

Troubleshooting Weblogic 8.1 Crashes

|

This white paper is a practical guide for technical users, installers, system administrators, and programmers who implement and maintain PeopleSoft Enterprise systems running on WebLogic 8.1. In this white paper, we discuss guidelines on how to diagnose and troubleshoot WebLogic 8.1 crashes. PeopleSoft Online Performance Guidelines, including PeopleSoft Internet Architecture and Portal configuration are not covered in this document.

Much of the information contained in this document originated within the Global Support Center and is therefore based on "real-life" problems encountered in the field. Although every conceivable problem that one could encounter with the web server is not addressed in this document, the issues that appear in this document are the problems that prove to be the most common or troublesome.

Keep in mind that Oracle updates this document as needed so that it reflects the most current feedback we receive from the field. Therefore, the structure, headings, content, and length of this document are likely to vary with each posted version. To see if the document has been updated since you last downloaded it, compare the date of your version to the date of the version posted on Oracle Metalink or Customer Connection.

This material has not been submitted to any formal PeopleSoft test and is published AS IS. It has not been the subject of rigorous review. Oracle assumes no responsibility for its accuracy or completeness. The use of this information or the implementation of any of these techniques is a customer responsibility and depends upon the customer's ability to evaluate and integrate them into their operational environments

Table of Contents

<i>Troubleshooting Weblogic 8.1 Crashes.....</i>	<i>1</i>
<i>Introduction.....</i>	<i>4</i>
<i>Out of Memory Problems (OOM).....</i>	<i>4</i>
<i>Other Memory Leaks Contributing Factors</i>	<i>8</i>
Check WebLogic Service Pack level	8
Check JRE version.....	9
Check OS file descriptor Settings.....	10
Lower OS TCP/IP Cleanup/Timeout Settings	10
<i>Monitoring WebLogic Server Health.....</i>	<i>11</i>
Memory utilization and thread count	11
Monitoring HTTP Sessions	13
HTTP Requests – access log.....	15
<i>Generating a thread dump.....</i>	<i>17</i>
Creating a thread dump on Windows:.....	17
Creating a thread dump on Unix:	17
Reading a thread dump	18
Garbage Collection	21
<i>Monitoring the Tuxedo Application Server.....</i>	<i>23</i>
1. Check how much work the PSAPPSRVs are doing.....	23
2. How many requests are sitting in each server’s queue, primarily, PSAPPSRVs	24
3. Numerous SVCTIMEOUTs in the TUXLOG:.....	25
4. Enough services or domains to handle a large number of users?.....	25
<i>Webserver and Application Server Timeouts</i>	<i>27</i>
Webserver timeouts:	28
Appserver timeouts:	28
PIA/Web Profile timeouts:	29
<i>Conclusion.....</i>	<i>31</i>

Introduction

WebLogic 8.1 is one of the leading J2EE™ application servers in today's marketplace and one of two supported with PeopleTools 8.44 and higher, as well as PeopleTools 8.21 and 8.22. 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
 - Javacores and heapdumps produced on AIX and Linux
- WebLogic running out of threads
 - Webserver stops responding
- Slow performance
- Internal Server Errors 500

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

Out of Memory Problems (OOM)

System Administrators can catch memory and/or 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 identified by looking at the WebLogic log files. Below is an excerpt of a sample out of memory exception:

```
<Mar 7, 2006 11:45:07 AM CST> <Error> <HTTP> <BEA-101017>
<[ServletContext(id=331647017,name=PORTAL,context-path=)] Root cause of ServletException.
java.lang.OutOfMemoryError
  at bea.jolt.NwHdlr.recv(NwHdlr.java(Compiled Code))
  at bea.jolt.CMGr.recv(CMGr.java(Compiled Code))
  at bea.jolt.JoltSession.recv(JoltSession.java(Compiled Code))
  at bea.jolt.JoltRemoteService.call(JoltRemoteService.java(Compiled Code))
  at bea.jolt.JoltRemoteService.call(JoltRemoteService.java(Compiled Code))
  at psft.pt8.net.NetReqRepSvc.sendRequest(NetReqRepSvc.java(Compiled Code))
  at psft.pt8.net.NetService.requestService(NetService.java(Compiled Code))
  at psft.pt8.net.NetReqRepSvc.requestService(NetReqRepSvc.java(Compiled Code))
  at psft.pt8.jb.JBEntry.processRequest(JBEntry.java(Compiled Code))
  at psft.pt8.psc.onActionGen(psc.java(Compiled Code))
```

```

at psft.pt8.psc.onAction(psc.java(Compiled Code))
at psft.pt8.psc.service(psc.java(Compiled Code))
at javax.servlet.http.HttpServlet.service(HttpServlet.java(Compiled Code))
at weblogic.servlet.internal.ServletStubImpl$ServletInvocationAction.run(ServletStubImpl.java(Inlined
Compiled Code))
at weblogic.servlet.internal.ServletStubImpl.invokeServlet(ServletStubImpl.java(Compiled Code))
at weblogic.servlet.internal.TailFilter.doFilter(TailFilter.java(Compiled Code))
at weblogic.servlet.internal.FilterChainImpl.doFilter(FilterChainImpl.java(Inlined Compiled Code))
at psft.pt8.psfiler.doFilter(psfiler.java(Compiled Code))
at weblogic.servlet.internal.FilterChainImpl.doFilter(FilterChainImpl.java(Inlined Compiled Code))
at
weblogic.servlet.internal.WebAppServletContext$ServletInvocationAction.run(WebAppServletContext.java(
Compiled Code))
at weblogic.security.acl.internal.AuthenticatedSubject.doAs(AuthenticatedSubject.java(Compiled Code))
at weblogic.security.service.SecurityManager.runAs(SecurityManager.java(Inlined Compiled Code))
at weblogic.servlet.internal.WebAppServletContext.invokeServlet(WebAppServletContext.java(Compiled
Code))
at weblogic.servlet.internal.ServletRequestImpl.execute(ServletRequestImpl.java(Compiled Code))
at weblogic.kernel.ExecuteThread.execute(ExecuteThread.java(Compiled Code))
at weblogic.kernel.ExecuteThread.run(ExecuteThread.java(Compiled Code))

```

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

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. Many customers have their JVM heap size set to the default heap minimum size of 32MB and maximum size of 200MB. Setting the JVM heap size to a larger minimum value (preferably equal to the maximum value) avoids the performance hit incurred by dynamically growing the JVM and improves predictability; it also lessens the frequency of JVM garbage collection. To set the heap size for PeopleSoft Internet Architecture open the setEnv.cmd/sh file located under <PS_HOME>\weberv\<DOMAIN_NAME> for editing. 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 -Xms32m -Xmx200m -XX:MaxPermSize=128m
```

To increase the heap size, you just need to change the options -Xms and -Xmx as shown here:

```
SET JAVA_OPTIONS_WIN32=-server -Xms512m -Xmx512m -XX:MaxPermSize=128m
```

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 solution# 200766607 on how to install the service(s)).

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

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

Double click on **CmdLine** and change the default values of -ms32m -mx200m to your desired settings. After doing this, restart your service. Sometimes, out of memory exceptions occur when the native heap has been exhausted. Here's a sample exception:

```
<Mar 16, 2006 10:10:20 AM EST> <Error> <HTTP> <BEA-101017>  
<[ServletContext(id=829102,name=PORTAL,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 is set to 1 GB, then the max amount of native memory would be 1 GB.

For RH Linux AS 2.1 – 3 GB is available for the application. 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 sig-abort 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 WebLogic 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, are 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, then this is a **Native OOM**

Other Memory Leaks Contributing Factors

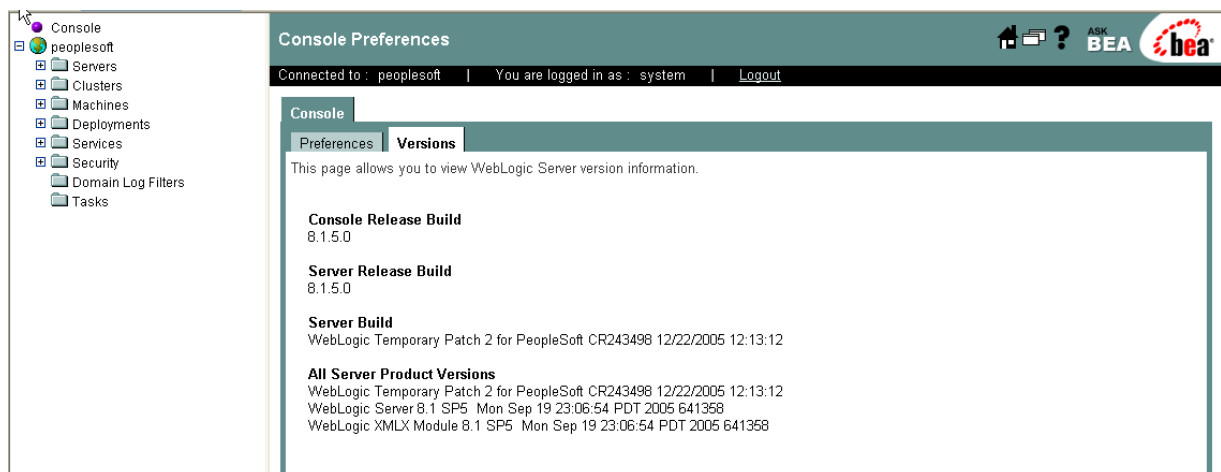
Out of memory problems caused by a small heap or bad implementation planning (physical server not big enough or not enough physical memory) are the easiest to fix. As described above, increasing the heap size or the physical memory can resolve the problem. However, constant memory growth in either java 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 situations are the same as the out of memory situations. Along with the heap size, here are a few more things that need to be checked in any OOM event:

Check WebLogic Service Pack level

For PeopleTools 8.21, 8.22 and 8.44 to 8.46, SP3 is the minimum required although SP5 is recommended. For PeopleTools 8.47 SP4 is the minimum required although SP5 is recommended. To verify the current Service Pack level, check the PIA_weblogic.log. The server startup sequence will show something along these lines:

```
#####May 1, 2006 1:18:07 PM PDT> <Info> <WebLogicServer> <MMANCHUK-us> <PIA> <main>
<<WLS Kernel>> <> <BEA-000214> <WebLogic Server "PIA" version:
  WebLogic Temporary Patch 2 for PeopleSoft CR243498 12/22/2005 12:13:12
  WebLogic Server 8.1 SP5 Mon Sep 19 23:06:54 PDT 2005 641358
  WebLogic XMLX Module 8.1 SP5 Mon Sep 19 23:06:54 PDT 2005 641358 (c) 1995, 1996, 1997,
  1998 WebLogic, Inc.
  (c) 1999, 2000, 2001 BEA Systems, Inc.>
```

Alternatively, you can log into the admin console and on the left hand side menu, click on "Console":



Check JRE version

The Java Runtime Environment (JRE) or Java Development Kit (JDK) is Operating System dependent and is bundled in the WebLogic CD. Depending on the Service Pack and Operating System, different builds of the JRE 1.4.2 are delivered (1.4.2_xx). Also, if you are upgrading from an older version of WebLogic, it is possible that the JAVA_HOME environment variable is pointing to an older version of JRE. The JAVA_HOME environment variable is defined in the setEnv.cmd/sh file. Make sure it is pointing to the location of the JRE delivered with WebLogic. For example, if WebLogic is installed under D:\BEA\wls81, the JRE binaries will be located under D:\BEA\wls81\jdk142_xx\bin, so the JAVA_HOME should point to: SET JAVA_HOME=%BEA_HOME%\jdk142_xx.

There is a script delivered by BEA with WebLogic **called commEnv.cmd/sh** located under BEA_HOME\weblogic81\common\bin (D:\BEA\wls81\weblogic81\common\bin in the example above). This script is invoked by the setEnv.cmd/sh script when the WebLogic Server is started. If JAVA_HOME is not defined in the setEnv.smd/sh (commented out by default), then the JAVA_HOME takes the default defined in the commEnv.cms/sh script. Otherwise, it is overwritten by the JAVA_HOME in setEnv.cmd/sh.

To verify the JRE version, you can also run the following command:

```
D:\MyServers\PT847\webserv\peoplesoft>java -version
```

```
java version "1.4.2_08"  
Java(TM) 2 Runtime Environment, Standard Edition (build 1.4.2_08-b03)  
Java HotSpot(TM) Client VM (build 1.4.2_08-b03, mixed mode)
```

```
D:\MyServers\PT847\webserv\peoplesoft>
```

Operating System vendors publish new JRE builds on their websites on a regular basis. It is a good practice to stay on top of these new builds. PeopleSoft and BEA support newer builds as long as the JRE version used by WebLogic 8.1 remains 1.4.2. Here are a few download links:

Solaris, Windows:

<http://java.sun.com/j2se/1.4.2/download.html>

HP:

<http://www.hp.com/products1/unix/java/index.html>

AIX:

<http://www-128.ibm.com/developerworks/java/jdk/aix/service.html>

Please contact your OS vendor with any JRE download specific questions.

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 appserver, 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. It is implicitly limited by hardware resources, mainly system memory.

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 secs (this one is measured in secs). For other OS platforms, please refer to the Online Performance Guidelines Red Paper posted on Customer Connection or contact your OS vendor:

http://www.peoplesoft.com/media/cupa/pdf/red_paper/rp_e_opcg_845_846_847.htm#c2

Monitoring WebLogic Server Health

In a situation where the PeopleSoft Application performance appears to be degrading or simply as a good practice, monitoring the WebLogic Server(s) resources and the Tuxedo Application Server(s) is a good idea. There are a number of resources that should be monitored regularly:

- Memory utilization and thread count
- HTTP Sessions
- HTTP Requests – access log

Memory utilization and thread count

To monitor JVM memory (heap) and execute thread usage

Log into the adminconsole: <http://webserver:9999/console>

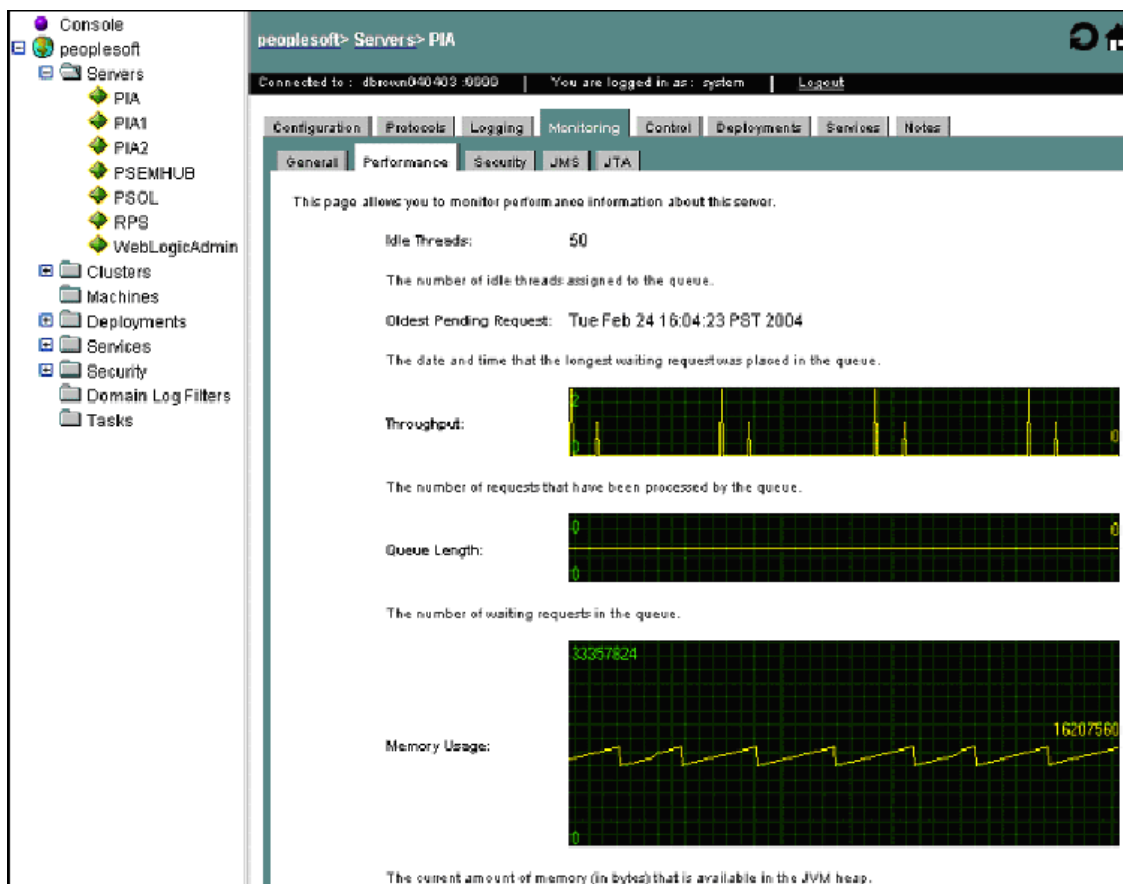
Expand your WebLogic domain (e.g. peoplesoft)

Expand Servers

Select the Server you intend to monitor (e.g. PIA)

Select the first level 'Monitoring' tab

Select the second level 'Performance tab'



Idle Threads: The number of idle threads assigned to the queue.

PendingRequestOldestTime: The time that the longest waiting request was placed in the queue.

Throughput: The number of requests that have been processed by the queue.

Queue Length: The number of waiting requests in the queue.

Memory Usage: The current amount of memory (in bytes) that is available in the JVM heap.

As work enters a WebLogic Server, it is placed in an execute queue. This work is then assigned to a thread within the queue that performs the work.

By default, a new server instance is configured with a default execute queue, `weblogic.kernel.default`, that contains 50 threads. In addition, WebLogic Server provides two other pre-configured queues:

`weblogic.admin.HTTP`—Available only on Administration Servers, this queue is reserved for communicating with the Administration Console; you cannot reconfigure it.

`weblogic.admin.RMI`—Both Administration Servers and Managed Servers have this queue; it is reserved for administrative traffic; you cannot reconfigure it.

PeopleSoft applications use the `weblogic.kernel.default` queue. It is important that you keep an eye on the idle threads to find out the optimal thread count for your environment. A count of 50 threads is the default, but that has proven to be a small pool in large environments, unless multiple WebLogic Servers are setup in a cluster. Also, keeping an eye on the idle threads help catch possible thread issues such as threads not being released causing the WebLogic Server to hang or run out of memory. This problem can also be caught with the help of a thread dump (stuck threads).

To adjust the execute thread count:

Expand WebLogic domain (e.g. peoplesoft)

Expand Servers

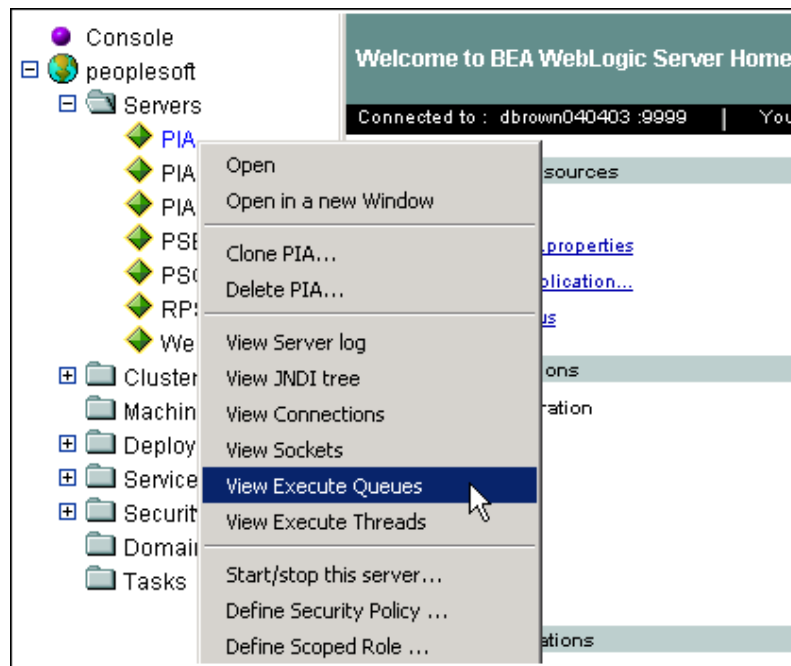
Right click on Server (e.g. PIA)


Select 'View Execute Queue'

Click on the queue that your server is using, most likely 'weblogic.kernel.Default'

Adjust queue size settings to fit needs

Click 'Apply'



<u>Name</u>	<u>Queue Length</u>	<u>Thread Priority</u>	<u>Thread Count</u>	
weblogic.kernel.Default	65536	5	50	

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 expired (e.g. if the user closed the browser). Dangling sessions take up memory, and over time, if there are hundreds or thousands of users in the system, this 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.

It is recommended to monitor the HTTP Sessions, particularly in scenarios where the Enterprise Portal is involved. In a Portal-Content scenario, a user logs into Portal and from there, navigates to the content, either by clicking on a hyperlink or via pagelet. When the user logs out from the Portal application, the Portal is responsible for logging out the content session also. Often times, this is not the case. Some customers have reported that they still see old content sessions hanging out for weeks and even months. If your WebLogic is running out of memory and you see dangling old sessions in the admin console, you may want to report this to the GSC.

How to monitor HTTP Sessions:

Log into the console: <http://webserver:9999/console>

Expand WebLogic domain (e.g. peoplesoft)

Expand Deployments

Expand Applications

Expand peoplesoft

Expand PORTAL

Click on 'Monitoring' tab

Click on 'Sessions' tab

Check 'Session Monitoring Enabled'

Click 'Apply'

Restart WebServer

The screen shot below shows the default monitoring view, which can be customized. PeopleSoft user ID, IP address and PIA site to which the user is connected is shown in the "Main Attribute" column. The "Time Last Accessed" column shows the time when the last user-server interaction occurred. The third column shows the server to which the user is connected and the "Name" column displays the session ID.

The screenshot displays the WebLogic Admin Console interface. On the left, a navigation tree shows the hierarchy: Console > peoplesoft > Servers > PIA > PIA1 > PIA2 > PIA3 > PSEMHUB > PSOL > RPS > WebLogicAdmin. The main panel shows the 'peoplesoft' domain expanded, with 'Applications' > 'peoplesoft' > 'PORTAL' selected. The 'Monitoring' tab is active, and the 'Sessions' sub-tab is selected. The page title is 'peoplesoft> Applications> peoplesoft> PORTAL'. Below the title bar, it says 'Connected to : peoplesoft | You are logged in as : system | Logout'. The 'Sessions' tab contains a message: 'This page allows you to view statistics about all of the sessions that are currently active for this Web application. You can also customize the information that is presented by clicking the Customize this view... link.' Below this, it states 'If the Web application is deployed as an exploded archive rather than as a WAR or part of an EAR, the Session Monitoring Enabled check box appears. Use this to specify whether you want session monitoring enabled for this Web application.' A checkbox labeled 'Session Monitoring Enabled' is checked. Below it, a link 'Customize this view...' is visible. A table with four columns: 'Main Attribute', 'Time Last Accessed', 'Server', and 'Name' displays two active sessions. The first session is for user 'VP1' at IP '10.138.234.239' on server 'PIA'. The second session is for user 'PS' at IP '10.138.234.239' on server 'PIA'. An 'Apply' button is at the bottom right of the table.

Main Attribute	Time Last Accessed	Server	Name
VP1@10.138.234.239/C881G71P	Mon May 08 14:28:45 PDT 2006	PIA	Gf4JsFyFxFwWXB9Jp1zQkMnyYqcybT1! 1147123721343
PS@10.138.234.239/C881G71P	Mon May 08 14:28:54 PDT 2006	PIA	Gf4ST15T5lhXp1ZnHYT2QMpJ4ZPVbIdp! 1147123730328

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:

Log into the console: <http://webserver:9999/console>

Expand WebLogic domain (e.g. peoplesoft)

Expand Servers

Select the Server you intend to monitor (e.g. PIA)

On right pane, select Logging > HTTP.

On the HTTP tab, click Enable HTTP Logging.

In the Format list, determine format of the HTTP log file by selecting Common or Extended.

If extended format is selected, at the beginning of the access log you will have to specify which information you want to be logged. Enterprise Development normally requests that data such as date, time, method, user, URI and cookie name be logged. In this case, the following two lines need to be manually added at the beginning of the PIA_access.log.

#Version: 1.0

#Fields: date time c-ip s-ip cs-method time-taken sc-status bytes cs(User-Agent) cs-uri cs(Cookie)

Please refer to solution# 200958613 for more details on log size and format.

Below is a sample excerpt of a common access log:

```
10.138.234.239 - - [08/May/2006:11:56:42 -0700] "GET /C881G71P/signon.html HTTP/1.1" 304 0
10.138.234.239 - - [08/May/2006:11:56:44 -0700] "GET /psp/C881G71P/?cmd=login HTTP/1.1" 200 8628
10.138.234.239 - - [08/May/2006:11:56:52 -0700] "GET /C881G71P/signin.css HTTP/1.1" 200 2544
10.138.234.239 - - [08/May/2006:11:56:52 -0700] "GET /C881G71P/images/ps_logo.gif HTTP/1.1" 304 0
10.138.234.239 - - [08/May/2006:11:56:52 -0700] "GET
/C881G71P/images/ENG/people_image_lf_ENG.gif HTTP/1.1" 200 567
10.138.234.239 - - [08/May/2006:11:56:52 -0700] "GET
/C881G71P/images/ENG/people_image_rt_ENG.gif HTTP/1.1" 304 0
10.138.234.239 - - [08/May/2006:11:56:52 -0700] "GET /C881G71P/images/language_cze.gif HTTP/1.1"
304 0
10.138.234.239 - - [08/May/2006:11:56:53 -0700] "GET /C881G71P/images/language_japanese.gif
HTTP/1.1" 304 0
10.138.234.239 - - [08/May/2006:11:56:53 -0700] "GET /C881G71P/images/korean.gif HTTP/1.1" 304 0
10.138.234.239 - - [08/May/2006:11:56:53 -0700] "GET /C881G71P/images/language_tha.gif HTTP/1.1"
304 0
10.138.234.239 - - [08/May/2006:11:56:53 -0700] "GET /C881G71P/images/language_rus.gif HTTP/1.1"
304 0
10.138.234.239 - - [08/May/2006:11:56:53 -0700] "GET /C881G71P/images/schinese.gif HTTP/1.1" 200
151
10.138.234.239 - - [08/May/2006:11:56:53 -0700] "GET /C881G71P/images/arabic.gif HTTP/1.1" 304 0
10.138.234.239 - - [08/May/2006:11:56:53 -0700] "GET /C881G71P/images/tchinese.gif HTTP/1.1" 200
154
10.138.234.239 - - [08/May/2006:11:56:53 -0700] "GET /C881G71P/images/blue_stripe_chip.gif HTTP/1.1"
304 0
```

```
10.138.234.239 - - [08/May/2006:11:56:53 -0700] "GET /C881G71P/images/white_shadow_chip.gif
HTTP/1.1" 304 0
10.138.234.239 - - [08/May/2006:11:56:53 -0700] "GET /C881G71P/images/blue_shadow_chip.gif
HTTP/1.1" 200 72
10.138.234.239 - - [08/May/2006:11:56:58 -0700] "POST /psp/C881G71P/?cmd=login&languageCd=ENG
HTTP/1.1" 302 351
10.138.234.239 - - [08/May/2006:11:57:00 -0700] "GET
/psp/C881G71P/EMPLOYEE/CRM/h/?tab=DEFAULT HTTP/1.1" 200 7356
```

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

```
10.138.234.239 - - [08/May/2006:11:56:53 -0700] "GET /C881G71P/images/blue_shadow_chip.gif
HTTP/1.1" 200 72
```

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, unstandardized 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>

Generating a thread dump

If you have to contact the Global Support Center, you will most likely be asked to provide a thread dump. A thread dump is a file that shows the status of each thread at the time it was generated. It is written to the PIA_stdout.log, unless WebLogic is running as a Windows Service, in which case it is written to the NT Service log.

Creating a thread dump on Windows:

Option 1:

Open a command prompt and enter the following command:

```
WL_HOME\bin\beasvc -dump -svcname:service-name
```

where WL_HOME is the directory in which you installed WebLogic Server and service-name is the Windows service that is running a server instance.

For example, go to D:\wls81\weblogic81\server\bin> and run the following
beasvc -dump -svcname:peoplesoft-PIA

This will create the thread dump in NTservice-peoplesoft-PIA.log file.

Option 2:

Go to PS_HOME\webserve\peoplesoft. Run the script called createthreaddump.cmd. This will create the thread dump in the NTservice-peoplesoft-PIA.log file.

Usage: createthreaddump 'hostname/IP:http_port' [username] [password]

Hostname/IP:http_port: Hostname or IP address and HTTP port of WebLogic instance which you would like to have generate a thread dump.

Username: WebLogic username which will be used to connect to WebLogic.

Password: Password for WebLogic username which will be used to connect to WebLogic.

If a username and password are not specified, the WLS username and password specified in setEnv will be used.

Creating a thread dump on Unix:

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

This will create the thread dump in the nohup.out file if this is being used by WebLogic or PIA_stdout.log. On AIX systems, look for a javacore file with ".txt" extension.

Note: On Linux, each execute thread appears as a separate process under the Linux process stack. To use Kill -3 on Linux you must supply the PID of the main WebLogic execute thread, otherwise no thread dump will be produced. In other words, PID should be the root of the process tree.

To obtain the root PID, perform a:

```
ps -efHl | grep 'java' **. **
```

using a grep argument that is a string that will be found in the process stack that matches the server startup command. The first PID reported will be the root process, assuming that the ps command has not been piped to another routine.

Reading a thread dump

Edit the PIA_stdout.log or the NTservice-peoplesoft-PIA.log depending on your OS. 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, 2005 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: /psft/PAPROD/EMPLOYEE/PAPROD/h/", which is more than the configured time
(StuckThreadMaxTime) of "600" seconds.>
[Mon Jan 10 12:48:21 2005] [I] [ControlHandler] 1
Full thread dump Java HotSpot(TM) Server VM (1.4.2_03-b02 mixed mode):
```

```
"NwWriter" daemon prio=5 tid=0x385ddbfb8 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)
```

```
.
.
.
.
```

```
"ExecuteThread: '262' for queue: 'weblogic.kernel.Default'" daemon prio=5 tid=0x390607b8 nid=0x8f8
waiting on condition [3dd6f000..3dd6fdb0]
at java.lang.Thread.sleep(Native Method)
at psft.pt8.util.PSHttpUtil.waitLogoutProcess(PSHttpUtil.java:1510)
at psft.pt8.psp.service(psp.java:428)
at javax.servlet.http.HttpServlet.service(HttpServlet.java:853)
at weblogic.servlet.internal.ServletStubImpl$ServletInvocationAction.run(ServletStubImpl.java:971)
at weblogic.servlet.internal.ServletStubImpl.invokeServlet(ServletStubImpl.java:402)
```

```

at weblogic.servlet.internal.TailFilter.doFilter(TailFilter.java:28)
at weblogic.servlet.internal.FilterChainImpl.doFilter(FilterChainImpl.java:27)
at psft.pt8.psfilter.doFilter(psfilter.java:76)
at weblogic.servlet.internal.FilterChainImpl.doFilter(FilterChainImpl.java:27)
at
weblogic.servlet.internal.WebAppServletContext$ServletInvocationAction.run(WebAppServletContext.java:6374)
at weblogic.security.acl.internal.AuthenticatedSubject.doAs(AuthenticatedSubject.java:317)
at weblogic.security.service.SecurityManager.runAs(SecurityManager.java:118)
at weblogic.servlet.internal.WebAppServletContext.invokeServlet(WebAppServletContext.java:3645)
at weblogic.servlet.internal.ServletRequestImpl.execute(ServletRequestImpl.java:2585)
at weblogic.kernel.ExecuteThread.execute(ExecuteThread.java:197)
at weblogic.kernel.ExecuteThread.run(ExecuteThread.java:170)

```

WebLogic Server and the Tuxedo Application Server use Jolt to talk to each other. PIA creates two threads inside the WebLogic's JVM per jolt connection. So for each jolt connection made between WebLogic and the Tuxedo Application Servers in the cluster, you will see a `bea.jolt.NwReader` and a `bea.jolt.NwWriter` thread in the thread dump. This is not, however, the main information the Global Support Center and development look for in a thread dump.

If you scroll down to the Execute Threads, like the "ExecuteThread: '262'" shown in the example above, 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. What we really look for in these call stacks is, *who's code was being executed* when the thread dump was generated: WebLogic's, Java, PeopleSoft's or any other application's. You want to identify the last code that was being executed, right when the problem happened.

In the example above, ExecuteThread: '262' was executing `PSHttpUtil.waitLogoutProcess` from `PSHttpUtil.java`, is PeopleSoft's code. You can tell it's PeopleSoft code if the line starts with "at psft...". If you see all the execute threads or many of them "stuck" at the same place, then we have found the root cause of the problem and it's up to development to find a solution. 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 you see that the threads are "moving", meaning they were caught executing different pieces of code each time, then that is not an issue and you will need to further monitor and investigate the source of the problem.

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

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)

```

Thread waiting for a response from the Tuxedo Application Server:

```
"ExecuteThread: '30' for queue: 'weblogic.kernel.Default'" daemon prio=10 tid=00095ab0 nid=43
lwp_id=6973558 in Object.wait() [0x40157000..0x401564f0]
  at java.lang.Object.wait(Native Method)
  at bea.jolt.IOBuf.waitOnBuf(IOBuf.java:119)
  - locked <6e8d2dc0> (a bea.jolt.IOBuf)
  at bea.jolt.NwHdlr.recv(NwHdlr.java:1413)
  at bea.jolt.CMGr.recv(CMGr.java:126)
  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:565)
  at psft.pt8.net.NetService.requestService(NetService.java:141)
  at psft.pt8.net.NetReqRepSvc.requestService(NetReqRepSvc.java:329)
  - locked <6e8d5428> (a psft.pt8.net.NetSession)

  at psft.pt8.jb.JBEntry.processRequest(JBEntry.java:340)
  - locked <6e8d55b0> (a psft.pt8.util.JBStatusBlock)
  at psft.pt8.psc.onActionGen(psc.java:1689)
  at psft.pt8.psc.onActionDirect(psc.java:1334)
  - locked <439d66f8> (a java.lang.String)
  at psft.pt8.portal.PIADirectConnection.connect(Unknown Source)
  at psft.pt8.portal.ContentGetter.getPIADirectContent(Unknown Source)
  at psft.pt8.portal.ContentGetter.getPIADirectContent(Unknown Source)
  at psft.pt8.portal.ContentGetter.getPIADirectContent(Unknown Source)
  at psft.pt8.portal.ContentGetter.getContent(Unknown Source)
  at psft.pt8.portal.ContentGetter.getContent(Unknown Source)
  at psft.pt8.portal.ContentGetter.getContent(Unknown Source)
  at psft.pt8.portal.template.IClientComponent.doGetContent(Unknown Source)
  at psft.pt8.portal.template.IClientComponent.getContent(Unknown Source)
  at psft.pt8.portal.PageAssembler.getPIAContent(Unknown Source)
  at psft.pt8.portal.PageAssembler.replaceComponent(Unknown Source)
  at psft.pt8.portal.PageAssembler.replaceComponentTags(Unknown Source)
  at psft.pt8.portal.PageAssembler.assemble(Unknown Source)
  at psft.pt8.psp.onStartTab(Unknown Source)
  at psft.pt8.psp.service(Unknown Source)
  at javax.servlet.http.HttpServlet.service(HttpServlet.java:853)
  at weblogic.servlet.internal.ServletStubImpl$ServletInvocationAction.run(ServletStubImpl.java:1006)
  at weblogic.servlet.internal.ServletStubImpl.invokeServlet(ServletStubImpl.java:419)
  at weblogic.servlet.internal.TailFilter.doFilter(TailFilter.java:28)
  at weblogic.servlet.internal.FilterChainImpl.doFilter(FilterChainImpl.java:27)
  at psft.pt8.psfiler.doFilter(psfiler.java:76)
  at weblogic.servlet.internal.FilterChainImpl.doFilter(FilterChainImpl.java:27)
  at
weblogic.servlet.internal.WebAppServletContext$ServletInvocationAction.run(WebAppServletContext.java:
6724)
  at weblogic.security.acl.internal.AuthenticatedSubject.doAs(AuthenticatedSubject.java:321)
  at weblogic.security.service.SecurityManager.runAs(SecurityManager.java:121)
  at weblogic.servlet.internal.WebAppServletContext.invokeServlet(WebAppServletContext.java:3764)
  at weblogic.servlet.internal.ServletRequestImpl.execute(ServletRequestImpl.java:2644)
  at weblogic.kernel.ExecuteThread.execute(ExecuteThread.java:219)
  at weblogic.kernel.ExecuteThread.run(ExecuteThread.java:178)
```

PeopleSoft thread:

```
"ExecuteThread: '192' for queue: 'weblogic.kernel.Default'" daemon prio=5 tid=0x38cec9f0 nid=0x7e0
waiting on condition [3cbef000..3cbefdb0]
```

```

at java.lang.Thread.sleep(Native Method)
at psft.pt8.util.PSHttpUtil.waitLogoutProcess(PSHttpUtil.java:1510)
at psft.pt8.psp.service(psp.java:428)
at javax.servlet.http.HttpServlet.service(HttpServlet.java:853)
at weblogic.servlet.internal.ServletStubImpl$ServletInvocationAction.run(ServletStubImpl.java:971)
at weblogic.servlet.internal.ServletStubImpl.invokeServlet(ServletStubImpl.java:402)
at weblogic.servlet.internal.TailFilter.doFilter(TailFilter.java:28)
at weblogic.servlet.internal.FilterChainImpl.doFilter(FilterChainImpl.java:27)
at psft.pt8.psfiler.doFilter(psfiler.java:76)
at weblogic.servlet.internal.FilterChainImpl.doFilter(FilterChainImpl.java:27)
at
weblogic.servlet.internal.WebAppServletContext$ServletInvocationAction.run(WebAppServletContext.java:6374)
at weblogic.security.acl.internal.AuthenticatedSubject.doAs(AuthenticatedSubject.java:317)
at weblogic.security.service.SecurityManager.runAs(SecurityManager.java:118)
at weblogic.servlet.internal.WebAppServletContext.invokeServlet(WebAppServletContext.java:3645)
at weblogic.servlet.internal.ServletRequestImpl.execute(ServletRequestImpl.java:2585)
at weblogic.kernel.ExecuteThread.execute(ExecuteThread.java:197)
at weblogic.kernel.ExecuteThread.run(ExecuteThread.java:170)

```

If the thread dump shows many threads waiting on the Tuxedo Application Server, then you clearly will have to look at the Tuxedo Application Server overall status: client, queue and server. We will discuss that 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 enabled, like verbose garbage collection.

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 env. Variable:

For example, in a Windows server:

```
SET JAVA_OPTIONS_WIN32=-server -Xms512m -Xmx512m -XX:MaxPermSize=128m -verbosegc
```

This will print GC activity info to the stdout/stderr log. Redirect the stdout/stderr to a file. Run the application until the problem gets reproduced.

Here's a sample excerpt of a gc log:

```

36474.196: [Full GC 273281K->107947K(471872K), 2.4162651 secs]
36940.867: [Full GC 265259K->63150K(471872K), 1.7558893 secs]
37420.880: [GC 220453K->85813K(471872K), 0.1842687 secs]
37973.671: [GC 243125K->92402K(471872K), 0.2165158 secs]
38458.415: [GC 249671K->90848K(471872K), 0.2028610 secs]
38782.157: [GC 239723K->109377K(471872K), 0.2799934 secs]
39071.539: [Full GC 266689K->84653K(471872K), 1.6340919 secs]
39594.731: [GC 241965K->103113K(471872K), 0.1444709 secs]

```

```
39994.236: [GC 260425K->110009K(471872K), 0.1860339 secs]
40382.497: [Full GC 267320K->67184K(471872K), 1.7912783 secs]
40807.809: [GC 224496K->93729K(471872K), 0.2150907 secs]
41286.643: [GC 251022K->96573K(471872K), 0.2244152 secs]
```

Format varies depending on the JVM:

```
[memory ] 7.160: GC 131072K->130052K (131072K) in 1057.359 ms
```

```
[memory ] <start>: GC <before>K-><after>K (<heap>K), <total> ms
```

```
[memory ] <start> - start time of collection (seconds since jvm start)
```

```
[memory ] <before> - memory used by objects before collection (KB)
```

```
[memory ] <after> - memory used by objects after collection (KB)
```

```
[memory ] <heap> - size of heap after collection (KB)
```

```
[memory ] <total> - total time of collection (milliseconds)
```

If the GC cycle doesn't happen before java OOM, then there could be a JVM bug. However, if the GC cycle happens but just little memory gets released, it could be that either the application is not "marking" its objects as "garbage collectable" or there is a JVM bug.

Full Compaction:

Another possible cause of OOM is full compaction. Java objects need the memory to be contiguous. If the available free memory is fragmented, then the JVM will not be able to allocate a large object, as it may not fit in any of the available free chunks. In this case, the JVM should do a full compaction so that more contiguous free memory can be formed to accommodate large objects.

Compaction work involves moving of objects (data) from one place to another in the java heap memory and updating the references to those objects to point to the new location. JVMs may not compact all the objects unless there is a need. This is to reduce the pause time of GC cycle.

We can check whether the java OOM is due to fragmentation by analyzing the verbose gc messages. If you see output similar to the following where the OOM is being thrown even when there is free java heap available, then it is due to fragmentation.

```
[memory ] 8.162: GC 73043K->72989K (131072K) in 12.938 ms
[memory ] 8.172: GC 72989K->72905K (131072K) in 12.000 ms
[memory ] 8.182: GC 72905K->72580K (131072K) in 13.509 ms
java.lang.OutOfMemoryError
```

In the above case you can see that the max heap specified was 128MB and the JVM threw OOM when the actual memory usage was only 72580K. The heap usage is only 55%. Therefore, the effect of fragmentation in this case is to throw OOM even when there is 45% of free heap. This is a JVM bug or limitation. You should contact the JVM vendor.

For more info on tuning JVM please refer to the following document:

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.

In PeopleTools 8.44 and higher releases, Performance Monitor is available (please see the 8.44 Release Notes and PeopleBooks for more details). It can help determine if the application server is the bottleneck. If Performance Monitor has not been configured, there are a number of other things that can be looked at:

1. 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.
Tuxedo 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    ANALYTIC+     1      0      0 < IDLE >
PSAPPSRV.exe    APPQ           APPSRV         1      0      0 < IDLE >
PSWATCHSRV.exe  WATCH         WATCH          1      0      0 < IDLE >
PSANALYTICSRV.  00080.00002    ANALYTIC+     2      0      0 < IDLE >
PSAPPSRV.exe    APPQ           APPSRV         2      0      0 < IDLE >
PSANALYTICSRV.  00080.00003    ANALYTIC+     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 >
JREPSRV.exe     00094.00250    JREPGRP       250      0      0 < IDLE >
JSL.exe         00095.00200    JSLGRP        200      0      0 < IDLE >

>
```

2. 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. (See number 3.)

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
-----
PSWATCHSRU.exe  WATCH          1          -          0          - mmanchuk-+
PSMONITORSRU.e  MONITOR         1          -          0          - mmanchuk-+
PSANALYTICSRU.  00080.00002    1          -          0          - mmanchuk-+
PSANALYTICSRU.  00080.00001    1          -          0          - mmanchuk-+
BBL.exe         54341          1          -          0          - mmanchuk-+
PSSAMSRU.exe    SAMQ            1          -          0          - mmanchuk-+
JREPSUR.exe     00094.00250    1          -          0          - mmanchuk-+
PSAPPSRU.exe    APPQ            3          -          0          - mmanchuk-+
PSANALYTICSRU.  00080.00003    1          -          0          - mmanchuk-+
JSL.exe         00095.00200    1          -          0          - mmanchuk-+
WSL.exe         00001.00020    1          -          0          - mmanchuk-+

>
```

3. Numerous SVCTIMEOUTs in the TUXLOG:

When checking the TUXLOG and you notice a large number of SVCTIMEOUTs, this 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 app 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.

4. 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 app server domains (all pointing to the same database.) And if one domain goes down or needs to be brought down, all users will failover to the other domain(s). In order to implement this, 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/Cmmt/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

```

>

```

For more details, see:

Solution 200949701 - E-AS: Troubleshooting application server performance issues

Solution 200776884 - E-AS: The PSAPPSRV process appears to be hung and/or taking up large amounts of CPU.

Webserver 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 servers. 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 configuration.properties/WebProfile depending on your PeopleTools version, followed by the tuxedo receive timeout (pstools.properties or WebProfile, Security page depending on your PeopleTools version), followed by the Jolt Client Cleanup timeout, followed by the application server service timeouts for the PeopleSoft processes, then the database if any. You don't want to have one expire higher up the chain, because threads will then be left processing farther down the line.

Webserver timeouts:

SessionTimeout – configuration.properties, PT8.2x. The inactive interval permitted for a user's PIA/Portal usage session. This is akin to an idle browser auto logout time.
session-timeout - configuration.properties is replaced by the Web Profile Inactivity timeouts in PeopleTools 8.44 and higher. See PIA timeouts below.

tuxedo_receive_timeout – pstools.properties, PT8.2x: The amount of time permitted to elapse between the issue of a Jolt Request from the client (servlet) and the arrival of the ensuing response from the application server. This value should be set equal or greater than the maximum online service timeout value on the application server. This value should be considerably larger than the tuxedo_send_timeout.

session-timeout - web.xml: Determines the time period that can elapse before the web server (WebLogic) will remove the HttpSession. This is akin to an abandoned session cleanup timeout.

If this value is set less than the sessionTimeout in configuration.properties in PeopleTools 8.2x it will not result in the termination of the user's online session. The HttpSession will be removed but the user's session will remain valid due to the presence of cookie in the user's browser. Adjusting this setting will affect the user by causing their state (which is stored in the HttpSession) to be lost.

If this setting is too high it will affect resource utilization on the web server.

Ideally the value of this setting will be the same as the sessionTimeout in configuration.properties for PeopleTools 8.2x. This will prevent state loss and dangling HttpSession on the webserver.

session-timeout - web.xml is superseded by the Web Profile timeouts in PeopleTools 8.44 and higher.

Appserver timeouts:

JOLT Listener / Client Cleanup Timeout: The inactive interval permitted for the server-side JoltSession.

Raising this value can keep unnecessary server-side resources allocated. Setting this value too low can cause the re-instantiation of resources for a client who has surpassed their inactivity interval.

Note: this value will not affect the user experience; it has a server side performance impact. Evidence has come to light that an optimal value for this setting is ~5 minutes. This was shown to relieve concurrency issues.

Appserver processes /Service Timeout: The time period permitted for the service to run in the process in question. If the service has not completed within the allotted time period Tuxedo will terminate the server processing being run and then restart the server process.

This value should be set to the longest that any service is expected to take for that particular server.

PIA/Web Profile timeouts:

Most of the following values formerly resided in the Web Server. These settings are now stored on the database and primarily accessible from PIA. The defaults here are determined by the WebProfile parameter in the configuration.properties file on the web server.

The specific profile type is prompted during the PIA installation but can be changed directly post installation in the configuration.properties file.

Profiles include

DEV: standard developer options

TEST: standard testing options

PROD: standard production options

KIOSK: like PROD but has additional settings relevant to KIOSK usage

HTTP Session Inactivity (authenticated users): this field specifies the interval of inactivity after which time the web server will release their HTTP session. This will cause PIA to release their application states from memory. If the user takes another link they will regain access to the application at the search dialog if it's a public user. Authenticated users will see an Expire Page message. If not set, the http interval for an authenticated user is the same value as the inactivity logout. It is specified in seconds. This setting prevents an overload of web server resources for inactive public users.

Receive Timeout: same as the pstools.properties tuxedo_receive_timeout used in PeopleTools 8.2x. The amount of time permitted to elapse between the issue of a Jolt Request from the client (servlet) and the arrival of the ensuing response from the application server

A number of additional timeouts may be set through PIA but will not be discussed in this document. Please refer to the Internet Technology PeopleBooks for more details.

Check the PIA_weblogic.log for timeout exceptions like this one:

```
[Mon Jan 23 08:07:52 CST 2006]bea.jolt.ServiceException: bea.jolt.JoltRemoteService(ICPanel)call():
Timeout\nbea.jolt.SessionException: Connection recv error\nbea.jolt.JoltException: [3] NwHdlr.recv():
Timeout Error
```

A Jolt error has occurred while communicating with the Application server. Cancel the current operation and retry. If the problem persists contact your system administrator. Error Code:100

Application Server last connected //pamarq.mu.edu_50750

```
bea.jolt.ServiceException: bea.jolt.JoltRemoteService(ICPanel)call(): Timeout\nbea.jolt.SessionException:
Connection recv error\nbea.jolt.JoltException: [3] NwHdlr.recv(): Timeout Error
```

```
  at bea.jolt.JoltRemoteService.call(JoltRemoteService.java:350)
  at bea.jolt.JoltRemoteService.call(JoltRemoteService.java:257)
  at psft.pt8.net.NetReqRepSvc.sendRequest(NetReqRepSvc.java:565)
  at psft.pt8.net.NetService.requestService(NetService.java:141)
  at psft.pt8.net.NetReqRepSvc.requestService(NetReqRepSvc.java:329)
  at psft.pt8.jb.JBEntry.processRequest(JBEntry.java:340)
  at psft.pt8.psc.onActionGen(psc.java:1688)
  at psft.pt8.psc.onActionDirect(psc.java:1334)
  at psft.pt8.portal.PIADirectConnection.connect(Unknown Source)
  at psft.pt8.portal.ContentGetter.getPIADirectContent(Unknown Source)
  at psft.pt8.portal.ContentGetter.getPIADirectContent(Unknown Source)
  at psft.pt8.portal.ContentGetter.getPIADirectContent(Unknown Source)
  at psft.pt8.portal.ContentGetter.getContent(Unknown Source)
  at psft.pt8.portal.ContentGetter.getContent(Unknown Source)
  at psft.pt8.portal.ContentGetter.getContent(Unknown Source)
  at psft.pt8.portal.template.IClientComponent.doGetContent(Unknown Source)
  at psft.pt8.portal.template.IClientComponent.getContent(Unknown Source)
  at psft.pt8.portal.PageAssembler.getPIAContent(Unknown Source)
  at psft.pt8.portal.PageAssembler.replaceComponent(Unknown Source)
  at psft.pt8.portal.PageAssembler.replaceComponentTags(Unknown Source)
  at psft.pt8.portal.PageAssembler.assemble(Unknown Source)
  at psft.pt8.psp.onStartTab(Unknown Source)
  at psft.pt8.psp.service(Unknown Source)
  at javax.servlet.http.HttpServlet.service(HttpServlet.java:853)
  at weblogic.servlet.internal.ServletStubImpl$ServletInvocationAction.run(ServletStubImpl.java:1006)
  at weblogic.servlet.internal.ServletStubImpl.invokeServlet(ServletStubImpl.java:419)
  at weblogic.servlet.internal.TailFilter.doFilter(TailFilter.java:28)
  at weblogic.servlet.internal.FilterChainImpl.doFilter(FilterChainImpl.java:27)
  at psft.pt8.psfiler.doFilter(psfiler.java:76)
  at weblogic.servlet.internal.FilterChainImpl.doFilter(FilterChainImpl.java:27)
  at
weblogic.servlet.internal.WebAppServletContext$ServletInvocationAction.run(WebAppServletContext.java:
6724)
  at weblogic.security.acl.internal.AuthenticatedSubject.doAs(AuthenticatedSubject.java:321)
  at weblogic.security.service.SecurityManager.runAs(SecurityManager.java:121)
  at weblogic.servlet.internal.WebAppServletContext.invokeServlet(WebAppServletContext.java:3764)
  at weblogic.servlet.internal.ServletRequestImpl.execute(ServletRequestImpl.java:2644)
  at weblogic.kernel.ExecuteThread.execute(ExecuteThread.java:219)
  at weblogic.kernel.ExecuteThread.run(ExecuteThread.java:178)
```

This exception indicates that a Jolt error has occurred while waiting for a response from the Application server. Most likely in this case, the receive timeout is lower than the service timeout of the Appserver process processing the request. Since the call is to

ICPanel, PSAPPSRV is the process in question. In this example you would want to make sure that the Receive Timeout is higher than the PSAPPSRV Service timeout in psappsrv.cfg. Timeout mismatches can lead to dangling objects in memory that over time, can affect the WebLogic performance.

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 the GSC can often times help solve the problem outright. If a subsequent call to the GSC 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.