

JSFUnitWithMaven

JSFUnit can be run as part of your Maven build just like any JUnit test. In fact JSFUnit uses JSFUnit to test itself in this way. You can browse the test results of our nightly builds [here](#). Just go to the latest build under "Build History" and then look at the Console Output.

However, there are some changes that need to be made to your WAR to make this work. The current recommendation is that you take the same approach as JSFUnit itself. That is, have one Maven module for your application and another Maven module for JSFUnit. The JSFUnit module will simply declare your application as a dependency and make the needed tweaks to the WAR before running tests.

If you want to see a full-blown example you can [check out the full JSFUnit project](#) and you will see the `jboss-jsfunit-examples-hellojsf` module. Under that, there are four sub-modules. `jboss-jsfunit-examples-hellojsf-webapp` is a simple WAR application and the rest are JSFUnit test modules that depend on that WAR.

Steps to create a "JSFUnified" Maven WAR

Soon we will create tools that simplify the process, but until then, you can get this working with the steps below.

Step 1: Create a new submodule for your JSFUnit WAR

This submodule will use WAR overlays to create your "JSFUnified" WAR

Step 2: Add your Maven WAR as a dependency

```
<dependency>
<groupId>myGroupId</groupId>
<artifactId>myWARartifactId</artifactId>
<version>x.x.x</version>
```

```
<type>war</type>  
<scope>runtime</scope>  
</dependency>
```

Step 3: Add JSFUnit as a dependency

```
<dependency>  
  <groupId>org.jboss.jsfunit</groupId>  
  <artifactId>jboss-jsfunit-core</artifactId>  
  <version>1.1.0.GA</version>  
  <scope>compile</scope>  
</dependency>
```

Step 4: Override the web.xml file

Copy the web.xml file from your Maven WAR to the JSFUnit submodule. The new web.xml will be placed in:

```
$/src/main/webapp/WEB-INF
```

Step 5: Add the following to the new web.xml

```
<filter>  
  <filter-name>JSFUnitFilter</filter-name>  
  <filter-class>org.jboss.jsfunit.framework.JSFUnitFilter</filter-class>  
</filter>  
  
<filter-mapping>  
  <filter-name>JSFUnitFilter</filter-name>  
  <servlet-name>ServletTestRunner</servlet-name>  
</filter-mapping>  
  
<filter-mapping>  
  <filter-name>JSFUnitFilter</filter-name>  
  <servlet-name>ServletRedirector</servlet-name>  
</filter-mapping>  
  
<servlet>  
  <servlet-name>ServletRedirector</servlet-name>  
  <servlet-class>org.apache.cactus.server.ServletTestRedirector</servlet-class>  
</servlet>  
  
<servlet>  
  <servlet-name>ServletTestRunner</servlet-name>
```

```

    <servlet-class>org.apache.cactus.server.runner.ServletTestRunner</servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>ServletRedirector</servlet-name>
    <url-pattern>/ServletRedirector</url-pattern>
  </servlet-mapping>

  <servlet-mapping>
    <servlet-name>ServletTestRunner</servlet-name>
    <url-pattern>/ServletTestRunner</url-pattern>
  </servlet-mapping>

```

For simplicity, the servlet-name and filter-name should match the above exactly. Otherwise, you would be forced to do more configuration to override these default names.

Step 6: Configure the Surefire plugin to run your tests during a Maven build

```

<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-surefire-plugin</artifactId>

  <!-- We only want test to run during integration-test phase -->
  <configuration>
    <skip>true</skip>
  </configuration>

  <executions>
    <execution>
      <id>surefire-it</id>
      <phase>integration-test</phase>
      <goals>
        <goal>test</goal>
      </goals>
      <configuration>
        <skip>false</skip>
        <systemProperties>
          <property>
            <name>cactus.contextURL</name>
            <value>http://localhost:8080/${artifactId}</value>
          </property>
        </systemProperties>
      </configuration>
    </execution>
  </executions>
</plugin>

```

Step 7: Configure Cargo to start and stop your container for the tests. (This example uses the default container which is Jetty)

```

<plugin>
  <groupId>org.codehaus.cargo</groupId>
  <artifactId>cargo-maven2-plugin</artifactId>
  <configuration>
    <wait>false</wait>
    <configuration>
      <deployables>
        <deployable>
          <location>${project.build.directory}/${artifactId}.war</location>
          <type>war</type>
        </deployable>
      </deployables>
    </configuration>
  </configuration>
  <executions>
    <execution>
      <id>start-container</id>
      <phase>pre-integration-test</phase>
      <goals>
        <goal>start</goal>
      </goals>
    </execution>
    <execution>
      <id>stop-container</id>
      <phase>post-integration-test</phase>
      <goals>
        <goal>stop</goal>
      </goals>
    </execution>
  </executions>
</plugin>

```

Step 8: Set the test and source directory to be the same

This way, the tests will be bundled with your WAR.

```

<build>
  <sourceDirectory>src/test/java</sourceDirectory>
  <testSourceDirectory>src/test/java</testSourceDirectory>
</build>

```

Also See

- [Maven Surefire Plugin](#)
- [Maven WAR Plugin](#)