Geronimo Server Release Process

Procedure

1. Whenever possible, use the maven release plugin. If something doesn't work file a bug against it.
2. Use extreme caution in creating branches as opposed to releasing from trunk. While "core" geronimo may need to keep branches, most smaller projects such as specs, plugins, components, and most likely tools should avoid the complexity of branches unless clearly necessary and agreed upon.
3. When branches are needed, branches/x.y would be the branch for all x.y.z releases

The next sections are copied from http://maven.apache.org/developers/release/releasing.html with modifications for Geronimo.

Releasing a Geronimo Project

What follows is a description of releasing a Geronimo project to a staging repository, whereupon it is scrutinized by the community, approved, and transfered to a production repository.
Prerequisite

Be sure that:

- you have all Maven servers defined in your `settings.xml`. For more information, please refer to Maven Committer settings which also apply for Geronimo committers.
- you have created your GPG keys. For more information, please refer to Making GPG Keys.

In order to release a project you must also have the following setup in your `$HOME/.m2/settings.xml` which is a profile that defines the staging repository.

Here's what your release profile might look like in your `$HOME/.m2/settings.xml` :

```
<settings>
    <profiles>
        <profile>
            <id>release</id>
            <properties>
                <gpg.passphrase>secretPhrase</gpg.passphrase>
                <deploy.altRepository>apache.releases::default::scp://people.apache.org/x1/home/[your apache id]
/public_html/staging-repo/${siteId}</deploy.altRepository>
                <staging.siteURL>scp://people.apache.org/x1/home/[your apache id]/public_html/staging-site<
/staging.siteURL>
           </properties>
        </profile>
        <profile>
            <!-- use for local site deploy testing and local deploy testing -->
            <id>local</id>
            <properties>
                <deploy.altRepository>djencks::default::file://[home directory]/staging-repo/${siteId}</deploy.
altRepository>
                <gpg.passphrase>secretPhrase</gpg.passphrase>
                <staging.siteURL>file://[home directory]/staging-site</staging.siteURL>
            </properties>
        </profile>
    </profiles>

    <servers>
        <server>
            <id>apache.releases</id>
            <username>[your apache id]</username>
            <passphrase>[secret passphrase]</passphrase>
            <filePermissions>664</filePermissions>
            <directoryPermissions>775</directoryPermissions>
        </server>
        <server>
            <id>geronimo-website</id>
            <username>[your apache id]</username>
            <passphrase>[secret passphrase]</passphrase>
            <filePermissions>664</filePermissions>
            <directoryPermissions>775</directoryPermissions>
        </server>
    </servers>
</settings>
```


The server name apache.releases at the start of deploy.altRepository must correspond to the apache.releases server definition.
Also that your apache id does not start with "~".

Everything that you need to release has (will have, actually) been configured in the genesis root pom all Geronimo projects inherit from.

Your project should adhere to standard trunk,branches,tags svn layout in which case no further release profile configuration should be needed. Some slight deviation such as our specs project still works without extra configuration. Avoid more complex layouts that require special configuration.

This is the base release configuration in the genesis root pom:

```
<profile>
            <id>release</id>

            <build>
                <plugins>

                    <plugin>
                        <groupId>org.apache.maven.plugins</groupId>
                        <artifactId>maven-release-plugin</artifactId>
                        <configuration>
                            <useReleaseProfile>false</useReleaseProfile>
                            <goals>deploy</goals>
                            <arguments>-Prelease</arguments>
                        </configuration>
```

```
                </plugin>

                <!-- We want a source jar -->
                <plugin>
                    <groupId>org.apache.maven.plugins</groupId>
                    <artifactId>maven-source-plugin</artifactId>
                    <executions>
                        <execution>
                            <goals>
                                <goal>jar</goal>
                            </goals>
                        </execution>
                    </executions>
                </plugin>

                <!-- We want to sign the artifact, the POM, and all attached artifacts -->
                <plugin>
                    <groupId>org.apache.maven.plugins</groupId>
                    <artifactId>maven-gpg-plugin</artifactId>
                    <inherited>true</inherited>
                    <configuration>
                        <passphrase>${gpg.passphrase}</passphrase>
                    </configuration>
                    <executions>
                        <execution>
                            <goals>
                                <goal>sign</goal>
                            </goals>
                        </execution>
                    </executions>
                </plugin>

                <!-- We want to deploy the artifact to a staging location for perusal -->
                <plugin>
                    <groupId>org.apache.maven.plugins</groupId>
                    <artifactId>maven-deploy-plugin</artifactId>
                    <inherited>true</inherited>
                    <configuration>
                        <altDeploymentRepository>${deploy.altRepository}</altDeploymentRepository>
                        <updateReleaseInfo>true</updateReleaseInfo>
                    </configuration>
                </plugin>

                <!-- We want the JavaDoc JAR published with the release -->
                <plugin>
                    <groupId>org.apache.maven.plugins</groupId>
                    <artifactId>maven-javadoc-plugin</artifactId>
                    <inherited>true</inherited>
                    <configuration>
                        <source>1.5</source>
                    </configuration>
                    <executions>
                        <execution>
                            <id>attach-javadocs</id>
                            <goals>
                                <goal>jar</goal>
                            </goals>
                        </execution>
                    </executions>
                </plugin>
            </plugins>
        </build>
    </profile>
```

Release Process as used for Geronimo 2.1.x:

This incorporates some aspects of the maven release process but stops short of actually using the maven release plugin. It utilizes a branch for the release process.

1. when a release is frozen, we spin off a branch with that **exact** name, as in branches/x.y.z, where z starts at zero and increments by one.
2. at that time branches/x.y is immediately updated to version x.y.(z+1)-SNAPSHOT
3. We cut releases from the frozen branch
4. When a release passes final tck testing and final vote, the frozen branch is moved to tags

Updating the poms after making a new branch

Once a new branch is created you will generally need to manage the version number in the poms for parent entries. The following Perl scripts will assist in that task. It could use some polishing but given the relatively infrequent use.

Pom Version Changer

```
perl -i.orig -pe '
$done = 0 if /<?xml/;
$inParent = 1 if not $done and /<parent>/;
s,oldVersion</version>,newVersion</version>, if $inParent and not $done;
$done = $inParent = 1 if /<\/parent>/;
' $(find GeronimoDirectory -name pom.xml | grep -v "GeronimoDirectory/pom.xml")
```

Remember to properly escape periods in the ***oldVersion***. For instance, to change 1.1.1-SNAPSHOT to 1.1.1 you would have

Example

```
perl -i.orig -pe '
$done = 0 if /<?xml/;
$inParent = 1 if not $done and /<parent>/;
s,2\.1\.2-SNAPSHOT</version>,2.1.2</version>, if $inParent and not $done;
$done = $inParent = 1 if /<\/parent>/;
' $(find GeronimoDirectory -name pom.xml | grep -v "GeronimoDirectory/pom.xml")
```


making the above script work

You must replace ***GeronimoDirectory*** above with the fully qualified path to the directory (using "~" will not work).
Also Note: there are references to versions outside of the pom parent entries updated by the script which will also need to be updated with the new version.

In addition to the pom version changes there are several other places where version updates are necessary. To be certain all have been updated you should **grep** the source for old version references to ensure that those remaining are correct. Here is an attempt to capture other version changes necessary:

1. scm entried in root pom.xml
2. assemblies/geronimo-boilerplate-minimal/src/main/underlay/etc/gsh-classworlds.conf
3. framework/configs/geronimo-gbean-deployer/src/it/j2ee-system-2/src/test/resources/META-INF/geronimo-plugin.xml
4. framework/configs/geronimo-gbean-deployer/src/it/j2ee-system-2/src/test/resources/META-INF/plan.xml
5. framework/configs/geronimo-gbean-deployer/src/it/j2ee-system/src/test/resources/META-INF/geronimo-plugin.xml
6. framework/configs/geronimo-gbean-deployer/src/it/j2ee-system/src/test/resources/META-INF/plan.xml
7. framework/configs/geronimo-gbean-deployer/src/it/metadatageneration-2/src/test/resources/META-INF/geronimo-plugin.xml
8. framework/configs/geronimo-gbean-deployer/src/it/metadatageneration/src/test/resources/META-INF/geronimo-plugin.xml
9. framework/configs/plugin/src/main/plan/plan.xml
10. framework/modules/geronimo-plugin/src/test/resources/geronimo-plugins.xml
11. framework/modules/geronimo-upgrade/src/test/data/gbean_1.xml
12. framework/modules/geronimo-upgrade/src/test/data/gbean_1_result.xml
13. plugins/remote-deploy/geronimo-remote-deploy/src/main/webapp/WEB-INF/geronimo-web.xml
14. plugins/welcome/geronimo-welcome/src/main/webapp/WEB-INF/geronimo-web.xml
15. testsuite/webservices-testsuite/jaxrpc-tests/jaxrpc-war/src/main/webapp/WEB-INF/geronimo-web.xml

In addition, there are some special version updates required in some pom.xml files beyond those addressed by the script:

1. assemblies/geronimo-boilerplate-minimal/pom.xml (RELEASE-NOTES filename)
2. framework/configs/geronimo-gbean-deployer/src/it/j2ee-system/pom.xml
3. framework/configs/plugin/pom.xml
4. framework/configs/pom.xml
5. plugins/j2ee/j2ee-deployer/pom.xml
6. plugins/client/pom.xml
7. plugins/corba/client-corba-yoko/pom.xml
8. plugins/j2ee/j2ee-server/pom.xml
9. plugins/pom.xml
10. configs/plugin/pom.xml (plugin-repository-list)
11. root pom ... pom.xml
12. README.txt
13. RELEASE_NOTES-*.txt

Update the on-going maintenance branch

Make similar changes in the branch that will continue on for the next release.

1. Update poms to the next snapshot version
2. Update the same files referenced above
3. rename RELEASE_NOTES-x.y.z.txt as appropriate
4. Some of the "special" updates required are really just applicable here (when creating a new version rather than just removing the snapshot from an existing version). This is especially true for reference to the geronimo plugin repository (which normally named without the snapshot even when the release is still a snapshot release).

In addition, there are special updates for references to release notes and the plugin website:

1. assemblies/geronimo-boilerplate-minimal/pom.xml
2. framework/configs/plugin/pom.xml
3. framework/configs/plugin/src/main/plan/plan.xml

In addition, there are special updates for artifact-alias entries:

1. framework/configs/pom.xml
2. plugins/client/pom.xml
3. plugins/corba/client-corba-yoko/pom.xml
4. plugins/pom.xml

Rationale

We create a branch at freeze time for the following reasons:

1. it takes at least one week from freeze to ship due to voting, tck testing and potential repeats of that process (re-cut, re-certify, re-vote). There is no reason why work on x.y.z+1 needs to be delayed - only 52 weeks a year.
2. stronger guarantee no one is updating the branch once frozen
3. less likely that people and ci systems (continuum) will checkout and build pre-release versions of x.y.z (not x.y.z-SNAPSHOT) which would need to be removed manually and may accidentally be distributed.
4. it is currently very difficult to roll version numbers forward, entries here and there are often missed. Far better to have branches/x.y have a few straggling old x.y.z-SNAPSHOT versions than a few overlooked x.y.z final numbers that needed to go back to SNAPSHOT - they never leave SNAPSHOT and need to be reverted back later if that process happens in the frozen branch.

Creating a Release Candidate

1. Download and install the Gnu Privacy Guard (GPG) from http://www.gnupg.org. Read the documentation on that site and create a key. Have the key signed and verified by others. Submit your public key to http://pgp.mit.edu/. This is a one time process.
2. Be sure that you have your `~/.m2/settings.xml` updated as specified above in Releasing a Geronimo Project
3. Copy (or move as per situation, for eg specs) the trunk/branch to the new branch using the following command.

```
svn mv SRC-URL DEST-URL -m "Reason for this commit".
```

4. Checkout or update this branches tree on your machine.
5. Update the <scm> urls in the `pom.xml` to point to the final url in tags. Eg:

```
<scm>
     <connection>scm:svn:http://svn.apache.org/repos/asf/geronimo/specs/tags/geronimo-servlet_2.5_spec-
1.1</connection>
     <developerConnection>scm:svn:https://svn.apache.org/repos/asf/geronimo/repos/asf/geronimo/specs/tags
/geronimo-servlet_2.5_spec-1.1</developerConnection>
     <url>http://svn.apache.org/viewvc/geronimo/repos/asf/geronimo/specs/tags/geronimo-servlet_2.5_spec-
1.1</url>
</scm>
```

6. Add the following release profile to the root pom.xml (Note: This is a subset of the profile included in the maven release process). This will utilize the maven gpg plugin to sign the artifacts produced and the maven deploy plugin to stage the release to your people.apache staging location but will not utilize the maven release plugin to create the tag, rename versions, etc...

```
<profile>
          <id>release</id>
            <build>
                <plugins>
                    <!-- We want to sign the artifact, the POM, and all attached artifacts -->
                    <plugin>
                        <groupId>org.apache.maven.plugins</groupId>
                        <artifactId>maven-gpg-plugin</artifactId>
                        <inherited>true</inherited>
                        <configuration>
                            <passphrase>${gpg.passphrase}</passphrase>
                        </configuration>
                        <executions>
                            <execution>
                                <goals>
                                    <goal>sign</goal>
                                </goals>
                            </execution>
                        </executions>
                    </plugin>
                    <!-- We want to deploy the artifact to a staging location for perusal -->
                    <plugin>
                        <groupId>org.apache.maven.plugins</groupId>
                        <artifactId>maven-deploy-plugin</artifactId>
                        <inherited>true</inherited>
                        <configuration>
                            <altDeploymentRepository>${deploy.altRepository}</altDeploymentRepository>
                            <updateReleaseInfo>true</updateReleaseInfo>
                        </configuration>
                    </plugin>
                </plugins>
            </build>
        </profile>
```

7. Build the new branches tree as you normally would (including the testsuite) to ensure that it is correct.

```
    mvn install
```

8. After verifying the updated branch can be built and all tests pass, use the rat-maven-plugin to verify that the source contains required license headers.

```
    find . -name target | xargs rm -rf
    mvn rat:check
```


There are a few source files that cannot contain license headers due to the expected format of the file (like LogFactory classes.) Compare these files against the last release to verify it's okay to release them without headers and ask the prior release manager and /or dev list for guidance.

9. Stage the maven artifacts
When you are ready to create a release candidate, clean out the previous build results from the branches tree (otherwise assemblies will not be rebuilt correctly) and rebuild (without tests or testsuite) to now deploy the artifacts to the staging site that will be used for the release vote using the following command.

```
Server 2.0.x/2.1.x -
    find . -name target | xargs rm -rf
    mvn -Pstaging deploy
Server 2.2 -
    find . -name target | xargs rm -rf
    mvn -Prelease,no-it deploy
```

Be extremely careful with this step. There have been times when a slight modification in the above command will result in items being deployed directly to the rsync location rather than to the staging location.
Before running this step, verify that there is a corresponding "staging" or "release" profile in pom.xml.
If there are issues/problems with the deploy to the staging location you may have to stage to a local repository and then scp the content to an apache staging location.

10. Export and tar/zip the source to be voted on for the release using the following commands:

```
svn export https://svn.apache.org/repos/asf/geronimo/server/branches/2.1.2 geronimo-2.1.2-src
tar -zcvf geronimo-2.1.2-src.tar.gz geronimo-2.1.2-src
```

Also create a zip of the source image.

11. Create appropriate md5, sha1, and signatures (asc) for the artifacts added manually. A script similar to this can be used to create the checksums and signature:

signDist.sh

```
gpg --print-md MD5 ./geronimo-2.1.2-src.tar.gz > ./geronimo-2.1.2-src.tar.gz.md5
gpg --print-md MD5 ./geronimo-2.1.2-src.zip > ./geronimo-2.1.2-src.zip.md5
gpg --print-md MD5 ./RELEASE_NOTES-2.1.2.txt > ./RELEASE_NOTES-2.1.2.txt.md5
gpg --print-md MD5 ./README.txt > ./README.txt.md5
gpg --print-md MD5 ./NOTICE.txt > ./NOTICE.txt.md5
gpg --print-md MD5 ./LICENSE.txt > ./LICENSE.txt.md5
gpg --print-md MD5 ./DISCLAIMER.txt > ./DISCLAIMER.txt.md5
gpg --print-md SHA1 ./geronimo-2.1.2-src.tar.gz > ./geronimo-2.1.2-src.tar.gz.sha1
gpg --print-md SHA1 ./geronimo-2.1.2-src.zip > ./geronimo-2.1.2-src.zip.sha1
gpg --print-md SHA1 ./RELEASE_NOTES-2.1.2.txt > ./RELEASE_NOTES-2.1.2.txt.sha1
gpg --print-md SHA1 ./README.txt > ./README.txt.sha1
gpg --print-md SHA1 ./NOTICE.txt > ./NOTICE.txt.sha1
gpg --print-md SHA1 ./LICENSE.txt > ./LICENSE.txt.sha1
gpg --print-md SHA1 ./DISCLAIMER.txt > ./DISCLAIMER.txt.sha1
gpg --armor --output ./geronimo-2.1.2-src.tar.gz.asc --detach-sig ./geronimo-2.1.2-src.tar.gz
gpg --armor --output ./geronimo-2.1.2-src.zip.asc --detach-sig ./geronimo-2.1.2-src.zip
gpg --armor --output ./RELEASE_NOTES-2.1.2.txt.asc --detach-sig ./RELEASE_NOTES-2.1.2.txt
gpg --armor --output ./README.txt.asc --detach-sig ./README.txt
gpg --armor --output ./NOTICE.txt.asc --detach-sig ./NOTICE.txt
gpg --armor --output ./LICENSE.txt.asc --detach-sig ./LICENSE.txt
gpg --armor --output ./DISCLAIMER.txt.asc --detach-sig ./DISCLAIMER.txt
```


Ensure that your public PGP key is in appropriate public locations (esp. /www/www.apache.org/dist/geronimo/KEYS on people and http://pgp.mit.edu/) and that your pgp key has been signed by several other apache committers to create a web of trust.

You will need to copy the RELEASE_NOTES-x.x.x.txt from the target location of one of the built assemblies rather than the root of the branch as the version attributes are updated as part of the build.

12. Update the plugins site - http://cwiki.apache.org/GMOxDEV/steps-to-create-a-plugin-repository-for-a-new-geronimo-release.html
    a. Update your local ~/.m2/repository/geronimo-plugins.xml file from a clean server build to:
        i. Remove all occurrences of your local repo

```
<source-repository>~/.m2/repository/</source-repository>
```

        ii. Verify that each entry points to the 2 external repos

```
<source-repository>http://repo1.maven.org/maven2/</source-repository>
<source-repository>http://repository.apache.org/snapshot</source-repository>
```

        iii. Verify that each entry lists JVM 1.5 and 1.6 targets if the release is prior to 3.0. For all 3.x releases, only 1.6 should be specified.

```
<jvm-version>1.5</jvm-version>
<jvm-version>1.6</jvm-version>
```

        iv. Update the default repository values by replacing the local repo reference with

```
<default-repository>http://geronimo.apache.org/plugins/geronimo-2.1.3/</default-repository>
<default-repository>http://repo1.maven.org/maven2/</default-repository>
<default-repository>http://www.ibiblio.org/maven2/</default-repository>
```

b. Upload the updated geronimo-plugins.xml to the plugin website in svn

```
https://svn.apache.org/repos/infra/websites/production/geronimo/content/plugins/geronimo-2.1.3/
```

This will get automatically published on http://geronimo.apache.org/plugins/geronimo-2.1.3.

c. Create the plugin-repository-list-2.1.4.txt for the on-going maintenance branch with

```
http://geronimo.apache.org/plugins/geronimo-2.1.4/
```

d. Create the geronimo-2.1.4 directory for the on-going maintenance branch and seed it with a copy of the .htaccess from the prior release and the following geronimo-plugins.xml content (until a 2.1.4-SNAPSHOT build has been published and new geronimo-plugins.xml file created)

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<geronimo-plugin-list xmlns:ns2="http://geronimo.apache.org/xml/ns/attributes-1.2" xmlns="
http://geronimo.apache.org/xml/ns/plugins-1.3">

    <default-repository>http://geronimo.apache.org/plugins/geronimo-2.1.4/</default-repository>
    <default-repository>http://repo1.maven.org/maven2/</default-repository>
    <default-repository>http://www.ibiblio.org/maven2/</default-repository>
</geronimo-plugin-list>
```

e. Commit the changes to svn

13. Create a distribution directory on people.apache.org (like ~/public_html/releases/geronimo-2.1.3-RC1) and upload the following artifacts for the vote:
    a. Source tar & zip
    b. All assembly images that are part of the vote. This is more for convenience of the voters and to serve as a distribution location once the vote is complete.
    c. Release notes, README, LICENSE, NOTICE, and DISCLAIMER
    d. MD5, SHA1 and ASC files for all of the above

Calling a Vote

1. Put it up for a vote along with the staging release you created earlier. In the vote notice, please include the precise names and versions being voted on (e.g. geronimo-jetty6-javaee5-2.1.3), the svn revision and urls to the current source along with where the tag will be created.
    a. Start a vote thread for the release

```
To: "Geronimo Developers List" <dev@geronimo.apache.org>
Cc: "Geronimo Users List" <user@geronimo.apache.org>
Subject: [VOTE] Release Geronimo Server 2.1.3 RC1

All,

I've prepared a release candidate of Geronimo Server 2.1.3 for your review and vote.

The source for RC1 is Rev<XXXXX> from the following svn branch:
<branch URL>

When the release vote is approved, I will svn mv the code to:
<future tag URL>

The following staging directory contains the server binary assemblies/distributions
to be released (framework, tomcat/jetty, Java EE/Minimal, tar/zip) as well as the
RELEASE_NOTES, README, NOTICE, LICENSE and source code archives for the release.
<distribution dir URL>

For your convenience, here are pointers to the JavaEE distributions:
<URLs>

The maven artifacts for the release can be found here:
<staging-repo URL>

When the release vote is approved, these maven artifacts will be moved
to the m2-ibiblio-rsync-repository at Apache.


[ ] +1 Release Geronimo Server 2.1.3
[ ] 0 No opinion
[ ] -1 Do not release Geronimo Server 2.1.3 (please provide rationale)

The voting will be open for 72 hours or until the tck results are verified, whichever is longer.
```

b. Start a discussion thread for the vote

```
To: "Geronimo Developers List" <dev@geronimo.apache.org>
Cc: "Geronimo Users List" <user@geronimo.apache.org>
Subject: [DISCUSS] Release Geronimo Server 2.1.3 RC1

Discussion thread for "[VOTE] Release Geronimo Server 2.1.3 RC1"
```

2. Send out the vote results in a reply email
   a. If the vote passed, send a reply to the original [VOTE] email with -

```
Subject: [RESULTS] Release Geronimo Server 2.1.3 RC1

The vote passed with:

[xx] +1
[xx] +0
[xx] -1

I'll start publishing the artifacts, updating the download website with the new release and create
a News entry for our website.
```

   b. If the vote was canceled and another release candidate is being created for another vote

```
Subject: [CANCELED] Release Geronimo Server 2.1.3 RC1

The vote is being canceled due to xxxx and a new release candidate will be created for another
vote.
```

Publishing a Release Candidate

1. Copy from the staging repo to the production repo.
   Once the release is deemed fit for public consumption it can be transfered to a production repository where it will be available to all users. Note: Current version of the stage plugin is 1.0-alpha-1.
   Here is an example on how to use the stage plugin:

   ```
   mvn stage:copy -Dsource="http://people.apache.org/<your apache id>/staging-repo/<siteId>" \
     -Dtarget="scp://people.apache.org/www/people.apache.org/repo/m2-ibiblio-rsync-repository" \
     -Dversion=2.1.2 \
     -DtargetRepositoryId=apache.releases
   ```


   The version parameter is currently ignored but specify the correct version anyway. The entire staging repository is synced, not just the given version or the current project.

   Also note, this process has been known to have problems at times if a maven version other than 2.0.8 is used.

2. Copy the distribution content from the staging location to the production repo.
   Check in the distribution content into the appropriate location in https://dist.apache.org/repos/dist/release/geronimo/.
3. Move the branches to tags using the following command.

   ```
   svn mv SRC-URL DEST-URL -m "Reason for this commit".
   ```

4. Update the Geronimo site download pages with a new page for the release Geronimo Downloads . Include the list JIRA for know issues by first generating a JIRA query for the release, right clicking on the XML format and saving the url to inclusion in the download page.
5. Administer the GERONIMO JIRA project to update the released and unreleased versions
6. Update the Geronimo Home Page with a News item that a new Geronimo Server release is available
7. Update the Geronimo Release Roadmaps with the release date.
8. Final step, is to force the Confluence Autoexport plugin to run for the following spaces: Apache Geronimo (GMOxSITE), Apache Geronimo Project Management (GMOxPMGT), Apache Geronimo 2.1 (GMOxDOC21)

Notice

The original process in this document was voted on by the Geronimo community. Please formally propose all changes to dev@geronimo.apache.org.

See:

1. http://marc.theaimsgroup.com/?l=geronimo-dev&m=115094116905426&w=4

Revised process using maven tools voted on in March 2008. Only major structural changes now require votes.

See: (not yet in archive)