

Ant support

Definition

Cargo provides Ant tasks to perform all the operations available from the Java API

Functional tests

The usage of Cargo for executing functional tests on a container does not mandate these ANT tasks. You could directly use the Cargo Java API from your Java unit test classes (JUnit, TestNG, etc), as described on the [Functional testing](#) page.

Explanation

Before using the Ant API you need to register the Cargo Ant tasks into Ant. This is done in the following manner:

```
<taskdef resource="cargo.tasks">
  <classpath>
    <pathelement
location="$ {cargo-core-uberjar.jar}" />
    <pathelement
location="$ {cargo-ant.jar}" />
  </classpath>
</taskdef>
```

Some additional dependencies might also be required for the ANT task. Please see the [Installation](#) page for details.

The Cargo ANT tasks in detail

Here are the different task actions available to call on this plugin:

Action	Description
--------	-------------

<p>start</p>	<p>Start a container. That task will:</p> <ul style="list-style-type: none"> • If the task configuration requires so, installs the container. • If the task configuration defines a container with a standalone local configuration, it will create the configuration. • If the task configuration contains one or more deployables, it will deploy these to the container automatically. • And, of course, start the container. <p>Note: A container that's started with the <code>start</code> task will automatically shut down as soon as the parent ANT instance quits (i.e., you see a <code>BUILD SUCCESSFUL</code> or <code>BUILD FAILED</code> message). If you want to start a container and perform manual testing, see our next task <code>run</code>.</p>
<p>run</p>	<p>Start a container and wait for the user to press <code>CTRL + C</code> to stop. That task will:</p> <ul style="list-style-type: none"> • If the task configuration requires so, installs the container. • If the task configuration defines a container with a standalone local configuration, it will create the configuration. • If the task configuration contains one or more deployables, it will deploy these to the container automatically. • And, of course, start the container and wait for the user to press <code>CTRL + C</code> to stop.
<p>stop</p>	<p>Stop a container.</p>
<p>restart</p>	<p>Stop and start again a container. If the container was not running before calling <code>restart</code>, it will simply be started.</p>
<p>configure</p>	<p>Create the configuration for a local container, without starting it. Note that the <code>start</code> and <code>run</code> actions will also install the container automatically.</p>
<p>deploy</p>	<p>Deploy a deployable to a running container.</p>
<p>undeploy</p>	<p>Undeploy a deployable from a running container.</p>
<p>redploy</p>	<p>Undeploy and deploy again a deployable. If the deployable was not deployed before calling <code>redploy</code>, it will simply be deployed.</p>

Wait after the container has started

Many wonder the difference between the `start` and `run` actions:

- If you want to just start the container and then do other tasks (for example, execute tests), use the `start` action. That action should therefore ONLY be used for integration testing.
- If you want start the container and have ANT "blocked" afterwards (i.e., until you press `CTRL + C` to stop), use the `run` action. `run` is therefore the action to use for manual testing.

Examples

Orion 2.x

Here's a full example showing how to deploy a WAR, and expanded WAR and an EAR in an Orion 2.x container. Please note that the `output` and `log` attribute are optional. The `property` elements allow you to tune how the container is configured. Here we're telling it to start on port 8180 and to generate the maximum amount of logs in the container `output` file.

```

<taskdef resource="cargo.tasks">
  <classpath>
    <pathelement
location="path/to/cargo-uberjar.jar"/>
    <pathelement
location="path/to/cargo-ant-tasks.jar"/>
  </classpath>
</taskdef>

<cargo containerId="orion2x"
home="c:/apps/orion-2.0.3"
output="target/output.log"
  log="target/cargo.log" action="start">
  <configuration>
    <property name="cargo.servlet.port"
value="8180"/>
    <property name="cargo.logging"
value="high"/>
    <deployable type="war"
file="path/to/my/simple.war"/>
    <deployable type="war"
file="path/to/my/expandedwar/simple"/>
    <deployable type="ear"
file="path/to/my/simple.ear"/>
  </configuration>
</cargo>

```

Tomcat 5.x

This example gives a walk through of how to get a Cargo Ant build to work with Tomcat 5.x .

Prerequisites

- It is assumed that [Tomcat 5.x](#) is already installed
- The cargo-core-uberjar.jar and cargo-ant.jar JARs have been downloaded
- A minimum knowledge of Ant is required
- User already has a war target that properly generates a working war file

Steps

Follow the following steps to configure your build.xml :

- Create a folder under your basedir called cargolib that will hold cargo-core-uberjar.jar and cargo-ant.jar
- Define a property for cargolib

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

- Define 2 new properties cargo-uberjar and cargo-antjar as shown below:

```
<property name="cargo-uberjar"
value="${cargolib.dir}/cargo-core-uberjar.jar"/>
<property name="cargo-antjar"
value="${cargolib.dir}/cargo-ant.jar"/>
```

- Add additional properties for defining the following:

Property	Description
tomcat.home	Installation directory of tomcat5x
tomcatlog.dir	This is where our logs are going to be generated
tomcatconfig.dir	Cargo needs an empty config folder
pathtowarfile	The full path of the war file e.g c:/devtools/myapp/dist/myfile.war

- Add the following code to your build.xml :

```

<taskdef resource="cargo.tasks">
  <classpath>
    <pathelement location="${cargo-uberjar}"/>
    <pathelement location="${cargo-antjar}"/>
  </classpath>
</taskdef>

<target name="cargostart" depends="war">
  <delete dir="${tomcatconfig.dir}" />
  <mkdir dir="${tomcatlog.dir}"/>
  <mkdir dir="${tomcatconfig.dir}"/>
  <echo message="Starting Cargo..." />
  <echo message="Using tomcat.home = ${tomcat.home}" />
  <echo message="Using war = ${mywarfile}" />
  <echo message="Jars used = ${cargo-uberjar} , ${cargo-antjar}"/>

  <cargo containerId="tomcat5x" home="${tomcat.home}"
output="${tomcatlog.dir}/output.log"
  log="${tomcatlog.dir}/cargo.log" action="start">
    <configuration home="${tomcatconfig.dir}">
      <property name="cargo.servlet.port" value="8080"/>
      <property name="cargo.logging" value="high"/>
      <deployable type="war" file="${mywarfile}"/>
    </configuration>
  </cargo>

</target>

```

Remote deployment

Here's a full example showing how to deploy a WAR to a remote Tomcat 6.x container.

```

<taskdef resource="cargo.tasks">
  <classpath>
    <pathelement
location="path/to/cargo-uberjar.jar"/>
    <pathelement
location="path/to/cargo-ant-tasks.jar"/>
  </classpath>
</taskdef>

<cargo containerId="tomcat6x"
action="deploy" type="remote">
  <configuration type="runtime">
    <property name="cargo.hostname"
value="production27"/>
    <property name="cargo.servlet.port"
value="8080"/>
    <property name="cargo.remote.username"
value="admin"/>
    <property name="cargo.remote.password"
value=""/>
    <deployable type="war"
file="path/to/simple-war.war">
      <property name="context"
value="application-context"/>
    </deployable>
  </configuration>
</cargo>

```

For more details, please check the example in the [Remote Container](#) section for the ANT tasks. The ANT tasks support the deployer actions `deploy`, `undeploy` and `redeploy`.

