IBM Performance Harness for Java(tm) Message Service

For all updates and feedback, please visit our GitHub site:

https://github.com/ot4i/perf-harness

Table of Contents

What is Performance Harness for JMS?

The Java™ Message Service API offers a vendor-neutral approach to messaging in Java and J2EE environments. Both for vendors developing JMS services and customers using those services, an understanding of the performance characteristics is one of the key features of creating the best solution possible.

Performance Harness for Java™ Message Service is a flexible and modular Java package for performance testing of JMS scenarios and providers. It provides a complete set of JMS functionality as well as many other features such as throttled operation (a fixed rate and/or number of messages), multiple destinations, live performance reporting, JNDI and multiple vendor plug-ins. It is one of the many tools used by WebSphere MQ, WebSphere Message Broker, and WebSphere ESB performance teams for tests ranging from a single client to more than 10,000 clients.

There are many modules implementing point-to-point and publish-subscribe modes of operation, which can be explored through the documentation. Each of these modules, when selected, sends and/or receives messages from the selected JMS provider as fast they can (unless a certain rate is specified). They share a common command-line reporting mechanism and will print their current throughput rate on a user-selected periodic basis and also output summary statistics at the end of a test. The included help and documentation provide detailed usage instructions and describe many further features and configuration parameters for investigation.

With the release of v1.2 the tool is now extending beyond the realms of JMS Messaging to include native MQ as well as support for other transports such as HTTP/SOAP to allow the tool to be used to more fully drive the various transports that ESB implementations such as WebSphere ESB and WebSphere Message Broker provide.

What's new in version 1.2?

- Support for WebSphere MQv7
- Support for Java5
- HTTP module added to allow the testing of HTTP and SOAP Transports
- New options to many modules
- Many Bug Fixes

Using Performance Harness for JMS

As with any tool, this one has many different uses depending on the goals of the user and can also be thoroughly misdirected to produce useless data. Ensure the performance scenarios you choose to measure bear some relation to the real world. Failure to do so will inevitably lead to incorrect facts, figures, assumptions and decisions. For instance, it is common to see competitive product comparisons being "won conclusively" by using scenarios that mean nothing in real customer environments. It is also worthy of note that performance is usually not the most important factor in any such comparison, it is simply the easiest to create charts from.

Requirements

- Java 5 or later.
- Client jars from your JMS provider.
- WebSphere MQv7 Client libraries if you wish to run any of the MQ Java related classes

Migration from previous versions

- As stated in the requirements Java 5 is the minimal pre-req.

- Users of the WebSphere MQ related modules will need to use WebSphere MQ v7 client libraries.
- Support for 1.02 JMS clients has been removed as it's expected all providers to now be JMS 1.1 compliant. Users of these older modules should migrate to the 1.1 equivalent.
- The WBIMB module has now been renamed to WMB.

HOWTO

This section should explain how to get up and running with JMSPerfHarness. There are many more parameters beyond those discussed here, please use the parameter reference in this doc to see the many additional capabilities.

How to set the Java classpath

JMSPerfHarness does not run as an executable Jar. This is unfortunate, but unavoidable due to the nature of Jar file manifests and the requirement for this application to be deployed against multiple products. Make sure the perfharness.jar and any required provider jars are on the Java classpath when invoking the tool. It is assumed you have already accomplished this although the tool will warn you if you have not.

Therefore, invoking JMSPerfHarness can be done in two ways

- Add it to the classpath.
  On Windows use:
- `set CLASSPATH=perfharness.jar;%CLASSPATH%`
- `java JMSPerfHarness`

  On UNIX platforms use:

  ```
  export CLASSPATH=perfharness.jar:$CLASSPATH
  java JMSPerfHarness
  ```

- Include the classpath on the Java invocation.
  On Windows use
- `java -cp "perfharness.jar;%CLASSPATH%" JMSPerfHarness`

  On UNIX platforms use

  ```
  java -cp "perfharness.jar:$CLASSPATH" JMSPerfHarness
  ```

How to use the built-in help

Performance Harness for Java Message Service is a very modular tool and certain modules need to be selected via the command-line for different modes of operation. At any time, using "-h" will print help on the current context (i.e. the currently loaded modules).

Parameter	Description
-h	At any time, using "-h" will print help on the current context (i.e. the currently active modules). It will not give help on modules that are not active.
-hf	This performs the same as "-h", but prints additional details on parameters and includes other, seldom used, parameters
-hm	This gives the full help for a named module, regardless of whether it is active or not. You do not need to pass the full classname as the tool will search intelligently for the named class. Example: -hm WebSphereMQ or -hm mqjava.Responder

How to choose your JMS test class

The tool's operation is defined by the test class being run and there are many selections of test class. Each of the following classes may provide a few additional options to fine tune behaviour. More details can be found in the previous section.

Parameter	Description
-tc jms.r11.Sender	Sends messages to a named queue destination.
-tc jms.r11.Receiver	Receives messages from a named queue destination. This can be used in conjunction with the Sender class.
-tc jms.r11.PutGet	Sends a message to queue then retrieves the same message (using CorrelationId). This is the default setting.
-tc jms.r11.Requestor	Sends a message to a queue then waits for a corresponding reply on a second queue.
-tc jms.r11.Responder	Waits for a message on a queue then replies to it on another queue. This can be used in conjunction with the Requestor class.
-tc jms.r11.Publisher	Sends messages to a named topic destination.
-tc jms.r11.Subscriber	subscribes and receives messages from a named topic. This can be used in conjunction with the Requestor class.

The above classes all use the JMS 1.1 API.

How to choose from the available JMS provider classes

The tool comes packages with three JMS provider classes. These are selected with the "-pc" parameter (which is *case-specific*)

Parameter	Description
-pc WebSphereMQ	This allows simple command-line access to IBM WebSphere MQ JMS settings. WebSphereMQ provides a full JMS implementation using an industry-proven messaging engine with ten years of pedigree. The parameters this gives access to care listed in the previous section.
-pc WMB	This allows simple command-line access to IBM WebSphere Business Integrator Message Broker (and Event Broker) JMS settings. Aside from its many transformation and routing capabilities, WMB builds upon the JMS capability of WebSphere MQ. It provides additional high-performance nonpersistent publish-subscribe, multicast networking and content-based-routing (above and beyond the capability of JMS selectors). The parameters this gives access to are listed in the previous section.
-pc JNDI	This is the default setting. JNDI (Java Naming and Directory Interface) provides a method for a pure JMS application to work with any JMS vendor (including the above products).

How to use JNDI administered objects

JNDI is the default provider class and the most flexible, allowing object lookup from any JMS vendor. The JNDI objects will need to be created beforehand, in whatever provider-specific fashion is documented by that vendor (although some products also provide auto-creation facilities linked to JNDI lookup).

To enable JNDI you must specify the "-cf" property. This can be done even with vendor-specific provider classes (in which case most of the vendor-specific parameters those modules provide will be ignored).

JNDI parameters

The JNDI specification provides for several ways to pass parameters to an application. JNDI parameters may be specified from three sources and are evaluated in the following order in the case of duplication.

1. The "-ii" and "-iu" parameters are passed directly to the InitialContext.
2. The Java environment parameters (eg. -Djava.naming.initial.context=xxx).
3. A jndi.properties file placed on the classpath of the application. These are described in more detail here. **You must consult the client manuals for your product to fill in these settings.**

Parameter	Description

-ii com.sun.jndi.fscontext.RefFSContextFactory	The class your vendor specifies for the JNDI InitialContextFactory.
-iu file:/C:/JNDI-Directory	A URL parameter to the above class which tells it which configuration to use.
-cf test/connfactory	The name of the JNDI ConnectionFactory object JMSPerfHarness is to use.

4. When using JNDI, the destinations you specify are the names of the JNDI objects, not necessarily the same as the underlying queue or topic destinations. This should only be of importance if you mix usage of JNDI and non-JNDI provider classes.

JNDI with WebSphere MQ

If using JNDI with WebSphere MQ, SupportPac ME01 (WebSphere MQ - Initial Context Factory) allows the queue manager to be used directly as a JNDI provider and avoids the requirement to predefine WMQ queues or topics. SupportPac ME01 pre-requires SupportPac MS0B (MQSeries Java classes for PCF). An example of use is given below.

How to use multiple JMS destinations

The tool will handle multiple destinations (publish-subscribe topics or point-to-point queues) with the right configuration parameters. This allows more complicated scenarios to be constructed across multiple instances of the tool.

The general concept being applied is that an ordered set of destinations are created and then distributed evenly amongst the active threads.

Parameter	Description
-d	Destination prefix. The default is "DEST"
-db	First number in the range.
-+	Last number in the range.
-dn	Number of destinations in the range (or the first destination t use in a fixed range, see example XXX).

The module will infer the set of destinations from the parameters being passed. The inferences can best be explained by example:

parameters description -d PARIS Without specifying any other parameters, all threads will use a single destination, PARIS -d MYTOPIC -db 10 Specifying

only anumeric base creates an open ended list, each subsequent thread will get an incremented destination, MYTOPIC10... -d MYTOPIC -db 4 -dn 5 Destinations are distributed round-robin in the order MYTOPIC4...MYTOPIC8 -d MYTOPIC -dx 8 -dn 5 As above -d MYTOPIC -dx 4 -dx 8 As above -d MYTOPIC -dx 4 -dx 8 -dn 6 This is a special case, the range is as above (4 to 8) but the sequence will start at 6

Notes:

- These parameters only control the names given to destinations. Specifying "-d TOPIC" does not, in itself, enable publish-subscribe (you could have a queue named TOPIC).
- In JNDI mode, these become the names of the JNDI destinations.
- Each single thread is assigned a single destination.
- It is not considered invalid to specify more destinations than there are threads or to create an uneven balance of destinations amongst threads.

How to use the non-JMS "WebSphere MQ classes for Java"

The "WebSphere MQ classes for Java" provide a non-JMS interface onto the full detail and capabilities of the WebSphere MQ messaging API (the MQI). These are not part of the JMS portions of this tool and therefore do not accept many of the JMS specific parameters, they are most closely related (naturally enough) to the WebSphereMQ provider module. The following test classes are part of the mqjava package:

Parameter	Description
-tc mqjava.Sender	Sends messages to a named queue destination.
-tc majava.Receiver	Receives messages from a named queue destination. This can be used in conjunction with the Sender class.
-tc mqjava.Requestor	Sends a message to a queue then waits for a reply on a second queue with a matching CorrelationId.
-tc mqjava.Responder	Waits for a message on a queue then replies to it by pla message on another queue with a corresponding Correla This can be used in conjunction with the Requestor class
-tc mqjava.PutGet	Sends a message to queue then retrieves the same message (using CorrelationId).

The following limitations should be noted:

- There are no publish-subscribe classes implemented for mqjava, although such functionality is achievable.

- Since this is not JMS, the "-pc" parameter is not valid, nor are any other JMS specific values. Users commonly try to change "jms.r11" with" mqjava", this will often not work.
- In this version of JMSPerfHarness, multiple destination options ("-db", "-dx" , "-dp" and "-dn") are not available for the mqjava package

How to use the HTTP module

The module is a simple HTTP client application which sends and receives a HTTP message and reports message rates.

Parameter	Description
-tc http.HTTPRequestor	Sends messages to a named URL. It sends the contents of a file as a HTTP Request and waits for a response.

How to use the AMQP Channel module

This module utilises the AMQP Java Client to send and receive AMQP messages and report message rates.

Parameter	Description
-tc amqp.Requestor	Subscribes to a unique topic and waits to receive a message after previously sending to a different topic.
-tc amqp.Responder	Performs a shared subscribed on a topic and forwards received message to name topics based on the message.

The Requestor and Responder is designed to operate together to simulate a typical worker off-loader scenario. The responder functions like a worker off-loader as one or more responders read messages from a shared topic. These messages read have a number of optional parameters embedded at the beginning of the message. One of those parameters indicate which topic to forward the message to. In this case its the returning topic name for the originating Requestor. The Requestor subscribes to that topic and generate messages with the topic embedded. The perform iteration to send and receiving message.

Note: start Responder before Requestor otherwise message will be discarding and will not be returned to the Requestor.

See the Example invocations for more explanation

How to use the Command Processor

You can specify a command processor to issue commands to a running PerfHarness process. A command processor must extend and implement the com.ibm.uk.hursley.perfharness.cmd.Command abstract class. A simple telnet command processor is included which can be run by specifying:

```
-cmd_c SocketCommandProcessor
```

By default this listens on port 4444. A telnet client can then open a link to the process and issue commands to

- Request statistics
- Set the message rate
- Increase the number of threads

E.g.

```
[~]$ telnet localhost 4444
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
,connected to SocketCommandProcessor on port 4444
REPORT -stats
,rate=6050.00,total messages=6050,Snapshot period=1,threads=1
ALTER -rt 500
,SUCCESS: ALTER -rt 500
REPORT -stats
,rate=501.00,total messages=501,Snapshot period=1,threads=1
START -nt 2
,SUCCESS: START -nt 2
REPORT -stats
,rate=1500.00,total messages=1500,Snapshot period=1,threads=3
```

Example invocations

Putting together the lessons from the HOWTO section will give you a basic operational performance tool. The following are some sample invocations of the functionality in this tool, see the command-line reference for the meaning of any unknown parameters:

Point-to-point using JNDI

Persistent, transacted point-to-point JMS in a send-receive loop (a very basic operational test of queuing) on a single queue using 6 clients and JNDI administered objects provided by WebSphere MQSupportPac ME01.

```
java JMSPerfHarness -tc jms.r11.PutGet -nt 6 -pp -tx -pc JNDI -ii
com.ibm.mq.jms.context.WMQInitialContextFactory -iu
localhost:1414/SYSTEM.DEF.SVRCONN -cf QM_RED -d QUEUE
```

Point-to-point with WebSphere MQ

Persistent, transacted point-to-point with WebSphereMQ in local bindings. 10 queue-triplets (each queue has 1 sender and 1 receiver) running on queues (QUEUE1..QUEUE10). The number of triplets can be varied arbitrarily. A corresponding test with topics simply requires different test class parameters.

```
export TRIPLETS=10

java JMSPerfHarness -tc jms.r11.Sender   -nt $TRIPLETS -pc WebSphereMQ -jb
QM_RED -jt mqb -d QUEUE -db 1 -pp -tx

java JMSPerfHarness -tc jms.r11.Receiver -nt $TRIPLETS -pc WebSphereMQ -jb
QM_RED -jt mqb -d QUEUE -db 1 -pp -tx
```

Nonpersistent point-to-point with WebSphereMQ in local bindings. Put a million 100-byte messages to a destination QUEUE, then get them back again.

```
export MSGSIZE=100

java JMSPerfHarness -tc jms.r11.Sender   -pc WebSphereMQ -jb QM_RED -jt
mqb -d QUEUE -ms $MSGSIZE -mg 1000000

java JMSPerfHarness -tc jms.r11.Receiver -pc WebSphereMQ -jb QM_RED -jt
mqb -d QUEUE
```

Put-Get with WebSphere MQ, using client channels defined in a CCDT

Client bindings are specified with the -jt MQC parm. In this case you either need to explicity set the client channel properties (host, port, channel name etc) with the corresponding cmd line options (jh,jp, jc), or you can point to a client channel definition table, generated by the queue manger. If a ccdt is specified only the queue manager is needed on the JMSPerfHarness command line (-jb), which will be used to find a suitable channel definition in the table. The simple example below runs 10 threads Putting and Getting from a pair of queues (QUEUE1 & QUEUE2), but you could set ccdt on any MQ JMS test.

```
export THREADS=10
export QUEUES=2
```

```
JMSPerfHarness -su -wt 10000 -wi 0 -nt $THREADS -sc BasicStats -ms
$msgsize -tc jms.r11.PutGet -co -d QUEUE -db 1 -dx $QUEUES -dn 1 -jb
QM_RED -jt mqc -pc WebSphereMQ -ri 1 -pp true -tx true -rt 1 -vo SEVERE  -
ccdt file:///mqperf/pharris/HA/AMQCLCHL.TAB
```

Reconnection testing with WebSphere MQ

You can use the jms.r11.ReconnectTimer class to report on how long it takes for all of your clients to reconnect to a secondary/standby queue manager, after a switch/fail-over scenario in an MIQM or RDQM HA topology.

E.g. If you have two queue managers QM1 (active, on primaryHost), and QM1 (standby, on secondaryHost), both with listeners on port 1414, then the following command will test how long it takes for 3 threads to re-connect following (e.g.) the issue of endmqm -s QM1 in an MIQM environment.

```
java -Xms4096M -Xmx4096M -Xmn3072M JMSPerfHarness -su -wt 10000 -wi 0 -nt 3 -
id 1 -ss 1 -sc BasicStats -ms 2018 -rl 0 -tc jms.r11.ReconnectTimer -co -d
REQUEST  -to 30 -mt text -db 1  -dx 3 -dn 1 -jp 1414 -jc SYSTEM.DEF.SVRCONN -
jb PERF0 -jt mqc -pc WebSphereMQ -jh primaryHost -jq
SYSTEM.BROKER.DEFAULT.STREAM -ja 100 -ri 1 -ro 600 -h2 secondaryHost -pp true
-tx true -rt 1 -wp true -wc 3 -vo SEVERE
```

```
Output:

ReconnectTimer1: All threads initially connected. Start/End times:
14:05:01.477 / 14:05:02.082

Creating receiver on queue:///REQUEST2 with selector:
JMSCorrelationID='ID:3100000000000000000000000000000000000000000032'

Creating receiver on queue:///REQUEST3 with selector:
JMSCorrelationID='ID:3100000000000000000000000000000000000000000031'

Creating receiver on queue:///REQUEST1 with selector:
JMSCorrelationID='ID:3100000000000000000000000000000000000000000033'

id=1,rate=3.00,total messages=3,Snapshot period=1,threads=2
id=1,rate=3.00,total messages=3,Snapshot period=1,threads=3
id=1,rate=3.00,total messages=3,Snapshot period=1,threads=3

.. .. ← endmqm -s issue here

com.ibm.msg.client.jms.DetailedJMSException: JMSWMQ2007: Failed to send a
message to destination 'REQUEST2'.

.. .. .. (jms errors thrown)

Creating receiver on queue:///REQUEST2 with selector:
JMSCorrelationID='ID:3100000000000000000000000000000000000000000032'

Creating receiver on queue:///REQUEST3 with selector:
JMSCorrelationID='ID:3100000000000000000000000000000000000000000031'

Creating receiver on queue:///REQUEST1 with selector:
JMSCorrelationID='ID:3100000000000000000000000000000000000000000033'

ReconnectTimer2: Time to connect to secondary host is 44 ms (min: 44 ms  max:
44 ms)
```

```
ReconnectTimer3: Time to connect to secondary host is 43 ms (min: 43 ms  max: 44 ms)
ReconnectTimer1: Time to connect to secondary host is 43 ms (min: 43 ms  max: 44 ms)
ReconnectTimer1: All threads reconnected at 14:05:07.334
id=1,rate=0.00,total messages=0,Snapshot period=1,threads=3
id=1,rate=3.00,total messages=3,Snapshot period=1,threads=3
... ...
```

If the QM on the primaryHost is re-started, and the QM on the secondary ended with endmqm -s, then the client will reconnect back to the QM on the primaryHost, and so forth.

Instead of using h2 & (optionally) p2 to specify the secondary host & port, you can use a ccdt to provide the channel definitions of the queue managers:

```
java -Xms4096M -Xmx4096M -Xmn3072M JMSPerfHarness -su -wt 10000 -wi 0 -nt 3 -id 1 -ss 1 -sc BasicStats -ms 2018 -rl 0 -tc jms.r11.ReconnectTimer -co -d REQUEST  -to 30 -mt text -db 1  -dx 3 -dn 1 -jb QM1 -jt mqc -pc WebSphereMQ -jq SYSTEM.BROKER.DEFAULT.STREAM -ja 100 -ri 1 -ro 600  -pp true -tx true -rt 1 -wp true -wc 3 -vo SEVERE -ccdt file:///mydirectory/AMQCLCHL.TAB
```

There is also a jms.r11.AutoReconnectTimer class which uses JMS automatic client re-connection functionality, but this is less usefull from a timing perspective as many (sometimes all) of the threads may switch/fail-over to the secondary host without the application code being made aware, so no times will be reported for these.

Point-to-point with WebSphere Application Server

Here is a sample batch file that can be used to send/receive messages from WAS/WESB using the default JMS bindings (I will just use WESB to denote either WESB/WAS/WPS). I will step through the parameters of note.

For WAS 6.0.2

```
SET JAVA_PATH=%WESB_HOME%\java\bin\
SET
EXTDIRS=%WESB_HOME%\classes;%WESB_HOME%\lib;%WESB_HOME%\installedChannels;%WESB_HOME%\java\jre\lib\ext;
SET JAR_FILE=c:\JMSPerfHarness\perfharness.jar
%JAVA_PATH%\java \-Xms768M \-Xmx768M \-Djava.ext.dirs=%EXTDIRS% \-cp
%JAR_FILE% JMSPerfHarness \-tc jms.r11.Requestor \-cf jndi_JMS_BASE_QCF \-ii
com.ibm.websphere.naming.WsnInitialContextFactory \-iu
iiop://machinename.hursley.ibm.com:2812 \-iq jndi_INPUT_Q \-oq jndi_OUTPUT_Q
\-nt %1 \-to 15 \-pf c:\JMSPerfHarness\soabench\soab-jms.properties \-mf
c:\JMSPerfHarness\soabench\soa_med_jms_2k_1k.xml \-co true \-pp true \-tx
true
```

Access to various directories from the server installation or the J2SE JMS client (http://www-1.ibm.com/support/docview.wss?uid=swg24012804) is required and is setup in the EXTDIRS command variable. I use a WESB_HOME environment variable to indicate the location of the server/client jars I wish to run against. Using a large heap and setting ms equal to mx offers consistent performance levels as heap expansion is prevented during the run.

All the parameters are shown in detail later, but the main ones are shown below:

```
-tc     This indicates the client type. For most WESB Messaging scenarios,
messages are consumed from an inbound queue and placed on an outbound queue.
The jms.r11.Requestor client will send a message to the queue defined by the
JNDI name specified by the -iq parameter and look to receive the
corresponding response from the JNDI name specified by the -oq parameter.
-cf    This is the name of the Default Messaging Provider's Queue Connection
Factory you wish to use when sending/receiving messages. This is generally
the best entity in which to define which QualityOfService you wish to use
(http://publib.boulder.ibm.com/infocenter/wasinfo/v6r0/index.jsp?
topic=/com.ibm.websphere.pmc.express.doc/ref/rjn_jmscf_modify.ht
ml) - Pay particular attention to the JNDI Name, Persistent and NonPersistent
message reliability and providerEndpoints properties. (The providerEndpoints
must be set if your JMS client is not co-located on the same machine as your
WESB installation)


-ii     InitialContextFactory implementation class to be used when
communicating with WESB
-iu    Provider URL - This is the RMI port on the server on which the client
will use to lookup JNDI entries
-iq     JNDI name of inbound queue
-oq     JNDI name of outbound queue
-nt     Number of producer threads
-to     Producer timeout value (how long each thread waits for its response)
-pf    This denotes a properties file which is used to set bespoke JMS
properties in the request message's JMS header
    When sending JMS messages into WESB using JMS Exports, I use this file to
set the TargetFunctionName. The JMS Export will match this name against the
operations supported by the JMS Export interface
    and invoke the requested operation
-mf    This denotes an input file to be sent as the payload of the message
instead of generated text/bytes data.
-co     This is set to true to enable Correlation ID matching to ensure each
requestor thread processes its own reply.
-pp     Set to true to send a Persistent message.
-tx     Set to true to commit each message being sent within a session.
```

For WAS 6.1

A few additional directories are required to be set on the EXTDIRS variable.

```
            SET
EXTDIRS=%WESB_HOME%\classes;%WESB_HOME%\lib;%WESB_HOME%\plugins;%WESB_HOME%\j
ava\jre\lib\ext;%WESB_HOME%\runtimes;
```

Publish-subscribe

- Persistent, transacted publish-subscribe with WebSphereMQ over TCP/IP. 4 publishers and 40 durable subscribers spread evenly across 4 topics (TOPIC1..TOPIC4). Durable subscribers will use the same name (by setting -id) and do not unsubscribe (un=false). This means the subscribing application can be started and stopped without message loss.

```
java JMSPerfHarness -tc jms.r11.Publisher  -nt 4  -pc WebSphereMQ -jh server1
-jb QM_RED -jt mqc -jp 1415 -d TOPIC -db 1 -dx 4
java JMSPerfHarness -tc jms.r11.Subscriber -nt 40 -pc WebSphereMQ -jh server1
-jb QM_RED -jt mqc -jp 1415 -d TOPIC -db 1 -dx 4 -du -id SUBS -un false
```

- Nonpersistent publish-subscribe with WebSphere Message Broker using the Real-time transport. 1 publisher and 1 subscriber using 1 topic. The real-time flow on the broker is assumed to be configured to port 1506. Both are set to run for 120 seconds, with the publisher sending 100 messages a second. It will send JMSTextMessage objects using the contents of MyMessage.xml for the message body.

```
java JMSPerfHarness -tc jms.r11.Publisher  -pc WMB -jh server1 -jp 1506 -jt
ip -d Topic/0000 -rt 100 -mt text -rl 120 -mf .\MyMessage.xml

java JMSPerfHarness -tc jms.r11.Subscriber -pc WMB -jh server1 -jp 1506 -jt
ip -d Topic/0000 -rl 120
```

- Nonpersistent publish-subscribe with WebSphere Message Broker using the Real-time transport. 1 publisher and 1 subscriber using 1 topic. The real-time flow on the broker is assumed to be configured to port 1506. Both are set to run for 120 seconds, with the publisher sending 100 messages a second. It will send JMSTextMessage objects using the contents of MyMessage.xml for the message body.

```
java JMSPerfHarness -tc jms.r11.Publisher  -pc WMB -jh server1 -jp 1506 -jt
ip -d Topic/0000 -rt 100 -mt text -rl 120 -mf .\MyMessage.xml

java JMSPerfHarness -tc jms.r11.Subscriber -pc WMB -jh server1 -jp 1506 -jt
ip -d Topic/0000 -rl 120
```

- Nonpersistent publish-subscribe with WebSphere Message Broker using the Reliable Multicast transport for the subscriber. Topic "Topic/0000" must be defined as a Multicast topic (see product docs). This subscriber could be run in conjunction with the real-time publisher shown above. 1 subscriber using 1 topic set to run for 120 seconds, the subscriber's buffer size is set to 3000 messages.

```
java JMSPerfHarness -tc jms.r11.Subscriber -pc WMB  -jh server1 -jp 1506 -jt
ipmcr -d Topic/0000 -jz 3000 -rl 120
```

- Nonpersistent publish-subscribe with WebSphere Message Broker using the WebSphere MQ transport. 1 publisher and 1 subscriber using 1 topic. Both are set to run for 240 seconds, with the publisher sending 100 messages a second. It will send a 1024-byte JMSBytesMessage to a queue called PUBLISH on queue manager QM_RED on port 1414.

```
java JMSPerfHarness -tc jms.r11.Publisher  -pc WMB -d Topic/0000 -jh server1
-jp 1414 -jt mqc -rl 240 -jb QM_RED -jq PUBLISH -ja 100 -rt 100 -mt bytes -ms
1024

java JMSPerfHarness -tc jms.r11.Subscriber -pc WMB -d Topic/0000 -jh server1
-jp 1414 -jt mqc -rl 240 -jb QM_RED
```

WebSphere MQ classes for Java

- Persistent, transacted point-to-point Send and Receive using the mqjava.Requestor using 5 clients. The message file referenced is one that contains an RFH2 header at the start as produced from RFHUtil SupportPac IH03. Each client will send a message to INPUTQUEUE and wait for a reponse on the OUTPUTQUEUE. The tool will run for 120 secs and print stats every 5 seconds.

```
java JMSPerfHarness -tc mqjava.Requestor -iq INPUTQUEUE -oq  OUTPUTQUEUE -rl
120 -jb QM_RED -jh server1 -jp 1414 -mf c:/Input_XML.msg -rf 2  -ss 5 -nt 5 -
pp -tx
```

HTTP Module

- Driving a HTTP URL. An example invocation is shown below which will start one http client thread against the URL "http://10.16.112.39:7080/test". The test will run for 120 secs and will print out message rates every 10 seconds.

```
java JMSPerfHarness -tc http.HTTPRequestor -jh 10.16.112.39 -jp 7080 -ss 10 -
nt 1 -mf E:\input.xml -wi 50 -to 500000 -ur test -wo 10000 -rl 120
```

- Driving SOAP Input Node Example. An example invocation is shown below which will start 20 http client thread against the URL "http://hound:7800/WssSale/services/WssSale". The test will run for 310 secs and will print out message rates every 3 seconds. It will set the SOAPAction header to be "SummerSale".

```
java JMSPerfHarness -tc http.HTTPRequestor -jp 7800 -wi 100 -to 50000 -jh
HOUND -sa SummerSale -mf i:\InputMessages\SOAP_Sale_4K.xml -nt 20 -ss 3 -rl
310 -ur WssSale/services/WssSale
```

Some of the main parameters for this module are shown below:

```
rl = runlength in seconds
ss = stats interval how often rate is printed,
```

```
nt = number of threads(clients)
mf = message file path to use as the message contents
wo = write out a copy of the response message to a file after x messages
wi = wait interval between starting each client in ms,
to = time out interval in milli secs - how long each client will wait for a
response before throwing a timeout,
cs = close socket after each message (uses non persistent HTTP connections
i.e. HTTP 1.0)
sl = sleep time in milliseconds between sending each message
ur = url to append to http://hostname:port/url,
sa = Text to put in the SOAPAction Header.
```

AMQP Channel Module

Worker off-loading

Requestor

The following starts a Requestor, which will generate messages to the outbound topic "REQUEST" and receive the messages back on "REPLY-<thread Id>-1 (1 is the -id value).

```
java JMSPerfHarness -id 1 -wt 10 -wi 0 -nt 10 -ss 10 -rl 120  -tc
amqp.Requestor -ot REQUEST -it REPLY -jp 5672
```

```
gr = The shared name used by all worker off-loaders.
id = Unique identifier number for this Requestor
wt = Timeout in seconds waiting for the thread to start,
ot = The outbound topic name
it = The inbound topic name
jp = The AMQP Channel port number
rl = runlength in seconds
ss = stats interval how often rate is printed,
nt = number of threads(clients)
wi = wait interval between starting each client in ms
```

Responder

The following starts a Responder with all threads performing a shared read of the outbound topic "REQUEST". The name of the shared group will be "SHARED"

```
java JMSPerfHarness -gr SHARED -wt 10 -wi 0 -nt 10 -ss 0 -rl 120 -tc
amqp.Responder  -ot REQUEST -jp 5672
```

```
gr = The shared name used by all worker off-loaders.
wt = Timeout in seconds waiting for the thread to start,
ot = The outbound topic name
it = The inbound topic name
jp = The AMQP Channel port number
rl = runlength in seconds
ss = stats interval how often rate is printed,
nt = number of threads(clients)
```

```
wi = wait interval between starting each client in ms
```

Command-line Parameter reference

The system is self documenting through the command-line. Use -h, -hf and -hm to learn about the functionality.

The following is a snapshot of the parameters of the tool, the latest lists and descriptions are always available using the tools help options:

com.ibm.uk.hursley.perfharness.Config

Centralises parsing, access and reporting of configuration parameters.

Arg	Default	Description
h	false	Display basic help on current configuration.
hm		Display detailed help on a specific module or modules. Specify multiple modules as space-separated tokens. Example: -hm "WebSphereMQ JNDI"
hf	false	Display detailed help on current configuration.
hx	false	Display help as XML.
hp		Config properties filename. Any values on the commandline take precedence over the contents of this file.
hs	true	If true, this enables strict registration and checking of all application parameters.
v	false	Show version.

com.ibm.uk.hursley.perfharness.Log

A proxy to output to stdout or stderr. There are currently no extensions to support
external logging, though this could be easily added. There are 5 levels (0-4) of
verbosity defined as (NONE, ERROR, WARNING, INFO, VERBOSE).

Arg	Default	Description
vo	4	Verbosity below which goes to stdout. The default is such that everything goes to stdout.
ve	0	Verbosity below which goes to stderr. The default is such that nothing goes to stderr.

com.ibm.uk.hursley.perfharness.ControlThread

Manage the lifecycle of the application and any WorkerThreads.
This also controls the aggregation and reporting of performance counters.

Arg	Default	Description
wk	120	Shutdown wait (s). The application will wait this long for WorkerThreads to shutdown before exiting anyway.
nt	1	Number of WorkerThreads.
mu	false	Display memory usage. This reports the number of bytes in use. Freed memory awaiting garbage collection is not counted as "used".
ss	10	Statistics reporting period. Setting this to 0 will disable periodic reporting entirely.
sd	normal	Sets what is reported as totalDuration. "normal" = from 1st iteration to last iteration, excluding setup/takedown. "tlf" = Time to Last Fire, from start of main thread till last iteration completes (includes setup time but not takedown)
su	true	Display final summary. This setting is independant of the periodic statistics reporting.
sp	false	Display per-thread performance data.
rl	60	Run length in seconds. Setting this to 0 will disable the timer and run forever.
id		Process identfier. If set, this will be displayed in the statistics reporting. This is of use if you have to merge the output of more than one instance of the tool.
wi	1000	WorkerThread start interval (ms). This controls the pause between starting multiple threads.
sh	true	Use signal handler to trap SIGINT (CTRL-C).

com.ibm.uk.hursley.perharness.WorkerThread

Base class for all varieties of test. This class implements a general pacing algorithm for those tests that wish to use it. The performance overhead of this
is minimal.

Arg	Default	Description
tc	jms.r11.PutGet	Test definition class. This defines the actual type of WorkerThreads that will be started. The selections listed are those packaged with this tool, there may be others on the classpath that will not be shown here. Known modules include • jms.r11.Sender • jms.r11.Receiver

		• jms.r11.Requestor • jms.r11.Responder • jms.r11.Publisher • jms.r11.Subscriber • mqjava.Sender • mqjava.Receiver • mqjava.Requestor • mqjava.Responder • mqjava.XAResponder • amqpRequestor • amqpResponder
rp	0	Time period (s) to ramp up to the full rate.
rt	0	Desired rate (operations/sec). If this rate is greater than the maximum achievable, the behaviour is such that it runs as fast as possible. A value of 0 means to always run as fast as possible. Rates of <1 op/sec are not currently possible.
mg	0	Fixed number of iterations to run. The default setting of 0 means there is no iteration limit.
yd	0	Frequency to call Thread.yield(). This may be of use if the WorkerThreads are not being evenly scheduled.
df	DestinationFactory	JMS Destination factory class. Currently, there is only one option. Known modules include • DestinationFactory

com.ibm.uk.hursley.perfharness.jms.providers.JMSProvider

Abstract superclass of all JMS providers supported by this tool. Properties file: AbstractJMSProvider.properties

Arg	Default	Description
pc	JNDI	JMS provider class. Known modules include • WebSphereMQ • WMB • JNDI
tx	false	Transactionality.
am	1	JMS acknowledgement mode. 1 = auto acknowldege, 2 = client acknowledge (not currently supported), 3 = dups_ok acknowledgement.

us		Username to authenticate as.
pw		Password to authenticate as.
to	5	Polling timeout on receiving messages. Threads will not stop if this timeout occurs, it simply the polling interval.
pp	false	Use persistent messages.
du	false	Durable subscriptions. Note, if using more than one JVM, these names will clash. To avoid this, use the -id parameter to differentiate the JVMs.
un	true	Unsubscribe subscribers when closing. Set this to false to leave durable subscriptions after the tool exits.
cc	1	Commit count (transaction batching). The number of operations completed within a single transaction. This only applies to test classes which only normally perform a single operation (such as Sender or Subscriber).
cd	0	Time in milliseconds to wait between JMS send and commit (commit delay). Can be used to introduce long running transactions. Only applies to PutGet and Sender classes.
cdm	false	Print message before each delay of commit (if cd is set). This is useful for cases where you want to test scenarios where there are lots of live transactions when an event occurs (e.g. so might want shutdown the queue manager when these messages appear).

com.ibm.uk.hursley.perfharness.jms.DefaultMessageFactory

This manages the type, size and use of the messages used in the JMS test classes.

Arg	Default	Description
ms	1000	Message size in bytes.
mf		External file to use as message contents.
mt	text	Message type (text,bytes,stream,map,object,empty).
co	true	Use correlation-id. This will use the correlation-id per thread model.
cp	true	Use provider-specific correlation-id if possible.

pf		

com.ibm.uk.hursley.perfharness.jms.DestinationFactory

This handles destinations for publish-subscribe and point-to-point domains. These options only control the **names** given to destinations. Specifying "-d TOPIC" does not enable publish-subscribe ("-tc jms.r11.Publisher -d TOPIC" does that)
Examples:
-d QUEUE
All threads operate on destination named QUEUE
-d MYTOPIC -dn 3
destinations are distributed round-robin in the order MYTOPIC1..MYTOPIC3
-d MYTOPIC -db 6 -dn 3
destinations are distributed round-robin in the order MYTOPIC6..MYTOPIC8
-d MYTOPIC -dx 6 -dn 3
destinations are distributed round-robin in the order MYTOPIC4..MYTOPIC6
-d MYTOPIC -db 4 -dx 6 -dn 5
destinations are distributed round-robin in the order MYTOPIC4..MYTOPIC6 starting with MYTOPIC5

Arg	Default	Description
db	0	Multi-destination numeric base.
dn	0	Multi-destination numeric range.
d	QUEUE	Destination prefix. If no other destination parameters are set, then nothing will be appended to this.
dx	0	Multi-destination numeric maximum.

com.ibm.uk.hursley.perfharness.jms.r11.PutGet

Sends a message then receives one from the same queue. Normal usage is with
correlation identifier to ensure the same message is received.

com.ibm.uk.hursley.perfharness.jms.r11.Sender

Send messages to a Queue.

com.ibm.uk.hursley.perfharness.jms.r11.Receiver

Receives messages from a Queue. Currently this class, although JMS 1.1 compliant, is
only coded to accept Queue-domain messages. Use the Subscriber class for topic-domain messages.

Arg	Default	Description
as	false	Use asyncronous (onMessage) receiving mode.

com.ibm.uk.hursley.perfharness.jms.r11.Requestor

Puts a message to a queue then waits for a reply on another queue. The same
correlation-id is used for every request. This is much faster for JMS applications.

Arg	Default	Description
iq	REQUEST	Queue to place requests on.
oq	REPLY	Queue to place replies on. Setting this value to "" implies the use of temporary queues for each reply. Correlation-ids are not used in this mode.

com.ibm.uk.hursley.perfharness.jms.r11.Responder

Takes messages off the request queue and places the same message on the reply queue.

Arg	Default	Description
iq	REQUEST	Queue to place requests on.
oq	REPLY	Queue to place replies on. Setting this value to "" causes the use of temporary queues for each reply, using an anonymous MessageProducer.

com.ibm.uk.hursley.perfharness.jms.r11.ReconnectTimer & com.ibm.uk.hursley.perfharness.jms.r11.AutoReconnectTimer

Special classes (based on the PutGet class above) to time reconnection of threads after an MQ switch/fail-over, in an MIQM or RDQM HA environment. Multiple queue managers (via a ccdt), or a pair of queue managers (using the addition h2 & (optionally) p2 parameters) can be specified.

AutoReconnect timer is only useful to test JMS automatic reconnection in your environment, as many, if not all threads will fail-over silently, resulting in no re-connect times being reported. Use ReconnectTimer to get times for all threads.

If you use h2 & p2, then the QM defined by jh & jp must be available at the start of the test.

Arg	Default	Description

h2		Hostname/IP of secondary host machine (not used if ccdt is specified)
p2	Value of jp	Port of secondary host machine (not used if ccdt is specified)
ft	U	Reconnection timestamp format (u=UTC l=local timezone)

com.ibm.uk.hursley.perfharness.jms.providers.WebSphereMQ

Settings for direct connection to a WMQ broker.
This allows the tool to be run directly against this provider without the need for JNDI
Note that this module inherits from JNDI module so those parameters are still applicable
and that all parameters of this module be ignored if you are use JNDI.

Arg	Default	Description
an		Sets the application name in the client connection factory (MQ APPLTAG in runmqsc). If not specified here the application name will be 'JMSPerfharness'.
ar		Sets the client connection factory reconnect option. Options are WMQ_CLIENT_RECONNECT_AS_DEF WMQ_CLIENT_RECONNECT WMQ_CLIENT_RECONNECT_DISABLED or WMQ_CLIENT_RECONNECT_Q_MGR. Any option other than null (the default) or WMQ_CLIENT_RECONNECT_DISABLED will require a ccdt to be used.
ccdt	Not defined (no ccdt)	MQ Client channel definition table (CCDT) URL. This will be used to set the client channel, if defined. Parms jh, h2, jp, p2, jc and jl will all be ignored if specified.
jg	false	Communicate with non-JMS application (targetClient=1) Setting this to true will cause the JMS client to send or recieve messages without RFH2 headers. This is

		primarily for communication with MQI applications. Certain JMS functionality is not available.
js	false	Use JMS reliable messaging. The queue manager needs altering before this will be enabled.
jq	SYSTEM.BROKER.DEFAULT.STREAM	Publish queue. This defines the stream for a WMQ broker, and can therefore be set on both publishers and subscribers. Note that publications cannot cross streams. Never cross the streams!
jp	1414	Port of provider host machine (ignored if ccdt is specified).
jh	localhost	DNS/IP of provider host machine (ignored if ccdt is specified).
jl	Not defined (no SSL)	SSLCipherSuite name. This controls the SSL encryption methodology (ignored if ccdt is specified).
jx	false	Use optimistic publication.
jo	true	Use JMS connection pooling.
jr	false	Force compatibility with WMQ 5.3 older than CSD 6. Forces use of QueueSubscriptionStore.
jc	SYSTEM.DEF.SVRCONN	WMQ Channel to connect to (ignored if ccdt is specified).
jt	mqc	WMQ transport (mqb, mqc). "mqb" is local-bindings connections, "mqc" is TCP/IP connections.
ja	-1	Publish acknowledgement interval (-1 = jms default). The maximum messages that can be placed on the publish queue is 1.5 * ja, the publisher will then be waiting for an acknowledgement from the broker.
ju	true	Use unique-queue-per-subscriber.
jb	QM	WMQ queue manager to connect to.

com.ibm.uk.hursley.perfharness.jms.providers.WMB

Settings for direct connection to a WebSphere Business Integration Message/Event Broker.
This allows the tool to be run directly against this provider without the need for JNDI.
Note that this module inherits from JNDI and WebSphereMQ modules so

those parameters are
still applicable and that all parameters of this module be ignored if you are use JNDI.

Arg	Default	Description
jq	PUBLISH	Publish queue. This sets the parameter equivalent to the BROKERPUBQUEUE property with JMSAdmin. For more information see the WMQ "Using Java" Manual. The value of this should be set to the value of the Queue set on the MQInput Node in your PubSub message flow.
jz	1000	Subscriber buffer size in number of messages.This sets the parameter equivalent to the MAXBUFFSIZE property with JMSAdmin. For more information see the WMQ "Using Java" Manual.
jt	mqc	WMQ transport (mqb,mqc,ip,ipmcr or ipmcn).This sets the parameter equivalent to the TRANSPORT property with JMSAdmin. For more information see the WMQ "Using Java" Manual. "mqb" is queued operation using local-bindings connections, "mqc" is queued operation client connections, "ip" is direct connection to the broker using the real-time transport, "ipmcr" is direct connection to the broker using the reliable multicast transport (this is equivalent to setting the MULICAST JMSAdmin property to RELIABLE), "ipmcn" is direct connection to the broker using the non-reliable multicast, (This is equivalent to setting the MULICAST JMSAdmin property to NOTR)

com.ibm.uk.hursley.perfharness.jms.providers.JNDI

Provider-independant access to JMS resources. If using JNDI, all destination names will be interpreted as the lookup name rather than the absolute name. Other JMS providers extend the JNDI module. If using any of the three JNDI parameters below, all other provider specific settings are ignored. All three parameters three must be specified together.

Arg	Default	Description
cf		JNDI name of ConnectionFactory. Using this parameter implicitly turns on JNDI for whatever provider class you are using.
iu		JNDI providerURL. Examples: file:/C:/JNDI-Directory, ldap://polaris/o=ibm,c=us.
ii		JNDI initialContextFactory. Examples: com.sun.jndi.fscontext.RefFSContextFactory, com.sun.jndi.ldap.LdapCtxFactory, com.ibm.websphere.naming.WsnInitialContextFactory.

iz		Set to true when using Suns JRE for WAS communication set into HashTable provided to InitialContext Additional properties need to be set when setting up InitialContext (default: false)

com.ibm.uk.hursley.perfharness.mqjava.MQProvider

Provides access to WebSphere MQ classes for Java

Arg	Description
jp	Port to connect to. (default: 1414)
jh	Host to connect to. (default: localhost)
mc	MQ Message Character Set. This value is set on the MQ Message in the MQMessage.characterSet field. (default: 1208)
rf	Use RFH Header from msg file, as produced by RFHUtil. (default: 0) "rf" value of 1 means read as RFH1 header (MQC.MQFMT_RF_HEADER_1), Value of 2 means RFH2 header (MQC.MQFMT_RF_HEADER_2).
jc	WMQ Channel to connect to. (default: SYSTEM.DEF.SVRCONN)
jt	WMQ transport (mqb, mqbf, mqc). (default: mqc) "mqb" is local-bindings connections, "mqbf" is local fastpath connections, "mqc" is TCP/IP connections.
bf	Force MQOO_BIND_NOT_FIXED for clusters. (default: false) This is normally done by changing the queue definition to DEFBIND(NOTFIXED).
ir	Port to connect to. (default: 1)
mq	MQ message format. This value is set on the MQ Message in the MQMessage.format field. (default: MQC.MQFMT_STRING) The value set should be the actual string that is the constant in the MQC.MQFMT* class, For example to specify MQC.MQFMT_TRIGGER then set -mq MQTRIG
me	MQ Message Encoding. This value is set on the MQ Message in the MQMessage.encodingfield field. (default: 546)
jb	WMQ queue manager to connect to. (default: QM)
jl	SSLCipherSuite name (default:) This controls the SSL encryption methodology.

com.ibm.uk.hursley.perfharness.mqjava.PutGet

Sends a message then receives one from the same queue. Normal usage is with
correlation identifier to ensure the same message is received.

com.ibm.uk.hursley.perfharness.mqjava.Sender

Send messages to a Queue.

com.ibm.uk.hursley.perfharness.mqjava.Receiver

Receives messages from a Queue.

com.ibm.uk.hursley.perfharness.mqjava.Requestor

Puts a message to a queue then waits for a reply on another queue. The same
correlation-id is used for every request.

Arg	Default	Description
iq	REQUEST	Queue to place requests on.
oq	REPLY	Queue to place replies on.

com.ibm.uk.hursley.perfharness.mqjava.Responder

Takes messages off the request queue and places the same message on the reply queue.

Arg	Default	Description
iq	REQUEST	Queue to place requests on.
oq	REPLY	Queue to place replies on.

com.ibm.uk.hursley.perfharness.mqjava.XAResponder

Takes messages off the request queue, performs a database update, then places
the same message on the reply queue.

Arg	Default	Description
iq	REQUEST	Queue to place requests on
ds	COM.ibm.db2.jdbc.DB2XADataSource	XA Data source provider class or com.ibm.db2.jcc.DB2DataSource for Type 4 driver
oq	REPLY	Queue to place replies on

com.ibm.uk.hursley.perfharness.http.HTTPProvider:

Arg	Description
hc	Specify a value of the content type header (default: text/xml)
nm	Number of messages to send for each thread (default: 0).
cs	Close socket after each message. Set to use a new HTTP connection for each request. (default: false)

rb	Size of the receive buffer in bytes in which the HTTP Reply message is read. This MUST be big enough for the reply message. (default: 10000)
ur	URL of servlet to send data to (default: "/")
jp	HTTP Listener Port to connect to. (default: 7080)
jh	Hostname of server to connect to. (default: localhost)
wo	Number of messages sent before a reponse message is written to a file (default: 0)
mc	Number of HTTP requests to send before renewing an HTTP persistent connection. (default: 0) If this value is 0 then the HTTP connection is never closed
sa	Specify a SOAPAction: Tag in the header.The value is the value of the SOAPAction field. (default:)

com.ibm.uk.hursley.perfharness.amqp.Resquestor:

com.ibm.uk.hursley.perfharness.amqp.Responder:

Arg	Default	**Description**
-jh	localHost	Host name of the AMQP Channel
-jp	5672	Port number of the AMQP Channel.
-qs	1	Quality of service: 0 = At most once 1 = At least once
-ttl	0	The "time-to-live" in seconds before message is discarded.
-qt		Shared topic has thread id append.
-ot		The body name for the outbound shared topic.
-mi	1	Number of messages sent and received in an iteration.
-id		Unique index for this harness. Used in generating unique topics. Required when running multiple harness.
-fl		Harness will terminate on first lost of connection with AMQP Channel.
-it		The body name for inbound unique topic.
-ms	1000	Message size in bytes.
-gr	Disabled	The name of the share used by a collection of worker off-loaders.
-mf		File name to generate message contains with. Default generate internally.
-dm	string	Message type, options are "string" or "buffer".

Troubleshooting

Please check the "known issues" section in the release notes.

- **Invalid parameter: Parameter [??] is not known/valid in this configuration**.
  Parameters in this tool belong to specific modules. If that module is not loaded, no knowledge of its parameters exists. If you look at the help for the current context (parameter "-h"), you will see that the corresponding module is not included. Check your "-pc" and "-tc" settings, you are probably not referencing the correct module. A full list of options for these parameters is given in this manual or by

parameter "-hf". In particular, check the *case* of "-pc WebSphereMQ" as this is a common fault.

- **Invalid parameter: Destination range is negative**.
  You have either set the minimum value ("-db") greater than the maximum ("-dx") or have used a combination of "-dx" and "-dn" which implies a negative starting range. Consult the HOWTO on multiple destinations.

- **Why is it printing ugly stack traces**.
  This tool is intended to be a utility, as such it is a deliberate choice to print out all stack traces by default, these may contain the information that helps solve the problem. If the problem is a simple one such as a misspelt parameter, these will be seen in the first few lines of output. Staqc traces can be turned off by using "-st false".

Requesting help

If these few tips do not answer your query, or you have a suggestion for improvements, please ask on the alphaWorks forum page for this product. When submitting a problem (particularly a crash report) then please do the following to help us help you:

- Run the tool with "-vo=ALL" to turn on as much debugging output as possible.
- Include the commandline invocation you used to run the program.

Acknowledgements

We would like to acknowledge the contribution of the original tool author Marc Carter who has now left IBM for his work on which this release is building upon.

Feedback

Feedback can be given on github : https://github.com/ot4i/perf-harness