Troubleshooting Deployment

Broken Pipe

A current app I'm working freezes up for about a minute when calling editingContext.saveChanges. The following message is added to the system output:

```
<WorkerThread3> <WOWorkerThread id=3 socket=Socket[addr=/xxx.xxx.xxx.xxx,port=51634,localport=51563]>
Exception while sending response: java.net.SocketException: Broken pipe
```

The odd thing is that no exception is actually being thrown and the changes are saved to the database. Any clues out there about what this message means?

Chuck Hill

This message means that the woadaptor did not receive a response from your application within the alloted amount of time. At this point, the woapdator hangs up on your application assuming that it is dead or overly busy. If you have multiple instances running, the woadaptor will then forward the request to another instance. For component actions this will produce a session expired or can't restore page error message.

When your application finally finishes processing the request it attempts to return the result to the woadaptor only to discover that it has hung up on it (aka closed the socket, broken the pipe). At this point the exception above is output to the log.

There are a few ways to avoid this:

- Optimize your application so that the request is processed more quickly.
- Use the WOLongReponsePage to handle long running requests.
- Increase the Connect Timeout and Receive Timeout values in JavaMonitor so that the woadaptor will wait long enough for your application to provide the response.

If this message happens in other contexts (i.e. not when a request takes a long time to process) it might just mean that the user hit stop in their browser or clicked another link.

Where's my stderr!?

Wotaskd launches new WOA instances using a script called SpawnOfWotaskd.sh that is located in /System/Library/WebObjects/JavaApplications/wotaskd.woa/Contents/Resources/SpawnOfWotaskd.sh on OS X. For some reason, this script was written to throw away stdout and redirrect stderr to stdout. This means that if you ever want to get a thread stack dump, you're out of luck. Fortunately this is an easy fix. If you edit SpawnOfWotaskd.sh, the stock version looks like:

```
#!/bin/sh
$@ 1>/dev/null 2>&1 &
```

Notice that stream 1 goes to /dev/null, and stream 2 writes to stream 1. Boo. Change it to:

```
#!/bin/sh
$@ 1>>/var/log/webobjects.err 2>&1 &
```

The one problem now is that typically WebObjects apps run as appserver user, which means they won't be able to write to this file. To fix this, you can 'touch /var/log/webobjects.err' as root, and then chown the blank file to whatever user your WebObjects apps run as.

Can't connect to window server - not enough permissions

Ah yes, the joys of AWT. If you touch any AWT class (event the ones that don't make sense, like Dimension, or Rectangle), an AWT Toolkit will be created (in a static block) and AWT will attempt to connect to the window server. Of course when you ran in development mode, you had access to the window server and everything was peachy. As soon as you tried to deploy as the appserver user, all hell broke loose. You have a couple of options for fixing this one:

- Just stop using them. This is the most sure fire way to not get hit by this problem. Don't use AWT classes when you don't need to. Obviously this is often easier said than done.
- Add -Djava.awt.headless=true to your VM launch parameters. This tells AWT to use headless mode. If you are using older versions of Apple's JVM, this may still cause issues (weird debug statements requesting for you to hit a key on the console).
- PJA Toolkit. This is actually kind of a cool library. It's a pure java implementation of an AWT toolkit. On a normal system, your AWT implementation is JNI (to talk to your native window system). PJA provides pura java implementations of all of the capabilities a normal AWT toolkit would attempt to perform. Obviously it doesn't actually use the window server, rather it's working akin to a VNC server with a virtual frame buffer.

- Run as root. This is not generally recommended, but you can modify your /System/Library/StartupItems/WebObjects/WebObjects script to run WO as root instead of appserver user. You'll find the instructions on doing so inside of that script.

Java Monitor Issues

For those who have problem with the monitor on Mac OS X please check the following parameters:

- Check that the machine is properly identified in the DNS including the reverse lookup.
- Verify that when you added the machine in the monitor you included the fully qualifier name i.e. "machine.domain.com" and not only "machine" using the shorter version may appear to work but it does not (this is documented by Apple).
- Check the adaptor configuration (/System/Library/WebObjects/Adaptors/Apache/apache.conf) and in particular the instance discovery method: this needs to be in accordance with the command line launch of wotaskd (see the script in /System/Library/StartupItems/). You may need to modify the script by hand.
- When everything is setup properly reboot. It should work.

WOtaskd Didn't Start

In older versions of OS X server, WOtaskd and JavaMonitor were launched using the StartupItems system. In more recent versions of OS X server and WebObjects, Apple uses its new launchd framework instead. If you install WebObjects with developer tools, you typically only end up with the Startupitem scripts, so you should follow those instructions.

WOtaskd/WOMonitor LaunchDaemon

For an overview of launchd, you can read Apple's documentation.

The WebObjects launch daemon items are disabled by default, so the first thing you will need to do is enable them:

1. cd /System/Library/LaunchDaemons
2. edit "com.apple.womonitor.plist" and "com.apple.wotaskd.plist"
3. change the Disabled key from "true" to "false"
4. save and exit

Next you will need to load the launch daemon items manually:

1. Run "launchctl" as root
2. "list" and look for com.webobjects.womonitor and com.webobjects.wotaskd
3. If they do exist, you will need to load them manually
    a. "load /System/Library/LaunchDaemons/com.apple.wotaskd.plist"
    b. "load /System/Library/LaunchDaemons/com.apple.womonitor.plist"
4. "list" again and verify they both appear
5. "start com.webobjects.wotaskd"
6. "start com.webobjects.womonitor"
7. Quit launchctl (ctrl-c it)

wotaskd and womonitor should now both be running. You can test monitor by going to http://localhost:56789.

If this does not work, check for file system permissions on /Library/WebObjects/Configuration, which should be appserver:appserverusr or at least they should have write permission to this directory.

Running this following command in a terminal window will tell you exactly why wotaskd is not launching...

Manually Launch wotaskd

```
sudo -u appserver /System/Library/WebObjects/JavaApplications/wotaskd.woa/Contents/Resources/javawoservice.sh \
-appPath /System/Library/WebObjects/JavaApplications/wotaskd.woa/wotaskd
```

WOtaskd/WOMonitor StartupItems

On developer installs and older Mac OS X Server installs, WebObjects is launched using the old StartupItems system. By default, wotaskd and womonitor are disabled. Fortunately, enabling them is easy:

1. cd /System/Library/StartupItems/WebObjects
2. Edit "WebObjects"
3. Search for "-appPath" and you will find four commented out lines in this area of the script.
    - To run WebObjects as root, uncomment the first two commands (you should see a WOTASKD and a WOMONITOR line)
    - To run WebObjects as another user (appserver user by default), uncomment the second set of the lines
4. Save and exit
5. Start the WebObjects service using one of the two methods:
    - sudo systemstarter start "WebObjects Services"
    - ./WebObjects start

After these changes, wotaskd and womonitor will autostart after a reboot.

Note: Do not follow these instructions on a MacOS X 10.4.* Server machine with WO 5.3.*. There, the process is started as LaunchDeamon (described above). If you enable the Startup Item, the system tries to start wotaskd twice which leads to nothing working at all.

Dealing with Deadlocks/Application Hanging

There are several techniques for dealing with deadlocks and application hangs.

If you are using a profiler or debugger, you can generally pause execution and see where exactly your application is dying.

If you are using JDK 1.4, you can send your application a QUIT signal and it will dump all active threads. On OS X, find the pid of your application, then execute a 'kill -QUIT yourAppPID'. The thread stack traces will dump out to the log files. If you do not see any output in your logs, see the *Where's my stderr!?* above for possible reasons.

If you are using JDK 1.5, in addition to being to send the application a QUIT signal, you can also attach to it (as root or the launching user) and get the thread stack dumps using the 'jstack' application by calling 'jstack yourAppPID'.

If you see hangs in WOWorkerThreads waiting to restore a session, this generally means a previous request threw an exception from a location that didn't properly check its session back in. Session access in WebObjects is single-threaded. Common cases that can cause this are throwing exceptions from your awake methods as well as throwing exceptions from DirectAction methods.

In JDK 1.4, you will be able to see the list of object locks that each stack trace is holding. You can find deadlocks by looking for multiple threads that contain two or more overlapping locks. In JDK 1.5, the VM detects deadlock situations and will identify them in the thread stack dumps.