CodeCharge Studio Java Deployment Guide

Table of contents

CodeCharge Studio requirements for Java deployment

Environment:

- JDK 1.3 or 1.4 (JDK Home)
- Servlet 2.2 and JSP 1.1 (Servlet Home)
- Ant 1.4 (Ant Home)

Setup:

- JAVA_HOME environment variable should indicate path to JDK that should be used to build project.
- ANT_HOME environment variable or Ant Home site property should indicate path to Ant build tool installation directory.

Optional:

- JDBC 2.0 Optional package (JDBC Home) if you choose to use DataSource extension
- JAXP 1.1 and SAX 2.0 if you choose to generate JSP (Apache XML) (Sun XML)
- Jakarta ORO if you want regexp validations for e-mail, phone, zip code, etc...

Class Path requirements (compilation-time and run-time)

JSP

Class Path should contain:

- Servlet 2.2 API library
- JAXP 1.1 library
- SAX 2.0 parser (such as crimson or xerces)
- JDBC 2.0 extensions library if you use Data Sources
- Jakarta ORO if you use regexp for validations

Servlets with Templates

Class Path should contain:

- Servlet 2.2 API library
- JDBC 2.0 extensions library if you want to use Data Sources
- Jakarta ORO if you use regexp for validations

Tomcat 4.0 deployment

Check that the DB drivers are in the class path of your server. Choose from one of the following scenarios:

Scenario 1
Auto-deployment from webapps directory (DataSource extension is not used)

Before generating the code, ensure that the “Use Data Source” site property is set to “No”. This is in the Project Explorer window.
Fill in connection properties to indicate Driver class name, Database url, user login and password.
Ensure that your database driver is in your application server class path.
Place the generated war file at the location where the appBase attribute of the Virtual Host configuration element points. Usually, it is <TOMCAT_HOME>/webapps.

Scenario 2
Deployment through Context configuration (DataSource extension is not used)

Add Context element as a child to Host element in your server.xml configuration file.

```
<Context path="/JSPPortal" docBase="JSPPortal.war">
</Context>
```

The “path” is the context path of your application.
“docBase” is the absolute path to the war file, or relative to the appBase path.

Scenario 3
Deployment through Context configuration (DataSource extension is used)

Before code generation, verify that the “Use Data Source” site property is set to “Yes”.
Set the connection “DataBase URL” property to “java:comp/env/jdbc/Connection1” where “Connection1” is the name of your connection.
Define Context element as described in the previous scenario.
Add a Resource child element with ResourceParams to the Context element:

```
<Resource name="jdbc/Connection1" auth="Container"
type="javax.sql.DataSource"/>
  <ResourceParams name="jdbc/Connection1">
    <parameter>
      <name>user</name>

      <value>User Name</value>
    </parameter>
    <parameter>
      <name>password</name>
      <value>User Password</value>

    </parameter>
    <parameter>
      <name>driverClassName</name>
      <value>Driver Class Name</value>
    </parameter>

    <parameter>
      <name>driverName</name>
      <value>Database Connection URL</value>
    </parameter>
  </ResourceParams>
```

name is Connection name as indicated in your CCS project with **jdbc/** prefix prepended.

Tomcat 4.0 + Apache deployment

Scenario 1. MOD_JK connector.

1. Download **mod_jk.dll** and **libapr.dll** and copy them to the **apache/modules** directory.
2. Create **workers.properties** file in the **$CATALINA_HOME/conf/jk/** directory.

```
workers.tomcat_home=$(CATALINA_HOME)
workers.java_home=$(JAVA_HOME)
ps=\
worker.list=ajp13
worker.ajp13.port=8009
worker.ajp13.host=localhost
worker.ajp13.type=ajp13
```

3. At the end of httpd.conf LoadModule section add the following line:

```
LoadModule jk_module modules/mod_jk.dll
```

4. At the end of httpd.conf AddModule section add the following line:

```
AddModule mod_jk.c
```

5. At the end of httpd.conf add the following:

```
JkWorkersFile "$CATALINA_HOME/conf/jk/workers.properties"
JkLogFile "$CATALINA_HOME/logs/mod_jk.log"
JkLogLevel info
<VirtualHost localhost>
    ServerName localhost
    DocumentRoot $CATALINA_HOME/webapps/
    JkMount /*.jsp ajp13
    JkMount /*.do ajp13
</VirtualHost>
```

6. If needed uncomment the following declarations in Tomcat server.xml config file:

```
<Connector className="org.apache.ajp.tomcat4.Ajp13Connector"
port="8009" minProcessors="5" maxProcessors="75"
acceptCount="10" debug="0"/>
```

7. In CCS set publishing path to $CATALINA_HOME/webapps/ and publish your project.
8. Start Tomcat and then Apache.

Scenario 2. WARP connector.

1. Download **mod_webapp.so** and **libapr.dll** and copy them to the **apache/modules** directory.
2. At the end of httpd.conf LoadModule section add the following line:

```
LoadModule webapp_module modules/mod_webapp.so
```

3. At the end of httpd.conf AddModule section add the following line:

```
AddModule mod_webapp.c
```

4. At the end of httpd.conf add the following:

```
WebAppConnection warpConnection warp localhost:8008
WebAppDeploy JSPPortal warpConnection /Portal/
```

5. If needed uncomment Tomcat-Apache service configuration in Tomcat server.xml config file.
6. In CCS set publishing path to $CATALINA_HOME/webapps/ and publish your project.
7. Start Tomcat and then Apache.

Resin 2.0 deployment

Check that DB drivers are in class path of your server. Choose the appropriate scenario below.

Scenario 1
Auto-deployment from webapps directory (DataSource extension is not used)

Before code generation, verify that the “Use Data Source” site property is set to “No”.
Fill in connection properties to indicate Driver class name, Database url, user login and password.
Ensure that your database driver is in your application server class path.
Place the generated war file at the location where the war-dir element of the Virtual Host configuration element points. Usually, it is <RESIN_HOME>/webapps.

Scenario 2
Deployment through web-app configuration (DataSource extension is not used)

Add Context element as a child to Host element in your resin.conf configuration file.

```
<web-app id="/JSPPortal"
 app-dir="c:/projects/Portal/JSPPortal.war"/>
```

The id attribute is the context path of your application while app-dir is the absolute or relative path to the war file from the app-dir.

Scenario 3
Deployment through web-app configuration (DataSource extension is used)

Before code generation, verify that the “Use Data Source” site property is set to “Yes”.
Set the connection “DataBase URL” property to “java:comp/env/jdbc/Connection1”, where Connection1 is name of your connection.
Add the resource-ref element as the child of the web-app element:

```
<resource-ref>

  <res-ref-name>jdbc/Connection1</res-ref-name>
  <res-type>javax.sql.DataSource</res-type>
  <init-param driver-name="Driver Class Name"/>
  <init-param url="Database connection URL"/>

  <init-param user="User Name"/>
  <init-param password="User Password"/>
  <init-param max-connections="20"/>
  <init-param max-idle-time="30"/>
  <init-param enable-transaction="false"/>

</resource-ref>
```

The res-ref-name attribute is the Connection name as indicated in your project with the “jdbc/” prefix prepended

WebLogic 7.0 deployment

Check that DB drivers are in class path of your server. Choose the appropriate scenario below.

Scenario 1
Auto-deployment from applications directory (DataSource extention is not used)

Before code generation, verify that the “Use Data Source” site property is set to “No”.
Fill in connection properties to indicate Driver class name, Database url, user login and password.
You can quickly deploy your application on the administration server if auto-deploy is enabled by copying the war file into the \applications directory of the administration server.

Scenario 2
Deployment through Administration Console

Set the “**Use Data Source**” site property appropriately to indicate whether you want to use a Data Sources in your code.
If you want to work with Data sources and you have not configured it already, add and configure the JDBC Pool and DataSource to your server as described in (http://e-docs.bea.com/wls/docs70/ConsoleHelp/jdbc.html#1104939). Bear in mind that the WebLogic JNDIName property should be equal to the “**DataBase URL**” property of your CodeCharge Studio connection. Install your application as described in (http://e-docs.bea.com/wls/docs70/ConsoleHelp/applications.html#1104914).

WebLogic 5.1 deployment (JSP)

WebLogic 5.1 requires SAX 2.0 and the JAXP 1.1 libraries in order to work with the JSP generated by CodeCharge Studio. WebLogic has its own SAX and JAXP classes located in *lib/weblogicaux.jar*. Your xml libraries must be referenced earlier in the class path before weblogicaux.jar. To accomplish this, we recommend that you prepend the classpath with the xml libaries using the wlconfig.exe utility located in the *weblogic/bin* directory. An example command would be:

wlconfig.exe -classpath c:\java\xerces-2_0_1\xercesImpl.jar;C:\java\xerces-2_0_1\xmlParserAPIs.jar

This will reference the SAX and JAXP classes before your JVM class path. You will also need to restart WebLogic before your new class path can be used.

WebSphere 4.0 deployment

Begin by clicking on the Project name in the Project Explorer window then under the Data tab of the Properties window, configure the Site properties including the Server side connection properties. Make sure that JDBC drivers are in the application class path.

1. Get the latest version of JAXP 1.1 and SAX 2.0 for compilation and runtime execution of JSP files. You can get them from the Ant installation.
2. For the 'Runtime libs' Site property, enter in semicolon-delimited format the absolute paths to the jar files corresponding to JAXP and SAX e.g. "c:\ant\lib\jaxp.jar;c:\ant\lib\crimson.jar". These files will be copied to the WEB-INF\lib directory of the application.
3. Select 'WebSphere 4.0' in 'Target server' Site property.
4. Specify 'Publishing/Server Path' property and publish your project.
5. Use the WebSphere console to install your application. On the second screen of the Application Installation Wizard select 'No' for the 'Precompile JSPs' property of your web module.
6. Start the new application from WebSphere Admin console (if needed, restart the server).

Project folder structure

The structure of a generated Java project is as follows:

```
/YourProject
    - /CCSBuild
        - /src - all java source code
        - /config - web.xml, CCStags.tld, properties etc.
        - build.xml - makefile for Ant
    - *.jsp
    - *.xml
    - *.ccp
    - *.html
    - YourProject.ccs
    - /images
    - /Themes
    - /Manually created folders
```

The Ant tool can create four more folders relative to the CCSBuild folder, based on the target you build:

```
- /app     - application ready to be jared or copied to the directory
             configured as context root in application server.
- /appWar - war file for your application
- /bin    - all compiled classes
- /doc    - javadoc
```

WAR file vs. application folder vs. plain JSP

CodeCharge Studio always creates and publishes WAR file, which is a standard method for all servers. To work with the un-archived version of the application you can perform the following steps:

1. Right click on your Project in Project Explorer and select **Generate Code**
2. Save All
3. Got to the directory **YourProject/CCSBuild**
4. Run Ant. It will find the **build.xml** file then compile and build your project.
5. You will then find your project ready to be configured in the **YourProject/CCSBuild/app** folder.

You may also need to edit the application properties in your servers config file, such as context path etc.

Class Path

The Class Path for your application can be specified using the **Class Path** property of the Site. This should be a list of directories and jar files separated by semicolon, and will be added to classpath at the time of compilation.
For example:

c:\java\xerces\xercesImpl.jar;c:\java\xerces\xmlParserAPIs.jar;c:\java\lib\Servlet.jar

Note that to set classpath for run-time, you will need to configure your application server.

How Controller Servlet finds Action classes

ControllerServlet serves all *.do HTTP queries. This is defined in web.xml file of the application:

```
<servlet>
    <servlet-name>controller</servlet-name>
    <servlet-class>com.codecharge.ControllerServlet</servlet-class>
</servlet>

<servlet-mapping>
    <servlet-name>controller</servlet-name>
    <url-pattern>/*.do</url-pattern>
</servlet-mapping>
```

After receiving query, ControllerServlet needs to determine from query which Action class to call. Correspondence between page URL and Action classes is of the way:

- all folders in application context correspond to packages names
- Page name corresponds to Class name (without .do)

For example the URL

http://server:port/AppName/FolderName/PageName.do?parameters

corresponds to FolderName.PageName class (PageName class in FolderName package) in AppName application.

Troubleshooting - JSP deployment

WAR file is not being created

Symptom:
WAR file is absent in target folder and the following error message appears in CodeCharge Studio's message window:

```
[copy] Could not find file
C:\WINNT\Profiles\ADMINI~1\LOCALS~1\Temp\~aim0\CCSBuild\config\CCStags.tld to
copy
```

Reason:
Wrong publishing settings prevent the file CCStags.tld from being copied to the temporary directory for publishing.

Solution:
In Project -> Settings -> Publishing change one of the following:
1. Check Files to Publish -> **All files with extensions**. And add **tld** extension to the list.
or
2. Check Files to Publish -> **All files excluding extensions**