1

Setting Up and Running Geronimo

Installing Geronimo

In July, 2004, we released Geronimo 1.0 milestone build M2, which is available at http://www.apache.org/dist/geronimo/. By the time this book is finished, a few more milestone builds will have been cut.

How do I do that?

Download the latest greatest version to your system, then read on—we can chat while that is downloading.

At the same time as we're working on Geronimo, we're writing this book in a way that takes into account planned development and features to be added that aren't ready at current time. In short, unless there is a second edition of this book on the shelves, rest assured that the latest, greatest, Geronimo release will either be compatible with the steps described in this book or will come with a complete list of exactly what has changed.

Okay, that binary should be downloaded by now. Assuming you have downloaded a tar.gz file, and we have not yet released the 1.0 final, there should be a file like this on your system:

```
geronimo-1.0-M42-bin.tar.gz
```

Alright, so 42 is a little extreme considering M2 was just released*, but it's no worse than those movies with flying cars that take place only 20 years in the future. Windows users can use WinZip to open that file and go through the GUI to extract it to the file system. Everyone else, including Cygwin users, can run the following command to extract the archive.

* It's also a poor excuse to make a reference to the geek-bible, The Hitchhiker's Guide to the Galaxy.

```
$ tar xzvf geronimo-1.0-M42-bin.tar.gz
```

That command will run for a bit and tons of text will scroll by. When that is all done, there will be a directory named *geronimo-1.0-M42* next to the tar.gz file.

Example 1-1. Contents of a Geronimo install

```
$ ls geronimo-1.0-M42
BUILDING.txt
LICENSE.txt
NOTICE.txt
PROPOSAL.txt
README.txt
STATUS
bin/
config-store/
lib/
repository/
schema/
var/
```

Now take out your number 2 pencil and write down the complete path to that directory, including the 'geronimo-1.0-M42' part, and label it "Geronimo Home." We refer to this directory several times throughout this book; you need to know where it is in order for any of the labs to work.

Also, put down the book for a moment and read the README.txt file. That file contains critical information about the release and any notes on things that have changed since this book was published. It's your friend.

What just happened?

Assuming you've completed any extra steps in the README.txt file, you should be done installing Geronimo. Give yourself a pat on the back. Feel free to poke about the RELEASE notes and generally snoop around before continuing on to the rest of the book.

What About …

On my Linux machines, I typically like to install all Java applications into /usr/local and create a symlink that doesn't contain any version numbers in it.

```
ln –s /usr/local/geronimo-1.0-M42 /usr/local/geronimo
```

Then every time I upgrade the software, I only need to change the symlink and everyone else is unaffected.

Installing Maven

To build Geronimo from source we need a tool called Maven.

Maven is a pretty exciting piece of software as it is more focused on the project level rather than an individual Task or Target level as Ant is. Built on Ant, it provides similar features, but is a real time saver when it comes time to adding new modules or

dependencies to your software. No more stuffing jars into your source code repository, just declare what libraries you need and Maven will download them before compiling.

We'll need to download and unpack Maven, then we will set the MAVEN_HOME environment variable, as well as add Maven's bin directory to the system's PATH. This will allow you to build all the Geronimo source code into a runnable program.

Why do I care?

Aside from using Maven to build the Geronimo source code, this book makes extensive use of Maven for creating Servlet and EJB projects, deploying, and even running Geronimo itself. If you were thinking to skip this section, you should definitely come back to this step before starting the next chapter.

How do I do that?

Download the latest Maven release from their website at *http://maven.apache.org*. There should be a link named "Download" on the left side of the page that will bring you to a list of binaries for the latest release.

> The current release is maven-1.0; however, this chapter was developed with 1.0-rc2, a slightly older version. You should be safe with whatever the latest, greatest, 1.x release is.

Download the zip file binary onto your machine. If you are in Windows, you can easily use WinZip to extract the file. Otherwise the jar command will work quite well. In a command shell, move to the directory where you downloaded the Maven zip and type:

```
C:\> jar xvf maven-1.0-rc2.zip
```

After that finishes unzipping, there should be a new maven directory with the following contents:

```
C:\> dir maven-1.0-rc2
 Volume in drive C has no label.
 Volume Serial Number is 54E7-6152

 Directory of C:\maven-1.0-rc2

03/22/2004  10:53 PM    <DIR>          .
03/22/2004  10:53 PM    <DIR>          ..
05/17/2004  08:00 PM    <DIR>          bin
05/17/2004  08:00 PM    <DIR>          lib
03/22/2004  10:53 PM             2,491 maven-navigation-1.0.xsd
03/22/2004  10:53 PM            14,760 maven-project.xsd
05/17/2004  08:00 PM    <DIR>          plugins
               2 File(s)         17,251 bytes
               5 Dir(s)  24,789,557,248 bytes free
```

We will need to set this directory as an environment variable called MAVEN_HOME as well as add the bin directory to the PATH. If you are a Windows user and you unzipped Maven directly under your 'C:\' drive, the following commands should do the trick:

```
C:\> set MAVEN_HOME=C:\maven-1.0-rc2
C:\> set PATH=%PATH%;%MAVEN_HOME%\bin
```

If you are on a UNIX, Linux, or Mac OS X machine with Bash installed, and you unzipped Maven into your home directory, you should be able to use the following command:

```
~$ export MAVEN_HOME=~/maven-1.0-rc2
~$ export PATH=$PATH:$MAVEN_HOME/bin
```

To give maven a test run to see if everything is setup correctly, we can simply run the maven command with -v to have maven spit out it's version information. From the same command prompt, run this command

```
C:\> maven -v
 __  __
|  \/  |__ _Apache__ ___
| |\/| / _` \ V / -_) ' \  ~ intelligent projects ~
|_|  |_\__,_|\_/\___|_||_|  v. 1.0-rc2
```

What just happened?

To summarize, we unzipped the Maven zip to a directory. Then we setup a command shell to include the `maven-1.0-rc2/bin` directory in your PATH variable so we can execute the maven command from anywhere.

A note of warning, though, as soon as you close that shell, your path settings will be gone and you will have to repeat the last step. In a Bash environment you can add the 'export' statements to the end your .bashrc file, which can be found in your home directory. In a Windows environment you can do this by modifying your system properties, which can be found by going to the Control Panel and clicking the icon named "System."

Why didn't it work?

If you received an error like the following when executing the maven command from a command prompt, you didn't setup your PATH variable correctly.

```
'maven' is not recognized as an internal or external command,
operable program or batch file.
```

If you received an error like the following, you will also need to set the JAVA_HOME environment variable.

```
ERROR: JAVA_HOME not found in your environment.
Please set the JAVA_HOME variable in your environment to match the
location of your Java installation
```

Most Windows users will have the JDK installed at c:\jsdk1.4.2_01, adjust the command bellow if yours is elsewhere.

```
C:\> set JAVA_HOME=c:\jsdk1.4.2_01
```

Note we didn't add \bin to the JAVA_HOME as in c:\jsdk1.4.2_01\bin. It is a common mistake to do this, so make sure to avoid it.

> Geronimo only supports JDK 1.4 or higher, so if you have multiple versions on your machine make sure to set the JAVA_HOME variable to the right one.

What About …

On Linux machines, I probably wouldn't install maven into my home directory. I'd likely throw it into /usr/local and do the same symlink trick we did while installing geronimo.

```
~$ ln –s /usr/local/maven-1.0-rc2 /usr/local/maven
```

Then you can just add /usr/local/maven/bin to your PATH variable and not have to worry about changing it every time you upgrade maven to the next version.

Downloading the Source with CVS

Like most open source projects, Geronimo uses CVS as its code repository. CVS stands for Concurrent Version System in case you were wondering. If you were, then you should take a moment and visit www.cvshome.org to get yourself familiar with the tool. CVS is older that the hills in terms of open source software, so we are going to assume basic CVS knowledge in this section.

Why do I care?

Using Geronimo does not require having the source code. However, many people are drawn to Gernomimo specifically because it can be built from source, modified, and patched.

I'll also note, as a Geronimo developer, there is nothing we like better than a patch along with a bug report. Good patches have been known to get you a free beer or two and some time with the developers at any of the conferences we speak at. Who knows, you may even wind up with commit privileges.

How do I do that?

If you have CVS already installed and are ready to go, the following two commands are all you need to get the source. The first command logs you into the system. When it prompts you for the CVS password, simply type "anoncvs" then hit the Enter key.

```
~$ cvs –d :pserver:anoncvs@cvs.apache.org:/home/cvspublic login
~$ cvs –d :pserver:anoncvs@cvs.apache.org:/home/cvspublic co incubator-geronimo
```

When that second command runs, you will see several lines of output like that shown in Example 1-2.

Example 1-2. Output from CVS checkout

```
cvs server: Updating incubator-geronimo
U incubator-geronimo/.cvsignore
U incubator-geronimo/LICENSE.txt
U incubator-geronimo/PROPOSAL.txt
U incubator-geronimo/README.txt
U incubator-geronimo/STATUS
U incubator-geronimo/maven.xml
U incubator-geronimo/project.properties
U incubator-geronimo/project.xml
```

Feel free to go get a cup of coffee, this command will take awhile. At the time of this writing, the repository contains nearly 2000 files totaling somewhere over 8 megabytes of data. More than three fourths of those are java files.

What just happened?

When that finishes, you should have a new directory called "incubator-geronimo", in it will be all the source and related files from the Geronimo CVS. These will be used to build a binary, runnable, version of Geronimo.

What About…

…other CVS clients like WinCVS or Eclipse's CVS support? Those will work, though they generally take more work to setup. Just make sure you specify the information as shown above.

Building Geronimo with Maven

Once you have the source from CVS and Maven is setup correctly, building the code is easy. It will, however, take a bit longer than you might expect. This is for two reasons:

- maven must download all the dependent libraries
- all the unit tests will run as part of the build.

Downloading the dependencies is unavoidable, but fortunately only has to happen the first time you build. Once Maven has downloaded the required jars, they will be placed in a local repository in your home directory and will not need to be downloaded again.

You can avoid running the unit tests; we will see how to do that.

Why do I care?

As mentioned earlier, Open Source software naturally involves the source. You don't need the source, however, and are just fine sticking with the latest Geronimo binary release shipped from Apache.

How do I do that?

Normally, to build something with Maven you simply type 'maven' at the command prompt. However, since we aren't interested in running the tests as well – you can generally assume things are OK if you haven't changed the code at all – we will add an extra option to the command line to shut the tests off.

You simply need to move into the 'incubator-geronimo' directory and type the following command:

```
C:\incubator-geronimo> maven –Dmaven.test.skip=true
```

The output of that command should start out looking very similar to that shown in Example 1-3.

Example 1-3. Output from Geronimo build

```
 __  __
|  \/  |__ _Apache__ ___
| |\/| / _` \ V / -_) ' \  ~ intelligent projects ~
|_|  |_\__,_|\_/\___|_||_|  v. 1.0-rc1-SNAPSHOT

Starting the reactor...
Our processing order:
Geronimo :: Java Transaction API Specification
Geronimo :: Enterprise JavaBeans Specification
Geronimo :: J2EE Application Management Specification
Geronimo :: J2EE Application Deployment Specification
Geronimo :: Kernel
Geronimo :: Common
Geronimo :: Servlet Specification
Geronimo :: Java Server Pages Specification
Geronimo :: Web Console
Geronimo :: Twiddle
Geronimo :: Java Authorization Contract for Containers Specification
Geronimo :: J2EE Connector Architecure Specification
Geronimo :: Core
Geronimo :: Remoting
Geronimo :: Explorer
Geronimo :: Java Activation Framework Specification
Geronimo :: JavaMail Specification
Geronimo :: JavaMail Implementation
Geronimo :: Web
Geronimo :: XBeans
Geronimo :: JMS Specification
```

After this you should see screen-fulls of output; don't be worried, this is completely normal. Each of the lines starting with "Geronimo" represents a module that has to be built. This will take a while, so feel free to go make a whole pot of coffee.

When all is said and done, we should see a "BUILD SUCCESSFUL" message at the end of all the output.

```
BUILD SUCCESSFUL
Total time: 8 minutes 40 seconds
Finished at: Wed Dec 04 14:30:13 CST 2003
```

What just happened?

In short, Maven built each module one-by-one, stopping on occasion to download any required libraries from various places on the internet.

Where are the Configuration Files

This is really a tricky question; there are no configuration files to speak of. Geronimo is a rare bird in that a running instance of the server it is not bound to any text for or XML files. Everything in Geronimo from the EJB Container to the transaction manager is deployed into a bootstrapped kernel, much like Enterprise JavaBeans are deployed into the EJB Container itself. In fact, once both are deployed, they both live at the same level in the eyes of the kernel and are equally as manageable.

Each component deployed into Geronimo also has a deployment descriptor of sorts. It's in this xml file that the initial values for a component are set. Once deployed, however, all data in the xml file as well as the jar file itself is sucked into a new file directly in to Geronimo's non-editable repository called the config-store. Everything in the config-store is non-editable, read-only, and should never be touched directly even by administrators. All changes should come by deploying a new version of the component, or via the console tools, which will essentially create a new version of the component. The idea at work here is to make it impossible to "configure" your server into a totally unusable state. If a new version of a component causes problems or instability of any kind, you can always roll back to the Last Known Good Configuration that will be safely sitting in the config-store.

Geronimo is also young, so I don't want to go into too much detail on the implementation details, but the basic idea will always remain core to Geronimo. The versioning of configurations will stay, though the mechanisms and features around dealing with their storage will certainly change.

Starting Geronimo

Starting Geronimo is actually very simple. Most Java applications require some sort of FOO_HOME system variable to be set before the application can be successfully run. Geronimo does have a geronimo.home system property that can be set, but it is really not needed. Additionally, most Java applications have a lot of classpath hocus-pocus to deal with before they can be started.

Geronimo uses executable jars for starting the server. This alleviates the need to muck with classpaths as we can the MANIFEST.MF file to setup the classpath when we create the jar. It also allows us to be a little sneaky and determine exactly where the geronimo.home is relative to where the jar is sitting on your filesystem.

How do I do that?

Open a command prompt, bash shell, terminal, or whatever your OS calls the place where you type in commands to execute. Assuming you installed Geronimo in the 'C:\geronimo-1.0-M42' directory as described in the 'Installing Geronimo' section, you can start Geronimo with the following command.

```
java –jar C:\geronimo-1.0-M42\bin\server.jar
```

When that command runs, it should spit out a bunch of text like the following:

```
03:59:53,814 INFO  [Kernel] Starting boot
03:59:53,894 INFO  [MBeanServerFactory] Created MBeanServer with ID:
1dfc547:fc5eb0a69c:-8000:Pepe:1
03:59:54,065 INFO  [Kernel] Booted
03:59:54,095 INFO  [ConfigurationManager] Loaded Configuration
geronimo.config:name="org/apache/geronimo/System"
03:59:54,255 INFO  [Configuration] Started configuration org/apache/geronimo/System
03:59:54,275 INFO  [ReadOnlyRepository] Repository root is file:/C:/geronimo-1.0-
M42/repository/
```

The server won't exit; it sits there waiting for some action. As I mentioned, Geronimo knows how to find the directory where it's installed, so you can execute that command from anywhere and don't need to be right there in the 'C:\geronimo-1.0-M42' directory.

What just happened?

Not much actually. We really only started the Geronimo kernel with nothing in it. You have to specify the items in the config-store you wish to be started. We can use a JMX Debug application that ships with Geronimo as an example. To start that application, just add the URI 'org/apache/geronimo/DebugConsole' to the start command as such (let's assume you are executing from the 'C:\geronimo-1.0-M42' directory) .

```
java –jar bin\server.jar org/apache/geronimo/DebugConsole

04:15:30,772 INFO  [Kernel] Starting boot
04:15:30,852 INFO  [MBeanServerFactory] Created MBeanServer with ID:
1dfc547:fc5ebef290:-8000:Pepe:1
04:15:31,022 INFO  [Kernel] Booted
04:15:31,042 INFO  [ConfigurationManager] Loaded Configuration
geronimo.config:name="org/apache/geronimo/System"
04:15:31,202 INFO  [Configuration] Started configuration org/apache/geronimo/System
04:15:31,212 INFO  [ReadOnlyRepository] Repository root is file:/C:/geronimo-1.0-
M42/repository/
04:15:31,292 INFO  [ConfigurationManager] Loaded Configuration
geronimo.config:name="org/apache/geronimo/DebugConsole"
04:15:31,312 INFO  [ConfigurationManager] Loaded Configuration
geronimo.config:name="org/apache/geronimo/Server"
04:15:31,673 INFO  [LogSupport] Log instance is class
org.apache.geronimo.kernel.log.GeronimoLog
04:15:31,883 INFO  [Configuration] Started configuration org/apache/geronimo/Server
04:15:31,883 INFO  [HttpServer] Starting Jetty/5.0.beta0
04:15:31,893 INFO  [HttpServer] Started org.mortbay.jetty.Server@137d090
04:15:31,903 INFO  [SocketListener] Started SocketListener on 0.0.0.0:8080
04:15:32,093 INFO  [Configuration] Started configuration
org/apache/geronimo/DebugConsole
04:15:32,364 INFO  [Credential] Checking Resource aliases
04:15:33,726 INFO  [HttpContext] Started WebApplicationContext[/debug-
tool,file:/C:/geronimo-1.0-M42/config-store/22/war/]
04:15:34,016 INFO  [TcpTransportServerChannel] Listening for connections at:
tcp://localhost:61616
04:15:34,016 INFO  [ActiveMQBrokerImpl] ActiveMQ JMS Broker started
```

With that command you should get a lot more output. The last line in this example output may not be the same one you see. Geronimo will deploy everything that relates to the component that is started; it stops writing to the screen when all the work is done.

Why didn't it work?

If you saw minimal or no output at all, good. Console output is extremely slow and we should only be doing very minimal amounts of it. By the time you read this book, the output shouldn't be so copious. Check the log files for more verbose output. (See "Where are the log files" later in this chapter.)

If you received a NoSuchConfigException or something of the sort, you probably mistyped the URI or switched the forward slashes (/) to backslashes. The URI used to

start the application is just a unique identifier that keys each entry in the config-store; it is not a file path. There is no need to change the slashes to match the path syntax of your operating system.

What About …

Jane, stop this crazy thing!

This one has to be done the old fashioned way. Hit Control^C in the window where you started Geronimo to kill the server. We haven't yet provided an more elegant means for stopping the server. If you are running Cygwin on WindowsXP, then you know that Control^C frequently doesn't kill anything, it just pushes the current process into the background. To kill the server dead in this situation, hit Control^ALT^Delete, click the "Processes" tab, highlight the java.exe line and click "End Process." That should do the trick.

We are aware how much of a pain this is, so make sure your check the release notes and README.txt files. This is definitely on the TODO list and will likely be fixed by the time this book reaches your hands.

Taking a Test Drive

OpenEJB, the EJB Container for Geronimo, has a pretty slick integration test suite that it's been using over the years. Geronimo uses the test suite religiously to ensure the integration is working A-OK. The test suite is basically a plain Java client and about 19 large Enterprise JavaBeans. Ok, so maybe the client isn't so plain, it's a spicy JUnit test suite, but does interact with the server in a completely standards-compliant way.

The test suite is a great way to give things a whirl and see if things are more or less working on your machine. Though it tests just EJB components, it does tax the Geronimo kernel and facilities quite heavily. If this thing runs on your setup, you're in good shape.

How do I do that?

Open up your browser, steer over to *http://openejb.org/example-app/* and download the two openejb-itests jars, the openejb-jca.rar, and build.xml file.

Once you get everything downloaded open the build.xml file, shown in Example 1-4, in your editor of choice and update the value of the "geronimo.home" property to the actual geronimo.home for your system.[†]

Example 1-4. Test suite build.xml

```
<?xml version="1.0"?>
<project name="OpenEJB itests" default="test" basedir=".">
  <property name="geronimo.home" value="C:/geronimo-1.0-M42"/>
```

[†] If you don't know what a geronimo.home is, get your number 2 pencil ready then jump back to the "Installing Geronimo" section and give it a quick read.

```
  <path id="test.class.path">
    <pathelement path="openejb-itests-2.0M5.jar"/>
    <pathelement path="${geronimo.home}/repository/openejb/jars/openejb-core-
2.0M5.jar"/>
    <pathelement path="${geronimo.home}/repository/geronimo-spec/jars/geronimo-
spec-j2ee-1.4-RC1.jar"/>
    <pathelement path="${geronimo.home}/lib/cglib-full-2.0.jar"/>
    <pathelement path="${geronimo.home}/repository/junit/jars/junit-3.8.jar"/>
  </path>

  <target name="install" description="Install the integration test suite into
Geronimo">
    <echo>Installing the Axion Datasource</echo>

    <java jar="${geronimo.home}/bin/deployer.jar" fork="true" failonerror="true">
      <arg value="--install"/>
      <arg value="--module"/>
      <arg value="openejb-jca-2.0M5.rar"/>
    </java>
    <echo>Installing the OpenEJB Integration Test Suite</echo>

    <java jar="${geronimo.home}/bin/deployer.jar" fork="true" failonerror="true">
      <arg value="--install"/>
      <arg value="--module"/>
      <arg value="openejb-itests-beans2.jar"/>
    </java>
  </target>

  <target name="test" description="Runs the integration test suite">

    <java classname="org.openejb.test.TestRunner" fork="true" failonerror="true">
      <classpath refid="test.class.path"/>
      <jvmarg value="-Dopenejb.test.database=org.openejb.test.AxionTestDatabase"/>
      <jvmarg value="-Dopenejb.test.server=org.openejb.test.RemoteTestServer"/>
      <arg value="org.openejb.test.ClientTestSuite"/>
    </java>
  </target>
</project>
```

The OpenEJB version numbers are hardcoded in the script and will change over time. Take a quick peek in the repository/openejb/jars/ directory of your Geronimo installation and see what version of OpenEJB came with your distribution; update the version number in the script if needed.

Let's also make sure you have Ant installed. To see if you have Ant set up well enough to run this example, just type this on a command line:

```
C:\> ant -version
```

If you don't see something like "Apache Ant version 1.6.0 compiled on December 18 2003" printed out, then Ant isn't installed correctly on your system and you should get that set up before continuing.

If ant is setup, then run this command in the same directory where you downloaded the build.xml and openejb jar files.

```
C:\example-app> ant install
```

You should see the following output:

```
Buildfile: build.xml

install:
     [echo] Installing the Axion Datasource
     [echo] Installing the OpenEJB Integration Test Suite

BUILD SUCCESSFUL
Total time: 22 seconds
```

If everything went OK, then congratulations are in order. You just deployed your first set of EJBs into Geronimo! If you got any sort of failure, the geronimo.home property was probably not set correctly. Double check the path you used in the build.xml.

Next, start the server so it loads the new ejb application.

```
C:\geronimo-1.0-M42> java -jar bin\server.jar org/openejb/itests
```

Now for the really exciting part; running our first EJB client application! Pop open another console window as other previous one is now busy and execute the ant task to run the client against our EJBs.

```
C:\example-app> ant test
```

What just happened?

The test suite should be gloriously piling dots down your screen as in Example 1-5. The EJB client app, which as also a JUnit app, is connecting to the Geronimo server you started and is making all sorts of EJB calls into it.

Example 1-5. Test suite output

```
Buildfile: build.xml

test:
     [java] ....................................
     [java] ....................................
     [java] ....................................
     [java] ....................................
     [java] ....................................
     [java] ....................................
     [java] ....................................
     [java] ....................................
     [java] ....................................
     [java] ..................................
     [java] Time: 12.55

     [java] OK (405 tests)


BUILD SUCCESSFUL
Total time: 34 seconds
```

Oh the beautiful dots of JUnit test cases—passing test cases to be exact. Not much to look at for most people, but to a computer geek, it's the sign of a job well done. So, well done!

Why didn't it work?

If some of the tests fail on your computer there is nothing to worry about. That's just a natural part of testing, not all the tests always pass.

If none of the tests pass and you just get errors spilling out rather than beautiful dot goodness, then something is wrong in your setup. The Geronimo and OpenEJB version numbers may not match up with the tests downloaded or the ones hardcoded in our script. Double check those and ask for help on the Geronimo mailing list if you can't seem to get it working.

What About…

If you thought that was easy, you have my respect--we have entirely different opinions of what easy is. Ant is a good tool, but really not that great when it comes towards managing library versions, paths, and jar names. In order to "get it right" you have to create a ton of properties so things aren't quite so hardcoded.

Maven is the tool of choice for this book. It's much easier to use when creating projects from scratch, managing library dependencies, and deploying things into Geronimo. The rest of the examples in this book all use Maven. Trust me, it is a lot simpler.

Where are the Log Files

We have the log files safely tucked away in the 'var/log/' directory of the distribution.

```
~$ ls geronimo-1.0-M42/var/log
deployer.log
geronimo.log
```

The geronimo.log file contains output from any running Geronimo kernels on the system, and the deployer.log file contains messages from running the deploy tools which we see in the next section.

Why do I care?

Ideally, you never encounter any problems and never need to look at the log files for the source of any failures. Obviously, that is dreaming, so it's good to be aware of the log files and poke at them on occasion.

What can I do?

Again, those running on some sort of Linux machine will probably want to perform some symlink magic to get the log files under the real /var directory on your system, rather than geronimo-1.0-M42/var. If you created the symlink described in the 'Installing Geronimo' section, then you can create a maintenance link with the following command.

```
~$ ln –s /usr/local/geronimo/var/log /var/log/geronimo
```

What About …

If you have a special partition for /var and want the log files to be stored on that partition rather than linked to somewhere else, you can to a reverse of what we just did.

```
~$ mkdir /var/log/geronimo
~$ rm -r /usr/local/geronimo/var/log
~$ ln –s /var/log/geronimo /usr/local/geronimo/var/log
```

That would result in Geronimo writing to the /var/log/geronimo directory but still seeing it as /usr/local/geronimo/var/log. This is very nice if you are running a special RAID level for the /var partition and want the log files to live there.