JES2 & JES3 z/OS 1.7: SSI and Other Common Functions Enhancements

SHARE 105, Session 2658
Wednesday, August 24, 2005

Tom Wasik
JES2 Design/Development/Service
Poughkeepsie, NY
wasik@us.ibm.com

JES2 & JES3 z/OS 1.7: SSI and Other Common Function Enhancements
A number of enhancements have been made to SSIs supported by JES2 and JES3 in z/OS 1.7. In particular, the extended status SSI can now return additional job and data set level information (including JES2 PDDB level information). JES2 also has made some minor enhancements to the SAPI SSI and added a new SSI to obtain JES2 health monitor data. This session will discuss those changes and will be of interest to users of JES2 and JES3 alike.

Table of Contents

Today we will discuss the enhancements to the extended status and SAPI SSIs and a new SSI to obtain information from the JES2 health monitor. The extended status enhancements are available in both JES2 and JES3 while the other changes are currently available in JES2 only.

What is Extended Status?

- Extended status provides JES (2 or 3) information about selected jobs.
- The interface is similar to SAPI
  - Selection criteria (filters) provided by caller
  - Information returned for the job(s) matching the filters
  - A mechanism to say “I’m finished”
  - Interface is via the SubSystem Interface (SSI) via SSI 80.

Extended status is more than just an extension to the original STATUS SSI (SSI 3). It is intended as a programming interface into both JES2 and JES3 to obtain not only status information but general information about jobs and SYSOUT. It can be used to obtain information to present on a front end such as SDSF or to programmatically check the status of jobs. It can be also used as a screener for a SAPI print application to select work to process next.
The mapping macro used for SSI 80 is IAZSSST. z/OS 1.7 introduced a new version level of the IAZSSST (version 4). The new information presented here is only available if a version 4 IAZSSST is passed on the interface.

A Little SAPI History

- **OS/390 R3 Original implementation**
  - Select jobs based on job level criteria
  - Return information about jobs only
- **OS/390 R4**
  - Add Service Class criteria and returned information

Extended Status was added in OS/390 release 3 to support the TSO GR (Time Sharing Option Generic Resource) line item. That line item required that a given time sharing user could be found in the JESplex and to give back the MVS system name where the user was logged on.
OS/390 release 4 (the “WLM release”) added information about service classes.

A Little SAPI History

- **OS/390 R5 – Added output group**
  - Select jobs based on output group characteristics (Show me all jobs with output destined to Timbuktu)
  - Output group information (for JES2 show JOE level information)
  - Selecting all output with certain output group selection criteria would show all JOE level and the job level information

OS/390 release 5 added (for JES2) JOE based selection criteria and JOE based returned data. Support to access output data was later added to JES3.

This support added SYSOUT level filters and the ability to get information about SYSOUT elements (JOEs in the case of JES2). The use of filters was independent of the data being returned. You could ask for only job level data and pass in SYSOUT level filters. However, you cannot get SYSOUT data without also getting JOB level data.

So What's New?

- **Ability to access additional JOB data**
  - Details not available in JES2 checkpoint
  - Bulk data that requires extra processing
    - JCT needs to be read
- **Ability to access data set level SYSOUT info**
  - Details in the JES2 PDDB
  - Info on each data set in a JOE
    - IOT needs to be read to get PDDB
- **Live ckpt used for single job request (STATTERS or STATVRBO)**
  - Most current information returned

Prior to z/OS 1.7, extended status only returned data that was easily obtainable (in the case of JES2, this implied that the data was in the checkpoint). But applications that want to display job information may need data that is not in the checkpoint. The changes made in z/OS 1.7 added the ability to access data that is on SPOOL. In particular, information in the JCT can be obtained for a job. For SYSOUT, information on each data set in a JOE (PDDB level information) can also be obtained. Included in the SYSOUT data is a data set token that can be passed into SAPI to access a data set directly.
Another enhancement is if job level information is requested for a singe job (STATTERS or STATVRBO) and no SYSOUT filters are passed, then JES2 will automatically use a "live checkpoint version". This is actually a shadow copy of the actual checkpoint data areas. By doing this, JES2 can obtain the latest possible job level information for the job being requested. Since this is a live version, JES2 cannot run any chains and thus cannot return anything but JQE and JCT level information and cannot filter on any SYSOUT characteristics.

IAZSSST Output areas

- **Data returned in sections that look very much like NJE headers.**
- **There are four major sections**
  - Job terse (SJQE)
  - Output group terse (SOUT)
  - Job Verbose (SJVE) NEW!
  - Output data set Verbose (SSVE) NEW!
- **The verbose sections require I/O**

The output areas for extended status look like NJE headers that have a prefix that is used for chaining. Each section has:

•Type fields

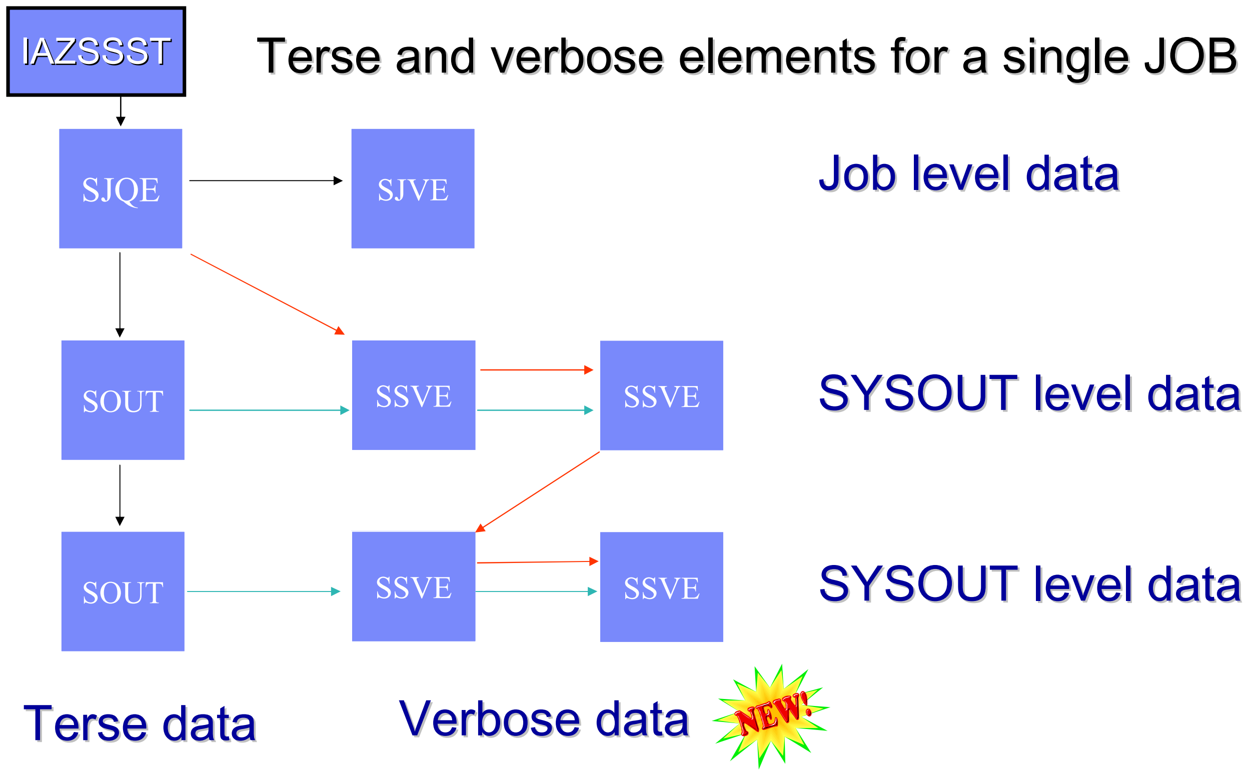
•Modifier fields (to differentiate two sections with the same type)

•Length fields so that the sections can be traversed easily

When moving from section to section, be aware that not all sections exist for all output types and that the lengths in the sections should be used to advance to the next section and not the assembly time equates.

Job level verbose data comes from the JCT, data set level data comes from PDDBs (in IOTs). Therefore I/O is required to fulfill these requests.

IAZSSST Output Chaining



The SJQE (Job Terse) and SOUT (SYSOUT terse) sections are pre – z/OS V1R7.

The SJVE is the Job Verbose Element and is new with z/OS V1R7.

The SSVE is the SYSOUT Verbose Element.

SSVEs are chained in two ways: grouped under related Terse SYSOUT element (blue arrows) and sequential within the scope of an entire job (red arrows).

IAZSSST Filters

- **Jobs and the SYSOUT for jobs are selected via filters provided in the IAZSSST.**
- **New SYSOUT filters in z/OS 1.7**
  - Forms code
  - Process mode (PRMODE)
  - Spin attribute (select only spin or only non-spin)
- **New job filters in z/OS 1.7**
  - Submitting Userid (JES3 only)

There are many filters for job and output selection. There are four new filters in z/OS 1.7

1. Forms code
2. PRMODE
3. Spin attribute
4. Submitting userid (only supported in JES3 because the information requires an I/O in JES2)

Verbose Requests

- **Verbose data can be obtained for only one job per call to the SSI**
- **If verbose job level data, set STATTYPE to STATVRBO and …**
  - Job id (STATJBIL = STATJBIH) or (STATJBIL = jobid and STATJBIH = binary zero)

    or
  - SYSOUT token (client token, JOE token, or SAPI token)

    or
  - STATTRSA pointing to SOUT or SJQE

The interface is purposely designed to allow requesting data for only one job per call. The single job limit was implemented to prevent an application from requesting verbose data for all (or most) SYSOUT in the system. Because of the effort needed to retrieve that level of information, such a request may not complete for hours (or perhaps days). By requiring an application to make multiple requests, performance should not be impacted (since the extra SSI overhead is small relative to the overhead to get the needed data) and the actions of the application are more easily seen by external monitors. Also, since a checkpoint version is needed to complete the request, having a long request will result in the checkpoint version becoming out of data before the request completes.

If STATTRSA is zero, use STATSJBI with the filter value field STATJBIL set to the jobid of the job required (STATJBIH if supplied must be the same as STATJBIL)

If STATTRSA is zero, supply a sysout token (client token, JOE token or SAPI token). These tokens define a given data set or collection of data sets for a single job.

Supply a non-zero STATTRSA pointing to a SJQE or SOUT returned in an earlier request for data (and with no intervening “memory” call).

Verbose Requests *(Cont…)*

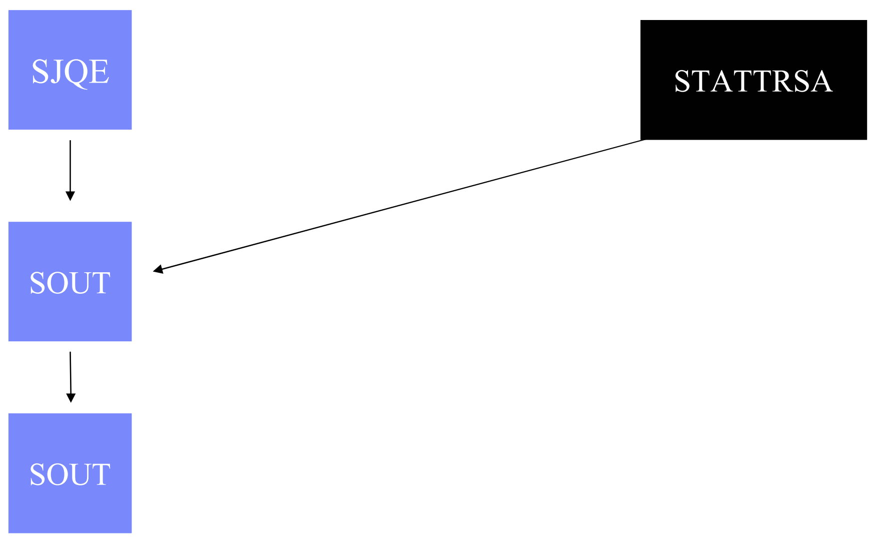
- **There are 2 ways to request verbose data**
  - Ask for it when you get the terse data
  - Get terse data first, determine what verbose data you need, and then add the verbose data to existing terse data (JES2 only)
- **Asking for terse data only when needed**
  - Can reduce time to complete initial (terse) request
  - Only perform extra processing when needed

There are 2 ways to get verbose data returned to you, You can simply ask for it at the same time you request the terse data (by specifying function STATVRBO or STATOUTV) or you can get the terse data first, determine what jobs/output you need more information on, and then request verbose data be added to only those jobs. By requesting verbose data only when needed, then you reduce the time to get the initial data and only go through the overhead of the verbose data when needed.

For example, lets say you are implementing a job display. Your display has both terse and verbose data. If you only display 24 jobs at a time, you could request terse data for all jobs that you want to display and then verbose data only for those that are actually displaying on the screen. As the user scrolls, you can request additional verbose data. This allows you to get the initial display up faster and will only slightly slow scrolling.

Adding Verbose Data Example

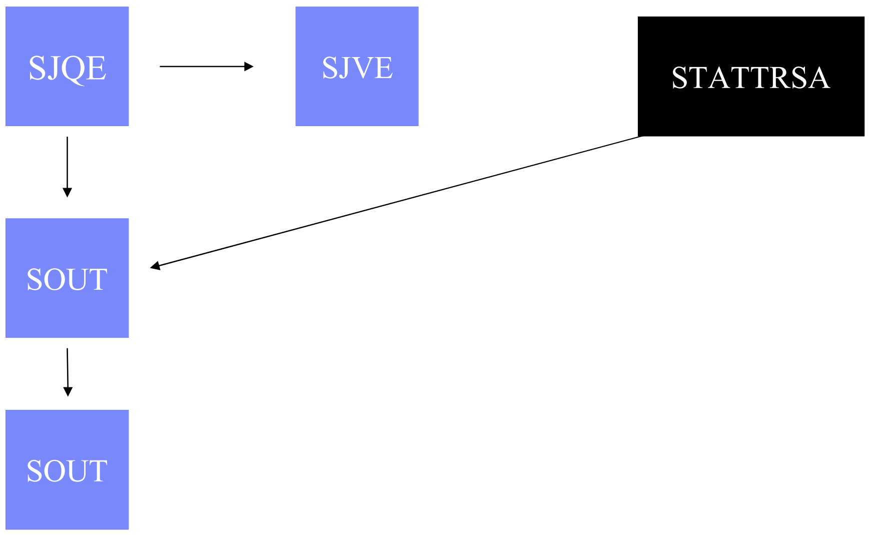
§ Here is an example of expanding a terse element (STATTRSA)



Here is an example of expanding a terse element in JES2. The application has previously done a terse output request and wants to expand the first output element that was returned from that request. The application puts the address of the desired terse element into STATTRSA.

Adding Verbose Data Example Results

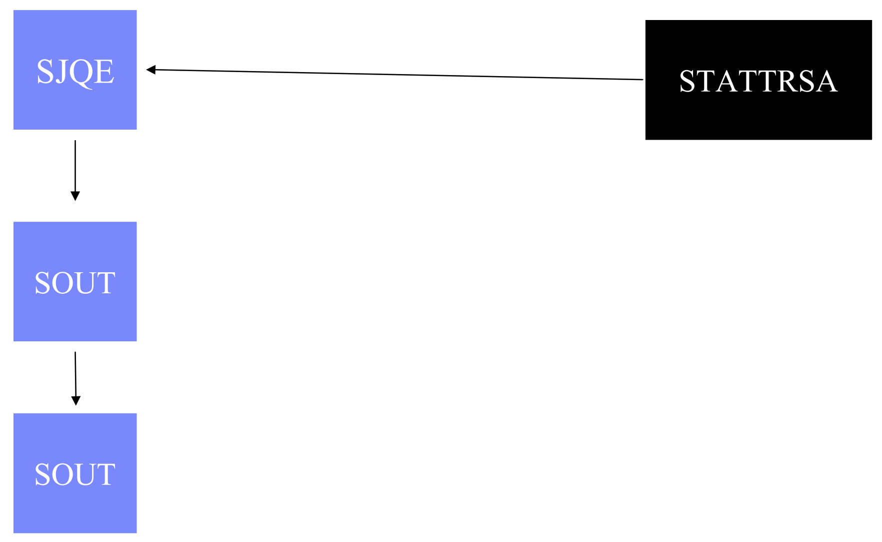
- Here is a layout of the structure after the requested expansion. (STATTYPE=STATVRBO)



Upon return from the expansion request, the job level SJVE is returned.

Adding Verbose Data Example

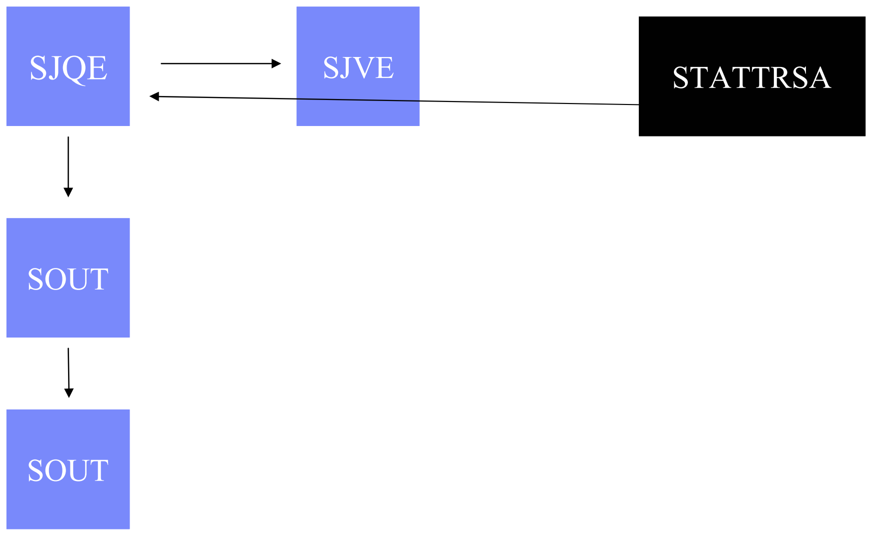
§ Here is an example of expanding a terse element (STATTRSA)


$$\begin{tikzcd}
\boxed{\text{SJQE}} \arrow[d] & \boxed{\text{STATTRSA}} \arrow[l] \\
\boxed{\text{SOUT}} \arrow[d] & \\
\boxed{\text{SOUT}} &
\end{tikzcd}$$

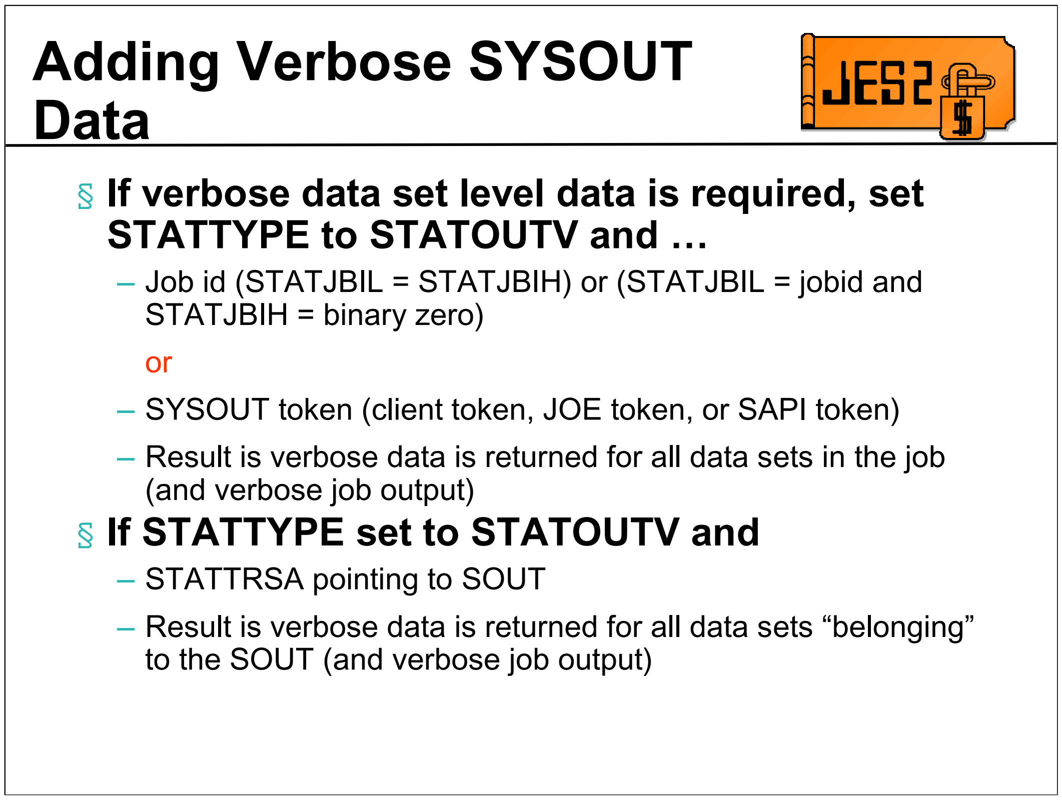
Here is an example of expanding a terse element in JES2. The application has previously done a terse output request and wants to expand the job level output. The application puts the address of the SJQE into STATTRSA.

Adding Verbose Data Example Results

- Here is a layout of the structure after the requested expansion. STATTYPE=STATVRBO



Upon return from the expansion request, the requested SJQE is extended with its verbose output (SJVE)



Adding Verbose SYSOUT Data

- **If verbose data set level data is required, set STATTYPE to STATOUTV and …**
  - Job id (STATJBIL = STATJBIH) or (STATJBIL = jobid and STATJBIH = binary zero)

  or

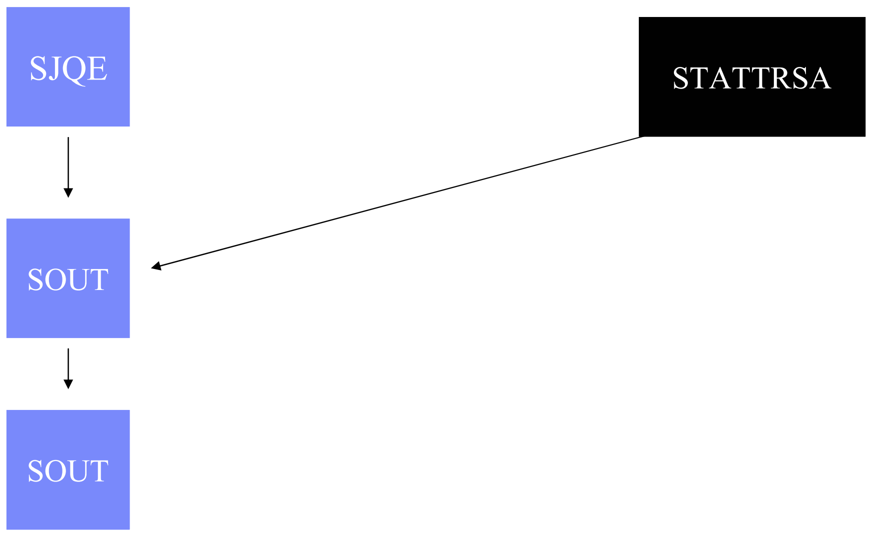
  - SYSOUT token (client token, JOE token, or SAPI token)
  - Result is verbose data is returned for all data sets in the job (and verbose job output)
- **If STATTYPE set to STATOUTV and**
  - STATTRSA pointing to SOUT
  - Result is verbose data is returned for all data sets “belonging” to the SOUT (and verbose job output)

Just as for the STATVRBO request, the single job can be selected by supplying the jobid or by supplying one of the SYSOUT tokens.

If STATTRSA points to a SJQE, then verbose data for **all** data sets of the job is returned. If STATTRSA points to a SOUT, then verbose output is returned for just that single SOUT.

Adding SYSOUT Verbose Data Example

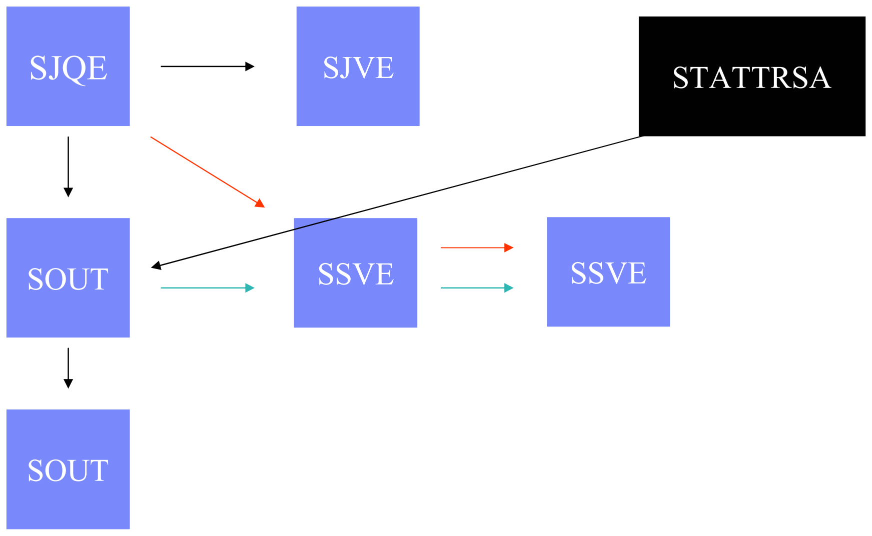
- Here is an example of expanding a terse element (STATTRSA)



Here is an example of expanding a terse element in JES2. The application has previously done a terse output request and wants to expand the first output element that was returned from that request. The application puts the address of the desired terse element into STATTRSA.

Adding SYSOUT Verbose Data Example Results

- Here is a layout of the structure after the requested expansion. (STATTYPE=STATOUTV)

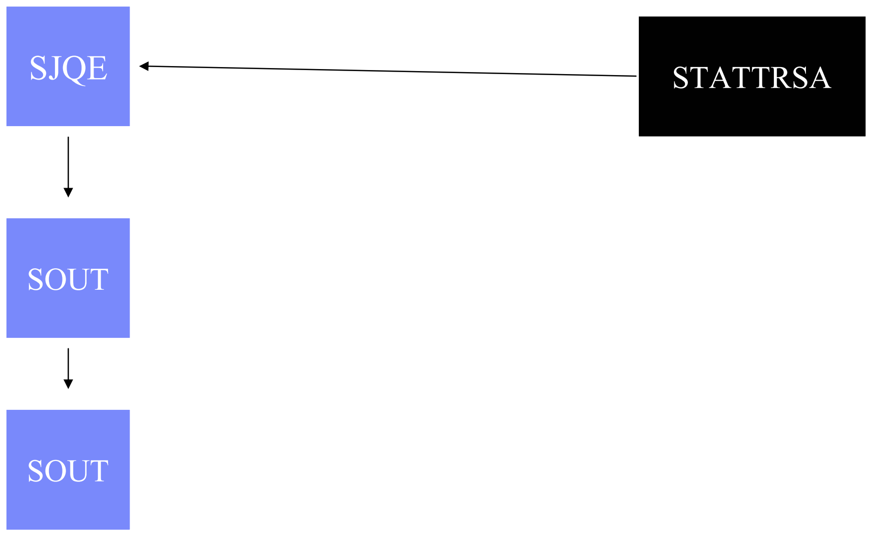


Upon return from the expansion request, the requested SOUT is expanded and its associated SSVEs are chained in.

Since a request type of STATOUTV returns both job and SYSOUT verbose information, the SJQE is also expanded and an associated SJVE is chained in.

Adding SYSOUT Verbose Data Example

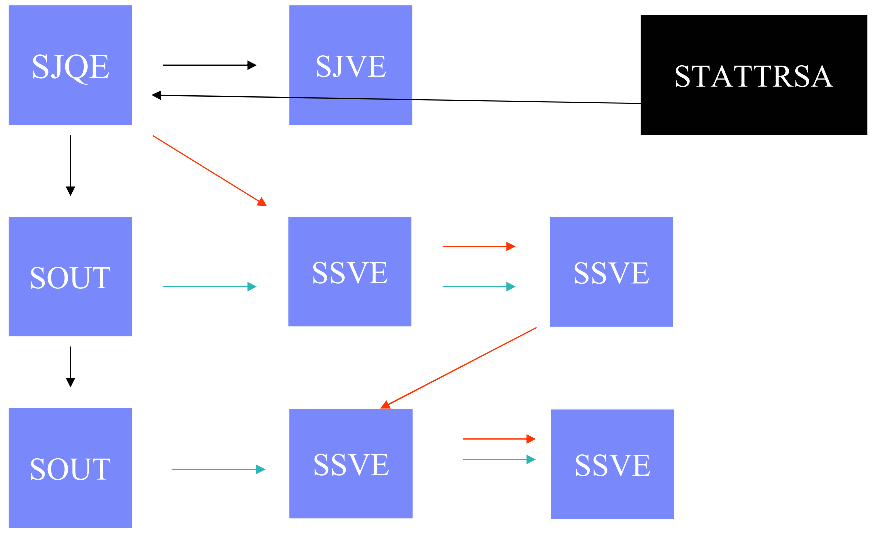
- Here is an example of expanding a terse element (STATTRSA)



Here is an example of expanding a terse element in JES2. The application has previously done a terse output request and wants to request information for all the data sets for the job. The application puts the address of the job level terse into STATTRSA.

Adding SYSOUT Verbose Data Example Results

- Here is a layout of the structure after the requested expansion. STATTYPE=STATOUTV



Upon return from the expansion request, all SOUTs are expanded and their associated SSVEs are chained in.

Since a request type of STATOUTV returns both job and SYSOUT verbose information, the SJQE is also expanded and an associated SJVE is chained in.

Extended Status Output Fields

- Here is a list of verbose job fields returned

STVBJCPY Job Copy count
STVBLNCT Job Line count
STVBIDEV Input device name
STVBISID Input system/member
STVBJCIN Job input count
STVBJLIN Job line count
STVBJPAG Job page count
STVBJPUN Job card (output) count
STVBRTST Input start time
STVBRTSD Input start date
STVBRTET Input end time
STVBRTED Input end date
STVBSYS Execution MVS system name
STVBMBR Execution JES2 member name
STVBXTST Execution start time
STVBXTSD Execution start date
STVBXTE Execution end Time/Date
STVBXTET Execution end time
STVBXTED Execution end date
STVBJUSR JMRUSEID field
STVBMCLS Message class (Job card)
STVBNOTN Notify Node
STVBNOTU Notify Userid
STVBPNAM Programmer's name (Job card)
STVBACCT Account number (Job card)
STVBDEPT NJE department
STVBBLDG NJE building
STVBROOM Job card room number
STVBJDVT JDVT name for job
STVBSUBU Submitting userid

Security section containing SAF token

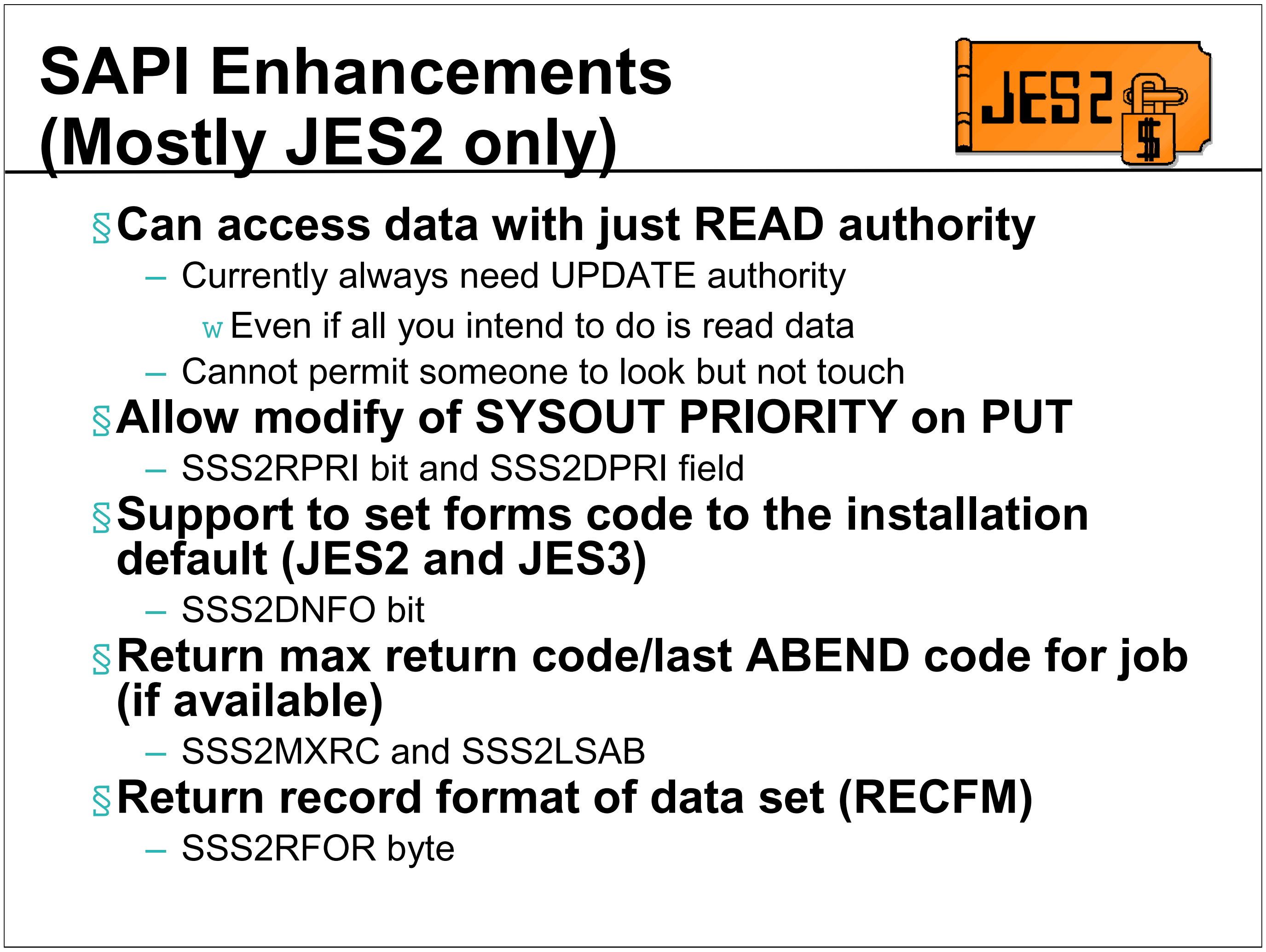
Accounting section containing accounting strings

Extended Status Output Fields *(Cont…)*

- Here is a list of verbose SYSOUT fields returned

STVSRECF Record format
STVSPRCD Procname for the step
STVSSTPD Stepname for the step
STVSDDND DDNAME for the data set
STVSTJN APPC Transaction jobname
STVSTJID APPC Transaction jobid
STVSTOD Date/time data set available
STVSSEGM Segment id (JES2 only)
STVSDSKY Data set number (key)
STVSMLRL Maximum LRECL
STVSLNCT Line count
STVSPGCT Page count
STVSBYCT Byte count after truncation
STVSRCCT Record count (JES3 only)
STVSDSN SYSOUT data set name
STVSCTKN SYSOUT data set token

Security section containing SAF token



SAPI Enhancements (Mostly JES2 only)

- **Can access data with just READ authority**
  - Currently always need UPDATE authority
    - Even if all you intend to do is read data
  - Cannot permit someone to look but not touch
- **Allow modify of SYSOUT PRIORITY on PUT**
  - SSS2RPRI bit and SSS2DPRI field
- **Support to set forms code to the installation default (JES2 and JES3)**
  - SSS2DNFO bit
- **Return max return code/last ABEND code for job (if available)**
  - SSS2MXRC and SSS2LSAB
- **Return record format of data set (RECFM)**
  - SSS2RFOR byte

The SAPI interface allows application to access SYSOUT data sets on SPOOL. The interface was designed to act similar to a printer. It can only access output that is not busy elsewhere and is not in an operator or application hold. When accessing a SYSOUT data set, it locks (busies) the data set blocking other applications from accessing the same data set.

Prior to z/OS 1.7, an application wanting to access a SYSOUT data set had to have UPDATE access to the data set. This was true even if the application had no intention of altering or deleting the data set. With z/OS 1,7, an application can indicate it intends to only read the data and SAPI will only require READ access to the data set.

In addition, application can modify the output priority during PUT processing, and set the forms code to the installation default forms.

The data returned to the requestor has been updated to include the max return code or ABEND code for the creating job as well as the record format of the data set being returned.

SAPI Changes

- **A SAF call is made when the SAPI interface is about to return a data set to the application.**
  - Prior to z7, the SAF call was made requiring that the application have UPDATE authority
  - In z7, if the application “promises” to make no changes to the data set, then the SAF authority required is READ
  - The promise is made by setting SSS2SRON
  - Return code of SSS2BDIS and reason code of SSS2RRON is given if the promise is broken

A SAF call is always made (except for COUNT calls) to ensure that applications can access a data set. Prior to z/OS 1.7, this call always requested UPDATE access to the data set. In z/OS 1.7, an application can set SSS2SRON to indicate it only requires READ access to the data set being returned. If the application has READ access, then the data set is returned. When the a PUT call is later done, a check is made to ensure that nothing about the data set needs to be changed (hold state, priority, forms, etc). If nothing needs to be changed then processing continues normally. However if READ access is requested and a change is made to any characteristic, then the PUT call will fail with a return code of SSS2BDIS and a reason code of SSS2RRON. This happened even if the caller had UPDATE authority but only requested READ access to the data set (it is based on what was requested not what authority the application has).

SAPI Changes

- **New returned data**
  - Highest condition code
  - Last ABEND code
  - Record format of the data set
- **New update capability**
  - Application can change the queueing priority of each data set (which in turn could result in the data set being removed from its current JOE)
  - Set FORM to installation default

New data is being returned in the IAZSSS2. The highest completion code or last ABEND code is being returned in SSS2MXRC and SSS2LSAB

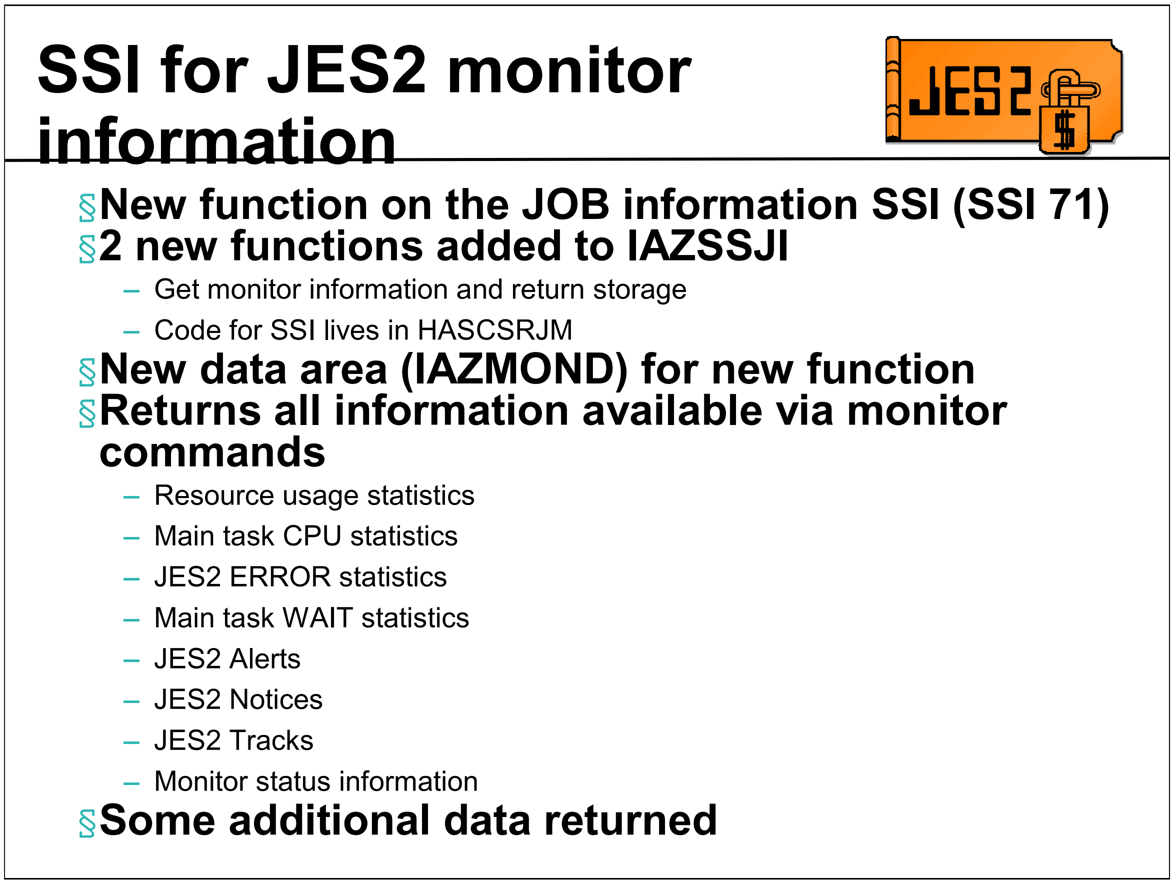
The record format of the data set is being returned in SSS2RFOR

In addition, when returning a data set (PUT call) the application can update the output priority of the data set, It can also request that the FORM associated with the data set be set to the installation default.

New Monitor SSI

- **Problem: The JES2 health monitor only displays data on the console**
  - Not practical way to study the data
  - Bad human factors
- **Solution: Provide monitor data via an SSI to applications**
  - Applications can format data as they want to
  - GUIs can be used to better show what is happening.

The current JES2 health monitor (or the JES2 monitor) only has a MVS command interface. Since it can produce hundreds of lines of output for a single command, it is not the best way to gather data to study to look for problems. This is especially true of the history information it maintains. What was needed is an API to pass the data to a GUI for a better presentation.

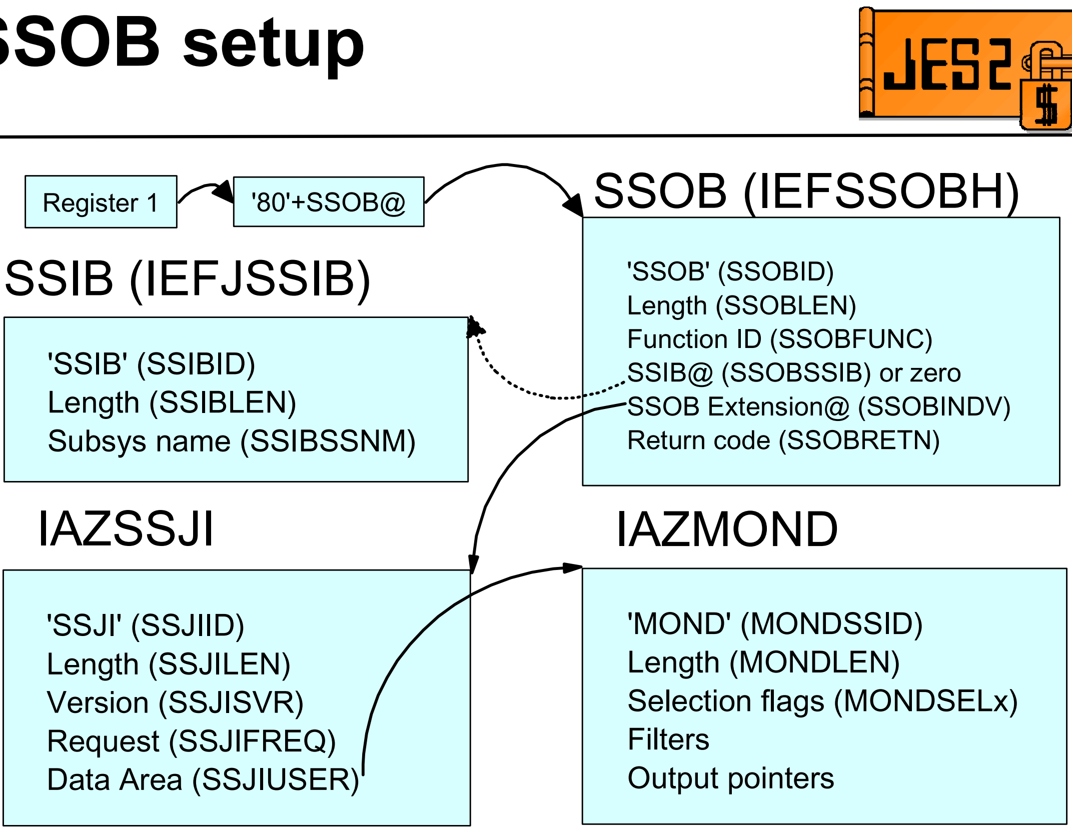


SSI for JES2 monitor information

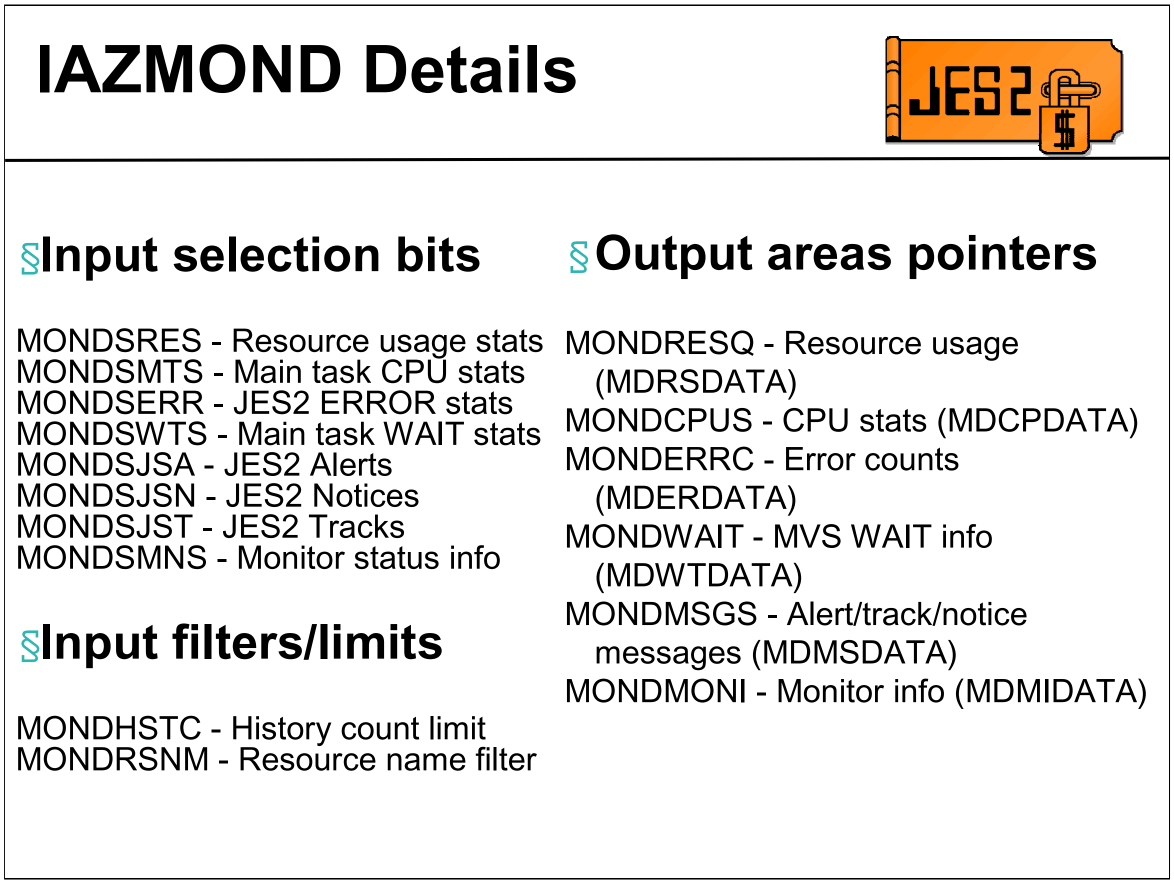
- New function on the JOB information SSI (SSI 71)
- 2 new functions added to IAZSSJI
  - Get monitor information and return storage
  - Code for SSI lives in HASCSRJM
- New data area (IAZMOND) for new function
- Returns all information available via monitor commands
  - Resource usage statistics
  - Main task CPU statistics
  - JES2 ERROR statistics
  - Main task WAIT statistics
  - JES2 Alerts
  - JES2 Notices
  - JES2 Tracks
  - Monitor status information
- Some additional data returned

The job information SSI will be expanded in this release to allow the return of JES2 monitor information to application. Information returned will be the same as what can be obtained via monitor commands. There are some additional fields that are returned to complete the data that is available to applications.

SSOB setup



These are the data areas needed to invoke the monitor SSI.



IAZMOND Details

§Input selection bits

MONDSRES - Resource usage stats
MONDSMTS - Main task CPU stats
MONDSERR - JES2 ERROR stats
MONDSWTS - Main task WAIT stats
MONDSJSA - JES2 Alerts
MONDSJSN - JES2 Notices
MONDSJST - JES2 Tracks
MONDSMNS - Monitor status info

§Input filters/limits

MONDHSTC - History count limit
MONDRSNM - Resource name filter

§Output areas pointers

MONDRESQ - Resource usage (MDRSDATA)
MONDCPUS - CPU stats (MDCPDATA)
MONDERRC - Error counts (MDERDATA)
MONDWAIT - MVS WAIT info (MDWTDATA)
MONDMSGS - Alert/track/notice messages (MDMSDATA)
MONDMONI - Monitor info (MDMIDATA)

This provides an overview of the IAZMOND input and output areas. The type of information that is available is similar to what you can obtain using monitor commands. The output area pointers generally point to a chain of output areas, each containing the information listed.

Output Data Areas

- **MDRSDATA - Resource usage data**
- **Chain pointed to by MONDRESQ**
  - Each element on chain matches a monitored resource
- **Each element has an array based on amount of history requested**
- **Filtering based on resource name (MONDRSNM) and amount of history (MONDHSTC)**
- **Each history element contains**
  - Time interval started, resource limit, current/last, low, high, and average usage, warn level, and percentage of time over warn level

The MDRSDATA contains information on the resources the JES2 monitors. The output elements are chained out of the MOND DSECT using the MONDRESQ queue head. Each element on the chain represents a resource. Each element also has an array of usage history (at least 1 long) for that resource. The elements returned and the amount of history is determined by the MOND fields MONDRSNM and MONDHSTC. Included in the history elements are the standard information on the $JD HISTORY command plus the warn level and the percentage of sampling intervals the resource was above the warn level in the interval.

Output Data Areas

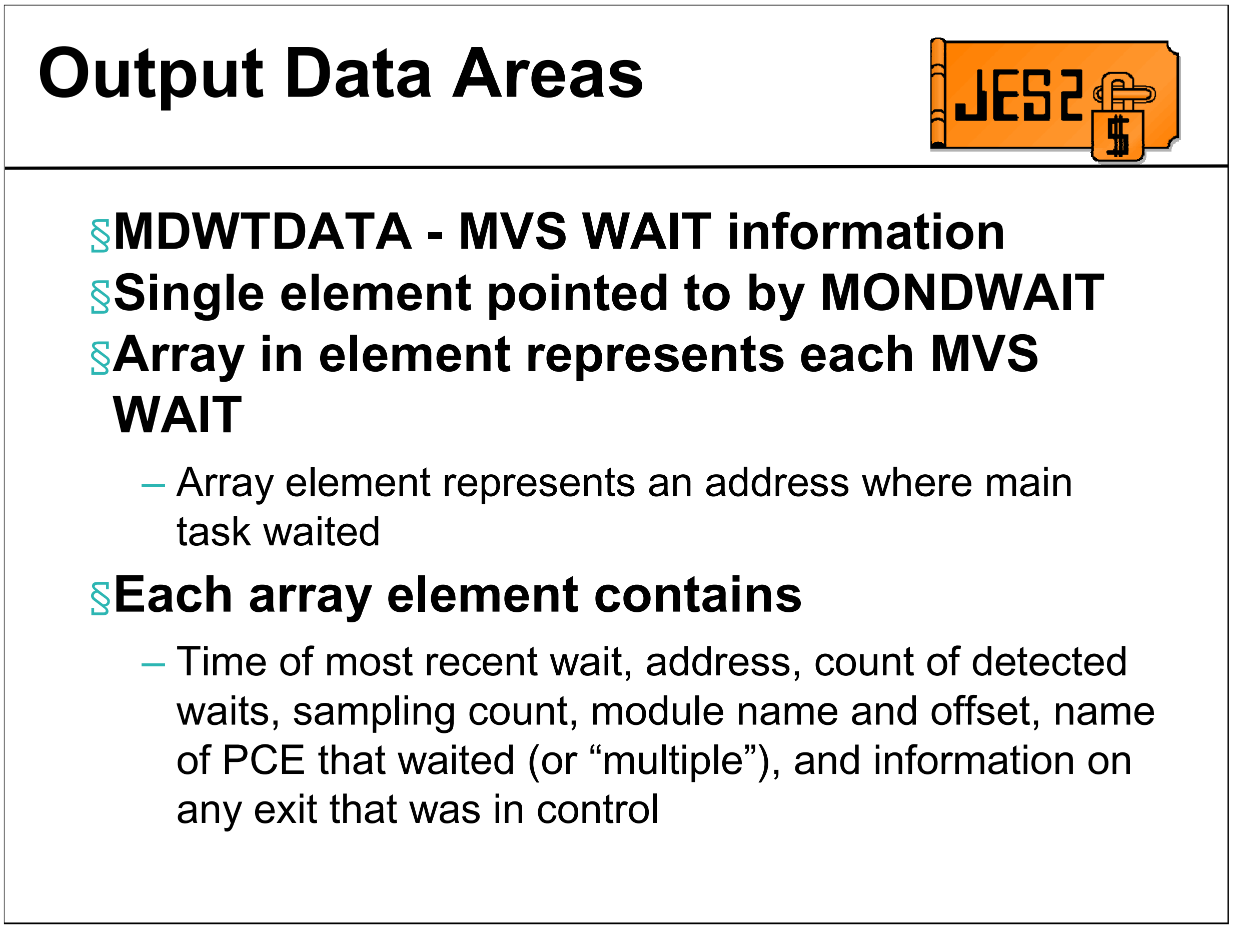
- MDCPDATA - CPU statistics
- Single element pointed to by MONDCPUS
- The element has an array based on amount of history requested
  - MONDHSTC controls size of history array
- Each history element contains
  - Time interval started, sample counts for active, idle, wait, local lock, non-dispatchable, and paging

The MDCPDATA contains the CPU statistics that the JES2 monitor accumulates. There is one entry created and pointed to by MONDCPUS. It contains an array of history elements that have the same information available in the $JDHISTORY display.

Output Data Areas

- **MDERDATA - Error counts**
- **Chain pointed to by MONDERRC**
  - Each element on chain matches a monitored error type (RECVOPTS subscript)
- **Each element has an array based on amount of history requested**
  - MONDHSTC controls size of history array
- **Each history element contains**
  - Time interval started, error type and count

The MDERDATA contains information on errors encountered in JES2. Each element on the chain represents a monitored error type (for example MAIN, DISTERR, SPOOL). Within each element is a history array for that particular type of error. The amount of history returned is based on MONDHSTC. The array elements contain the time the interval started, the error type and the error count.



Output Data Areas

- MDWTDATA - MVS WAIT information
- Single element pointed to by MONDWAIT
- Array in element represents each MVS WAIT
  - Array element represents an address where main task waited
- Each array element contains
  - Time of most recent wait, address, count of detected waits, sampling count, module name and offset, name of PCE that waited (or “multiple”), and information on any exit that was in control

The MDWTDATA contains information on detected MVS waits in the JES2 main task. There is one element pointed to by MONDWAIT. It contains an array, each element of which is a place where the main task waited. The elements contain the time of the most recent wait, the address of the wait, the count of unique times this wait was detected and the count of samples where JES2 was waiting here. If available, the module and offset of the wait is returned along with the PCE name and information on what exit was in control. There are also bits indicating if JES2 was initializing or terminating when this wait was encountered.

Output Data Areas

- **MDMSDATA - Alert/track/notice messages**
- **Chain pointed to by MONDMSGS**
  - Each element represents an ALERT, TRACK, or NOTICE message
- **Up to 4 lines of message text per entry**
- **Each entry contains**
  - Time condition started (alerts and tracks), Type of message (alert, notice or track), and up to 4 lines of text

The MDMSDATA contains information on current alerts, tracks and notices that the monitor has detected. There is no history available. Each element represents a message. There can be up to 4 lines of text per element. If this message is for a track or alert, there is also a time when the event started.

Output Data Areas

- MDMIDATA - Monitor information
- Single element pointed to by MONDMONI
- Array in element represents each monitor task
- Each array element contains
  - Task name and current status

The MDMIDATA tracks information on the status of the various tasks in the monitor address space. There is a single element with an array where each element represents a task. The task name and current status is stored in the array.

Additional Items Monitored

- **New tracks/alerts**
  - HASP9213 LONG JES2 COMMAND PROCESSING
- **New notices**
  - HASP9155 MEMBER IS NOT BOSS (WILL NOT RUN WLM INITS)
  - HASP9164 NETWORK PATH MANAGER FUNCTIONS SUSPENDED
  - HASP9165 NODE INFORMATION NOT STORED IN CHECKPOINT
  - HASP9166 LOCAL NODE NAME CHANGED SINCE XCF JOIN
  - HASP9167 WLM POLICY DIFFERENCE DETECTED (CLASSIFICATION STOPPED)
  - HASP9168 JES2 DUBBED BY OMVS BUT NOT PERMANENT PROCESS
  - HASP9169 MEMBER CANNOT GET CKPT LOCK (PROBABLY HELD BY ANOTHER MEMBER)
  - HASP9170 AT LEAST ONE SPOOL VOLUME HALTED DUE TO I/O ERRORS
  - HASP9250 RESOURCE *nnnn* UTILIZATION IS AT *xxx.xx*%

The JES2 health monitor was enhanced to monitor additional items. The long command processor message is issued if there is a single JES2 command that takes too long to run. The other items are notices that will be issued in response to the $JDSTATUS or $JDJES commands. The HASP9250 message is issued when a resource exceeds the current warning threshold. Items are added based on feedback from service as well as a result of new line items.

Questions?