CHAPTER 4

Understanding the CWCS Execution Environment

The following topics describe the CWCS execution environment and provide guidelines for creating applications to run in this environment:

- Understanding the Java Application Launch Process
- Launching a Java Application
- Using Servlets and JSPs with Tomcat
- Using JavaBeans

Note As of this release, CWCS has dropped all support for Macromedia JRun/JSP. JRun is no longer included in any version of the CWCS distribution.

Understanding the Java Application Launch Process

The CWCS Java runtime environment is similar to the standard Java2 environment, with the following exceptions:

- The bootstrap class path contains a Visibroker ORB implementation followed by standard JRE2 classes. This ensures that the VisBroker classes will be used to implement ORB.
- The com.cisco.nm.cw2000.home system property is defined and points to the location of the CWCS installation root.
- The default user class path contains the CWCS server class hierarchy (lib/classpath) and the CWCS client class hierarchy (www/classpath).
- The current working directory is set to the CWCS installation root.
- On UNIX platforms, the path for JNI code (the value of the *LD_LIBRARY_PATH* variable for Solaris, and so on) contains the CWCS shared objects directory library.
- On Windows platforms, it is assumed that cwjava resides in the same directory as the CWCS JNI DLLs.
- CWCS supports two JRE versions on the server side: 1.3.1_06 and 1.4.2_10. Version 1.4.2_10 is available under *NMSROOT*/lib/jre. Version 1.3.1_06 is available under *NMSROOT*/MDC/jre.

When cwjava launches a Java application in the CWCS runtime environment, it performs these functions:

1. Sets CW2000 to the CWCS installation directory. The default value is the value of the *NMSROOT* environment variable. To override the default, use the **-cw** option (see the "Launching a Java Application" section on page 4-2).
2. Sets JRE to the Java Runtime Environment root directory. By default, it is *NMSROOT*/MDC/jre (which means cwjava uses JRE 1.3.1_06 by default). To override the default, use the **-cw:jre** option.
3. Sets VB to the VisiBroker ORB path. By default, the path is

   `CW2000/www/classpath/vbjapp.jar:CW2000/www/classpath/vbjorb.jar.`

   To override the default, use the **-cw:vb** option.
4. Sets the value of com.cisco.nm.cw2000.home system property to CW2000.
5. Sets the current directory to the value of the **-wd** option. If **-wd** is omitted, it sets the current directory to CW2000.
6. Processes the rest of the options as standard Java and launches the interpreter.
7. The following ORB options are automatically added by cwjava. You need not specify these in the command line.
   - -Dorg.omg.CORBA.ORBClass=com.inprise.vbroker.orb.ORB
   - -Dorg.omg.CORBA.ORBSingletonClass=com.inprise.vbroker.orb.ORBSingleton
   - -Djavax.rmi.CORBA.StubClass=com.inprise.vbroker.rmi.CORBA.StubImpl
   - -Djavax.rmi.CORBA.UtilClass=com.inprise.vbroker.rmi.CORBA.UtilImpl
   - -Djavax.rmi.PortableRemoteObjectClass=com.inprise.vbroker.rmi.PortableRemoteObjectImpl

Launching a Java Application

Use this syntax to launch a Java application:

cwjava [options] class [argument ...]

cwjava [options] –jar file.jar [argument ...]

If the **-jar** option is specified, the first non-option argument is the name of the JAR archive containing class and resource files for the application. The startup class is indicated by the Main-Class manifest header.

Table 4-1 summarizes the command line options that modify the runtime environment.

Table 4-1 cwjava Command Line Options

Option	Description
-cw *dir*	Sets the CiscoWorks installation directory to *dir*.
-cw:jre *dir*	Sets the JRE directory root to dir. This directory should contain the standard hierarchy of JRE files—the interpreter in the dir/bin and standard classes in the dir/lib. If the path name is not absolute, it is relative to the CWCS Server installation directory. For example: `cwjava -cw:jre NMSROOT/MDC/jre`.
-cw:vb *vbfile*	The path containing VisiBroker library. Elements can be directories or JAR files. All non-absolute elements are relative to the CWCS Server installation directory.
-cw:nojit -cw:jit	Disables/enables the just-in-time (JIT) compiler. Unless the **-Xdebug** option is present, the JIT compiler is enabled by default.
-cw:xrs	This new option is applicable on Windows platforms only. It forces CWCS to ignore the CTRL_LOGOFF_EVENT *only*. Use this option if: • Your application specifies JRE 1.3 or above, via the -cw:jre option. • The application runs as a Windows service. • The service aborts when a Windows user on the CiscoWorks Server logs off from the machine. This option is a subset of the Java -**Xrs** option in its current definition. Use it instead of the Java -Xrs option. If you use Java -Xrs, it will ignore other events and signals in addition to CTRL_LOGOFF_EVENT. This will cause Ctrl-Break thread dumps to be unavailable, and will force your code to be responsible for causing shutdown hooks to run.
-wd *workingdir*	Sets the current directory to *workingdir* before launching the interpreter. All non-absolute elements are relative to the CWCS Server installation directory.
-cp:a *classpath* **-cp:p** *classpath*	Prepends/appends classpath to the class path. All non-absolute elements are relative to the CWCS Server installation directory. These two options cannot be specified at the same time.
-cp:amf *manifest_file* **-cp:pmf** *manifest_file*	Prepends/appends Manifest file containing directories, zip or jar files to the class path. All non-absolute elements are relative to the CWCS Server installation directory. These two options cannot be specified at the same time.
-classpath *classpath* **-cp** *classpath*	Sets class path to *classpath*. All non-absolute elements are relative to the CWCS Server installation directory.
-? -help	Displays help information and exits.
-version	Displays version information and exits.

Table 4-2 summarizes the options that cwjava processes in the same manner as the Java tool from JRE2.

Table 4-2 cwjava JRE2 Options

Option	Description
-D*property*=*value*	Sets a system property value.
-jar	Runs a program encapsulated in a JAR file. The first argument is the name of a 'JAR file instead of a startup class name. For this option to work, the manifest of the JAR file must contain a line of the form Main-Class: *classname*. Here, *classname* identifies the class that contains this method, which serves as your application's starting point: `public static void main(String[] args)` When you use this option, the JAR file is the source of all user classes, and other user class path settings are ignored.
-verbose **-verbose:class**	Displays information about each class that is loaded.
-verbose:gc	Reports on each garbage collection event.
-verbose:jni	Reports information about the use of native methods and other Java Native Interface activity.
-Xbootclasspath:*bootclasspath*	Specifies a list of directories, JAR archives, and ZIP archives to search for boot class files. These are used in place of the boot class files included in the JDK 1.2 software. *Use with care.*
-Xdebug	Starts with the debugger enabled. This option automatically disables JIT. The Java interpreter prints out a password for jdb's use.
-Xnoclassgc	Disables class garbage collection.
-Xms*n*	Specifies the initial size of the memory allocation pool. This value must greater than 1000: • To multiply the value by 1000, append the letter k. • To multiply the value by 1 million, append the letter m. The default value is 1m.
-Xmx*n*	Specifies the maximum size of the memory allocation pool. This value must greater than 1000: • To multiply the value by 1000, append the letter k. • To multiply the value by 1 million, append the letter m. The default value is 64m.
-Xrunhprof{:help | :*suboption*=*value*,...**}**	Enables CPU, heap, or monitor profiling. This option is typically followed by a list of comma-separated *suboption*=*value* pairs. Run this command to obtain a list of suboptions and their default values: **`cwjava -Xrunhprof:help`**
-Xrs	Reduces the use of operating system signals.
-Xcheck:jni	Performs additional checks for Java Native Interface (JNI) functions.
-X	Prints help on java non-standard options

Using Servlets and JSPs with Tomcat

This version of CWCS ships with Tomcat 4.1.29. This version of Tomcat contains a reference implementation of the Servlet 2.3 and JSP 1.2 specifications. To execute your Servlets and JSP under the Tomcat Servlet Engine, please observe the following guidelines.

- There are no special procedures needed to get Tomcat enabled and working with Apache. CWCS installs and configures Tomcat and Apache for you.
- Remember to include the package statement in the servlet code.
- Before executing your servlets and JSPs, you must register your Servlet Context with the Tomcat Servlet Engine. CWCS provides the UpdateTomcatMain API to register your servlet context with Tomcat. During installation, call this API in your package's post-install code. Calls to the registration API have the following form:

  ```
  UpdateTomcatMain(UrlPattern, RelPath, Mapping, AppID, IsUninstall);
  ```

  Where:

  - *URlPattern* is the name of your Servlet Context (for example: "MDC/Servlet"). To refer to "SampleServlet" in this context, you would call
    `http://server-name:port/MDC/Servlet/SampleServlet`
  - *RelPath* is the document base for this Servlet Context. The path should be a relative path with respect to the *NMSROOT*/MDC/tomcat directory (for example: "MDC/maas/servlet").
  - *Mapping* is normally an empty string (for example: "").
  - *AppID* is the name of the application using this context (for example: "core").
  - *IsUninstall* is a flag indicating whether to add or remove the servlet context. To add the servlet context, set the value to "false". To remove it, set the value to "true".

  For example:

  To register a typical Servlet Context on Windows:

  ```
  UpdateTomcatMain("/MAAS/JSP", "vms/maas/JSP", "\"\"", "maas-jsp", "false");
  ```

  To do the same thing on Solaris:

  ```
  UpdateTomcatMain "/MAAS/JSP" "vms/maas/JSP" $EMPTY_STRING "maas-jsp" "false" ;
  ```

Using JavaBeans

For JavaBean classes used in a JSP file that runs within CWCS, follow these guidelines::

- Place your JSP files in your webapp directory. The webapp directory needs to be specified while adding your Servlet Context. The directory is the same as the *RelPath* value specified when you call the UpdateTomcatMain API during installation (see the "Using Servlets and JSPs with Tomcat" section on page 4-5). The *RelPath* value is a relative path; it must be converted into an absolute path while placing the JSP.
- If your application is using the Struts framework or the User Interface Infrastructure (UII), then JSP files must be placed according to the framework specification.

- Place your JavaBean class file under the $WebAppDir/WEB-INF/classes directory. If your JavaBean is in a JAR file, place it under the $WebAppDir/WEB-INF/lib directory.
- In **jsp:usebean**, include the fully qualified classname. For example:

```
<jsp:useBean id= 'myBean' class='com.cisco.nm.example.Testbean' scope='page'/>
```