

---

# **AOLserver Tomcat Plug-in**

## **User Guide**

**March 2004**

**Release: 1.3**

---

# Table of Contents

1	History .....	4
2	Basic Concepts .....	5
2.1	Jakarta Connectors .....	5
2.2	Notation .....	5
3	Installation .....	7
3.1	Supported Platforms .....	7
3.2	Prerequisites for Installation .....	7
3.2.1	AOLserver .....	7
3.2.2	JDK/JVM .....	7
3.2.3	Tomcat 4 .....	7
3.2.4	Jakarta Ant .....	8
3.2.5	Plug-in Binary .....	8
3.3	Using the Ant-based Installer .....	8
3.3.1	Assumptions .....	8
3.3.2	Installation Steps .....	8
3.4	Manual Installation .....	9
3.4.1	Copy Libraries .....	9
3.4.2	Tomcat In-process .....	9
3.4.3	Tomcat out-of-process .....	12
4	Java-TCL Bridge .....	15
4.1	Assumptions .....	15
4.2	Installation .....	15
4.3	Usage Examples .....	15
4.3.1	Pure TCL script .....	16
4.3.2	ADP script .....	16
4.3.3	ADP Page include .....	16
4.4	Error Handling .....	17
4.5	Future .....	17
5	References .....	18
5.1	JK2 Configuration .....	18
5.2	AOLserver Configuration .....	18
5.3	Tomcat Deployment .....	18
5.3.1	Use war .....	18
5.3.2	Deployment Manager .....	18

5.3.3	Server.xml context .....	18
5.3.4	Developers direct .....	18
5.3.5	Raw mode .....	18
6	Advanced Users .....	19
6.1	Building from source .....	19

---

# 1 History

There have been multiple attempts to create Tomcat plug-ins for AOLserver.

Briefly,

1. AOLserver 3.1 with jk connector using older Ajp12 protocol. For more information, see this link: [http://satori.com/aolserver/Howto\\_aol.txt](http://satori.com/aolserver/Howto_aol.txt)
2. nstomcat : Builds on the previous Ajp12 plugin but adds Java Native Interface (JNI) support. For more information, see this link: <http://www.cs.pub.ro/~gaburici/nstomcat/>
3. nsjk2: This uses the latest jk2 connector to provide JNI support.

This plug-in replaces those and is derived from the original work done on nsjk2. Significant work has gone into performance and scalability over the existing connectors, along with J2EE certification.

---

## 2 Basic Concepts

### 2.1 Jakarta Connectors

*Jakarta* refers to the project charged with creation and maintenance of commercial-quality, open-source, server-side solutions for the Java Platform. *Tomcat* is the servlet container that is used in the official Reference Implementation for the Java Servlet and JavaServer Pages technologies. Tomcat code is written mostly in Java, with some native code mixed in. Since Tomcat executes within the Java Virtual Machine, one can run it standalone, or within context of a user process, activating it by means of the Java Native Interface (JNI) invocation API.

The Tomcat *connector* responds to incoming requests and satisfies them by running servlets. There are connector classes that respond to HTTP 1.0/1.1 and SSL requests. One can run Tomcat as a stand-alone Web server. Alternately, one can have an Apache or other Web server communicating with Tomcat via IPC, such as TCP or local domain socket (AJP protocol), or via JNI. Advantages of AJP 1.3 protocol are that it can be used for communication across machines and with multiple Tomcat instances. JNI comes in handy when you need Java classes executing within an arbitrary process thread context, and has slightly better performance than AJP 1.3. One needs a Web server plug-in to implement AJP or JNI connectivity.

There are three generations of Tomcat plug-ins. The deprecated `mod_jserv` specifically targeted Apache-Tomcat. The JK family of modules covered more server platforms, and implemented more means of communication with Tomcat. The latest JK2 family is the most sophisticated. Its features can be tuned to satisfy performance, scalability, or reliability requirements.

Two important concepts about a JK2 module are *workers* and *channels*. A channel is responsible for client-side of communication with the Tomcat connector. The JK2 channel implements most of the specific protocol details. JK2 software includes channels for JNI and socket communications. The JK2 worker uses channel(s) to marshal Web server requests to Tomcat and send results from Tomcat back to the Web client. Workers typically implement such generic aspects of communication protocols as error recovery and routing, and depend on selected channels to carry out the rest of the work. The same worker can use more than one channel between requests. JK2 4.1 uses the AJP 1.3 worker class internally to carry out all socket or JNI communications. Configuration may bind a worker type to a specific URI, restrict channels to use, and so forth.

Other notable worker classes are

- The *load balancer* (`lb`), which uses other workers in a revolving manner to serve requests
- The *status* worker, which doesn't use a channel and writes JK2 status report to the client
- The *jni* worker, which is responsible for JVM startup and shutdown.

### 2.2 Notation

The following abbreviations will be used in this document:

1. **<TC>** – Directory pointing to the Tomcat installation which would be used for running Web applications.

2. **<AOLS>** - Top-level Directory of the AOLserver installation.
3. **<PG>** – Top-level directory containing the unzipped and untar'd plug-in package from nsjk2.tar.gz.

---

## 3 Installation

When you build nsjk2 from source, the 'packager' target will create a distribution tarball. The installation instructions in this document are for installation from that resulting tarball, [nsjk2.tar.gz](http://nsjk2.tar.gz). This tarball does not contain all prerequisites needed to complete the installation. The prerequisites are detailed in Prerequisites for Installation. Development efforts have addressed primarily in-process configuration (JNI) only. You may also configure Tomcat out-of-process using the Ajp13 connector; however, the out-of-process connector has not gone through our rigorous performance and scalability tests. We recommend that you use the in-process connector.

### 3.1 Supported Platforms

Development effort to date has targeted the Solaris and Linux platforms. Testing has been performed on Solaris 2.8 and Red Hat Advanced Server 2.1 (Linux 2.4.9).

### 3.2 Prerequisites for Installation

The following pre-requisites are needed to install the nsjk2 plug-in.

#### 3.2.1 AOLserver

First, install AOLserver (which in turn will require Tcl). Additional information on AOLserver can be found at [aolserver.com](http://aolserver.com). Be sure that you can serve a simple page out of your current AOLserver configuration before installing the nsjk2 plug-in.

#### 3.2.2 JDK/JVM

To run Tomcat, you will need Java Virtual Machine (JVM) software. All development to date has been with Sun's Java Development Kit (JDK). On Linux, we recommend that you use Sun JDK 1.4.1\_03 or greater, as earlier versions cause a signal handling issue. As of this writing, JDK 1.4.2 is available and works well on both Linux and Solaris, and is the recommended version.

We have not yet tested or certified with the IBM JVM yet.

You can download Sun's JDK from: <http://java.sun.com/j2se/downloads.html>

#### 3.2.3 Tomcat 4

You will need a Tomcat installation that can be modified for configuration with AOLserver and installation of our modified JKJNI connector. Testing was performed with Tomcat version 4.1.27. Tomcat can be retrieved from: <http://jakarta.apache.org/builds/jakarta-tomcat-4.0/release>

**Please ensure your stand-alone Tomcat installation works before going to the next step. This is very important, in order to isolate any Tomcat-specific issues before integrating with AOLserver.**

### 3.2.4 Jakarta Ant

To use the installation script from the tarball, you will need Ant 1.5.3 or greater installation. (Alternately, you can review the build.xml file and perform the copies manually.)

Download Ant from <http://ant.apache.org/bindownload.cgi>

Once the above prerequisites are in place, you are ready to install.

### 3.2.5 Plug-in Binary

Download and un-tar the [nsjk2.tar.gz](#) file in a temp directory (which we will refer to as PG). The PG directory contains the platform-specific dynamic modules, Java jar files, configuration templates, documentation, and the Ant-based installer file (build.xml).

## 3.3 Using the Ant-based Installer

You have the choice of using the Ant-based installer or directly modifying your Tomcat and AOLserver configuration. If you have a fresh installation of AOLserver and Tomcat, using the Ant-based installer is easiest way to get up and going.

The installer will install the necessary libraries, Tomcat configuration files, AOLserver start script (start-java) and configuration file (java.tcl). This default installation supports only JNI mode.

### 3.3.1 Assumptions

Prior to running the installer, ensure the following:

1. <PG>, <TC>, and <AOLS> directories exist independently – (<TC> is not inside <PG>)
2. Tomcat and AOLserver work in stand-alone mode.
3. You are doing a fresh installation (i.e., you don't already have Tomcat configured with some other Web applications). Specifically you have not modified the <TC>/conf files previously. The installer puts in fresh configuration files, it does not merge with existing ones in the <TC>/conf.
4. The JAVA\_HOME environment variable has been set to the JDK directory. Add \$ANT\_HOME/bin and \$JAVA\_HOME/bin to your PATH.

### 3.3.2 Installation Steps

1. cd to the <PG> directory and edit <PG>/installer.properties file.
2. In the installer.properties specify
  - a. Set property tomcat41.install.dir to point to the <TC> directory
  - b. Set property aolserver.home to point to the <AOLS> directory
3. cd <PG> ; ant
4. This will run Ant with the build.xml in <PG> directory as the default build file. The Ant script will automatically install all the configuration files and binaries to the <AOLS> and <TC> config directories. In addition, the Ant script will back up any configuration file in <TC>/conf directory that it modified. These include:

- a. server.xml moved to .server.xml.tc
  - b. workers2.properties moved to .worker2.properties.tc
  - c. jk2.properties moved to .jk2.properties.tc
5. Once the installation is complete, there is only a single command to run to start up Tomcat with AOLserver. Since Tomcat is running in-process, do not try to run <TC>/bin/startup.sh etc. To run: cd <AOLS>; bin/start-java
  6. Examine the <AOLS>/bin/start-java script if you want to run Tomcat in your custom script. Do not try to launch Tomcat standalone as is customary with Apache/Tomcat out-of-process mode. We use the CoyoteConnector as the basic connector for JNI channel and not the Ajp13 connector (out of process)
  7. By default, the bin/start-java script will log AOLserver's log to stdout. This can be changed by not passing the -f option to nsd in start-java script. Please read the AOLserver documentation of command line arguments to the nsd (Web server executable)
  8. The start-java script reads its AOLserver configuration from <AOLS>/java.tcl file. To configure AOLserver's properties, such as connection thread pool size, please modify it directly.

## 3.4 Manual Installation

If you have an existing installation of AOLserver and/or Tomcat, you may not want to use the Ant-based installer, as it will replace the existing configuration files. You may instead choose to perform a manual installation.

### 3.4.1 Copy Libraries

First, copy over the following libraries:

```
<PG>/lib/*-><AOLS>/lib/.
```

```
<PG>/bin/*-><AOLS>/bin/.
```

```
<PG>/jars/* -> <TC>/server/lib/.
```

The following documents configuration for both in- and out-of-process mode.

### 3.4.2 Tomcat In-process

The following configuration changes will need to occur to run Tomcat in the AOLserver process.

#### 3.4.2.1 File – <TC>/conf/jk2.properties

Enable the JNI handler.

```
# In order to enable jni use any channelJni directive
channelJni.disabled = 0

# This will enable the starting of the Tomcat from AOLserver plugin
apr.jniModeSo=inprocess
```

### 3.4.2.2 File – <TC>/conf/workers2.properties

Enable channel JNI and disable socket channel (Ajp13) and shared memory channel.

```
[channel.jni:jni]
disabled=0
debug=0

[channel.socket:localhost:8009]
disabled=1

[shm:]
disabled=1
```

Specify the JVM and its start-up arguments. This is a good place to specify JPDA options, if any.

```
[vm:]
JVM=@JAVA_HOME@/jre/lib/@OS_ARCH@/server/libjvm.so
OPT=-server
OPT=-Dtomcat.home=@TOMCAT41_HOME@
OPT=-Dcatalina.home=@TOMCAT41_HOME@
OPT=-Xmx384M
OPT=-Xss512k
OPT=-Xms384m
OPT=-Xcheck:jni
OPT=-verbose

# JPDA debugging flags
OPT=-Xdebug
OPT=-Xnoagent
OPT=-Djava.compiler=NONE
OPT=-Xrunjdp:transport=dt_socket,server=y,suspend=n,address=5005

OPT=-Djava.class.path=
OPT=-Djava.library.path=...
OPT=-Xrs
```

In the above configuration, if `-Xrs` is specified, it allows clean termination with `SIGINT|HUP|TERM` signals. Without it, JVM traps `SIGINT`, causing Hotspot core dump to be generated instead!

Note: the disadvantage of specifying `-Xrs` is that `SIGQUIT` (CTRL-`^`) Java thread dumps are not available.

Specify start-up and stop configuration for the JNI workers.

```
[worker.jni:onStartup]
class=org/apache/jk/apr/TomcatStarter
ARG=start
disabled=0
#stdout=${serverRoot}/logs/stdout.log
#stderr=${serverRoot}/logs/stderr.log
debug=0

[worker.jni:onShutdown]
```

```
class=org/apache/jk/apr/TomcatStarter
ARG=stop
disabled=0
```

Do not forget to specify the subset of the URI space to be handled by Tomcat. By default, /examples is typically mapped to Tomcat.

```
[uri:/examples/*]
info=Example webapp in the default context.
context=/examples
debug=0
group=ajp13:jni
```

Note that we specify group → ajp13:jni to inform the jk2 connector to map the URI space to JNI workers. For more documentation of the jk2 worker2 file, please consult the Jakarta jk2 Web page. Templates of all these configuration files are included in the <PG> directory

### 3.4.2.3 File – <TC>/conf/server.xml

Comment out all the connectors specified inside the <Service name="Tomcat-Standalone"> tag except for CoyoteConnector. Without CoyoteConnector in server.xml, you will see a strange error in AOLserver log files complaining about no registered handler.

```
<!-- Define a Coyote/JK2 Connector on port 8001 -->
<Connector className="org.apache.coyote.tomcat4.CoyoteConnector"
  port="8001" minProcessors="5" maxProcessors="75"
  enableLookups="true" redirectPort="8443"
  acceptCount="10" debug="0" connectionTimeout="0"
  useURIVValidationHack="false"
  protocolHandlerClassName="org.apache.jk.server.JkCoyoteHandler"/>
```

The jk2 connector requires you to specify a port even though channel JNI is used.

### 3.4.2.4 File – <AOLS>/<config.tcl>

Modify the following sections in your AOLserver configuration file.

Load the Tomcat4 plug-in:

```
# Modules to load
#
ns_section "ns/server/${servername}/modules"
  ns_param ...
  ns_param nsjk2 ${bindir}/nsjk2.so
```

Tell AOLserver plug-in where the Tomcat root directory is, so that it can read the workers2.properties and jk2.properties files:

```
ns_section "ns/module/nsjk2"
  ns_param serverRoot @TOMCAT41_HOME@
```

Finally, configure the Tomcat plug-in with the server hostname to use for redirects, etc.

```
ns section "ns/server/${servername}/module/nsjk2"  
  ns_param serverName "${hostname}"
```

### 3.4.2.5 File – <AOLS>/bin/start-java

You can pick up the sample start-up script from the <PG> directory. The script sets up environment variables. These can be manually set:

1. TOMCAT\_HOME
2. JAVA\_HOME
3. AOL\_SERVER
4. LD\_LIBRARY\_PATH should at least contain:

```
$JAVA_HOME/jre/lib/@OS_ARCH@/server:$JAVA_HOME/jre/lib/@OS_ARCH@:$AOL_SERVER/lib:$AOL_SERVER/bin
```

5. CLASSPATH should at least contain:

```
$TOMCAT_HOME/common/lib/servlet.jar:$TOMCAT_HOME/server/lib/tomcat-util.jar
```

## 3.4.3 Tomcat Out of Process

To date, development of the nsjk2 package has been dedicated to running Tomcat inside the web server process. There are several advantages to running Tomcat inside AOLserver using the JNI mode. Here are some issues that you may encounter when running Tomcat out of process:

### 3.4.3.1 Issues with Out-of-Process Mode

#### 3.4.3.1.1 Performance

The Ajp13 persistent connection[s] can become a bottleneck if there are many concurrent connections. In addition, there is an AOLserver worker thread context switch every time it tries to do network I/O via Ajp13 connection.

For an application that has a severe database bottleneck, JNI or AJP may not matter from a raw throughput or response time perspective.

#### 3.4.3.1.2 Administration

Managing multiple processes on a box from an administrative perspective implies more ports to be configured and administered.

### 3.4.3.1.3 Scalability

The out-of-process mode consumes more resources. TCP ports (not sockets) on a box are bounded, so if you want to support many open concurrent connections, out-of-process mode will cause it to be divided between the Web server and Tomcat processes.

### 3.4.3.2 Installation Out of Process

#### 3.4.3.2.1 File – <TC>/conf/jk2.properties

Modify <TC>/conf/jk2.properties .Set the handler list to contain:

```
handler.list=apr,request,channelSocket
```

Make sure you either disable JNI mode or do not specify any JNI directives

```
# In order to enable jni use any channelJni directive
#channelJni.disabled = 0
# This will enable the starting of the Tomcat in-process
#apr.jniModeSo=inprocess
```

#### 3.4.3.2.2 File – <TC>/conf/workers2.properties

In jk2 workers2.properties, disable JNI and non-socket channels and enable Ajp13 tcp channel. Ajp13 socket channel will need to specify the Ajp13 listener port (Tomcat side).

```
# Define the communication channel
[channel.socket:localhost:5000]
info=Ajp13 forwarding over socket
tomcatId=localhost:5000
lb factor=1
group=lb workers
debug=0

[channel.socket:localhost:6000]
info=Ajp13 forwarding over socket
tomcatId=localhost:6000
lb factor=1
group=lb workers
debug=0
```

To specify load-balanced workers with sticky load-balancing:

```
[lb:lb workers]
info=AOL server/tomcat load balancer.
debug=0
worker=ajp13:localhost:6000
worker=ajp13:localhost:5000

# Now map to URI space
[uri:/aol/*]
```

```
group=lb workers
stickySession=1
```

In theory, this should cause AOLserver/Apache Web server to do sticky load balancing to the two Tomcat instances.

Also delete the [vm:] section.

In the URI map, specify the lb\_workers as above or directly the ajp13 worker group name.

### 3.4.3.2.3 File – <TC>/conf/server.xml

Enable the Ajp13 connector and disable other connectors (CoyoteConnector):

```
<!-- Define an AJP 1.3 Connector on port 5003 -->
<Connector className="org.apache.ajp.tomcat4.Ajp13Connector"
    port="5003" minProcessors="5" maxProcessors="75"
    acceptCount="10" debug="0"/>
```

### 3.4.3.2.4 Run scripts

Start up Tomcat standalone using the TC/bin/startup.sh scripts or your own custom script.

Then, run AOLserver with the same tcl configuration file as the in-process case.

---

## 4 Java-TCL Bridge

A significant advantage to using nsjk2 to combine AOLserver and Tomcat is to leverage AOLserver's ADP and TCL technology. This can be particularly helpful in leveraging existing AOLserver/TCL-implemented services without forcing a port to Java. The Java-TCL Bridge enables you to process TCL-based scripts and pages from within your Java Servlet or JSP. The bridge supports following access patterns:

1. Evaluate pure TCL scripts from Java.
2. Evaluate ADP scripts from Java.
3. Include ADP pages in a JSP or a servlet.

The first two return the result of TCL interpreter's evaluation as Java strings that can be appended into the output response by the application writer. The third access pattern commits the TCL interpreter's evaluation results into the print writer passed by the application to the eval method. In other words, you can have the Tcl response directly written to servlet's response stream.

### 4.1 Assumptions

1. AOLserver and the Tomcat plug-in are already installed and working
2. TCL 8.4.x is being used.

### 4.2 Installation

1. Ensure that the Tomcat classpath contains javatclbridge.jar. This can be done in several ways. One of them would be to modify the workers2.properties file and add the following lines:

```
[vm:]  
...  
classpath=${AOLSERVER_HOME}/lib/javatclbridge.jar
```

2. Add <AOLS>/lib to AOLserver's LD\_LIBRARY\_PATH.

### 4.3 Usage Examples

Here are some typical usage examples. For further details, please see the Javadoc-generated files in <PG>/docs/api. In addition, our distribution tar includes two examples: HelloJavaAdp.java and JavaTclExamples.java

### 4.3.1 Pure TCL script

```
try {
    // Simple TCL command eval
    NsTcl tcl = new NsTcl(TclInterp.forThread());
    TclScript scr = new TclScript ("info command *");
    String res = tcl.eval (scr);
    PrintWriter out = response.getWriter();
    out.println("<p>" + res);
} catch (TclEvalException e) {
    log("TCL interp error", e);
    throw new ServletException(e); // rethrow for fun
}
```

In this example, we have used the NsTcl Java object to evaluate Tcl script in string. Objects of this class can be cached within lifetime of the respective interpreter object. Do not use this class for ADP script evaluation.

### 4.3.2 ADP script

NsAdp provides a Java wrapper around Tcl interpreter so that ADP commands can be called from within Java. Do not cache the NsAdp objects in any way. They are valid only within the context of the current request.

```
try {
    NsAdp nadp = new NsAdp(this.getServletContext(), out);
    String fooVal = null;
    fooVal = nadp.eval("ns queryget foo");
    out.println("<p>" + fooVal);
} catch (TclEvalException e) {
    log("TCL interp error", e);
    throw new ServletException(e); // rethrow for fun
}
```

In the above example, we use the ADP command “ns\_queryget” to get the value associated with the query parameter “foo”.

### 4.3.3 ADP Page include

Often, ADP pages exist that do the ADP/TCL processing and produce dynamic content that just needs to be directly streamed into the servlet response. To reuse existing ADP pages, you can use the NsAdp.eval() method and pass it “ns\_adp\_include <adp-filename>” string or you can use a convenient method we provide : NsAdp.include(String adpFile)

```
public class HelloJavaAdp extends HttpServlet {

    public void doGet(HttpServletRequest request,
                     HttpServletResponse response)
        throws IOException, ServletException
    {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        // Now test evaluation of an adp command
        NsAdp nadp = new NsAdp(this.getServletContext(), out);

        // Now test include of an adp page. After page eval, content should be
        // appended into the servlet output response stream automatically.
```

```
try {
    nadp.include("adp/hello.adp");
} catch (TclEvalException e) {
    log("TCL Interp error", e);
    throw new ServletException(e); // rethrow for fun
}
//nadp.eval("ns_adp_include adp/hello.adp");
}
```

Remember that all file references in the include are treated relative to the servlet context. In this case, the file would be fetched from "<tomcatroot>/webapps/examples/adp/hello.adp". It is good practice to put all the ADP pages in a separate folder. The ADP pages can also be packaged in a regular .war file and accessed as above in your servlet.

Includes starting with "/" are still mapped to within the servlet context. No absolute paths are allowed.

## 4.4 Error Handling

If the TCL interpreter doesn't like the syntax of the passed scriptlet, it puts the error results in a TCL variable, `errorInfo`. We propagate that as a "TclEvalException" in Java. The exception raised contains the message from TCL variable `errorInfo`. This will normally provide important debug information such as the TCL code back trace to where the error occurred.

## 4.5 Future

Unlike Java Servlet and JSP, which can be tightly coupled and share states like `HttpSession`, the Java-TCL bridge is essentially stateless. Potential enhancements could include session sharing, back-end connection pool sharing, request chaining, and interleaving.

---

## 5 References

### 5.1 JK2 Configuration

For further details, look at <http://jakarta.apache.org/tomcat/tomcat-4.1-doc/jk2>

### 5.2 AOLserver Configuration

For detailed configuration options and setup, look at <http://www.aolserver.com/>

### 5.3 Tomcat Deployment

Tomcat lets you deploy at least five different ways. Depending upon the deployment unit (have a war file or un-war'd distribution or have your own docroot with classes) you have flexibility.

#### 5.3.1 War files

If you have a .war already, the simplest method is to copy the Web application war into the \$CATALINA\_HOME/webapps.

When Tomcat is started, it will automatically expand the war and deploy it. With this approach, if you want to redeploy, you need to copy the .war again and delete the expanded war first.

This is our recommended approach. There are a number of tools available to easily create war files.

#### 5.3.2 Deployment Manager

You can use Tomcat's deployment manager and tasks in your own scripts. You can find documentation and samples on the Tomcat Web site.

#### 5.3.3 Server.xml context

You can add your context to \$CATALINA\_HOME/conf/server.xml and have it point to your docroot anywhere. There is no need to copy it into Tomcat's distribution directly. This will require a restart of Tomcat. It is assumed your context points to the root of the Web application tree (has WEB-INF, and so forth, already).

#### 5.3.4 Developers direct

Developers can use third party IDEs that let them automate deployment to Tomcat. Tools such as Eclipse, CodeGuide, IntelliJ, JBuilder all support direct deployment to Tomcat using the GUI.  
Raw mode

Finally, you can expand your .war or copy your classes into the webapps/ dir

See <http://jakarta.apache.org/tomcat/tomcat-4.1-doc/appdev/deployment.html> for more details.

---

## 6 Advanced Users

### 6.1 Building from Source

You can download a source tarball from the SourceForge downloads page at [http://sourceforge.net/project/showfiles.php?group\\_id=3152](http://sourceforge.net/project/showfiles.php?group_id=3152) or check it out directly from SourceForge's CVS repository:

```
cvs co -d :pserver:<USER>@cvs.sourceforge.net:/cvsroot/aolserver nsjk2.
```

Consult the README file and build properties in the source for additional information on build steps and requirements.