#SHAREorg

SHARE
Technology · Connections · Results

z/OS 2.1 JES2 Symbol Services and Other New Services

Tom Wasik
IBM Rochester, MN

Thursday 4:30PM
Session Number 14257

SHARE in Boston

Trademarks

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

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

The following are trademarks or registered trademarks of other companies.

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

Notes:

Performance is in Internal Throughput Rate (ITR) ratio based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput improvements equivalent to the performance ratios stated here.

IBM hardware products are manufactured from new parts, or new and serviceable used parts. Regardless, our warranty terms apply.

All customer examples cited or described in this presentation are presented as illustrations of the manner in which some customers have used IBM products and the results they may have achieved. Actual environmental costs and performance characteristics will vary depending on individual customer configurations and conditions.

This publication was produced in the United States. IBM may not offer the products, services or features discussed in this document in other countries, and the information may be subject to change without notice. Consult your local IBM Business contact for information on the product or services available in your area.

All statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only.

Information about non-IBM products is obtained from the manufacturers of those products or their published announcements. IBM has not tested those products and cannot confirm the performance, compatibility, or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

Prices subject to change without notice. Contact your IBM representative or Business Partner for the most current pricing in your geography.

Overview

- This presentation will cover the following topics
  - Exporting symbols in JCL
  - JCL and JES symbol services
  - Passing symbols to internal readers
  - JES symbols set by internal readers
  - The job correlator
  - Job notification ENF
  - SAPI performance improvements
  - Job modify SSI

EXPORT JCL statement

- Previously, JCL symbols existed only during JCL conversion processing
  - Not saved as part of the output of the converter
- New EXPORT JCL statement places symbols in converter output
  - Select what symbols are to be made available
  - Signals that NEXT setting of symbol should be exported
  - Scope of any symbol is the step
- Exported symbols can be used in a number of ways:
  - Programmatically accessed using new symbol services
    - JCL service IEFSJSYM
    - JES service IAZSYMBL
  - Passed to a job being submitted through an internal reader
  - Substituted in instream data sets

EXPORT JCL statement

- EXPORT statement has the following syntax

```
//MYEXPRT EXPORT SYMLIST=(A,B,C,…)
```

  - SYMLIST is a list of symbols to export, without the “&” character
    - A,B,C in the example are JCL symbol names
      - *Same syntax rules as JCL symbol*
    - Specifying SYMLIST=* exports all JCL symbols
      - *No other use of generics is supported*
- EXPORT statement can appear anywhere after the JOB statement
  - Applies to all steps after the export statement
  - SET statements must set the symbol AFTER the EXPORT
  - Last value SET it a step is the value exported
    - A JCL symbol has only ONE value during the execution of a step
  - No limit as to the number of EXPORT statements in a job

EXPORT JCL statement - Example

```
//EX1     EXPORT SYMLIST=(S1,L1)
//SET1    SET    S1=STEWART,J1=JFK,N1=NIAGARA,L1=LAGUARDIA
//SET2    SET    S1=SANDIEGO,F1=FRESNO
//EX2     EXPORT SYMLIST=F1
//STEP1   EXEC   PGM=USERPGM1
//STEP2   EXEC   PGM=USERPGM2
//SET3    SET    S1=MSP
```

- In STEP1 the following symbols are exported (available at run time)
  S1 with value SANDIEGO
  L1 with value LAGUARDIA
  F1 with a null value (not SET after export)
- In STEP2 the following symbols are exported (available at run time)
  S1 with value MSP
  L1 with value LAGUARDIA
  F1 with a null value (not SET after export)

EXPORT JCL statement - Example

```
//EX1     EXPORT SYMLIST=(DSN,MEMB)
//SET1    SET    DSN='SYS1.SAMPLIB‘
//SET2    SET    MEMB=SAMP2
//STEP1   EXEC   PGM=USERPGM1
//INPUT   DD     DSN=&DSN(&MEMB)
//SET3    SET    MEMB=SAMP3
//OUTPUT  DD     DSN=&DSN(&MEMB)
```

- In STEP1 the following symbols are exported (available at run time)
  - DSN with value SYS1.SAMPLIB
  - MEMB=SAMP3
- Note that symbol MEMB had 2 values in the JCL for STEP1
  - SAMP2 from SET2 and SAMP3 from SET3
- EXPORT can only set one value per step
  - SAMP3 was the last value set to symbol in the JCL for the step

JCL Symbol service (IEFSJSYM)

- New service to access to JCL symbols at runtime
  - Symbols scope is a job step
    - all tasks running in the job step
  - EXPORT JCL statement specifies symbols to make available
  - Symbols are static and cannot be modified
  - Symbol names, length, and value follow JCL rules
  - Must be used after step has started running
    - Not available in exits like IEFUJI
- Also available via LE interface CEEGTJS

JCL Symbol service (IEFSJSYM)

- Usage information
  - All symbols (input and returned) do not include leading '&'
  - Output areas provided by caller
    - Specified as SYMBAREA= and SYMBAREALEN=
    - Mapped by IEFSJSYD
  - Two request types
    - REQUEST=GETALL returns all JCL symbols and values for job step
    - REQUEST=GETBYNAME returns symbol values for the symbol names passed
      - *SYMLISTARRAY= and NUMENTRIES= provides list*
      - *Generics '*' and '?' supported*

JES Symbols

- New type of symbols – JES Symbols
- JES symbols can be used:
  - To communicate between applications running in the same job step
  - To be passed as JCL symbols to jobs submitted through internal reader (SYMLIST feature)
  - To be used for substitution in instream data sets (SYMBOLS feature)
  - To communicate between application and JES internal reader (via a set of special predefined JES symbols)
- JES Symbols are very similar to JCL symbols with some exceptions:
  - Longer name – up to 16 characters
  - Longer value – up to 4096 bytes
  - Can be associated with task or step
  - Can be updated and deleted
- Symbol names starting with “SYS_” should only be set by JES

JES Symbol Service (IAZSYMBL)

- JES Symbol Service (IAZSYMBL) manages JES symbols
- Functions available are
  - CREATE – create one or more JES symbols and assign initial values
  - UPDATE – update values of selected JES Symbols
  - DELETE – delete JES Symbols and their values
  - EXTRACT – return values of selected JES Symbols
- Service supports operations over multiple JES symbols at a time
  - Including use of wildcards for EXTRACT and DELETE requests
- EXTRACT operation searches for requested symbols in this order:
  - Task level JES symbols
  - Step level JES symbols
  - Exported JCL symbols

JES Symbol service (IAZSYMBL)

- Interface to JES Symbol Service includes two macros
  - IAZSYMBL – invocation macro
  - IAZSYMDF – input/output mapping macro
    - Define input parameter list
    - Also defines various structures used by the service
- Similar in style to SSI interface
- Key data structures used by the JES Symbol Service:
  - Parameter structure
    - Filters, options, return and reason codes, feedback data etc
  - JES Symbol table structure which is used
    - for input - to create and update JES Symbols
    - for output – to return symbol information

Passing symbols to internal readers

- Symbols (JCL and JES) can now be passed on the internal reader
  - No need to update JCL to set parameters to batch jobs
  - Treated as SET commands after the JOB card
  - Available for substitution during JCL processing
    - Like SET symbols
  - Implicitly exported in the submitted job
    - EXPORT statement is not required for passed symbols
  - Parent job and submitted job can now use consistent set of symbols
- The following symbols can be passed to internal reader:
  - JCL symbols available to a parent job
  - JES symbols that conform to JCL requirements
    - Symbol name and value length
    - Character set for symbol name

Passing symbols to internal readers

- Syntax for statically allocated internal reader:

```
//SYSUT2    DD SYSOUT=(A,INTRDR),SYMLIST=(A,B,C)
```

  - SYMLIST is a list of symbols to pass, without the “&” character
    - A,B,C in the example are symbol names
    - Specifying SYMLIST=* exports all JCL compatible symbols
      - *No other use of generics is supported*
  - SYMLIST is a list of symbol names only
    - Values are extracted during job submission
    - Values of JES symbols CAN change between jobs
- Dynamically allocated internal reader has equivalent function
  - Text unit DALSYML (TU key X'802B')

Symbols on internal reader – example

- Job SYMSAMP submitting job CATALOG with INTRDR symbols

```
//SYMSAMP   JOB   MSGLEVEL=(1,1),MSGCLASS=A,NOTIFY=IBMUSER
//         EXPORT SYMLIST=(DSN,VOLSER)
//         SET    DSN=TEST.JES.LINKLIB,VOLSER=STORAG
//STEP1    EXEC   PGM=IEBGENER
//SYSPRINT DD     SYSOUT=*
//SYSUT2   DD     SYSOUT=(A,INTRDR),SYMLIST=(*)
//SYSIN    DD     DUMMY
//SYSUT1   DD     DATA,DLM='%%'
//CATALOG  JOB    1,CATALOG,MSGLEVEL=(1,1),CLASS=A
//*
//CATUSER  EXEC   PGM=IDCAMS
//SYSPRINT DD     SYSOUT=A
//SYSIN    DD     *,SYMBOLS=(JCLONLY,SYMLOG)
   DEFINE NONVSAM (NAME(&DSN) DEVT(3390) VOL(&VOLSER))                -
            CAT(PAGE08.CATALOG)
//SYMLOG   DD     SYSOUT=A
%%
```

Symbols on internal reader – example

- Job SYMSAMP as submitted (from SDSF SJ)

```
//CATALOG   JOB    1,CATALOG,MSGLEVEL=(1,1),CLASS=A
//*
//CATUSER   EXEC   PGM=IDCAMS
//SYSPRINT  DD     SYSOUT=A
//SYSIN     DD     *,SYMBOLS=(JCLONLY,SYMLOG)
   DEFINE NONVSAM (NAME(&DSN) DEVT(3390) VOL(&VOLSER))              -
             CAT(PAGE08.CATALOG)
//SYMLOG    DD     SYSOUT=A
```

- Job from JCLIN listing data set

```
//CATALOG   JOB    1,CATALOG,MSGLEVEL=(1,1),CLASS=A
// SET DSN=TEST.JES.LINKLIB                          GENERATED STATEMENT
//DSN      EXPORT EXPSET=TEST.JES.LINKLIB            GENERATED STATEMENT
// SET VOLSER=STORAG                                 GENERATED STATEMENT
//VOLSER   EXPORT EXPSET=STORAG                      GENERATED STATEMENT
//*
//CATUSER   EXEC   PGM=IDCAMS
//SYSPRINT  DD     SYSOUT=A
//SYSIN     DD     *,SYMBOLS=(JCLONLY,SYMLOG)
//SYMLOG    DD     SYSOUT=A
```

Symbols on internal reader – example

- IDCAMS SYSPRINT data set

```
IDCAMS  SYSTEM SERVICES
   DEFINE NONVSAM (NAME(TEST.JES.LINKLIB) DEVT(3390) VOL(STORAG))   -
              CAT(PAGE08.CATALOG)
IDC0001I FUNCTION COMPLETED, HIGHEST CONDITION CODE WAS 0
IDC0002I IDCAMS PROCESSING COMPLETE. MAXIMUM CONDITION CODE WAS 0
```

- SYMLOG data set

```
SYSIN    : RECORD 1 BEFORE SUBSTITUTION
SYSIN    :    DEFINE NONVSAM (NAME(&DSN) DEVT(3390) VOL(&VOLSER))           -
SYSIN    : RECORD 1 AFTER SUBSTITUTION
SYSIN    :    DEFINE NONVSAM (NAME(TEST.JES.LINKLIB) DEVT(3390) VOL(STORAG))   -
SYSIN    : RECORD 2 BEFORE SUBSTITUTION
SYSIN    :                CAT(PAGE08.CATALOG)
SYSIN    : RECORD 2 AFTER SUBSTITUTION
SYSIN    :                CAT(PAGE08.CATALOG)
```

Symbols returned from internal reader (submit)

- Internal reader now sets JES symbols for jobs processed
  - Processing always performed
  - Symbols set when a job is successfully submitted
- List of JES symbols set:
  - SYS_CORR_LASTJOB
    The 64-character job correlator of the job which was just submitted
  - SYS_LASTJOBID
    The 8-character JES job identifier of the job which was just submitted
- When job submission fails, these symbols are set to a null value
- Use JES Symbol Service (IAZSYMBL) EXTRACT function to access
  - These are task level symbols
    - Extract must be done in same task as job submission

New Job Correlator

- New identifier associated with each job (batch, STC, TSU)
  - Assigned during input processing and never changes
  - Intended to be forever unique within SYSPLEX
  - Contains user supplied data to associate or separate jobs
  - Can be use to track, manage and modify jobs
- Job correlator is a printable 64-character value
  - First 31 bytes generated by JES to ensure uniqueness
  - Followed by a ":" separator for parsing
  - Last 32 bytes is optional user data for the job (A-Z, 0-9, national, "_")
- Multiple ways to set the user portion of the correlator
  - The JES Symbol Service (IAZSYMBL) symbol SYS_CORR_USRDATA
  - UJOBCORR= keyword on the job card
  - JES2 exits 2/52 or 20/50

Obtaining the Job Correlator

- The JES Symbol Service (IAZSYMBL) can access the correlator
  - The current job' correlator is in symbol SYS_CORR_CURRJOB
  - The last submitted job correlator is in symbol SYS_CORR_LASTJOB
- MVS control blocks
  - The correlator for an address space is in the IAZJSAB (ECSA)
    - Use the IAZXJSAB macro to access it
  - The MVS JMR records has been extended to contain it
    - Passed to various MVS exits
- JES2 job display commands can display the correlator (eg $DJ)
- SSI requests
  - Extended status SSI in the JQE terse section (and in SDSF)
  - SAPI SSI in the output fields
- ENF data areas
  - 26, 30, 58, 70, and 78

Job Correlator Uses

- Input filter on SSIs
  - Extended status SSI
  - SYSOUT API (SAPI) SSI
  - Job modify SSI
- Input filter (JOBCORR=) on a number of JES2 operator commands
  - $A J, $C J, $CO J, $D J, $DO J, $E J, $H J, $L J, $O J, $P J, $PO J, $S J, $T J, $TO J
  - JC= is abbreviation for JOBCORR=
- Example of displaying job correlator

```
$djq,jobcorr
$HASP890 JOB(INIT)      JOBCORR=S0000023NODE1...C910E4AD.......:
$HASP890 JOB(IBMUSER)   JOBCORR=T0000024NODE1...C910E4BA.......:
$HASP890 JOB(TESTINS)   JOBCORR=J0000025NODE1...C910E4EC.......:TEST
$HASP890 JOB(INSACB01)  JOBCORR=J0000028NODE1...C910E980.......:PAYROLL
```

Job Correlator - Filtering with Generics

- Users of correlators support generics in the filter
  - Very useful when selecting based on user portion of correlator
  - Should NOT assume anything about contents of system portion
    - Always 31 bytes followed by a ":"
    - Will never contain a ":"
    - '.' are used as pad for short/reserved portions
- Example assuming user portion set to "function_location"
  - User portion could be
    - PAYROLL_NEWYORK or PAYROLL_BOSTON
    - ROSTER_NEWYORK or ROSTER_BOSTON
  - Filters can then select categories of jobs

    :PAYROLL_ – All PAYROLL functions

    ***:*_BOSTON** – All Boston locations

Job notification ENF

- New multi system ENF 78 to notify of job completion
  - Requested by applications specifying SYS_JOB_NOTIFY JES symbol
    - Set prior to submission with up to 4K of data
  - Issued when job moves beyond execution phase
    - Completes execution, is canceled, has JCL error, etc
- ENF contents is similar to existing ENF 70 and includes
  - The job correlator
  - Application data from SYS_JOB_NOTIFY symbol
  - Completion code (reason) for the job

SAPI performance improvements

- New tree structures to improve SAPI performance
  - Improves search for jobs for SAPI selection
  - Improves SAPI search when new output is created
- Controlled by parameters on OUTDEF
  - SAPI_OPT=YES|NO controls SAPI notification ($#POST)
  - WS_OPT=YES|NO controls SAPI selection performance ($#GET)
    - Requires all members z/OS 2.1 and $ACTIVATE LEVEL=z11
- Benefit requires selection on at least
  - QUEUE,ROUTECDE,OUTDISP
  - QUEUE,OUTDISP
  - ROUTECDE,OUTDISP
  - OUTDISP
- Consider using optimized selection criteria
  - Selecting on a single criteria such as writer will not be optimized

SAPI performance improvements

- New JOE type (index JOE) implements one of the trees
  - Consumes additional JOEs based on number of unique combinations of
    - SYSOUT class, destination, and OUTDISP
- Index JOEs are never “cleaned up” in this release
  - Every unique combination of characteristics creates an index JOE
  - Once index JOE is created, it persists until index is rebuilt
  - Index is rebuilt when it is toggled off and on by setting SAPI_OPT
  - New command $D OUTDEF,JOEUSE displays how JOEs are used

```
$doutdef,joeuse
$HASP836 OUTDEF
$HASP836 OUTDEF  CURRENT JOE UTILIZATION
$HASP836         TYPE          COUNT
$HASP836         -------- ---------
$HASP836         WORK            104
$HASP836         CHAR              1
$HASP836         INDEX             3
$HASP836         FREE             92
```

SAPI performance improvements – POST expected results

- Results from a measurement in the following scenario:
  - 50 SAPI printers selecting work by unique destination, waiting for output to become available
  - One job runs and creates one output group
  - Measurement is done in a steady state
- CPU cost of SAPI POST with SAPI_OPT=YES is 3.5 times smaller than with SAPI_OPT=NO when no device is selected (output does not match any device)
- CPU cost of SAPI POST with SAPI_OPT=YES is 1.8 times smaller than with SAPI_OPT=NO when a device is selected
- CPU cost of processing with SAPI_OPT=NO and SAPI_OPT=YES scale differently, so with larger number of devices larger performance benefit is expected

SAPI performance improvements – GET expected results

- Optimized path for active SYSOUT work selection (GET) uses index structure over JOEs
- CPU cost of the optimized path depends on many factors:
  - Number of selectable JOEs
  - A match between selection requests and the JOE index.
    - If selection request does not match index, the request goes through the old path
  - Statistical distribution of JOE attributes
    - e.g. many unique destinations in multiple classes vs many JOEs with the same class and destination
- This makes it hard to predict expected performance in all possible environments

SAPI performance improvements – GET expected results

- Results from a measurement in the following scenario:
  - 15000 JOEs selectable JOEs
  - 3000 JOEs match the selection expression
- CPU cost of selecting a JOE with WS_OPT=YES is about 17.6 times smaller than with WS_OPT=NO when a JOE is selected
- Processing with WS_OPT=NO and WS_OPT=YES scale differently, so with larger number of JOES larger performance benefit is expected

SAPI performance improvements – GET cautionary statement

- Optimized path for active SYSOUT work selection (GET) uses index structure over JOEs
- This JOE index requires maintenance, which has its own CPU overhead.
- It is unlikely but possible that in environment where majority of job SYSOUT is never processed (never selected by SAPI applications), the overhead of index maintenance will not be offset by the performance improvements of work selection. In this case, using old path (WS_OPT=NO) may be preferable.
- Note index JOEs are never deleted
  - May need to clean up JOEs by toggling WS_OPT no and yes

Job Modify SSI

- The Job Modify Service can perform the following functions:
  - Modify job characteristics
    - jobclass, priority, SYSAFF, service class, SCHENV, or offload status
  - Hold a job
  - Release a job
  - Purge a job
  - Cancel a job, with options to purge and/or dump
  - Restart a job, with cancel or step, and hold options
  - SPIN a job
  - Change a job's execution node
  - Start a job
- Each SSI call specified a single action and filters to select jobs
  - Action performed on each job matching filter based on authorization
  - A list of jobs affected is always returned to the caller
  - Filtering is the same as the extended status SSI (80)

Job Modify SSI - Authorization

- Authorization checks defined in JESJOBS class for actions
  - Ask if requestor can take action against the job
  - Resource names include
    - Action, local NJE node, user assigned to job, and jobname
    - Action is major action (like MODIFY)
      - *Does not include what is being modified*
- All security checks made in the requestor address space
  - The authorization to take the action on the job (JESJOBS check)
  - Any additional checks at the job level
    - JOBCLASS check to alter the job to a new class
    - Authorization to the new destination when jobs are re-routed

Job Modify SSI - Authorization

- Entities were patterned after the existing CANCEL entity

SSI Action	JESJOBS Class Entity	Access
Modify	MODIFY.nodename.userid.jobname	Update
Hold	HOLD.nodename.userid.jobname	Update
Release	RELEASE.nodename.userid.jobname	Update
Purge	PURGE.nodename.userid.jobname	Alter
Cancel	CANCEL.nodename.userid.jobname - Existing profile	Alter
Restart	RESTART.nodename.userid.jobname	Control
Spin	SPIN.nodename.userid.jobname	Control
Reroute execution	REROUTE.nodename.userid.jobname	Update
Start	START.nodename.userid.jobname	Control

- *Nodename* is the name of the NJE node the request is made to
- *Userid* is the user associated with the target job (not the requester)
- *Jobname* is the name of the target job (not the requester)

Job Modify SSI – Authorization Migration action

- JESJOBS class used to control access to actions and jobs
  - Review existing profile to ensure adequate protection exists
  - Beware profiles with generic high level qualifiers
    - Could create unexpected matches with new profiles
    - Profiles of '*' could cover the new requests
      - *UACC(NONE) is OK (prevents access)*
      - *Other UACC or access lists can unintentionally grant access*
    - Recommend creating profiles with the HLQ for each actions
      - *For example HOLD.** to cover hold action*
      - *Give UACC of NONE and then work to determine appropriate access*

Job Modify SSI - invocation

- Two modes of operation for request (caller selectable)
  - Asynchronous mode sends request and does not get any response
    - Success or failure of request cannot be determined
    - Returns list of jobs for which requests were made
      - *Includes only jobs that passed authorization*
    - Faster since no waiting for JES2 to respond
  - Synchronous mode waits for change to complete and returns feedback
    - All requests queued at once, then wait for all to complete
    - Returns list of jobs for which requests were made
      - *Includes only jobs that passed authorization*
      - *Returned area includes success indicator or failure reason*
    - Can be delayed if many requests/JES2 is busy
- Some request will be queued to other MAS members for processing
  - Synchronous requests wait for response from other member

Job Modify SSI - invocation

- Invoking the Job Modify Service (SSI 85)
  - Construct normal SSI SSOB and extension (IAZSSJM is extension)
    - Interface is similar to extended status (IAZSSST)
  - Request type is either an action or return storage
  - Results returned in chain of job feedback block (SSJF)
    - One per authorized job request sent to JES2
    - Can request results in 64 bit storage
  - Values returned are in feedback block :
    - Job name, job ID, original job ID, and job correlator
    - Owner userid
    - Processing member and system name
    - Job processing status indicator
      - *ASYNC – queuing success indicator*
      - *SYNC – processing success indicator*
      - *Both binary and character values returned*

System z Social Media

- System z official Twitter handle:
  - @ibm_system_z

- Top Facebook pages related to System z:
  - Systemz Mainframe
  - IBM System z on Campus
  - IBM Mainframe Professionals
  - Millennial Mainframer

- Top LinkedIn Groups related to System z:
  - Mainframe Experts Network
  - Mainframe
  - IBM Mainframe
  - System z Advocates
  - Cloud Mainframe Computing

- YouTube
  - IBM System z

- Leading Blogs related to System z:
  - Evangelizing Mainframe (Destination z blog)
  - Mainframe Performance Topics
  - Common Sense
  - Enterprise Class Innovation: System z perspectives
  - Mainframe
  - MainframeZone
  - Smarter Computing Blog
  - Millennial Mainframer

Questions?

Questions?

Session 14257