

Troubleshooting WebLogic 9.2 Issues

Mary Manchukian
Sr. Principal Support Engineer
PeopleTools Enterprise Server Tools

Introduction

WebLogic 9.2 is one of the leading J2EE™ application servers in today's marketplace and one of three supported with PeopleTools 8.49. Monitoring WebLogic for its performance and availability is an unavoidable task for system and web administrators. This document intends to provide some guidelines to prevent performance and availability problems and to help diagnose existing ones such as WebLogic crashes and hangs.

The most common WebLogic issues that customers encounter are:

- WebLogic running out of memory
Java cores and heap dumps on AIX; Jrockit dumps on Linux
- Web server stops responding
- Multiple sessions for same user due to non-stickiness
- Slow performance
- Internal Server Errors 500

All of these problems or symptoms can be diagnosed using the same techniques that will be discussed in this document.

The tools used to monitor the WebLogic Server health are:

- Admin Console:
 - Monitor WebLogic Memory Usage
 - Monitor WebLogic Thread Usage
 - Monitor WebLogic Sessions
- Log files:
 - HTTP Access Logs
 - Thread Dumps
 - IDDA logs
 - Stdout log for verbose garbage collection
- psadmin
 - Monitor the Tuxedo Application Server

Out of Memory Problems (OOM)

System Administrators can catch memory and thread leaks in early stages by simply monitoring the server through the admin console. This will be discussed later in this document. However, as in many production situations, customers do not identify these issues until the WebLogic Server crashes or the technical personnel receive a call from the end users.

An application displays Out of Memory (OOM) errors due to memory exhaustion, either in the java heap or native memory.

Out of memory problems can be easily verified by looking at the WebLogic log files.

Below are some samples of out of memory exceptions:

```
<Feb 4, 2009 3:04:25 PM EST> <Error> <HTTP> <BEA-101017>  
<[weblogic.servlet.internal.WebAppServletContext@29b6928 - appName: 'peoplesoft', name: '/',  
context-path: ''] Root cause of ServletException.
```

java.lang.OutOfMemoryError: allocLargeArray - Object size: 43176, Num elements: 43154

```
at bea.jolt.SInputBuffer.getBytes(SInputBuffer.java:674)  
at bea.jolt.JoltMessage.unserialize(JoltMessage.java:1123)  
at bea.jolt.JoltRemoteService.decodeCALL(JoltRemoteService.java:439)  
at bea.jolt.JoltRemoteService.call(JoltRemoteService.java:345)  
at bea.jolt.JoltRemoteService.call(JoltRemoteService.java:267)  
Truncated. See log file for complete stack trace
```

```
#####<Jun 17, 2008 2:43:47 PM CDT> <Error> <HTTP> <mymachine> <PIA> <[ACTIVE]  
ExecuteThread '13' for queue: 'weblogic.kernel.Default (self-tuning)')> <<WLS Kernel>> <> <>  
<1213731827172> <BEA-101017> <[weblogic.servlet.internal.WebAppServletContext@129a75a  
- appName: 'peoplesoft', name: '/', context-path: ''] Root cause of ServletException.
```

java.lang.OutOfMemoryError: Java heap space

```
#####<Sep 14, 2008 12:27:04 PM GMT> <Error> <HTTP> <mymachine> <PIA> <[ACTIVE]  
ExecuteThread '5' for queue: 'weblogic.kernel.Default (self-tuning)')> <<WLS Kernel>> <> <>  
<1189772824435> <BEA-101017> weblogic.servlet.internal.WebAppServletContext@1dc5ae5  
appName: 'peoplesoft' name: '/', context-path: ''] Root cause of ServletException
```

java.lang.OutOfMemoryError: unable to create new native thread

Although the call stack in the exception may vary depending on the condition that triggered the out of memory, the exceptions clearly indicate a memory problem.

Types of Memory

JVM Heap

The **Java Virtual Machine (JVM) heap** is the memory region where the Java objects (both live and dead) reside. When the Java heap runs out of space, the Java Garbage Collector will be invoked to de-allocate unreferenced objects and free up more space for the program to continue its operation. The JVM cannot service user requests during garbage collection. The JVM heap size is set to 256MB by default. This might or might not be large enough for your environment. Running load tests is recommended to determine the optimal heap settings and number of domains required for your environment. Setting the JVM heap size minimum value equal to the maximum value is recommended to avoid the performance hit incurred by dynamically growing the JVM. This also improves predictability and lessens the frequency of JVM garbage collection.

There are two types of garbage collection (GC) that take place:

1. A **full GC** runs over the entire heap to clean up memory. It typically takes no more than 3-5 seconds. The JVM cannot service user requests during full GC.
2. A **minor GC** moves/cleans objects within the 'new heap' and takes less than 2 milliseconds (can happen every 3-10 seconds).

To set the heap size for the PeopleSoft Internet Architecture domain (PIA), open the setEnv.cmd/sh file located under

```
<PS_HOME>\webserv\<DOMAIN_NAME>\bin
```

Locate the variable JAVA_OPTIONS_OS, where OS should be the Operating System platform where the WebLogic Server is running. For example, in a Windows environment, you would have:

```
SET JAVA_OPTIONS_WIN32=-server -Xms256m -Xmx256m
```

To increase the heap size, you need to change the options -Xms and -Xmx to the desirable value. For example:

```
SET JAVA_OPTIONS_WIN32=-server -Xms512m -Xmx512m
```

If WebLogic is running on Windows as a service, you will need to rerun the service install script after making the above changes to the Java Options (see *Doc ID 622372.1: E-WL: How to Install WebLogic Admin and Managed Servers as Windows Services?*)

You can also manually edit the value for the heap size in the registry using regedit. Once regedit is open navigate to:

[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\PeopleSoft-PIA\Parameters]

Replace PeopleSoft-PIA with the name of your Windows service.

Double click on **CmdLine** and change the default values for –Xms and –Xmx to your desired settings. After doing this, restart your service.

Refer to knowledge document 638298.1 (E-WL: How To Increase/Decrease JVM Heap Size for WebLogic 8.1 and 9.2) for details on changing the java heap settings.

Native Memory

Sometimes, out of memory exceptions occur when the native heap has been exhausted. Here's a sample exception:

```
#####<Sep 14, 2007 12:27:04 PM GMT> <Error> <HTTP> <mymachine> <PIA> <[ACTIVE]
ExecuteThread '5' for queue: 'weblogic.kernel.Default (self-tuning)'> <<WLS Kernel>> <> <>
<1189772824435> <BEA-101017> weblogic.servlet.internal.WebAppServletContext@1dc5ae5
appName: 'peoplesoft' name: '/', context-path: "]" Root cause of ServletException
java.lang.OutOfMemoryError: unable to create new native thread
```

Native memory is the memory that the JVM uses for its own internal operations. The amount of native memory that will be used by the JVM depends on the amount of code generated, threads created, memory used during GC for keeping java object information and temporary space used during code generation, optimization, etc.

The maximum amount of native memory is limited by the virtual process size limitation on any given OS and the amount of memory already committed for the java heap with the -Xmx flag. For example, if the application can allocate a total of 3 GB and the max java heap is 1 GB, then the max possible native memory is approximately 2 GB.

Process size – Process size will be the sum of the java heap, native memory and the memory occupied by the loaded executables and libraries. On 32-bit operating systems, the virtual address space of a process can go up to 4 GB. Out of this 4 GB, the OS kernel reserves some part for itself (typically 1 – 2 GB). The remaining is available for the application.

For example, in Windows – by default, 2 GB is available for the application and 2 GB is reserved for Kernel's use (with the exception of some variants of

Windows). In this case, if the maximum java heap were set to 1 GB, then the max amount of native memory would be 1 GB.

For other operating systems, please refer to the OS documentation for your configuration.

Out of memory in the native heap usually happens when the process reaches the process size limitation on that OS, or the machine runs out of RAM and swap space. In this case, the JVM will exit and may generate a core file when it gets a sigabrt signal. If the machine doesn't have enough RAM and swap space, then the OS will not be able to give more memory to this process that could also result in out of memory. Make sure that the sum of RAM and swap space in the disk is sufficient to cater to all the running processes in that machine.

In any of the scenarios described above, the end users logged into the application (or trying to log in) at the time when the Web Logic Server ran out of memory will get an Internal Server Error 500 on the screen:

Error 500--Internal Server Error

From RFC 2068 Hypertext Transfer Protocol -- HTTP/1.1:

10.5.1 500 Internal Server Error

The server encountered an unexpected condition, which prevented it from fulfilling the request.

Note: Error 500--Internal Server Errors, although a symptom of an out of memory issue, is not always caused by lack of memory.

In summary, to determine whether the OOM problem is a Java OOM or Native OOM:

- If the stdout/stderr message says that this is a `java.lang.OutOfMemoryError`, then this is **Java OOM**
- If the stdout/stderr message says that it failed to acquire memory, such as unable to create new native thread, then this is a **Native OOM**

Refer to knowledge document 638298.1 (E-WL: WebLogic Crashes/Hangs with Error: "java.lang.OutOfMemoryError: unable to create new native thread").

To learn more about types of memory in the heap, how garbage collection works and garbage collection tuning, please refer to the following documentation:

http://java.sun.com/docs/hotspot/gc5.0/gc_tuning_5.html

Other Memory Issues Contributing Factors

Out of memory problems caused by a small heap or bad implementation planning (not enough physical memory) are the easiest to fix. As described previously in this document, increasing the heap size or the physical memory can resolve memory problems. However, constant memory growth in either java heap or native memory, that eventually ends up in an out of memory situation might be an indication of a leak. If a leak is present, increasing the heap size or even physical memory will only delay the inevitable. The techniques to debug the memory leak scenarios are the same as the out of memory scenarios.

Along with the heap size, here are a few more things that need to be checked in any OOM event:

Check WebLogic Maintenance Pack level

For PeopleTools 8.49, MP0 is the minimum required although MP3 is strongly recommended. To verify the current Maintenance Pack level, check the PIA_weblogic.log. The server startup sequence will show something along these lines:

```
###<Mar 26, 2009 1:17:41 PM CDT> <Notice> <WebLogicServer> <> <> <main> <> <> <>
<1238091461639> <BEA-000365> <Server state changed to STARTING>
####<Mar 26, 2009 1:17:41 PM CDT> <Info> <WorkManager> <> <> <main> <> <> <>
<1238091461639> <BEA-002900> <Initializing self-tuning thread pool>
####<Mar 26, 2009 1:17:41 PM CDT> <Info> <WebLogicServer> <> <> <main> <> <> <>
<1238091461733> <BEA-000214> <WebLogic Server "PIA" version:
WebLogic Server Temporary Patch 3 for PeopleSoft CR384662 Dec 18 19:15:21 IST 2008
WebLogic Server 9.2 MP3 Mon Mar 10 08:28:41 EDT 2008 1096261 (c) 1995, 1996, 1997,
1998 WebLogic, Inc.
(c) 1999, 2000, 2001 BEA Systems, Inc.>
```

Check OS File Descriptor Settings

A file descriptor is required for every file that is opened, every *.class file read in by WebLogic, every Jolt connection PIA/Portal make to the Application Server, every connection that has to open back to a client, plus any other socket based communication that occurs on that machine.

To raise the file descriptors for UNIX, use the following command:

```
ulimit -n 4096
```

On Windows there is not an explicit parameter for the number of file descriptors. Hardware resources, mainly system memory, implicitly limit it.

Lower OS TCP/IP Cleanup/Timeout Settings

Socket based applications that are opening and closing hundreds or thousands of sockets need to have sockets they have marked as closed, truly closed. Once a process closes a socket, it is really only marked as closed until the OS, based on a cleanup/flush timeout, makes that socket available again.

The default TCP Wait Time for both Windows and most UNIX operating systems is 4 minutes, which is too long for PIA usage and tends to leave too many socket connections in TIME_WAIT state. By reducing the TCP Wait time, the socket can be recycled more efficiently.

For example, on Windows:

Use the registry editor, regedit, and create a REG_DWORD named TcpTimedWaitDelay under HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\TcpIp\Parameters

Set the value to 60 sec (this one is measured in seconds).

Refer to knowledge document 747389.1 (Online Performance Configuration Guidelines for PeopleTools 8.45, 8.46, 8.47, 8.48 and 8.49) or contact your OS vendor.

Using Java I/O Instead of Native I/O (Performance Packs)

WebLogic Server uses software modules called muxers to read incoming requests on the server and incoming responses on the client. There are two types of muxers:

1. Java Muxer (Java I/O): uses pure java to read data from sockets
2. Native Muxer (Native I/O): uses platform specific-binaries, also known as 'performance packs' to read data from sockets.

By default, WebLogic is configured to use Performance Packs (also called "Native I/O"). Benchmarks have shown major performance improvements when performance packs are used on machines that host WebLogic Server instances. When a system is under high load, the Web Server may experience poor performance and/or run out of memory if the Performance Pack is not being used. Also memory leaks have been reported in the past when using Java I/O instead of Native I/O.

Review the WebLogic log to check if the Performance Pack is being loaded:

Open the file

<PS_HOME>/webserv/<DOMAIN_NAME>/servers/PIA/logs/PIA_weblogic.log

The WebLogic startup sequence should show the following message if the Performance Pack is being used:

WLS Kernel>> <> <> <1224794141609> <BEA-000446> <Native IO Enabled.>

If the Performance Pack is not being used, you may see a message like this:

####<Jun 1, 2008 11:03:23 AM PDT> <Error> <Socket> <abc.cde..com> <PIA> <[ACTIVE]
ExecuteThread: '0' for queue: 'weblogic.kernel.Default (self-tuning)'> <<WLS Kernel>> <> <>
<1212343403519> <BEA-000438> <Unable to load performance pack. Using Java I/O instead.
Please ensure that a native performance library is in...

Refer to the following knowledge documents for details on how to resolve issues with loading the Performance Pack:

- 656085.1: *E-WL: PT 8.49: Unable to Load Performance Pack when PIA Server is installed as a Windows Service*
- 661277.1: *E-WL: Unable to Load Performance Pack on WebLogic 9.2 HP-UX*
- 785594.1: *E-WL How to Validate that Native I/O is Being Used in WebLogic 9.2*
- 658629.1: *E-WL: Error BEA-000438 "Unable to load performance pack. Using Java I/O instead"*

Monitoring WebLogic Server Health

In situations where the PeopleSoft Application performance appears to be degrading or simply as a good practice, you should monitor the WebLogic Server(s) resources as well as the Tuxedo Application Server(s).

There are a number of resources that should be monitored regularly:

Thread Usage

When a request is submitted to the web server, a thread is created. The thread is not released until the request has been completed.

In prior WebLogic releases (e.g. WebLogic 8.1), it was necessary to define the maximum thread count in the WebLogic configuration and all threads were allocated when the WebLogic server was started.

However, starting with WebLogic 9.2, there is a 'self tuned thread pool' and it automatically increases/decreases available threads depending on demand. In other words, in WebLogic 9.2, there is no need to adjust the thread count.

You can monitor thread usage via WebLogic Console as follows:

1. Log into the WebLogic Admin Console <http://webserver:9999/console>, where 9999 is the default Admin Port.
2. Navigate to Environment->Servers on the left menu
3. Select 'PIA'
4. Go to the Monitoring tab and then 'Threads' sub tab
5. The bottom of the page shows all active threads.

Status

Running Servers

Failed (0)

Critical (0)

Overloaded (0)

Warn (0)

OK (1)

Self-Tuning Thread Pool Threads(Filterd - More Columns Exist)

Showing 1 - 5 of 5 Previous | Next

Name ^	Total Requests	Current Request	Transaction	User	Idle	Stuck	Hogger	Standby
[ACTIVE] ExecuteThread: '0' for queue: 'weblogic.kernel.Default (self-tuning)'	729	Http Request: /psc/ps/EMPLOYEE/PT_LOCAL/c/OPTIMIZATION.KDCOMP.GBL		<anonymous>	false	false	true	false
[ACTIVE] ExecuteThread: '1' for queue: 'weblogic.kernel.Default (self-tuning)'	452	Http Request: /psc/ps/EMPLOYEE/PT_LOCAL/c/OPTIMIZATION.KDCOMP.GBL		<anonymous>	false	false	true	false
[ACTIVE] ExecuteThread: '2' for queue: 'weblogic.kernel.Default (self-tuning)'	68			<WLS Kernel>	true	false	false	false
[ACTIVE] ExecuteThread: '3' for queue: 'weblogic.kernel.Default (self-tuning)'	37	Http Request: /console/console.portal		system	false	false	false	false

Refer to knowledge document 660080.1 (E-WL: Managing Threads in WebLogic Server 9.2) for more details.

Memory utilization

There are two main methods to monitor JVM memory (heap):

Method 1: WebLogic Admin Console

1. Log into the WebLogic Admin Console <http://webserver:9999/console>, where 9999 is the default Admin Port.
2. Navigate to Environment->Servers on the left menu
3. Select 'PIA'
4. Go to the Monitoring tab and then 'Performance' sub tab
5. This will show you the current size of the JVM heap (in bytes), the current amount of memory that is available in the JVM heap (in bytes) and the percent of JVM free heap.

The screenshot shows the WebLogic Server Administration Console. On the left is the 'Domain Structure' tree with 'sing80' selected. The main area is titled 'Settings for PIA' and has tabs for Configuration, Protocols, Logging, Debug, Monitoring, Control, Deployments, Services, Security, and Notes. The 'Performance' tab is active, showing sub-tabs for General, Health, Channels, Performance, Threads, Timers, Workload, Security, Default Store, JMS, SAF, JDBC, and JTA. Below these are buttons for 'Garbage Collect' and 'Dump Thread Stacks'. A text box states: 'This page allows you to monitor performance information about this server. You can also use this page to force garbage collection or a thread dump.' Below this is a section titled 'Java Virtual Machine Memory Utilization Statistics' containing a table:

Java Virtual Machine Memory Utilization Statistics		
Heap Size Current:	260505600	The current size (in bytes) of the JVM heap. More Info...
Heap Free Current:	124376376	The current amount of memory (in bytes) that is available in the JVM heap. More Info...
Heap Free Percent:	47	Percentage of the maximum memory that is free. More Info...
Heap Size Max:	260505600	The maximum free memory configured for this JVM. More Info...

Refer to Knowledge Document 662595.1 (E-WL: WebLogic 9.2: How to monitor WebLogic performance via graphs) for information on graphing memory

Method 2: Enable Verbose Garbage Collection

To enable verbose garbage collection, add '**-verbosegc**' flag in the java command line. Edit the setEnv.cmd/sh file and find the JAVA_OPTIONS_OS environment variable.

Refer to Knowledge Document 759137.1 (E-WL: What is Verbose Garbage Collection and How do I Enable it on WebLogic 9.2?) for more details.

Below are a sample GC output and its interpretation

[Full GC 1572864K->150404K(1572864K), 290.973 ms][Mon Nov 24 11:42:56 2008]

Format varies depending on the JVM:

GC <before>K-><after>K (<heap>K), <total> ms
 <before> - memory used by objects before collection (KB)
 <after> - memory used by objects after collection (KB)
 <heap> - size of heap after collection (KB)
 <total> - total time of collection (milliseconds)

In the above example, the memory used before GC started was 1.5G and 150m after GC. The heap size is set to 1.5G. It took .291 seconds for the GC to complete.

If the line starts with [GC it is a 'minor GC', if it starts with '[Full GC' it is a 'full GC'

To learn more about how garbage collection works and garbage collection tuning, please refer to the following documentation:

http://java.sun.com/docs/hotspot/gc5.0/gc_tuning_5.html

Monitoring HTTP Sessions

HTTP is a request/response protocol between clients (e.g. browsers, end users) and servers. An HTTP Session, commonly referred just as "session", is the active connection between a client and the server, typically beginning with the user's first interaction and ending when the user signs out or the session is expired by the web server.

It is important that the sessions are destroyed when the user logs out or when they expire (e.g. if the user closed the browser). Dangling sessions take up memory, and over time, they can consume all the heap space and cause the WebLogic Server to crash. Unreferenced objects are destroyed when the garbage collection runs, freeing up space in the heap. Since "dangling" sessions are technically still alive, objects associated with them cannot be considered "unreferenced" and therefore cannot be destroyed.

You can view HTTP sessions via the WebLogic Admin Console as follows:

1. Log into the WebLogic Console and select 'Deployments' on the left pane
2. Go to the 'Control' tab
3. Click on the '+' sign next to deployment 'PeopleSoft' in order to expand it
4. Click on the '/' in the first entry under 'Modules' (no name is specified in this entry)
5. Go to the 'Monitoring' tab, then the 'Sessions' tab
6. The first time you go here, click 'customize this table' and add 'Monitoring id' to the 'chosen columns'

The screenshot shows the WebLogic Administration Console interface. The top navigation bar includes the 'bea' logo, 'WEBLOGIC SERVER ADMINISTRATION CONSOLE', and a 'Welcome, system' message. The left sidebar contains a 'Change Center' section with 'View changes and restarts' and 'Lock & Edit' buttons, and a 'Domain Structure' tree showing the hierarchy from 'sing80' down to 'Diagnostics'. The main content area is titled 'Settings for /' and has tabs for 'Overview', 'Configuration', 'Security', 'Testing', and 'Monitoring'. Under the 'Monitoring' tab, there are sub-tabs for 'Web Applications', 'Servlets', 'Sessions', 'PageFlows', and 'Workload'. The 'Sessions' sub-tab is selected, displaying a table of 'Servlet Sessions (Filtered - More Columns Exist)'. The table has columns for 'Monitoring ID', 'Context Root', 'Server', 'Creation Time', 'Time Last Accessed', and 'Max Inactive Interval'. Two sessions are listed: one for 'KDRIV@10.138.250.172/ps' and another for 'PTDMO@10.138.250.92/ps', both on the 'PIA' server. The 'Time Last Accessed' column shows timestamps from November 20, 2008.

Monitoring ID	Context Root	Server	Creation Time	Time Last Accessed	Max Inactive Interval
KDRIV@10.138.250.172/ps		PIA	Thu Nov 20 15:51:44 PST 2008	Thu Nov 20 15:51:45 PST 2008	1440
PTDMO@10.138.250.92/ps		PIA	Thu Nov 20 15:51:32 PST 2008	Thu Nov 20 15:51:34 PST 2008	1440

The above screen shot shows the default monitoring view, which can be customized. PeopleSoft user ID, clients IP address and PIA site to which the user is connected is shown in the “Monitoring ID” column. The “Time Last Accessed” column shows the time when the last user-server interaction occurred. The “Server” column shows the server to which the user is connected.

Refer to Knowledge Document 638313.1 (E-WL: How to Enable HTTP Session Monitoring in WebLogic?).

HTTP Requests – Access log

Enabling the HTTP Access log allows you to keep a log of all the HTTP requests on a given server instance:

1. Log into the WebLogic Admin Console <http://webserver:9999/console>, where 9999 is the default Admin Port.
2. Navigate to Environment->Servers on the left menu bar
3. If you have not already done so, in the Change Center of the Administration Console, click Lock & Edit
4. In the Servers table, select the name of the server instance whose HTTP logging you want to configure (e.g. PIA)
5. Select the 'Logging' tab and then 'HTTP' sub tab
6. Select the HTTP Access Log File Enabled check box, if it is not already selected.
7. Go to the 'Advanced' section at the bottom of the page

8. Choose format 'Extended' and for 'Extended Logging Format Fields', enter the following string as is:
date time c-ip cs-method time-taken sc-status bytes cs-uri cs(Cookie)
9. Save and click on 'Activate Changes'

Below is a sample excerpt of a common access log:

```
10.138.234.239 - - [08/Jan/2009:11:56:42 -0700] "GET /C881G71P/signon.html HTTP/1.1" 304 0
10.138.234.239 - - [08/Jan/2009:11:56:44 -0700] "GET /psp/C881G71P/?cmd=login HTTP/1.1"
200 8628
10.138.234.239 - - [08/Jan/2009:11:56:52 -0700] "GET /C881G71P/signin.css HTTP/1.1" 200
2544
10.138.234.239 - - [08/Jan/2009:11:56:52 -0700] "GET /C881G71P/images/ps_logo.gif HTTP/1.1"
304 0
```

One key piece of information shown in the access logs is the HTTP status code for each HTTP request:

```
10.138.234.239 - - [08/Jan/2009:11:56:44 -0700] "GET /psp/C881G71P/?cmd=login HTTP/1.1"
200 8628
```

The following is a partial list of HTTP response status codes and standard associated phrases, intended to give a short textual description of the status. These status codes are specified by RFC 2616, along with additional, non-standard status codes sometimes used on the Web.

200 – OK

The request has succeeded.

304 – Not Modified

GET request - Document has not been modified

404 – Not Found

The server has not found anything matching the Request-URI.

500 – Internal Server Error

The server encountered an unexpected condition, which prevented it from fulfilling the request.

Check the PIA_weblogic.log for an exception.

For a complete list of HTTP status codes you can refer to the following Website:

<http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html>

Refer to Knowledge Document 662319.1 (E-WL: Enabling and Configuring an HTTP Log for WebLogic 9.2) for additional details on enabling HTTP access logs.

IDDA log

Enabling the IDDA log is useful when troubleshooting WebLogic crashes due to session stickiness issues. Refer to the “Session Stickiness” section for more details.

How to enable the IDDA log:

1. Log into PIA and navigate to "PeopleTools -> Web Profile -> Web Profile Configuration". Search for the Web Profile in use
2. Go to the 'Custom Properties' tab and add a new field called "IDDA". Set the property to type of Number and a Value of 2.
3. Restart the Web Servers.

Profile Name: TEST

Custom Properties

Find View All First 1-4 of 4 Last

Row 1	*Property Name	IDDA	Validation Type	Number	Property Value	2
Row 2	*Property Name	auditPWD	Validation Type	String	Property Value	dayoff

+ -

The IDDA log is prefixed psftIDDA and is written to the PIA installation's domain directory <PS_HOME>/weberv/<DOMAIN_NAME>/psftIDDA*.log. You should see two IDDA logs, one for the HTTP and one for the HTTPS port.

Below is an excerpt of an IDDA log:

```
[Tue Dec 23 12:28:16 PST 2008]<debug src=psp.service>10.138.250.92
http:myserver:8000/psp/ps/EMPLOYEE/PT_LOCAL/c/WEB_PROFILE.WEB_PROFILE.GBL?PO
RTALPARAM_PTCNAV=PT_WEB_PROFILE&EOPP.SCNode=PT_LOCAL&EOPP.SCPortal=E
MPLOYEE&EOPP.SCName=PT_WEB_PROFILE&EOPP.SCLabel=Web%20Profile&EOPP.SCP
Tfname=PT_WEB_PROFILE&FolderPath=PORTAL_ROOT_OBJECT.PT_PEOPLETOOLS.PT_
WEB_PROFILE.PT_WEB_PROFILE&IsFolder=false
USERID=PTDMO
sessionId=STP6JRJXpQHhJnnSh8tpctIXQYSMyC8j!-466540792!1230064087328
sessionNew=false
requestMethod=GET
```


Number of tools' cookies 7
SignOnDefault: PTDMO
ExpirePage: http://myserver:8000/ps/ps/
PS_LOGINLIST: http://myserver:8000/ps
http%3a%2f%2fkdriver-us%3a8000%2f%2f%2femployee%2fpt_local%2frefresh: list:
PS_TOKENEXPIRE: 23_Dec_2008_20:28:15_GMT Tue Dec 23 12:28:15 PST 2008
PS_TOKEN:
pQAAAAQDAgEBAAAAvAIAAAAAAAsAAAAABABTaGRyAk4AbQg4AC4AMQAwABTOdWR/+jlc
fBGF9PSeqe9qlYBLDWUAAAAFAFNkYXRhWXicHYpLDkBAFARrEAsrFzEZwwhLQWz8FvZO4I
YOp3mdVCfV7waSODJG/UT8lx2cjKzspBMbM/lnLhaZgV5de5zSUIDiRU8lOmqsTCMGrfb/ctoCH
bx1MQtj

Generating a Thread Dump

If you have to contact Global Customer Support, you will most likely be asked to provide a few thread dumps. A thread dump is a debug file that shows the status of each thread at the time it was generated. It is written to the PIA_stdout.log on all Unix platforms except Linux, where it is redirected to the PIA_stderr.log. On Windows, if WebLogic is running as a Windows Service, the thread dump is written to the Windows Service log NTservice-
<DOMAIN_NAME>-PIA.log.

Creating a Thread Dump on Windows:

Option 1:

Open a command prompt and enter the following command:

```
BEA_HOME\weblogic92\server\bin\beasvc -dump -svcname:peoplesoft-PIA
```

Where *peoplesoft-PIA* is the name of the PIA Windows service.
This will create the thread dump in NTservice-peoplesoft-PIA.log file.

Option 2:

```
PS_HOME\websevr\<DOMAIN_NAME>\bin\createthreaddump.cmd -svcname  
peoplesoft-PIA
```

Where *peoplesof-PIA* is the name of the PIA Windows service.
This will create the thread dump in the NTservice-peoplesoft-PIA.log file.

Creating a Thread Dump on Unix:

Run "kill -3 <PID of the java process>"

This will create the thread dump in the PIA_stdout. On AIX systems, look for a java core file with .txt extension. On Linux systems, the thread dump will be written to the PIA_stderr.log.

Refer to Knowledge Document 659452.1 (E-WL: How to Create a Thread Dump with WebLogic Server 9.2) for more options on generating thread dumps.

Reading a Thread Dump

Reading or interpreting a thread dump is not a simple task and most likely, you will need assistance from PeopleSoft and/or BEA Global Customer Support. However, here are a few things to look out for:

Edit the PIA_stdout.log, PIA_stderr.log or the NTservice-*peoplesoft-PIA*.log depending on your OS platform. Scroll down until you reach the beginning of the dump, which will contain the words "Full thread dump". Here's a sample excerpt:

```
<Jan 10, 2009 12:47:14 PM EST> <Warning> <WebLogicServer> <BEA-000337>
<ExecuteThread: '262' for queue: 'weblogic.kernel.Default' has been busy for "1,739" seconds
working on the request "Http Request: /psp/PAPROD/EMPLOYEE/PAPROD/h/", which is
more than the configured time (StuckThreadMaxTime) of "600" seconds.>
[Mon Jan 10 12:48:21 2009] [I] [ControlHandler] 1
Full thread dump Java HotSpot(TM) Client VM (1.5.0_12-b04 mixed mode):
```

```
"NwWriter" daemon prio=5 tid=0x385ddb8 nid=0x1cfc in Object.wait() [4981f000..4981fdb0]
at java.lang.Object.wait(Native Method)
at java.lang.Object.wait(Object.java:429)
at bea.jolt.OutQ.getFromQ(OutQ.java:89)
- locked <0x1b0a6818> (a bea.jolt.OutQ)
- at bea.jolt.NwWriter.run(NwHdlr.java:3932)
```

```
"NwReader" daemon prio=5 tid=0x3ebbf810 nid=0x2184 runnable [497df000..497dfdb0]
at java.net.SocketInputStream.socketRead0(Native Method)
at java.net.SocketInputStream.read(SocketInputStream.java:129) at
java.io.DataInputStream.readFully(DataInputStream.java:266) at
bea.jolt.NwReader.run(NwHdlr.java:3577)
```

The beginning of the thread dump will show the JDK version your Web Server is running on:

```
[Mon Jan 10 12:48:21 2009] [I] [ControlHandler] 1
Full thread dump Java HotSpot(TM) Client VM (1.5.0_12-b04 mixed mode):
```

On most Unix and Windows systems, the first section of the thread dump will show the Jolt NwReader and NwWriter threads followed but all of the Execute Threads. Note that there may be slight variations depending on your JDK vendor.

WebLogic Server and the Tuxedo Application Server use Jolt to communicate with each other. PIA creates two threads inside the WebLogic's JVM per Jolt connection. For each Jolt connection made between WebLogic and the Tuxedo Application Servers, you will see a `bea.jolt.NwReader` and a `bea.jolt.NwWriter` thread in the thread dump:

```
"NwWriter" daemon prio=5 tid=0x00dc69e0 nid=0x7c in Object.wait()
[0xe2831000..0xe28319c0]
at java.lang.Object.wait(Native Method)
- waiting on <0xee3fd440> (a bea.jolt.OutQ)
at java.lang.Object.wait(Object.java:429)
at bea.jolt.OutQ.getFromQ(OutQ.java:89)
- locked <0xee3fd440> (a bea.jolt.OutQ)
at bea.jolt.NwWriter.run(NwHdlr.java:4033)
```

```
"NwReader" daemon prio=5 tid=0x00dc6838 nid=0x7b runnable
[0xe2861000..0xe28619c0]
at java.net.SocketInputStream.socketRead0(Native Method)
at java.net.SocketInputStream.read(SocketInputStream.java:129)
at java.io.DataInputStream.readFully(DataInputStream.java:266)
at bea.jolt.NwReader.run(NwHdlr.java:3678)
```

Scroll down to the Execute Threads, where you will see the state of each thread by number in descending order. Each thread is shown with a call stack that is read, like any other Java or C++ call stack, from bottom up. One thing you want to find out in these call stacks is, *who's code was being executed* when the thread dump was generated: WebLogic's, Java, PeopleSoft's or some other application's. You want to identify the last code that was being executed, right when the problem happened.

If you see all the execute threads or many of them "stuck" at the same place, then the root cause of the problem might be that piece of code. If you see different execute threads "stuck" at different places (different vendor's code, or same vendor, but different methods or functions), then it might be useful to generate multiple consecutive thread dumps, e.g. three or four, one every 15 or 20 seconds. If the multiple thread dumps show the same thread (e.g. Execute Thread 50) executing different pieces of code on each one, then that is considered normal. In this case further monitoring will be necessary.

Below are some execute thread sample call stacks that might help you when reading a thread dump:

Ex#1: Idle thread:

```
"ExecuteThread: '48' for queue: 'weblogic.kernel.Default'" daemon prio=1 tid=0x086a4ce8
nid=0x1d60 in Object.wait() [886d5000..886d5cc8]
at java.lang.Object.wait(Native Method)
at java.lang.Object.wait(Object.java:429)
at weblogic.kernel.ExecuteThread.waitForRequest(ExecuteThread.java:154)
```

at weblogic.kernel.ExecuteThread.run(ExecuteThread.java:174)

Ex#2: Call stack of a PIA request after a Jolt request was submitted to the Application Server:

```
"ExecuteThread: '25' for queue: 'weblogic.kernel.Default'" daemon prio=5
tid=0x00a0a118 nid=0x27 in Object.wait() [0xe5890000..0xe58919c0]
at java.lang.Object.wait(Native Method)
- waiting on <0xea641800> (a bea.jolt.IOBuf)
at bea.jolt.IOBuf.waitOnBuf(IOBuf.java:119)
- locked <0xea641800> (a bea.jolt.IOBuf)
at bea.jolt.NwHdlr.recv(NwHdlr.java:1468)
at bea.jolt.CMGr.recv(CMGr.java:163)
at bea.jolt.JoltSession.recv(JoltSession.java:550)
at bea.jolt.JoltRemoteService.call(JoltRemoteService.java:313)
at bea.jolt.JoltRemoteService.call(JoltRemoteService.java:257)
at psft.pt8.net.NetReqRepSvc.sendRequest(NetReqRepSvc.java:569)
at psft.pt8.net.NetService.requestService(NetService.java:141)
at psft.pt8.net.NetReqRepSvc.requestService(NetReqRepSvc.java:330)
- locked <0xee3f9b20> (a psft.pt8.net.NetSession)
at psft.pt8.jb.JBEntry.processRequest(JBEntry.java:340)
- locked <0xee4dd3c8> (a psft.pt8.util.JBStatusBlock)
at psft.pt8.psc.onActionGen(psc.java:1673)
at psft.pt8.psc.onAction(psc.java:1155)
at psft.pt8.psc.service(psc.java:584)
- locked <0xf6f868c0> (a java.lang.String)
at javax.servlet.http.HttpServlet.service(HttpServlet.java:853)
```

Ex#3: PIA request waiting on a long running Jolt request:

Part 1:

```
"ExecuteThread: '25' for queue: 'weblogic.kernel.Default'" daemon prio=5
tid=0x00a0a118 nid=0x27 in Object.wait() [0xe5890000..0xe58919c0]
at java.lang.Object.wait(Native Method)
- waiting on <0xea641800> (a bea.jolt.IOBuf)
at bea.jolt.IOBuf.waitOnBuf(IOBuf.java:119)
- locked <0xea641800> (a bea.jolt.IOBuf)
at bea.jolt.NwHdlr.recv(NwHdlr.java:1468)
at bea.jolt.CMGr.recv(CMGr.java:163)
at bea.jolt.JoltSession.recv(JoltSession.java:550)
at bea.jolt.JoltRemoteService.call(JoltRemoteService.java:313)
at bea.jolt.JoltRemoteService.call(JoltRemoteService.java:257)
at psft.pt8.net.NetReqRepSvc.sendRequest(NetReqRepSvc.java:569)
at psft.pt8.net.NetService.requestService(NetService.java:141)
at psft.pt8.net.NetReqRepSvc.requestService(NetReqRepSvc.java:330)
- locked <0xee3f9b20> (a psft.pt8.net.NetSession)
at psft.pt8.jb.JBEntry.processRequest(JBEntry.java:340)
- locked <0xee4dd3c8> (a psft.pt8.util.JBStatusBlock)
at psft.pt8.psc.onActionGen(psc.java:1673)
at psft.pt8.psc.onAction(psc.java:1155)
at psft.pt8.psc.service(psc.java:584)
- locked <0xf6f868c0> (a java.lang.String)
at javax.servlet.http.HttpServlet.service(HttpServlet.java:853)
```

Part 2:

```
"ExecuteThread: '21' for queue: 'weblogic.kernel.Default'" daemon prio=5
tid=0x006ae918 nid=0x23 waiting for monitor entry [0xe5950000..0xe59519c0]
at psft.pt8.psc.onActionDirect(psc.java:1191)
- waiting to lock <0xf6f868c0> (a java.lang.String)
at
psft.pt8.portal.PIADirectConnection.connect(PIADirectConnection.java:111)
at psft.pt8.portal.ContentGetter.getPIADirectContent(ContentGetter.java:362)
at psft.pt8.portal.ContentGetter.getPIADirectContent(ContentGetter.java:281)
at psft.pt8.portal.ContentGetter.getPIADirectContent(ContentGetter.java:277)
at psft.pt8.portal.ContentGetter.getContent(ContentGetter.java:519)
at psft.pt8.portal.ContentGetter.getContent(ContentGetter.java:470)
at psft.pt8.portal.ContentGetter.getContent(ContentGetter.java:467)
at psft.pt8.psp.getTemplateDoc(psp.java:2645)
at psft.pt8.psp.onAction(psp.java:2114)
at psft.pt8.psp.service(psp.java:674)
at javax.servlet.http.HttpServlet.service(HttpServlet.java:853)
```

Part 3:

```
"ExecuteThread: '17' for queue: 'weblogic.kernel.Default'" daemon prio=5
tid=0x005772e8 nid=0x1f waiting for monitor entry [0xe5a10000..0xe5a119c0]
at psft.pt8.psp.onStartTab(psp.java:1769)
- waiting to lock <0xf6f868c0> (a java.lang.String)
at psft.pt8.psp.service(psp.java:609)
at javax.servlet.http.HttpServlet.service(HttpServlet.java:853)
```

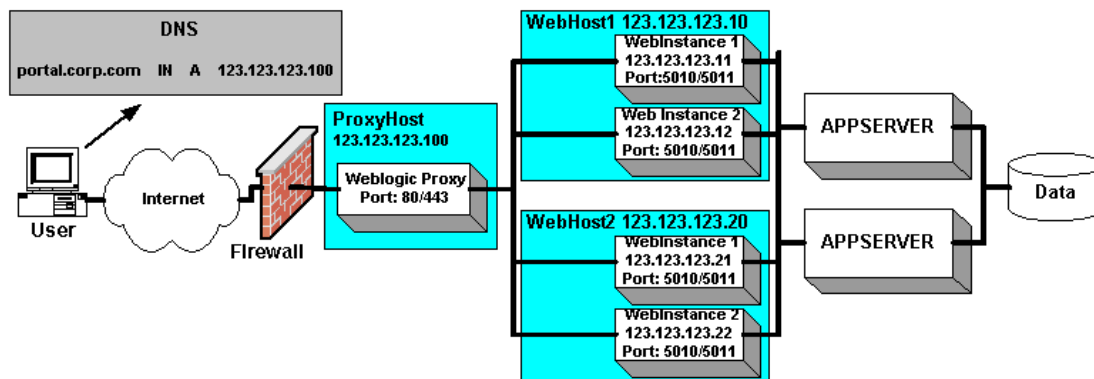
Note that the thread shown on “Part 1” has submitted a request to the Application Server via Jolt and has acquired the HTTP Session, which is referenced as to lock **0xf6f868c0**. The threads in “Part 2 and 3” are trying to acquire the same HTTP Session, in other words, they are trying to acquire the lock **0xf6f868c0**. If a user who has submitted a long running request, attempts to perform another action or submit another request before the response of the first request is back or if the request is resubmitted by a Proxy, then many threads may be “locked” by that user, awaiting the release of the HTTP Session by the long running request.

A scenario like the one above could cause the Web Server to hang because all the threads could go into a “waiting to lock” state. In this case, it is recommended to switch focus to the Tuxedo Application Server and monitor overall status: client, queue and server. We will discuss this in a later section.

If the thread dump shows many idle threads, then it is highly possible that the source of the problem is not WebLogic. Unfortunately, in cases like this where the thread dump does not provide any clues, further debugging will need to be performed to find the root cause of the issue.

Session Stickiness

Session Stickiness also referred to as session persistence, states that once a session is created for a user, all subsequent requests, must go to the same web server. You need to be concerned about this only if you have two or more web servers. For example, in the diagram below, there are two physical web servers with two instances each, all behind a proxy server. This configuration is also referred to as simple WebLogic Cluster and it must be configured so that all traffic, from a single session, goes to the same web server instance.



Refer to Knowledge Document 747378.1 (Clustering and High Availability for Enterprise Tools 8.4x) for more information on cluster configurations.

An HTTP Session is established when a user logs into the PeopleSoft Application. PIA requires that the session stick to one particular web server because some 'state' information such as PeopleCode variables and Page Processing data is stored in the HTTP Session. In other words, for PIA, the HTTP Session must be 'sticky' once user credentials have been verified. If the subsequent requests from the same user do not get routed to the same web server, PIA may not be able to maintain continuity of dialog.

Some symptoms of sessions not being sticky or persistent are:

1. Users being kicked back to the search or sign on page when interacting with a PIA page
2. Users getting an unusual message on the browser such as "Page no longer available" or "Page cannot be displayed" after sign on.
3. Users being kicked out with "An error has occurred"
4. Web servers running out of memory due to excessive number of HTTP sessions.
5. Multiple Jolt connections for the same user on the Application Server.

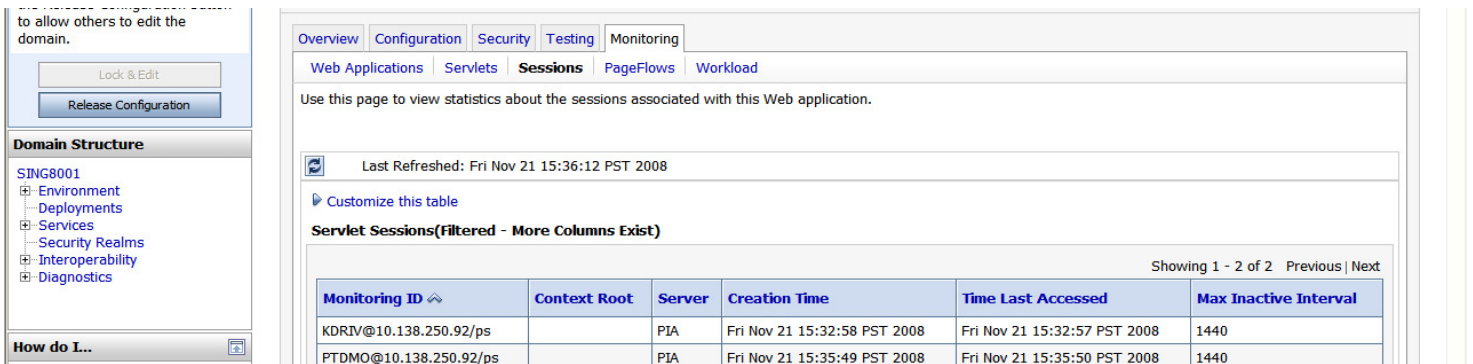
How to Confirm if Sessions are not Sticky

There are a few methods that can be used to check if sessions are not sticky:

Method 1: Use WebLogic Console to Check for Duplicate Sessions

An HTTP session for the same PeopleSoft user should not exist on two different Web Servers at the same time:

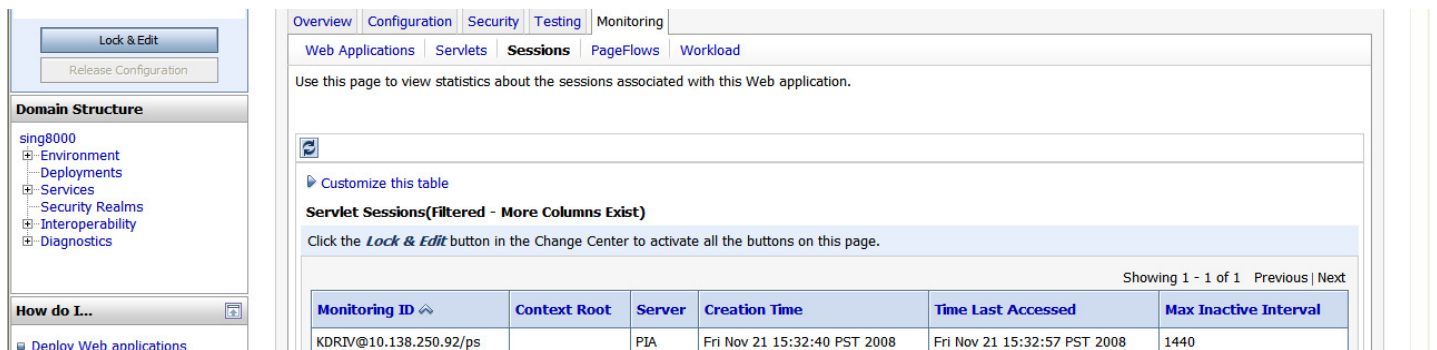
Web Server 1:



The screenshot shows the WebLogic Console interface for Web Server 1. The left sidebar contains a 'Domain Structure' tree with nodes for Environment, Deployments, Services, Security Realms, Interoperability, and Diagnostics. The main content area is titled 'Servlet Sessions' and displays a table of active sessions. The table has columns for Monitoring ID, Context Root, Server, Creation Time, Time Last Accessed, and Max Inactive Interval. Two sessions are listed, both for the user KDRIV@10.138.250.92/ps on the PIA server.

Monitoring ID	Context Root	Server	Creation Time	Time Last Accessed	Max Inactive Interval
KDRIV@10.138.250.92/ps		PIA	Fri Nov 21 15:32:58 PST 2008	Fri Nov 21 15:32:57 PST 2008	1440
PTDMO@10.138.250.92/ps		PIA	Fri Nov 21 15:35:49 PST 2008	Fri Nov 21 15:35:50 PST 2008	1440

Web Server 2:



The screenshot shows the WebLogic Console interface for Web Server 2. The left sidebar contains a 'Domain Structure' tree with nodes for Environment, Deployments, Services, Security Realms, Interoperability, and Diagnostics. The main content area is titled 'Servlet Sessions' and displays a table of active sessions. The table has columns for Monitoring ID, Context Root, Server, Creation Time, Time Last Accessed, and Max Inactive Interval. One session is listed for the user KDRIV@10.138.250.92/ps on the PIA server.

Monitoring ID	Context Root	Server	Creation Time	Time Last Accessed	Max Inactive Interval
KDRIV@10.138.250.92/ps		PIA	Fri Nov 21 15:32:40 PST 2008	Fri Nov 21 15:32:57 PST 2008	1440

In the above example, user KDRIV is logged in on both Web Servers around the same time. This confirms that sessions are not sticky.

Method 2: Check Application Server Logs for Multiple Jolt Connections per User

Check the Application Server log (APPSRV_MMDD.LOG) for any indication of multiple authentication requests for the same user. There should be only one Get Certificate Request per login from the same client IP (unless the client IP/name is from a proxy or load balancer, in which case, the actual end-user's client IP/name cannot be determined). A scenario where a user is first authenticated for the PeopleSoft user/password and later followed by one or more "PeopleSoft Token authentication succeeded" messages, would indicate a session stickiness problem. Subsequent "PeopleSoft Token authentications" should not occur because the first Get Certificate Request has already authenticated the user/password. "PeopleSoft Token authentications" are expected only in scenarios where content is pulled in by Portal.

For example, first GetCertificate request for PSJOBS from machine:
machine50.com:

PSAPPSRV.807128 (38) [08/21/08 10:31:04 GetCertificate](3) PeopleSoft ID and Password authentication succeeded for user PSJOB@machine50.com.

Notice that this GetCertificate request authenticates the PeopleSoft ID and Password PSJOB.

Subsequent GetCertificate for PSJOBS from same client machine:

PSAPPSRV.807128 (54) [08/21/08 10:31:54 GetCertificate](3) **PeopleSoft Token authentication** succeeded: PSJOB@machine50.com.

This indicates that that ID PSJOB has been authenticated more than once, which is a symptom of a stickiness issue.

Method 3: Enable IDDA Log and Check for POST Request for New Session

The IDDA log is another powerful tool to detect issues with session stickiness. One thing to look in an IDDA log is for POST requests from a new session as in the example below:

Excerpt from psftIDDA*.log

```
[Fri Jan 02 16:09:06 PST 2009]<debug src=PSAuthenticateor.authenticate>141.144.89.44
http.myuserid.mycompany.com:80/psc/ps/EMPLOYEE/PT_LOCAL/c/WEB_PROFILE.WEB_PRO
FILE.GBL?null
USERID=PTDMO
sessionId=S1v3JpsBbpdgrxvFwpJzZ1w17qQVybR!-439282144!1230941345796
sessionNew=true
requestMethod=POST
Number of tools' cookies 12
SignOnDefault: PTDMO
```


PeopleSoft Session Stickiness Check List:

Load balancers not configured properly cause most session stickiness issues, however, there are a few things that need to be checked on the PIA side as well. Below is a checklist:

1. Session cookie domain

The cookie domain must match the host's fully qualified domain name (FQDN) or domain suffix in the sign on URL. If the session cookie domain has not been set or is incorrect, the browser will not send the session cookie.

The cookie domain is automatically set when you specify the authentication domain during the PIA installation. If the authentication domain was not specified, it is recommended to reinstall PIA making sure you specify the authentication domain.

The cookie domain can be verified by looking at the weblogic.xml located under <PS_HOME>/webserve/<DOMAINNAME>/applications/peoplesoft/PORTAL/WEB-INF

```
<session-param>
<param-name>CookieDomain</param-name>
<param-value>.peoplesoft.com</param-value>
</session-param>
```

2. Load Balancer's Timeout

If a Load Balancer (or any request sprayer) uses a timeout to maintain stickiness, that timeout must be set higher than the session timeout configured in the Web Profile.

3. In a clustered environment, all web servers must have the same cookie name in their respective weblogic.xml file if the load balancer is configured for session stickiness based on Web Server cookie. Below is a sample default cookie name:

```
<session-param>
<param-name>CookieName</param-name>
<param-value>HOSTNAME-80-PORTAL-PSJSESSIONID</param-value>
</session-param>
```

If the session cookie name is changed manually, it must only contain legal characters. As per RFC 2965, this is "a sequence of non-special, non-white space characters". The session cookie name must be unique for each application site, e.g. in an Enterprise Portal/Content scenario, the Portal session cookie name should differ from the Content cookie name.

4. Ensure Virtual Addressing properties have been populated for the Web Profile in use:

In PIA, navigate to "PeopleTools -> Web Profile -> Web Profile Configuration".

Search for the Web Profile in use.

Go to the Virtual Addressing tab and populate your default addressing.

For example, if your load balancer's sign on URL is as follows:

<http://myhost.mycompany.com/ps/signon.html>

You would need to set the following:

Default addressing Protocol: HTTP

Default addressing Name: myhost.mycompany.com

Default addressing Port: 80

Refer to Knowledge Documents:

- 614979.1: *E-PIA: Web Server Load Balancing - Information on Sticky Sessions*
- 653998.1: *E-PIA: PeopleSoft and Load Balancers*
- 747378.1: *Clustering and High Availability for Enterprise Tools 8.4x*

Monitoring the Tuxedo Application Server

There are various tools that can be used to troubleshoot Tuxedo Application Server issues (also known as PeopleSoft Application Server). It is not the purpose of this document to provide Application Server troubleshooting guidelines, nor to replace the Online Performance Guidelines Red Paper. However, we will go over the basic and often times overlooked monitoring data that can help find the root cause of a WebLogic Server crash.

Check How Much Work the PSAPPSRVs are doing

The PSAPPSRV is responsible for the bulk of the work on the application server. It is just as important to start enough PSAPPSRVs as it is to not start too many. By default, each PSAPPSRV has its own Cache folder. For example, if you start 20 PSAPPSRV processes and you only have 5 users, it will take a very long time to build up cache for all 20 PSAPPSRV processes. On the other hand, if you have 5 users and 2 PSAPPSRV processes, the cache should get built up pretty quickly and 2 PSAPPSRVs should be able to handle the number of requests coming in from 5 users.

Keep in mind that app servers on Unix handle requests differently than app servers on Windows. On Unix systems, the requests are distributed evenly amongst the PSAPPSRV servers. So it is normal to see the Rq Done/Load Done columns around the same number for all PSAPPSRVs. (The Load Done column is simply Rq Done multiplied by 50.) On Windows, however, requests are distributed to the first available PSAPPSRV... in which case it is normal to

see the first couple of PSAPPSRVs handling the bulk of the load and the Rq Done/Load Done columns for those PSAPPSRVs having higher values. (Note, the Rq Don/Load Done columns do not get set back to 0 until the app server domain is re-started).

Keep an eye on the Domain Status menu to see how much work the PSAPPSRVs are doing. From the PeopleSoft Domain Administration menu in psadmin, select the Domain Status menu. The Server Status menu will give you an idea of how much work each of the PeopleSoft processes are doing. You may notice that the PSQCKSRV and PSSAMSRV have a low number or 0 for the Rq Done/Load Done columns. This is because these two processes are only used for 3-tier processing. The PSQRYSRV is only used when submitting queries via the Run button in Query Manager. The rest of the work will go to PSAPPSRV.

Domain Status:

```

Command to execute <1-3, q> [q]: 1
Loading command line administration utility ...
tmadmin - Copyright (c) 1996-1999 BEA Systems, Inc.
Portions * Copyright 1986-1997 RSA Data Security, Inc.
All Rights Reserved.
Distributed under license by BEA Systems, Inc.
Muxedo is a registered trademark.
>

```

Prog Name	Queue Name	Grp Name	ID	RqDone	Load Done	Current	Service
BBL.exe	54341	mmanchu+	0	0	0	< IDLE >	
PSMONITORSRV.e	MONITOR	MONITOR	1	0	0	< IDLE >	
PSANALYTICSRV.	00080.00001	ANALYTI+	1	0	0	< IDLE >	
PSAPPSRV.exe	APPQ	APPSRV	1	0	0	< IDLE >	
PSWATCHSRV.exe	WATCH	WATCH	1	0	0	< IDLE >	
PSANALYTICSRV.	00080.00002	ANALYTI+	2	0	0	< IDLE >	
PSAPPSRV.exe	APPQ	APPSRV	2	0	0	< IDLE >	
PSANALYTICSRV.	00080.00003	ANALYTI+	3	0	0	< IDLE >	
PSAPPSRV.exe	APPQ	APPSRV	3	0	0	< IDLE >	
WSL.exe	00001.00020	BASE	20	0	0	< IDLE >	
PSSAMSRV.exe	SAMQ	APPSRV	100	0	0	< IDLE >	
JREPSUR.exe	00094.00250	JREPGRP	250	0	0	< IDLE >	
JSL.exe	00095.00200	JSLGRP	200	0	0	< IDLE >	

```

>

```

How Many Requests are Sitting in Each Server's Queue, Primarily, PSAPPSRVs

Each PeopleSoft process has its own queue. The Queue Status menu will tell you how many requests are sitting in the queue (waiting for one of the servers to free up). It is normal to see some queuing during peak usage. But if you are seeing it often and seeing more than about 5 requests sitting in the queue, this is usually an indication that you need to start more of that particular server. However, if you are seeing queuing on the app server and you also notice

SVCTIMEOUTs in the tuxedo log, this could be pointing to a problem on the database.

Queue Status:

```
Command to execute <1-3, q> [q]: 3
Loading command line administration utility ...
tmadmin - Copyright (c) 1996-1999 BEA Systems, Inc.
Portions * Copyright 1986-1997 RSA Data Security, Inc.
All Rights Reserved.
Distributed under license by BEA Systems, Inc.
Tuxedo is a registered trademark.

> Prog Name      Queue Name      # Serve Wk Queued  # Queued  Ave. Len  Machine
-----
PSWATCHSRV.exe  WATCH          1          -          0          - mmanchuk-+
PSMONITORSRV.e  MONITOR        1          -          0          - mmanchuk-+
PSANALYTICSRV.  00080.00002    1          -          0          - mmanchuk-+
PSANALYTICSRV.  00080.00001    1          -          0          - mmanchuk-+
BBL.exe         54341          1          -          0          - mmanchuk-+
PSSAMSRV.exe    SAMQ           1          -          0          - mmanchuk-+
JREPSRV.exe     00094.00250    1          -          0          - mmanchuk-+
PSAPPSRV.exe    APPQ           3          -          0          - mmanchuk-+
PSANALYTICSRV.  00080.00003    1          -          0          - mmanchuk-+
JSL.exe         00095.00200    1          -          0          - mmanchuk-+
WSL.exe         00001.00020    1          -          0          - mmanchuk-+
>
```

Numerous SVCTIMEOUTs in the TUXLOG

A TUXLOG that shows a large number of SVCTIMEOUTs, is generally an indication of a problem on the database. Check for high CPU usage or memory consumption on the database server. If the database is stressed, it will not return data to the application server in a timely manner, thus triggering a Service Timeout on the Application Server. The Service Timeout is set in the psappsrv.cfg file for each server.

By default, the Service Timeout for PSAPPSRV is set to 300. It is not recommended to raise this value unless you expect users to submit requests that will take longer than 5 minutes to run.

Enough Services or Domains to Handle a Large Number of Users?

It is beneficial to set up multiple domains for application server load balancing and failover. This will distribute the load amongst separate Application Server domains (all pointing to the same database.) If one domain goes down or needs to be brought down, all users will failover to the other domain(s). In order to implement failover, go to the configuration.properties file on the web server and specify the domains in the psserver line:

To enable jolt failover and load balancing, provide a list of application server
 # domains in the format of; psserver=AppSrvr:JSLport,...
 # For example: psserver=SERVER1:9000,SERVER2:9010,SERVER3:9020
 psserver=<machine name>:9000,<machine name>:9050

Client Status:

```

Command to execute <1-3, q> [q]: 2
Loading command line administration utility ...
tmadmin - Copyright (c) 1996-1999 BEA Systems, Inc.
Portions * Copyright 1986-1997 RSA Data Security, Inc.
All Rights Reserved.
Distributed under license by BEA Systems, Inc.
Tuxedo is a registered trademark.

>      LMID      User Name      Client Name      Time      Status      Bgn/Cmnt/Abrt
-----
mmanchuk-us     NT           WSH              4:55:38     IDLE        0/0/0
mmanchuk-us     NT           JSH              4:55:38     IDLE        0/0/0
mmanchuk-us     NT           JSH              4:55:38     IDLE        0/0/0
mmanchuk-us     NT           JSH              4:55:38     IDLE        0/0/0
mmanchuk-us     NT           JSH              4:55:37     IDLE        0/0/0
mmanchuk-us     NT           JSH              4:55:37     IDLE        0/0/0
mmanchuk-us     PS           MMANCHUK-us.us+  0:00:05     IDLE/W      0/0/0
mmanchuk-us     UP1          MMANCHUK-us.us+  0:00:15     IDLE/W      0/0/0
mmanchuk-us     NT           tmadmin          0:00:00     IDLE        0/0/0
>

```

Refer to Knowledge Documents:

- 635570.1: E-AS: Troubleshooting Application Server Performance Issues
- 624231.1: E-AS: PSAPPSRV hung and/or taking up large amounts of CPU or Memory (refer to attach document "PSAPPSRV_hung.ZIP").

Web Server and Application Server Timeouts

There are numerous timeouts defined across the different layers of the PeopleSoft Internet Architecture. In this section, we will discuss the timeouts that directly affect the WebLogic Server/PIA performance.

The general rule to follow is timeout values increase as you get farther from the database server. Draw a diagram so it is easier to see. The farthest (and thus, longest timeout) would be the Load Balancer/Proxy if you have any, followed by the session timeout and session warning defined in the Web Profile, followed by the tuxedo receive timeout (Web Profile > Security), followed by the Jolt Client Cleanup timeout, followed by the application server's service timeouts for the PeopleSoft processes, then the database if there's any timeout at that level. You don't want to have one expire higher up the chain, because threads will then be left processing farther down the line.

In summary, the timeouts should decrease as you go down the following list:

- Load Balancer/Proxy/Firewall
- Web Server Session timeout and session warning (set in Web Profile)
- Tuxedo receive timeout (set in Web Profile)
- Jolt Client Cleanup timeout (set in psappsrv.cfg)
- Application Server services timeouts for the PeopleSoft processes (set in psappsrv.cfg)
- Database

Please note that it is not recommended to set the 'Inactivity Logout' to a high value, because more of the WebLogic memory will be consumed, as the web server will need to retain information, for inactive sessions, for a longer period of time

For more information on timeouts across the PeopleSoft Internet Architecture components, please refer to the PeopleBooks > Enterprise PeopleTools 8.49 PeopleBook: System and Server Administration > System and Server Administration Preface > Appendix: PeopleSoft Timeout Settings.

Also refer to Knowledge Document 653467.1 (E-PIA: Timeout Setting Guidelines for PeopleTools 8.48 and Above).

Conclusion

The topics discussed in this document represent the primary diagnostic techniques that can be used to identify the cause behind WebLogic crashes. Going through this type of troubleshooting before calling Global Customer Support can often times help solve the problem outright. If a subsequent call to Global Customer Support is still necessary to resolve the problem, we will be able to find a solution much faster with the results of the diagnostics that have already been performed.

Contributors:

Karen Driver
Jim Pastorino

Principal Support Engineers
PeopleTools Enterprise Server Tools