Dwindling Assembler Skills
Some Tips to Resolve the Problems

Mike Shorkend

mike@shorkend.com

About me

- *Independent Consultant*
- *35+ years of Mainframe Infrastructure Experience*
- *Experience includes z/OS system programming,capacity planning and control, performance tuning, software pricing, architecture, BCP/DRP, software development and support, management training, mentoring.*
- *Customers include financial institutes, government agencies, VARs, ISVs and training companies*
- *SHARE volunteer since 2018*

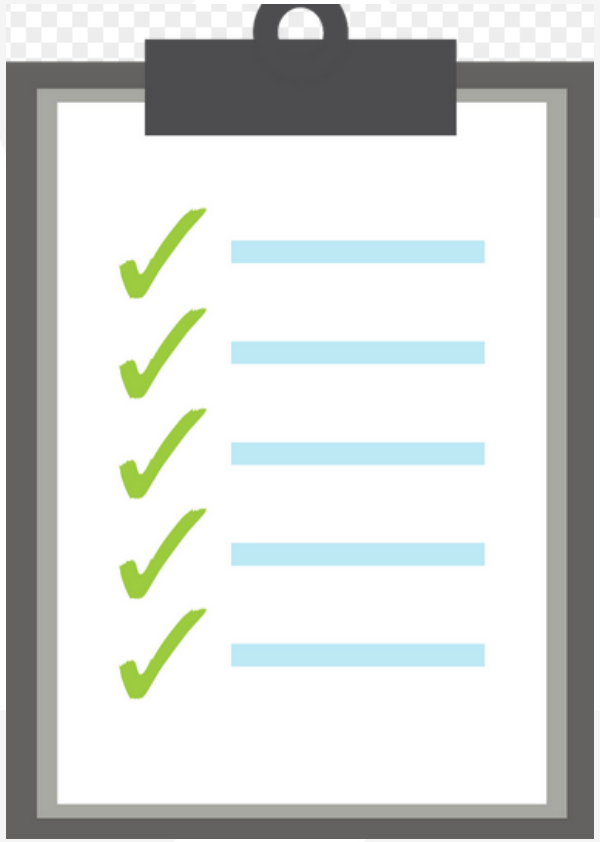
Agenda

- Introduction
- JES2 policies
- SMFLIMxx PARMLIB member
- IRRPRMxx PARMLIB member
- RACF dynamic class descriptor table
- CEEOPTS

Introduction

- Assembler programmers are an endangered species
- Many customers have assembler program objects, particularly for system customization
- Over many z/OS releases, IBM has provided many ways to eliminate the need for assembler objects
- Current z/OS versions provide several opportunities to replace assembler code with millennial-friendly alternatives

JES2 Policy



z/OS
2.4

JES2 exits

- Go back a long time
- z/OS 2.5 has 60 possible JES2 exits
- These installation exits can control many aspects of JES2 behaviour
- Examples: Print separators, naming conventions, security, accounting, customizing/adding messages
- JES2 exits are written in assembler but they use their own version of macros and control blocks

JSON

- JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate. Source: json.org
- JSON is the building block for JES2 Policies

JES2 policies

- Provide a way of using JSON to replace JES2 exits
- There are several types of policies to provide for custom logic at different phases of JES2 execution(just like the exits)
- z/OS 2.4 provided just one type
- Latest 3.1 provides five

JES2 Policy types (1/2)

Type	Function	Equivalent exit(s)
PreConversion	Filtering on job default attributes (such as Completion Code, Job Class, and Service Class) and also modification of job default attributes (such as JESLOG and Message Class) prior to entering the job conversion phase.	2/52,3/53
JobConversion	Filtering on job card values (such as Job Class, Programmer's name, and Accounting information) and also modification of job attributes (such as Job Class and scheduling environment) prior to entering the job input phase.	3/53
JobCreate	Filtering on basic job attributes, after exit 2/52. Some attributes can be modified.	

JES2 Policy types (2/2)

Type	Function	Equivalent exit(s)
JobInput	Analysis and customization of key job attributes as part of job input processing	20/50
SYSOUTGroup	filtering on certain SYSOUT data set attributes (such as DD Name and Job Class) and also modification of certain SYSOUT data set attributes (such as Class and Destination)	40

What does a policy look like?

```
{
 "policyName": "JCONV1",
 "policyVersion": 1,
 "policyType": "JobConversion",
 "definitions":
  [
   {
      "condition" : " jobprty < 5 ",
      "actions" :
        [
          {
              "action" : "HoldJob"
          },
          {
              "action" : "modifyJob",
              "attribute" : "JOBPRTY",
              "value" : " jobprty + 1 "
          },
          {
             "action" : "SendMessage",
             "message" : " 'Job ' || JOBNAME || ' was held' "
          }
        ]
  },
   ]
}
```

If JOB PRTY < 5
Then
Hold the job;
Bump up Priority by 1;
Send a message to the console

How does it work?

1. Create a policy member in a PDS, e.g. SYS3.POLICY(MYPOLICY)
2. Define the policy library to JES2 using the following command:

```
$ADD POLICYLIB(PLCY01),DD1=DSN=SYS3.POLICY
```

3. Import the policy

```
$POLICY IMPORT,POLICYLIB=PLCY01,MEMBER=MYPOLICY
```

4. Voila!

What can I ask ?

Depending on the policy type, you can query different attributes. Examples:

- *PreConversion – what is the job class?*
- *JobConversion – who is the job owner?*
- *SYSOUTGroup – what is the output class?*

JES2 Attributes for JobConversion (5 of 14)

Table 16. JES2 attributes supported for policy type JobConversion

Attribute name	Data type	Description	Modifiable
JobAcct	Character list	List of accounting entries specified on the JOB JCL statement	No
JobClass	Character	Job class	Yes
JobHasAffinity(member-list)	Logical	Tests if job has affinity to one of the members provided in the member-list, for example, JobHasAffinity(('N1M1','N2M2')) returns **True** if job has affinity to MAS member 'N1M1' or MAS member 'N1M2' or both.	No
JobIsHeld	Logical	Tests if job is held	No
JobIsDupl_Exempt	Logical	Tests if job is exempt from processing for duplicate jobs (jobs with the same name)	Yes

How can I ask ? Some definitions

- *The JSON supported by JES2 policies has the basic characteristics you would find in any language*
- *Data-types, literals, operators, functions etc.*
- *Example*

```
Substr(JobName,1,3) = JobSubmitter
```

Will return TRUE if the submitter's userid matches the first three characters on the job name

- *Full rules are here*

What can I do ?

After you have asked about a job attribute you can perform an actions.

- For all Policies: Write a message to SYSLOG or console ,modify an attribute
- For JobConversion: CancelJob, HoldJob
- For PreConversion – SetDefaults
- For SYSOUTGroup - DeleteDataset

What can I Modify ?

Depends on the policy type. Some examples

For PreConversion – Allow system symbols, REGION

For JobConversion: Job Class, System Affinity, Scheduling Environment, Service class

For SYSOUTGroup: Output Class, Destination, Disposition

User case

Exit 3/53 - JOB/JOBGROUP statement accounting field scan

1. Inspects the accounting information and sets the job class depending on the accounting information

2. Two classes are exempt (F and Z, sysprog privilege)

3. The first accounting field is assumed to be Tnn where nn is a decimal number

```
//MIKEJOB1 JOB (T00),'MIKE',MSGCLASS=X,NOTIFY=&SYSUID
```

4. Don't ask why this is done (older than the hills)

Exit Logic

If CLASS=F or CLASS=Z then exit

Set default CLASS=L

acctstring= whatever comes after the T on the accounting information

If acctstring is not numeric, then exit

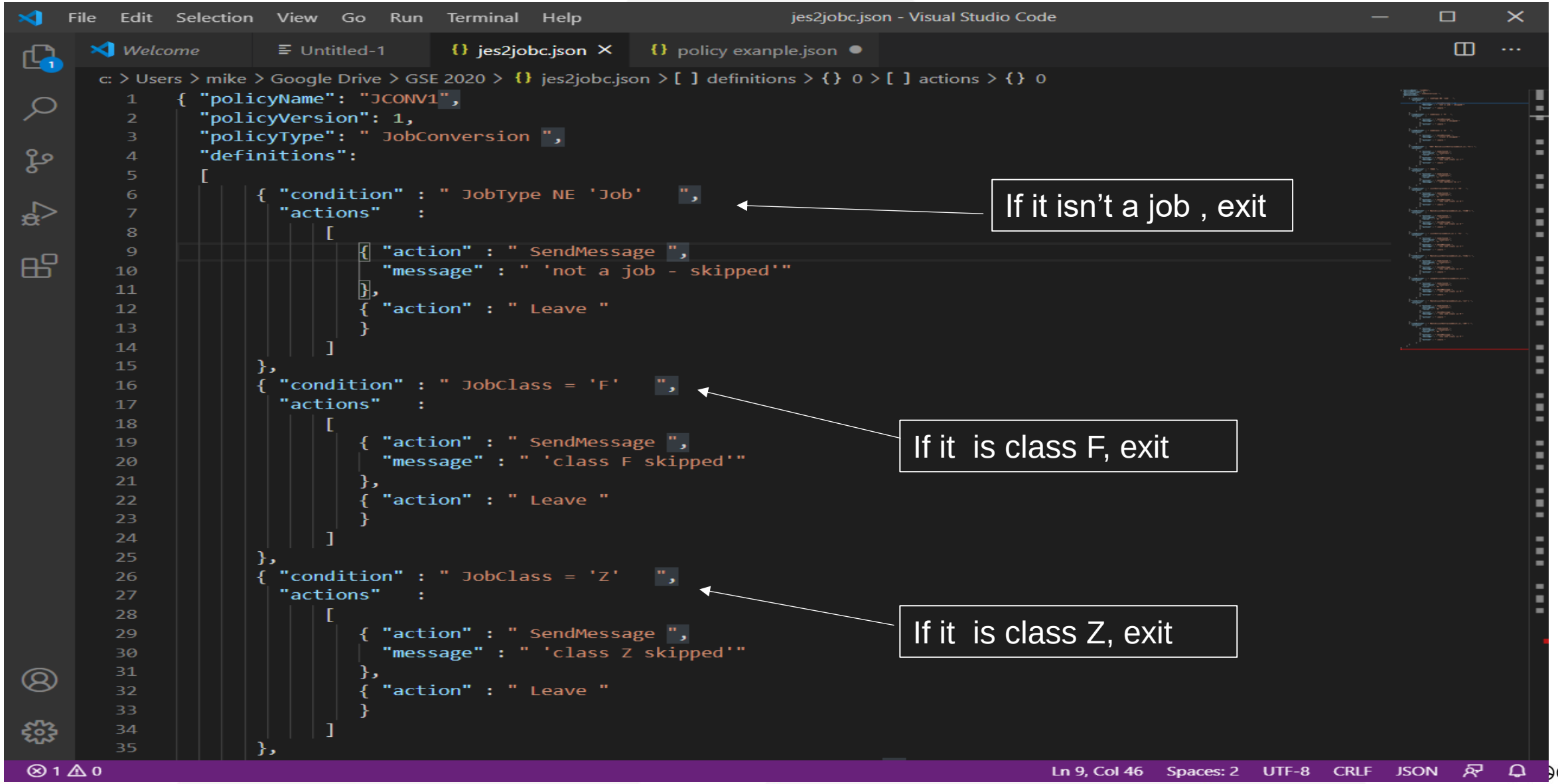
If acctstring = 0 then set CLASS=Q and exit

If acctstring = 1 then set CLASS=S and exit

If 1<acctstring<20 then set CLASS=M and exit

Anything else, exit

Some sample code 1/2



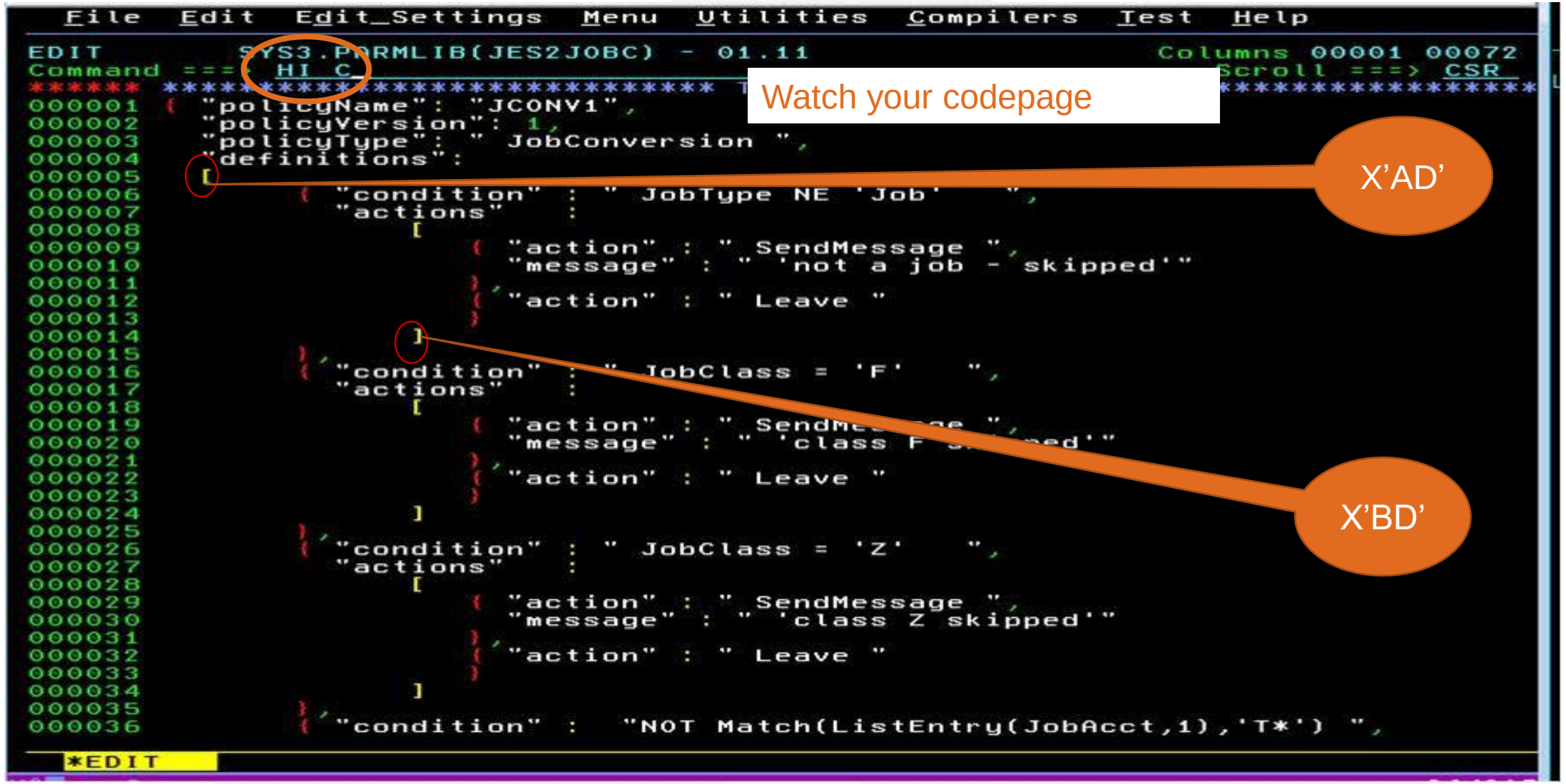
Some sample code 2/2

```
{ "condition" :  "NOT Match(ListEntry(JobAcct,1),'T*') ",
        "actions"    :
            [
              { "action" : " modifyJob ",
                "attribute" : "JobClass",
                "value" : " 'L' "
              },
              { "action" : " SendMessage ",
                "message" : " 'new job class is L'"
              },
              { "action" : " Leave "
              }
            ]
      },
      { "condition" : "FindNumber(ListEntry(JobAcct,1)) = 0",
        "actions"    :
            [
              { "action" : " modifyJob ",
                "attribute" : "JobClass",
                "value" : " 'Q' "
              },
              { "action" : " SendMessage ",
                "message" : " 'new job class is Q'"
              },
              { "action" : " Leave "
              }
            ]
```

If the accounting information Does not start with a T, Assign CLASS=L and exit

If the number after the T is 0 Assign CLASS=Q and exit

Step 1 – code the policy



Step 2 – Add Policy Library

```
$ADD POLICYLIB(PLCY01),DD1=DSN=xxxx.PARMLIB
IEF196I IEF237I 1C10 ALLOCATED TO $PL00003
$HASP737 POLICYLIB(PLCY01) 877
$HASP737 POLICYLIB(PLCY01)
$HASP737                DD(1)=(DSNAME=xxxx.PARMLIB,
$HASP737                VOLSER=vvvvvv)
```

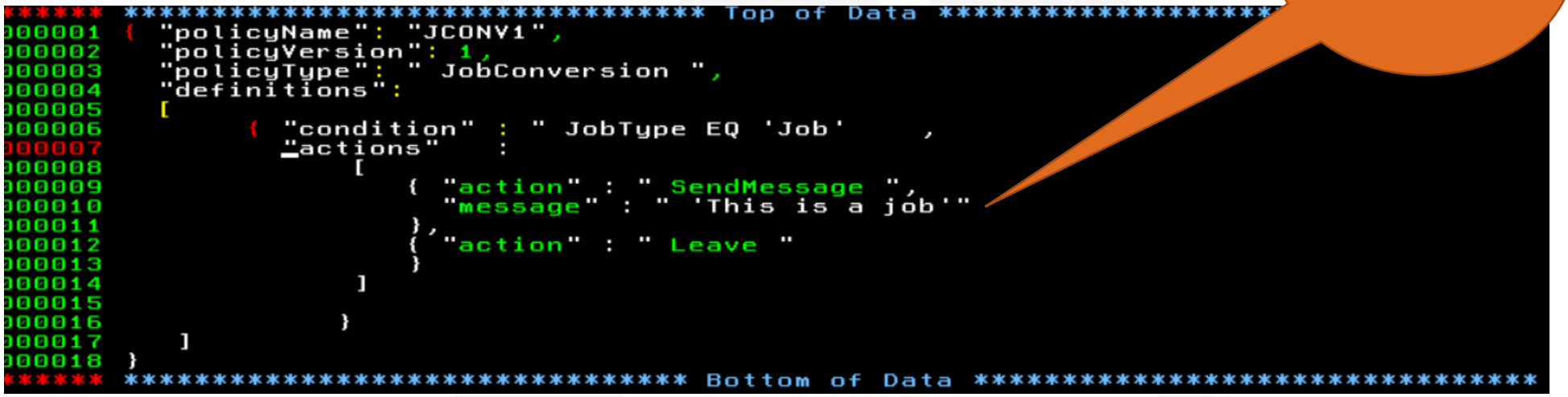
```
$DPOLICYLIB
$HASP737 POLICYLIB(PLCY01)
$HASP737 POLICYLIB(PLCY01)
$HASP737                DD(1)=(DSNAME=xxxx.PARMLIB,
$HASP737                VOLSER=vvvvvv)
```

Step 3 – Import the policy

```
$POLICY IMPORT,POLICYLIB=PLCY01,MEMBER=JES2JOBC,REPLACE=YES
$HASP1600 POLICY IMPORT request accepted.
$HASP1603 Validation of policy JCONV1 type JobConversion is complete.
$HASP1611 Policy JCONV1 type JobConversion saved in the JES2 813
          checkpoint.
$HASP1614 Policy JCONV1 type JobConversion added to runtime 814
          repository.
$HASP1601 IMPORT policy JCONV1 request complete.
```

Step 3a – Syntax error

Syntax errors look like this:



Missing “

```
$POLICY IMPORT,POLICYLIB=PLCY01,MEMBER=JES2BUG,REPLACE=YES
$HASP1600 POLICY IMPORT request accepted.
$HASP1630 JSON parser reported error 00000109, reason code 103. 768
          Expected comma between pairs in object at offset 490.
$HASP1631 Location of error:. 769
        "                    "actions"   :              "
        ........................*.........................
$HASP1602 IMPORT request for policy *UNKNOWN failed.
```

Translates to Line 7, column 10(LRECL=80)

What does it look like?

```
12:08:36.71 T0008852 00000090  $HASP100 MIKE      ON TSOINRDR
 12:08:36.77 T0008852 00000090  not a job - skipped
 12:08:36.82 T0008852 00000090  $HASP373 MIKE      STARTED
```

```
//MIKEBR14  JOB (T00),'MIKE',MSGCLASS=X,NOTIFY=&SYSUID,CLASS=A
```

```
 12:10:14.88 J0008853 00000090  $HASP100 MIKEBR14 ON INTRDR      MIKE  FROM T0008852 MIKE
 12:10:14.92 J0008853 00000290  IRR010I  USERID MIKE      IS ASSIGNED TO THIS JOB.
 12:10:15.05 J0008853 00000090  set default to L
 12:10:15.05 J0008853 00000090  new job class is Q
 12:11:05.54 MIKE3B11 00000290  $DJ8853
  12:11:06.55 J0008853 00000090  $HASP890 JOB(MIKEBR14) 773
            773 00000090  $HASP890 JOB(MIKEBR14)  STATUS=(AWAITING EXECUTION),CLASS=Q,
            773 00000090  $HASP890              PRIORITY=9,SYSAFF=(ANY),HOLD=(NONE)
```

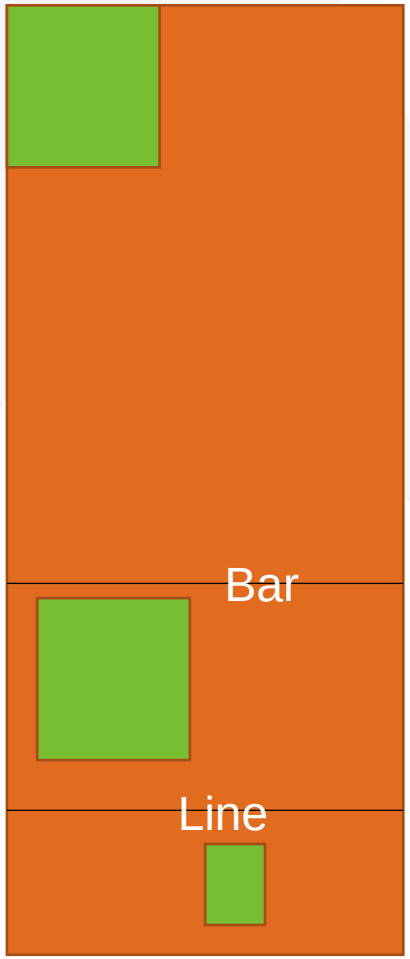
Some useful commands

```
$ADD POLICYLIB(PLCY01),DD1=DSN=SYS3.POLICY
$DPOLICYLIB
$POLICY IMPORT,POLICYLIB=PLCY01,MEMBER=MYPOLICY[,REPLACE=YES]
$POLICY DISPLAY,NAME=JCONV1
$POLICY DISABLE,NAME=JCONV1
$POLICY ENABLE,NAME=JCONV1
```

Summary

- *Easy to setup and use*
- *JSON is pretty straight forward*
- *Convert your exits while you still have some assembler skills available*
- *JSON can not do everything that assembler can*
- *JES Policies survive a warm start*
- *There are no init statements, all done via commands*
- *Keep current with your maintenance(always a good idea)*

SMFLIMxx



z/OS
2.2

The old way for virtual storage management

- *Many sites have coded IEFUSI exits for various reasons:*
  - *Adding virtual storage to a job step to avoid S878 and S80A abends*
  - *Restrict the excessive use of virtual storage by a rogue program*
  - *Provide reasonable defaults*
- *Requires quite a bit of assembler programming, can get quite fancy*
- *Needs to be revised/updated for hardware and software changes*

The new way

- *Introduced in z/OS 2.2, enhanced for 2.4(security controls, improved defaults).*
- *Specify SMFLIM=xx in IEASYSxx*
- *Use filters to have jobs qualify for processing (if this is Mike's Job)*
  - *Some common ones: JOBACCT,JOBNAME,JOBCLASS,SUBSYS*
- *Use attributes to perform actions (give Mike's job a lorry-load of storage)*
  - *Some common ones: REGIONABOVE,REGIONBELOW,EXECUTE*
- *Full details here*
- *Ed Jaffe explains SMFLIM here*

The new way

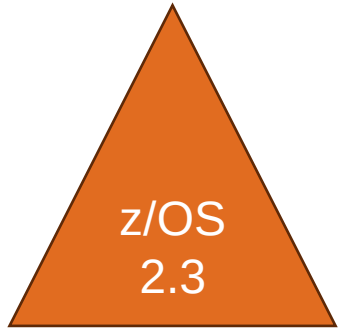
```
REGION REGIONBELOW(10M)
       REGIONABOVE(256M)
       MEMLIMIT(128G)

REGION JOBNAME(MIKE*)
       REGIONBELOW(NOLIMIT)
       REGIONABOVE(NOLIMIT)
       MEMLIMIT(NOLIMIT)
```

Set a default

Special cases

RACF

IRRPRMxx



The old way

- *For years we have been specifying the name of the RACF databases and other RACF options in a load module named* ICHRDSNT.
- *You had to build a load module for each RACF environment and every time you upgraded z/OS.*
- *Probably wrapped in a USERMOD.*

The old way

```
  PUNCH '++USERMOD (RACFDB).'
  PUNCH '++ VER (Z038) FMID(HRF77A0).'
  PUNCH '++ MOD (ICHRDSNT) DISTLIB(AOSBN) LMOD(ICHRDSNT).'
ICHRDSNT CSECT ,
ICHRDSNT AMODE 31
ICHRDSNT RMODE 24
          DC   AL1(1)                  NUMBER OF ENTRIES     ONE
          DC   CL44'SYS1.RACF.RACFDB.PRIMARY'
          DC   CL44'SYS1.RACF.RACFDB.BACKUP'
          DC   AL1(255)                NUMBER OF RESIDENT BLOCKS
          DC   B'10000000'             UPDATES DUPLICATED ON BACK-UP DS
*                                      USE RESIDENT DATA BLOCKS
*       BIT 0 - DUPLICATE UPDATES TO BACK-UP                          *
*       BIT 1 - DUPLICATE STATISTICAL UPDATES TO BACK-UP              *
*       BIT 2 - RESERVED                                              *
*       BIT 3 - RESERVED                                              *
*       BIT 4 - ENABLE FOR SYSPLEX COMMUNICATIONS                     *
*       BIT 5 - DEFAULT MODE                                          *
*                0 - NON DATA SHARING                                 *
*                1 - DATA SHARING                                     *
*       BIT 6 - RESERVED                                              *
*       BIT 7 - RESERVED
          END
```

The new way

- *Specify the database names and RACF options in a dedicated PARMLIB member -* IRRPRMxx
- Specify RACF=(xx) in IEASYSxx.
- There is a sample(with loads of comments) in SYS1.SAMPLIB

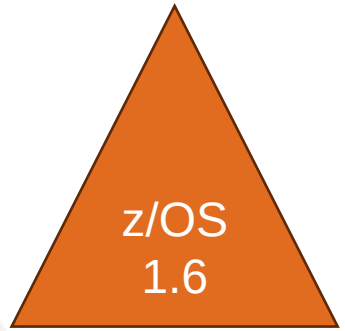
The new way

```
DATABASE_OPTIONS
  SYSPLEX(NOCOMMUNICATIONS)
  DATASETNAMETABLE
    /* ENTRY 1 - FIRST DATA SET PAIR                    */
    ENTRY
      PRIMARYDSN(SYS1.RACF.RACFDB.PRIMARY)
      BACKUPDSN(SYS1.RACF.RACFDB.BACKUP)
      UPDATEBACKUP(NOSTATS)
      BUFFERS(255)
```

RACF

SHARE
EDUCATE • NETWORK • INFLUENCE

Dynamic CDT



The old way

- *Historically, the method to add your own RACF classes was by using a macro called* ICHERCDE *that you assemble and bind into a load module called* ICHRRCDE
- *Re-assemble needed for new z/OS versions*
- *Probably wrapped in a USERMOD*
- *Requires an IPL to pick up a new version*

The new way

- Use the RACF RDEFINE command to add a new class for example:

```
RDEFINE CDT TSTCLAS8 UACC(NONE)
            CDTINFO(DEFAULTUACC(NONE) FIRST(ALPHA) MAXLENGTH(42)
            OTHER(ALPHA,NUMERIC,SPECIAL) POSIT(303)      RACLIST(REQUIRED)
            SECLABELSREQUIRED(YES))
```

- Issue the SETROPTS command to refresh the CDT

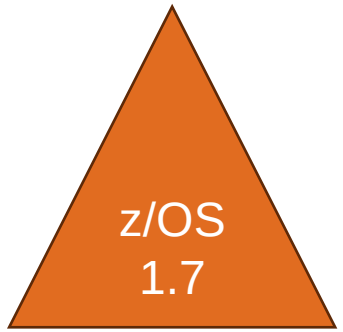
```
SETROPTS RACLIST(CDT) REFRESH
```

- CDT explained

LE

SHARE
EDUCATE • NETWORK • INFLUENCE

CEEOPTS



The old way

- *Historically, the method to specify your Language Environment run-time options was by creating different load module options for different environments(CICS, Batch, Unix etc) using the* CEEXOPT *macro*
- *System defaults are also specified in a load module*
- *Re-assemble for new z/OS versions*
- *Probably wrapped in a USERMOD.*
- *Requires a restart of some kind to pick up new options*

The new way

- For system wide options use the CEEOPTxx parmlib member
- Some useful MVS commands: D CEE, SET CEE=xx, SETCEE
- Use the CEEOPT DD card for specific address spaces(CICS is not supported, use the load module)
- Dynamic changes
- LE options explained

Questions?

Your feedback is important!

SHARE
EDUCATE • NETWORK • INFLUENCE

Submit a session evaluation for each session you attend:

www.share.org/evaluation