Configuration files option

Definition

The configuration files are the files to add to your container's configuration. It is internally used by some containers and is accessible for you can to add extra files.

> This feature is only available for local containers (i.e., not for remote containers nor any kind of deployers).

Explanation

In some cases, it is necessary to enrich your container configuration with extra files. This can be done using Cargo's `configFiles` option, accessible via the Cargo APIs.

The advantage of using this option is that it can replace configuration properties in your files: simply use, for example, `@cargo.servlet.port@` to have it replaced with the port on which the Servlet/JSP container is listening to.

> **Careful with binary files!**
>
> If you want to inject binary configuration files (JAR files, for example), you should use `setFileProperty` (instead of `setConfigFileProperty`) with the Java API and `<files>` and `<file>` (instead of `<configfiles>` and `<configfile>`) with the Maven2/Maven3 plugin.

You might for example want to add the advanced login configuration to your JBoss instance.

Example using the Java API

> The method's name is `setConfigFileProperty` rather than `addConfigFileProperty`; but it actually adds files to the `configFiles` list.

```
LocalContainer container = ...;
StandaloneLocalConfiguration configuration =
(StandaloneLocalConfiguration)
getLocalContainer().getConfiguration();

FileConfig loginConfigXml = new
FileConfig();
loginConfigXml.setConfigfile("src/main/jboss
5/login-config.xml");
loginConfigXml.setToDir("conf");
configuration.setConfigFileProperty(loginCon
figXml);

FileConfig sampleRolesProperties = new
FileConfig();
sampleRolesProperties.setConfigfile("src/mai
n/jboss5/sample-roles.properties");
sampleRolesProperties.setToDir("conf");
configuration.setConfigFileProperty(sampleRo
lesProperties);

FileConfig sampleUsersProperties = new
FileConfig();
sampleUsersProperties.setConfigfile("src/mai
n/jboss5/sample-users.properties");
sampleUsersProperties.setToDir("conf");
configuration.setConfigFileProperty(sampleUs
ersProperties);
```

Example using the ANT tasks

```
<cargo containerId="@{containerId}"
action="@{action}">
  <configuration
home="${configuration.home}">
    <configfile
file="${basedir}/src/main/jboss5/login-confi
g.xml" todir="conf"/>
    <configfile
file="${basedir}/src/main/jboss5/sample-role
s.properties" todir="conf/props"/>
    <configfile
file="${basedir}/src/main/jboss5/sample-user
s.properties" todir="conf/props"/>
  </configuration>
</cargo>
```

Example using the Maven 2 plugin

```
<plugin>
  <groupId>org.codehaus.cargo</groupId>

<artifactId>cargo-maven2-plugin</artifactId>
  <configuration>
    <container>
      [...]
    </container>
    <configuration>
      <type>standalone</type>
      [...]
```

```
      <configfiles>
        <configfile>

<file>${project.basedir}/src/main/jboss5/log
in-config.xml</file>
          <todir>conf</todir>
        </configfile>
        <configfile>

<file>${project.basedir}/src/main/jboss5/sam
ple-roles.properties</file>
          <todir>conf/props</todir>
        </configfile>
        <configfile>

<file>${project.basedir}/src/main/jboss5/sam
ple-users.properties</file>
          <todir>conf/props</todir>
        </configfile>
      </configfiles>
```

```
    </configuration>
  </configuration>
</plugin>
```

More advanced example: adding the jetty-env.xml file

To define a central `jetty-env.xml` file and reuse it later from your WARs, you can proceed as follows:

1. As explained on the top of this page, use a standalone local container (i.e., not an embedded configuration)
2. Make sure the property `JettyPropertySet.CREATE_CONTEXT_XML` (i.e. `cargo.jetty.createContextXml`) is set to `false`; so that CARGO doesn't create a `context.xml` file for your WAR's deployable but rather copies it from the source to the configuration home folder.
3. For each deployable, make sure the deployable is defined as an expanded WAR; as explained on the Static deployment of expanded WAR page.
4. Finally, for each deployable, add the following to your local container's configuration:

   ```
   <configfiles>
     <configfile>
       <file>${basedir}/src/main/config/jetty-env.xml</file>
       <todir>webapps/webapp-context/WEB-INF</todir>
     </configfile>
   </configfiles>
   ```

   ... where `webapp-context` is the Web application's context

This way:

- To deploy your WAR, CARGO will copy your WAR's expanded version (i.e., the WAR directory) into the `webapps` subdirectory of the Jetty configuration
- Once the WAR is copied, CARGO will also copy the `jetty-env.xml` file into that WAR's WEB-INF directory

As a reference, please find below the full configuration for Jetty 7.x:

```
<configuration>
  <container>
    <containerId>jetty7x</containerId>
    <zipUrlInstaller>
      <url>

http://download.eclipse.org/jetty/7.6.3.v201
```

```
20416/dist/jetty-distribution-7.6.3.v20120416.zip
      </url>
    </zipUrlInstaller>
  </container>
  <configuration>

<home>${project.build.directory}/jetty-home</home>
    <properties>

<cargo.jetty.createContextXml>false</cargo.jetty.createContextXml>
    </properties>
    <configfiles>
      <configfile>

<file>${project.basedir}/src/main/config/jetty-env.xml</file>

<todir>webapps/datasource-war/WEB-INF</todir>
      </configfile>
    </configfiles>
  </configuration>
  <deployables>
    <deployable>
      <groupId>org.codehaus.cargo</groupId>

<artifactId>datasource-war</artifactId>
      <type>war</type>
```

```
<location>datasource-war-extracted</location
>
      <properties>
        <context>datasource-war</context>
      </properties>
```

```
    </deployable>
  </deployables>
</configuration>
```