Building Apache Geronimo

Overview

This documents how to build the Apache Geronimo Server project.

This guide is intended to cover how to build the latest `server/trunk`, though other newer branches should also follow similar instructions. Server trees that use the same basic build tooling include:

- `server/trunk`
- `server/branches/3.0`
- `server/branches/2.2`
- `server/branches/2.1`
- `server/branches/2.0`
- `server/branches/1.2` (not tested recently)

nightly built snapshots from trunk
If you are only interested in obtaining a compiled binary of what is most recent in trunk, this is already made available. Geronimo `server/trunk` is built from the repository on a nightly basis, and the resulting compiled binaries are available as snapshots on Apache's Snapshot Repository. Particularly you will want the Apache Geronimo assemblies which is what is published when Geronimo is released as a major revision.

Typically one would compile Geronimo when testing code modifications, otherwise the snapshots the Geronimo community provides is sufficient for testing the latest code from the repository. And when submitting a bug, the community appreciates if these latest Geronimo assemblies can be tested for the bug to make sure we haven't already fixed it.

other relevant documentation
The Geronimo community also maintains build documentation specific to each Geronimo version. While the community attempts to keep all these related docs in sync with each other, you may find some varying information among them.

- Geronimo 3.0: Building Geronimo with Maven
- Geronimo 2.2: Building Geronimo with Maven

a general warning about the state of trunk
The source code in `server/trunk` typically will be in SNAPSHOT release, as opposed to a final release. E.g. 3.0-SNAPSHOT versus 3.0-beta. The community typically finalizes the version from SNAPSHOT release to a regular release when it is branched out of trunk. E.g. branch `server/trunk` to `server/branches/3.0-beta`.

Please be aware that when building Geronimo from trunk while in SNAPSHOT release the source code typically depends on the SNAPSHOT releases from all dependency projects. There are two risks from this matter-of-fact:

1. Other projects release SNAPSHOTS on a daily basis. Every day you would build Geronimo, the build process will download and build against newly released artifacts.
2. The nature of SNAPSHOT releases is that they are the latest development builds. Though they will be compiled and released, they are not necessarily error free. Additionally, SNAPSHOTS are actively developed and may introduce some new feature or API that might cause Geronimo to fail during its own build.

- Due to the nature of item #1, item #2 may occur on any future day, and item #2 may persist for an unknown amount of time until issues are corrected.
  The Apache Geronimo community works with dependency projects to resolve issues that prevent Geronimo from building or passing tests, but this is not always resolvable in a speedy fashion. Sometimes the Geronimo community must make dependency changes to resolve issues with other project's artifacts that persist with issues that go unresolved.

To help mask issues like these that can surface, you can setup a local maven proxy repo (like with nexus) and control what artifacts are downloaded and used in the build process. The local proxy repository performs the download of 3rd party artifacts Geronimo depends on, and the Geronimo build process obtains the artifacts from the local proxy repository. In doing this, artifacts that causes issues can be rolled back to a previous version, or instead you can pause updates on certain problem artifacts by controlling which are updated from the remote repositories for the Geronimo build process to use.

Prerequisites

Build Machine

It is recommended that a dedicated computer used to build `server/trunk` have a minimum of 2GB of real memory. The computer will also need a connection to the Internet to download artifact dependencies.

Build Time
The first-time-build will download all artifact dependencies into your local maven repository, so the build time will partly depend on the speed of your internet connection. The builds that are executed after the first complete build will be faster because the artifact dependencies are already downloaded during the first build. As Geronimo trunk is being developed as SNAPSHOT (e.g. Geronimo 3.0-SNAPSHOT), it also depends on 3rd-party artifacts that are being released as SNAPSHOTs. As a result, new artifacts will be downloaded as snapshots are released, which is a daily occurrence.

As of June 2011 the following build time results were recorded:

- 90 minutes - one 2.4GHz dual-core CPU; 2GB of real memory; 2Mb Internet connection; empty local maven repository (first build)
- 13+ hours - two 2.0GHz dual-core CPUs; 1GB of real memory; 2Mb Internet connection; populated maven repository; (not first build)

The available real memory makes a big difference. During the build process, maven self-loads the artifacts it builds. Having enough real memory allows the build process to quickly start up the dependencies that are built. Without enough memory, your real memory must be swapped so the dependencies can be started, and then swapped again to go back to the build process.

Build Tips

Refer to the **Build Options** section below for more options that can speed up your build time.

Java Developer Kit (JDK)

You will need a JDK 6.0+ (Java SE 1.6.0+ SDK) or compatible JDK to build Apache Geronimo from trunk. It is recommended you use SUN's implementation, or something compatible like Apples implementation. Other JDK vendors implementations may work, but use at your own risk.

Apache Geronimo 2.2
JDK 5.0+ (J2SE 1.5.0+) or compatible JDK.
The Java SE (JDK) 6.0 works with Geronimo 2.2.

Apache Geronimo 2.1 and earlier
JDK 5.0+ (J2SE 1.5.0+) or compatible JDK.

Windows Tip

Windows users should not double-quote JAVA_HOME (or MAVEN_HOME for the same reason), according to MAVEN-666.
Setting:
set JAVA_HOME="C:\Program Files\Java\jdk1.6.0_20"
doesn't work, but this does:
set JAVA_HOME=C:\Program Files\Java\jdk1.6.0_20

Apache Maven 2

To execute the build process you need to have Apache Maven version `2.2.1` (or newer) installed to build Apache Geronimo from trunk. Maven 3.0 can also be used to build trunk.

To check if your installation is working and you have the required minimum version run:

```
mvn -version
```

And it should produce something like:

```
Maven version: 2.2.1
```

If you have an incompatible version the server build will probably fail with a message complaining 😉

Apache Geronimo 2.2
Apache Maven version `2.0.10` (or newer)

maven repository

When building Geronimo 2.2. Add the following to your `settings.xml` for maven so that you can avoid the redirect (and hence avoid the bogus poms/jars) and get beyond compilation failure problem to build Geronimo using maven. See this message for more details.

excerpt of setting.xml for maven

```
...
   <mirrors>
       <mirror>
           <id>java.net</id>
           <name>Mirror of https://maven-repository.dev.java.net/nonav/repository/</name>
           <url>http://download.java.net/maven/2/</url>
           <mirrorOf>java.net</mirrorOf>
       </mirror>
   </mirrors>
...
```

Subversion

To fetch the source code for the server, you will need to have a Subversion client version 1.5 (or newer, 1.5 is recommended) installed.

Windows Tip

Windows users are **strongly encouraged** to change the M2 local repository (the place where dependencies are downloaded) to a shorter path with no spaces, e.g. `C:\.m2`.
Using a longer path may cause the build (and Geronimo itself) to behave very strangely when it hits the 260 char limit for filenames on Windows.

In order to change the m2 local repository go to `%USERPROFILE%\.m2` and edit or create `settings.xml` file to contain the following content:

```
<?xml version="1.0"?>
<settings>
    <localRepository>C:\.m2</localRepository>
</settings>
```

Checkout Geronimo

```
svn co https://svn.apache.org/repos/asf/geronimo/server/trunk server
```

Tip

If you are using Chinese system, please change your locale to en_US. Otherwise, the maven 2.0.7(or below) can't parse Chinese characters.

Windows Tip

Windows users are **strongly encouraged** to checkout Geronimo into `c:\g`.
Using a longer path may cause the build (and Geronimo itself) to behave very strangely when it hits the 260 char limit for filenames on Windows.

You can also use Git mirrors to checkout Geronimo source code, and make sure you have a Git client installed before using the following command:

```
git clone git://git.apache.org/geronimo server
```

Preparing to Build for the First Time

Chances are you will need to increase the heap size for Maven. Add the following lines to `~/.mavenrc`:

The following snips only set `MAVEN_OPTS` if its not already set, so that you can override these values on the command line if needed.

```
# Increase the heap size Maven
if [ "x$MAVEN_OPTS" = "x" ]; then
    MAVEN_OPTS=-Xmx512m
fi
```

If you are using the SUN JDK (or a JDK with compatible flags, like the Apple JDK), you should also increase the maximum permanent size as well as the heap:

```
# Increase the heap and max permanent size for Maven
if [ "x$MAVEN_OPTS" = "x" ]; then
    MAVEN_OPTS="-Xmx1024m -XX:PermSize=256m -XX:MaxPermSize=1024m -XX:ReservedCodeCacheSize=64m"
fi
```

For Apache Geronimo 2.2, use at least:

```
if [ "x$MAVEN_OPTS" = "x" ]; then
    MAVEN_OPTS="-Xmx512m -XX:MaxPermSize=128m -XX:ReservedCodeCacheSize=64m"
fi
```

Windows Tip

Windows users should create `mavenrc_pre.bat` under `c:\documents and settings\<username>\mavenrc_pre.bat` or `c:\mavenrc_pre.bat` depending on how the `%HOME%` property is set on your system.

Variables will need to use the batch set syntax:

```
set <VARIABLE>=<VALUE>
```

Building

To build all changes incrementally:

```
mvn install
```

To perform clean builds, which are sometimes needed after some changes to the source tree:

```
mvn clean install
```

Building Stages

In some cases you may need to build Geronimo in stages, such as when you have changed the geronimo version. Most users will not need to do this, but its documented here for clarity.

To build modules, testsupport and maven-plugins:

```
mvn install -Dstage=bootstrap
```

To build apps, configs and assemblies:

```
mvn install -Dstage=assemble
```

As mentioned, most users will not need to build Geronimo in stages. But in some cases, when bootstrapping new versions (when no artifacts are deployed into remote repositories for the current version), then you **must** build the stages separately for the *first* build and then for all builds afterwards, the staged build is not necessary.

Using Sonatype nexus can help make large maven builds faster and more reliable. However you have to configure all the repositories specified in the geronimo poms in your nexus instance.

Build Options

HeapDumpOnOutOfMemoryError
Add the HeapDumpOnOutOfMemoryError flag to MAVEN_OPTS to get a dump on OutOfMemory errors

```
-XX:+HeapDumpOnOutOfMemoryError
```

Skip tests
Use -DskipTests=true to turn off tests during server build. You can use this flag if you are simply running a build to save a little more time. However, you should not use this flag if you are making code changes, as you should want to build with the tests to make sure your changes do not break anything. Testing the build is necessary if you are submitting a patch or code change recommendations.

```
mvn clean install -DskipTests=true
```

Build offline
You can also build in an offline mode. Use -o to avoid searching remote repositories to save more time. However, you have to complete one full online build before this will work so that all dependencies will be downloaded into your local maven repository. After the first complete build, using the -o flag will cause maven to only use what you have locally and not search remote repositories.

```
mvn -o clean install
```

More error and debug information
To get Maven to provide you with more error details, use -e to get stack traces on errors, and use -X to produce execution debug output.

```
mvn -X clean install
mvn -e clean install
```

Restricting a build to only one assembly
If your only interested in either the tomcat or jetty assemblies, you can save build time by indicating which of these assembly types should be built. There are two profiles in the `geronimo/server/trunk/assemblies/pom.xml` named `tomcat` and `jetty`. To only build the tomcat assemblies and not the jetty ones, you would indicate it like this:

```
mvn clean install -Ptomcat
```

You should be able to do this with built-in maven stuff, see maven tips and tricks advanced reactor options.
Example: This should build everything needed for the named assembly including the assembly:

```
mvn clean install -pl :geronimo-tomcat7-javaee6 -am
```

Geronimo Assemblies Resulting from the Build

After a complete build has successfully finished, the resulting assemblies are located in associated target subdirectories of each assembly. These are the same as what are provided from the nightly Apache Geronimo Snapshots.

```
BUILD_ROOT/assemblies/geronimo-jetty8-javaee6/target/geronimo-jetty8-javaee6-3.0-SNAPSHOT-bin(.tar.gz|.zip)
BUILD_ROOT/assemblies/geronimo-jetty8-javaee6-web/target/geronimo-jetty8-javaee6-web-3.0-SNAPSHOT-bin(.tar.gz|.
zip)
BUILD_ROOT/assemblies/geronimo-tomcat7-minimal/target/geronimo-tomcat7-minimal-3.0-SNAPSHOT-bin(.tar.gz|.zip)
BUILD_ROOT/assemblies/geronimo-tomcat7-javaee6/target/geronimo-tomcat7-javaee6-3.0-SNAPSHOT-bin(.tar.gz|.zip)
BUILD_ROOT/assemblies/geronimo-tomcat7-javaee6-web/target/geronimo-tomcat7-javaee6-web-3.0-SNAPSHOT-bin(.tar.
gz|.zip)
BUILD_ROOT/assemblies/geronimo-jetty8-minimal/target/geronimo-jetty8-minimal-3.0-SNAPSHOT-bin(.tar.gz|.zip)
BUILD_ROOT/plugins/clustering/geronimo-plugin-farm-node/target/geronimo-plugin-farm-node-3.0-SNAPSHOT-bin(.tar.
gz|.zip)
BUILD_ROOT/framework/assemblies/geronimo-framework/target/geronimo-framework-3.0-SNAPSHOT-bin(.tar.gz|.zip)
BUILD_ROOT/framework/modules/geronimo-plugin/target/test-resources/ArchiverGBeanTest/server1/bar-1.0-bin(.tar.
gz|.zip)
```

Testing the Assembly

using the geronimo-maven-plugin

Once you have build the server fully, which will produce a number of `zip` and `tar.gz` archives from the assembly modules, you can use the `geronimo-maven-plugin` to start the server. From the project root directory run:

```
mvn -Ptools geronimo:start
```

Windows Tip

Windows users may need to specify an alternative `installDirectory` to avoid long path problems:

```
mvn -Ptools geronimo:start -DinstallDirectory=c:\g
```

And to stop, either `CTRL-C` or from a separate terminal, from the project root directory run:

```
mvn -Ptools geronimo:stop
```

or by hand

cd assemblies/geronimo/jetty6-javaee5/target
tar xzf geronimo-jetty6-javaee5-<version>-bin.tar.gz
./geronimo-jetty-javaee5-<version>/bin/gsh geronimo/start-server

IDE Setup

The server project does not have any IDE files checked in since major IDEs have better support for maven than maven for them.

Intellij IDEA

For the IDEA 8.1 and later:
The project is too large to easily open at once in IDEA. Generally the framework project and individual plugins are a good size to work with. In the maven projects tab, check the it profile to get more of the maven projects.

Eclipse

Use the maven2 eclipse:eclipse plugin. See Developing Geronimo in Eclipse for detailed instructions. As for IDEA, opening framework or individual plugins is more likely to work than trying to open the whole project.

Troubleshooting

Build was not able to download and install artifacts

If for some reason maven is unable to download and install an artifact, and the build stops with an error complaining of such, you will need to manually download the archive and install it in your local maven repository. You can do that with a maven command similar to this:

```
mvn install:install-file
    -DgroupId=org.apache.geronimo.framework
    -DartifactId=geronimo-kernel
    -Dversion=jar
    -Dpackaging=3.0-20110608.081951-139
    -Dfile=/path/to/file-artifact.jar
```

Maven can download and install dependencies for you:

```
mvn dependency:get
    -DrepoUrl=https://repository.apache.org/content/groups/snapshots/
    -Dartifact="org.apache.geronimo.framework:geronimo-kernel:jar:3.0-20110608.081951-139"
```

You will need to figure out which repository the artifact is available from. Look in the `BUILD_ROOT/pom.xml` file within the `<repositories>` tag to see the primary repositories utilized. You can also review the build output to see where maven attempted to download the artifact from.

Overriding libraries provided by JVM at compile time

In some cases, Java EE 6 requires newer library versions then what is provided by Java SE 6. For example, EE 6 requires annotations 1.1 api but SE 6 provides annotation 1.0 api. So it is necessary that during a build we override the JVM provided libraries with newer versions.

You can use `geronimo-property-plugin` in the pom.xml file of artifacts to set _*bootClassPath*_ property which is used by `maven-compiler-plugin` and `maven-surefire-plugin` as followed. The `geronimo-property-plugin` executes in the **validate** phase and sets _*bootClassPath*_ system property. The value of _*bootClassPath*_ property is set to _***-Xbootclasspath/p:<path><pathSeparator><path>...***_ string where each **<path>** is the jar file to place in the front of boot classpath.

pom.xml

```
...
            <plugin>
                <groupId>org.apache.geronimo.buildsupport</groupId>
                <artifactId>geronimo-property-plugin</artifactId>
                <configuration>
                    <propertyName>bootClassPath</propertyName>
                    <propertyValuePrefix>-Xbootclasspath/p:</propertyValuePrefix>
                    <classpath>
                        <dependency>
                            <groupId>org.apache.geronimo.specs</groupId>
                            <artifactId>geronimo-annotation_1.1_spec</artifactId>
                            <version>1.0.1</version>
                        </dependency>
                        <dependency>
                            <groupId>org.apache.geronimo.specs</groupId>
                            <artifactId>geronimo-jaxws_2.2_spec</artifactId>
                            <version>${geronimojaxws.version}</version>
                        </dependency>
                    </classpath>
                </configuration>
            </plugin>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <configuration>
                    <compilerArgument>${bootClassPath}</compilerArgument>
                </configuration>
            </plugin>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-surefire-plugin</artifactId>
                <configuration>
                    <argLine>${bootClassPath}</argLine>
                </configuration>
            </plugin>
...
```

Meanwhile, you can use *listFiles* options so that the **set** property of `geronimo-property-plugins` could contain files or directories. For example, if you want the `geronimo-property-plugins` to set "-Djava.endorsed.dirs" property, then configure the plugin with:

```
<propertyValuePrefix>-Djava.endorsed.dirs=</propertyValuePrefix>
<listFiles>false</listFiles>
```

If you want the plugin to set "-Xbootclasspath/p:" property configure the plugin with:

```
<propertyValuePrefix>-Xbootclasspath/p:</propertyValuePrefix>
<listFiles>true</listFiles>
```

Build can't find xbean jar

Apache Geronimo 2.0
If you're building the 2.0 branch and the build fails because it can't find an xbean jar (for example org.apache.xbean:xbean-naming:jar:3.2-r579367), then add the

```
http://svn.apache.org/repos/asf/openejb/repo/
```

repository to the `pom.xml` file in the root of the source tree. See this message for more details.