same
  - SSS2DSH is on if processing CLONE JOE

Clone JOEs exist when multiple JOEs point to the same set of PDDBs. Third qualifier in JOE id indicates whether a JOE may be a clone. JOE with ids of 1.1.1 and 1.1.2 are clones. Clone JOEs are created when JOBPARM COPIES= is specified (greater than 1) or a printer is repeated ($N PRTnnn command). They can also be created by an exit that does a $#ADD passing in an existing JOE as the prototype. The problem with clone JOEs is caused by the fact that PDDBs are associated with JOEs using the first 2 identifiers of the JOE id. So with clone JOEs, one PDDB is actually associated with multiple JOEs. If SAPI were to move a PDDB to a new JOE, it would not only affect the JOE being processed, but also the clone JOE. This side effect can cause data to be lost. As a result of this, there is a restriction when processing clone JOEs that all data sets must be processed the same way (you cannot split data sets out of a clone JOE). SAPI sets the SSS2DSH bit when it is processing a clone JOE.

SYSOUT API (SAPI) SSI – Program examples

- **There are 3 sample TSO programs**
  - All are TSO commands
  - SAPICNT is sample count request
  - SAPIBULK is a sample bulk modify
  - SAPIOUT is a sample PUT/GET command
- **Using a "?" option displays operand information**

There are 3 example programs for SAPI. Each program explores a different type of the SAPI request. All are TSO commands. SAPICNT and SAPIBULK are examples of count and bulk modify requests. The two examples are similar except that SAPIBULK includes modification operands that SAPICNT does not. SAPIOUT is a very primitive OUTPUT command that processes data sets that match the input criteria and displays the output on the terminal.

All samples take as input a "?" which displays simple help.

SPOOL Browse

- **Not an SSI, rather parameters on dynamic allocation**
  - Triggered by DALBRTKN allocation key (Browse token)
    - Mapped by IAZBTOKP
- **Can access any data set on SPOOL**
  - SYSIN, SYSOUT, JCL
- **Data set can be busy on a device**
- **Data set can still be open**
  - Instorage (unwritten) buffers are available
- **Allocation can be by client or data set token**
- **Can also allocate a concatenated SYSLOG**
  - All data set for MVS SYSLOG still on SPOOL

SPOOL browse is not an SSI, but rather a subsystem dynamic allocation. Processing is triggered by an allocation key that contains a browse token (mapped by IAZBTOKP). This browse token, along with the data set name, identifies the data set to browse. SPOOL browse can access any data set on SPOOL including SYSIN and SYSOUT data sets and the input JCL. Data sets can be busy on a device, or even still open by the creating address space. For data sets that are still open, the unwritten (instorage) buffers can be read. Browse can also use client or data set tokens to allocate a data set.

Browse can also be used to allocate a concatenated SYSLOG for an MVS system by specifying a special data set name.

SPOOL Browse - IAZBTOKP

ID length (4)
ID ('BTOK')
Version length (2)
Type (0-3)
Version (3)
SPOOL token length (4)
SPOOL token
Job key length (4)
Job key
ASID length (2)
ASID
Receiver length (8)
Receiver
Log string length (0-255)
Log string

- **There are 3 token types:**
  - 0 or 1 - Original browse token
    - SPOOL token is IOT MTTR
  - 2 - Reserved for SAPI
  - 3 - SYSOUT Token
    - SPOOL token is client or data set (SAPI or extended status) token
- **Receiver and Log string are for RACF checks**

The browse token is used to identify to JES2 what data set to allocate. There are 2 forms available to applications (the third is reserved for use by SAPI). The first form sets a type field to either x'00' or x'01'. This form can be used by applications that do not have a data set or client token available. The other form that can be used by applications is a type 3 token (type field of x'03'). In this form, the application passes the address of either the client or data set token of the data set to allocate in the IAZBTOKP. Data set tokens can be obtained using either SAPI (SSS2DSTR) or extended status (SYSOUT verbose section field STVSCTKN).

Format of the browse token is length/data/length/data... The length values are verified by dynamic allocation processing and must contain the values indicated.

Receiver and log string parameters are used for RACF calls. Unauthorized callers can only pass a receiver value that is set to their userid.

SPOOL Browse - IAZBTOKP type 1

- **Type 1 calls with SPOOL token = 0**
  - DSN= is name of data set to allocate
    - can contain generic characters
    - DSN must have jobname and jobid
    - Job key is optional (will be validated if passed)
- **Type 1 calls with SPOOL token <> 0**
  - Token is MTTR of IOT with PDDB for DS
  - DSN= name of data set (no generics)
  - Job key is required

Type 1 calls come in 2 flavors. If SPOOL token=0 (BTOKIOTP=0) then JES2 will locate the data set to allocate from the data set name passed. The data set name is the standard JES2 SPOOL data set name

userid.jobname.jobid.Ddskey.dsname

In this flavor, the jobname and jobid must be specified. But other fields may be specified as generics. The minimum data set name is

.JOBNAME.J0123456.

If a job key is passed (BTOKJKEY), then it is used to ensure that the data set allocated is associated with the correct job. However, the job key can be passed as zero.

The other flavor of type 1 calls specify an IOT spool address in the SPOOL token (BTOKIOTP = IOTMTTR). In this flavor, the full data set name must be specified with no generic, and the job key must be set. This is lower overhead than the first flavor but requires more knowledge of JES2 internals.

SPOOL Browse - IAZBTOKP type 3

- **Type 3 calls pass a SYSOUT token**
  - BTOKSPLT can be client or data set token
    - Client token from allocation
    - Data set token from
      - SAPI - SSS2DSTR
      - Extended status - STVSCTKN
  - JOE token cannot be used
  - Job key (BTOKJKEY) is ignored
  - DSN= is also ignored

Type 3 calls pass in a SYSOUT token (client or data set) in BTOKSPLT. These are the tokens passed from a client token allocation (key DALRTCTK), from a SAPI request (field SSS2DSTR) or extended status (SYSOUT verbose section field STVSCTKN). JOE tokens from extended status are not supported. For type 3 calls, the job key and DSN= passed on the allocation are ignored.

SPOOL Browse - IAZBTOKP other fields

- **ASID (BTOKASID)**
  - If zero, no unwritten buffers obtained
  - If non-zero, attempt to get unwritten buffers
    - ASID of running address space that is writing SYSOUT
    - x'FFFF' have JES2 figure out ASID and system
- **Receiver (BTOKRCID)**
  - Passed to RACF as RECEIVER= on AUTH call
- **Log string (BTOKLSDL & BTOKLSDA)**
  - Passed to RACF as LOGSTR=

JES2 will attempt to obtain unwritten buffers if BTOKASID is non-zero. It can either specify the ASID on local system where data set is being written, or a x'FFFF'. A value of x'FFFF' tells JES2 to figure out what ASID and system the job is running on.

Receiver (BTOKRCID) is the userid the output is routed to (passed as RECEIVER= on RACF AUTH call). If the value passed is the same as the owner of the SYSOUT (in the token associated with the job) then RACF will permit access without needing a profile in the JESSPOOL class. Unauthorized callers can only pass a receiver value that is set to their userid.

The log string (BTOKLSDL and BTOKLSDA) is information passed on the RACF AUTH call. This value is placed in the audit record for the request.

SPOOL Browse - Special DSN=

- **Some nonstandard DSN= are supported**
  - Type 1 calls with SPOOL token = 0

userid.jobname.jobid.JCL
userid.jobname.jobid.JESJCLIN
userid.jobname.jobid.JESJCL
userid.jobname.jobid.JESMSGLG
userid.jobname.jobid.JESYSMSG

  - JESMSGLG and JESYSMSG attempt to link spun JESLOG data sets

To allocate system data sets without having to know the data set key, you can use the special data set names specified. These data set names are only valid if the corresponding data sets exist and the data set was not created on a JES3 system and sent via NJE. JCL and JESJCLIN refer to the input JCL for the job. JESJCL is the JCL listing (JCL images) that were output from the converter. JESMSGLG and JESYSMSG are the JES2 message log and system messages data sets. If the job specifies JESLOG=SPIN, the JESMSGLG and JESYSMSG data set names attempts to link together the SPIN data sets into one logical data set for processing.

SPOOL Browse - Special DSN=

- **SYSLOG for a system can be access**
  - All SYSLOG jobs on SPOOL concatenated in time order
- **Special data set name passed in**
  - *sysname*.SYSLOG.SYSTEM
    - *sysname* is the MVS system name not the JES2 member name
- **Slightly modified JESSPOOL SAF/RACF check**
  - node.userid.jobname.*SYSTEM*.sysname
    - *userid* is SYSLOG userid (+MASTER+)
    - *jobname* is SYSLOG job name (SYSLOG)

Another special data set name that can be passed to SPOOL browse is the SYSLOG data set name. This name contains the MVS system name whose SYSLOG you want to browse. This logical data set includes all the SYSLOG data sets on SPOOL (across multiple jobs).

The RACF check for accessing this logical data set is not the same as the data set name itself. The data set name allocated only has the system name variable, The name used for RACF checks is more consistent with the standard JESSPOOL data set name.

SPOOL Browse - Allocation TUs

- **Required allocation TUs**
  - 'Subsystem request' to allocate a data set
    - DALSSREQ – authorized callers only
    - DALUASSR – all callers (authorized or not)
  - DALBRTKN - Browse token
  - DALDSNAM - Name of the data set
  - DALSTATS - DISP = SHR
  - DALRTDDN - Return DDNAME - (optional)

When invoking allocation, the listed keys are needed. The name of the subsystem that the allocation is to be directed to must be specified as DALSSREQ or DALUASSR. The only difference between the 2 keys is that DALUASSR supports unauthorized callers.

SPOOL Browse - SYSLOG POINT options

- **New POINT options (SYSLOG data set ONLY)**
- **RBA passed start with a x'FF' indicates new option**
  - Second byte indicates type of POINT
  - Remaining 6 bytes specify where to POINT to
    - Can be a time stamp
      - First 6 bytes of a STCKE (system time)
    - Can be a record number
      - Relative or absolute
- **Old format RBA values still work**
  - Value returned on GET
    - JES2 format is 00mmtttt rrnnnnn

The new POINT options allow an efficient way to locate a record by record number or time stamp. The format of the RBA used for POINT has been changed for this support. The first byte of the RBA is set to x'FF' to indicate that one of the new options is being requested. The next byte of the RBA is set to the option requested. The last 6 bytes is the operand that indicates where to POINT to. The 6 bytes can be a time stamp (the first 6 bytes of an STCKE) or a record number (absolute or relative). These options use an index that JES2 writes when the SYSLOG data set is created. This index is read in when the SYSLOG data set is allocated and updated as needed when reading the SYSLOG data set. Using this technique, JES can quickly (with minimal I/O) locate the requested record.

The old format RBA still works to locate a specific record. This is still the fastest and lowest overhead way to locate a specific record. However, this support eliminates the need for an application to maintain an index of RBAs returned from GET and instead uses the index that JES creates and maintains.

SPOOL Browse - Program example

- **BROWSE is sample application**
  - Batch program (not TSO application)
  - WTOR used to
    - Specify JES2 data set name
    - Whether instorage buffers are wanted
  - Allocates data set
  - Uses IEBGENR to copy selected data set to output DD

The sample program uses WTORs to determine what data set to allocate and whether instorage buffers are wanted. It then allocates the data set (a type 1 token is used with SPOOL token=0). If the allocation succeeds, then IEBGENR is used to copy the SPOOL data set to an output DD.

Who-am-I SSI

- **Gets subsystem information**
  - SSI function 54 (IEFSSVI mapping macro)
  - Supports unauthorized callers
  - Directed SSI (does not require job structure)
- **Returned information has fixed and variable section**
- **WHOAMI is program example**
  - Program invokes SSI and issue WTOs

The Who Am I SSI (Subsystem Version Information) gets information about a subsystem. It supports unauthorized callers and directed SSI requests. Callers are not required to have a job structure associated with the JES2 that is processing the request. The output has a static section that is common to all subsystems. That is followed by a variable section with subsystem dependent data. The variable section has a KEYWORD='VALUE' format that is similar to REXX assignment statements.

The example program issues the SSI request and issues WTOs with the response.

Who-am-I SSI - Data returned

```
SSVIVERS=z/OS 1.9
SSVIFMID=HJE7740
SSVICNAM=JES2
NO USER DATA PRESENT,
JES_NODE='POK',JES_MEMBERNAME='IBM1',
DYNAMIC_OUTPUT='YES',INITIATOR_RESTART='YES',
MULTIPLE_STCTSO='YES',FOUR_DIGIT_DEVNUMS='YES',
AUTO_RESTART_MANAGER='YES',SAPI='YES',
SAPI_CHARS='NO',CLIENT_PRINT='YES',
TSO_SYSOUT_CLASS='G,H',
WTR_SYSOUT_CLASS='A,B,C,D,E,F,I,J,K,L,M,N,O,P,Q,R,S,T,
  U,V,W,X,Y,Z,0,1,2,3,4,5,6,7,8,9',
COMMAND_PREFIX='$'
```

This is the output from the sample WHOAMI SSI. The first 3 fields listed are in the base section of the SSOB (IEFSSVI). The later fields are the variable data that is returned.

Notify SSI

- **Send notify message to user**
  - SSI function 75 (IAZSSNU mapping macro)
  - Directed SSI (does not require job structure)
- **Destination can be user on another node or member**
- **Optional SAF token passed on SSI**
  - Associated with message being sent
  - Authorized callers only
- **MSG is program example**
  - TSO command to send a message to a node.userid
  - MSG ? gives quick help

The notify SSI can be used by an application to send a notify message to a user in the MAS or on another NJE node. The caller can pass a SAF token to associate with the message for authentication purposes. This token is only supported for authorized callers.

The sample program is a TSO command that issued a message to a *node.userid*.

JES2 Job Information SSI – SPOOL Read Subfunction

- **SSI reads blocks from SPOOL**
  - Subfunction of SSI function 71 (IAZSSJI mapping macro)
  - Functions SSJISIRS and SSJISIOM (IAZSPLIO mapping macro)
  - Directed SSI (does not require job structure)
- **Any MTTR can be read from SPOOL**
- **JESSPOOL or JESJOBS SAF check**
  - Required if caller not authorized
  - Optional for authorized callers

SPOOL read is a subfunction of the job information SSI 71. The SSI 71 SSOB extension is mapped by IAZSSJI. SSI 71 acts as a router for various JES SSI. The extension has a function code to identify what subfunction is needed and a pointer to a function dependent data area. The functions for SPOOL read are SSJISIOM (read data area) and SSJISIRS (free work areas).

SSI 71 supports directed SSI requests. Callers are not required to have a job structure associated with the JES2 that is processing the request.

This SSI will allow any record on SPOOL to be read including signature records associated with tracks. The SSI deals with relative vs. absolute track addressing.

JESSPOOL or JESJOBS SAF check are made for unauthorized callers. The same checks are optional for authorized callers.

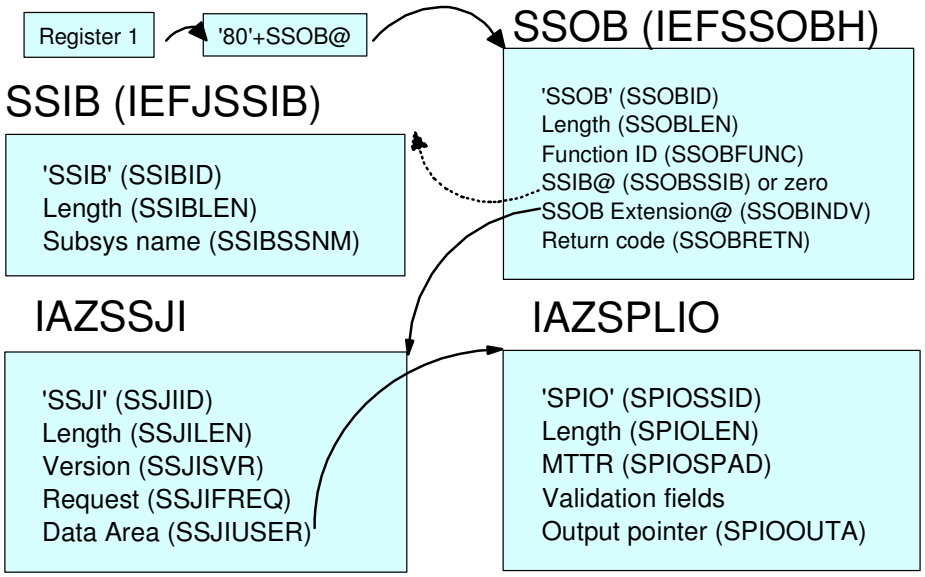
JES2 Job Information SSI – SPOOL Read Subfunction

- **SAF authorization check**
  - Optional if authorized caller (default is no RACF call)
  - Always done if unauthorized caller
  - Entity name and class used depends on control block
    - JESSPOOL for control blocks for CBs JES2 can find RTOKENs for *node.userid.jobname.jobid*.SPOOLIO.*cbname*
    - JESJOBS for control blocks for CBs without RTOKENs (JES2 RTOKEN passed)
      SPOOLIO.*node.jobname.jobid.cbname*
      - UNKNOWN can be used if JOBNAME is not available
      - JOBID always starts with 'J' or 'JOB' and could be JOB00000
      - UNKN can be used for *cbname* if the control block is not known
- **Passwords in JCT are cleared before it is returned**

The SAF authorization call uses either the JESSPOOL or JESJOBS class. The FUNCODE for exit 36/37 is $SEASPLR, and the control block that was read is passed to the exit. The auth call is always made if the caller is not authorized. Authorized caller can set the SPIORACF bit in SPIOOPT to force the RACF call. The entity name and class passed on the auth call depends on the SSI being able to locate the security token (RTOKEN) associated with the user that owns the control block. The SSI code will read at most one additional data area to locate the control block. If the associated security token is located, then a JESSPOOL check will be made passing in the profile listed above and the RTOKEN found. If the associated security token is not found, then a JESJOBS class call is made passing the JES2 address space security token as RTOKEN.

One thing to note, the code will “clear” the password fields in the JCT before returning them to the caller. The password is zeroed if there is no password in the fields and set to X’FF’ is there was a password specified. This adds additional protection to the password fields.

JES2 Job Information SSI – SPOOL Read Subfunction



This is the data areas associated with the SPOOL read SSI. It is the normal data area associated with an SSI request plus the extra function dependent data area (IAZSPLIO).

JES2 Job Information SSI – SPOOL Read Subfunction

- **SPOOLRD is batch program**
  - Uses WTORs to determine what to read
    - MTTR, CB type, JOBNAME, JOBID, Job key, Data set key
    - Asks whether instorage buffers are needed
  - Output is WTOs about what was read
  - Print is dump format of record read

The SPOOLRD example is a batch program that reads a single block from SPOOL. The input is obtained using WTORs. It includes not only the MTTR to read but also any validation fields that are to be passed. The ASID of a running address space can be passed to obtain unwritten (instorage) HDB data areas (only valid when a control block type of HDB is passed).

The output is a brief display via WTOs of what was read and a print data set that contains a hex dump of the record read.

JES2 Job Information SSI – Monitor Info Subfunction

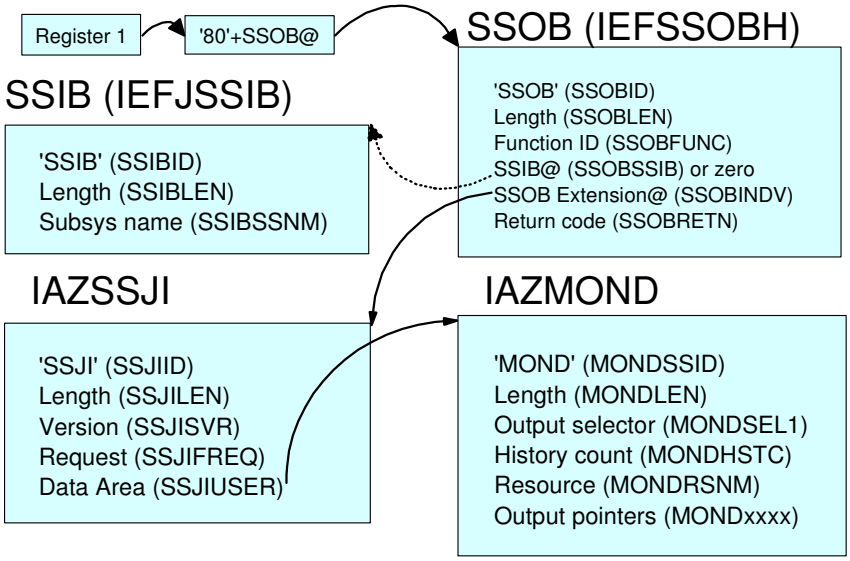
- **Subfunction on the JOB information SSI (SSI 71)**
- **2 functions, get monitor information and return storage**
- **Data area (IAZMOND) maps input and returned data**
- **Returns all information available via monitor commands**
  - Resource usage statistics
  - Main task CPU statistics
  - JES2 ERROR statistics
  - JES2 address space storage usage statistics
  - Main task WAIT statistics
  - JES2 Alerts
  - JES2 Notices
  - JES2 Tracks
  - Monitor status information
- **Some additional data returned**

Monitor information is a subfunction of the job information SSI 71. The SSI 71 SSOB extension is mapped by IAZSSJI. SSI 71 acts as a router for various JES SSI. The extension has a function code to identify what subfunction is needed and a pointer to a function dependent data area. The functions for monitor information are SSJIMNOD (obtain monitor information areas) and SSJIMNRS (free work areas).

SSI 71 supports directed SSI requests. Callers are not required to have a job structure associated with the JES2 that is processing the request.

This SSI can return current tracks, alerts and notices that are being maintained by the JES2 monitor. It also returns various statistics that the JES2 monitor maintains. The caller of the SSI can select which information they want returned and how much history to return. In addition, for resource usage, the caller can specify what resources the SSI returns.

JES2 Job Information SSI – Monitor Info Subfunction



This is the data areas associated with the monitor information subfunction. It is the normal data area associated with an SSI request plus the extra function dependent data area (IAZMOND).

JES2 Job Information SSI – Monitor Info Subfunction

- **MONSSI is a TSO command**
  - Parameters indicate what data to return
    - Indicate what data areas to return
    - Filter on resource name
    - Indicate amount of history to return
  - Output is dump format of input data and data returned
  - MONSSI ? Gives a brief help display

MONSSI is the sample program for the monitor subfunction. It is a TSO command with parameters to indicate what data needs to be returned to the caller. Also included are options to filter on resource name and to indicate how much history to return. There is also an option to direct the SSI to a specific subsystem. The output of the command is a dump format display of the input parameters and the data returned. MONSSI ? Gives a brief help display.