Apache UIMA PEAR Packaging Maven Plugin Documentation

Authors: The Apache UIMA Development Community

Version 2.2.2

Table of Contents

Overview

To make UIMA analysis components easily consumable for other users, UIMA defines the PEAR (Processing Engine ARchive) file format. This allows applications and tools to manage UIMA analysis components automatically for verification, deployment, invocation and testing. So far the PEAR file format is the desired output format for an UIMA analysis component. With the PEAR packaging Maven plugin it is possible to create such a PEAR package for the analysis component automatically during the Maven artifact build.

Chapter 1. Installing The PEAR Packaging Plugin From Source

The PEAR packaging Maven plugin is modeled as custom plugin for the Maven build system. To use it, the plugin must be available in the local Maven repository. So far the plugin is not available by default when building the UIMA SDK, so it have to be built manually using the provided Maven build. The plugin code is available in the Apache subversion repository at:

> http://svn.apache.org/repos/asf/incubator/uima/sandbox/trunk/PearPackagingMavenPlugin

After downloading the code to the Apache UIMA build environment just call the provided maven build in the main directory of the PEAR Packaging Plugin as shown in the listing below.

```
#PearPackagingMavenPlugin> mvn install
```

After successfully building the plugin is available in the local Maven repository and can be used in other Maven artifact builds from now. To integrate the plugin into other builds it is necessary to know the PEAR Packaging Plugin version number as it is available in the Maven repository. The plugin version number is displayed at the end of the Maven build as shown in the example below. For this example, the plugin version number is: `2.2.2-incubating`

```
[INFO] Installing
/code/apache/PearPackagingMavenPlugin/target/
PearPackagingMavenPlugin-2.2.2.-incubating.jar
to
/maven-repository/repository/org/apache/uima/PearPackagingMavenPlugin/
2.2.2-incubating/
PearPackagingMavenPlugin-2.2.2-incubating.jar
[INFO] [plugin:updateRegistry]
[INFO] ------------------------------------------------------------------
[INFO] BUILD SUCCESSFUL
[INFO] ------------------------------------------------------------------
[INFO] Total time: 6 seconds
[INFO] Finished at: Tue Nov 13 15:07:11 CET 2007
[INFO] Final Memory: 10M/24M
[INFO] ------------------------------------------------------------------
```

To use the PEAR Packaging Plugin in a Maven build, it is necessary to add it as dependency to the POM. Important while adding the dependency is the version number of the plugin. With that the correct version of the plugin is used and can be found in the local Maven repository. The version number of the plugin was displayed when installing the PEAR packaging Maven plugin to the Maven repository. The snippet below shows how the plugin dependency is added to the dependencies sections of the POM.

```
<dependencies>
   ...
   <dependency>
     <groupId>org.apache.uima</groupId>
     <artifactId>PearPackagingMavenPlugin</artifactId>
     <version>2.2.0-incubating-SNAPSHOT</version>
     <scope>package</scope>
   </dependency>
   ....
</dependencies>
```

Chapter 2. Installing The PEAR Packaging Plugin Using The Binary

It is also possible to use the binary version of the plugin. In that case the `uima-pear-maven-plugin.jar` file must be added to the dependency set of your POM like for example:

```
<dependencies>
   ...
   <dependency>
     <groupId>org.apache.uima</groupId>
     <artifactId>PearPackagingMavenPlugin</artifactId>
     <version>2.2.2-incubating</version>
     <scope>system</scope>
     <systemPath>${basedir}/lib/uima-pear-maven-plugin.jar</systemPath>
   </dependency>
   ....
</dependencies>
```

Chapter 3. Using the PEAR Packaging Plugin

3.1. Calling The PEAR Packaging Plugin

To call the PEAR Packaging Plugin within a Maven build, the plugin must be added to the plugins section of the POM as shown below:

```
<build>
 <plugins>
  ...
  <plugin>
    <groupId>org.apache.uima</groupId>
    <artifactId>PearPackagingMavenPlugin</artifactId>
    <extensions>true</extensions>
    <executions>
      <execution>
        <phase>package</phase>
        <configuration>

          <classpath>
            <!-- PEAR file component classpath settings -->
            $main_root/lib/sample.jar
          </classpath>

          <mainComponentDesc>
            <!-- PEAR file main component descriptor -->
            desc/${artifactId}.xml
          </mainComponentDesc>

          <componentId>
            <!-- PEAR file component ID -->
            ${artifactId}
          </componentId>

          <datapath>
            <!-- PEAR file UIMA datapath settings -->
            $main_root/resources
          </datapath>

        </configuration>
        <goals>
          <goal>package</goal>
        </goals>
      </execution>
    </executions>
  </plugin>
  ...
 </plugins>
</build>
```

To configure the plugin with the specific settings of a PEAR package, the `<configuration>` element section is used. This sections contains all parameters that are provided by the PEAR Packaging Plugin to package the right content and set the specific PEAR package settings. The details about each parameter and how it is used is shown below:

- `<classpath>` - This element specifies the classpath settings that are necessary to start up the PEAR component. The jar artifact that is built during the current Maven build is automatically added to the PEAR classpath settings and do not have to be added manually. The classpath element can be removed if the component jar artifact is the only classpath entry.

  Note: Use $main_root variables to refer libraries inside the PEAR package. For more details about PEAR packaging please refer to the Apache UIMA PEAR documentation.

- `<mainComponentDesc>` - This element specifies the relative path to the main component descriptor that should be used to run the PEAR content. The path must be relative to the project root. It is a good default to use `desc/${artifactId}.xml` here.

- `<componentID>` - This element specifies the PEAR package component ID. It is a good default to use `${artifactId}`.

- `<datapath>` - This element specifies the PEAR package UIMA datapath settings. If no datapath settings are necessary, this element can be removed.

  Note: Use $main_root variables to refer libraries inside the PEAR package. For more details about PEAR packaging please refer to the Apache UIMA PEAR documentation.

For most Maven projects it is sufficient to specify the parameters described above. In some cases, for more complex projects, it may be necessary to specify some additional configuration parameters. These parameters are listed below with the default values that are used if they are not added to the configuration section shown above.

- `<mainComponentDir>` - This element specifies the main component directory where the UIMA nature is applied. By default this parameter points to the project root directory - ${basedir}.

- `<targetDir>` - This element specifies the target directory where the result of the plugin are written to. By default this parameters points to the default Maven output directory - ${basedir}/target

3.2. Automatically including dependencies

The PEAR Packaging Plugin does not take care of automatically adding its dependencies to the PEAR archive. However, this behavior can be manually added. The following two

build plugins hook into the build cycle in order to make sure all runtime dependencies end up being packaged into the final PEAR file.

> **Note:** While the dependencies will be automatically included in the PEAR file using this procedure, you still need to add them manually to the PEAR classpath.

The first uses the `maven-dependency-plugin` to resolve the runtime dependencies of the PEAR into the `lib` folder, which is where the PEAR packaging plugin expects them. The version is stripped from the file names of the dependencies so it is not necessary to adapt the PEAR classpath when changing the version of a dependency.

```
<build>
 <plugins>
  ...
  <plugin>
   <groupId>org.apache.maven.plugins</groupId>
   <artifactId>maven-dependency-plugin</artifactId>
   <executions>
    <!-- Copy the dependencies to the lib folder for the PEAR to copy -->
    <execution>
     <id>copy-dependencies</id>
     <phase>package</phase>
     <goals>
      <goal>copy-dependencies</goal>
     </goals>
     <configuration>
      <stripVersion>true</stripVersion>
      <outputDirectory>${basedir}/lib</outputDirectory>
      <overWriteReleases>false</overWriteReleases>
      <overWriteSnapshots>true</overWriteSnapshots>
      <includeScope>runtime</includeScope>
     </configuration>
    </execution>
   </executions>
  </plugin>
  ...
 </plugins>
</build>
```

The second hooks into the `clean` phase of the build phase. It deletes the `lib` folder again.

> **Note:** The `lib` folder is automatically filled and removed during the build process. Therefore, it should not go into the source control system and neither should you manually place any jars in there.

```
<build>
 <plugins>
  ...
  <plugin>
   <artifactId>maven-antrun-plugin</artifactId>
   <executions>
```

```
    <!-- Clean the libraries after packaging -->
    <execution>
     <id>CleanLib</id>
     <phase>clean</phase>
     <configuration>
      <tasks>
       <delete dir="lib" deleteOnExit="true" quiet="true" failOnError="false">
        <fileset dir="lib" includes="*.jar"/>
       </delete>
      </tasks>
     </configuration>
     <goals>
      <goal>run</goal>
     </goals>
    </execution>
   </executions>
  </plugin>
  ...
 </plugins>
</build>
```