Overview

It is occasionally useful to measure the code coverage for manual and/or automated tests which run against a test environment. (Not to be confused with measuring code coverage of unit tests executed automatically by the build system.) For example, you may have a set of automated regression tests written using JMeter, Selenium, etc. which for some reason or another are not appropriate to run as part of the maven build.

To accomplish this the following high level steps are required.

1. Create the deployment artifact (typically a WAR file) using Cobertura instrumented classes.
2. Install the deployment artifact within an appropriate test environment.
3. Execute a relevant set of tests which leverage the deployed code.
4. Collect and merge the cobertura ser file(s) and generate a coverage report.

Technical Considerations

Measuring coverage for a build artifact deployed within a test infrastructure differs in several meaningful ways from the more common scenario which focus on unit test coverage. An awareness of these differences makes it much easier to understand the nitty-gritty Maven details which follow.

- Execution of the tests happens outside of a Maven environment. In many cases Maven is simply the wrong tool for executing these sorts of tests.
- To generate a coverage report will require the cobertura ser file associated with each jar in the deployment artifact to be merged with the cobertura file created when the deployment artifact is run.

  The cobertura ser file produced when the classes within a given jar are instrumented record the number of executable lines within the jar along with various other meta-data. The cobertura ser file created (or modifications thereof) when the deployment artifact is executed only records information related to the lines of code which were executed. Consequently, a coverage report created using only the cobertura ser created by the execution of the deployment artifact will falsely report 100% coverage.

  This can take a few seconds to wrap your mind around. The trick is to understand that since the cobertura ser file created by the deployment artifact only records lines which were executed, a report based on such a ser file observes that 100% of what the ser file knows about has been covered.

- To generate a coverage report will require the source code corresponding to every jar of interest within the deployment artifact. The source used should line up exactly with that used to create the jars. The most recent source code from source control won't do. The easiest way to obtain source code which correlates with an instrumented classes jar artifact is simply to ensure the maven source plugin creates a source jar as an attached artifact to the instrumented classes jar.

Maven Configuration Details

Overview

The approach shown here addresses the technical considerations discussed in the overview section.

Several key aspects of the maven configuration are listed below.

- Each child of a multi-module project which contains java code provides the cobertura.ser and sources jar as attached artifacts whenever the relevant profile is engaged.
- Each child of a multi-module project which contains java code includes the cobertura-runtime as a "provided" scope dependency whenever the relevant profile is engaged. (The use of provided scope assumes the container into which the deployment artifact is deployed will provide the cobertura jar. If this is not the case you may need to use compile scope.)
- An additional cobertura focused child module is introduced to the multi-module build. This child uses the assembly plugin to create an archive which collects all the necessary files needed to generate a coverage report and registers it as an attached artifact. The assembled archive will include sources and cobertura.ser files from each sibling, along with a short ant script capable of building a coverage report for the deployment artifact.
- Assuming the deployment artifact is deployed within a web container the container must be configured to provide the cobertura jar. In the case of Tomcat you simply drop the cobertura jar in Tomcat's common/lib directory. You should also ensure all the runtime dependencies of the cobertura jar are available, but as it turns out at the time of writing there are not any. (See the cobertura-runtime artifact's POM for runtime dependency details.)
- Assuming the deployment artifact is deployed within a web container the coberturaFlush.war should be installed as well. Every time the coberturaFlush/flushCobertura URL is accessed, the cobertura.ser file used by the deployment artifact will updated. The coberturaFlush.war can be found in newer Cobertura binary releases. It doesn't currently appear in the maven repository. If the coberturaFlush.war is not used the cobertura.ser file will only be written when the container is shutdown and then only if the container shuts down cleanly such that the shutdown hooks run properly. See http://cobertura.sourceforge.net/faq.html for details.

  For the coberturaFlush WAR to work correctly, it must load cobertura using the same classloader the deployment artifact being measured is using. This is why the cobertura jar is listed at provided scope within Maven. By using provided scope the cobertura jar is excluded from the deployment artifact (WAR file), with the deployed code relying upon the web container to provide it. In the case of Tomcat common/lib is used rather than shared/lib since cobertura must be loaded by a classloader common to the coberturaFlush WAR and the WAR being measured.

Relevant Enhancements to the Parent POM

The parent pom should set up the cobertura:instrument and source:jar goals. Notice the instrument goal has been configured with a true attach value.

```
<?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:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xs
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.mycompany.someproject</groupId>
  <artifactId>SomeProject-parent</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>pom</packaging>
  <name>SomeProject Parent</name>
  <modules>
    <module>SomeProject-libA</module>
    <module>SomeProject-libB</module>
    <module>SomeProject-webapp</module>
  </modules>
  <build>
    ...
    <pluginManagement>
      <plugins>
        <plugin>
```

```
          <groupId>org.codehaus.mojo</groupId>
          <artifactId>cobertura-maven-plugin</artifactId>
          <version>2.4</version>
        </plugin>
        ...
      </plugins>
    </pluginManagement>
  </build>
  <properties>
    <cobertura.version>1.9.2</cobertura.version>
    ...
  </properties>
  <profiles>
    <profile>
      <id>cobertura-instrument</id>
      <activation>
        <property>
          <name>cobertura-build</name>
        </property>
      </activation>
      <modules>
        <module>SomeProject-cobertura</module>
      </modules>
      <build>
        <plugins>
          <plugin>
            <groupId>org.codehaus.mojo</groupId>
            <artifactId>cobertura-maven-plugin</artifactId>
            <configuration>
            </configuration>
            <executions>
              <execution>
                <id>instrument-code</id>
                <phase>process-classes</phase>
                <goals>
                  <goal>instrument</goal>
                </goals>
                <configuration>
                  <attach>true</attach>
                </configuration>
              </execution>
            </executions>
          </plugin>
          <plugin>
            <artifactId>maven-source-plugin</artifactId>
            <executions>
              <execution>
                <id>attach-sources</id>
                <goals>
                  <goal>jar</goal>
                </goals>
              </execution>
            </executions>
            <inherited>true</inherited>
          </plugin>
        </plugins>
      </build>
      <dependencies>
        <dependency>
          <groupId>net.sourceforge.cobertura</groupId>
          <artifactId>cobertura-runtime</artifactId>
          <version>${cobertura.version}</version>
          <scope>provided</scope>
          <type>pom</type>
        </dependency>
      </dependencies>
    </profile>
    ...
  </profiles>
</project>
```

Changes to Typical Child POM

In most cases the child POMs will remain untouched. All the necessary changes will be inherited from the parent pom. This is true for both jar and war packaging types.

Even if the cobertura:instrument goal is configured to include a module's cobertura.ser file as an attached artifact, it will gracefully skip doing so if no cobertura.ser file was produced. This feature makes it much easier to avoid additional cobertura specific clutter in a typical child POM.

Contents of Cobertura Specific Child POM

As discussed above the cobertura specific child module will assemble all the files necessary to generate a coverage report for the deployed artifact. This assembly will be registered as an attached artifact and therefore be co-deployed with all the other build artifacts.

The following example files should help you to quickly produce an appropriate cobertura focused child module.

SomeProject-cobertura POM file

This is an example POM file for the cobertura focused child module. Following the example it would be located at SomeProject-cobertura/pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.x
  <parent>
    <groupId>com.mycompany.someproject</groupId>
    <artifactId>SomeProject-parent</artifactId>
    <version>1.0-SNAPSHOT</version>
  </parent>
  <modelVersion>4.0.0</modelVersion>
  <artifactId>SomeProject-cobertura</artifactId>
  <name>SomeProject Cobertura</name>
  <packaging>pom</packaging>
  <build>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-assembly-plugin</artifactId>
        <configuration>
          <descriptors>
            <descriptor>${basedir}/src/main/assembly/coberturaReporting.xml</descriptor>
          </descriptors>
          <attach>true</attach>
        </configuration>
```

```
          <executions>
            <execution>
              <id>bundle-configs</id>
              <phase>package</phase>
              <goals>
                <goal>single</goal>
              </goals>
            </execution>
          </executions>
        </plugin>
      </plugins>
    </build>
    <dependencies>
      <dependency>
        <groupId>net.sourceforge.cobertura</groupId>
        <artifactId>cobertura</artifactId>
        <!--cobertura.version is inherited from the parent POM-->
        <version>${cobertura.version}</version>
        <scope>compile</scope>
      </dependency>
      <dependency>
        <groupId>${project.groupId}</groupId>
        <artifactId>SomeProject-webapp</artifactId>
        <version>${project.version}</version>
        <classifier>sources</classifier>
        <type>jar</type>
        <scope>compile</scope>
      </dependency>
      <dependency>
        <groupId>${project.groupId}</groupId>
        <artifactId>SomeProject-webapp</artifactId>
        <version>${project.version}</version>
        <classifier>cobertura</classifier>
        <type>ser</type>
        <scope>compile</scope>
      </dependency>
      <dependency>
        <groupId>${project.groupId}</groupId>
        <artifactId>SomeProject-libA</artifactId>
        <version>${project.version}</version>
        <classifier>sources</classifier>
        <type>jar</type>
        <scope>compile</scope>
      </dependency>
      <dependency>
        <groupId>${project.groupId}</groupId>
        <artifactId>SomeProject-libA</artifactId>
        <version>${project.version}</version>
        <classifier>cobertura</classifier>
        <type>ser</type>
        <scope>compile</scope>
      </dependency>
      <dependency>
        <groupId>${project.groupId}</groupId>
        <artifactId>SomeProject-libB</artifactId>
        <version>${project.version}</version>
        <classifier>sources</classifier>
        <type>jar</type>
        <scope>compile</scope>
      </dependency>
      <dependency>
        <groupId>${project.groupId}</groupId>
        <artifactId>SomeProject-libB</artifactId>
        <version>${project.version}</version>
        <classifier>cobertura</classifier>
        <type>ser</type>
        <scope>compile</scope>
      </dependency>
    </dependencies>
  </project>
```

Assembly Descriptor

This assembly descriptor will typically be found at src/main/assembly/coberturaReporting.xml within the cobertura focused child module.

```
<assembly>
  <!--id is used for classifier in attached artifact-->
  <id>coberturaReporting</id>
  <formats>
    <format>tar.gz</format>
    <!--format>dir</format-->
  </formats>
  <includeBaseDirectory>false</includeBaseDirectory>
  <files>
    <file>
      <source>${basedir}/src/main/coberturaScripts/cobertura.xml</source>
      <outputDirectory>/</outputDirectory>
      <filtered>false</filtered>
    </file>
    <file>
      <source>${basedir}/src/main/coberturaScripts/cobertura.properties</source>
      <outputDirectory>/</outputDirectory>
      <filtered>true</filtered>
    </file>
  </files>
  <dependencySets>
    <dependencySet>
      <unpack>false</unpack>
      <outputDirectory>/antjars</outputDirectory>
      <scope>compile</scope>
      <includes>
        <include>net.sourceforge.cobertura:cobertura</include>
      </includes>
      <useTransitiveDependencies>true</useTransitiveDependencies>
      <useTransitiveFiltering>true</useTransitiveFiltering>
    </dependencySet>
    <dependencySet>
      <unpack>true</unpack>
      <outputDirectory>/sources/SomeProject-webapp</outputDirectory>
      <scope>compile</scope>
      <includes>
        <include>com.mycompany.someproject:SomeProject-webapp:jar:sources</include>
      </includes>
    </dependencySet>
    <dependencySet>
      <unpack>true</unpack>
      <outputDirectory>/sources/SomeProject-libA</outputDirectory>
      <scope>compile</scope>
      <includes>
        <include>com.mycompany.someproject:SomeProject-libA:jar:sources</include>
      </includes>
    </dependencySet>
    <dependencySet>
      <unpack>true</unpack>
      <outputDirectory>/sources/SomeProject-libB</outputDirectory>
      <scope>compile</scope>
      <includes>
        <include>com.mycompany.someproject:SomeProject-libB:jar:sources</include>
      </includes>
    </dependencySet>
    <dependencySet>
      <unpack>false</unpack>
      <outputDirectory>/cobertura</outputDirectory>
      <scope>compile</scope>
      <includes>
        <include>com.mycompany.someproject:SomeProject-webapp:ser:cobertura</include>
      </includes>
    </dependencySet>
    <dependencySet>
      <unpack>false</unpack>
      <outputDirectory>/cobertura</outputDirectory>
      <scope>compile</scope>
      <includes>
        <include>com.mycompany.someproject:SomeProject-libA:ser:cobertura</include>
      </includes>
    </dependencySet>
    <dependencySet>
      <unpack>false</unpack>
      <outputDirectory>/cobertura</outputDirectory>
      <scope>compile</scope>
      <includes>
        <include>com.mycompany.someproject:SomeProject-libB:ser:cobertura</include>
      </includes>
    </dependencySet>
  </dependencySets>
</assembly>
```

Cobertura Ant Script Included in Assembly

This ant script will typically be found at src/main/coberturaScripts/cobertura.xml within the cobertura focused child module.

This ant script uses the cobertura-merge ant task to merge the cobertura.ser files (including the one produced by the deployed artifact), and then uses the the cobertura-report ant task to generate an html report.

Although the script happens to be an ant script, any appropriate scripting tool would work here so long as it can execute the necessary cobertura libraries. It is important to realize, that although the script (as well as the entire reporting bundle) is built by Maven it is not part of the build system. It is simply a release artifact which knows how to build a coverage report for an instrumented deployment artifact.

```
<project>

  <property name="antjars.dir" value="${basedir}/antjars/" />

  <path id="antjars.classpath">
    <fileset dir="${antjars.dir}">
        <include name="*.jar" />
    </fileset>
  </path>
```

```
<taskdef classpathref="antjars.classpath" resource="tasks.properties" />

<target name="init">
  <property file="cobertura.properties"/>
  <property name="src.dir" location="${basedir}/sources"/>
  <property name="cobertura.dir" location="${basedir}/cobertura"/>
  <property name="merge.dir" location="${basedir}/merges"/>
  <property name="merged.ser" location="${merge.dir}/cobertura.ser"/>
  <property name="coveragereport.dir" location="${basedir}/report"/>

  <echoproperties>
    <propertyset>
      <propertyref prefix="coveragereport"/>
      <propertyref prefix="cobertura"/>
      <propertyref prefix="merge"/>
      <propertyref prefix="src.dir"/>
    </propertyset>
  </echoproperties>
</target>

<target name="create-merged-sers"
        depends="init">

  <cobertura-merge datafile="${merged.ser}">
    <fileset dir="${cobertura.dir}">
      <include name="*-cobertura.ser" />
    </fileset>
    <fileset file="${cobertura.ser.file}"/>
  </cobertura-merge>
</target>

<target name="create-report"
        description="Generate an html coverage report using the contents of the cobertura.se
        depends="init, clean, create-merged-sers">


 <cobertura-report
   format="html"
   destdir="${coveragereport.dir}"
   datafile="${merged.ser}">
   <fileset dir="${src.dir}/SomeProject-webapp">
      <include name="**/*.java" />
   </fileset>
   <fileset dir="${src.dir}/SomeProject-libA">
      <include name="**/*.java" />
   </fileset>
   <fileset dir="${src.dir}/SomeProject-libB">
      <include name="**/*.java" />
   </fileset>
 </cobertura-report>
</target>

<target name="clean-tomcat-data"
        description="Delete the cobertura.ser file created by tomcat"
        depends="init">
  <delete file="${cobertura.ser.file}"/>
</target>

<target name="clean"
        description="clean local build artifacts.  Leaves tomcat cobertura.ser file alone."
        depends="clean-report, clean-merges"/>

<target name="clean-report"
```

```
            description="Delete the html reports"
            depends="init">
      <delete dir="${coveragereport.dir}"/>
    </target>

    <target name="clean-merges"
            description="Delete the merged cobertura.ser file"
            depends="init">
      <delete dir="${merge.dir}"/>
    </target>
  </project>
```

Cobertura Ant Properties File Included in Assembly

This properties file will typically be found at src/main/coberturaScripts/cobertura.properties within the cobertura focused child module.

The properties here have intentionally been left outside of the cobertura.xml ant script so that the Maven assembly plugin can perform variable substitution if required.

```
#specify location of cobertura.ser file produced by the deployed artifact.
#cobertura.ser.file=${deployment.artifact.workingdir}/cobertura.ser
cobertura.ser.file=/opt/tomcat/bin/cobertura.ser
```

Deployment and Report Generation

Once the necessary Maven build changes described above have been made, you are ready to deploy the deployment artifacts and start making coverage measurements/reports.

The steps involved to use the tooling are as follows:

1. Run the Maven build with the cobertura-instrument profile engaged. prompt>mvn -Dcobertura-build clean deploy
2. Deploy the resulting deployment artifact. This typically involves simply copying a WAR file into the webapps directory of a J2EE container such as Tomcat.
3. Assuming you are deploying to a web container you will need to copy the cobertura-runtime jars into the appropriate location for common file. In the case of Tomcat you will simply copy the cobertura jar to $CATALINA_BASE/common/lib.
4. Assuming you are deploying to a web container you will also need to deploy the coberturaFlush.war into the web container. This war can be found in recent binary distributions of the cobertura project.
5. Start up the container (Tomcat, etc.).
6. Exercise the code for which you are measuring coverage. This will occassionally be a few simply manual tests and in others will involve launching some test infrastructure such as JMeter against the deployed infrastructure.
7. Shutdown the container and ensure the cobertura.ser file is generated when the shutdown hooks fire. By default the cobertura.ser file will be created in the working directory of the container.

   Assuming you have deployed to a web container which also has the coberturaFlush.war deployed you can instead call the coberturaFlush/flushCobertura URL to force a flush of cobertura.ser without having to shutdown the web container.

8. Expand the coberturaReporting assembly created by the Maven build in an appropriate work directory.
9. Ensure the cobertura.ser.file property in the cobertura.properties file correctly points to the cobertura.ser file produced by the deployment artifact. By default this is the working directory of the container. (If I start Tomcat from tomcat/bin the cobertura.ser file shows up in tomcat/bin.)
10. Run the cobertura.xml ant build.

    The relevant ant command to create the report is:

    ```
    >ant -f cobertura.xml create-report
    ```

11. Open the report/index.html file created by the ant script within the web browser of your choice and review the results.

Adaptations

The example above has largely focused on working with WAR deployment artifacts, but the principles remain the same regardless of what type of deployment artifact you are creating. For example in many deployment scenarios you will need to include the cobertura-runtime dependencies in your deployment artifact. As long as you keep the issues listed in the technical considerations section in mind, you should be able to easily work out an acceptable solution.

Debugging Hints:

- You can determine if an implementation class in your deployment artifact has been instrumented simply by running the javap tool against it. Have javap disassemble a random implementation class and look for the very obvious cobertura calls. (WARNING: Interface classes won't have what your looking for.)

  The relevant javap command is:

  ```
  >javap -c SomeImplementationClass
  ```