IBM Education Assistant

JES2 assembler exit config & options remediation

Agenda

- Trademarks
- Session Objectives
- Overview
- Usage & Invocation
- Interactions & Dependencies
- Migration & Coexistence Considerations
- Installation
- Appendix

Trademarks

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

Session Objectives

- In this presentation we will introduce new way to customize JES2 processing - JES2 policies.

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
- What (Solution)
  - Provide a capability to customize JES2 processing in a way that does not require:
    - low-level programming of JES2 exit programs (Assembler)
    - knowledge of the details of JES2 internal control structures
    - understanding of the JES2 internal processing logic
- Wow (Benefit / Value, Need Addressed)
  - Reduce the need for specific JES2 skills
  - Improve JES2 reliability by isolating JES2 from bugs in JES2 exit programs

Usage & Invocation - overview

- At a very high level, JES2 policy defines in user-level terms what JES2 must do in certain strategic points in JES2 processing.
- Externally, a JES2 policy definition is a JSON object residing in a human-readable editable z/OS data set.
- After having been created by a system administrator, a JES2 policy definition is imported into JES2 by a new JES2 command and the policy becomes available for JES2 processing.
- New JES2 commands are provided to manage JES2 policies – import, delete, enable/disable.
- Note that JES2 policies do not replace JES2 exits – the exits are still supported.

Usage & Invocation – policy types

- There could be many types of JES2 policies, that differ in what JES2 function they impact and what JES2 objects they can reference and/or affect.
- Some policy types will be shipped with z/OS 2.4. More policy types will be supported in the future as JES2 policy function evolves.
- A basic building block for most policy types is a condition and a set of actions that must be applied when the condition is met.
- JES2 will consider all policies of a particular type at appropriate point of processing, as long as policy conditions are satisfied. Administrator can freely choose between multiple simple policies (few conditions and actions defined) or fewer complex polices (with many conditions/actions).

Usage & Invocation – policy syntax

- Policy type determines the specific syntax used in the policy definition.
- The external definition of a policy is a JSON document and therefore must conform to JSON syntax rules.
- All policy types have common standard definition sections and other syntax rules that are common for all policy types.
- In addition, each policy type supports having sections that are unique for the type.

Usage & Invocation – policy example (1)

- An example on the next slide defines a simple policy with the name JCONV1 of type JobConversion, that will be applied at the end of job conversion phase.
- This policy has one definition (a set of a condition and associated actions):
  - if job has affinity to JES2 member N1M2
    - add N1M1 to the job's affinity and
    - issue a message to operator which displays the resulting affinity list of the job

Usage & Invocation – policy example (2)

```
{  "policyName":     "JCONV1",
   "policyVersion":  1,
   "policyType":     " JobConversion ",
   "definitions":
      [
        { "condition" : " JobHasAffinity('N1M2')  ",
           "actions"  :
              [
                 {  "action"    : " modifyJob ",
                    "attribute" : " SYSAFF ",
                    "value"     : " listAdd(sysaff, 'N1M1') "
                 },
                 {  "action"  : " SendMessage ",
                    "message" : " 'New job affinity is ' || string(SYSAFF) "
                 }
              ]
        }
      ]
}
```

Usage & Invocation – policy commands (1)

```
$ADD POLICYLIB(policylib),DDn= { DSNAME=dsname }
                              { PATH=dirname  }
```

- This command creates a new logical concatenation of PDSs, PDSEs and z/OS UNIX directories used as a location for importing policy definitions.
- The syntax is the same as for $ADD PROCLIB command with some exceptions (in particular, no static concatenations are supported).
- Member of a concatenation can be a PDS/PDSE data set (DSNAME=) or a directory in a zFS file system (PATH=).
- If PATH= is coded:
  - Code the name of the directory, not a file
  - The files names in the directory must match member name rules

Usage & Invocation – policy commands (2)

```
$POLICY IMPORT,POLICYLIB(policylib),MEMBER=mbrname
          [,ENABLE={YES | NO } ]
          [,REPLACE={NO | YES } ]
```

- This command imports policy definition from a data set into JES2 MAS.
- POLICYLIB= and MEMBER= point to the policy definition in external format – a member in a POLICYLIB concatenation previously created via $ADD POLICYLIB command.
- ENABLE= keyword specifies whether policy is initially created in an enabled state. Enabled policy immediately becomes effective.
- REPLACE= keyword specifies whether new policy is allowed to replace an existing policy with the same name.

Usage & Invocation – policy commands (3)

```
$POLICY DELETE,NAME=policy-name
               [,TYPE=policy-type ]
```

- This command deletes one or more policies from JES2 MAS.
- Policy name parameter (NAME=) can include wildcard characters.
- Optional TYPE= keyword can be used to further qualify the set of policies to be deleted.

Usage & Invocation – policy commands (4)

```
$POLICY DISABLE,NAME=policy-name
                [,TYPE=policy-type ]
```

- This command changes one or more policies to a disabled state.
- Disabled policies still exist in JES2 MAS, but are not used.
- Policy name parameter (NAME=) can include wildcard characters.
- Optional TYPE= keyword can be used to further qualify the set of policies to be disabled.

Usage & Invocation – policy commands (5)

```
$POLICY ENABLE,NAME=policy-name
                [,TYPE=policy-type ]
```

- This command changes one or more policies to an enabled state.
- Policies may be originally imported as disabled or may be changed to disable state by $POLICY DISABLE command. A policy can also become disabled when JES2 encounters an error when applying the policy.
- Policy name parameter can include wildcard characters.
- Optional TYPE= keyword can be used to further qualify the set of policies to be enabled.

Usage & Invocation – future

- The main deliverable in V2R4 is the policy engine and a limited set of policy types with limited capabilities.
- In the future, JES2 will provide more policy types and more capabilities based on the feedback from sponsor users.
- Depending on the effort required and the feedback, mid-release delivery of enhancements is a possibility.

Interactions & Dependencies

- To exploit this item, all systems in the Plex must be at the new z/OS level: No

- Software Dependencies
  - None.
- Hardware Dependencies
  - None.
- Exploiters
  - None.

Migration & Coexistence Considerations

- No specific migration actions are required.
- However, when defining policies covering the same areas as JES2 exits, care must be taken to avoid contradictions between policies and exits.
- Down-level JES2 members will not use policies created by JES2 members at V2R4 level.

Installation

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

Appendix

- z/OS V2R4.0 JES2 Installation Exits – SA32-0995-40
- z/OS V2R4.0 JES2 Commands – SA32-0990-40
- z/OS V2R4.0 JES2 Messages – SA32-0989-40