WebSphere Application Server for z/OS

Some Useful Plug-ins for the WebSphere Application Server SMF Record Browser

Version Date: December 19, 2018
See "Document change history" on page 17 for a description of the changes in this version of the document

IBM Software Group
Application and Integration Middleware Software

Written by David Follis
IBM Poughkeepsie
845-435-5462
follis@us.ibm.com

Many thanks go to ...Don Bagwell, John Hutchinson, Mike Loos, Ed McCarthy and the many customers whose data I've looked at with these

Introduction

The Whitepaper WP101342, “Understanding SMF Record Type 120, Subtype 9” discusses the WebSphere SMF record in detail. That paper also briefly discusses the SMF Browser provided by WebSphere development. The browser can accept plug-ins to perform special analysis and reporting. This paper (and the accompanying .jar file) is just a collection of plugins that I've written/used that have helped me. I thought I would share them in case they are useful to somebody else. See the copyright at the top of each source file for appropriate disclaimers.

This document goes with the 6/26/2013 version of the accompanying .jar file (see the history.txt file inside the .jar).

See also W102311 where I used these plugins to look through some sample data and see what we could learn.

Update 10/11/13 – Doc updates for changes to the plugins shipped on this date are flagged just like this line.

Update 10/28/13 - Doc updates for changes to the plugins shipped on this date are flagged just like this line.

Update 11/30/18 – Added JCL invocation support

Getting and Using the Browser

In case you don't already have the browser, I've included this section from WP101726 which covers writing your own plugins.

The browser can be downloaded from this URL:

https://www.ibm.com/services/forms/preLogin.do?source=zosos390&S_PKG=smf5

If you do not already have an id, you can easily register and create one. When you get to the download page, you should be able to select the “SMF Browser for WebSphere Application Server”. It works for any release of WebSphere.

The result of the download will be the file `bbomsmfv.jar`. You will want to copy `bbomsmfv.jar` to a directory in your USS file system on z/OS. The .jar file includes the executable, the source code for the browser, some documentation, license material, and a change history file.

After you have the `bbomsmfv.jar` file in your USS file system, you can use the `.jar` file to view the SMF 120 records in an SMF dump dataset. For the rest of this paper we will assume you have run some WebSphere requests through a Version 7 (or up) server with SMF type 120, subtype 9, recording enabled and dumped the SMF data into an MVS dataset called `‘WAS.SMF.DATA’`.

To run the browser you need to first set the classpath. The browser is dependent on another .jar file to read the MVS dataset where the SMF data exists. That `.jar` file is called `ibmjzos.jar`. The `.jar` file should be part of any z/OS JVM you have installed (there is a copy in the Java `lib/ext` directory where WebSphere is installed).

If both `.jar` files are in your current directory, this command will run the browser with the default plug-in (on one line):

```
java –cp bbomsmfv.jar:ibmjzos.jar
                              com.ibm.ws390.sm.smfview.SMF “INFILE(WAS.SMF.DATA)”
```

You might consider putting that into a file, we will call the file `smf.sh`, and use `chmod` to make the file executable with the command:

```
chmod 700 smf.sh
```

Now you can just type `smf.sh` to run the browser. The shell script could also export the jar files to the classpath. To use the plugins you will also need to put the their `.jar` file on the classpath. In the examples that follow we show a shell script exporting CLASSPATH.

Running the Browser from JCL

There is a different interface which allows you to more easily invoke the browser from JCL. You can do this by exploiting the JZOS launcher. Here's some sample JCL that could be used:

```
//SMF1 EXEC PROC=JVMPRC86,
//  JAVACLS='com.ibm.ws390.sm.smfview.JclSmf'
//*
//* Change to the dataset containing the SMF data
//SMFDATA DD DISP=SHR,DSN=FOLLIS.WSC.SMF.DATA
//*
//* Sets up where .jar files are etc.
//STDENV DD DISP=SHR,DSN=FOLLIS.SMF.JCL(SMFENV)
//*
//* Options to the SMF utility
//SMFENV  DD DISP=SHR,DSN=FOLLIS.SMF.JCL(RPS)
```

The SMFDATA DD points to the dataset where the SMF data to be processed resides.

The STDENV DD points to a script which sets up variables required to run the browser. This includes things like LIBPATH and CLASSPATH. Here's a sample:

```
#This is a shell script which configures env-vars for the launcher

#Export where the browser and plugin jars live
CLASSPATH=/u/follis/bbomsmfv.jar
CLASSPATH="$CLASSPATH":/u/follis/smf/WP102312_Plugins.jar
export CLASSPATH="$CLASSPATH"

#Export wherever Java lives
export JAVA_HOME=/usr/lpp/java/J8.0_64

# Configure JVM options - Uncomment if you need to change heap size
#IJO="-Xms1024m -Xmx1024m"
#export IBM_JAVA_OPTIONS="$IJO

# Other required exports
export PATH=/bin:"${JAVA_HOME}"/bin
LIBPATH=/lib:/usr/lib:$"${JAVA_HOME}"/bin
LIBPATH="$LIBPATH":"${JAVA_HOME}"/lib/s390
LIBPATH="$LIBPATH":"${JAVA_HOME}"/lib/s390/j9vm
LIBPATH="$LIBPATH":"${JAVA_HOME}"/bin/classic
export LIBPATH="$LIBPATH":
```

The SMFENV DD points to input name/value properties to be used by the browser. Recognized properties include:

plugin – specifies the plugin to be used. If the package is com.ibm.ws390.smf.plugins then you can leave it out (i.e. just specify RequestsPerServer) and that package will be assumed.

output – the zFS file to use for output (or the fully qualified dataset name for ReWrite)

matchServer – for 120-9 records, only records from the named server will be processed

matchSystem – for 120-9 records, only records from the named z/OS image will be processed

excludeInternal – set to true, some reports will ignore 120-9 records for internal requests

Here's a sample:

```
# Specify the plugin to use
plugin=RequestsPerServer

# Specify where the output goes
# Usually a zFS file (/ecurep/...)
# or a fully qualified datset name for ReWrite
output=/u/follis/jclSMF.txt

# Uncomment (and change the value as appropriate) to filter
matchServer=XDSR01B
```

Plugin: RequestsPerServer

This is a good report to run if somebody just hands you a big pile of WebSphere SMF data. I always run this first on any customer SMF data I receive. It gives me a quick overview of how much data I have and what it represents.

The plugin looks at each 120-9 record and remembers all the different servers that it finds. It keeps track of the different types of work it sees for each server (e.g. HTTP, IIOP, etc.). When it is finished it produces a report that shows the system name and server name for each server with the counts of the different work types it saw. The data is provided in comma-separated-value format (CSV) which is easily pulled into most spreadsheet programs. Here is some very basic output from a dump of SMF data from just one server.

System	Server	Requests	Unknown	IIOP	HTTP	HTTPS	MDBA	MDBB	MDBC	SIP	SIPS	MBEAN	OTS	OTHER	WOLA	ASYNC
SYSC	QZSR01C	1001	0	0	1000	0	0	0	0	0	0	1	0	0	0	0
Totals		1001	0	0	1000	0	0	0	0	0	0	1	0	0	0	0

If you have data for a lot of different servers a lot of internal requests (Mbeans, etc) then you might want to use the SMF 120-9 filter properties with the browser to thin out the data it handles for the other plugins.

Note that async work will only be recorded if you are running on WAS V8, have async beans running, and have SMF recording for async work enabled.

Update 10/11/13 – This report now also shows you the code level of the server (e.g. 8.0.0.7) as well as the earliest and latest timestamps for records seen for this server. The level is based on the first record we see for a particular server so if the data spans a migration of levels it will just report the first one it sees.

Update 10/28/13 - This report now also counts the number of SMF 120-10 records (outbound requests) and reports it to the right of the ASYNC count.

Example:

```
export CLASSPATH=~/smfplugins/bbomsmfv.jar:~/smfplugins/WP102312_plugins.jar
java com.ibm.ws390.sm.smfview.SMF "INFILE(WAS.SMF.DATA)"
                             "PLUGIN(com.ibm.ws390.smf.plugins.RequestsPerServer,
                                            ~/smfplugins/v8_reqs_per_server.csv)"
```

Plugin: CSVExport

This plugin generates a file with one record per SMF 120-9 record processed. Most of the fields from the SMF record are turned into comma-separated-values in the file. The time stamps are presented in both human-readable values and `STCK` values converted to milliseconds. Math is done with the millisecond values to calculate the time spent in each stage (e.g. in the WLM queue).

Records for async work are ignored.

This is a good report to use the server name filter mentioned above (and maybe the no-internal work filter too) if you have a lot of data.

I'm not going to include a sample of the output because its very wide.

Update 10/28/13 - This plugin now reports the EJB and WOLA classification information for those request types, in addition to the already-supported HTTP request information (host/port/URI).

Example, with filtering by servername and excluding internal requests:

```
export CLASSPATH=~/smfplugins/bbomsmfv.jar:~/smfplugins/WP102312_plugins.jar
java  -Dcom.ibm.ws390.smf.smf1209.ExcludeInternal=TRUE
                                 -Dcom.ibm.ws390.smf.smf1209.MatchServer=qzsr01c
                              com.ibm.ws390.sm.smfview.SMF "INFILE(WAS.SMF.DATA)"
                                     "PLUGIN(com.ibm.ws390.smf.plugins.CSVExport,
                                                  ~/smfplugins/smf_records.csv)"
```

Plugin: LibertyExport

This plugin is the equivalent of CSVExport for the SMF 120-11 Version 2 records written by Liberty for incoming HTTP requests as of 16.0.0.2. For the Version 1 records see the zOSConnectCSV later in this document.

Example:

```
export CLASSPATH=~/smfplugins/bbomsmfv.jar:~/smfplugins/WP102312_plugins.jar
java com.ibm.ws390.sm.smfview.SMF "INFILE(WAS.SMF.DATA)"
      "PLUGIN(com.ibm.ws390.smf.plugins.LibertyExport,
      ~/smfplugins/smf_records.csv)"
```

Plugin: LibertyBatchExport

This plugin is the equivalent of CSVExport for the SMF 120-12 records written by Liberty for JSR-352 Java Batch job execution. Records are written at step and job end (which record type is indicated by a field in the record).

Example:

```
export CLASSPATH=~/smfplugins/bbomsmfv.jar:~/smfplugins/WP102312_plugins.jar
java com.ibm.ws390.sm.smfview.SMF "INFILE(WAS.SMF.DATA)"
       "PLUGIN(com.ibm.ws390.smf.plugins.LibertyBatchExport,
       ~/smfplugins/smf_records.csv)"
```

Plugin: zOSConnectCSV

This plugin does the same thing as CSVExport, except that it processes the SMF 120-11 Version 1records written by Liberty when you use z/OS Connect. Each record is converted into a single comma-separate-value line in the output file. Timestamps are presented in both human-readable and as a millisecond value so you can do math. Math is done for you to give you response time.

NOTE: z/OS Connect EE writes SMF 123 records which are not handled by bbomsmfv.jar or any of the plugins here. What we're talking about here is the SMF records written by the audit interceptor that comes with Liberty itself.

Example:

```
export CLASSPATH=~/smfplugins/bbomsmfv.jar:~/smfplugins/WP102312_plugins.jar

java  com.ibm.ws390.sm.smfview.SMF "INFILE(WAS.SMF.DATA)"
                     "PLUGIN(com.ibm.ws390.smf.plugins.zOSConnectCSV,
                                                              ~/smfplugins/zosConnect.csv)"
```

Plugin: ReWrite

If you have a huge volume of data and you are using the filter properties as shown above to thin out the amount being processed, it might be nice to just have less data. For some of the more complex reports it can speed things up considerably if we don't have to read, parse, and then ignore a lot of records we don't care about.

This plugin simply reads in all the records that get past the filter properties and writes them back out to a pre-allocated `dataset`.

Allocate the new `dataset` with the same attributes as the original `dataset`.

Example, copying records from one server to a new dataset:

```
export CLASSPATH=~/smfplugins/bbomsmfv.jar:~/smfplugins/WP102312_plugins.jar
java -Dcom.ibm.ws390.smf.smf1209.MatchServer=qzsr01c
                              com.ibm.ws390.sm.smfview.SMF "INFILE(WAS.SMF.DATA)"
                 "PLUGIN(com.ibm.ws390.smf.plugins.ReWrite,WAS.SMF.DATA.QZSR01C)"
```

Update 10/28/13 - This plugin now supports copying SMF 120-10 records as well as SMF 120-9 records.

Plugin: SplitByServer

Similar to ReWrite, this plugin tries to automatically break the data up by server name (a common usage patter with ReWrite). Instead of telling it which server you are interested in, just have DD cards whose name matches the server (or servers) for which you want data extracted.

For example, if you have data from 100 different servers in one SMF data file but only care about data from SERVER1 and SERVER2 then run the browser with DD's allocated named SERVER1 and SERVER2. Data for those servers will be written into the appropriate file and data from other servers will be ignored.

The plugin will print messages indicating which servers it is keeping (and not keeping) data for. So running with no DDs matching server names is kind of a cheap way to get a list of all the servers for which you have data..

Plugin: RequestDensity and RequestDensity2

These two plugins look at the timestamps for each request and manage counters for the state the request was in at one second intervals. We do this two ways. The first plugin, RequestDensity, just increments a counter (e.g. the in-queue-in-this-second counter) when the request enters this state in this second. So, for example, if a request gets put on the queue at 10:30:29 and stays there for 3 seconds, the “In Q” counter only gets bumped for that one second. So the report shows you how many requests ENTERED each state during that second.

The second plugin, RequestDensity2, looks at the windows where the request was in each state and increments the counters for all the seconds the request was in that state. So in our example above, the “In Q” counter would be incremented for 10:30:29, 10:30:30, and 10:30:31. So the resulting report shows you how many requests were IN each state during each second.

Note that with one-second granularity it is very possible for a single request to be in all four states in the same second, as can be seen in the sample data below. This report ignores async work. Use the ExcludeInternal filter to ignore internal work.

Other weirdness... Suppose you know you have 40 dispatch threads and you know you are running the server at full capacity. You would then expect to see “In Disp” at 40 most of the time. But remember it can only report using data it has. If a request started running on a thread and finished AFTER the cutoff for your set of SMF data, then the plugin doesn't have that SMF record. So if one request was on a thread for the last 5 minutes of your data, but didn't finish until just after the cutoff for your chunk of SMF data, then the report might show 39 requests “In Dispatch” for those last five minutes, but really there were 40. We can't know what the other thread was doing until it finishes and writes its SMF record. This is probably obvious, but I stared at one of these reports for quite a while before I figured out where that other thread went.

Update 10/11/13 – I decided to experiment with exploiting Chart.js (you can get a copy here http://www.chartjs.org/). If you use RequestDensity2 and specify an output file that ends in “.html” the plugin will generate HTML/Javascript. If you put that html file and chart.js in the same directory and open the HTML file with your browser it will probably (no guarantees for different browser support) show you a nice graph of how work is being processed. The blue line is work arriving in the controller, the yellow line is work in the queue, the red line is work in dispatch, and the orange line is work in completion. As the note below points out, the data produced is skipping seconds in which nothing happens. So if the server is idle (or down) there should be a gap but the data and graph won't show it. Also, due to limitations in Chart.js, we are only putting 600 data points (10 minutes) per chart. So if you have a larger data range you'll get multiple charts. And if you have ten minutes (or a multiple) and a little bit you'll get a final graph with very few data points on it which looks a little weird. Basically this support is experimental. When in doubt, look at the actual data. But sometimes the picture helps.

```
Time                                     In CR  In Q  In Disp  In Comp
Friday  June 21  2013 3:55:07 AM GMT         1     1        0        0
Friday  June 21  2013 4:01:25 AM GMT         9     9        9        9
Friday  June 21  2013 4:03:07 AM GMT         2     2        2        2
Friday  June 21  2013 4:03:08 AM GMT         1     1        2        2
Friday  June 21  2013 4:03:09 AM GMT        42    42       42       42
Friday  June 21  2013 4:03:10 AM GMT         2     2        2        2
Friday  June 21  2013 4:03:17 AM GMT         2     2        2        2
Friday  June 21  2013 4:03:21 AM GMT         2     2        2        2
```

Example:

```
export CLASSPATH=~/smfplugins/bbomsmfv.jar:~/smfplugins/WP102312_plugins.jar
java com.ibm.ws390.sm.smfview.SMF "INFILE(WAS.SMF.DATA)"
                                "PLUGIN(com.ibm.ws390.smf.plugins.RequestDensity,
                                               ~/smfplugins/v8_req_density.csv)"
```

Note: Timestamps only appear for times where data exists. There can be gaps. It seemed better to have to look for the gaps than have potentially huge swaths of zeros where nothing happened.

Plugin: ThreadRequestDensity and ThreadRequestDensity2

This pair of plugins track when requests begin (or end) dispatch on each thread in each servant region. A report is produced, in CSV format, showing minute-by-minute for each servant region how many requests started (or ended) dispatch on the available threads.

ThreadRequestDensity uses the dispatch begin timestamps and ThreadRequestDensity2 uses the dispatch end timestamps. Both reports ignore async work and internal work. Internal work is ignored automatically since it runs on a separate thread pool.

Update 10/11/13 – Both of these reports have been updated to include a count on the far right showing the number of non-zero thread columns. Basically the count of threads that begin/ended work in this minute. This can give you a quick look at how many threads are being used by the server.

Remember that, like other reports, this one skips rows where nothing happened. So if no work started/ended in a given minute (because the server was idle, the server was down, or requests just spanned multiple minutes) then there won't be rows for those minutes. So if you are looking at thread usage over time, be careful about gaps.

Here is some sample output:

```
Data for Server QZSR01C in servant with stoken 3200000002F
Day      Hour  Minute  8CE4F8  8CC108  8CCE88  8CBC68  8CB828  8CC328  8CCC68  Total
     159   19      20       0       0       0       0       0      55      59    114
     159   19      21      74       1     313      65     183     194      56    886
Total                      74       1     313      65     183     249     115   1000
```

Example invocation:

```
export CLASSPATH=~/smfplugins/bbomsmfv.jar:~/smfplugins/WP102312_plugins.jar
java com.ibm.ws390.sm.smfview.SMF "INFILE(WAS.SMF.DATA)"
                         "PLUGIN(com.ibm.ws390.smf.plugins.ThreadRequestDensity,
                                            ~/smfplugins/v8_thread_req_density.csv)"
```

Plugin: DispatchThreadUsage

This report determines the total time for which it has SMF data for work executed in each servant region. It then examines the dispatch begin/end time for every request and tracks the total time for which each dispatch TCB was actually dispatching something. It also tracks CPU time and determines the total CPU time used by requests dispatched on each TCB.

A summary report, in CSV, shows how many requests were processed by each TCB and what percent of the total report time was spent in dispatch and what percent of the dispatch time was spent using CPU.

This report ignores async work and internal work. Internal work is ignored automatically since it runs on a different thread pool.

Update 10/11/13 – Just a note of caution...I've seen some weirdness with this one where it reported the CPU time as a percentage of elapsed time was greater than 100%. I don't understand how that happened. Might just be weirdness in the data I had or a bug in the plugin code. Just be careful.

Here's an example of the output:

Server	SR-Stoken	SR-STCname	TimeRange (ms)	TCB	UsedElapsed	UsedElapsedPercent	Used CPU	UsedCPUPercent	WorkCount
QZSR01C	3200000002F	STC01381	29362	8CE4F8	283	0	188	66	74
QZSR01C	3200000002F	STC01381	29362	8CC108	15	0	2	13	1
QZSR01C	3200000002F	STC01381	29362	8CCE88	1077	3	688	63	313
QZSR01C	3200000002F	STC01381	29362	8CBC68	258	0	172	66	65
QZSR01C	3200000002F	STC01381	29362	8CB828	715	2	455	63	183
QZSR01C	3200000002F	STC01381	29362	8CC328	971	3	601	61	249
QZSR01C	3200000002F	STC01381	29362	8CCC68	457	1	233	50	115

Example invocation:

```
export CLASSPATH=~/smfplugins/bbomsmfv.jar:~/smfplugins/WP102312_plugins.jar
java com.ibm.ws390.sm.smfview.SMF "INFILE(WAS.SMF.DATA)"
                           "PLUGIN(com.ibm.ws390.smf.plugins.DispatchThreadUsage,
                                     ~/smfplugins/v8_dispatch_thread_usage.csv)"
```

Plugin: FindHighWaterInQueue

How deep did the WLM queue get? This plugin tries to determine how bad it got, for the available data. It scans all the data and determines the earliest time something was placed on the queue and the latest time something was taken off. It then goes through that time range, millisecond by millisecond, looking at each request to see if it was on the queue during that millisecond. The report shows, for each millisecond in the window, how many requests were on the queue at that time. Async work is ignored.

Sounds very slow and could produce a huge volume of output. And it is both. To get around that, and because queue spikes probably last longer than a millisecond anyway, you can specify how many samples you want it to take across the interval. The default is 1000 samples.

The output will look like this:

```
Data for server QZSR01C
analyzing range 3579708056920 to 3579708086278 which is 29358 slots
Generating 1000 samples every 29ms
3579708056920,1
3579708056949,0
3579708056978,0
3579708057007,0
3579708057036,0
3579708057065,0
.
.
.
Highwater Queued is 2 at 3579708065417
```

The timestamps are STCK values in milliseconds. You should be able to match it to a real time (or close to one) in the CSVExport output.

You can adjust the number of samples by setting the property

`com.ibm.ws390.smf.smf1209.FindHighWaterInQueue.Interval.`

Example invocation:

```
export CLASSPATH=~/smfplugins/bbomsmfv.jar:~/smfplugins/WP102312_plugins.jar
java -Dcom.ibm.ws390.smf.smf1209.FindHighWaterInQueue.Interval=500
                              com.ibm.ws390.sm.smfview.SMF "INFILE(WAS.SMF.DATA)"
                          "PLUGIN(com.ibm.ws390.smf.plugins.FindHighWaterInQueue,
                                              ~/smfplugins/v8_queue_highwater.txt)"
```

Plugin: AffinityCreation

Version 8 of WebSphere Application Server includes information about affinities in the SMF 120-9 record. An affinity is usually created when an HTTP request creates an HTTPSession object (although there are other ways you can get an affinity). So I built a couple of reports to take a look at affinity data.

The first report looks at affinity creation over time. It groups affinity information by servant within a particular server. The server name and servant identifiers are in a header line. That is followed by CSV data showing the number of affinities created over time. You can count affinities per hour, per minute, or per second. Here's some sample output shown per minute for two minutes:

```
Data for Server null in servant STC01381 with stoken 3200000002F
Day   Hour   Minute   CreatedAff
  159      19        20            114
  159      19        21            886
```

And here's the same data shown per second:

```
Data for Server null in servant STC01381 with stoken 3200000002F
Day   Hour   Minute   Second   CreatedAff
  159      19        20         56              1
  159      19        20         57             24
  159      19        20         58             44
  159      19        20         59             45
  159      19        21          0             42
  159      19        21          1             43
  159      19        21          2             42
  159      19        21          3             42
  159      19        21          4             38
  159      19        21          5             41
  159      19        21          6             37
```

You can control the granularity of the report by setting this property to `Hours`, `Minutes`, or `Seconds`. The default is `Seconds`.

`com.ibm.ws390.smf.plugins.AffinityCreation.Interval`

Here's an example invocation yielding minute-level reporting:

```
export CLASSPATH=~/smfplugins/bbomsmfv.jar:~/smfplugins/WP102312_plugins.jar
java -Dcom.ibm.ws390.smf.plugins.AffinityCreation.Interval=Minutes
                              com.ibm.ws390.sm.smfview.SMF "INFILE(WAS.SMF.DATA)"
                              "PLUGIN(com.ibm.ws390.smf.plugins.AffinityCreation,
                                    ~/smfplugins/v8_AffinityCreationMinutes.csv)"
```

Plugin: AffinityReport

If you have an application that creates affinities it might be nice to know if they are actually being used. An affinity usually represents an actual HTTPSession object in memory in a servant region. If left unused there is an expiration timer that will delete them. However, they remain in the heap until that timer runs out. An application that is creating session objects that are never used is causing unnecessary overhead. On the other hand, just knowing how creating affinities are being used might tell you something.

If you are running on WAS V8 the SMF 120-9 record will contain affinity data. This report consists of one row of CSV data for each created affinity. For each affinity we report the creation time (in human-readable and STCK millisecond values), the server and servant where the affinity was created, the actual affinity token, the number of times the affinity was used after it was created, and the URI of the HTTP request that created the affinity.

The actual token isn't very useful, but it is what makes the rows unique. Here is some sample output:

Created	Created-ms	Server	ServantSTC	Aff-Token	UseCount	CreateURI
Thursday June 27 2013 5:59:07 PM GMT	3581344747007	QZSR02C	STC04665	C3C2F7F0F0F9C4C2F7C4F0F9F1C3F1F7F0F0F0F0F0F2F7C3F0F0F0F0F0F0F2F2F0F0F0F0F0F0F4F898E3D6E5C7F486A6F78988C396D8C5A5A8D1E598C2C6F8	499	/SuperSnoopWeb/SuperSnoop
Thursday June 27 2013 5:59:08 PM GMT	3581344748176	QZSR02C	STC04665	C3C2F7F0F0F9C4C2F7C4F0F9F1C3F1F7F0F0F0F0F0F2F7C3F0F0F0F0F0F0F2F2F0F0F0F0F0F0F4F8F09495A4F991E6D691F7F49488D684926DC6E5E394F689	499	/SuperSnoopWeb/SuperSnoop
Thursday June 27 2013 5:59:07 PM GMT	3581344747064	QZSR02C	STC04664	C3C2F7F0F0F9C4C2F7C4F0F9F1C3F1F7F0F0F0F0F0F2F7C3F0F0F0F0F0F0F2F2F0F0F0F0F0F0F4F8A6D7E7F7D381D2A596F68984D982D5F4A5F586F8C5E384	499	/SuperSnoopWeb/SuperSnoop
Thursday June 27 2013 5:59:05 PM GMT	3581344745309	QZSR02C	STC04664	C3C2F7F0F0F9C4C2F7C4F0F9F1C3F1F7F0F0F0F0F0F2F7C3F0F0F0F0F0F0F2F2F0F0F0F0F0F0F4F8D1F1D8E29189E8938160D3E4A9D9D8F994F6D284C1D6A4	499	/SuperSnoopWeb/SuperSnoop
Thursday June 27 2013 5:59:07 PM GMT	3581344747517	QZSR02C	STC04664	C3C2F7F0F0F9C4C2F7C4F0F9F1C3F1F7F0F0F0F0F0F2F7C3F0F0F0F0F0F0F2F2F0F0F0F0F0F0F4F89883C6D3E2E9F19360F0D4E9E3D4E4C594D6E4E2F6F1C3	499	/SuperSnoopWeb/SuperSnoop
Thursday June 27 2013 5:59:05 PM GMT	3581344745320	QZSR02C	STC04665	C3C2F7F0F0F9C4C2F7C4F0F9F1C3F1F7F0F0F0F0F0F2F7C3F0F0F0F0F0F0F2F2F0F0F0F0F0F0F4F882D4A497C993E3F086929392F587F6C4C491E7C5A4A5E3	499	/SuperSnoopWeb/SuperSnoop
Thursday June 27 2013 5:59:09 PM GMT	3581344749491	QZSR02C	STC04664	C3C2F7F0F0F9C4C2F7C4F0F9F1C3F1F7F0F0F0F0F0F2F7C3F0F0F0F0F0F0F2F2F0F0F0F0F0F0F4F895C6F5A4E4A5A5C5E6E2F894958294F7E899A9D7A9D5F7	499	/SuperSnoopWeb/SuperSnoop
Thursday June 27 2013 5:59:05 PM GMT	3581344745845	QZSR02C	STC04664	C3C2F7F0F0F9C4C2F7C4F0F9F1C3F1F7F0F0F0F0F0F2F7C3F0F0F0F0F0F0F2F2F0F0F0F0F0F0F4F86D9791A9F8D6F7E4C3C8866DD8F6F66093A460F3D6E9A7	499	/SuperSnoopWeb/SuperSnoop
Thursday June 27 2013 5:59:04 PM GMT	3581344744741	QZSR02C	STC04665	C3C2F7F0F0F9C4C2F7C4F0F9F1C3F1F7F0F0F0F0F0F2F7C3F0F0F0F0F0F0F2F2F0F0F0F0F0F0F4F8A6E9D2C9D1A3E6E4D9E287E3F5F7A4A3C1A5A8F7A4E789	499	/SuperSnoopWeb/SuperSnoop
Thursday June 27 2013 5:59:07 PM GMT	3581344747393	QZSR02C	STC04665	C3C2F7F0F0F9C4C2F7C4F0F9F1C3F1F7F0F0F0F0F0F2F7C3F0F0F0F0F0F0F2F2F0F0F0F0F0F0F4F8D28260F992A9E5F481F3C3A5E9D888D8A7C5F3C587E6F6	499	/SuperSnoopWeb/SuperSnoop

Its a little tough to read here. Deleting the affinity token column lets you see it a little better:

Created	Created-ms	Server	ServantSTC	UseCount	CreateURI
Thursday June 27 2013 5:59:07 PM GMT	3581344747007	QZSR02C	STC04665	499	/SuperSnoopWeb/SuperSnoop
Thursday June 27 2013 5:59:08 PM GMT	3581344748176	QZSR02C	STC04665	499	/SuperSnoopWeb/SuperSnoop
Thursday June 27 2013 5:59:07 PM GMT	3581344747064	QZSR02C	STC04664	499	/SuperSnoopWeb/SuperSnoop
Thursday June 27 2013 5:59:05 PM GMT	3581344745309	QZSR02C	STC04664	499	/SuperSnoopWeb/SuperSnoop
Thursday June 27 2013 5:59:07 PM GMT	3581344747517	QZSR02C	STC04664	499	/SuperSnoopWeb/SuperSnoop
Thursday June 27 2013 5:59:05 PM GMT	3581344745320	QZSR02C	STC04665	499	/SuperSnoopWeb/SuperSnoop
Thursday June 27 2013 5:59:09 PM GMT	3581344749491	QZSR02C	STC04664	499	/SuperSnoopWeb/SuperSnoop
Thursday June 27 2013 5:59:05 PM GMT	3581344745845	QZSR02C	STC04664	499	/SuperSnoopWeb/SuperSnoop
Thursday June 27 2013 5:59:04 PM GMT	3581344744741	QZSR02C	STC04665	499	/SuperSnoopWeb/SuperSnoop
Thursday June 27 2013 5:59:07 PM GMT	3581344747393	QZSR02C	STC04665	499	/SuperSnoopWeb/SuperSnoop

Clearly these affinity tokens aren't being re-used.

Here's an invocation example:

```
export CLASSPATH=~/smfplugins/bbomsmfv.jar:~/smfplugins/WP102312_plugins.jar
java com.ibm.ws390.sm.smfview.SMF "INFILE(WAS.SMF.DATA)"
                                "PLUGIN(com.ibm.ws390.smf.plugins.AffinityReport,
                                           ~/smfplugins/v8_AffinityReport.csv)"
```

Plugin: ResponseTimes

This plugin shows averages of elapsed and CPU times. It does this per URI for HTTP requests. For everything else it lumps it into 'null' (meaning the URI is 'null'...live with it). It also averages across all the requests seen.

So the report has the following columns:

– number of requests seen per URI
– average response time (completion minus received) per URI
– average queue time per URI
– average dispatch time per URI
– average TCB CPU (in ms) per URI
– average bytes received per URI
– average bytes send (response) per URI

Here's some sample output (URI values are made up):

```
Requests  AvgResponse  AvgQueue  AvgDisp  AvgCPU  AvgBytesRcvd  AvgBytesSent  URI
 10627        352         11       330      84        675          187248     /some/http/request
  5484        354          8       341      84        663          210074     /some/other/http/request
  4019        268          0       266      63        711           62883     /yet/another/http/request
  3989         96         12        78      17        582           86293     /really/just/one/more/http/request
   …           …           …         …       …         …              …       …
 60578        311          7       296      70        679          152760     Overall
```

There were a lot of rows in this table...I left out a bunch and just showed the top and the last row.

Here's an invocation example:

```
export CLASSPATH=~/smfplugins/bbomsmfv.jar:~/smfplugins/WP102312_plugins.jar
java com.ibm.ws390.sm.smfview.SMF "INFILE(WAS.SMF.DATA)"
                                "PLUGIN(com.ibm.ws390.smf.plugins.ResponseTimes,
                                                ~/smfplugins/ResponseTimes.csv)"
```

Document change history

Check the date in the footer of the document for the version of the document.

June 27, 2013	Initial Version
July 24, 2013	Updated with techdoc number
July 30,2013	Updated to clarify what's in V8 and what isn't
September 28, 2013	Added RequestDensity2

October 11, 2013	`Updated RequestDensity2, RequestsPerServer, and ThreadRequestDensity/ThreadRequestDensity2`
	`Added ResponseTimes`

October 23, 2013	`Fixed typos in examples for RequestDensity and ThreadRequestDensity plugins (Thanks Amr)`
October 28, 2013	`Support SMF 120-10 records on RequestPerServer and ReWrite, plus EJB/WOLA information in CSVExport from classification section`
August 28, 2015	`Add z/OS Connect CSV Export support`
September 27, 2016	`Add LibertyExport and LibertyBatchExport plugins`
November 30, 2018	`Add JCL invocation method`
December 19, 2018	`Add SplitByServer`

End of WP102312