Addressing the Installation Exits Problem

Bob Rogers
Trident Services, Inc.

z Exchange

December 8, 2021

Session abstract

Over 50 years ago IBM allowed installations to customize their systems with assembler language code that runs as authorized extensions to the operating system. Since then, the underlying system has become more complex, the understanding of how to interact with the operating system has greatly diminished, the skills in assembler language have almost vanished, and concerns about the integrity and security of the system have increased.

All these factors lead us to seek a better solution for specifying JCL rules and other installation policies. This presentation describes a path to eliminating much of the installation exit code that now presents an issue, if not a threat, to the maintenance and smooth running of z/OS systems.

The big news is a new tool, called Exit Explorer, to identify and locate the active installation exit code on a system. Once identified, an installation can then re-implement the policy embodied in the assembler code with non-programming language specifications and retire the customer-written installation exit code. Going forward, the active policies can be transparently reviewed without having to decipher them from arcane decades old assembler language code.

The Exit Explorer is available for use from Trident Services free of charge or obligation. The presentation describes the problem in general terms and goes on to describing a specific solution approach based on examples using the zOSEM product customization capabilities.

Customization via Exit Programs

- From the earliest days of the operating system, provision has been made to customize system operation by augmenting the operating system with code provided by the installation.
- The operating system and associated products provided *exit points* from which installation *exit routines* are called.
- The installation exit code typically must be written in assembler language and execute in an authorized state.
- This is to say that an installation must express the JCL rules and other policies in assembler code, which because it runs in authorized state, can crash the system.

Current Concerns

- Even now, 50 years later, a good deal of customization must still be done using installation exits.
- But the level of skill in assembler language has diminished greatly.
- Worse, the knowledge of operating system internals required for implementing and maintaining installation exits has nearly vanished.
- The microfiche that was used years ago to learn about the operating system elements is no longer available (OCO).
- Many installations know little about the exit code that runs on their critical systems – sometimes not even the inventory of exits.

Why this presentation?

- In the following charts we try to point out the problems of maintaining installation code to customize z/OS systems – problems that many of us old-timers have become inured to.
- Poking around will find thousands of pages of documentation on customizing with exit routines – not a problem if you already know, but a real challenge to people coming on board.
- There are a number of solutions available for installations to start addressing the problem – ISV products and even some new capabilities from IBM like *JES2 Policies*.
- The *zOSEM* product from Trident is used in some examples.

"Who should read this information"[1]

"This information is primarily intended for system programmers who support accounting and billing services for an installation. It can be used by installation managers and system programmers who are responsible for problem resolution, system tuning, and capacity planning for a z/OS system. *This information assumes that the reader has **extensive experience with z/OS**, is familiar with its basic concepts, can code JCL statements to run programs or cataloged procedures, can code in assembler language, and can read assembler, loader, and linkage editor output."*

[1]*Excerpt from MVS System Management Facilities (SMF) SA38-0667-40 which contains the information on SMF exits.*

What does it take to write/maintain exit code

For a long-time experienced systems programmer:

- Have sufficient proficiency in assembler language
- Understand the policy that is being enforced by an exit routine
- Sufficiently understand the z/OS component or product that provides the customization exit point.
- Know (from the manual) the interface to the exit routine.
- Know how to verify that the exit routine does what it is supposed to do in all case…

  ….without blowing up the system

What one must learn

If the old-timer retires, the new guy must learn all of that, and…

- Assembler language programming is often learned “on the job” – dangerous when writing supervisor state code.
- It’s not unusual for the only documentation for policy that an exit is enforcing to be in the assembler module itself.
- Just for learning the interface to exit routines there is a ton that must be read and understood.
  - The JES2 Installation exits manual (SA32-0995-40) is almost 500 pages long and describes about 60 exit points.

Writing an exit for the SUBMIT command

- Just as an example, lets consider IKJEFF10, the TSO SUBMIT exit.
- It is called to examine the JCL of a job being submitted.
- It runs at high authority in Supervisor State.
- IBM provides a sample exit module that does nothing but housekeeping
- Among other things in the 10 pages of doc devoted to just this one exit, it lists some things you might do with it.
  - Cancel a SUBMIT request
  - Delete, Add or Modify a JCL statement
  - Customize the JOB statement, including adding a password
- Then the exit must be tested on a running system – typically a test LPAR or on z/VM.

Sample SUBMIT Exit

IBM provide IKJEXIT as a sample exit that does nothing but the housekeeping for the exit routine. Here is the list of housekeeping activities from the IKJEXIT sample prolog.

```
* OPERATION - IKJEXIT PERFORMS THE FOLLOWING FUNCTION:
*    1 - ESTABLISHES SAVE AREA TO CALLING PROGRAM
*    2 - SAVES PARAMETER POINTER (R1)
*    3 - DOES A GETMAIN FOR PROGRAM'S SAVE AREA
*    4 - CHAINS THE CALLER'S SAVE AREA AND THIS SAVE AREA
*    5 - DOES A GETMAIN FOR PROGRAM'S WORK AREA
*    6 - DETERMINES WHETHER TO USE PUTLINE OR WTO
*    7 - HEX FORMATS THE PARAMETER ENTRIES
*    8 - IDENTIFIES THE HEX PARAMETER ENTRIES
*    9 - PRINTS ALL OF THE PARAMETER ENTRIES VIA PUTLINE OR WTO
*   10 - DOES A FREEMAIN OF DYNAMIC STORAGE
*   11 - DOES A FREEMAIN OF DYNAMIC SAVE AREA
*   12 - LOADS REGISTER 14 WITH RETURN ADDRESS
*   13 - SETS THE RETURN CODE IN REGISTER 15 (ALWAYS RC=0)
*   14 - LOADS REGISTERS R0 - R12 WITH CALLER'S ENTRY CONTENTS
```

Must keep up with changes to IBM samples

- IBM provides a sample program for the exit IEFACTRT in SYS1.SAMPLIB(IEEACTRT). The sample is over 1200 lines of assembler code
- An installation can use this as a starting point for writing their own customized exit routine.
- IBM updates the sample as needed from time to time
  - IEEACTRT was updated to prevent it from taking a program check when the job/step or SRB CPU time rounds to zero.
  - If an installation used the sample as a base, someone needed to find out about this change and make the analogous change in the customized copy, test it, and put it into production.
  - Just upgrading to a fast processor can cause this abend to start occurring.

Do these changes affect your SMF exit code?

- **April 2021 refresh**
  - The Exit routine environment section for the IEFU86 - SMF record exit is updated. For more information see, “Exit routine environment” on page 240.
- **December 2020 refresh**
  - Updates are made to the description of parameter word 5 of the IEFUSI exit. See “Entry Specifications” on page 196. • Updates are made to the description of sub-word 4 of parameter word 7 of the IEFUSI exit. See “Entry Specifications” on page 196.
- **June 2020 refresh**
  - For BCP Exits, Table 9 on page 315 is updated.
- **Prior to June 2020 refresh**
  - The IEFACTRT installation exit is updated to add the parameter word 14. For more information, see Chapter 25, “IEFACTRT — SMF Job and Job Step Termination Exits,” on page 133.

Even if the answer turns out to be NO, time must be spent in research
… and this is just for the SMF exits.

Some help from IBM

- Mike Shorkend presented last year to UK GSE on recent work IBM has done to reduce the need for custom exit code.
- He covered:
  - JES2 Policies to replace JES2 exits with JSON. The z/OS 2.4 support is limited but expected to be increases with continuous delivery.
  - IRRPRMxx PARMLIB member to specify RACF data set names and options
  - SMFLIMxx PARMLIB member to mostly replace IEFUSI
- z/OS 2.5 adds new phases of processing and new attributes
- This excellent presentation provides examples of the parmlib members and a short tutorial on JES2 Policies.

https://conferences.gse.org.uk/2020/presentations/4AH.pdf

JES2 Policies Introduced in z/OS V2R4

- IBM says (z/OS 2.4.0 – z/OS JES2 Installation Exits),
  - *"The disadvantage of JES2 exit interface is that it involves programming at a low level and requires knowledge of JES2 control structures and detailed understanding of how JES2 internal processing works."*
  - *"JES2 policies provide an alternative way to customize JES2 processing. Creating JES2 policy does not require programming."*
  - *"JES2 policy definition text is a JSON object. Each policy type has its own set of JSON names (entries) that can be used in a policy definition. However, there are JSON names common for all JES2 policies and syntax rules that apply to policies of all types."*
- JES2 Policies seems to require a language, even if not a "programming language".

Addressing the Exits Issues

- The first step is to locate and make an inventory of the exit code.
- Trident Services has created a powerful utility to find and identify the installation exit routines associated with the exit points provided by z/OS elements and related products.
- Having identified the exit routines and located the source code, it's possible to "reverse engineer" the installation policy they are attempting to enforce.
- Even if there is written documentation, it must be verified that it describes the policies actually being enforced by the exit code.
- Once the installation policy is determined, much of the exit code can be replace with, for example, zOSEM functionality and the installation's customized exit code can be retired.

The Exit Explorer Tool Produces a Report

- The Exit Explorer searches the system for more than 500 exit points.
- The exit points are listed by Group, e.g. JES2 or TSO.
- The exit point name, the name and location of the exit load module are listed.
- The first 80 bytes of the exit load module are listed. This may provide a clue as to the source of the module.

This is the beginning of the listing for SMF exits:

```
...EXITPOINT....  ..NAME..  LOCATION  ..ADDR..  LENGTH  LOADMOD.  ..ADDR..  LENGTH    .........LIBRARY DATASET NAME
*****  SMF EXITS
SYSASCH.IEFACTRT  IEFACTRT  PLPA       46233A8     100  IEFACTRT   46233A8           SYS1.LPALIB
   EXIT TEXT: "    IEFACTRT05/13/90 HBB4410     00    IEFTB724  89.268         0 =     "   0  "
SYSJES2.IEFACTRT  IEFACTRT  PLPA       46233A8     100  IEFACTRT   46233A8           SYS1.LPALIB
   EXIT TEXT: "    IEFACTRT05/13/90 HBB4410     00    IEFTB724  89.268         0 =     "   0  "
SYSOMVS.IEFACTRT  IEFACTRT  PLPA       46233A8     100  IEFACTRT   46233A8           SYS1.LPALIB
   EXIT TEXT: "    IEFACTRT05/13/90 HBB4410     00    IEFTB724  89.268         0 =     "   0  "
SYSSTC.IEFACTRT   IEFACTRT  PLPA       46233A8     100  IEFACTRT   46233A8           SYS1.LPALIB
   EXIT TEXT: "    IEFACTRT05/13/90 HBB4410     00    IEFTB724  89.268         0 =     "   0  "
. . . . . .
. . . . . .
DEFINED EXITPOINTS:  70, ACTIVE EXITPOINTS:   64, ACTIVE EXITS:   77
```

Key to the Exit Explorer Report

Exit Entry: Each exit entry within an exit section consists of two lines of information.

Line One: The first line of information contains the following fields:

EXITPOINT The name of the exit point.

NAME The name of the exit program.

LOCATION The location where the exit program was found. Possible values are:

DYN-LPA The exit program was found in the Dynamic link pack area.
Dynamic LPA is also known as active LPA.
FLPA The exit program was found in the Fixed link pack area.
LINKLIB The exit program was found in a link list dataset.
MLPA The exit program was found in the Modified link pack area.
PLPA The exit program was found in the Pageable link pack area.
STEPLIB The exit program was found in a library specified on the STEPLIB DD statement.
UNKNOWN The zOSEM Exit Explorer utility could not determine the location of the exit program.

ADDR The address of the exit program. LENGTH The length, in bytes, of the exit program.

LOADMOD The load module name. ADDR The address of the load module.

LENGTH The length, in bytes, of the load module.

Line Two: Displays the first 80 bytes of the exit load module.

The Tool Finds Exit Routines in these Groups

- ABARS - Aggregate Backup And Recovery Support
- **ALLOC - Allocation**
- ARM - Automatic Restart Management
- CONSOLE - Consoles
- **DFSMS - Data Facility Storage Management Subsystem**
- DLF - Data Lookaside Facility
- DSS - Data Set Services
- DUMPS - Dumping Services
- DYN_LPA - Dynamic LPA Service
- FTP - File Transfer Protocol
- GRS - Global Resource Serialization
- HLTH_CHK - Health Checker
- HISSERV - Hardware Instrumentation Services
- **HSM - Hierarchical Storage Management***
- IEHINITT - System Utilities
- **JES2 - Job Entry Subsystem 2**
- LANG_ENV - Language Environment
- LLA - Library Lookaside
- LOGR - System Logger
- MESSAGE - Message Processing
- **RACF - Resource Access Control Facility**
- SAF - System Authorization Facility
- SDUMP - SVC Dump
- SLIP - Serviceability Level Indication Processing
- **SMF - System Management Facilities**
- SMS - Storage Management Subsystem
- SUBSYS - Subsystem Interface (SSI)
- SYMREC - Symptom Record
- **SYSTEM - System**
- **TSO - Time Sharing Option**
- USS - UNIX System Services
- XCF - Cross-system Coupling Facility

Installing and running the Exit Explorer

- The distribution file for the utility is a binary file image of a TERSED dataset (i.e. it is compressed using the AMATERSE utility).
- A JCL sample for unloading the tool onto your system is provided.
- The tool must reside in an authorized library.
- A users' guide is provided which contains information on how to run the tool and interpret the report.
- Requests for a copy of the Trident Exit Explorer can be sent to:
  INQUIRY@TRISERV.COM

Determining Installation Policy

- Once the active installation exit code has been identified the corresponding source code must be located.
- The source code can be “reverse engineered” to determine the active JCL rules and other policies.
- You can ask the vendor for help with this if you are willing to send the source modules and macro libraries for examination.
- Once the policy is determined, it can be specified in the non-programming way and, after activation, be enforced.

Specifying Policy and Rules

In the case of zOSEM, the installation specifies the JCL and Job rules on ISPF panels.

- The policy can be updated and later activated.
- The updated policy can be activated dynamically without re-IPL or recycling JES2 or other subsystems.
- Additional custom exit routines can be daisy-chained if there is something not addressed by the panel-driven policies.

Assembler Language vs ISPF Panels

Here's just a small snippet from the IEEACTRT assembler sample to create a usage statistics "flowerbox" for steps and jobs.

```
         LR    R04,R09             Point to the SMF30 record.      @PAA
         A     R04,SMF30POF        Point to Perf section.          @PAA
         USING SMF30PRF,R04        Perf section addressability     @PAA

         SGR   R00,R00             Zero for later division.        @PAA
         SGR   R01,R01             Clear reg1                      @02A
         LH    R15,SMF30CPC        Acquire CPU SDC scaled by 10.   @02M
         LTR   R15,R15             Check for non-zero SMF30CPC     @02A
         JZ    FMTSCPU             Jump around computations        @02A
         LG    R01,SMF30CSU_L      Acquire CPU service units.      @PAA
         MSGF  R01,F10             SMF30CSU multiply by 10         @PAA
*        Perform multiplications before the division to preserve   @PAA
*        as much precision as possible.                            @PAA
         MSGF  R01,SMF30SUS        multiply by SMF30SUS            @PAA

         LLGTR R15,R15             Extend to 64-bits.              @PAA
         DLGR  R00,R15             divide by SMF30CPC              @PAA
         SRLG  R01,R01,4           divide by 16.                   @PAA
FMTSCPU  DS    0H                  Format step CPU time            @02A
```

The IBM-supplied skeleton for this routine is over 1200 lines of assembler code.

An installation may have dozens of hand-coded modules like this.

zOSEM provides dozens of panels for the installation to clearly and conveniently state the JCL rules and policies.

This is the zOSEM panel for specifying options for job and step ends

Step End Statistics

```
****************************************************************************************************************************************
*                                          *                           *                            *                                *
*           TRIDENT SERVICES               *          TIMINGS          *    VIRTUAL STORAGE USE     *          PAGING DATA           *
*              DEVELOPMENT                 *                           *                            *                                *
*              CAPITOLA, CA                * ELAPSED.......  01:11.40  * REGION REQD......6,144K   *  PAGES IN................0     *
*                                          * CPU(TCB)......  01:04.83  * USED BELOW.........96K    *  PAGES OUT...............0     *
*      z/OS REL 2.2 - 1090/SYSA            * CPU(SRB)......  00:00.45  * USED ABOVE.......4,964K   *  SWAPS IN................0     *
*                                          * CPU(ZAAP).....  00:00.00  * SYSTEM BELOW.......492K   *  SWAPS OUT...............0     *
* JOB.....OSEMTVFY  STEP.........STPJOB    * CPU(ZIIP).....  00:00.00  * SYSTEM ABOVE....12,144K   *  VIO PAGES...............0     *
*                                          * INIT..........  00:00.05  * MAX BELOW.......10,216K   *  SWAP COUNT..............0     *
* PRCSTP..........  DATE........03/22/19   * I/O INTRPT....  00:00.92  * MAX ABOVE........1,741M   *  WORKING SET.........4520K     *
*                                          * RESIDENT......  00:29.08  * 64-BIT PVT.........16M    *                                *
* PGM.....IKJEFT1B  COMP CODE.......0000   * ACTIVE........  00:29.08  * 64-BIT SHR..........0M    *           TOTAL I/O            *
*                                          * DEV CONNECT...  00:00.00  * MEMLIMIT   ..........0M   *  TAPE....................0     *
* STEP NUMBER....3  SRV CLASS...BATCHSYP   *                           *                            *  DASD...................73     *
*                                          * STEP START....  10:15:06  *       SERVICE UNITS        *  VIO.....................0     *
* EXEC PRTY......0  TAPE MOUNTS........0   * ALLOC START...  10:15:06  * TCB.............468,596    *  OTHER...................0     *
*                                          * PGM START.....  10:15:06  * SRB...............3,896    *  SYSTEM..............4,482     *
* ACCOUNT:........................OS$EM    * STEP END......  10:16:17  * I/O...............4,546    *  TOTAL...............4,555     *
*                                          *                           * MSO...................0    *                                *
* ESTIMATED COST:.................$0.00    * INST(TCB).......24,656M   * ESU...................0    *  TIOT SIZE..........65,503     *
*                                          * INST(SRB).......386,403   * TOTAL...........477,038    *  PERCENTAGE USED........1%     *
****************************************************************************************************************************************
```

Job End Statistics

```
****************************************************************************************************************************************
*                                          *                           *                            *                                *
*           TRIDENT SERVICES               *          TIMINGS          *       SERVICE UNITS        *          I/O ACTIVITY          *
*              DEVELOPMENT                 *                           *                            *                                *
*              CAPITOLA, CA                * QUEUE.........  00:11.63  * TCB...........1,171,081    *  TAPE....................0     *
*                                          * ELAPSED.......  14:54.08  * SRB...............5,069    *  DASD..................161     *
*      z/OS REL 2.2 - 1090/SYSA            * CPU(TCB)......  02:42.01  * I/O...............6,260    *  VIO.....................0     *
*                                          * CPU(SRB)......  00:00.59  * MSO...................0    *  OTHER...................0     *
* JOB.....OSEMTVFY  COMP CODE.......0000   * CPU(ZAAP).....  00:00.00  * ESU...................0    *  SYSTEM..............6,184     *
*                                          * CPU(ZIIP).....  00:00.00  * TOTAL.........1,182,410    *  TOTAL...............6,345     *
* STEPS..........7  DATE........03/22/19   * INIT..........  00:00.85  *                            *                                *
*                                          * I/O INTRPT....  00:01.18  *         EXTENSIONS         *          TAPE MOUNTS           *
* JOB PRTY.......0  SRV CLASS...BATCHSYP   * RDR START.....  10:10:40  *                            *                                *
*                                          * JOB START.....  10:10:51  * CPU......0   WAIT......0   *  SPECIFIC................0     *
* ACCOUNT:........................OS$EM    * JOB END.......  10:25:45  *                            *  NON-SPECIFIC............0     *
*                                          *                           * TIOT STORAGE USE.....1%    *                                *
* ESTIMATED COST:.................$0.00    * INST(TCB).......62,877M   * STEP 14.............ASM    *                                *
*                                          * INST(SRB).....1,424,338   * PRCSTP.........ASSEMBLE    *                                *
*                                          *                           *                            *                                *
****************************************************************************************************************************************
```

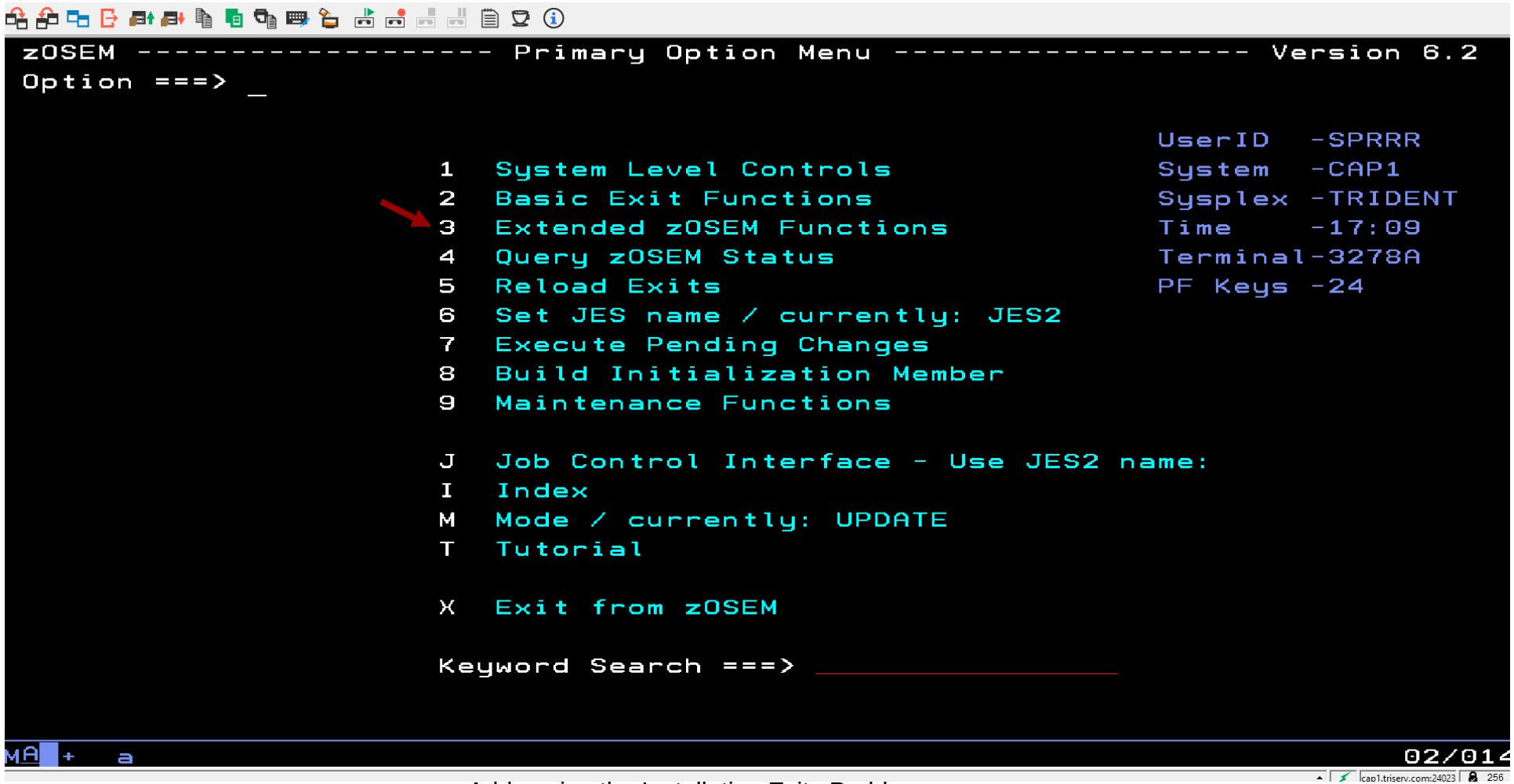
“Flowerboxes” with a click

Simplifying z/OS Upgrades

- Translate your existing exit functionality to zOSEM functions or other ISV product or JES2 Policies (for JES2 exits)
  - Specify policy via ISPF panels or non-programming language instead of writing assembler code
  - Check for new support for customization with parmlib rather than code.
  - Ask the vendor or IBM for help with the specification
- Eliminate (or greatly reduce) your custom assembler language exits
- For the next z/OS upgrade:
  - ISVs and IBM typically supports new z/OS releases at GA
  - At worst, apply appropriate IBM or ISV PTFs
  - Your customizations are preserved with no effort

Some More Examples

Primary Option Menu



Controlling the use of SYSOUT Classes

It's not uncommon for an installation to want to control the SYSOUT classes that different categories of jobs can use.

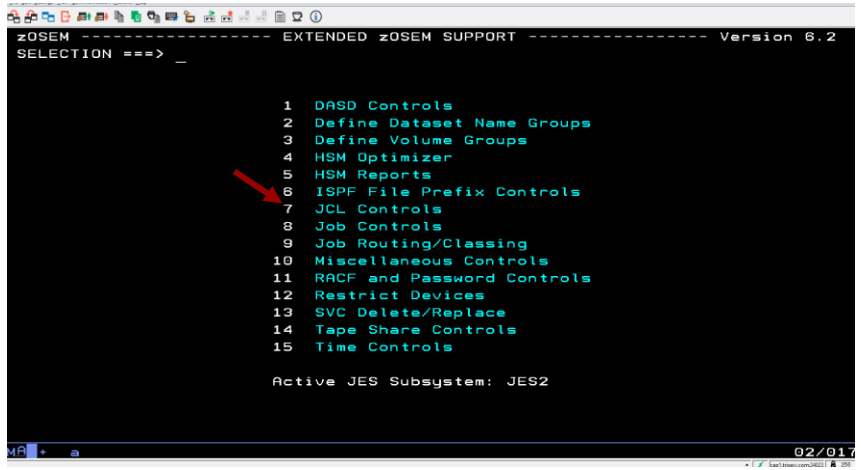
Suppose a site wants to restrict the use of certain SYSOUT classes to production workloads only.

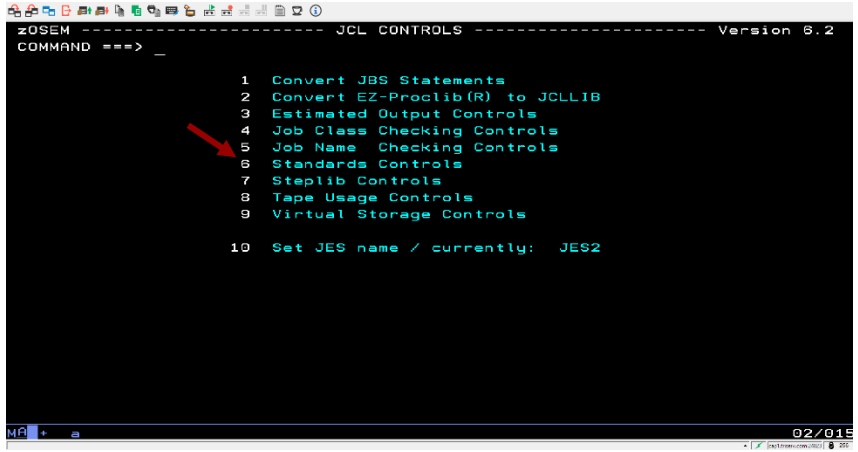
- Exit Implementation: Develop code in a JES2 EXIT 6 to:
  - Analyze DD JCL statements for SYSOUT= *class*
  - Extract the class value
  - Call security product after incorporating the SYSOUT class value in the security profile
  - Evaluate the response from security. If access is denied, issue notification messages and terminate job
- Estimated programming effort: 2K+ lines of code.

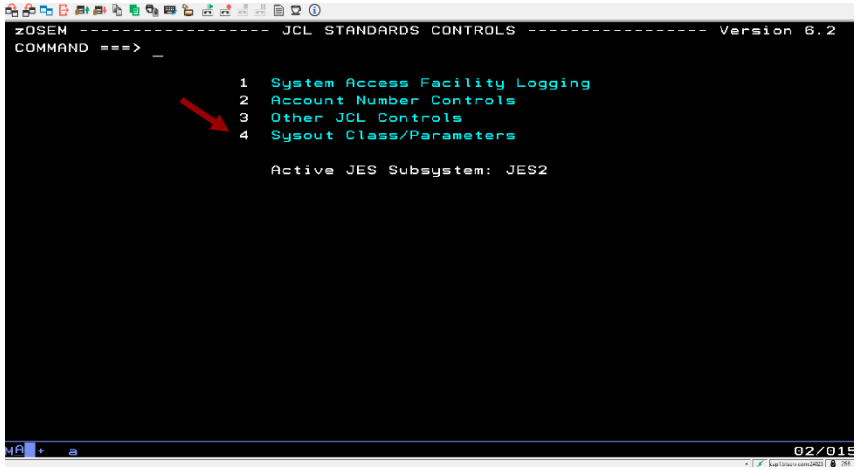
OR

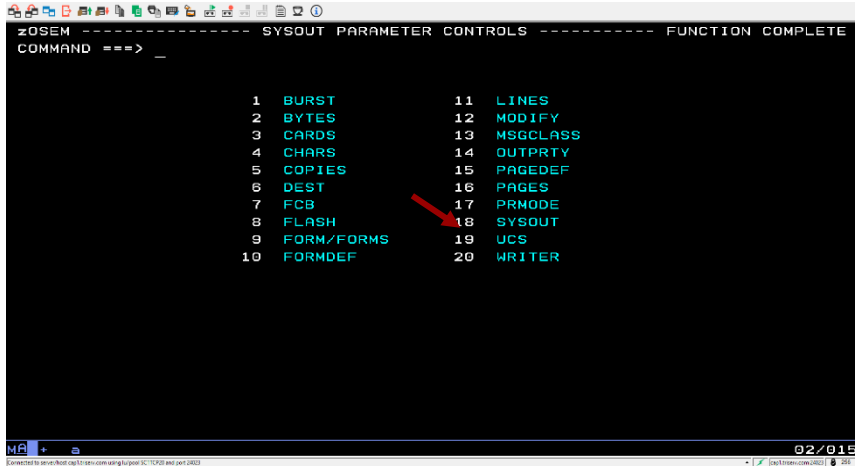
ISPF Dialog option path 3.7.6.4.18

Doing it without programming

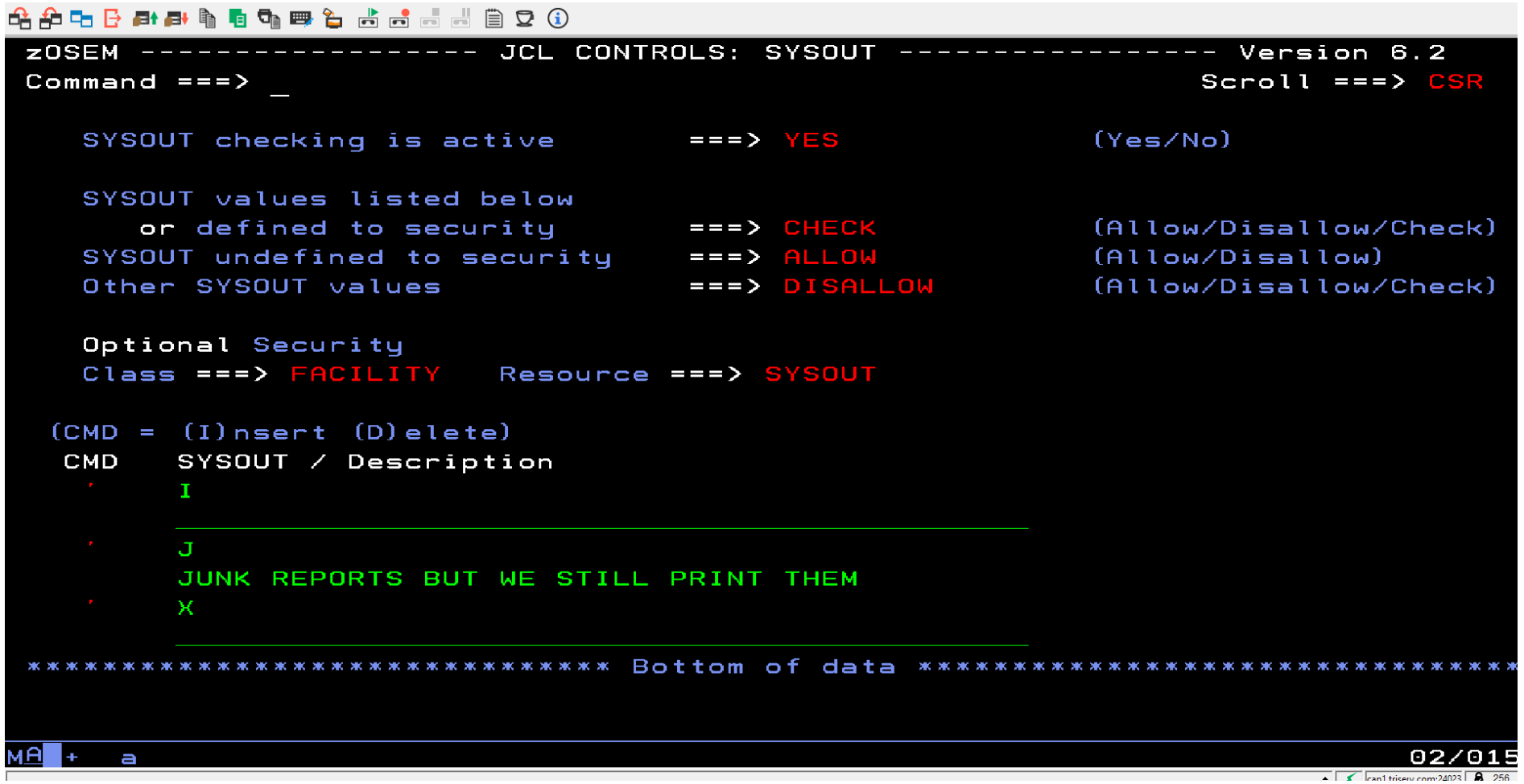








JCL Controls SYSOUT Menus



Controlling the Jobs Submitted from TSO

1. Have the job name start with the first 6 characters of the TSO userid
2. Ensure that jobs being submitted only use job classes authorized for TSO-submitted batch work.

Exit Implementation: Develop code in a TSO IKJEFF10 exit to analyze the JOB JCL statement

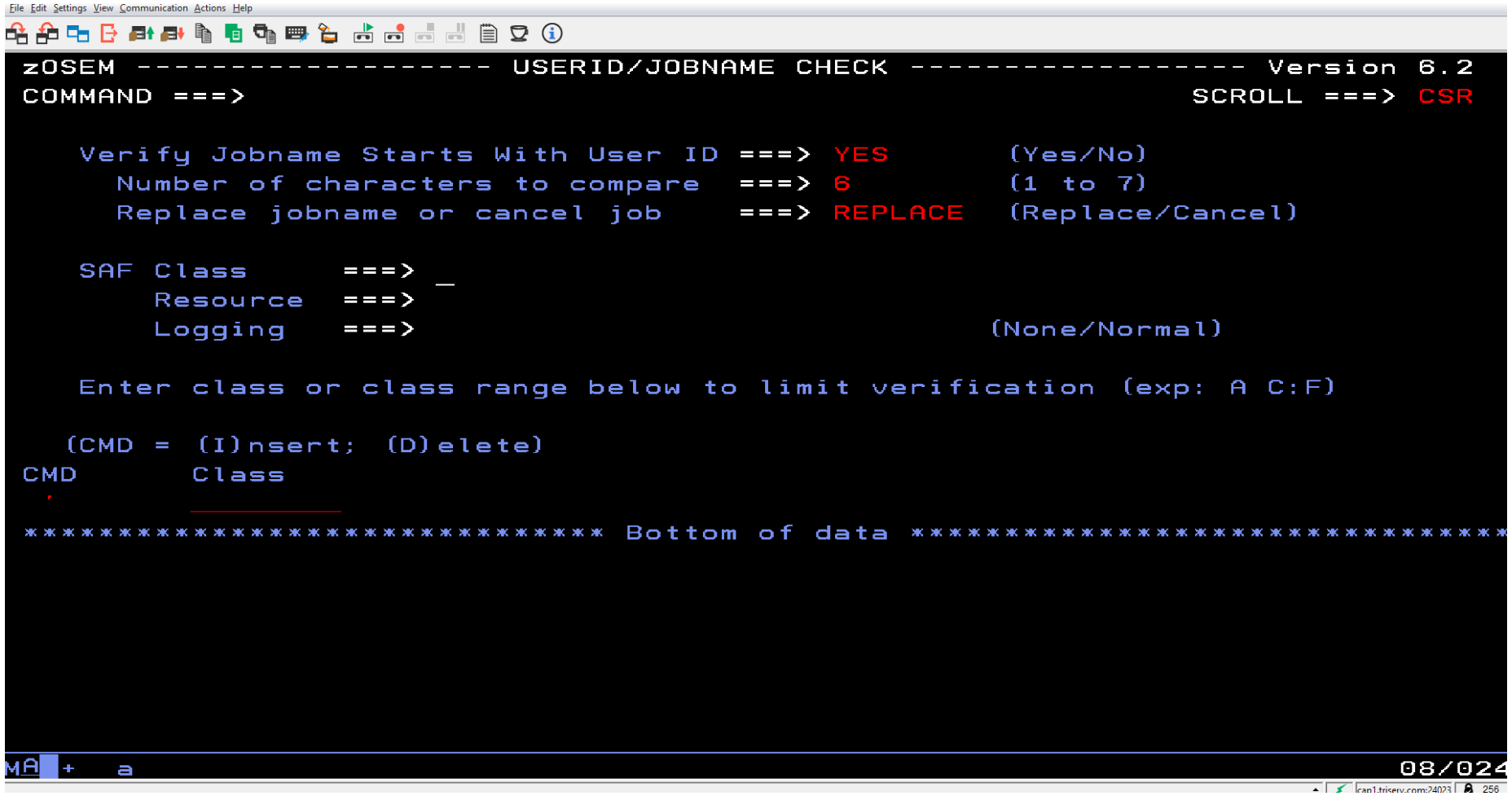
1. If the first 6 characters of the Jobname do not match the submitter's TSO USERID, move the USERID into the job name
2. Check external security to ensure that the user is authorized to submit jobs using the specified job class. Abort job submission if not.

- Estimated programming effort: 2K+ lines of code.

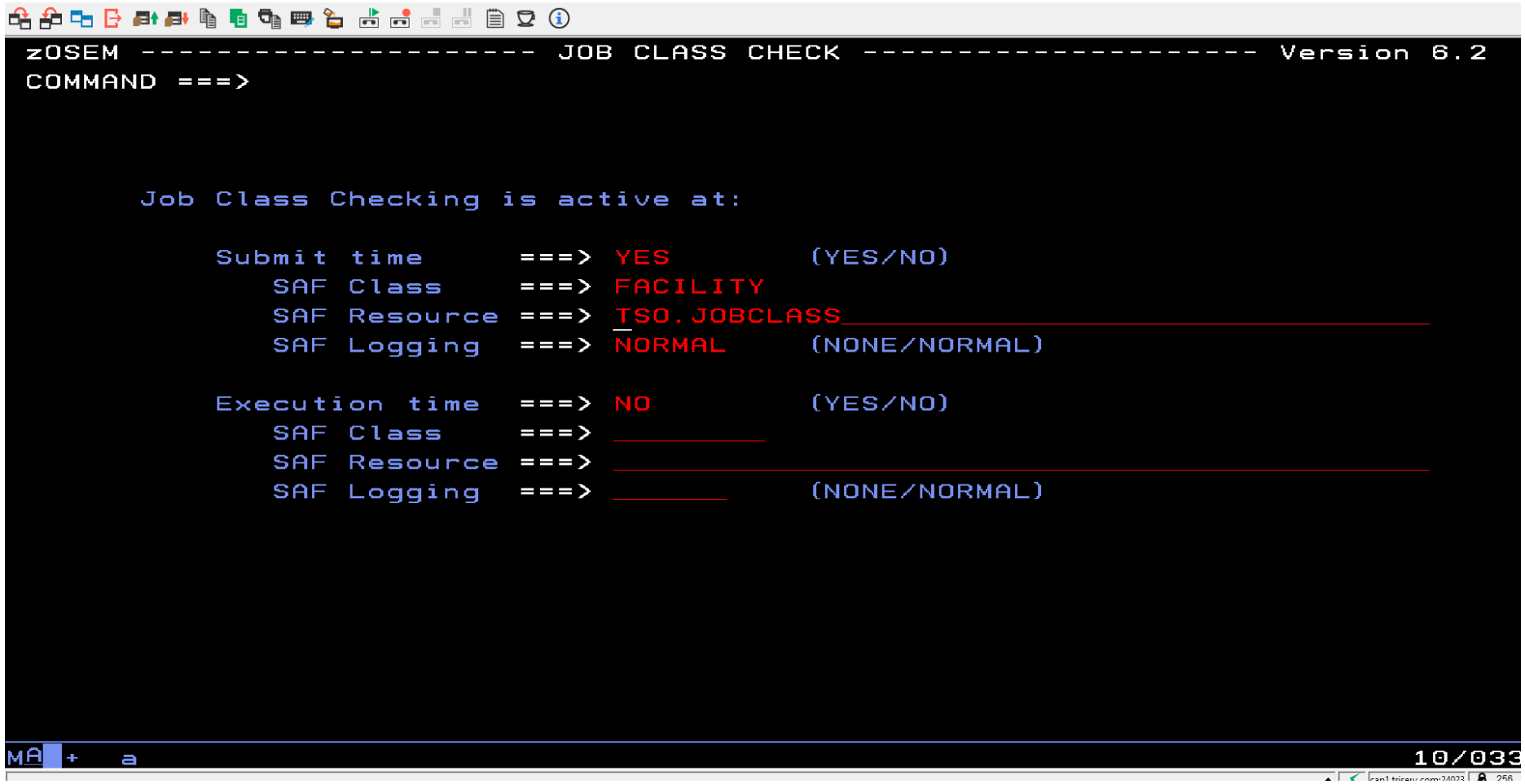
OR

1. ISPF Dialog option path 3.8.11
2. ISPF Dialog option path 3.7.4

Checking Jobname against USERID



Checking Authorization to Use Job Class



Controlling the Jobs Submitted from TSO

The installation wants to use job classes G & W are for ‘hot’ batch jobs (i.e. short running jobs). Jobs should not execute longer than the TIME specification in the JES2 JOBCLASS definition.

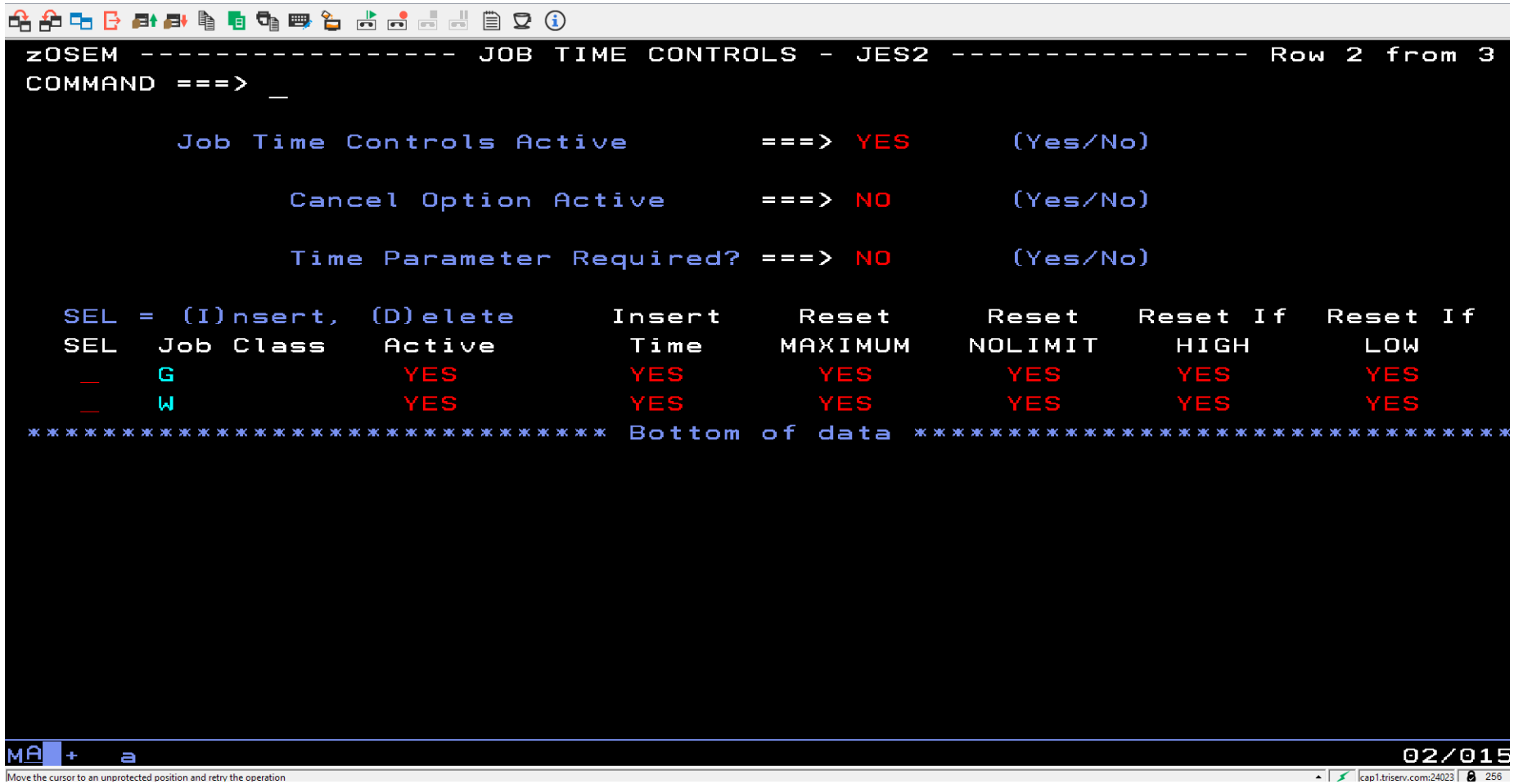
Exit Implementation: Develop code in a TSO IKJEFF10 exit to analyze the JOB JCL statement looking for the TIME= parameter

- Unless the TIME= value is less, set it to the value defined in the JES2 JOBCLASS definition
- Estimated programming effort: 1.5K+ lines of code.

OR

ISPF Dialog option path 3.15.1

Limiting CPU time for “Hot” Job Classes



Thank You