z/OS 3.2 IBM Education Assistant

Solution Name: Extend JES Policy Functions – JCL job input policy type, more actions and attributes (Stage 2)

Solution Element(s): JES2

July 2025

Agenda

Trademarks

- See url http://www.ibm.com/legal/copytrade.shtml for a list of trademarks.
- Additional Trademarks:
  - None

Objectives

- In this presentation, we will describe the extended capabilities of JES2 Policy being introduced in z/OS 3.2
- Extended capabilities include:
  - Ability to evaluate a job’s JCL at a statement level
  - Ability to write messages to a jobs JCLIN dataset
  - SSI82 enhanced to return information about JES2 policies in the MAS
  - Ability for communication between JES2 Policies and Exits via the job's JCT
  - Enhanced SendMessage/LogMessage actions with an additional “condition” attribute

Overview

- Who (Audience)
  - z/OS system administrators new to JES2
  - JES3 administrators overseeing JES3 to JES2 conversion
  - JES2 administrators who wish to phase out JES2 exit programs
  - JES2 administrators who wish to have more control over jobs entering the system
- What (Solution)
  - Enhance capabilities to customize JES2 processing in a way that does not require:
    - Low-level programming of JES2 exit programs (assembler)
- Wow (Benefit / Value, Need Addressed)
  - Reduce the need for specific JES2 skills
  - Improve JES2 reliability by isolating JES2 from bugs in JES2 exits programs

Usage & Invocation - Overview

1. New JES2 Policy Type JCLEvaluation
2. New action JCLINMessage
3. New JCL Statement related policy functions and attributes
4. Communications between policies and exits
   1. New action SetArea
   2. New functions for the JCT fields JCTUSER0 through JCTUSERF (reserved user area)
5. New policy fields added to SSI82
6. New job attributes
7. Enhancements to SendMessage/LogMessage actions
8. New miscellaneous quality-of-life functions
9. Additional policy type support for action HoldJob

Usage & Invocation – (2) JCLEvaluation

- New Policy type **JCLEvaluation**
  - Policies of this type can:
    - extract the details specified on individual JCL statements that make up a job
    - modify a selection of job attributes
    - mark a job to fail after conversion
    - write messages to a job's JCLIN dataset
  - Policies are applied early in JES2 INPUT phase when reading in a job entering the system
    - Applied to a limited selection of JCL Statements **after** the statement has been processed by JES2
      - After JES2 Exits 2/52 and 4/54 but before Exits 3/53
    - Policy is applied multiple times for the same job, once for each selective JCL Statement making up a job's JCL
  - Currently, policies are applied for ONLY these *JCL Statement Types*:
    - DD
    - EXEC
    - INCLUDE
    - JCLLIB
    - JOB
    - NOTIFY
    - OUTPUT

Usage & Invocation – (3) JCLEvaluation

- Actions supported
  - Standard policy actions
    - (e.g. SendMessage, Assign, Leave, etc.)
  - **ModifyJob** action - Modify the value for a job's attribute
    - Modifiable list of attributes is similar to *JobCreate* policy type
  - **HoldJob** action – Requests to hold job before JES2 EXECUTION
  - (new) **JCLINMessage** action
    - Write a message to the job's JCLIN dataset
- Functions supported
  - All standard policy functions (e.g., AuthorityCheck(*class*, *resource*))
  - (new) JCL Statement related functions
- Attributes (readable) supported
  - All standard policy attributes
  - Job attributes supported by *JobCreate* policy type
  - (new) JCL Statement related attributes

Usage & Invocation – (4) JCLEvaluation

- JES2 processing mentions
  - JES2 will check if there are any policies of type *JCLEvaluation* on the first JCL statement processed for a job (*JCL **JOB** Statement*)
    - If there are no policies of this type, JES2 will assume no policies exists for the job's subsequent JCL statements
  - Applying this policy type does not automatically parse and extract the details from the JCL statement (refer to new JCL Statement attributes and functions)
  - New JCL Statement attributes and functions are also supported by **JobCreate** policy type and can be used in-place of **JCLEvaluation** for *JCL JOB Statements*.
    - At this time there are only a few differences (e.g., JobCreate can change the job's name and associated JES2 resource group)

Usage & Invocation – (5) JCLEvaluation

- Policy JSON overview

```
{
        "policyName"    :   " Example ",
        "policyType"     :   " JCLEvaluation ",
        "policyVersion"  :   2,
        "variables"        : [ … ],
        "definitions"      : [
           {
               "condition" : " True  ",
               "actions"    : [
                  {
                      "action"       : " SendMessage ",
                      "message"   : " 'message' ",
                      "condition"   : " True "
                  },
                  { ... }
               ]
           },
           { ... }
        ]
}
```

JSON Object Notation

- Policy Name
- Policy Type
- Policy Version
- Policy variable declarations
- Policy Definitions
  - ➢ Definition 1
    - ○ Condition to apply actions
    - ○ Actions
      - ▪ Action 1
        - ❖ Action Name
        - ❖ Action property 1
        - ❖ Action property 2
      - ▪ (End of Action 1)
      - ▪ Action 2 ...
    - ○ (End of Actions)
  - ➢ (End of definition 1)
  - ➢ Definition 2 ...
- (End of Policy Definitions)

(End of JSON object)

Usage & Invocation – (6) JCLINMessage action

- New action **JCLINMessage**
  - Can be used to:
    - write a message to the job’s JCLIN dataset to be viewed by the submitter
    - issue a WTO containing the message
    - mark the job to fail during conversion
      - Similar to JES2 when reporting JCL errors
      - Does not prevent subsequent JCL statements being processed for the job
  - HASP1668 - new message id
    - Used as the message prefix for any messages issued
  - Supported by policy types: *JobCreate* and *JCLEvaluation*
  - Policy action serves as an interface for invoking the JES2 macro *$RMSGQUE*

Usage & Invocation – (7) JCLINMessage action

- JSON properties:
  - “**message**” – A character evaluated expression that will generate the message text

  - “**type**” – A character evaluated expression that will evaluate to 1 of the supported types

    Case insensitive and trims leading/trailing blanks

    Supported types

    - ‘*ERROR*’ – marks job to fail at conversion time, writes the message to the job’s JCLIN and issue a WTO containing the message
    - ‘*DEFER*’ – marks job to fail at conversion time, writes the message to the job’s JCLIN and if the destination is the local node, it will issue a WTO containing the message
    - ‘*WARNING*’ – writes the message to the job’s JCLIN. Does not impact how the job will be processed
    - ‘*INFORMATIONAL*’ – writes the message to the job’s JCLIN and issues a WTO containing the message. Does not impact how the job will be processed

Usage & Invocation – (8) JCLINMessage action

Example of a JCLINMessage function

- If an JCL EXEC Statement contains PGM=[PGM1 | PROGRAM2] :
  - The job is marked to fail during conversion
  - A WTO is issued with the message
  - The message is added to its JOBLOG

```
"condition" : " StmtOperation = 'EXEC' &
                        KeywordRaw('PGM') in ('PGM1', 'PROGRAM2') ",
"actions" : [
      {
            "action"     :  " JCLInMessage ",
            "message" :  " 'PGM=' || KeywordRaw('PGM') || ' -- is not allowed' ",
            "type"        :  " 'ERROR' "
      },
      …
```

Note – Assumed policy type is *JCLEvaluation*

Usage & Invocation – (9) JCL Statement

- There are 13 new policy attributes/functions related to obtaining details from a *JCL Statement* and can be used to obtain:
  - statement name (aka label)
  - statement operation (aka verb)
  - JCL positional parameter value
  - list of the JCL keyword parameter types (keyword names)
  - JCL keyword parameter value

- All are supported by policy types: **JobCreate** and **JobEvaluation**

- Policy utilizes the *statement buffer* for extracting JCL parameters
  - More on this later

Usage & Invocation – (10) JCL Statement

Attribute	**Description**	**Data Type**
StmtName	The statement's Name specified *Note – this field is also known as the JCL Statement Label as well as various aliases depending on the JCL Statement Type (e.g. JobName, ddName, Stepname)*	Character
StmtOperation	The statement's Operation specified *Note – this field is also known as the JCL Statement Type or Verb*	Character
NumStmtParms	The number of statement parameters defined *Note – Duplicate keywords are not counted*	Numeric
NumPositionals	The number of statement's positional parameters specified	Numeric
NumKeywords	The number of JCL keyword parameters defined *Note – Duplicates not counted*	Numeric

Usage & Invocation – (11) JCL Statement

Function • *parameter* **– data type**	**Description**	**Data Type**
PositionalRaw(*position*) • *position* - **numeric**	Obtain the value specified for a JCL positional parameter via a **numeric** *position* it occurs on the statement. • Numeric values for position start at 1 • If numeric value 0 or a value larger than the number of positionals on the statement, an empty string will be returned	character
KeywordRaw(*name*) • *name* - **character**	Obtain the value specified for a JCL keyword parameter via a keyword name (aka JCL parameter keyword type). • If duplicates were defined for the same keyword, the last occurrence of the keyword's value is returned • If keyword is not found, an empty string is returned	character
KeywordRaw(*alias-list*) • *alias-list* – **a character list**	Obtain the value specified for a JCL keyword parameter via a list of keyword names where each name is an alias for the same keyword. • If more then one alias is found, the last one to occur will have its value returned • Includes the same functionality of KeywordRaw (*name*)	character

Usage & Invocation – (12) JCL Statement

Attribute/Function • *parameter* **– data type**	**Description**	**Data Type**
Keywords	Obtain a list of names of all keyword parameters defined on the JCL statement. ** Does **NOT** contain duplicates*	character – list
KeywordExists(*name*) • *name* - **character**	Check if a keyword was defined on the JCL Statement identified by the keyword *name*.	logical
KeywordExists(*name-list*) • *name-list* - **a character list**	Check if **ANY** of the keywords were defined on the JCL Statement identified by a *name* in the *name-list*.	logical
KeywordHasDupl(*name*) • *name* – **character**	Check if a keyword was defined multiple times	Logical
KeywordHasDupl(*alias-list*) • *alias-list* – a **character list**	Check if **ANY** of the specified *aliases* were defined multiple times or if more then one *alias in the list* was defined on the JCL Statement.	Logical

Usage & Invocation – (13) JCL Statement

Some possible questions

- *Where does the **JCL Statement** information used by **JES2 policies** come from?*
  - **JES2 policies** parse the *statement buffer* (containing the positional and keyword parameters) as well as utilizing the *JES2 Job Receiver Work Area ($JRW)*
    - When a job is entering the system, JES2 processes 1 card image (record) at a time until it has a complete JCL statement
    - A JCL Statement contains a statement name, operation, 0 or more positional parameters, and 0 or more keyword parameters
    - During this process, positional and keyword parameters from the card images are combined into a continuous string referred to as the *statement buffer*.
    - Predating z/OS 3.2, Exits 2/52 and 4/54 are given the JCL Statement's card images and the *statement buffer* where they **<u>may modify one or both</u>**
      - If a JES2 exit modifies only the card images, the statement buffer will remain unchanged meaning JES2 policies will **NOT** see the changes

Usage & Invocation – (14) JCL Statement

- *When does JES2 Policy parse the statement buffer?*
  - A JES2 policy type (i.e., **JobCreate** or **JCLEvaluation**) will parse the *statement buffer* once for each JCL Statement making up a job and only if needed
  - The parsed data will contain any changes/additions/deletions by earlier JES2 Exits
- *Does JES2 Policy parser verify JCL syntax?*
  - No, JES2 Policy will attempt to map all positional and keyword parameters using as few syntax rules as possible and will give reasonable results for a JCL Statement with any syntax errors that may be caught be JES2 or the converter.
    1. Each JCL parameter is separated by a comma and last parameter ends with a space
    2. An equals sign "=" separates the key and value of a keyword parameter
    3. Supports JCL sub parameters (open/close parentheses)
    4. Supports single quotes (ignores text in-between)
- *Special Note*
  - JES2 policies **DO NOT** have access to the value specified for the keyword parameter *PASSWORD* of a JCL JOB Statement.

    Policy job attributes *HasPassword* and *HasPassphrase* have been provided as an alternative

Usage & Invocation – (15) JCL Statement

- Example policy use cases
  - "condition" : " not HasPassphrase "
    - Enforce passphrases
  - "condition" : " … & Match ('LIB1*', KeywordRaw(('DSNAME', 'DSN'))) "
    - React to use of a dataset specified by DD Statement DSName=
  - "condition" : " StmtOperation = 'JOB' & NumPositionals < 2 "
    - Enforce accounting and programmer name JCL positional fields
  - "condition" : " StmtOperation = 'JCLLIB' & KeywordRaw('MEMBER') = 'SYSOUT2' "
    - React to an include group

Usage & Invocation – (16) Policy and Exits

- The goal of JES2 Policy is to allow modification of JES2 processing to be more reliable, understandable, and expandable over JES2 Exits allowing installations to phase out their exits over time.
  - Problem
    - Installations may want to convert their existing exits into policies, but some functionality may not be available yet to completely convert into a JES2 Policy.
    - Multiple exits at different points in processing may work together to achieve a singular goal.
      - All these points in processing will require an associated policy type before any of the exits could be converted.
    - Installations may want to exploit features unique to JES2 Policy that would work in tandem with exits at different points in processing where there currently is no associated policy type.
- Exits have been utilizing the reserved user areas in the JCT to communicate to other exits. Policy will now be able to retrieve and modify these areas as well
- This will allow information to be passed between policies and exits to help address the above problems

Usage & Invocation – (17) SetArea action

- New action **SetArea**
  - Used to modify the value at a specified location
    - In z/OS 3.2, there is 1 location that can be modified:
      - The 64-byte *reserved user area* for a job's JCT (JES2 Job Control Table) fields (JCTUSER0 through JCTUSERF)
  - JSON properties:
    - "***attribute***" – The name of one of the supported locations
      - *JCTUSER* – The 64-byte user area of the JCT, fields JCTUSER0 through JCTUSERF
    - "***offset***" – A numerical expression representing the number of bytes beyond the beginning on the location where the *value* is intended to start
      - e.g., An offset of 0 means the beginning of the location (i.e., *JCTUSER0*). An offset of 8 indicates 8 bytes beyond the beginning of the location (i.e., *JCTUSER0+8* or *JCTUSER2*)
    - "***length***" – The numerical expression representing the number of bytes to modify.
      - The sum of offset and length cannot exceed 64 for attribute value JCTUSER
    - "***value***" – A numerical or character expression value to set at the location
      - A **numeric value** is padded left with zeros (0x00) and can have a maximum length of 8
      - A **character value** is padded right with spaces (0x40) and can have a maximum length of 64
      - Value may be truncated if longer than "length"
- Supported by all policy types where the location (i.e., JCT) is accessible

Usage & Invocation – (18) JCTUser functions

Function	Data Type
JCTUser(*offset*, *length*) • *offset* – **numeric** • *length* – **numeric**	numeric
JCTUserC(*offset*, *length*) • *offset* – **numeric** • *length* – **numeric**	character

Description
Returns the data stored in the 64-byte JCT reserved user area (beginning at the field JCTUSER0): • starting at ***offset*** (0-63) • and of ***length*** (1-64) *The sum of offset and length can not exceed 64* Returned data type and length depends on function • ***JCTUser*** *– The longest possible* ***numeric*** *value is* ***8 bytes*** • ***JCTUserC*** *– The longest possible* ***character*** *string is* ***64 bytes*** Example: 1. JCTUserC(60, 4) – Returns the last 4 bytes of the JCT reserved user area. These 4 bytes also correspond to the field JCTUSERF 2. JCTUserC((2*4), 52) – Returns 52 bytes of the JCT reserved user area. These bytes correspond to the fields JCTUSER2 through JCTUSERE 3. JCTUSER(0, 4) – Returns the 1st 4 bytes of the JCT reserved user area interpreted as a single numeric value. This also corresponds to JCTUSER0

Usage & Invocation – (19) Message actions

Example of a JCTUserC function

- Policy will **display** 8 bytes of text starting 14 bytes from the beginning of the JCT reserved user area

```
{
    "action“      : “SendMessage“,
    "message“ : “ JCTUserC(14, 8) "
}
```

Usage & Invocation – (20) Message actions

Example of a SetArea action

- Policy will **modify** 8 bytes of the JCT reserved user area starting 14 bytes from the beginning
- value will be padded right with blanks 0x40 (“Policy “)

```
{
    “action“     : “ SetArea “,
    “attribute“ : “JCTUser “,
    “offset“      : “ 14 “,
    “length“     : “ 8 “,
    “value“      : “ ‘Policy’ “,
}
```

Usage & Invocation – (21) SSI 82

- SSI82 has been enhanced to include information about the policies existing in the MAS (JESPLEX)

- SSI 82 (JES properties) - new JES2 Policy subfunction was added to return information on each policy imported:
  - Policy Name
  - Policy Type – can be filtered on (e.g., *JobCreate*, *JobConversion*, etc.)
  - Policy Version
  - Policy Path – 98 characters (*current maximum length*)
    - The dataset/directory and member/file name of the JSON document used to create the JES2 policy at the time it was imported
    - Field will **NOT** be populated for policies imported prior to z/OS 3.2 and policies of version 1

Usage & Invocation – (22) Job attributes

Attribute	**Description** • *Available to policy types:* ***JobCreate***, ***JCLEvaluation***, *and* ***JobInput***	**Data Type**
HasPassword	Indicates if a *password* or *passphrase* was specified for the job	Logical
HasPassphrase	Indicates if a *passphrase* was specified for the job	Logical
Attribute	**Description** • *Available to policy type:* ***PreConversion***	**Data Type**
DfltSteptime	Specifies the default step time for a job to be passed to the converter as a numeric list (*mm*, *ss*) • *mm* – number of minutes (max 357912) • *ss* – number of seconds (will be converted to minutes if > 59) • Overrides the *STEPTIME* defined for the job's assigned *JOBCLASS* • Attribute is **modifiable** • Via *Preconversion* action **SetDefaults**	Numeric - list

Usage & Invocation – (23) Message actions

- SendMessage and Logmessage actions have been enhanced with an additional JSON property:
  - “condition” – An **optional** logical evaluated expression that indicates if the message would be processed or skipped
    - TRUE – Default value if omitted, indicates to process this action and issue the appropriate message
    - FALSE – Indicates the action should **not** be processed and message will be skipped

- This will allow messages from JES2 Policy to be disabled/enabled for different specified conditions.
  - Some example use cases:
    - Production vs. Test systems
    - External values like system symbols
    - Policy debugging mode is enabled
    - Which JES2 member executed the policy

Usage & Invocation – (24) Message actions

Example of a LogMessage action with new *condition* property

- Policy will issue the message if policy debug mode is on

```
{
    "action"       : " LogMessage “,
    "message“  : “ ‘Policy Debug mode is ON‘ “,
    “condition" : " DEBUGMode "
}
```

Note *– Policy debug mode can be enabled with JES2 Command $TDEBUG POLICY=[ON | OFF]*

Usage & Invocation – (25) Message actions

Example of a SendMessage action with new *condition* property

- Policy will issue the message if the JES2 member name executing the current policy is in a list of allowed JES2 members.

```
{
    "action"       : " SendMessage “,
    "message“   : “ ‘Policy executed on an allowed member‘ “,
    “prefix"        : " LocalNodeName “,
    “condition" : " LocalMemberName in $JES2MsgAllow "
}
```

Note - The “$” on *$JES2MsgAllow* indicates a policy variable. It was defined and initialized prior to this action by this or a previous policy

Usage & Invocation – (26) MISC Functions

Function	Description	Data Type
ValidJobName(string) • *string* - **character**	Verify if the given *string* is syntaxial correct as a valid JCL JOB Statement Name	logical
Lowercase(string) • *string* - **character**	Returns an equivalent string with all characters lowercased	character
Uppercase(*string*) • *string* - **character**	Returns an equivalent string with all characters uppercased	character

Usage & Invocation – (27) HoldJob action

- The HoldJob action is supported in the following JES2 policy types:
  - **JobCreate** (new)
  - **JCLEvaluation** (new)
  - JobInput
  - JobConversion

- When applied, the HoldJob action:
  - Requests the system to hold the job after input and conversion processing, but before execution
  - Prevents the job from executing until it is explicitly released by an operator

- Important Notes
  - JES2 will not hold the job if:
    - An error occurs during input processing
    - The job's destination node differs from the node that applied the HoldJob action.

Interactions & Dependencies

- Software Dependencies
  - None
- Hardware Dependencies
  - None
- Exploiters
  - Any installation that has a need to customize JES2 processing.

Upgrade & Coexistence Considerations

- To exploit this solution, all systems in the Plex must be at the new z/OS level: No
  - JES2 compatibility APAR OA65446 is required to tolerate some new functionality.
- List any toleration/coexistence APARs/PTFs.
  - JES2 compatibility APAR OA65446
- List anything that doesn't work the same anymore.
  - None
- Compatibility APAR OA65446 is also recommended for fallback to z/OS 3.1 if policies exploiting new features have been created by z/OS 3.2 and remain in the JES2 checkpoint.

Installation & Configuration

- No special installation is required.
- Planning considerations for using JES2 policies are documented in JES2 Installation Exits publication.

Summary

- In this presentation we described z/OS 3.2 enhancements to the JES2 policy function.

Appendix

- Publications
  - z/OS 3.2 JES2 Commands
  - z/OS 3.2 JES2 Initialization and Tuning Guide
  - z/OS 3.2 JES2 Initialization and Tuning Reference
  - z/OS 3.2 JES2 Installation Exits
  - z/OS 3.2 JES2 Messages
  - z/OS 3.2 MVS Using the Subsystem Interface