MQ Performance Benchmarking

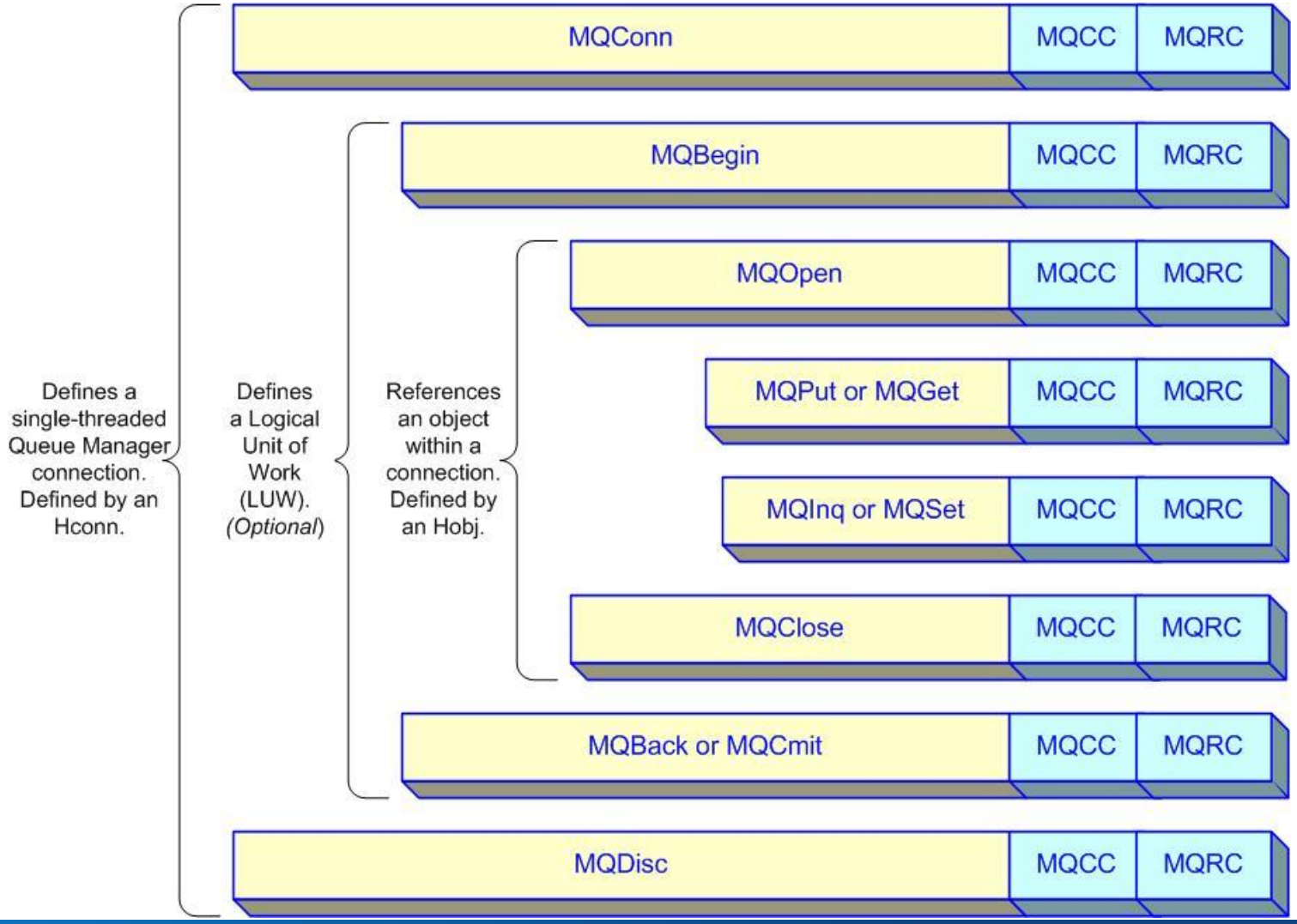
Methodology & Tools

Presentation Contents

- **Background Information**
  - MQ Programming Interface (MQI) & Programming
  - MQ Internal Processing
- **Benchmarking Approach**
  - Benchmark Testing Goals
  - Benchmark Limitations
- **Available Tools (Free)**
  - "**q**" Program (formerly SupportPac MA01 by Paul Clarke)
  - IBM SupportPac MH04 ("**xmqqstat**" Queue Statistics)
  - IBM **PerfHarness** & IBM **PerfRating**
  - IBM **amqsrua** & **amqsmon** commands
  - UNIX "**top**" command & Microsoft Windows **PerfMon** tool
- **Testing Automation**
  - "**JUnit**" & "**JMeter**" Test Frameworks
- **Summary**

Background Information

MQI (Message Queue Interface)



MQ Programming - 1

- **MQI Language Support**
  - C, COBOL, PL/I , RPG (MQI)
  - Java (MQ Classes for JMS, MQ Classes for Java)
  - C++, .Net (XMS)

- **MQ API Programming**
  - The MQConn call is the most expensive MQI action.
  - MQGet & MQPut calls should be performed within a loop.
    - The MQConn/MQDisc& MQOpen/MQClose are outside of the loop
    - MQPut1 can be used for exception calls, not routine processing.

- **Message Persistence**
  - All persistent messages are written to the log.
  - Both persistent & non-persistent messages **may** be written to disk for the queue.
  - Messages are processed from memory whenever possible.

- **MQ Thread (MQConn) Processing**
  - The MQConn handle is held by a single thread.
  - Within a MQConn handle, calls to MQ are single threaded.
  - Thus, MQ calls are synchronous and blocked within a handle.

MQ Programming - 2

- **Different language APIs will have different performance characteristics**
- **Different API calls have different costs**
  - "Connect "is the most expensive call (in terms of latency)
  - "Get" calls have things to consider
    - Message Filters – response times degrade as queue depths increase
    - Lock Contention – response time degrades as number of "Readers" increases
- **API Calls are one of the MQ Bottlenecks!**
  - Maximum number of API calls / second based upon the API call path length
  - Application Architects and WMQ Administrators should know this number!
  - Easy to determine, use the "Q" program (Thank you Paul Clarke)
    - crtmqm *TempQmgr*
    - strmqm *TempQmgr*
    - echo "define qlocal('*TempQueue*')" | runmqsc *TempQmgr*
    - date
    - echo "#!*1000000*/*1024*" | /…path…/q -m *TempQmgr* -ap –p1 -O *TempQueue*
    - date
    - The preceding commands write 1,000,000 messages of 1K size

MQ Internal Processing

- **API Calls**
  - Each Connection Handle (HCON) is associated with a single thread!
  - API calls through the same Connection Handle are single-threaded!
  - API Path Length is approximately 1-2 ms, resulting in < 1,000 MQ calls per second.
- **Persistent messages are written to the log**
  - Message cannot be released to the application until the log write completes.
  - Non-persistent messages are roughly 10 times faster than persistent messages!
- **WMQ channel protocol is a blocking protocol**
  - MCA waits for an acknowledgement after each block is transmitted.
    - Impacted by Batch Size (BATCHSZ) parameter.
    - Impacted by the Batch Interval (BATCHINT) parameter.
  - MCA agents on each Queue Manager must update the log for persistent messages.
  - Multiple channels between Queue Manager pairs will significantly increase throughput.
  - Message delivery sequence is generally “First In First Out” (FIFO)
    - Separating large from small messages can yield significant QoS improvements

Benchmarking Approach

Benchmarking Context

- **Performance Measurements**
  - Capacity; e.g. Transactions per Second (TPS).
  - Latency; e.g. Seconds per Transaction.
- **Benchmarking Targets**
  - Infrastructure capacity (maximum)
  - Application capacity (maximum)
- **Benchmarking Measurement Granularity**
  - Single thread.
  - Multiple threads per container (e.g. Integration Server).
  - Multiple servers.

Benchmarking Limitations

- **Benchmarking Challenges**
  - Infrastructure easier to benchmark
    - Test Tools exist & Stub programs easily constructed
    - Components can be tested in isolation
  - Applications more difficult to benchmark
    - Integrated End-to-end testing required
    - Database & other external software dependencies
    - Test data required
- **Data Limitations**
  - Test data normally based upon functional testing requirements
  - Test data often not able to detect:
    - Database lock management issues
    - Database deadlocks issues

Benchmarking Goals

- **Current application capacity determination**
  - Planning for peak load readiness
  - Planning for infrastructure capacity increases
    - Additional servers and/or licenses
- **Identify vertical scaling opportunities**
  - Upgraded software (e.g. newer release of MQ)
  - More server resources
    - More/Faster CPU, More memory, Faster disk
- **Identify horizontal scaling capabilities**
  - More threads per container (e.g. Message Flow).
  - More containers (e.g. Integration Servers).
  - Increased Application isolation (e.g. Integration Servers/Nodes)
  - More servers.

Benchmarking Approach

- **Benchmark measurements**
  - Reader & Writer performance
  - TPS & Latency
- **Benchmark Infrastructure (per thread)**
- **Benchmark Infrastructure scaling (multiple threads)**
  - Identify performance bottlenecks (Readers > Writers)
    - (Queue locking)
  - Identify performance bottlenecks (Writers > Readers)
    - (Queue search)
- **Benchmark Applications**
  - End-to-End benchmarks
    - Single-thread performance
    - Multiple-thread performance

MQ Performance Benchmarking

Tools – “Q”

“Q” Program

- **Tool Overview**
  - Program reads from a ”source” (STDIN) and writes to a “target” (STDOUT)
  - “Source” may be keyboard, file, Queue, or Subscription
  - “Target” may be screen, file, Queue, or Topic
  - Multiple test data generation and behavior options available

- **Tool Highlights**
  - Simple to use
  - Documented through a short “Readme.txt” file.
  - Supported on many platforms, but may require compilation first.
  - Capable of generating testing loads
  - Large number of command parameters available for specialized uses
    - Many MQI parameter options supported

- **Tool History**
  - Developed by Paul Clarke of the Hursley Laboratory
  - Developed by Paul as a Hursley testing tool
  - Originally released for public use as a SupportPac (MA01) – August 1995
  - Currently available as Open Source through GitHub
    - https://github.com/ibm-messaging/mq-q-qload

“Q” Program Invocation

- **Queue Manager Connection (-l for “library””)**
  - Default behavior is to use “Server Binding” mode (local Queue Manager) ➔ **-l mqm**
  - TCP/IP “Client Binding” mode is also support ➔ **-l mqic**
  - Identify Queue Manager (if not default) ➔ **-m** *queueManagerNameHere*
- **Program Input and Output**
  - Program reads from a ”source” (STDIN) and writes to a “target” (STDOUT)
    - Standard redirection operators (“<“, “>”, “>>”) for file I/O
  - Program parameters supported for specific I/O sources
    - “Source” may be keyboard, file, Queue, or Subscription
    - “Target” may be screen, file, Queue, or Topic
  - Standard pipe (“|”) processing supported
  - “**-f** *fileNameHere*” ➔ Read input records from the named File
  - “**-i** *queueNameHere*” ➔ Browse input messages from the named Queue
  - “**-I** *queueNameHere*” ➔ Read (destructive) input messages from the named Queue
  - “**-o** *queueNameHere*” ➔ Send output to the named Queue (Bind not Fixed)
  - “**-O** *queueNameHere*” ➔ Send output to the named Queue (Bind on Open)
  - “**-S** *subscribeOptions*” ➔ Subscription options (see “ReadMe”)
  - “**-T** *publishOptions*” ➔ Topic options (see “ReadMe”)

“Q” Program Invocation- continued

- **Queue Parameter Name Format (Supported for multiple parameters)**
  - One part format (Queue name only)
  - Two part format (Queue Manager Name & Queue Name)
  - Multiple “part” separators supported:
    - “**/**” , “****” , “**#**” , “**,**”

- **Key Performance Testing parameters**
  - “**-an**” ➔ MQPut non-persistent messages
  - “**-ap**” ➔ MQPut persistent messages
  - “**-p**” ➔ Number of messages between Commit points
  - “**-L**” ➔ Maximum number of messages to process
  - “**-r** *queueNameHere*” ➔ “Reply To” Queue Name
  - “**-r+** *queueNameHere*” ➔ Read message from “Reply” queue before next MQPut
  - “**-t**” ➔ Print timing information about each API call
  - “**-w**” ➔ Number of seconds to wait for a message to arrive (MQGet parameter)
  - “**-W**” ➔ Number of milliseconds to wait before MQGet call (simulates processing time)
  - “**-y** *tenthsOfSeconds*” ➔ Set message expiry interval (1/10 of second increments)
  - “**-1**” ➔ Use MQPut1 (MQOpen-MQPut-MQClose) instead of MQPut

“Q” Program Invocation- - continued

- **Test Data Input**
  - Input messages contained in an existing Queue
    - Use the “**-i**” parameter to save the messages!
  - Input messages contained in an existing File (records)
    - “**-f** *fileNameHere*” ➔ Read input records from the named File
    - “**<** *fileNameHere*” ➔ Redirect input to the named File

- **Test Data Generation**
  - Input data contains test data generation instructions!
  - Format is “#[**!**][**c**][*number*/[*size*/[*delay*/[*commit*]]]]” *messageDataHere*
    - “#” ➔ Indicates “test data generation command”
    - “**!**” ➔ Do not include this instruction in the message (Optional)
    - “**c**” ➔ Checksum messages (Optional)
    - “*number*” ➔ Number of messages to generate (Optional)
    - */size*” ➔ Message size in Bytes (Optional).
      - Data padded with low-values (x00)
    - */delay*” ➔ Delay (in seconds) between each put.
    - */commit*” ➔ Commit interval (in messages) between each MQCmit.

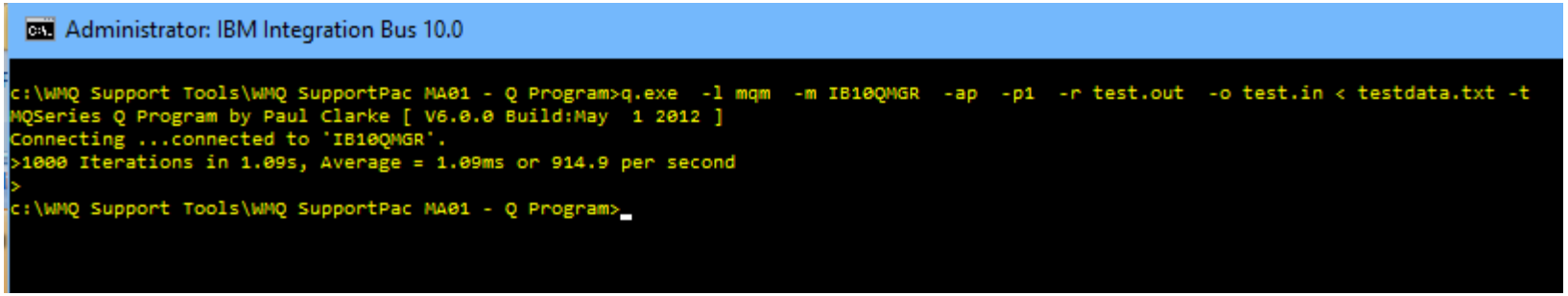
“Q” Program Examples

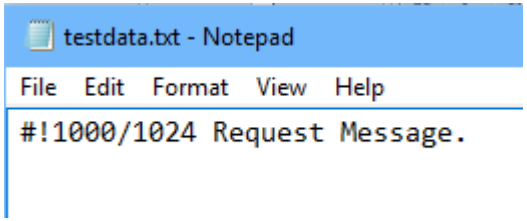
- **q -l mqm -m** *qmgrName* **-I** *requestQ* **-o** *replyQ* **-w** *60* **-W** *20* **-t**
  - **Read** input from the named queue (*requestQ*)
  - Write output to the named queue (*replyQ*)
  - Wait 60 seconds for message to arrive before terminating (e.g. allow testing to start)
  - Simulate 20 milliseconds of message processing time (e.g. 50 messages/sec)
- **q -l mqm -m** *qmgrName* **-ap -p1 –r** *replyToQ* **-o** *requestQ* **-t <** *fileName*
  - Read input from the named file (*fileName*)
    - File data is “#!100000/1040 Request Message.”
  - **Write** output messages to the named queue (*requestQ*)
  - 100,000 persistent 1k length messages (“Test Message…”) written to queue
  - Commit on every message
  - “Reply To” Queue name specified (*replyToQ*)
- **q -l mqm -m** *qmgrName* **-i** *inputQ* **-o** *outputQ1* **-o** *outputQ2*
  - Read input from the named Queue (*inputQ*)
  - **Copy** input messages to output (two) queues!
    - Output queue #1 (*outputQ1*)
    - Output queue #2 (*outputQ2*)

Using the “Q” program

- **Simulating “Writer” Applications**
  - Simulate application **(single thread**) “Writing” messages.
  - Multiple instances may be spawned, if required.
  - The “**-r+** *queueName*“ parameter may be used to slow down the writer.
    - Writer will read the reply message before putting the next message.
    - The reply message may be delayed (“**-W 20**”) to slow down the writer.
- **Simulating “Reader” Applications**
  - Simulate application **(single thread**) “Reading” messages.
  - Multiple instances may be spawned, if required.
  - The “**-W** *20*“ parameter may be used to slow down the reader.
- **Platform Benchmarking**
  - Use without delays to benchmark platform capability (single thread)
  - Test with increasing numbers of *Readers* and *Writers* to benchmark horizontal scaling
  - Combine with platform measurements (CPU, Memory, Network)
- **Application Benchmarking**
  - Use with Application software to simulate external systems

“Q” Program Output





MQ Performance Benchmarking

Tools – “xmqqstat” (MH04)

xmqqstat (MH04) Program

- **Tool Overview**
  - Queue Statistics monitoring tool (written in Java)
  - Category 2 SupportPac (“As Is” – no official IBM Support)
  - Authored by Oliver Fisse of IBM Software Group (ISSW) – November 2010
  - Some minor configuration is required.

- **Tool Highlights**
  - Simple to use
  - Documented through a short “Readme.txt” file.
  - Each instance of the program monitors a single queue.
  - Companion program may be used to monitor multiple queues.
  - Activity and queue status are reported at specified intervals.

- **Key Reported Data**
  - Time → Current Time
  - OIC/OOC → Input Count (e.g. reading threads) / Output Count (e.g. writing threads)
  - MEC/MDC → Enqueue count (messages written) / Dequeue count (messages read)
  - UNC → Uncommited messages (at end of monitoring interval)
  - QCD → Current Queue Depth (at end of monitoring interval)
  - MxQD → Maximum Queue Depth (during monitoring interval)

xmqqstat – Additional Features

- **Extended Data ("-e" parameter)**
  - PQF → Percentage Queue Full (during monitoring interval)
  - TQF → Time to Queue Full (at present enqueue rate)
  - TQE → Time to Queue Empty (at present dequeue rate)
  - The following extended data requires Queue Monitoring (MonQ) to be turned on
    - QOM → Queue Oldest Message (Age of oldest message in queue)
    - OQTS → Output Queue Time (Short) – Average time messages spent in queue
    - OQTL → Output Queue Time (Long) – Average time messages spent in queue
- **Application Handle Information Reported (-h option)**
  - Data displayed as per DIS QS(*queue*) TYPE(HANDLE)
- **Key Parameters**
  - -d Duration to collect statistics (in Seconds)
  - -eExtended statistics (some require MONQ enabled)
  - -h Display information about Application Handles
  - -i Statistics collection interval (in Seconds)
  - -m Queue Manager name
  - -q Queue name
  - -sSuppress display if no activity during interval
  - -t Display time

xmqqstat Examples - 1

- **Monitor local Queue Manager / Queue for 5 minutes; summarize each minute:**
  - xmqqstat -m *Qmgr* -q *Queue* -d 300 -i 60 -e -s -t
    - Connect to local Queue Manager using Server bindings
    - Collect statistics on *Queue* (-q) in *Qmgr* (-m)
    - **Collect statistics for 5 minutes (300 seconds) (-d)**
    - **Report statistics every minute (60 seconds) (-i)**
    - **Collect extended statistics (-e)**
    - Don't report an interval if there is no activity (-s)
    - Display the time (-t)
- **Monitor remote Queue Manager / Queue ... :**
  - xmqqstat -c *SYSTEM.DEF.SVRCONN* -x *hostname(1414)* -m *Qmgr* -q *Queue* …
    - Connect to remote Queue Manager using Client bindings
    - Collect statistics on *Queue* (-q) in *Qmgr* (-m)
    - **Connect to server** *hostname* **using port** *1414* **(-x)**
    - **Use SYSTEM.DEF.SVRCONN channel (-c)**

xmqqstat Examples - 2

- **Indefinitely Monitor local Queue Manager / Queue Application connections:**
  - xmqqstat -m *Qmgr* -q *Queue* -i 3600 -h -e -s -t
    - Connect to local Queue Manager using Server bindings
    - Collect statistics on *Queue* (-q) in *Qmgr* (-m)
    - **Collect statistics indefinitely** (no **-d** parameter)
    - **Report statistics every hour(3600 seconds) (-i)**
    - **Display Handle information (-h)**
    - Collect extended statistics (-e)
    - Don't report an interval if there is no activity (-s)
    - Display the time (-t)
- **Note on tool execution:**
  - **PCF commands used to reset Queue statistics for Enqueue/Dequeue calculations.**
    - PCF command "Reset Queue Statistics".
- **Note on execution duration:**
  - If Duration (-d) parameter is not specified, then duration is unlimited.
  - The Ctrl-C command can be used to stop execution.

xmqqstat Program Output

```
C:\MQ>xmqqstat -m TEST -q TEST -i 1 -s -t -h
Xmqqstat v1.1 - Developed by Oliver Fisse (IBM)

Connected to queue manager 'TEST'

PLATFORM(WINDOWS NT) LEVEL(701) CCSID(437)
MAXHANDS(256) MAXMSGL(4194304) MAXPRTY(9) MAXUMSGS(250000) MONQ(HIGH)

Processing LOCAL queue 'TEST'

DESC()
CRDATE(2010-09-09) CRTIME(15.29.02) ALTDATE(2010-10-03) ALTTIME(09.14.32)
CLUSTER() CLUSNL() DEFBIND(OPEN)
BOTHRESH(0) BOQNAME()
MONQ(QMGR) USAGE(NORMAL) NOTRIGGER


Dumping 1 handle(s)...

   PID    TID    AT   CHL/APPL TAG/CONN     USER ID                    B INP I O S
------------------------------------------------------------------------------
    7968      0 USER                        Administrator@IBM-6AE723B N  NO N Y N
                ere MQ\java\jre\bin\java.exe

   Time         MxML       MxQD G P    OIC    OUC      MDC      MEC    UNC      CQD
-------------------------------------------------------------------------------
 10:19:09    4194304    2500000 E E      0      1        0     6300      0     6300
 10:19:10    4194304    2500000 E E      0      1        0      350      0     6650
 10:19:11    4194304    2500000 E E      0      1        0        0      0     6650
 10:19:12    4194304    2500000 E E      0      1        0      350      0     7000
 10:19:14    4194304    2500000 E E      1      1     7000        0      0        0
 10:19:15    4194304    2500000 E E      1      1      350      350      0        0
 10:19:16    4194304    2500000 E E      1      1        0        0      0        0
 10:19:17    4194304    2500000 E E      1      1      350      350      0        0
 10:19:18    4194304    2500000 E E      1      1        0        0      0        0
 10:19:19    4194304    2500000 E E      1      1      350      350      0        0
 10:19:20    4194304    2500000 E E      1      1        0        0      0        0
 10:19:21    4194304    2500000 E E      1      1      350      350      0        0
 10:19:22    4194304    2500000 E E      1      1        0        0      0        0
 10:19:23    4194304    2500000 E E      1      1      303      316      0       16
 10:19:24    4194304    2500000 E E      1      1       47       34      0        0
 10:19:25    4194304    2500000 E E      1      1       18       62      0       40

Control-C caught. Shutting down...

Disconnected from queue manager 'TEST'
Xmqqstat v1.1 ended.
```

MQ Performance Benchmarking

Tools – “PefHarness”

“PerfHarness” Program

- **Tool Overview (v1.2)**
  - Tool built in Java
  - Tests MQ, JMS (MQ, WMB, JNDI), TCP/IP, HTTP, REST, & SOAP transport protocols
  - Significant configuration may be required.
- **Tool Highlights**
  - Large number of built-in test (“Test Classes”) supported
  - Powerful testing capabilities
  - Supports testing with multiple threads
  - Supports “throttled” operations (limiting messages/second)
- **Tool History**
  - Developed by Marc Carter as an internal IBM tool
  - Used by IBM to develop the WMB/IIB Performance Report SupportPacs
  - Previously available through IBM AlphaWorks & developerWorks
  - Currently available as Open Source through GitHub
    - https://github.com/ot4i/perf-harness

PerfHarness Installation - 1

- **Download and install Eclipse IDE for Java SE (Oxygen)**
  - Open Source software from Eclipse.org
    - http://www.eclipse.org/downloads/packages/eclipse-ide-java-ee-developers/oxygenr
- **Download and Install latest Java release (Java SE 9) into Eclipse**
  - Open Source software from Oracle
  - Download the Java Development Kit (JDK), not the Java Runtime Environment (JRE)!
    - http://www.oracle.com/technetwork/java/javase/downloads/jdk9-downloads-3848520.html
    - Add the Java 9 JRE (from the JDK) to Eclipse
    - Window → Preferences → Java → Installed JREs → Add → Standard VM
    - Specify location of JDK, not JRE!
- **Download and Import PerfHarness into Eclipse**
  - Open Source software (originally from IBM) through GitHub
    - https://github.com/ot4i/perf-harness
  - Import the PerfHarness projects into Eclipse

PerfHarness Installation - 2

- **Refer to the documentation on the PerfHarness GitHub Page**
- **Download and Import PerfHarness Prerequisite Jar Files**
  - Import prerequisite IBM MQ Jar files into Eclipse
  - Import AMQP Jar files into Eclipse (Only needed for AMQP protocol support)
    - https://developer.ibm.com/messaging/ibm-mq-light-downloads/
    - Download from Maven Central
  - Find and import ANT prerequisite Jar Files
    - https://sourceforge.net/projects/ant-contrib/files/ant-contrib/ant-contrib-1.0b2/
    - ant-contrib-1.0b2-bin.zip
- **Correct any Java Errors**
  - Java code errors
  - Build Path errors
- **Build PerfHarness Java Project**
  - Eclipse → PerfHarness → build_all.xml → (*Right Click*) Run As → 1 Ant Build
  - If successful, the PerfHarness.jar file will be created in the “build” folder

Eclipse Configuration

- **Follow the instructions on the GitHub PerfHarness page**
  - However, treat these instructions as guidelines
    - They do not reference MQ v9.x
    - They do not reference current names of required Jar files
    - They do not reference current versions of Java
  - The resulting project may contain “minor” errors
    - Jar file references
    - Java errors
  - These errors must be resolved before the PerfHarness Jar can be built!
- **Some familiarity with Eclipse and Java in Eclipse is essential!**
  - Java Project → Properties → Java Build Path
- **Others have already encountered these problems**
  - Google can help resolve many of the issues
  - IIB developers may have Eclipse & Java experience

PerfHarness – Test Preparation

- **PerfHarness does not run as an executable Jar!**
- **Setup Java Classpath**
  - Windows
    - set CLASSPATH=perfharness.jar;%CLASSPATH%
    - java JMSPerfHarness *-parameters*
    - java -cp "perfharness.jar;%CLASSPATH%" JMSPerfHarness *-parameters*
  - UNIX
    - export CLASSPATH=perfharness.jar:$CLASSPATH
    - java JMSPerfHarness *-parameters*
    - java -cp "perfharness.jar:$CLASSPATH" JMSPerfHarness *-parameters*
- **Design Test**
  - Writers
  - Readers
  - Request/Response
- **Determine PerfHarness "Test Class" parameter**
  - See "Notes"

PerfHarness – Test Designs

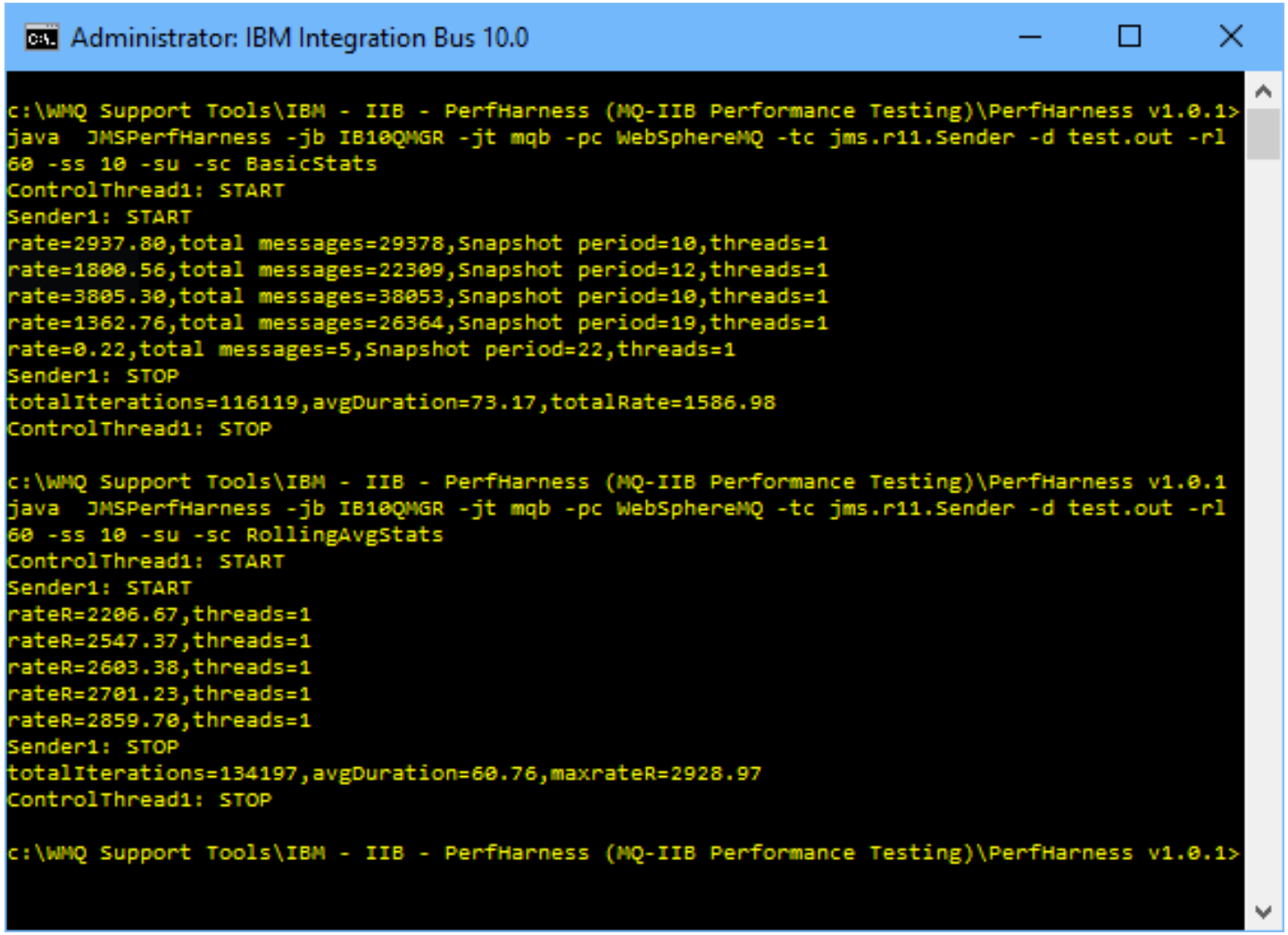
- Test "**Datagram**" message flows (MQMT_Datagram)
- Measure writing rate (messages/second) per thread.
- Test impact of additional "Writer" threads.
- Measure reading rate (messages/second) per thread.
- Test impact of additional "Reader" threads.

- Test "**Request**" / "**Reply**" message flows ("MQMT_REQUEST" / "MQMT_REPLY")
- Measure writing rate (messages/second) per thread.
- Test impact of additional "Writer" threads.
- Measure reading rate (messages/second) per thread.
- Test impact of additional "Reader" threads.

PerfHarness Example – MQ Requestor

- **java**
  - **-Xms**512M **-Xmx**512M (JVM parameters)
  - **-cp** "C:\path\perfharness.jar;%CLASSPATH%" **JMSPerfHarness**
  - **-tc** j**ms.r11.Requestor**
  - **-jh** *qmgrServer* **-jp** *qmgrPort* **-jc** *svrconn* (Client connection information)
  - **-jb** *queueMangerName*
  - **-jt** mqc (Use Client Bindings– "mqc")
  - **-iq** *inputQueue* **-oq** *outputQueue* (Queue names)
  - **-nt** 5 (Number of threads)
  - **-si** 100 (Start thread interval = 100 ms)
  - **-rl** 300 (Test duration in seconds)
  - **-ss** 60 (Reporting interval in seconds)
  - **-sc** BasicStats (Statistics module)
  - **-to** 120 (MQGet wait interval in seconds)
  - **-su** (Display final summary information)
  - **-mf** *inputFilePath&Name* (Input message from file)
  - **-mt** *messageText* (Input message text)
  - **-pc** WebSphereMQ (Use MQ as JMS provider)

PerfHarness Program Output



MQ Performance Benchmarking

Tools – “PerfRating”

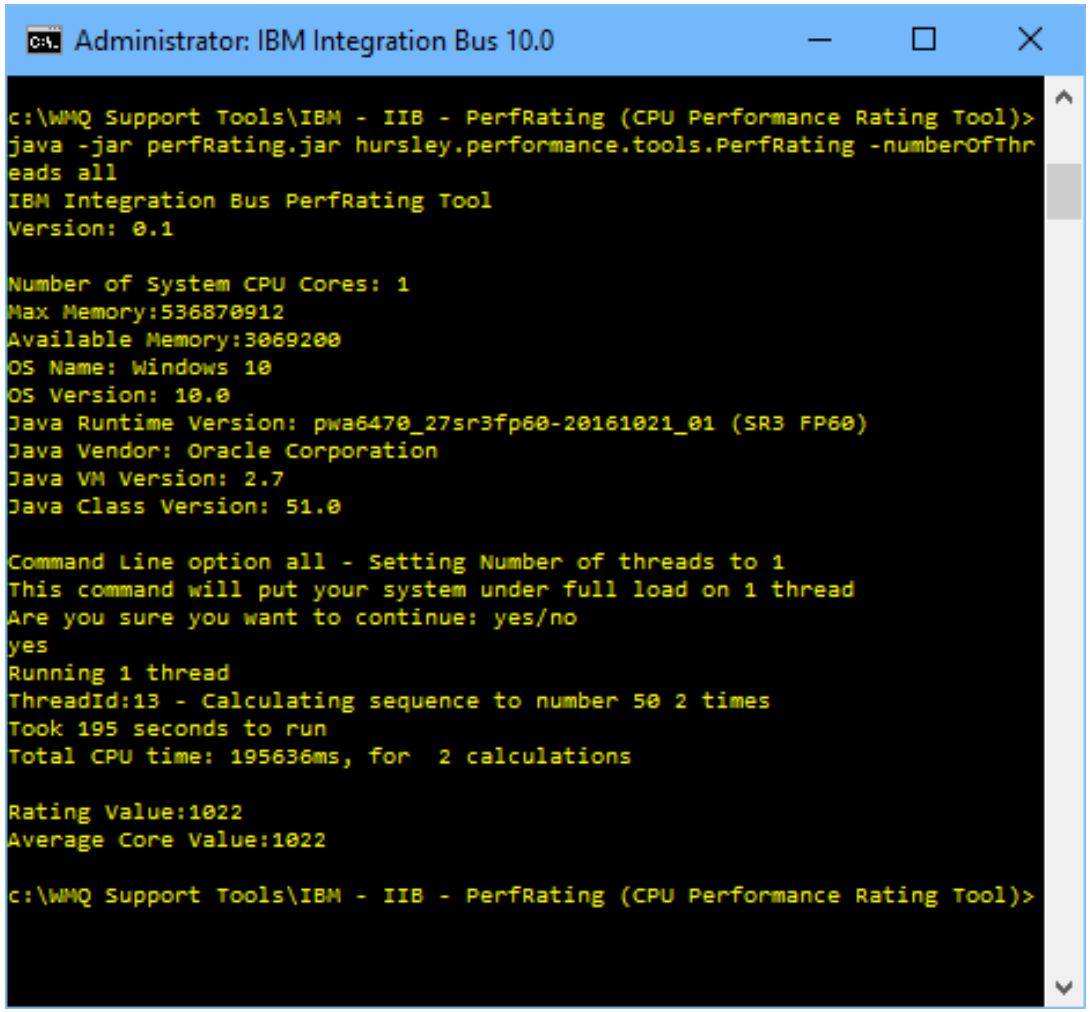
“PerfRating” Program

- **Tool Overview**
  - CPU Performance Rating tool
  - Requires Java 1.7 or above
  - Provides an abstract rating (“Core Value”) that allows server comparisons
- **Tool Highlights**
  - Extremely simple to use
  - Can be executed on each MQ and/or IIB server
  - Allows the CPU processing capability of each server to be benchmarked
    - Can detect server set-up issues if “identical” servers produce different results
    - Can provide a basis for comparing disparate servers
  - Especially useful for benchmarking Virtual Machine images
- **Tool History**
  - Developed by IBM Hursley to identify hardware performance issues
  - Currently available through developerWorks
    - https://developer.ibm.com/integration/blog/2015/11/21/perfrating-cpu-performance-rating-tool/

Using the “PerfRating” Program

- **Tool Invocation**
  - java -jar PerfRating.jar hursley.performance.tools.PerfRating -NumberOfThreads ***1***
    - Test a single thread (e.g. core)
  - java -jar PerfRating.jar hursley.performance.tools.PerfRating -NumberOfThreads ***all***
    - Test all available threads (e.g. cores)
- **Tool Results**
  - Server Description
    - Number of Cores
    - Amount of Memory
    - Operating System build
    - JRE Information
  - CPU Rating
    - Overall rating (“Value”)
    - Average Core rating (“Value”)

PerfRating Program Output



MQ Performance Benchmarking

Tools – “amqsrua”

“amqsrua” Program

- **Sample Program Supplied with MQ**
  - Displays performance information published by Queue Managers
  - Command displays information until stopped or “*Publication Count*” reached
- **Topic Tree**
  - **$SYS/MQ/INFO/QMGR**
- **Program Location**
  - UNIX: *installationPath*/samp/bin
  - Windows: *installationPath*\tools\c\Samples\Bin64
- **Command Parameters**
  - **-m** *qmgr* → Queue Manager name
  - **-c** *resourceClass* → Resource Class: “CPU”, “DISK”, “STATQ”, “STATMQI”
  - **-t** *typeName* → Resource Type
  - **-o** *objectName* → Resource Object
  - **-n** *pubCount* → Number of publications to report
  - **-d** *debugLevel* → Level of debugging information to report
  - **-h** → Display help information

amqsrua Class (“-c”) parameter values

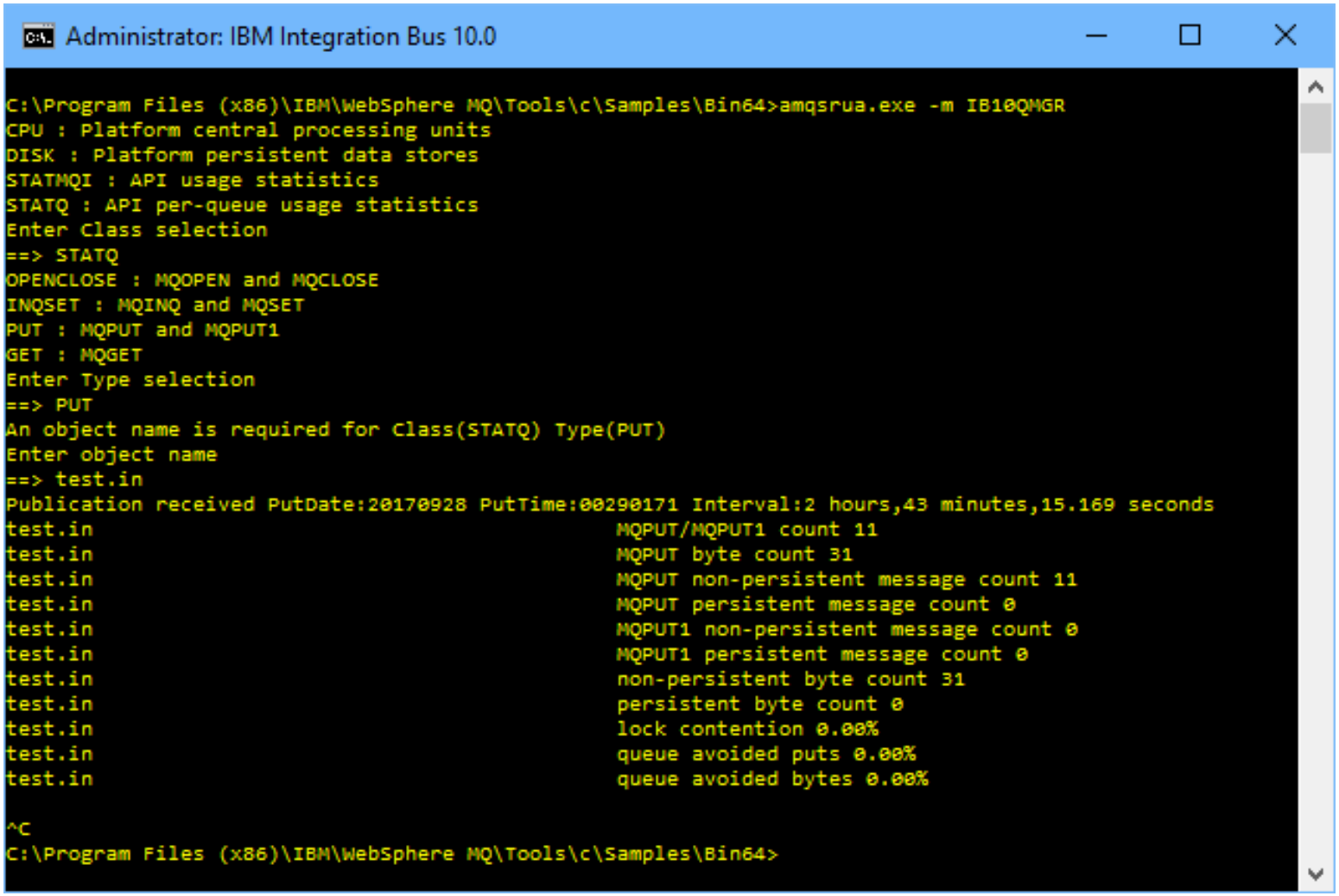
- **Topic Tree**
  - $SYS/MQ/INFO/QMGR
- **amqsrua Class (“-c”) parameter values**

CLASS	Class Description
CPU	Queue Manager CPU usage
DISK	Queue Manager disk usage
STATQ	MQI Calls per Queue
STATMQI	MQI Calls

- **Some Class/Type combinations require an “Object”**
  - “**-o**” parameter
  - e.g. Queue Name

Class(es)	Type	Type Description
CPU / DISK	SystemSummary	System wide CPU usage
	QMgrSummary	Queue Manager CPU usage
STATQ	OpenClose	MQOpen & MQClose statistics
	InqSet	MQInq & MQSet statistics
	Put	MQPut statistics
	Get	MQGet statistics
STATMQI	ConnDisc	MQConn & MQDisc statistics
	OpenClose	MQOpen & MQClose statistics
	InqSet	MQInq & MQSet statistics
	Put	MQPut statistics
	Get	MQGet statistics
	Syncpoint	MQBegin, MQCmit & MQBack statistics
	Publish	Message Publishing statistics
	Subscribe	Message Subscription statistics

“amqsrua” Sample Program Output



MQ Performance Benchmarking

Tools – “amqsmon”

“amqsmon” Program

- **Sample Program Supplied with MQ**
  - Displays Statistics & Accounting information generated by Queue Manager
    - SYSTEM.ADMIN.ACCOUNTING.QUEUE
    - SYSTEM.ADMIN.STATISTICS.QUEUE
  - Requires Queue Manager settings
    - ALTER QMGR **ACCTMQI** (ON) **ACCTQ** (ON) **ACCTINT** (1800)
    - ALTER QMGR **STATACLS** (ON) **STATMQI** (ON) **STATQ** (ON)
    - ALTER QMGR **STATCHL** (HIGH) **STATINT** (1800)
- **Program Location**
  - UNIX: *installationPath*/samp/bin
  - Windows: *installationPath*\tools\c\Samples\Bin64
- **Accounting & Statistics data introduced in v6.0**
  - “MQI” settings enable reporting and the connection (“MQConn”) level
  - While Accounting & Statistics messages have similar data, both can be useful
  - Can be run “before” (cleanup) and “after” (report) benchmark tests
  - Destructively reads messages unless the browse (“**-b**”) parameter is used!

MQ Accounting Messages

- Queue Manager settings
  - ALTER QMGR ACCTMQI (ON) ACCTQ (ON) ACCTINT(1800)
  - Data stored in: SYSTEM.ADMIN.ACCOUNTING.QUEUE
- Queue settings
  - ALTER QLOCAL … ACCTQ (ON)
- Record MQI data by connection (i.e. MQConn)
  - Messages written when connection is closed (i.e. MQDisc)
  - Messages also generated at Queue Manager ACCTINT intervals (30 min)
  - API call counts (e.g. MQGet & MQPut) & total byte counts provided
- Note:
  - Accounting information may also be specified on the MQCONNX call
  - ALTER QMGR ACCTCONO (ENABLED) must also be set!

MQ Statistics Messages

- Queue Manager settings
  - ALTER QMGR STATACLS (ON) STATCHL (HIGH) STATMQI (ON)
  - ALTER QMGR STATQ (ON) STATINT(1800)
  - Data stored in: SYSTEM.ADMIN.STATISTICS.QUEUE
  - "STATACLS" setting is for automatically defined Cluster Sender channels
- Queue & Channel settings
  - ALTER QLOCAL … STATQ (ON)
  - ALTER CHANNEL … STATCHL (HIGH)
- Records Queue Manager wide data
  - Messages generated at Queue Manager STATINT intervals (30 min)
  - Messages also generated at Queue Manager shut down
  - RESET QMGR TYPE (STATISTICS) ➔ Forces message write
    - Can be used at the end of a benchmark test to generate a Statistics message
    - Required to be able to immediately see statistics!
  - API call counts (e.g. MQGet & MQPut) & total byte counts provided

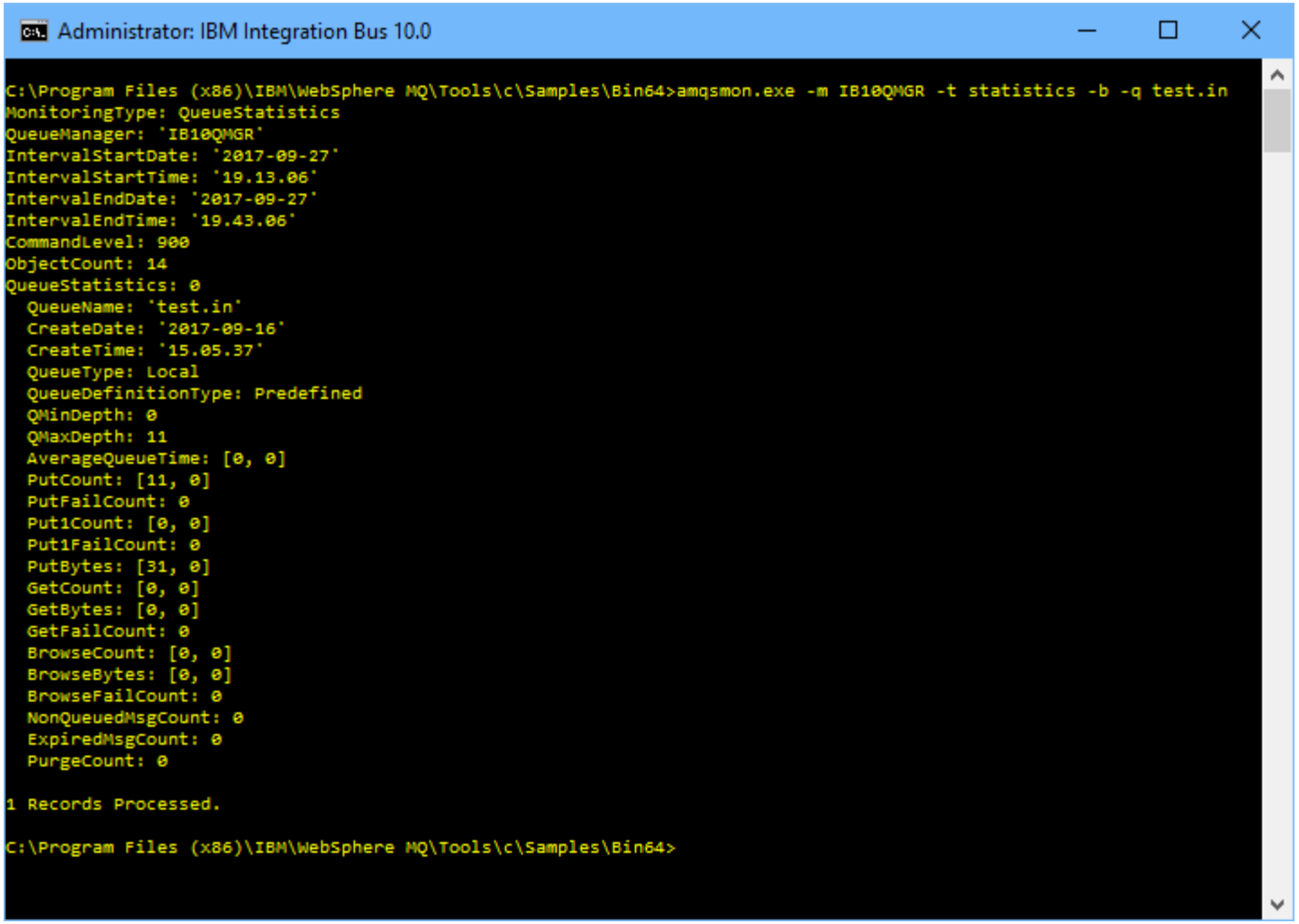
“amqsmon” Program Parameters

- **Command Parameters**
  - **-m** *qmgr* → Queue Manager name
  - **-t** *type* → “**statistics**” or “**accounting**”
  - **-s** *startTime* → Starting GMT reporting time (YYYY-MM-DD HH.MM.SS)
  - **-e** *endTime* → Ending GMT reporting time (YYYY-MM-DD HH.MM.SS)
  - **-l** *fieldList* → Comma separated list of fields to display
  - **-a** → Display MQI information
  - **-q** *queue* → Display Queue information (Queue name optional)
  - **-c** *channel* → Display Channel information (Channel name optional)
  - **-i** *connectionId* → Only display accounting Connection ID data (ID optional)
  - **-b** → Browse messages
  - **-d** *messages* → Maximum number of messages to process
  - **-w** *seconds* → MQGet wait interval (in seconds) for a message to arrive

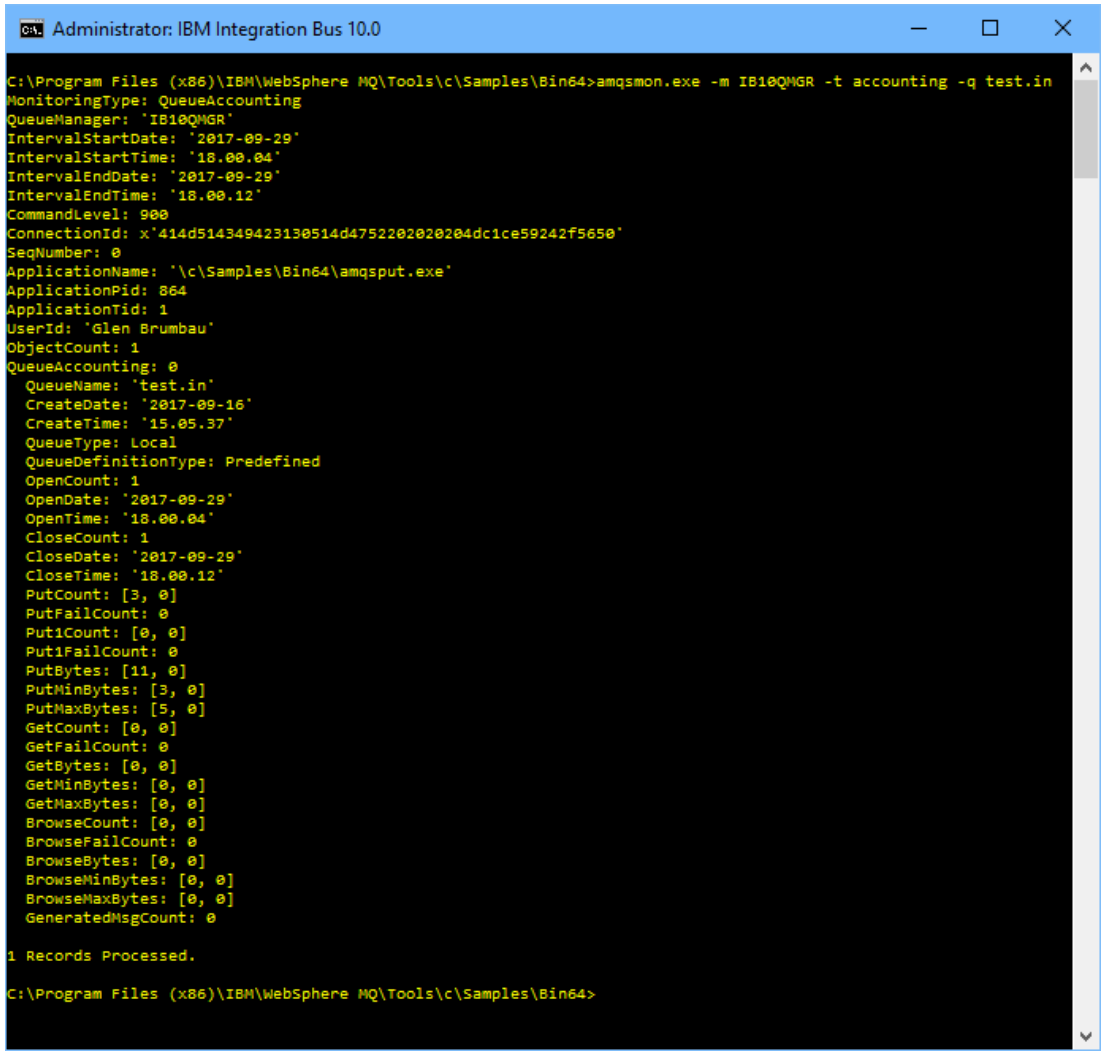
“amqsmon” Examples

- **amqsmon -m IB10QMGR -t accounting -b -q**
  - Display Accounting statistics for all queues -- save statistics messages
- **amqsmon -m IB10QMGR -t accounting -q *test.in***
  - Display Accounting statistics for the named queue (“test.in”)
- **amqsmon -m IB10QMGR -t statistics -q**
  - Display System statistics for all queues
- **amqsmon -m IB10QMGR -t statistics -q *test.in***
  - Display System statistics for the named queue (“test.in”)
- **amqsmon -m IB10QMGR -t statistics -c**
  - Display System statistics for all channels
- **amqsmon -m IB10QMGR -t statistics -c *test.svrconn***
  - Display Systems statistics for the named channel (“test.svrconn”)

“amqsmon” Sample Program Output



“amqsmon” Sample Program Output



MQ Performance Benchmarking

Tools – UNIX “top”

“top” Command Example

- **top**
  -U *userIDforMQ*
  -s *60*
  -n *10*
  -stats UID,COMMAND,PID,CPU,TIME,THREADS,TIME,MEM,STATE
  > *outputfile.txt*

- **Command description**
  - Interactive (e.g. On Screen) display
  - “-U” determines the User (Name or ID) for reporting
  - “-s” determines the “sample” (statistics) interval (60 seconds)
  - “-n” determines the number of “samples” (statistics) to report (10)
  - “-stats” determines the data fields to be reported.
  - “>” redirects “stdout” from screen to the named file.
  - This command will thus run for 10 minutes, with a summary reported every minute.

“top” Command Output

Terminal Shell Edit View Window Help 95% Sun Sep 24 9 31 AM Glen Brumbaugh

Glen — top — 179×50

```
Processes: 313 total, 2 running, 311 sleeping, 1431 threads                                                          09:31:50
Load Avg: 1.15, 1.75, 2.23  CPU usage: 4.12% user, 5.33% sys, 90.53% idle   SharedLibs: 189M resident, 37M data, 33M linkedit.
MemRegions: 62476 total, 1144M resident, 70M private, 539M shared. PhysMem: 8160M used (4775M wired), 30M unused.
VM: 914G vsize, 633M framework vsize, 11863200(0) swapins, 12323359(0) swapouts.  Networks: packets: 22444491/14G in, 60588979/74G out.
Disks: 9849371/324G read, 4433295/179G written.
```

UID	COMMAND	PID	%CPU	TIME	#TH	TIME	MEM	STATE
0	dprivacyd	99895	0.0	00:00.40	2	00:00.40	8192B	sleeping
501	AppleSpell	99214	0.0	00:48.65	2	00:48.65	5528K	sleeping
501	mdworker	95232	0.0	00:27.06	2	00:27.06	8192B	sleeping
0	mds	94348	0.0	05:40.15	5	05:40.15	6616K	sleeping
501	cfprefsd	89957	0.1	03:40.63	6	03:40.63	1484K	sleeping
501	mdwrite	88528	0.0	00:03.12	2	00:03.12	580K	sleeping
501	talagent	84914	0.0	00:06.50	2	00:06.50	1060K	sleeping
501	com.apple.Safari	84091	0.0	00:04.22	3	00:04.22	4476K	sleeping
501	com.apple.speech	78223	0.0	00:01.30	2	00:01.30	8192B	sleeping
501	com.apple.WebKit	77928	0.0	00:10.16	6	00:10.16	12M	sleeping
501	familycircled	74353	0.0	00:00.20	2	00:00.20	8192B	sleeping
0	system_installd	70328	0.0	00:15.56	2	00:15.56	760K	sleeping
501	nsurlsessiond	69608	0.0	01:28.63	4	01:28.63	7192K	sleeping
501	CoreServicesUIAg	66326	0.0	00:05.28	4	00:05.28	1336K	sleeping
501	DiskUnmountWatch	59245	0.0	00:00.21	2	00:00.21	8192B	sleeping
501	com.apple.WebKit	59137	0.0	00:04.47	6	00:04.47	2944K	sleeping
0	check_afp	59010	0.0	00:01.22	4	00:01.22	8192B	sleeping
501	coreauthd	58288	0.0	00:00.21	2	00:00.21	8192B	sleeping
501	IMRemoteURLConne	57430	0.0	00:02.72	3	00:02.72	772K	sleeping
0	installd	50225	0.0	02:19.68	2	02:19.68	836K	sleeping
501	System Events	47349	0.8	51:48.89	5	51:48.89	4892K	sleeping
501	cloudd	46703	0.0	01:36.51	5	01:36.51	11M	sleeping
501	com.apple.WebKit	45118	0.0	00:18.83	6	00:18.83	4320K	sleeping
501	com.apple.iCal.C	44297	0.0	00:47.44	4	00:47.44	1924K	sleeping
0	syspolicyd	42312	0.0	00:00.12	2	00:00.12	8192B	sleeping
501	SafariBookmarksS	41425	0.0	01:40.11	5	01:40.11	6372K	sleeping
0	top	36328	2.4	00:00.37	1/1	00:00.37	2724K+	running
0	amfid	35894	0.0	00:00.04	2	00:00.04	2088K	sleeping
0	wifivelocityd	35501	0.0	00:00.08	2	00:00.08	860K	sleeping
501	WiFiVelocityAgen	35495	0.0	00:00.08	3	00:00.08	548K	sleeping
0	ocspd	35204	0.0	00:00.03	2	00:00.03	1272K	sleeping
501	quicklookd	35193	0.0	00:00.13	4	00:00.13	2276K	sleeping
0	top	34756	0.0	00:00.95	1	00:00.95	8192B	sleeping
501	com.apple.WebKit	33705	0.0	00:06.33	6	00:06.33	34M	sleeping
501	com.apple.Safari	33399	0.0	00:00.16	2	00:00.16	3032K	sleeping
501	com.apple.access	33388	0.0	00:00.01	2	00:00.01	580K	sleeping
501	assistant_servic	31819	0.0	00:00.06	2	00:00.06	772K	sleeping
501	com.apple.iTunes	31797	0.0	00:00.07	2	00:00.07	2724K	sleeping
501	assistantd	31751	0.0	00:01.14	4	00:01.14	8484K	sleeping
501	PrintUITool	31442	0.0	00:00.18	2	00:00.18	68K	sleeping
243	nsurlstoraged	30909	0.0	00:00.02	2	00:00.02	20K	sleeping
501	com.apple.Safari	30818	0.0	00:00.23	3	00:00.23	3984K	sleeping
501	MTLCompilerServi	30611	0.0	00:00.19	2	00:00.19	8192B	sleeping

MQ Performance Benchmarking

Tools – Windows “PerfMon”

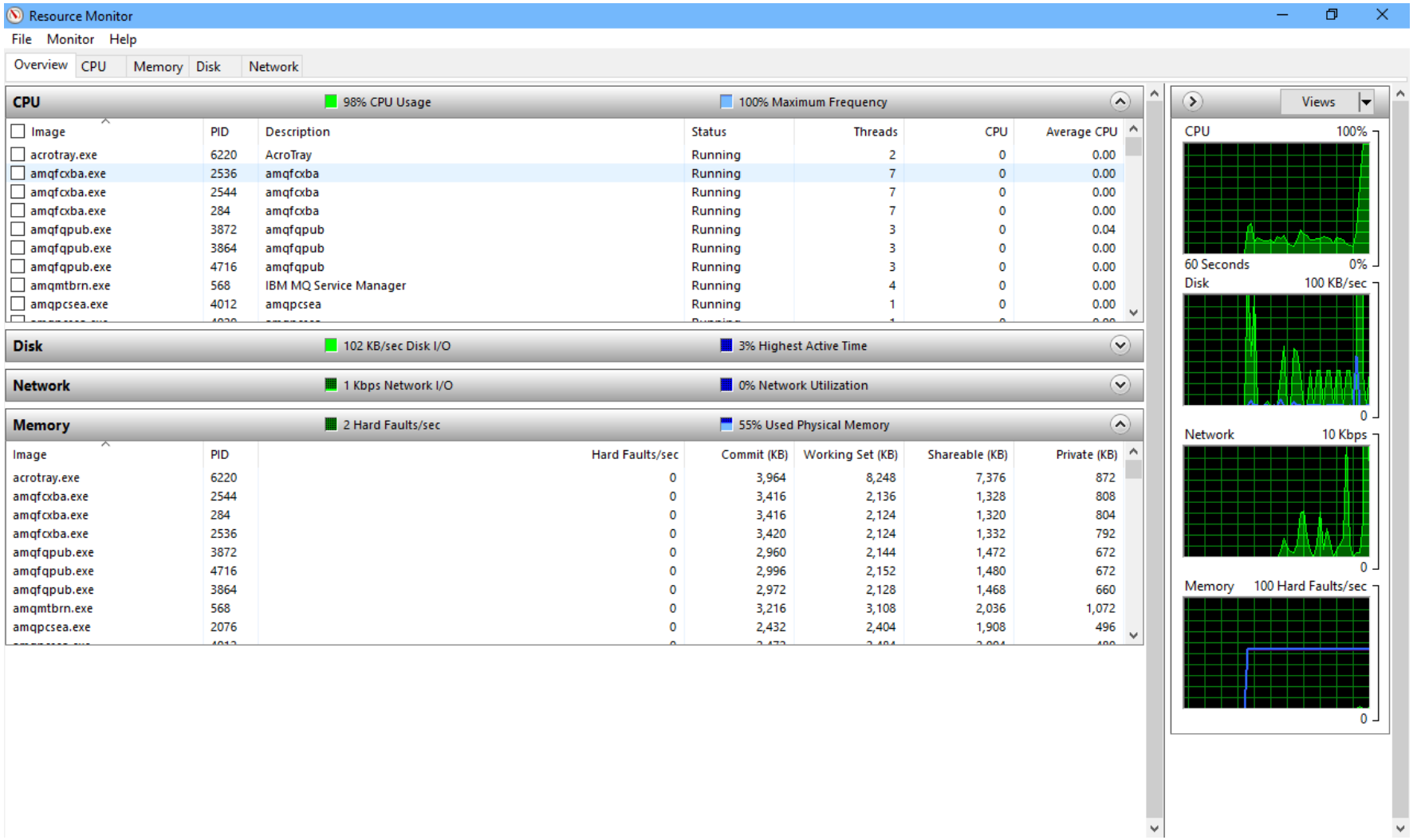
“PerfMon” Program

- **Tool Overview**
  - Microsoft Windows Performance Monitoring tool
    - Measure CPU, Memory, Disk, and Network usage
  - Microsoft Management Console (MMC) Snap-In
- **Tool Highlights**
  - Included in the standard Windows distribution
  - May require installation through Control Panel (Add Programs)
  - Customizable reporting
    - Data Collector Sets for defining collection & reporting data
    - Multiple output formats supported (e.g. CSV)
- **Tool History**
  - Developed by Microsoft and introduced in Windows NT 3.1
  - Location and tool launching process has differed across Windows software releases
- **Tool Launch (Windows 10.1)**
  - Windows → Run → PerfMon

“PerfMon” Data Collector Sets

- **Data Collector Sets can be started and stopped independently**
- **Creating a Data Collector Set**
  - Data Collector Sets → User Defined → (Right Click) New → Data Collector Set
    - Define Data Collector Set name (e.g. “IBM MQ”)
    - Create manually or from template
    - Select “Performance Counters” (fields to be reported)
      - e.g. “Process”, “Processor Information”, “Memory”,
    - Enter optional search criteria for each Performance Counter
      - e.g. “amq”, “runmq”, etc. for the “Process” probe
      - Select desired instances (or “<All Instances>”)
    - Select the fields to be reported with each Performance Counter
    - Final “Performance Counter” format is:
      - *****PeformanceCounterType***(***selectedInstance***)***reportedFields
      - e.g. \Process(amqrrmfa)*
    - Select log file data format
      - Binary, Comma Separated (CSV), Tab Separated, SQL

“PerfMon” Tool Output



MQ Performance Benchmarking

Tools – “JUnit”

“JUnit” Test Framework

- **Tool Overview**
  - Java based Open Source Test Framework
  - Supports Java 8 or later
  - Widely used (approximately 30% of Java projects)
  - http://junit.org/junit5/
- **Tool Highlights**
  - Flexible components (JUnit Platform, JUnit Jupiter, JUnit Vintage)
  - Formalized test descriptions for repeatable tests
  - Extensible to any software that interfaces with Java
- **Tool History**
  - Evolved from “SUnit”, which was written in Smalltalk in 1994
  - Currently available through junit.org
    - https://github.com/junit-team/junit5/

“JUnit” Integration with IIB

- **Open Source JUnit Extension**
  - Developed by Rocket-IT Consulting developed for IIB
  - Available for download
    - https://github.com/rockitconsulting/test.rockitizer
- **IBM developerWorks documentation**
  - https://developer.ibm.com/integration/blog/2017/08/29/junit-based-integration-testing-ibm-integration-bus/

Title

Tools – “JMeter”

Additional Testing Tools

- **JMeter**
  - Open Source test framework from Apache
  - Java based, so supports JMS testing
  - Build a JMS test plan through the JMeter GUI
    - https://blazemeter.com/blog/building-jms-testing-plan-apache-jmeter
  - Executes pre-built test plans
  - Supports multi-threaded load testing
  - http://jmeter.apache.org

Title

Summary

Take Away Points

- **Understand how Applications use MQ**
  - Datagram
  - Request/Response
- **Understand where Applications use MQ**
  - Servers
  - Queue Managers
  - Channels
- **Understand Application Infrastructure *Reader* and *Writer* thread counts**
- **Benchmark Infrastructure capability using "Q" or "PerfHarness"**
  - Use "*Q*" or "*PerfHarness*" to generate & measure test loads
  - Use "*xmqqstat*", "*amqsmon*", or "*amqsrua*" to generate additional reporting data
  - Use "PerfMon" (Windows) or "top" (Unix) to report OS level statistics
  - Use "PerfRating" to compare CPU performance
- **Benchmark Application performance (End to End)**
  - Use same reporting tools as with Infrastructure
  - Capture input data (if possible) for replay
  - Replay input data using "Q" or "PerfHarness"

Questions & Answers

Presenter

- Glen Brumbaugh
  - Glen.Brumbaugh@TxMQ.com
- Computer Science Background
  - Lecturer in Computer Science, University of California, Berkeley
  - Professorial Lecturer in Information Systems, Golden Gate University, San Francisco
- WebSphere MQ Background (25 years plus)
  - IBM Business Enterprise Solutions Team (BEST)
    - Initial support for MQSeries v1.0
    - Trained and mentored by Hursley MQSeries staff
  - IBM U.S. Messaging Solutions Lead, GTS
  - Platforms Supported
    - MVS aka z/OS
    - UNIX (AIX, Linux, Sun OS, Sun Solaris, HP-UX)
    - Windows
    - iSeries (i5OS)
  - Programming Languages
    - C, COBOL, Java (JNI, WMQ for Java, WMQ for JMS)


Thank
You