Administering plugins

{scrollbar}

Administering plugins

Creating a plugin

- You can create a plugin as part of a maven build using the car-maven-plugin.
- You can create a plugin "virtually" by installing a deployed application from a running Geronimo server acting as a plugin repository.
- You can create a plugin using the Geronimo administration console to create or edit the plugin metadata.

By far the easiest way to build a Geronimo plugin is with maven using the car-maven-plugin. Any such module will include a `geronimo-plugin.xml` descriptor with at least minimal information. When possible, such as the description and license, this information is taken from the pom itself. Normally you will build the dependency list from the modules dependencies which are constructed from the maven dependencies plus whatever additional dependencies the deployers determine are needed. For instance an ejb application will have the openejb plugin added as a dependency by the openejb deployer. If necessary you can specify the dependencies for both the module and plugin descriptor explicitly in the car-maven-plugin configuration.

Here's an example of a car-maven-plugin configuration using maven dependencies and configuring most of the additional information possible:

xmlexcerpt from pom.xml <?xml version="1.0" encoding="UTF-8"?> <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLofcation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd"> <modelVersion>4.0.0</modelVersion> <parent> <groupId>org.apache.geronimo.plugins</groupId> <artifactId>plugins</artifactId> <version>2.2-SNAPSHOT</version> </parent> <groupId>org.apache.geronimo.configs</groupId> <artifactId>sharedlib</artifactId> <name>Geronimo Plugins, Shared Library</name> <description>Shared Library GBean</description> <packaging>car</packaging> <build> <plugins> <plugin> <groupId>org.apache.geronimo.buildsupport</groupId> <artifactId>car-maven-plugin</artifactId> <configuration> <category>Shared</category> <useMavenDependencies> <value>true</value> <includeVersion>true</includeVersion> </useMavenDependencies> <instance> <plugin-artifact> <copy-file relative-to="server" dest-dir="var">shared</copy-file> </plugin-artifact> </instance> </configuration> </plugin> </plugins> </build> </project>

As you use maven to build plugins, a `geronimo-plugins.xml` plugin catalog is automatically maintained in your local maven repository. You can force this to be rebuilt by running

mvn org.apache.geronimo.buildsupport:car-maven-plugin:create-pluginlist

This might be necessary if you prune your maven repository and remove plugins listed in the catalog.

Alternatively, you can construct the `geronimo-plugin.xml` file by hand and include it in a deployed module in a geronimo server.

The administration console also allows limited editing of `geronimo-plugin.xml` files but editing the information about how the plugin fits into the server is not yet supported.

Installing a plugin

If the appropriate administration console plugin is installed (and your Geronimo server includes Web application support) you can install plugins from a plugin repository. After selecting the **Plugins** page from the navigation menu select the plugin repository you want, such as your local maven repository if you have been building your own plugins. Next you see a list of available plugins from the repository. Select multiple plugins using the checkboxes or a single plugin as a link, and on the next page you will see more information on the plugins. On your approval the plugins will be downloaded and installed.

Alternatively you can use GShell to install plugins using the **deploy/install-plugin** command. This can be run with a command line or interactively. Interactively you can select the plugin repository to use (if more than one is known), and then select the plugins to install. Again, they will be downloaded and installed. An example of command line usage will be seen as followed:

You can install a plugin into an existing server in different ways:

- GShell deploy/install-plugin command
- Geronimo administrative console
- Using maven and the geronimo-maven-plugin

You can also install a plugin into a new server assembly using the car-maven-plugin.
Note that in all cases the dependency system assure that if you install a plugin, everything needed to run the plugin will also be installed. For instance if you install a Java EE application plugin such as one of the samples into the framework server, openejb, openjpa, the transaction manager and connector framework and the appropriate Web container will also be installed as dependencies.

Updating a plugin

At times, you may need to upgrade a plugin or jar version, for instance if a new version of a dependency is released but you cannot rerelease all the artifacts that depend on it. Here are some methods to upgrade jar versions.

Simple jar upgrade

If the jar is to be installed as part of a plugin installation, see the section below. Otherwise, follow these steps.

1. If the server is running, stop the server.
2. Copy the new jar into the appropriate directory in your geronimo server's repository. For instance: mkdir -p repository/org/foo/myjar/1.1/ cp ~ /newFooJar/myjar-1.1.jar repository/org/foo/myjar/1.1/ Alternatively, the administration console portlet **Services->Repository** can be used to add artifacts to the server's repository.

Finally, after the new jar is installed in the server's repository, add a line to `var/config/artifact_aliases.properties` (or the equivalent file, if the server is using a non-standard alias file). For instance, to replace `myjar-1.0.jar` with `myjar-1.1.jar`:

org.foo/myjar/1.0/jar=org.foo/myjar/1.1/jar

With this configuration, the server will substitute myjar-1.1.jar for any myjar-1.0.jar dependency.

Upgrading a jar while releasing a plugin

If the jar is installed as part of a plugin installation, you can include configuration upgrade information in the `geronimo-plugin.xml`. During plugin installation, the upgraded jar will be automatically installed. This is easiest to specify in the car-maven-config configuration in the pom.xml, prior to building the plugin.

<artifact-alias key="org.foo/myjar/1.0/jar">org.foo/myjar/1.1/jar</artifact-alias>