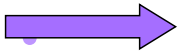
Considerations for Debuggin and Tuning Java in CICS

CICS and Java Wildfire Workshop

Steve Fowlkes, fowlkes@us.ibm.com
Leigh Compton, lcompton@us.ibm.com

Abstract

- This presentation covers Java debugging and tuning in CICS TS V5.x and V6.1
- It canvases several of the debugging and tuning options which, in most cases, apply to all CICS Java environments



Note:

This presentation is not meant to be comprehensive – to get more information on any area, refer to the CICS and Java documentation.

Agenda

- Debugging
  - CICS Infrastructure
  - The editor
    - CICS Explorer
  - CICS Messages and Abends
  - Stdout and stderr
  - CEDF/CEDX/CECI/CEBR
  - CICS Trace
  - CICS Statistics
  - Interactive Java debugger
  - Java Dumps
  - Infrastructure (IBM Healthcenter)
- Tuning
  - JVM Configuration
  - Heap usage
  - Garbage Collection
  - Miscellaneous

Sources of information

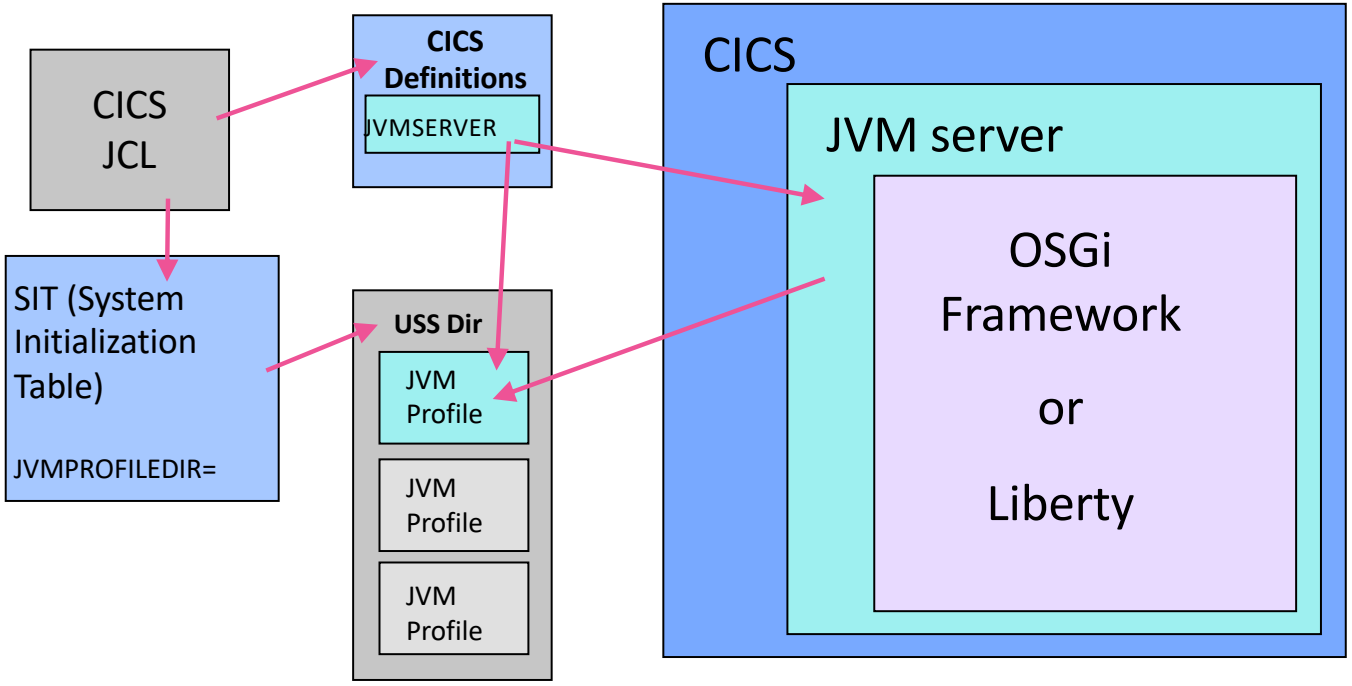
- CICS product documentation:
  - V5.5 -- https://www.ibm.com/docs/en/cics-ts/5.5
  - V5.6 -- https://www.ibm.com/docs/en/cics-ts/5.6
  - V6.1 -- https://www.ibm.com/docs/en/cics-ts/6.1
- Troubleshooting Java applications
  - ?topic=troubleshooting-java-applications
  - https://www.ibm.com/docs/en/cics-ts/5.6?topic=troubleshooting-java-applications
- Improving Java performance
  - ?topic=performance-improving-java
  - https://www.ibm.com/docs/en/cics-ts/5.6?topic=performance-improving-java

JVM servers in CICS

- CICS uses a JVMSERVER resource to define the properties of a Java Virtual Machine
- Types of JVM servers:
  - OSGi
  - Liberty Profile
  - Classpath
    - AXIS2 capable
    - Security Token Server (STS) capable
    - Batch capable
    - Mobile capable

```
JVmserver       : LYCWLP
Group           : LYCJAVA
DEScription     : CICS JVM server
Status          : Enabled
Jvmprofile      : LYCWLP
Lerunopts       : DFHAXRO
Threadlimit     : 015
```

CICS JVM Environment



JVM Profile

1 of 5

Samples provided

- DFHOSGI for OSGi environment (CICS-style applications)
- DFHWLP for Liberty Profile (JEE-style applications

Options starting with '-' are passed to the JVM without being parsed by CICS

Options starting with

- -D are standard JVM system properties
- -X are non-standard options

Symbols

- &APPLID; - represents the APPLID of this region
- &DATE; - the current date in the format Dyymmdd
- &JVMSERVER; - name of the JVMSERVER (unique output of dumps)
- &TIME; - JVM start time in the format Thhmmss

PRINT_JVM_OPTIONS={YES|NO}

- If yes, the JVM startup options are also printed to SYSPRINT, including those not visible in JVM profile

JVM Profile

2 of 5

- JAVA_HOME=/usr/lpp/java/J8.0_64/
- JAVA_DUMP_TDUMP_PATTERN=
  - Name of Java TDUMP in the event of a JVM abend
  - Can use symbols (&APPLID, etc)
- LIBPATH_SUFFIX=
  - Library files that are to be added after the LIBPATH that CICS builds
- TZ=
  - the local timezone
- JVMTRACE={&APPLID;.&JVMSERVER;.Dyyyymmdd.Thhmmss.dfhjvmtrc|file_name}
- STDOUT={&APPLID;.&JVMSERVER;.Dyymmdd.Thhmmss.dfhjvmout|file_name}
- STDERR={&APPLID;.&JVMSERVER;.Dyymmdd.Thhmmss.dfhjvmerr|file_name}
- STDIN={file_name}
- USEROUTPUTCLASS=
  - Name of a routine to process stdout and stderr messages

JVM Profile

3 of 5

- `-agentlib:jdwp=`
  - Specifies whether debugging support is enable in this JVM
- `-agentlib:jdwp=transport=dt_socket, server=y,address=<port>,suspend=n`
- `-Xshareclasses`
  - The JVM connects to a cache or creates one if it doesn't exist
- `-agentlib:healthcenter`
  - The IBM HealthCenter can be attached at the specified port
  - Alternative -Xhealthcenter
- `-Xms`
  - Specifies the initial size of the heap
- `-Xmx`
  - Specifies the maximum size of the heap
- `-Xscmx`
  - Specifies the size of the share class cache (minimum is 4KB)

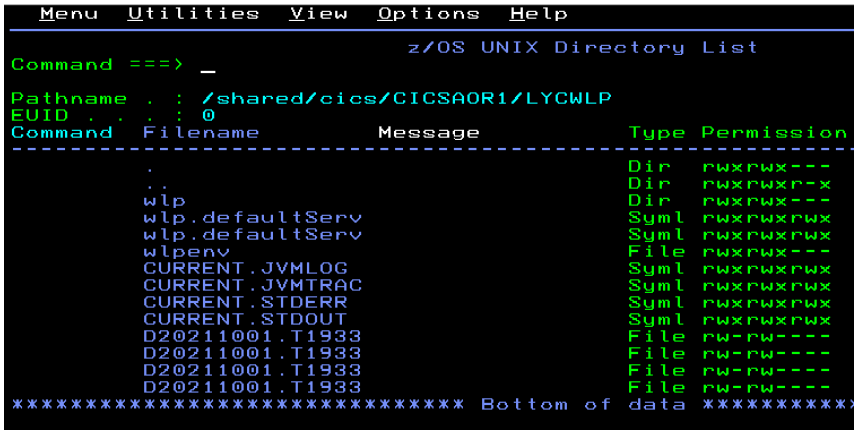
JVM Profile

4 of 5

- **`WORK_DIR={.|directory_name}`**
  - The working directory for this JVM
  - A '.' says use home directory of the User ID under which CICS region is executing
  - If WORK_DIR is omitted, /tmp is used
- WORK_DIR contains:
  - Log files
  - OSGi cache directory
  - OSGi log
  - Liberty Profile directory

Example:

- WORK_DIR=/shared/cics/



JVM Profile

5 of 5

Properties for OSGi

- **OSGI_BUNDLES=**
  - middleware OSGi bundles
- **OSGI_FRAMEWORK_TIMEOUT=nn**
  - timeout for OSGi infrastructure

Properties for Liberty

- `WLP_INSTALL_DIR=&USSHOME;/wlp`
- `CICS_WLP_MODE=STANDARD|INTEGRATED`
- `WLP_OUTPUT_DIR=./&APPLID;/&JVMSERVER;/wlp/user/servers`
- `-Dcom.ibm.cics.jvmserver.wlp.autoconfigure=true`
- `- Dcom.ibm.cics.jvmserver.wlp.jdbc.driver.location=/usr/lpp/db2v11/jdbc`

What tools are available for debugging?

Java-oriented tools

- stdout and stderr
- Interactive debugger
  - JPDA – works with any Java debugging tool
- Java Dumps
  - Javadump Heapdump, CEEdump, Sysdump
- Infrastructure Tuning
  - IBM Support Assistant
    - Heathcenter
    - Garbage collection and memory visualizer

CICS-oriented tools

- CICS messages and abends
- CEDF / CEDX / CECI / CEBR
- CICS Trace
- CICS Statistics
- CICS Explorer

Application tools

- Editors
  - IDz, RAD, Eclipse, VS Code, etc.

stdout and stderr

- Location for stdout and stderr specified in JVM profile
- Great for leaving bread crumbs (messages) during development (application writes messages to stdout and stderr)
  - Don’t make it needlessly large !
  - Be sure you don’t write to these more than necessary during production since these are usually ‘unmanned’ (operations doesn’t normally look at these)
- Can redirect stdout and stderr (some overhead)
  - USEROUTPUTCLASS in JVM profile
  - Sample source is com.ibm.cics.samples.SJMergedStream

JDPA

- Create a Debugging JVM server with –agentlib parameter
  - Specify suspend=n so JVM doesn't wait on attaching the debugger
- Only one person can use the JPDA debugger per JVM

Example:

- `-agentlib:jdwp=transport=dt_socket, server=y,address=8000,suspend=n`

Using JDPA

- Step through program and examine/change the variables
- Look inside a running program
  - Pause at any point
  - Look at the variables and their contents
  - Find ‘bugs’ in your program
- Use JPDA before you have problems
  - Learn more about how the program works
    - What the program does
    - How it works
  - Get knowledgeable on good programming techniques

Java dumps

Java dumps (-Xdump:java)

- contain information that relates to the JVM and the Java application
  - operating environment, locks, threads, hooks, shared classes, and class loaders
- https://www.ibm.com/docs/en/SSYKE2_8.0.0/openj9/dump_javadump/index.html

Heap dumps (-Xdump:heap)

- show the content of the Java heap
- https://www.ibm.com/docs/en/SSYKE2_8.0.0/openj9/dump_heapdump/index.html

System dumps (-Xdump:system)

- contain a raw process image or address space of an application.
- https://www.ibm.com/docs/en/SSYKE2_8.0.0/openj9/dump_systemdump/index.html

JVM dump agents

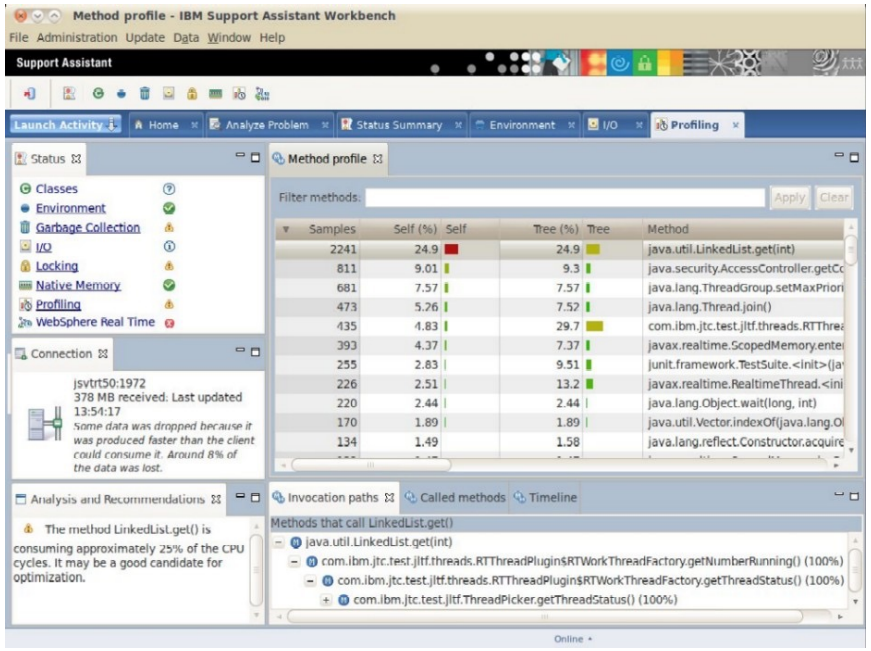
- https://www.ibm.com/docs/en/sdk-java-technology/8?topic=options-xdump#dump-agents

IBM Support Assistant

- Helps collect data and search for PMRs, can add tools: Java troubleshooting, configuration analysis, log analysis, and more
- Several tools for Java trouble shooting can be installed
  - Health Center
    - Discover which methods are taking the most time to run (Profiling)
    - Memory leaks
    - Visualize and tune garbage collection
    - Much more
  - Dump Analyzer
  - Garbage Collection and Memory Visualizer
  - Heap Analyzer
  - Performance Analysis Tool for Java

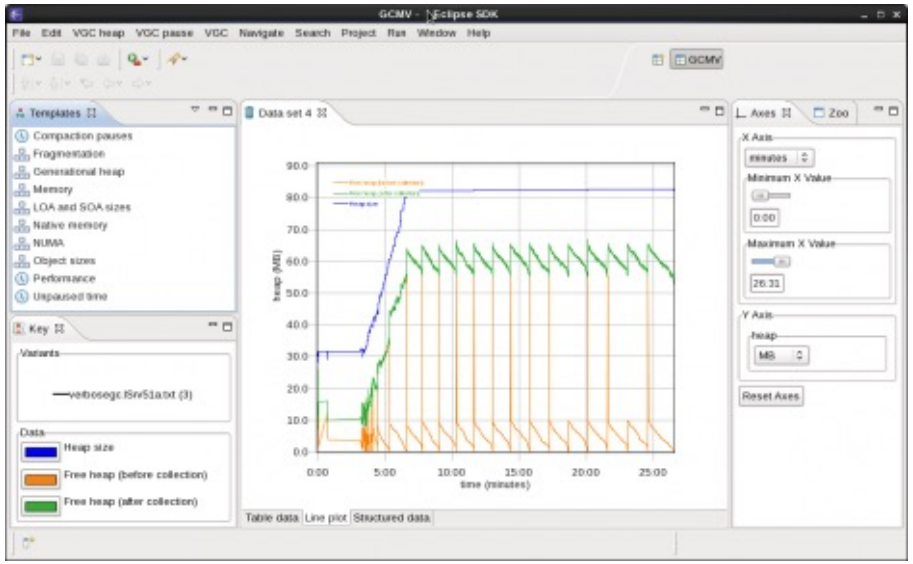
See https://www.ibm.com/support/pages/node/6376832 for release notes and download/installation instructions

IBM Health Center



- Health Center is a diagnostic tool for monitoring the status of a running Java or Node.js application.
  - Uses a small amount of processor time and memory
  - Can open some log and trace files for analysis
  - Available as part of IBM Support Assistant or independently via Eclipse Marketplace
- Overview:
  - https://www.ibm.com/support/knowledgecenter/SSYKE2_8.0.0/com.ibm.java.lnx.80.doc/diag/tools/tool_hctool.html
- Health Center Eclipse plug-in
  - https://marketplace.eclipse.org/content/ibm-monitoring-and-diagnostic-tools-health-center

IBM Garbage Collection and Memory Visualizer



- Garbage Collection and Memory Visualizer (GCMV) is a diagnostic tool for plotting and analyzing garbage collection data
  - Helps diagnose Node.js and Java application memory and performance problems
  - Provides recommendations to improve performance
  - Available as part of IBM Support Assistant or independently from Eclipse Marketplace
  - https://marketplace.eclipse.org/content/ibm-monitoring-and-diagnostic-tools-garbage-collection-and-memory-visualizer-gcmv

Memory Analyzer

- Memory Analyzer is a Java heap analyzer
  - Helps find memory leaks and reduce memory consumption
  - Analyze heap dumps
    - Calculate the retained sizes of objects
    - See who is preventing the Garbage Collector from collecting objects
    - Run a report to extract leak suspects
  - Available from Eclipse Marketplace
  - https://marketplace.eclipse.org/content/memory-analyzer-0

CICS Messages and Abends

- CICS job output
  - Missing CICS resource (programs, files, etc)
  - Abend codes
- CSMT, CEJL, and CJRM transient data destinations (default to the MSGUSR DD)
- SYSPRINT
  - For example messages if the JVM can't start
  - If SYSPRINT DD omitted, a JES file named SYSnnnnn will be dynamically allocated

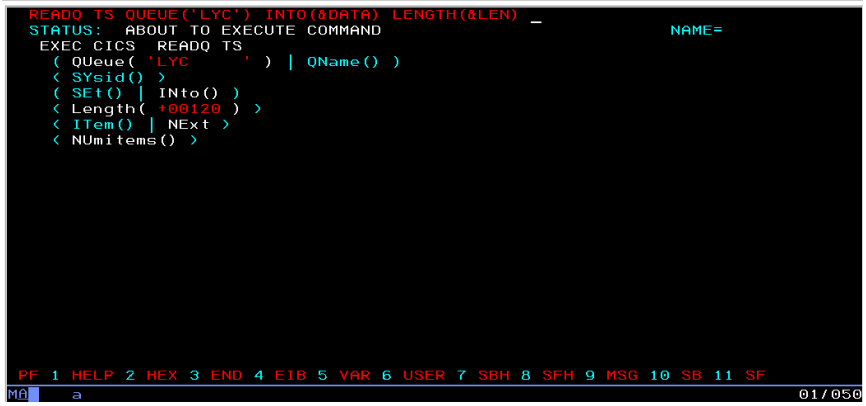
CICS Diagnostic Transactions

- CEDF & CEDX
  - Execution diagnostic facility
  - Read-only versions: CEDG & CEDY
- CECI
  - Command Interpreter
- CEBR
  - Browse queues

CEDF & CEDX

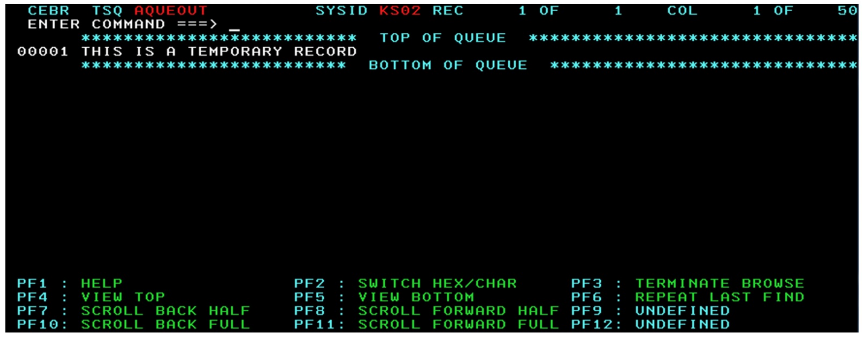
- CICS supplied transactions
  - Initiated in 3270 window
- Intercept all EXEC CICS commands
  - As well as EXEC SQL, EXEC DLI, and requests processed through RMI
- CEDF
  - Used to examine tasks executing on a terminal
  - CEDF <termid>
- CEDX <transid>
  - Used to examine non-terminal tasks
  - Puts TRANID into temporary TCLASS with limit of 1
- Displays before and after capture of each command
- Works for any supported programming language

CECI



- Command-level interpreter
  - Check the syntax of CICS commands
  - Process these commands interactively
  - Runs on a 3270 terminal
- Provides reference to syntax
  - Command level API
  - Command level SPI.
- Runs as a conversational transaction
  - Single unit of work

CEBR

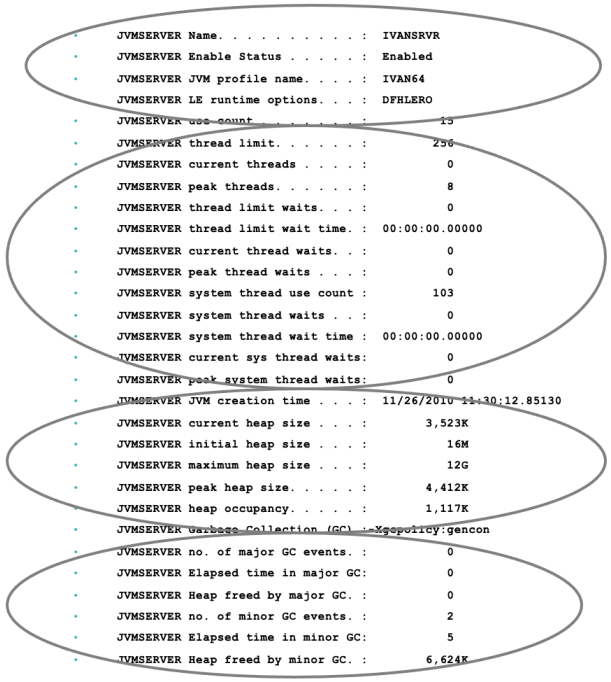


- Queue browse transaction
  - Browse temporary storage queues
  - Copy data from transient data queue into TS queue
  - Copy data from TS queue to TD queue

CICS Trace

- CICS Tracing for the SJ and AP components
  - 0, 1, and 2 trace levels
- Traces for setting up and managing JVM servers, installing and running OSGi applications, making JCICS calls.
- Use CETR to activate or STNTRxx SIT parm
- Trace output from CICS internal Java code written to zFS file controlled by JVMTRACE in JVM Profile
  - Written to &WORK_DIR/&APPLID.&JMVServer.dfhjvmtrc if no value specified in JVMTRACE in JVM profile
- SJ traces JVM server startup and shutdown, plus the initialization of the OSGi framework
  - Level 0ff, 1, and 2 no longer cause writes to z/FS
  - Level 3 – information, warning, and errors to zFS
  - Level 4 – debug, information, warning, and errors to zFS trace file
- The AP traces JVM server threads, activities of system threads such as enable, disable, and discard of OSGi bundles.
  - Written to same zFS file as SJ tracing
  - Level off, 1, 2, 3, 4 – Same as SJ tracing

CICS Statistics



```
JVMSERVER Name. . . . . . . . . . . :  IVANSRVR
JVMSERVER Enable Status . . . . . :  Enabled
JVMSERVER JVM profile name. . . . :  IVAN64
JVMSERVER LE runtime options. . . :  DFHLERO
JVMSERVER use count . . . . . :          13
JVMSERVER thread limit. . . . . :         256
JVMSERVER current threads . . . . :           0
JVMSERVER peak threads. . . . . :           8
JVMSERVER thread limit waits. . . :           0
JVMSERVER thread limit wait time. :  00:00:00.00000
JVMSERVER current thread waits. . :           0
JVMSERVER peak thread waits . . . :           0
JVMSERVER system thread use count :         103
JVMSERVER system thread waits . . :           0
JVMSERVER system thread wait time :  00:00:00.00000
JVMSERVER current sys thread waits:           0
JVMSERVER peak system thread waits:           0
JVMSERVER JVM creation time . . . :  11/26/2010 11:30:12.85130
JVMSERVER current heap size . . . :       3,523K
JVMSERVER initial heap size . . . :          16M
JVMSERVER maximum heap size . . . :          12G
JVMSERVER peak heap size. . . . . :       4,412K
JVMSERVER heap occupancy. . . . . :       1,117K
JVMSERVER Garbage Collection (GC) :-Xgcpolicy:gencon
JVMSERVER no. of major GC events. :           0
JVMSERVER Elapsed time in major GC:           0
JVMSERVER Heap freed by major GC. :           0
JVMSERVER no. of minor GC events. :           2
JVMSERVER Elapsed time in minor GC:           5
JVMSERVER Heap freed by minor GC. :       6,624K
```

- JVM server
- Thread usage
- Heap size
- Garbage collection

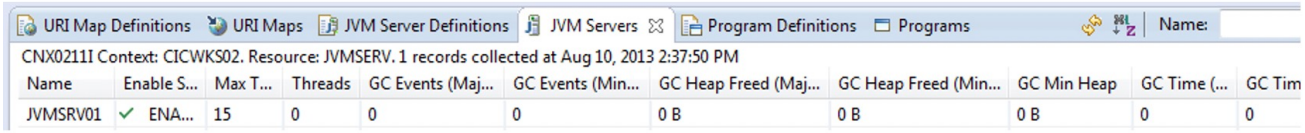
CICS Explorer

- System management tool for CICS
  - Based on z/OS Explorer and Eclipse
  - Connects to CICS TS for z/OS
  - Provides a way to manage one or more CICS systems
  - Provides a view of some CICSPlex SM
  - Provides a platform for the integration of current and future CICS Tools

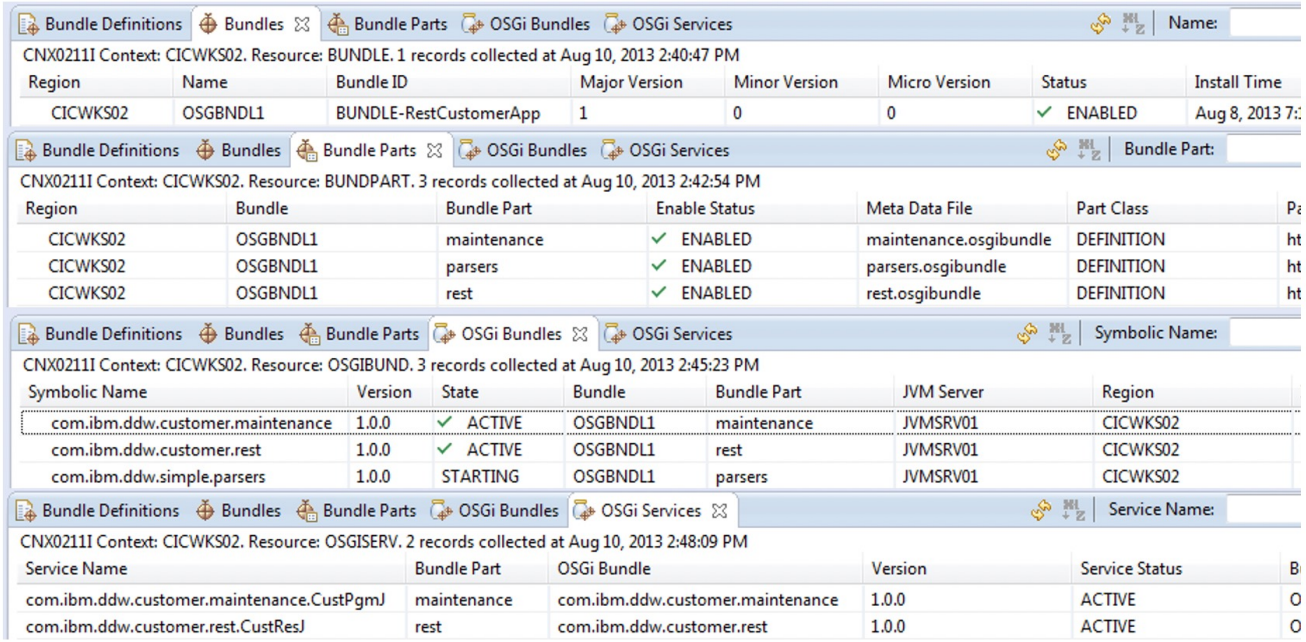
- Three features
  - CICS Explorer
    - Management of CICS resources and resource definitions.
  - IBM CICS SDK for Java
    - Support to develop Java applications with the JCICS classes
  - IBM CICS SDK for Java EE, Jakarta EE and Liberty
    - Support to develop web applications that can run in a Liberty profile server in CICS TS.

CICS Explorer – JVM view

- Information about the JVM server
  - Number of threads used
  - Number of major and minor garbage collections
  - JVM statistics



CICS Explorer – OSGi bundles and services



Troubleshooting
1 of 4

- **Your OSGi application**
  - Must have a ‘main’ method in your class for CICS to invoke it
    - public static void main(CommAreaHolder ca)
    - public static void main(String[ ] args)
  - OSGi project symbolic name and version in the .osgibundle file in the CICS BUNDLE project must match the details in the OSGi plug-in project manifest
  - Verify your configuration
    - For example, Import-Package: com.ibm.cics.server, com.ibm.db2.jcc
- Dependent bundles must be available before the OSGi bundle will go active
- CICS BUNDLE resource must point to the correct location where your bundle was uploaded
  - Spell the directory and file names correctly
  - If incorrect, will likely show message indicating missing manifest
- CICS region must have permission to read BUNDLE directory and files

Troubleshooting
2 of 4

- **ABEND AJ04**
  - CICS cannot find your program's main method or an unhandled exception has been thrown
  - Or, an Uncaught Exception passed back to CICS from application

- Caused by
  - The name of main class or OSGi service alias is incorrect on the PROGRAM definition
  - The CICS BUNDLE that refers to the OSGi bundle is not enabled
  - The OSGi bundle manifest does not have the correct CICS-MainClass attribute
  - The CICS BUNDLE has been installed via the OSGI_BUNDLES parameter
    - OSGI_BUNDLES are for middleware, not application bundles

Troubleshooting
3 of 4

- **JVMPROFILE**
  - Errors in middleware bundle cause JVM server not to go active
    - Message in DFHJVMTRC indicating an error
    - MSGUSR just says JVM server didn't start
    - Verify the OSGi bundles in OSGI_BUNDLES parameters of the JVMProfile file really are OSGi bundles

- OSGI_FRAMEWORK_TIMEOUT
  - Default is 60 seconds
  - May be too short if several OSGi frameworks are initializing at the same time
  - If the time limit is exceeded during startup, the JVM server will shut down
  - Consider mounting as read-only

Troubleshooting
4 of 4

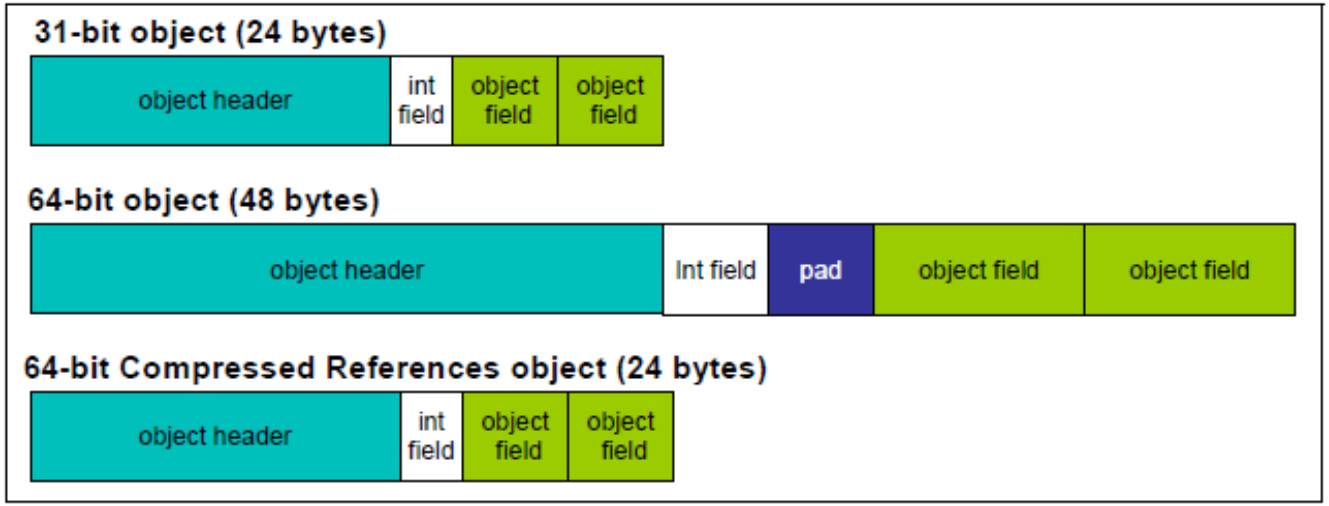
- **JVMPROFILE**
  - PRINT_JVM_OPTIONS=YES
    - Prints all the JVM startup options
    - Prints to SYSPRINT

Tuning the Java environment

- Execution configuration
  - JVM compressed references
  - Shared class cache
  - JIT compiler
  - AOT compilation
- JVM storage
  - Heap size
  - Garbage collection

JVM Compressed References

- **-Xcompressedrefs**
  - Decrease the size of Java objects
  - Make better use of available space in the Java heap
  - 31-bit memory usage increases
  - Enabled by default when -Xmx ≤ 57 GB.
  - Disable with -Xnocompressedrefs



Class data sharing
aka Shared Class Cache

- Improves start up performance
- Reduces memory footprint
- Automatically creates shared memory
  - Stores and shares the classes in memory between JVMs
- Updated dynamically when an application loads new classes
- Persists across JVM restarts
  - Does not survive an IPL
  - Shared class cache, by default is disabled
  - Add –Xshareclasses to JVM profile to activate
- The JIT compiler dynamically compiles certain methods into AOT code at runtime. Subsequent VMs that attach to the cache can take advantage of the compiled code to start faster.
- The JIT compiler stores profiling data and various compilation *hints* into the shared classes cache. This data enables subsequent VMs that attach to the cache to start faster, run faster, or both

Managing shared class cache

java -Xshareclasses:<command>

- Commands:
  - destroy
  - destroyAll
  - listAllCaches
  - printStats
  - printAllStats
  - reset

```
COMPTON:/u/compton: >java -Xshareclasses:listAllCaches

Listing all caches in cacheDir /tmp/javasharedresources/

Cache name                    level          cache-type       feature          layer
  OS shmid        OS semid        last detach time

Compatible shared caches
cicstg920CICSGRP          Java8 64-bit   non-persistent   cr               0
  8198            4110            In use
cicstsCICSGRP             Java8 64-bit   non-persistent   cr               0
  8199            4111            In use
```

Shared class cache tutorial

- https://developer.ibm.com/tutorials/j-class-sharing-openj9/

JIT (Just In Time Compiler)

- Improves the performance of Java applications
- Compiles bytecodes to native machine code at run time
- Enabled by default
- JIT compilation initiated based on number of times a method has been used
  - Default optimization level is warm
- After a method has been compiled, the JVM calls the compiled code of that method directly instead of interpreting it

- 5 optimization levels
  - cold
  - warm
  - hot
  - veryhot
  - scorching
- JIT compiler may recompile a method at a higher optimization level
  - Sampling thread

AOT (Ahead of Time) Compilation

- Generates native code dynamically while an application runs
- Caches any generated AOT code in the shared data cache
- Enabled by default
  - Only active when shared classes are enabled
- AOT-generated code does not perform as well as JIT-generated code
- AOT compiled code is also subject to JIT compilation
- AOT compilations are performed at the cold optimization level

Java Heaps

- JVM Heap size controlled by parameters in JVMPROFILE
  - -Xms, sets the initial size of the heap
  - -Xmx, sets the maximum size of the heap
- Additional memory is committed as the heap expands
- Control of free storage
  - -Xminf & -Xmaxf
  - Garbage collection

- -Xms
  - Should be big enough to avoid allocation failures from the time the application starts to the time it becomes ready.
  - Rule of thumb: shouldn't be any bigger than needed for previous bullet
- -Xmx
  - Normal load: free heap after GC should be greater than minf (default is 40%)
  - There should be no OutOfMemory errors
  - Heaviest load: if free heap after GC is greater than maxf (Default is 70%), heap size is too small

Garbage Collection

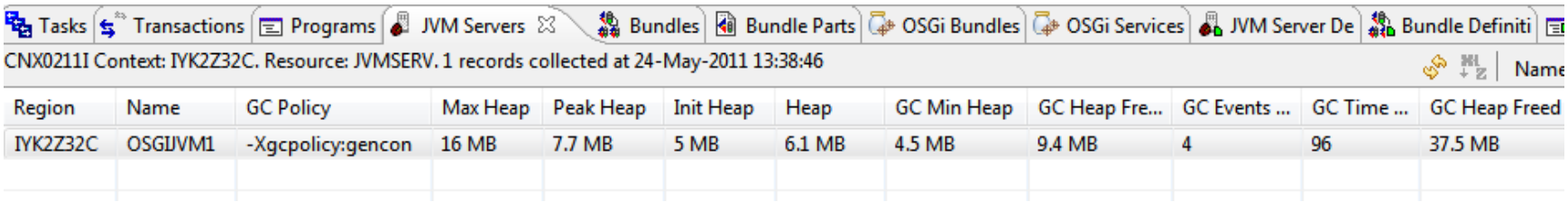
- Objects in the Java heap that are no longer required must be reclaimed
- Prevent applications running out of memory
- Partial and global GC cycles
  - Concurrent or StopTheWorld processes
  - Depending on GC policy
- -verbose:gc
  - Collect GC statistics
  - Output to stderr or file
  - -Xverbosegclog:<file>

- Garbage Collection – 3 steps
  - Mark: Find all active objects in the system
  - Sweep: Remove inactive objects
  - Compact: Reduce fragmentation within the free list
- Multiple GC policies – optimized for different scenarios
  - -Xgcpolicy:optthruput – for batch type apps; large systems with allocation contention
  - -Xgcpolicy:optavgpause – for apps with responsiveness criteria
  - -Xgcpolicy:gencon – highly transactional workloads
  - -Xgcpolicy:balanced – optimized for large heaps
- Pause-less GC – only available with gencon policy
  - -Xgc:concurrentScavenge

CICS Statistics

- Initial heap
  - Initial Java heap size, for example -Xms64M
- Current heap
  - Current size of heap (transient)
- Peak heap (GC Min heap)
  - Largest heap recorded (before GC)
- Occupancy
  - Heap available after last GC

- Max Heap
  - Maximum Java heap size, defaults to 256M
  - Storage reserved in MEMLIM
- Major GC and Minor GC events
  - Number of GC collection events
  - Total amount of heap freed
  - Elapsed time – Time JVM was paused during GC



Taking advantage of Z specialty engines

- TCBs executing Java code in a JVM are eligible for zAAP dispatching
- Newest Z processors only have zIIP engines
- zAAP on zIIP option allows use of zIIP engines for zAAP workloads
- Check your zAAP/zIIP usage
- Not all work run by a JVM is zAAP eligible
  - e.g. Native methods using Java Native Interface (JNI) are not eligible
  - Commonly used in CICS to provide CICS services from a Java app

- RMF can be used to report on expected and actual zIIP/zAAP usage
  - PROJECTCPU=YES in IEAOPTxx, enables RMF to report on zAAP/zIIP eligible work that runs on standard processor
- IIPHONORPRIORITY=NO forces all zIIP-eligiable work to run only on zIIP processors
- CMF fields – CPUTONCP / OFFLCPUT
- CMF fields – zAAP usage

Quick tuning hints

- Analyze your Java app to ensure it is running efficiently and does not generate too much garbage
  - JVM Health Center, verbosegc, CICS Explorer
- Tune JVM server. Use CICS statistics and IBM Health Center to analyze the storage settings, garbage collection, task waits, etc
- Optimize JIT. Takes CPU cycles. JIT'd code is lost when server restarted. Start JVM servers infrequently
- Check if best to enable class cache. Can improve JIT process: JIT can store profiling info and partially optimized code (AOT byte code) in shared cache
- Large heap size may minimize processor usage but may cause erratic response time (fewer garbage collections, but probably more time spent in garbage collection when it happens)
- Too small of heap may result in too much time spent in garbage collection
- Recommend gencon garbage collection policy

Summary

- Debugging
  - Configuration parameters
  - Tools
  - Troubleshooting
- Tuning
  - Configuration parameters
  - Memory usage
  - Statistics and monitoring into

Read more

- Java diagnostic data and tooling
  - https://www.ibm.com/docs/en/sdk-java-technology/8?topic=diagnostics-overview
- Getting started with CICS Explorer – including video tutorials
  - https://www.ibm.com/docs/en/cics-explorer/5.5.0?topic=getting-started-cics-explorer
- Monitoring CICS JVM servers with IBM Health Center
  - https://community.ibm.com/community/user/ibmz-and-linuxone/blogs/philip-wakelin1/2020/08/10/installing-ibm-health-center-into-cics-explorer