Links

- Docs Home

Getting Started

- Introduction
- READ ME
- Install and Run
- Building from Source

Configuration

- Reference

Administrators

- CGI HOW-TO
- Class Loader HOW-TO
- Connectors List
- HTML Manager App HOW-TO
- JK Documentation
- JNDI DataSource HOW-TO
- JNDI Resources HOW-TO
- JSP Engine Config HOW-TO
- Manager App HOW-TO
- MBean Descriptor HOW-TO
- Proxy Support HOW-TO
- Realm HOW-TO
- Security Mgr. HOW-TO
- SSI Config HOW-TO
- SSL Config HOW-TO

Application Developers

- App Developer Guide
- Servlet/JSP Javadocs

Catalina Developers

- Functional Specs.
- Javadocs

Jasper Developers

- Javadocs

The Tomcat 4 Servlet/JSP Container

print-friendly version

Documentation Index

Introduction

This is the top-level entry point of the documentation bundle for the **Tomcat 4** Servlet/JSP container. Tomcat 4 implements the Servlet 2.3 and JavaServer Pages 1.2 specifications from Java Software, and includes many additional features that make it a useful platform for developing and deploying web applications and web services.

Select one of the links from the navigation menu (to the left) to drill down to the more detailed documenation that is available. Each available manual is described in more detail below.

Getting Started

The following documents will assist you in downloading and installing Tomcat 4, and (if you wish to) building a distribution from the source code.

- **Introduction** - A brief, high level, overview of Tomcat.
- **README.txt** - Describes the contents and directory structure of a Tomcat 4 binary distribution.
- **RUNNING.txt** - Documents the steps necessary to download, install, start, and stop a Tomcat 4 server.
- **BUILDING.txt** - Details the steps necessary to download Tomcat 4 source code (and the other packages that it depends on), and build a binary distribution from those sources.
- **Server Configuration Reference** - Reference manual that documents all available elements and attributes that may be placed into a Tomcat 4 `conf/server.xml` file.

Administrators

The following documents are aimed at *System Administrators* who are responsible for installing, configuring, and operating a Tomcat 4 server.

- **CGI HOW-TO** - Configuring Tomcat to use CGI (Common Gateway Interface).
- **Connectors List** - List of the connectors (both HTTP and native webservers) available for use with Tomcat.
- **Class Loader HOW-TO** - Information about class loading in Tomcat 4, including where to place your application classes so that they are visible.
- **HTML Manager App HOW-TO** - Operating the `HTML Manager` web app to deploy, undeploy, and redeploy applications while Tomcat is running.
- **JNDI DataSource HOW-TO** - Configuring a JNDI DataSoure with a dB connection pool. Examples for many popular databases.

- **JK Documenation** - Complete documentation and HOWTOs on the JK native webserver connector, used to interface Tomcat with servers like Apache HTTPd, IIS and others.
- **JNDI Resources HOW-TO** - Configuring standard and custom resources in the JNDI naming context that is provided to each web application.
- **JSP Engine Config HOW-TO** - The Jasper 2 JSP Engine. How it works and how to configure it.
- **Manager App HOW-TO** - Operating the `Manager` web app to deploy, undeploy, and redeploy applications while Tomcat is running.
- **MBean Descriptor HOW-TO** - Configuring MBean descriptors files for custom components.
- **Proxy Support HOW-TO** - Configuring Tomcat 4 to run behind a proxy server (or a web server functioning as a proxy server).
- **Realm Configuration HOW-TO** - Description of how to configure *Realms* (databases of users, passwords, and their associated roles) for use in web applications that utilize *Container Managed Security* (FIXME - hyperlink to background info on this).
- **Security Manager HOW-TO** - Configuring and using a Java Security Manager to support fine-grained control over the behavior of your web applications.
- **SSI HOW-TO** - Configuring Tomcat to use SSI (Server Side Include).
- **SSL Configuration HOW-TO** - Installing and configuring SSL support so that your Tomcat will serve requests using the `https` protocol.

Application Developers

The following documents are aimed at *Application Developers* who are constructing web applications or web services that will run on Tomcat.

- **Application Developer's Guide** - An introduction to the concepts of a *web application* as defined in the Servlet 2.3 Specification. Covers basic organization of your web application source tree, the structure of a web application archive, and an introduction to the web application deployment descriptor (`/WEB-INF/web.xml`).
- **Servlet/JSP Javadocs** - The Servlet 2.3 and JSP 1.2 API Javadocs.

Catalina Developers

The following documents are for Java developers who wish to contribute to the development of the *Catalina* servlet container portion of Tomcat itself, or to better understand its internal architecture and operation.

- **Functional Specifications** - Requirements specifications for features of the *Catalina* servlet container portion of Tomcat 4.
- **Javadocs** - Javadoc API documentation for the *Catalina* servlet container portion of Tomcat 4.

Jasper Developers

The following documents are for Java developers who wish to contribute to the development of the *Jasper* JSP container portion of Tomcat itself, or to better understand its internal architecture and operation.

- **Javadocs** - Javadoc API documentation for the *Jasper* JSP container portion of Tomcat 4.

Links

- Docs Home

Getting Started

- Introduction
- READ ME
- Install and Run
- Building from Source

Configuration

- Reference

Administrators

- CGI HOW-TO
- Class Loader HOW-TO
- Connectors List
- HTML Manager App HOW-TO
- JK Documentation
- JNDI DataSource HOW-TO
- JNDI Resources HOW-TO
- JSP Engine Config HOW-TO
- Manager App HOW-TO
- MBean Descriptor HOW-TO
- Proxy Support HOW-TO
- Realm HOW-TO
- Security Mgr. HOW-TO
- SSI Config HOW-TO
- SSL Config HOW-TO

Application Developers

- App Developer Guide
- Servlet/JSP Javadocs

Catalina Developers

- Functional Specs.
- Javadocs

Jasper Developers

- Javadocs

The Tomcat 4 Servlet/JSP Container

print-friendly version

Introduction

Introduction

For administrators and web developers alike, there are some important bits of information you should familiarize yourself with before starting out. This document serves as a brief introduction to some of the concepts and terminology behind the Tomcat container. As well, where to go when you need help.

Terminology

In the course of reading these documents, you'll run across a number of terms; some specific to Tomcat, and others defined by the Servlet or JSP specifications.

- **Context** - In a nutshell, a Context is a web application.
- **Term2** - This is it.
- **Term3** - This is it!

Directories and Files

Throughout the docs, you'll notice there are numerous references to **$CATALINA_HOME**. This represents the root of your Tomcat installation. When we say, "This information can be found in your $CATALINA_HOME/README.txt file" we mean to look at the README.txt file at the root of your Tomcat install.

For the complete description of the Tomcat distribution, each folder can be found in the README.txt file, residing in the root directory of your Tomcat installation. Here, we will cover the ones where you'll be spending the majority of your time.

- **/bin** - Startup, shutdown, and other scripts. The `*.sh` files (for Unix systems) are functional duplicates of the `*.bat` files (for Windows systems). Since the Win32 command-line lacks certain functionality, there are some additional files in here.
- **/conf** - Configuration files and related DTDs. The most important file in here is server.xml. It is the main configuration file for the container.
- **/logs** - Log files are here by default.
- **/webapps** - This is where your webapps go.

Configuring Tomcat

This section will acquaint you with the basic information used during the configuration of the container.

All of the information in the configuration files is read at startup, meaning that any change to the files necessitates a restart of the container.

Where to Go for Help

While we've done our best to ensure that these documents are clearly written and easy to understand, we may have missed something. Provided below are various web sites and mailing lists in case you get stuck.

As Tomcat 4 is a new release of Tomcat, keep in mind that some of the issues and solutions vary between the major versions of Tomcat (3.x versus 4.x). As you search around the web, there will be some documentation that is not relevant to Tomcat 4, but 3.x. Doing 3.x things to 4.0 will probably not work in most cases as the server.xml files are very different.

- Current document - most documents will list potential hangups. Be sure to fully read the relevant documentation as it will save you much time and effort. There's nothing like scouring the web only to find out that the answer was right in front of you all along!
- Jakarta FAQ-o-matic - a repository of FAQs for the various Jakarta subprojects, including Tomcat of course.
- Tomcat FAQ at jGuru
- Tomcat mailing list archives - numerous sites archive the Tomcat mailing lists. Since the links change over time, clicking here will search Google.
- The TOMCAT-USER mailing list, which you can subscribe to here. If you don't get a reply, then there's a good chance that your question was probably answered in the list archives or one of the FAQs. Although questions about web application development in general are sometimes asked and answered, please focus your questions on Tomcat-specific issues.
- The TOMCAT-DEV mailing list, which you can subscribe to here. This list is **reserved** for discussions about the development of Tomcat itself. Questions about Tomcat configuration, and the problems you run into while developing and running applications, will normally be more appropriate on the TOMCAT-USER list instead.

And, if you think something should be in the docs, by all means let us know on the TOMCAT-DEV list, or send one of the doc authors email.

$Id: README.txt,v 1.20 2003/01/08 03:50:26 glenn Exp $

The Tomcat 4.1 Servlet/JSP Container

This subproject contains a server that conforms to the Servlet 2.3 and
JSP 1.2 specifications from Java Software. It includes the following contents:

BUILDING.txt	Instructions for building from sources
LICENSE	Apache Software License for this release
README.txt	This document
RELEASE-NOTES-*.txt	Release Notes for this (and previous) releases of Tomcat 4.1
RUNNING.txt	Instructions for installing Tomcat, as well as starting and stopping the server
bin/	Binary executables and scripts
common/	Classes available to both Catalina internal classes and web applications:
classes/	Unpacked common classes
lib/	Common classes in JAR files
conf/	Configuration files
logs/	Destination directory for log files
server/	Internal Catalina classes and their dependencies
classes/	Unpacked classes (internal only)
lib/	Classes packed in JAR files (internal only)
shared/	Classes shared by all web applications
classes/	Unpacked shared classes
lib/	Shared classes in JAR files
webapps/	Base directory containing web applications included with Tomcat 4.1
work/	Scratch directory used by Tomcat for holding temporary files and directories
temp/	Directory used by JVM for temporary files (java.io.tmpdir)

If you wish to build the Tomcat server from a source distribution,
please consult the documentation in "BUILDING.txt".

If you wish to install and run a binary distribution of the Tomcat server,
please consult the documentation in "RUNNING.txt".

Acquiring Tomcat 4.1 Releases

Nightly Builds

Nightly Builds of Tomcat 4.1 are built from the most recent CVS sources each evening (Pacific Time). The filename of the downloadable file includes the date it was created (in YYYYMMDD format). These builds are available at:

Binary: http://jakarta.apache.org/builds/jakarta-tomcat-4.0/nightly/
Source: http://jakarta.apache.org/builds/jakarta-tomcat-4.0/nightly/src/

Release Builds

Release Builds of Tomcat 4.1 are created and released periodically, and announced to the interested mailing lists. Each release build resides in its own directories. For example, the Tomcat 4.1.18 release is available at:

Binary: http://jakarta.apache.org/builds/jakarta-tomcat-4.0/release/v4.1.18/bin/
Source: http://jakarta.apache.org/builds/jakarta-tomcat-4.0/release/v4.1.18/src/

$Id: RUNNING.txt,v 1.5 2002/04/24 08:06:14 remm Exp $

Running The Tomcat 4.0 Servlet/JSP Container

This subproject contains Tomcat 4.0, a server that implements the Servlet 2.3 and JSP 1.2 Specifications from Java Software. In order to install and run this container, you must do the following:

(0) Download and Install a Java Development Kit

* Download a Java Development Kit (JDK) release (version 1.2 or later) from:

  http://java.sun.com/j2se/

* Install the JDK according to the instructions included with the release.

* Set an environment variable JAVA_HOME to the pathname of the directory into which you installed the JDK release.

(1) Download and Install the Tomcat 4.0 Binary Distribution

NOTE: As an alternative to downloading a binary distribution, you can create your own from the Tomcat source repository, as described in "BUILDING.txt". If you do this, the value to use for "${catalina.home}" will be the "dist" subdirectory of your source distribution.

* Download a binary distribution of Tomcat from:

  http://jakarta.apache.org/builds/jakarta-tomcat-4.0/nightly/

  On a Windows platform, you will need:
  jakarta-tomcat-4.0-YYYYMMDD.zip

  On a Unix platform, you will need:
  jakarta-tomcat-4.0-YYYYMMDD.zip

* Unpack the binary distribution into a convenient location so that the distribution resides in its own directory (conventionally named "jakarta-tomcat-4.0"). For the purposes of the remainder of this document, the symbolic name "${catalina.home}" is used to refer to the full pathname of the release directory.

(2) Start Up Tomcat 4.0

There are two techniques by which Tomcat 4.0 can be started:

* Via an environment variable:
  - Set an environment variable CATALINA_HOME to the path of the directory
    into which you have installed Tomcat 4.0.
  - Execute the shell command:

      %CATALINA_HOME%\bin\startup   (Windows)

      $CATALINA_HOME/bin/startup.sh   (Unix)

* By modifying your current working directory:
  - Execute the following shell commands:

      cd %CATALINA_HOME%\bin   (Windows)
      startup   (Windows)

      cd $CATALINA_HOME/bin   (Unix)
      ./startup.sh   (Unix)

After startup, the default web applications included with Tomcat 4.0 will be
available by browsing:

    http://localhost:8080/

Further information about configuring and running Tomcat 4.0 can be found in
the documentation included here, as well as on the Tomcat web site:

    http://jakarta.apache.org/tomcat/

(3) Shut Down Tomcat 4.0

There are two techniques by which Tomcat 4.0 can be stopped:

* Via an environment variable:
  - Set an environment variable CATALINA_HOME to the path of the directory
    into which you have installed Tomcat 4.0.
  - Execute the shell command:

      %CATALINA_HOME%\bin\shutdown   (Windows)

```
    $CATALINA_HOME/bin/shutdown.sh         (Unix)
```

* By modifying your current working directory:
  - Execute the following shell commands:

```
    cd %CATALINA_HOME%\bin                 (Windows)
    shutdown                               (Windows)

    cd $CATALINA_HOME/bin                  (Unix)
    ./shutdown.sh                          (Unix)
```

(4) Advanced Configuration - Multiple Tomcat 4 Instances

In many circumstances, it is desirable to have a single copy of a Tomcat 4 binary distribution shared among multiple users on the same server. To make this possible, you must configure a CATALINA_BASE environment variable (in addition to CATALINA_HOME as described above) that points to a directory that is unique to your instance.

When you do this, Tomcat 4 will calculate all relative references for files in the following directories based on the value for CATALINA_BASE instead of CATALINA_HOME:

* conf - Server configuration files (including server.xml)

* logs - Log and output files

* webapps - Automatically loaded web applications

* work - Temporary working directories for web applications

* temp - Directory used by the JVM for temporary files (java.io.tmpdir)

If you do not set CATALINA_BASE to an explicit value, it will be initialized to the same value as is set for CATALINA_HOME (which means that the same directory is used for all relative path resolutions).

The administration and manager web applications, which are defined in the $CATALINA_BASE/webapps/admin.xml and $CATALINA_BASE/webapps/manager.xml will not run in that configuration, unless either:
- The path specified in the docBase attribute of the Context element is made absolute, and replaced respectively by $CATALINA_HOME/server/webapps/admin and $CATALINA_HOME/server/webapps/manager
- Copying and linking both web applications in $CATALINA_BASE, and modify accordingly the path specified in the docBase attribute of the Context

element
- Disabling both web applications by removing $CATALINA_BASE/webapps/admin.xml and $CATALINA_BASE/webapps/manager.xml

(5) Troubleshooting:

There are only really 3 things that can go wrong during the stand-alone Tomcat 4.0 install:

1) The most common hiccup is when another web server (or any process for that matter) has laid claim to port 8080. This is the default HTTP port that Tomcat attempts to bind to at startup. To change this, open the file:

   $CATALINA_HOME/conf/server.xml

   ...and search for '8080'. Change it to a port that isn't in use, and is greater than 1024, as ports less than or equal to 1024 require superuser access to bind to.

   Restart Tomcat and you're in business. Be sure that you replace the "8080" in the URL you're using to access Tomcat. For example, if you change the port to 1977, you would request the URL http://localhost:1977/.

2) An "out of environment space" error when running the batch files in Win9X/ME-based operating systems.

   Right-click on the STARTUP.BAT and SHUTDOWN.BAT files. Click on "Properties" then on the "Memory" tab. For the "Initial environment" field, enter in something like 4096.

   After you click apply, Windows will create shortcuts in the directory with which you can use to start and stop the container.

3) The 'localhost' machine isn't found. This could happen if you're behind a proxy. If that's the case, make sure the proxy configuration for your browser knows that you shouldn't be going through the proxy to access the "localhost" machine.

   In Netscape, this is under Edit/preferences -> Advanced/proxies, and in Internet Explorer, Tools -> Internet Options -> Connections -> LAN Settings.

$Id: BUILDING.txt,v 1.33 2002/07/29 19:06:09 patrickl Exp $

Building The Tomcat 4.0 Servlet/JSP Container

This subproject contains the source code Tomcat 4.0, a server that implements the Servlet 2.3 and JSP 1.2 Specifications from Java Software. In order to build a binary distribution version of the container from a source distribution, you must have a Java Development Kit (JDK) for version 1.3 (or later) downloaded and installed (version 1.3.1 recommended), and do the following:

(0) Download and Install a Java Development Kit

* Download a Java Development Kit (JDK) release (version 1.3 or later) from:

  http://java.sun.com/j2se/

* Install the JDK according to the instructions included with the release.

* Set an environment variable JAVA_HOME to the pathname of the directory into which you installed the JDK release.

(1) Download and Install the Ant Binary Distribution

NOTE: Previous versions of Tomcat 4.0 relied on Ant 1.3 for the build process. The 1.5 release is now required.

* Download a binary distribution of Ant 1.5 from:

  http://jakarta.apache.org/builds/jakarta-ant/release/v1.5/bin/

  On a Windows platform, you will need:
  jakarta-ant-1.5-bin.zip

  On a Unix platform, you will need:
  jakarta-ant-1.5-bin.tar.gz

* Unpack the binary distribution into a convenient location so that the Ant release resides in its own directory (conventionally named "jakarta-ant-1.5"). For the purposes of the remainder of this document, the symbolic name "${ant.home}" is used to refer to the full pathname of

the release directory.

* Modify the PATH environment variable to include directory "${ant.home}/bin" in its list. This makes the "ant" command line script available, which will be used to actually perform the build.

(2) Download and Install the Java XML Pack Binary Distribution

* Download a binary distribution of Java XML Pack:

http://java.sun.com/xml/downloads/javaxmlpack.html

* Unpack the binary distribution into a convenient location so that the Java XML Pack release resides in its own directory (the JAXP libraries reside in the jaxp-1.1.3 subdirectory). For the purposes of the remainder of this document, the symbolic name "${jaxp.home}" is used to refer to the full pathname to the jaxp-1.1.3 subdirectory of the Java XML Pack release directory.

* Make the "xalan.jar" file of this distribution available to Ant (so that it can be used with the <style> tag) by copying it to "${ant.home}/lib".

* This is optional with JDK 1.4 or later.

(3) Download and Install the JNDI 1.2.1 Reference Implementation

* Download the Java Naming and Directory Interface (JNDI) package, (version 1.2.1 or later) from

http://java.sun.com/products/jndi/

* Unpack the reference implementation into a convenient location so that it resides in its own subdirectory.

* You will also need the LDAP Service Provider Maintenance package, (version 1.2.3 or later) available on the same download page. Be sure that you unpack "ldap.jar" and "jaas.jar" into the "lib" subdirectory of the JNDI directory, parallel to "jndi.jar".

* This is optional with JDK 1.3 or later.

(4) Download and Install the Xerces 1 or 2 Distribution

* Download a binary distribution from:

  http://xml.apache.org/dist/xerces-j/

  (Tomcat was tested with "Xerces-J-bin.1.4.3.zip")

* Unpack the binary distribution into a convenient location so that the
  distribution resides in its own directory (conventionally named
  "xerces-x_y_z").

* In your build.properties file, you will need to set properties differently
  based on which version of Xerces you are using:
  - For versions 1.3.1 up through and including 2.0.0beta3,
    set the "xerces.jar" property to point at the full
    pathname of the corresponding file.
  - For version 2.0.0beta4 and later, set the "xmlParserAPIs.jar"
    and "xercesImpl.jar" properties to point at the corresponding
    files
  - If you have defined both sets of properties, the newer (two files)
    packaging will be loaded into the Tomcat you are building

* This is optional with JDK 1.4 or later.

(5) Download and Install Subproject Source Code

* Use Anonymous CVS (as described on the Jakarta web site at
  <http://jakarta.apache.org/site/cvsindex.html>, or
  download a source distribution from:

    http://jakarta.apache.org/builds/jakarta-tomcat-4.0/nightly/src/

  On a Windows platform, you will need:
    jakarta-tomcat-4.0-src-YYYYMMDD.zip

  On a Unix platform, you will need:
    jakarta-tomcat-4.0-src-YYYYMMDD.tar.gz

  (NOTE: Alternatively, you can grab the source distribution from a
  formal release, if you wish, by following links on the Jakarta web site).

* Unpack the source distribution into a convenient location so that the
  distribution resides in its own directory (conventionally named
  "jakarta-tomcat-4.0"). For the purposes of the remainder of this document,
  the symbolic name "${tomcat.source}" is used to refer to the full
  pathname of the release directory.

* Use Anonymous CVS (as described on the Jakarta web site at
  <http://jakarta.apache.org/site/cvsindex.html>, or
  download a source distribution of the "jakarta-tomcat-connectors" repository.

* Unpack the source distribution into a convenient location so that the
  distribution resides in its own directory (conventionally named
  "jakarta-tomcat-connectors"). By default, the build system will try to find
  this repository in the "${tomcat.source}/../jakarta-tomcat-connectors"
  directory. Set the "jtc.home" property in the build.properties file
  described below (see 26) if it resides in a different directory.

(6) Download and Install the Servlet API Binary Distribution

NOTE: Alternatively, you can download the sources of the
"jakarta-servletapi-4" subproject, and build a binary distribution yourself,
or use the binary distribution that is available with binary distributions
of Tomcat 4.0.

* Download a binary distribution from:

    http://jakarta.apache.org/builds/jakarta-servletapi-4/nightly/

  On a Windows platform, you will need:
    jakarta-servletapi-4-YYYYMMDD.zip

  On a Unix platform, you will need:
    jakarta-servletapi-4-YYYYMMDD.tar.gz

* Unpack the binary distribution into a convenient location so that the
  distribution resides in its own directory.

(7) Download and Install the Commons Beanutils Binary Distribution

* Download a binary distribution of Version 1.1 or later from:

    http://jakarta.apache.org/builds/jakarta-commons/release/commons-beanutils

  On a Windows platform, you will need:
    commons-beanutils-X.Y.zip

  On a Unix platform, you will need:
    commons-beanutils-X.Y.tar.gz

* Unpack the binary distribution into a convenient location so that the
  distribution resides in its own directory.

(8) Download and Install the Commons Collections Binary Distribution

* Download a binary distribution of Version 1.0 or later from:

  http://jakarta.apache.org/builds/jakarta-commons/release/commons-collections

  On a Windows platform, you will need:
    commons-collections-X.Y.zip

  On a Unix platform, you will need:
    commons-collections-X.Y.tar.gz

* Unpack the binary distribution into a convenient location so that the
  distribution resides in its own directory.

(9) Download and Install the Commons Digester Binary Distribution

* Download a binary distribution of Version 1.1.1 or later from:

  http://jakarta.apache.org/builds/jakarta-commons/release/commons-digester

  On a Windows platform, you will need:
    commons-digester-X.Y.zip

  On a Unix platform, you will need:
    commons-digester-X.Y.tar.gz

* Unpack the binary distribution into a convenient location so that the
  distribution resides in its own directory.

(10) Download and Install the Commons Logging Binary Distribution

* Download a binary distribution (nightly build for now) from:

  http://jakarta.apache.org/builds/jakarta-commons/nightly/commons-logging

  On a Windows platform, you will need:
    commons-logging-YYYYMMDD.zip

  On a Unix platform, you will need:

commons-logging-YYYYMMDD.tar.gz

* Unpack the binary distribution into a convenient location so that the distribution resides in its own directory.

(11) Download and Install the Regexp Binary Distribution

* Download a binary distribution of Version 1.2 or later from:

http://jakarta.apache.org/builds/jakarta-regexp/release/

On a Windows platform, you will need:
jakarta-regexp-X.Y.zip

On a Unix platform, you will need:
jakarta-regexp-X.Y.tar.gz

* Unpack the binary distribution into a convenient location so that the distribution resides in its own directory.

(12) Steps (13)-(25) are optional, but are necessary to build a complete binary distribution of Tomcat 4.0. Set the "full.dist" property to "on" in the build.properties file (see step (26)) to build a complete distribution. Regular contributors to Tomcat are encouraged to use the complete build option.

(13) Download and Install JDBC Optional Package API Binary Distribution

* Download the JDBC Optional Pacakge API package (version 2.0) from:

http://java.sun.com/products/jdbc/download.html

* Place the jar in a convenient location.

(14) Download and Install an implementation of the JMX 1.0 specification. This can be either MX4J (http://mx4j.sourceforge.net) or Sun JMX 1.0 Reference Implementation.

NOTE: This step is only required if you wish to build the Config/Admin web application.

NOTE: The Tomcat binaries are distrbuted with MX4J.

* Download MX4J (version 1.0 or later) from

  http://sourceforge.net/project/showfiles.php?group_id=47745

* Alternately, download the JMX Instrumentation and Agent Reference
  Implementation (version 1.0 or later) from

  http://java.sun.com/products/JavaManagement/download.html

* Unpack MX4J or the reference implementation into a convenient location
  so that it resides in its own subdirectory.

(15) Download and Install the Java Activation Framework 1.0.1

* Download the Java Activation Framework package (version 1.0.1 or later) from

  http://java.sun.com/products/javabeans/glasgow/jaf.html

* Unpack the package into a convenient location so that it
  resides in its own subdirectory.

(16) Download and Install JavaMail 1.2

* Download the JavaMail package (version 1.2 or later) from

  http://java.sun.com/products/javamail/index.html

* Unpack the package into a convenient location so that
  it resides in its own subdirectory.

(17) Download and Install the JSSE 1.0.2 Reference Implementation

* Download the Java Secure Sockets Extension (JSSE) package,
  (version 1.0.2 or later) from

  http://java.sun.com/products/jsse/

* Unpack the reference implementation into a convenient location so that
  it resides in its own subdirectory.

(18) Download and Install the Java Transaction APIs

* Download the Java Transaction API (JTA) package (version 1.0.1) from:

    http://java.sun.com/products/jta/

* Unpack the package into a convenient location so that it resides in its
  own subdirectory.

(19) Download and Install the Struts Binary Distribution

* Download a binary distribution of Struts 1.0.1 from:

    http://jakarta.apache.org/builds/jakarta-struts/release/v1.0.1/

  On a Windows platform, you will need:
    jakarta-struts-1.0.1.zip

  On a Unix platform, you will need:
    jakarta-struts-1.0.1.tar.gz

* Unpack the binary distribution into a convenient location so that the
  distribution resides in its own directory.

(20) Download and Install the Tyrex Data Source Package

NOTE: This step is only required if you wish to build the Tyrex connection pool implementation for JNDI-accessed data sources.

* Download the Tyrex JAR or release (version 1.0) from:

    http://tyrex.exolab.org/download.html

* Unpack the package into a convenient location so that it resides in its
  own subdirectory.

(21) Download and Install the JUnit Testing Package (OPTIONAL)

NOTE: This step is only required if you wish to build and execute the unit tests that are part of the Tomcat 4.0 source base.

* Download the JUnit unit test package (version 3.7 or later) from:

    http://www.junit.org/

* Unpack the package into a convenient location so that it resides in its
  own subdirectory.

(22) Download and Install the Commons Modeler Binary Distribution

NOTE: This step is only required if you wish to build the Config/Admin
web application.

* Download a binary distribution of version 20020117 or later from:

    http://jakarta.apache.org/builds/jakarta-commons/nightly/commons-modeler

  On a Windows platform, you will need:
    commons-modeler-YYYYMMDD.zip

  On a Unix platform, you will need:
    commons-modeler-YYYYMMDD.tar.gz

* Unpack the binary distribution into a convenient location so that the
  distribution resides in its own directory.

(23) Download and Install the Commons DBCP Binary Distribution

NOTE: This step is only required if you wish to use the database JDBC data
source factory.

* Download a binary distribution of version 20011030 or later from:

    http://jakarta.apache.org/builds/jakarta-commons/nightly/commons-dbcp

  On a Windows platform, you will need:
    commons-dbcp-YYYYMMDD.zip

  On a Unix platform, you will need:
    commons-dbcp-YYYYMMDD.tar.gz

* Unpack the binary distribution into a convenient location so that the
  distribution resides in its own directory.

(24) Download and Install the Commons Pool Binary Distribution

NOTE: This step is only required if you wish to use the database JDBC data
source factory.

* Download a binary distribution of version 1.0 or later from:

    http://jakarta.apache.org/builds/jakarta-commons/release/commons-pool/v1.0/

  On a Windows platform, you will need:
    commons-pool-1.0.zip

  On a Unix platform, you will need:
    commons-pool-1.0.tar.gz

* Unpack the binary distribution into a convenient location so that the
  distribution resides in its own directory.

(25) Download and Install the Commons Daemon Binary Distribution

NOTE: This step is only required if you wish to build the classes needed to
run Tomcat as a native operating system daemon using commons-daemon.

* Download a binary distribution of version 20020730 or later from:

    http://jakarta.apache.org/builds/jakarta-commons/nightly/commons-daemon

  On a Windows platform, you will need:
    commons-daemon-YYYYMMDD.zip

  On a Unix platform, you will need:
    commons-daemon-YYYYMMDD.tar.gz

* Unpack the binary distribution into a convenient location so that the
  distribution resides in its own directory.

(26) Customize Build Properties For This Subproject

Most Jakarta subprojects allow you to customize Ant properties (with default
values defined in the "build.xml" file. This is done by creating a text file
named "build.properties" in the source distribution directory (for property
definitions local to this subproject) and/or your user home directory (for
property definitions shared across subprojects). You can use the included
"build.properties.sample" file as a starting point for this.

Tomcat has many external dependencies that are satisfied by configuring
appropriate values in your build.properties file. The easiest
way to satisfy these dependencies is to copy the "build.properties.sample"

file (in the top-level Tomcat source directory) to "build.properties", and
then edit it to suit your environment. On Unix, this would be done as:

```
cd ${tomcat.source}
cp build.properties.sample build.properties
emacs build.properties
```

NOTE: Be *sure* that you do not check "build.properties" in to the CVS
repository. This file is local to your own development environment, and
each developer will have their own version.

(27) Build A Binary Distribution

Open a command line shell, and issue the following commands:

```
cd ${tomcat.source}
ant -projecthelp
```

If everything is installed correctly, you should see a list of the Ant
"targets" that represent different commands you might wish to build.

You can check that all dependencies are correctly installed by using
the following commands:

```
cd ${tomcat.source}
ant detect
```

By convention, the "dist" target creates a complete binary distribution. To
execute it, type the following commands:

```
cd ${tomcat.source}
ant dist
```

This will create a complete binary distribution of the subproject (equivalent
in structure to the corresponding binary distribution downloadable from the
Jakarta web site), in the "${tomcat.source}/dist" directory. It will have
the contents described in the corresponding "README.txt" file.

See the document RUNNING.txt for instructions on how to start up and shut down
the servlet/JSP container.

Links

- Docs Home
- Config Ref. Home

Top Level Elements

- Server
- Service

Connectors

- JTC Connectors
- Coyote HTTP/1.1
- Coyote JK 2
- HTTP/1.1
- JK
- Webapp

Containers

- Context
- Engine
- Host

Nested Components

- Default Context
- Global Resources
- Loader
- Logger
- Manager
- Realm
- Resources
- Valve

Server Configuration Reference

print-friendly version

Overview

Overview

This manual contains reference information about all of the configuration directives that can be included in a `conf/server.xml` file to configure the behavior of the Tomcat 4 servlet/JSP container. It does not attempt to describe which configuration directives should be used to perform specific tasks - for that, see the various *HOW-TO* documents on the main index page.

The configuration element descriptions are organized into the following major categories:

- **Top Level Elements** - `<Server>` is the root element of the entire configuration file, while `<Service>` represents a group of Connectors that is associated with an Engine.
- **Connectors** - Represent the interface between external clients sending requests to (and receiving responses from) a particular Service.
- **Containers** - Represent components whose function is to process incoming requests, and create the corresponding responses. An Engine handles all requests for a Service, a Host handles all requests for a particular virtual host, and a Context handles all requests for a specific web application.
- **Nested Components** - Represent elements that can be nested inside the element for a Container. Some elements can be nested inside any Container, while others can only be nested inside a Context.

For each element, the corresponding documentation follows this general outline:

- **Introduction** - Overall description of this particular component. There will be a corresponding Java *interface* (in the `org.apache.catalina` pacakge) that is implemented by one or more standard implementations.
- **Attributes** - The set of attributes that are legal for this element. Generally, this will be subdivided into *Common* attributes that are supported by all implementations of the corresponding Java interface, and *Standard Implementation* attributes that are specific to a particular Java class that implements this interface.

The names of required attributes are **bolded**.

- **Nested Components** - Enumerates which of the *Nested Components* can be legally nested within this element.
- **Special Features** - Describes the configuration of a large variety of special features (specific to each element type) that are supported by the standard implementation of this interface.

Links

- Docs Home

Contents

- Contents
- Introduction
- Installation
- Deployment
- Source Code
- Processes
- Example App

Application Developer's Guide

print-friendly version

Table of Contents

Preface

This manual includes contributions from many members of the Tomcat Project developer community. The following authors have provided significant content:

- Craig R. McClanahan (craigmcc@apache.org)

Table of Contents

The information presented is divided into the following sections:

- **Introduction** - Briefly describes the information covered here, with links and references to other sources of information.
- **Installation** - Covers acquiring and installing the required software components to use Tomcat for web application development.
- **Deployment Organization** - Discusses the standard directory layout for a web application (defined in the Servlet API Specification), the Web Application Deployment Descriptor, and options for integration with Tomcat in your development environment.
- **Source Organization** - Describes a useful approach to organizing the source code directories for your project, and introduces the `build.xml` used by Ant to manage compilation.
- **Development Processes** - Provides brief descriptions of typical development processes utilizing the recommended deployment and source organizations.
- **Example Application** - This directory contains a very simple, but functionally complete, "Hello, World" application built according to the principles described in this manual. You can use this application to practice using the described techniques.

Links

- Docs Home

Contents

- Contents
- Introduction
- Installation
- Deployment
- Source Code
- Processes
- Example App

Application Developer's Guide

print-friendly version

Introduction

Overview

Congratulations! You've decided to (or been told to) learn how to build web applications using servlets and JSP pages, and picked the Tomcat server to use for your learning and development. But now what do you do?

This manual is a primer covering the basic steps of using Tomcat to set up a development environment, organize your source code, and then build and test your application. It does not discuss architectures or recommended coding practices for web application development, or provide in depth instructions on operating the development tools that are discussed. References to sources of additional information are included in the following subsections.

The discussion in this manual is aimed at developers who will be using a text editor along with command line tools to develop and debug their applications. As such, the recommendations are fairly generic -- but you should easily be able to apply them in either a Windows-based or Unix-based development environment. If you are utilizing an Interactive Development Environment (IDE) tool, you will need to adapt the advice given here to the details of your particular environment.

Links

The following links provide access to selected sources of online information, documentation, and software that is useful in developing web applications with Tomcat.

- http://java.sun.com/products/jsp/download.html - *JavaServer Pages (JSP) Specfication, Version 1.2*. Describes the programming environment provided by standard implementations of the JavaServer Pages (JSP) technology. In conjunction with the Servlet API Specification (see below), this document describes what a portable API page is allowed to contain. Specific information on scripting (Chapter 6), tag extensions (Chapter 7), and packaging JSP pages (Appendix A) is useful. The Javadoc API Documentation is included in the specification, and with the Tomcat download.

- http://java.sun.com/products/servlet/download.html - *Servlet API Specification, Version 2.3*. Describes the programming environment

that must be provided by all servlet containers conforming to this specification. In particular, you will need this document to understand the web application directory structure and deployment file (Chapter 9), methods of mapping request URIs to servlets (Chapter 11), container managed security (Chapter 12), and the syntax of the `web.xml` Web Application Deployment Descriptor (Chapter 13). The Javadoc API Documentation is included in the specification, and with the Tomcat download.

- http://java.sun.com/j2ee/blueprints/ - *Sun BluePrints (tm) Design Guidelines for J2EE*. Comprehensive advice and examples on application design for the Java2 Enterprise Edition (J2EE) platform, which includes servlets and JSP pages. The chapters on servlet and JSP design are useful even when your application does not require other J2EE platform components.

- **TODO** -- Add more entries here!

Links

- Docs Home

Contents

- Contents
- Introduction
- Installation
- Deployment
- Source Code
- Processes
- Example App

Application Developer's Guide

print-friendly version

Installation

Installation

In order to use Tomcat 4 for developing web applications, you must first install it (and the software it depends on). The required steps are outlined in the following subsections.

JDK

Tomcat will operate under any **Java Development Kit** (JDK) environment that provides a JDK 1.2 (also known as Java2 Standard Edition, or J2SE) or later platform. You will need a Java Development Kit, as opposed to a Java Runtime Environment, so that your servlets, other classes, and JSP pages can be compiled. Tomcat 4 has been extensively tested with JDK 1.3.1, which is recommended.

Compatible JDKs for many platforms (or links to where they can be found) are available at http://java.sun.com/j2se/.

Tomcat

Binary downloads of the **Tomcat** server are available from http://jakarta.apache.org/downloads/binindex.html. This manual assumes you are using the most recent release of Tomcat 4. Detailed instructions for downloading and installing Tomcat 4 are available here.

In the remainder of this manual, example shell scripts assume that you have set an environment variable `CATALINA_HOME` that contains the pathname to the directory in which Tomcat 4 has been installed.

Ant

Binary downloads of the **Ant** build tool are available from http://jakarta.apache.org/downloads/binindex.html. This manual assumes you are using Ant 1.4 or later. The instructions should also be compatible with later versions, but this has not been tested.

Download and install Ant from the distribution directory mentioned above. Then, add the `bin` directory of the Ant distribution to your `PATH` environment variable, following the standard practices for your operating system platform. Once you have done this, you will be able to execute the `ant` shell command directly.

CVS

Besides the required tools described above, you are strongly encouraged to download and install a *source code control* system, such as the **Concurrent Version System** (CVS), to maintain historical versions of the source files that make up your web application. Besides the server, you will also need appropriate client tools to check out source code files, and check in modified versions.

Detailed instructions for installing and using source code control applications is beyond the scope of this manual. However, CVS server and client tools for many platforms (along with documentation) can be downloaded from http://www.cvshome.org.

Links

- Docs Home

Contents

- Contents
- Introduction
- Installation
- Deployment
- Source Code
- Processes
- Example App

Application Developer's Guide

print-friendly version

Deployment

Background

Before describing how to organize your source code directories, it is useful to examine the runtime organization of a web application. Prior to the Servlet API Specification, version 2.2, there was little consistency between server platforms. However, servers that conform to the 2.2 (or later) specification are required to accept a *Web Application Archive* in a standard format, which is discussed further below.

A web application is defined as a hierarchy of directories and files in a standard layout. Such a hierarchy can be accessed in its "unpacked" form, where each directory and file exists in the filesystem separately, or in a "packed" form known as a Web ARchive, or WAR file. The former format is more useful during development, while the latter is used when you distribute your application to be installed.

The top-level directory of your web application hierarchy is also the *document root* of your application. Here, you will place the HTML files and JSP pages that comprise your application's user interface. When the system administrator deploys your application into a particular server, he or she assigns a *context path* to your application (a later section of this manual describes deployment on Tomcat). Thus, if the system administrator assigns your application to the context path `/catalog`, then a request URI referring to `/catalog/index.html` will retrieve the `index.html` file from your document root.

Standard Directory Layout

To facilitate creation of a Web Application Archive file in the required format, it is convenient to arrange the "executable" files of your web application (that is, the files that Tomcat actually uses when executing your app) in the same organization as required by the WAR format itself. To do this, you will end up with the following contents in your application's "document root" directory:

- ***.html, *.jsp, etc.** - The HTML and JSP pages, along with other files that must be visible to the client browser (such as JavaScript, stylesheet files, and images) for your application. In larger applications you may choose to divide these files into a subdirectory hierarchy, but for smaller apps, it is generally much simpler to

maintain only a single directory for these files.

- **/WEB-INF/web.xml** - The *Web Application Deployment Descriptor* for your application. This is an XML file describing the servlets and other components that make up your application, along with any initialization parameters and container-managed security constraints that you want the server to enforce for you. This file is discussed in more detail in the following subsection.

- **/WEB-INF/classes/** - This directory contains any Java class files (and associated resources) required for your application, including both servlet and non-servlet classes, that are not combined into JAR files. If your classes are organized into Java packages, you must reflect this in the directory hierarchy under `/WEB-INF/classes/`. For example, a Java class named `com.mycompany.mypackage.MyServlet` would need to be stored in a file named `/WEB-INF/classes/com/mycompany/mypackage/MyServlet.class`.

- **/WEB-INF/lib/** - This directory contains JAR files that contain Java class files (and associated resources) required for your application, such as third party class libraries or JDBC drivers.

When you install an application into Tomcat (or any other 2.2/2.3-compatible server), the classes in the `WEB-INF/classes/` directory, as well as all classes in JAR files found in the `WEB-INF/lib/` directory, are made visible to other classes within your particular web application. Thus, if you include all of the required library classes in one of these places (be sure to check licenses for redistribution rights for any third party libraries you utilize), you will simplify the installation of your web application -- no adjustment to the system class path (or installation of global library files in your server) will be necessary.

Much of this information was extracted from Chapter 9 of the Servlet API Specification, version 2.3, which you should consult for more details.

Shared Library Files

Like most servlet containers, Tomcat 4 also supports mechanisms to install library JAR files (or unpacked classes) once, and make them visible to all installed web applications (without having to be included inside the web application itself. The details of how Tomcat locates and shares such classes are described in the Class Loader HOW-TO documentation. For the purposes of our discussion, there are two locations that are commonly used within a Tomcat 4 installation for shared code:

- **$CATALINA_HOME/common/lib** - JAR files placed here are visible both to web applications and internal Tomcat code. This is a good place to put JDBC drivers that are required for both your application and internal Tomcat use (such as for a JDBCRealm).

- **$CATALINA_HOME/shared/lib** - JAR files placed here are visible to all web applications, but not to internal Tomcat code. This is the right place for shared libraries that are specific to your

application.

Out of the box, a standard Tomcat 4 installation includes a variety of pre-installed shared library files, including:

- The *JavaMail 1.2* (and associated *JavaBeans Activation Framework*) APIs, so you can write mail-enabled web applications.
- The *JDBC 2.0 Optional Package* APIs, which define things like `javax.sql.DataSource`.
- The *Servlet 2.3* and *JSP 1.2* APIs that are fundamental to writing servlets and JavaServer Pages.
- An *XML Parser* compliant with the JAXP (version 1.1) APIs, so your application can perform DOM-based or SAX-based processing of XML documents.

Web Application Deployment Descriptor

The description below uses the variable name $CATALINA_HOME to refer to the directory into which you have installed Tomcat 4, and is the base directory against which most relative paths are resolved. However, if you have configured Tomcat 4 for multiple instances by setting a CATALINA_BASE directory, you should use $CATALINA_BASE instead of $CATALINA_HOME for each of these references.

As mentioned above, the `/WEB-INF/web.xml` file contains the Web Application Deployment Descriptor for your application. As the filename extension implies, this file is an XML document, and defines everything about your application that a server needs to know (except the *context path*, which is assigned by the system administrator when the application is deployed).

The complete syntax and semantics for the deployment descriptor is defined in Chapter 13 of the Servlet API Specification, version 2.3. Over time, it is expected that development tools will be provided that create and edit the deployment descriptor for you. In the meantime, to provide a starting point, a basic web.xml file is provided. This file includes comments that describe the purpose of each included element.

NOTE - The Servlet Specification includes a Document Type Descriptor (DTD) for the web application deployment descriptor, and Tomcat 4 enforces the rules defined here when processing your application's `/WEB-INF/web.xml` file. In particular, you **must** enter your descriptor elements (such as `<filter>`, `<servlet>`, and `<servlet-mapping>` in the order defined by the DTD (see Section 13.3).

Deployment With Tomcat 4

In order to be executed, a web application must be deployed on a servlet container. This is true even during development. We will describe using Tomcat 4 to provide the execution environment. A web application can be deployed in Tomcat by one of the following approaches:

- *Copy unpacked directory hierarchy into a subdirectory in directory* `$CATALINA_HOME/webapps/`. Tomcat will assign a context path to your application based on the subdirectory name you choose. We will use this technique in the `build.xml` file that we construct, because it is the quickest and easiest approach during development. Be sure to restart Tomcat after installing or updating your application.

- *Copy the web application archive file into directory* `$CATALINA_HOME/webapps/`. When Tomcat is started, it will automatically expand the web application archive file into its unpacked form, and execute the application that way. This approach would typically be used to install an additional application, provided by a third party vendor or by your internal development staff, into an existing Tomcat installation. **NOTE** - If you use this approach, and wish to update your application later, you must both replace the web application archive file **AND** delete the expanded directory that Tomcat created, and then restart Tomcat, in order to reflect your changes.

- *Use the Tomcat 4 "Manager" web application to deploy and undeploy web applications*. Tomcat 4 includes a web application, deployed by default on context path `/manager`, that allows you to deploy and undeploy applications on a running Tomcat server without restarting it. See the administrator documentation (TODO: hyperlink) for more information on using the Manager web application.

- *Use "Manager" Ant Tasks In Your Build Script*. Tomcat 4 includes a set of custom task definitions for the `Ant` build tool that allow you to automate the execution of commands to the "Manager" web application. These tasks are used in the example build script discussed later, and are explained there.

- *Add a* `<Context>` *entry in the* `$CATALINA_HOME/conf/server.xml` *configuration file*. This approach is described briefly below, and allows you to position the document root of your web application at some point other than the `$CATALINA_HOME/webapps/` directory. You will need to restart Tomcat to have changes in this configuration file take effect. See the administrator documentation (TODO: hyperlink) for more information on configuring new Contexts in this way.

Deploying your app on other servlet containers will be specific to each container, but all containers compatible with the Servlet API Specification (version 2.2 or later) are required to accept a web application archive file. Note that other containers are **NOT** required to accept an unpacked directory structure (as Tomcat does), or to provide mechanisms for shared

library files, but these features are commonly available.

Links

- Docs Home

Contents

- Contents
- Introduction
- Installation
- Deployment
- Source Code
- Processes
- Example App

Application Developer's Guide

print-friendly version

Source Organization

Directory Structure

The description below uses the variable name $CATALINA_HOME to refer to the directory into which you have installed Tomcat 4, and is the base directory against which most relative paths are resolved. However, if you have configured Tomcat 4 for multiple instances by setting a CATALINA_BASE directory, you should use $CATALINA_BASE instead of $CATALINA_HOME for each of these references.

A key recommendation of this manual is to separate the directory hierarchy containing your source code (described in this section) from the directory hierarchy containing your deployable application (described in the preceding section). Maintaining this separation has the following advantages:

- The contents of the source directories can be more easily administered, moved, and backed up if the "executable" version of the application is not intermixed.
- Source code control is easier to manage on directories that contain only source files.
- The files that make up an installable distribution of your application are much easier to select when the deployment hierarchy is separate.

As we will see, the `ant` development tool makes the creation and processing of such directory hierarchies nearly painless.

The actual directory and file hierarchy used to contain the source code of an application can be pretty much anything you like. However, the following organization has proven to be quite generally applicable, and is expected by the example `build.xml` configuration file that is discussed below. All of these components exist under a top level *project source directory* for your application:

- **docs/** - Documentation for your application, in whatever format your development team is using.

- **src/** - Java source files that generate the servlets, beans, and other Java classes that are unique to your application. If your source code is organized in packages (**highly** recommended), the package hierarchy should be reflected as a directory structure underneath this directory.

- **web/** - The static content of your web site (HTML pages, JSP pages, JavaScript files, CSS stylesheet files, and images) that will be accessible to application clients. This directory will be the *document root* of your web application, and any subdirectory structure found here will be reflected in the request URIs required to access those files.

- **web/WEB-INF/** - The special configuration files required for your application, including the web application deployment descriptor (`web.xml`), tag library descriptors for custom tag libraries you have created, and other resource files you wish to include within your web application. Even though this directory appears to be a subdirectory of your *document root*, the Servlet Specification prohibits serving the contents of this directory (or any file it contains) directly to a client request. Therefore, this is a good place to store configuration information that is sensitive (such as database connection usernames and passwords), but is required for your application to operate successfully.

During the development process, two additional directories will be created on a temporary basis:

- **build/** - When you execute a default build (`ant`), this directory will contain an exact image of the files in the web application archive for this application. Tomcat 4 allows you to deploy an application in an unpacked directory like this, either by copying it to the `$CATALINA_HOME/webapps` directory, or by *installing* it via the "Manager" web application. The latter approach is very useful during development, and will be illustrated below.

- **dist/** - When you execute the `ant dist` target, this directory will be created. It will create an exact image of the binary distribution for your web application, including an license information, documentation, and README files that you have prepared.

Note that these two directories should **NOT** be archived in your source code control system, because they are deleted and recreated (from scratch) as needed during development. For that reason, you should not edit any source files in these directories if you want to maintain a permanent record of the changes, because the changes will be lost the next time that a build is performed.

External Dependencies

What do you do if your application requires JAR files (or other resources) from external projects or packages? A common example is that you need to include a JDBC driver in your web application, in order to operate.

Different developers take different approaches to this problem. Some will encourage checking a copy of the JAR files you depend on into the source code control archives for every application that requires those JAR files. However, this can cause significant management issues when you use the same JAR in many applications - particular when faced with a need to upgrade to a different version of that JAR file.

Therefore, this manual recommends that you **NOT** store a copy of the packages you depend on inside the source control archives of your applications. Instead, the external dependencies should be integrated as part of the process of **building** your application. In that way, you can always pick up the appropriate version of the JAR files from wherever your development system administrator has installed them, without having to worry about updating your application every time the version of the dependent JAR file is changed.

In the example Ant `build.xml` file, we will demonstrate how to define *build properties* that let you configure the locations of the files to be copied, without having to modify `build.xml` when these files change. The build properties used by a particular developer can be customized on a per-application basis, or defaulted to "standard" build properties stored in the developer's home directory.

In many cases, your development system administrator will have already installed the required JAR files into Tomcat 4's `common/lib` or `shared/lib` directories. If this has been done, you need to take no actions at all - the example `build.xml` file automatically constructs a compile classpath that includes these files.

Source Code Control

As mentioned earlier, it is highly recommended that you place all of the source files that comprise your application under the management of a source code control system like the Concurrent Version System (CVS). If you elect to do this, every directory and file in the source hierarchy should be registered and saved -- but none of the generated files. If you register binary format files (such as images or JAR libraries), be sure to indicate this to your source code control system.

We recommended (in the previous section) that you should not store the contents of the `build/` and `dist/` directories created by your development process in the source code control system. An easy way to tell

CVS to ignore these directories is to create a file named `.cvsignore` (note the leading period) in your top-level source directory, with the following contents:

```
build
dist
build.properties
```

The reason for mentioning `build.properties` here will be explained in the Processes section.

Detailed instructions for your source code control environment are beyond the scope of this manual. However, the following steps are followed when using a command-line CVS client:

- To refresh the state of your source code to that stored in the the source repository, go to your project source directory, and execute `cvs update -dP`.
- When you create a new subdirectory in the source code hierarchy, register it in CVS with a command like `cvs add {subdirname}`.
- When you first create a new source code file, navigate to the directory that contains it, and register the new file with a command like `cvs add {filename}`.
- If you no longer need a particular source code file, navigate to the containing directory and remove the file. Then, deregister it in CVS with a command like `cvs remove {filename}`.
- While you are creating, modifying, and deleting source files, changes are not yet reflected in the server repository. To save your changes in their current state, go to the project source directory and execute `cvs commit`. You will be asked to write a brief description of the changes you have just completed, which will be stored with the new version of any updated source file.

CVS, like other source code control systems, has many additional features (such as the ability to tag the files that made up a particular release, and support for multiple development branches that can later be merged). See the links and references in the Introduction for more information.

BUILD.XML Configuration File

We will be using the **ant** tool to manage the compilation of our Java source code files, and creation of the deployment hierarchy. Ant operates under the control of a build file, normally called `build.xml`, that defines the processing steps required. This file is stored in the top-level directory of your source code hierarchy, and should be checked in to your source code control system.

Like a Makefile, the `build.xml` file provides several "targets" that support optional development activities (such as creating the associated Javadoc documentation, erasing the deployment home directory so you can build your project from scratch, or creating the web application archive file so you can distribute your application. A well-constructed `build.xml` file will contain internal documentation describing the targets that are designed for use by the developer, versus those targets used internally. To ask Ant to display the project documentation, change to the directory containing the `build.xml` flie and type:

```
ant -projecthelp
```

To give you a head start, a basic build.xml file is provided that you can customize and install in the project source directory for your application. This file includes comments that describe the various targets that can be executed. Briefly, the following targets are generally provided:

- **clean** - This target deletes any existing `build` and `dist` directories, so that they can be reconstructed from scratch. This allows you to guarantee that you have not made source code modifications that will result in problems at runtime due to not recompiling all affected classes.
- **compile** - This target is used to compile any source code that has been changed since the last time compilation took place. The resulting class files are created in the `WEB-INF/classes` subdirectory of your `build` directory, exactly where the structure of a web application requires them to be. Because this command is executed so often during development, it is normally made the "default" target so that a simple `ant` command will execute it.
- **all** - This target is a short cut for running the `clean` target, followed by the `compile` target. Thus, it guarantees that you will recompile the entire application, to ensure that you have not unknowingly introduced any incompatible changes.
- **javadoc** - This target creates Javadoc API documentation for the Java classes in this web application. The example `build.xml` file assumes you want to include the API documentation with your app distribution, so it generates the docs in a subdirectory of the `dist` directory. Because you normally do not need to generate the Javadocs on every compilation, this target is usually a dependency of the `dist` target, but not of the `compile` target.

- **dist** - This target creates a distribution directory for your application, including any required documentation, the Javadocs for your Java classes, and a web application archive (WAR) file that will be delivered to system administrators who wish to install your application. Because this target also depends on the `deploy` target, the web application archive will have also picked up any external dependencies that were included at deployment time.

For interactive development and testing of your web application using Tomcat 4, the following additional targets are defined:

- **install** - Tell the currently running Tomcat 4 to make the application you are developing immediately available for execution and testing. This action does not require Tomcat 4 to be restarted, but it is also not remembered after Tomcat is restarted the next time.

- **reload** - Once the application is installed, you can continue to make changes and recompile using the `compile` target. Tomcat 4 will automatically recognize changes made to JSP pages, but not to servlet or JavaBean classes - this command will tell Tomcat to restart the currently installed application so that such changes are recognized.

- **remove** - When you have completed your development and testing activities, you can optionally tell Tomcat 4 to remove this application from service.

Using the development and testing targets requires some additional one-time setup that is described on the next page.

Links

- Docs Home

Contents

Application Developer's Guide

print-friendly version

Development Processes

Development Processes

Although application development can take many forms, this manual proposes a fairly generic process for creating web applications using Tomcat. The following sections highlight the commands and tasks that you, as the developer of the code, will perform. The same basic approach works when you have multiple programmers involved, as long as you have an appropriate source code control system and internal team rules about who is working on what parts of the application at any given time.

The task descriptions below assume that you will be using CVS for source code control, and that you have already configured access to the appropriate CVS repository. Instructions for doing this are beyond the scope of this manual. If you are using a different source code control environment, you will need to figure out the corresponding commands for your system.

One-Time Setup of Ant and Tomcat for Development

In order to take advantage of the special Ant tasks that interact with the *Manager* web application, you need to perform the following tasks once (no matter how many web applications you plan to develop).

- *Configure the Ant custom tasks*. The implementation code for the Ant custom tasks is in a JAR file named `$CATALINA_HOME/server/lib/catalina-ant.jar`, which must be copied in to the `lib` directory of your Ant installation.
- *Define one or more Tomcat users*. The *Manager* web application runs under a security constraint that requires a user to be logged in, and have the security role `manager` assigned to him or her. How such users are defined depends on which Realm you have configured in Tomcat's `conf/server.xml` file -- see the Realm Configuration HOW-TO for more information. You may define any number of users (with any username and password that you like) with the `manager` role.

Create Project Source Code Directory

The first step is to create a new project source directory, and customize the `build.xml` and `build.properties` files you will be using. The directory structure is described in the previous section, or you can use the sample application as a starting point.

Create your project source directory, and define it within your CVS repository. This might be done by a series of commands like this, where `{project}` is the name under which your project should be stored in the CVS repository, and {username} is your login username:

```
cd {my home directory}
mkdir myapp     <-- Assumed "project source directory"
cd myapp
mkdir docs
mkdir src
mkdir web
mkdir web/WEB-INF
cvs import -m "Initial Project Creation" {project} \
        {username} start
```

Now, to verify that it was created correctly in CVS, we will perform a checkout of the new project:

```
cd ..
mv myapp myapp.bu
cvs checkout {project}
```

Next, you will need to create and check in an initial version of the `build.xml` script to be used for development. For getting started quickly and easily, base your `build.xml` on the basic build.xml file, included with this manual, or code it from scratch.

```
cd {my home directory}
cd myapp
emacs build.xml         <-- if you want a real editor :-)
cvs add build.xml
cvs commit
```

Until you perform the CVS commit, your changes are local to your own development directory. Committing makes those changes visible to other developers on your team that are sharing the same CVS repository.

The next step is to customize the Ant *properties* that are named in the `build.xml` script. This is done by creating a file named `build.properties` in your project's top-level directory. The supported properties are listed in the comments inside the sample `build.xml` script. At a minimum, you will generally need to define the `catalina.home` property defining where Tomcat 4 is installed, and the manager application username and password. You might end up with something like this:

```
# Context path to install this application on
app.path=/hello

# Tomcat 4 installation directory
catalina.home=/usr/local/jakarta-tomcat-4.0

# Manager webapp username and password
manager.username=myusername
manager.password=mypassword
```

In general, you will **not** want to check the `build.properties` file in to the CVS repository, because it is unique to each developer's environment.

Now, create the initial version of the web application deployment descriptor. You can base `web.xml` on the basic web.xml file, or code it from scratch.

```
cd {my home directory}
cd myapp/web/WEB-INF
emacs web.xml
cvs add web.xml
cvs commit
```

Edit Source Code and Pages

The edit/build/test tasks will generally be your most common activities during development and maintenance. The following general principles apply. As described in Source Organization, newly created source files should be located in the appropriate subdirectory, under your project source directory.

Whenever you wish to refresh your development directory to reflect the work performed by other developers, you will ask CVS to do it for you:

```
```

```
cd {my home directory}
cd myapp
cvs update -dP
```

To create a new file, go to the appropriate directory, create the file, and register it with CVS. When you are satisfied with it's contents (after building and testing is successful), commit the new file to the repository. For example, to create a new JSP page:

```
cd {my home directory}
cd myapp/web            <-- Ultimate destination is document root
emacs mypage.jsp
cvs add mypage.jsp
... build and test the application ...
cvs commit
```

Java source code that is defined in packages must be organized in a directory hierarchy (under the **src/** subdirectory) that matches the package names. For example, a Java class named `com.mycompany.mypackage.MyClass.java` should be stored in file `src/com/mycompany/mypackage/MyClass.java`. Whenever you create a new subdirectory, don't forget to register it with CVS.

To edit an existing source file, you will generally just start editing and testing, then commit the changed file when everything works. Although CVS can be configured to required you to "check out" or "lock" a file you are going to be modifying, this is generally not used.

Build the Web Application

When you are ready to compile the application, issue the following commands (generally, you will want a shell window open that is set to the project source directory, so that only the last command is needed):

```
cd {my home directory}
cd myapp                <-- Normally leave a window open here
ant
```

The Ant tool will be execute the default "compile" target in your `build.xml` file, which will compile any new or updated Java code. If this is the first time you compile after a "build clean", it will cause everything to be recompiled.

To force the recompilation of your entire application, do this instead:

```
cd {my home directory}
cd myapp
ant all
```

This is a very good habit immediately before checking in changes, to make sure that you have not introduced any subtle problems that Javac's conditional checking did not catch.

Test Your Web Application

To test your application, you will want to install it under Tomcat. The quickest way to do that is to use the custom Ant tasks that are included in the sample `build.xml` script. Using these commands might follow a pattern like this:

- *Start Tomcat 4 if needed*. If Tomcat 4 is not already running, you will need to start it in the usual way.
- *Compile your application*. Use the `ant compile` command (or just `ant`, since this is the default). Make sure that there are no compilation errors.
- *Install the application*. Use the `ant install` command. This tells Tomcat to immediately start running your app on the context path defined in the `app.path` build

property. Tomcat does **NOT** have to be restarted for this to take effect.

- *Test the application*. Using your browser or other testing tools, test the functionality of your application.
- *Modify and rebuild as needed*. As you discover that changes are required, make those changes in the original **source** files, not in the output build directory, and re-issue the `ant compile` command. This ensures that your changes will be available to be saved (via `cvs commit`) later on -- the output build directory is deleted and recreated as necessary.
- *Reload the application*. Tomcat will recognize changes in JSP pages automatically, but it will continue to use the old versions of any servlet or JavaBean classes until the application is reloaded. You can trigger this by executing the `ant reload` command.
- *Remove the application when you re done*. When you are through working on this application, you can remove it from live execution by running the `ant remove` command.

Do not forget to commit your changes to the source code repository when you have completed your testing!

Creating a Release

When you are through adding new functionality, and you've tested everything (you DO test, don't you :-), it is time to create the distributable version of your web application that can be deployed on the production server. The following general steps are required:

- Issue the command `ant all` from the project source directory, to rebuild everything from scratch one last time.
- Use the `cvs tag` command to create an identifier for all of the source files utilized to create this release. This allows you to reliably reconstruct a release (from sources) at a later time.
- Issue the command `ant dist` to create a distributable web application archive (WAR) file, as well as a JAR file containing the corresponding source code.
- Package the contents of the `dist` directory using the **tar** or **zip** utility, according to the standard release procedures used by your organization.

Index of /tomcat/tomcat-4.1-doc/appdev/sample

Name	Last modified	Size	Description
Parent Directory		-	
build.xml	21-Mar-2003 21:46	16K	
docs/	21-Mar-2003 21:46	-	
src/	21-Mar-2003 21:46	-	
web/	21-Mar-2003 21:46	-	

Apache/2.0.48-dev (Unix) Server at jakarta.apache.org Port 80