What is ASF?

Established in 1999, all-volunteer The Apache Software Foundation (ASF) oversees more than seventy leading Open Source projects, including Apache HTTP Server — the world's most popular Web server software. Through The ASF's meritocratic process known as "The Apache Way," more than 300 individual Members and 2,000 Committers successfully collaborate to develop freely available enterprise-grade software, benefiting millions of users worldwide.

The Apache Software Foundation
www.apache.org

About Tomcat 7

Apache Tomcat (Jakarta Tomcat or simply Tomcat) is an open source servlet container developed by the Apache Software Foundation. Tomcat implements the Java Servlet and the JavaServer Pages (JSP) specifications, and provides a "pure Java" HTTP web server environment for Java code to run.

Tomcat should not be confused with the Apache web server, which is a C implementation of an HTTP web server; these two web servers are not bundled together, although they are frequently used together as part of a server application stack. Apache Tomcat includes tools for configuration and management, but can also be configured by editing XML configuration files.

ISBN 978-1-59682-271-9
90000
9 781596 822719

Published by Fultus Corporation
Fultus™
www.fultus.com

Apache Tomcat 7 User Guide

The Apache Software Foundation (ASF)
www.apache.org

Apache Tomcat 7

User Guide

By The Apache Software Foundation

Linbrary™ - Linux Documentation Library
www.linbrary.com

LAC™ Linbrary Advertising Club

Apache Tomcat 7

User Guide

by

The Apache Software Foundation

❖

Fultus™ Books

Apache Tomcat 7
User Guide

by

The Apache Software Foundation

ISBN-10: 1-59682-271-6
ISBN-13: 978-1-59682-271-9

Cover design and book layout by Fultus Corporation

Fultus Corporation is not affiliated with The Apache Software Foundation

Published by Fultus Corporation

Publisher Web: *www.fultus.com*
Linbrary - Linux Library: *www.linbrary.com*
Online Bookstore: *store.fultus.com*
email: *production@fultus.com*

Table of Contents

Abstract

Welcome to the *Apache Tomcat 7.0 User Guide*! Apache Tomcat version 7.0 implements the Servlet 3.0 and JavaServer Pages 2.2 specifications from the *Java Community Process*[1], and includes many additional features that make it a useful platform for developing and deploying web applications and web services.

[1] *http://www.jcp.org/*

Chapter 1. Introduction

1.1. 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.

1.2. 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*[1] or *JSP*[2] specifications.

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

1.3. 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 $CATALINA_HOME/README.txt file" we mean to look at the README.txt file at the root of your Tomcat install. Optionally, Tomcat may be configured for multiple instances by defining **$CATALINA_BASE** for each instance. If multiple instances are not configured, **$CATALINA_BASE** is the same as **$CATALINA_HOME**.

These are some of the key tomcat directories:

- **/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.

[1] *http://java.sun.com/products/servlet/*
[2] *http://java.sun.com/products/jsp/*

- **/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.

1.4. 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.

1.5. 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 7 is a new release of Tomcat, keep in mind that some of the issues and solutions vary between the major versions of Tomcat (6.x versus 7.x). As you search around the web, there will be some documentation that is not relevant to Tomcat 7, but 6.x, 5.x or earlier versions. Doing 3.x or 4.x things to 7 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!
- *Tomcat FAQ*[3]
- *Tomcat WIKI*[4]
- Tomcat FAQ at *jGuru*[5]
- Tomcat mailing list archives - numerous sites archive the Tomcat mailing lists. Since the links change over time, clicking here will search *Google*[6].
- The TOMCAT-USER mailing list, which you can subscribe to *here*[7]. If you don't get a reply, then there's a good chance that your question was probably answered in the

[3] *http://wiki.apache.org/tomcat/FAQ*
[4] *http://wiki.apache.org/tomcat/*
[5] *http://www.jguru.com/faq/home.jsp?topic=Tomcat*
[6] *http://www.google.com/search?q=tomcat+mailing+list+archives*

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*[8]. 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.

[7] *http://tomcat.apache.org/lists.html*
[8] *http://tomcat.apache.org/lists.html*

Chapter 2.
Tomcat Setup

2.1. Introduction

This document introduces several ways to set up Tomcat for running on different platforms. Please note that some advanced setup issues are not covered here: the full distribution (ZIP file or tarball) includes a file called RUNNING.txt which discusses these issues. We encourage you to refer to it if the information below does not answer some of your questions.

2.2. Windows

Installing Tomcat on Windows can be done easily using the Windows installer. Its interface and functionality is similar to other wizard based installers, with only a few items of interest.

- **Installation as a service**: Tomcat will be installed as a Windows NT/2k/XP service no matter what setting is selected. Using the checkbox on the component page sets the service as "auto" startup, so that Tomcat is automatically started when Windows starts. For optimal security, the service should be run as a separate user, with reduced permissions (see the Windows Services administration tool and its documentation).
- **Java location**: The installer will use the registry or the JAVA_HOME environment variable to determine the base path of a Java SE 6 JRE.
- **Tray icon**: When Tomcat is run as a service, there will not be any tray icon present when Tomcat is running. Note that when choosing to run Tomcat at the end of installation, the tray icon will be used even if Tomcat was installed as a service.
- Refer to the *Windows service HOW-TO* (see Chapter 29, page 218) for information on how to manage Tomcat as Windows NT service.

The installer will create shortcuts allowing starting and configuring Tomcat. It is important to note that the Tomcat administration web application can only be used when Tomcat is running.

2.3. Unix daemon

Tomcat can be run as a daemon using the jsvc tool from the commons-daemon project. Source tarballs for jsvc are included with the Tomcat binaries, and need to be compiled. Building jsvc requires a C ANSI compiler (such as GCC), GNU Autoconf, and a JDK.

Before running the script, the `JAVA_HOME` environment variable should be set to the base path of the JDK. Alternately, when calling the `./configure` script, the path of the JDK may be specified using the `--with-java` parameter, such as `./configure --with-java=/usr/java`.

Using the following commands should result in a compiled jsvc binary, located in the `$CATALINA_HOME/bin` folder. This assumes that GNU TAR is used, and that `CATALINA_HOME` is an environment variable pointing to the base path of the Tomcat installation.

Please note that you should use the GNU make (gmake) instead of the native BSD make on FreeBSD systems.

```
    cd $CATALINA_HOME/bin
    tar xvfz commons-daemon-native.tar.gz
    cd commons-daemon-1.0.x-native-src/unix
    ./configure
    make
    cp jsvc ../..
    cd ../..
```

Tomcat can then be run as a daemon using the following commands.

```
    cd $CATALINA_HOME
    ./bin/jsvc -cp ./bin/bootstrap.jar:./bin/tomcat-juli.jar \
        -outfile ./logs/catalina.out -errfile ./logs/catalina.err \
        org.apache.catalina.startup.Bootstrap
```

You may also need to specify `-jvm server` if the JVM defaults to using a server VM rather than a client VM. This has been observed on OSX.

jsvc has other useful parameters, such as `-user` which causes it to switch to another user after the daemon initialization is complete. This allows, for example, running Tomcat as a non privileged user while still being able to use privileged ports. Note that if you use this option and start Tomcat as root, you'll need to disable the `org.apache.catalina.security.SecurityListener` check that prevents Tomcat starting when running as root.

`jsvc --help` will return the full jsvc usage information. In particular, the `-debug` option is useful to debug issues running jsvc.

The file `$CATALINA_HOME/bin/commons-daemon-1.0.x-native-src/unix/native/Tomcat5.sh` can be used as a template for starting Tomcat automatically at boot time from `/etc/init.d`. The file is currently setup for running Tomcat 5.5.x, so it will be necessary to edit it a little.

Note that the Commons-Daemon JAR file must be on your runtime classpath to run Tomcat in this manner. The Commons-Daemon JAR file is in the Class-Path entry of the bootstrap.jar manifest, but if you get a ClassNotFoundException or a NoClassDefFoundError for a Commons-Daemon class, add the Commons-Daemon JAR to the -cp argument when launching jsvc.

Chapter 3. First webapp

3.1. 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*)

3.2. Introduction

3.2.1. 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.

3.2.2. 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/ - JavaServer Pages (JSP) Specification, Version 2.0.* 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 3.0.* 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!

3.3. Installation

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

3.3.1. JDK

Tomcat 7.0 was designed to run on Java SE 6.

Compatible JDKs for many platforms (or links to where they can be found) are available at *http://www.oracle.com/technetwork/java/javase/downloads/index.html*.

3.3.2. Tomcat

Binary downloads of the **Tomcat** server are available from *http://tomcat.apache.org/*. This manual assumes you are using the most recent release of Tomcat 7. Detailed instructions for downloading and installing Tomcat are available *here* (see Chapter 2, *Tomcat Setup* page 18).

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 has been installed. Optionally, if Tomcat has been configured for multiple instances, each instance will have its own `CATALINA_BASE` configured.

3.3.3. Ant

Binary downloads of the **Ant** build tool are available from *http://ant.apache.org/*. This manual assumes you are using Ant 1.8 or later. The instructions may also be compatible with other versions, but this has not been tested.

Download and install Ant. 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.

3.3.4. 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/*.

3.4. Deployment

3.4.1. 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.

3.4.2. 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.

3.4.3. Shared Library Files

Like most servlet containers, Tomcat 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* (see Chapter 10, page 122) documentation. The location commonly used within a Tomcat installation for shared code is **$CATALINA_HOME/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 or internal Tomcat use (such as for a JDBCRealm).

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

- The *Servlet 3.0* and *JSP 2.1* APIs that are fundamental to writing servlets and JavaServer Pages.
- An *XML Parser* compliant with the JAXP (version 1.2) APIs, so your application can perform DOM-based or SAX-based processing of XML documents.

3.4.4. Shared Library Files

Like most servlet containers, Tomcat 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* (see Chapter 10, page 122) documentation. The location commonly used within a Tomcat installation for shared code is **$CATALINA_HOME/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 or internal Tomcat use (such as for a JDBCRealm).

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

- The *Servlet 3.0* and *JSP 2.1* APIs that are fundamental to writing servlets and JavaServer Pages.
- An *XML Parser* compliant with the JAXP (version 1.2) APIs, so your application can perform DOM-based or SAX-based processing of XML documents.

3.4.5. Web Application Deployment Descriptor

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*[1] 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 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).

3.4.6. Tomcat Context Descriptor

A /META-INF/context.xml file can be used to define Tomcat specific configuration options, such as loggers, data sources, session manager configuration and more. This XML file must contain one Context element, which will be considered as if it was the child of the Host element corresponding to the Host to which the The Tomcat configuration documentation contains information on the Context element.

3.4.7. Deployment With Tomcat

The description below uses the variable name $CATALINA_BASE to refer the base directory against which most relative paths are resolved. If you have not configured Tomcat for multiple instances by setting a CATALINA_BASE directory, then $CATALINA_BASE will be set to the value of $CATALINA_HOME, the directory into which you have installed Tomcat.

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 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_BASE/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.

[1] *http://tomcat.apache.org/tomcat-7.0-doc/appdev/web.xml.txt*

- *Copy the web application archive file into directory* `$CATALINA_BASE/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 "Manager" web application to deploy and undeploy web applications*. Tomcat 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 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 Tomcat deployer.
- *Use the Tomcat Deployer*. Tomcat includes a packaged tool bundling the Ant tasks, and can be used to automatically precompile JSPs which are part of the web application before deployment to the server.

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.

3.5. Source Organization

3.5.1. Directory Structure

The description below uses the variable name $CATALINA_BASE to refer the base directory against which most relative paths are resolved. If you have not configured Tomcat for multiple instances by setting a CATALINA_BASE directory, then $CATALINA_BASE will be set to the value of $CATALINA_HOME, the directory into which you have installed Tomcat.

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`, defined in the *Servlet Specification*[2]), 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:

[2] *http://java.sun.com/products/servlet*

- **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 allows you to deploy an application in an unpacked directory like this, either by copying it to the `$CATALINA_BASE/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.

3.5.1.1. 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 the `lib` directory of Tomcat. 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.

3.5.2. 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 *Development Processes* section (see Section 3.6, page 32).

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* (see Section 3.2, page 21) for more information.

3.5.3. 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` file and type:

```
ant -projecthelp
```

To give you a head start, a *basic build.xml file*[3] 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.

[3] *http://tomcat.apache.org/tomcat-7.0-doc/appdev/build.xml.txt*

- **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, the following additional targets are defined:

- **install** - Tell the currently running Tomcat to make the application you are developing immediately available for execution and testing. This action does not require Tomcat 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 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 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.

3.6. 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.

3.6.1. 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/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-script` 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* (Chapter 6, page 63) for more information. You may define any number of users (with any username and password that you like) with the `manager-script` role.

3.6.2. 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* (see Section 3.5, *Source Organization* page 27), or you can use the *Sample Application* (see Section 3.7, page 37) 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*[4], 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 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 7 installation directory
catalina.home=/usr/local/apache-tomcat-7.0

# 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*[5], or code it from scratch.

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

Note that this is only an example web.xml file. The full definition of the deployment descriptor file is in the *Servlet Specification*[6].

[4] *http://tomcat.apache.org/tomcat-7.0-doc/appdev/build.xml.txt*
[5] *http://tomcat.apache.org/tomcat-7.0-doc/appdev/web.xml.txt*

3.6.3. 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* (see Section 3.5, page 27), 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.

3.6.4. 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
```

[6] *http://java.sun.com/products/servlet*

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.

3.6.5. 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 if needed*. If Tomcat 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!

3.6.6. 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.

3.7. Sample Application

The example app has been packaged as a war file and can be downloaded *here*[7] (Note: make sure your browser doesn't change file extension or append a new one).

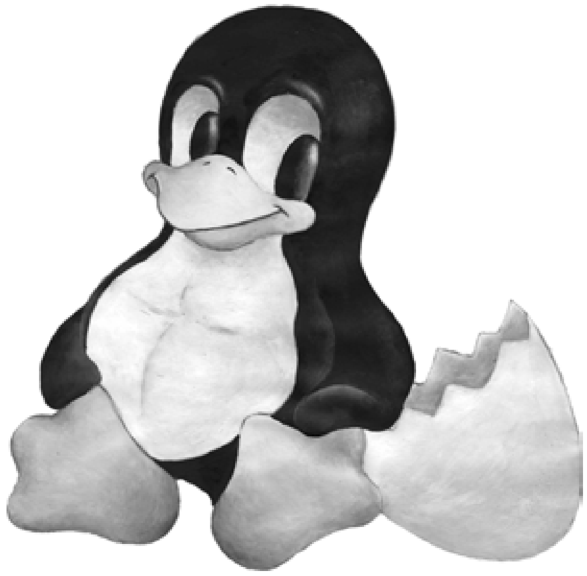
The easiest way to run this application is simply to move the war file to your **CATALINA_HOME/webapps** directory. Tomcat will automatically expand and deploy the application for you. You can view it with the following URL (assuming that you're running tomcat on port 8080 as is the default): *http://localhost:8080/sample*

If you just want to browse the contents, you can unpack the war file with the **jar** command.

```
jar -xvf sample.war
```

[7] *http://tomcat.apache.org/tomcat-7.0-doc/appdev/sample/sample.war*

Linbrary™ Advertising Club (LAC)



Linbrary™ Official Docs as a Real Books http://www.linbrary.com Linux Library

LAC™

Linbrary Advertising Club

Version	Title	Edition	ISBN- 10	ISBN- 13
Fedora 12	Fedora 12 Installation Guide	paperback	1-59682-179-5	978-1-59682-179-8
		eBook (pdf)	1-59682-184-1	978-1-59682-184-2
	Fedora 12 User Guide	paperback	1-59682-180-9	978-1-59682-180-4
		eBook (pdf)	1-59682-185-X	978-1-59682-185-9
	Fedora 12 Security Guide	paperback	1-59682-181-7	978-1-59682-181-1
		eBook (pdf)	1-59682-186-8	978-1-59682-186-6
	Fedora 12 SE Linux User Guide	paperback	1-59682-182-5	978-1-59682-182-8
		eBook (pdf)	1-59682-187-6	978-1-59682-187-3
	Fedora 12 Virtualization Guide	paperback	1-59682-183-3	978-1-59682-183-5
		eBook (pdf)	1-59682-188-4	978-1-59682-188-0
Fedora 11	Fedora 11 Installation Guide	paperback	1-59682-142-6	978-1-59682-142-2
		eBook (pdf)	1-59682-146-9	978-1-59682-146-0
	Fedora 11 User Guide	paperback	1-59682-143-4	978-1-59682-143-9
		eBook (pdf)	1-59682-147-7	978-1-59682-147-7
	Fedora 11 Security Guide	paperback	1-59682-144-2	978-1-59682-144-6
		eBook (pdf)	1-59682-148-5	978-1-59682-148-4
	Fedora 11 SE Linux User Guide	paperback	1-59682-145-0	978-1-59682-145-3
		eBook (pdf)	1-59682-149-3	978-1-59682-149-1
http://www.linbrary.com/fedora/				