Table des matières

```
<!--
  Licensed to the Apache Software Foundation (ASF) under one or more
  contributor license agreements.  See the NOTICE file distributed with
  this work for additional information regarding copyright ownership.
  The ASF licenses this file to You under the Apache License, Version 2.0
  (the "License"); you may not use this file except in compliance with
  the License.  You may obtain a copy of the License at

      http://www.apache.org/licenses/LICENSE-2.0

  Unless required by applicable law or agreed to in writing, software
  distributed under the License is distributed on an "AS IS" BASIS,
  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
  See the License for the specific language governing permissions and
  limitations under the License.
-->

<!--
     General purpose build script for web applications and web services,
     including enhanced support for deploying directly to a Tomcat
     based server.

     This build script assumes that the source code of your web application
     is organized into the following subdirectories underneath the source
     code directory from which you execute the build script:

        docs                 Static documentation files to be copied to
                             the "docs" subdirectory of your distribution.

        src                  Java source code (and associated resource
files)
                             to be compiled to the "WEB-INF/classes"
                             subdirectory of your web application.

        web                  Static HTML, JSP, and other content (such as
                             image files), including the WEB-INF
subdirectory
                             and its configuration file contents.
-->


<!-- A "project" describes a set of targets that may be requested
     when Ant is executed.  The "default" attribute defines the
     target which is executed if no specific target is requested,
     and the "basedir" attribute defines the current working directory
     from which Ant executes the requested task.  This is normally
     set to the current working directory.
-->

<project name="My Project" default="compile" basedir=".">
```

```
<!-- ===================== Property Definitions ===========================
-->

<!--

  Each of the following properties are used in the build script.
  Values for these properties are set by the first place they are
  defined, from the following list:
  * Definitions on the "ant" command line (ant -Dfoo=bar compile).
  * Definitions from a "build.properties" file in the top level
    source directory of this application.
  * Definitions from a "build.properties" file in the developer's
    home directory.
  * Default definitions in this build.xml file.

  You will note below that property values can be composed based on the
  contents of previously defined properties.  This is a powerful technique
  that helps you minimize the number of changes required when your
development
  environment is modified.  Note that property composition is allowed within
  "build.properties" files as well as in the "build.xml" script.

-->
  <property file="build.properties"/>
  <property file="${user.home}/build.properties"/>


<!-- ==================== File and Directory Names ========================
-->

<!--

  These properties generally define file and directory names (or paths) that
  affect where the build process stores its outputs.

  app.name             Base name of this application, used to
                       construct filenames and directories.
                       Defaults to "myapp".

  app.path             Context path to which this application should be
                       deployed (defaults to "/" plus the value of the
                       "app.name" property).

  app.version          Version number of this iteration of the application.

  build.home           The directory into which the "prepare" and
                       "compile" targets will generate their output.
                       Defaults to "build".
```

```
  catalina.home        The directory in which you have installed
                       a binary distribution of Tomcat.  This will
                       be used by the "deploy" target.

  dist.home            The name of the base directory in which
                       distribution files are created.
                       Defaults to "dist".

  manager.password     The login password of a user that is assigned the
                       "manager-script" role (so that he or she can execute
                       commands via the "/manager" web application)

  manager.url          The URL of the "/manager" web application on the
                       Tomcat installation to which we will deploy web
                       applications and web services.

  manager.username     The login username of a user that is assigned the
                       "manager-script" role (so that he or she can execute
                       commands via the "/manager" web application)
-->

  <property name="app.name"      value="myapp"/>
  <property name="app.path"      value="/${app.name}"/>
  <property name="app.version"   value="0.1-dev"/>
  <property name="build.home"    value="${basedir}/build"/>
  <property name="catalina.home" value="../../../.."/> <!-- UPDATE THIS! -->
  <property name="dist.home"     value="${basedir}/dist"/>
  <property name="docs.home"     value="${basedir}/docs"/>
  <property name="manager.url"
value="http://localhost:8080/manager/text"/>
  <property name="src.home"      value="${basedir}/src"/>
  <property name="web.home"      value="${basedir}/web"/>


<!-- ==================== External Dependencies ===========================
-->

<!--

  Use property values to define the locations of external JAR files on which
  your application will depend.  In general, these values will be used for
  two purposes:
  * Inclusion on the classpath that is passed to the Javac compiler
  * Being copied into the "/WEB-INF/lib" directory during execution
    of the "deploy" target.

  Because we will automatically include all of the Java classes that Tomcat
  exposes to web applications, we will not need to explicitly list any of
those
  dependencies.  You only need to worry about external dependencies for JAR
  files that you are going to include inside your "/WEB-INF/lib" directory.
```

```
-->

<!-- Dummy external dependency -->
<!--
  <property name="foo.jar"
           value="/path/to/foo.jar"/>
-->

<!-- ==================== Compilation Classpath ===========================
-->

<!--
  Rather than relying on the CLASSPATH environment variable, Ant includes
  features that makes it easy to dynamically construct the classpath you
  need for each compilation.  The example below constructs the compile
  classpath to include the servlet.jar file, as well as the other components
  that Tomcat makes available to web applications automatically, plus
anything
  that you explicitly added.
-->

  <path id="compile.classpath">
    <!-- Include all JAR files that will be included in /WEB-INF/lib -->
    <!-- *** CUSTOMIZE HERE AS REQUIRED BY YOUR APPLICATION *** -->
<!--
    <pathelement location="${foo.jar}"/>
-->

    <!-- Include all elements that Tomcat exposes to applications -->
    <fileset dir="${catalina.home}/bin">
      <include name="*.jar"/>
    </fileset>
    <pathelement location="${catalina.home}/lib"/>
    <fileset dir="${catalina.home}/lib">
      <include name="*.jar"/>
    </fileset>
  </path>

<!-- ================== Custom Ant Task Definitions =======================
-->

<!--
  These properties define custom tasks for the Ant build tool that interact
  with the "/manager" web application installed with Tomcat.  Before they
  can be successfully utilized, you must perform the following steps:

  - Copy the file "lib/catalina-ant.jar" from your Tomcat
    installation into the "lib" directory of your Ant installation.

  - Create a "build.properties" file in your application's top-level
```

```
    source directory (or your user login home directory) that defines
    appropriate values for the "manager.password", "manager.url", and
    "manager.username" properties described above.

  For more information about the Manager web application, and the
functionality
  of these tasks, see
<http://localhost:8080/tomcat-docs/manager-howto.html>.
-->

  <taskdef resource="org/apache/catalina/ant/catalina.tasks"
           classpathref="compile.classpath"/>

<!--  ==================== Compilation Control Options ====================
-->
<!--
  These properties control option settings on the Javac compiler when it
  is invoked using the <javac> task.

  compile.debug        Should compilation include the debug option?

  compile.deprecation  Should compilation include the deprecation option?

  compile.optimize     Should compilation include the optimize option?
-->
  <property name="compile.debug"       value="true"/>
  <property name="compile.deprecation" value="false"/>
  <property name="compile.optimize"    value="true"/>

<!-- ==================== All Target ======================================
-->
<!--
  The "all" target is a shortcut for running the "clean" target followed
  by the "compile" target, to force a complete recompile.
-->

  <target name="all" depends="clean,compile"
   description="Clean build and dist directories, then compile"/>

<!-- ==================== Clean Target ====================================
-->
<!--
  The "clean" target deletes any previous "build" and "dist" directory,
  so that you can be ensured the application can be built from scratch.
-->

  <target name="clean"
   description="Delete old build and dist directories">
    <delete dir="${build.home}"/>
    <delete dir="${dist.home}"/>
  </target>
```

```
<!-- ==================== Compile Target ===================================
-->

<!--
  The "compile" target transforms source files (from your "src" directory)
  into object files in the appropriate location in the build directory.
  This example assumes that you will be including your classes in an
  unpacked directory hierarchy under "/WEB-INF/classes".
-->
  <target name="compile" depends="prepare"
   description="Compile Java sources">

    <!-- Compile Java classes as necessary -->
    <mkdir    dir="${build.home}/WEB-INF/classes"/>
    <javac srcdir="${src.home}"
          destdir="${build.home}/WEB-INF/classes"
            debug="${compile.debug}"
      deprecation="${compile.deprecation}"
         optimize="${compile.optimize}">
        <classpath refid="compile.classpath"/>
    </javac>

    <!-- Copy application resources -->
    <copy  todir="${build.home}/WEB-INF/classes">
      <fileset dir="${src.home}" excludes="**/*.java"/>
    </copy>
  </target>

<!-- ==================== Dist Target =====================================
-->

<!--
  The "dist" target creates a binary distribution of your application
  in a directory structure ready to be archived in a tar.gz or zip file.
  Note that this target depends on two others:
  * "compile" so that the entire web application (including external
    dependencies) will have been assembled
  * "javadoc" so that the application Javadocs will have been created
-->

  <target name="dist" depends="compile,javadoc"
   description="Create binary distribution">

    <!-- Copy documentation subdirectories -->
    <mkdir   dir="${dist.home}/docs"/>
    <copy    todir="${dist.home}/docs">
      <fileset dir="${docs.home}"/>
    </copy>
```

```
    <!-- Create application JAR file -->
    <jar jarfile="${dist.home}/${app.name}-${app.version}.war"
         basedir="${build.home}"/>

    <!-- Copy additional files to ${dist.home} as necessary -->

  </target>

<!-- ==================== Install Target ==================================
-->

<!--
  The "install" target tells the specified Tomcat installation to
dynamically
  install this web application and make it available for execution.  It does
  *not* cause the existence of this web application to be remembered across
  Tomcat restarts; if you restart the server, you will need to re-install
all
  this web application.

  If you have already installed this application, and simply want Tomcat to
  recognize that you have updated Java classes (or the web.xml file), use
the
  "reload" target instead.

  NOTE:  This target will only succeed if it is run from the same server
that
  Tomcat is running on.

  NOTE:  This is the logical opposite of the "remove" target.
-->

  <target name="install" depends="compile"
   description="Install application to servlet container">

    <deploy url="${manager.url}"
       username="${manager.username}"
       password="${manager.password}"
           path="${app.path}"
       localWar="file://${build.home}"/>
  </target>

<!-- ==================== Javadoc Target ==================================
-->

<!--
  The "javadoc" target creates Javadoc API documentation for the Java
  classes included in your application.  Normally, this is only required
  when preparing a distribution release, but is available as a separate
  target in case the developer wants to create Javadocs independently.
-->
```

```
  <target name="javadoc" depends="compile"
   description="Create Javadoc API documentation">

    <mkdir          dir="${dist.home}/docs/api"/>
    <javadoc sourcepath="${src.home}"
                destdir="${dist.home}/docs/api"
           packagenames="*">
      <classpath refid="compile.classpath"/>
    </javadoc>
  </target>

<!-- ====================== List Target ===================================
-->

<!--
  The "list" target asks the specified Tomcat installation to list the
  currently running web applications, either loaded at startup time or
  installed dynamically.  It is useful to determine whether or not the
  application you are currently developing has been installed.
-->

  <target name="list"
   description="List installed applications on servlet container">

    <list    url="${manager.url}"
        username="${manager.username}"
        password="${manager.password}"/>
  </target>

<!-- ==================== Prepare Target ==================================
-->
<!--
  The "prepare" target is used to create the "build" destination directory,
  and copy the static contents of your web application to it.  If you need
  to copy static files from external dependencies, you can customize the
  contents of this task.

  Normally, this task is executed indirectly when needed.
-->

  <target name="prepare">
    <!-- Create build directories as needed -->
    <mkdir  dir="${build.home}"/>
    <mkdir  dir="${build.home}/WEB-INF"/>
    <mkdir  dir="${build.home}/WEB-INF/classes"/>

    <!-- Copy static content of this web application -->
    <copy todir="${build.home}">
      <fileset dir="${web.home}"/>
```

```
    </copy>

    <!-- Copy external dependencies as required -->
    <!-- *** CUSTOMIZE HERE AS REQUIRED BY YOUR APPLICATION *** -->
    <mkdir  dir="${build.home}/WEB-INF/lib"/>
<!--
    <copy todir="${build.home}/WEB-INF/lib" file="${foo.jar}"/>
-->

    <!-- Copy static files from external dependencies as needed -->
    <!-- *** CUSTOMIZE HERE AS REQUIRED BY YOUR APPLICATION *** -->
  </target>



<!-- ==================== Reload Target ===================================
-->

<!--
  The "reload" signals the specified application Tomcat to shut itself down
  and reload. This can be useful when the web application context is not
  reloadable and you have updated classes or property files in the
  /WEB-INF/classes directory or when you have added or updated jar files in
the
  /WEB-INF/lib directory.

  NOTE: The /WEB-INF/web.xml web application configuration file is not
reread
  on a reload. If you have made changes to your web.xml file you must stop
  then start the web application.
-->

  <target name="reload" depends="compile"
   description="Reload application on servlet container">

    <reload url="${manager.url}"
       username="${manager.username}"
       password="${manager.password}"
           path="${app.path}"/>
  </target>

<!-- ==================== Remove Target ===================================
-->

<!--
  The "remove" target tells the specified Tomcat installation to dynamically
  remove this web application from service.

  NOTE:  This is the logical opposite of the "install" target.
-->

  <target name="remove"
```

```
   description="Remove application on servlet container">

    <undeploy url="${manager.url}"
         username="${manager.username}"
         password="${manager.password}"
             path="${app.path}"/>
  </target>

</project>
```