Article

Guidelines, Patterns, and Code for End-to-End Java Applications

Print-friendly Version

Java Blueprints Guidelines
Project Conventions for Enterprise Applications Early Access 1

This document contains recommended conventions for structuring applications developed using Java 2 Platform, Enterprise Edition (J2EE platform) technology (hereafter referred to as J2EE applications). The J2EE 1.4 Specification (which can be downloaded from http://java.sun.com/j2ee/1.4/download.html#platformspec) indicates that certain files, such as deployment descriptors, class files, interface files, and other files, must be present as part of an application. However, this specification does not specify a recommended or required directory structure for these files. The guidelines in this document are intended to assist developers with organizing the files and directories associated with an application in a logical fashion. Organizing your applications as shown in these guidelines will make it easier to manage and maintain a project, especially when multiple developers contribute to the same project or projects are maintained during an extended lifetime. Having a predefined, consistent, standard workspace layout saves time, especially at the onset of a project.

Following these conventions will help developers establish an overall directory structure for applications. The conventions suggest where to place different types of files generally present in an application, such as `ant` files, test files, compiled code, Java and non-Java source code files, portable and application server-specific deployment descriptors, utility code, binary libraries, documentation files, configuration files, Enterprise Application archive (EAR) files, Web Application archive (WAR) files, copyright and license documents, etc. They also help developers determine how to best separate project files from files that are distributed as part of an application download.

Following these conventions will also help developers package J2EE applications into EAR or WAR files because consistent project structure is helpful for proper packaging using common `ant` targets.

These conventions, which the Java BluePrints team has followed with its applications (which can be accessed from http://java.sun.com/blueprints/code/index.html), assume that developers use the `Ant` tool for building projects. Developers using other build tools may have to make slight modifications to the conventions.

The guidelines and conventions discussed in this document include the following types of projects:

- *J2EE applications* containing a combination of Web and Enterprise JavaBeans (EJB) components that are packaged as EAR files. The conventions for J2EE applications are discussed more in Strategy for J2EE Applications.
- *Web applications* containing only Web components that are packaged as WAR files but not packaged into EAR files. The conventions for Web applications are discussed more in Strategy for Web Applications.
- *Reusable J2EE components* that are used by external projects and applications. Conventions for reusable J2EE components are discussed more in Strategy for Reusable J2EE Components and Modules.

Strategy for J2EE Applications 1.0

The following figure shows the recommended structure for projects created with J2EE technology (J2EE projects) that contain one or more applications.

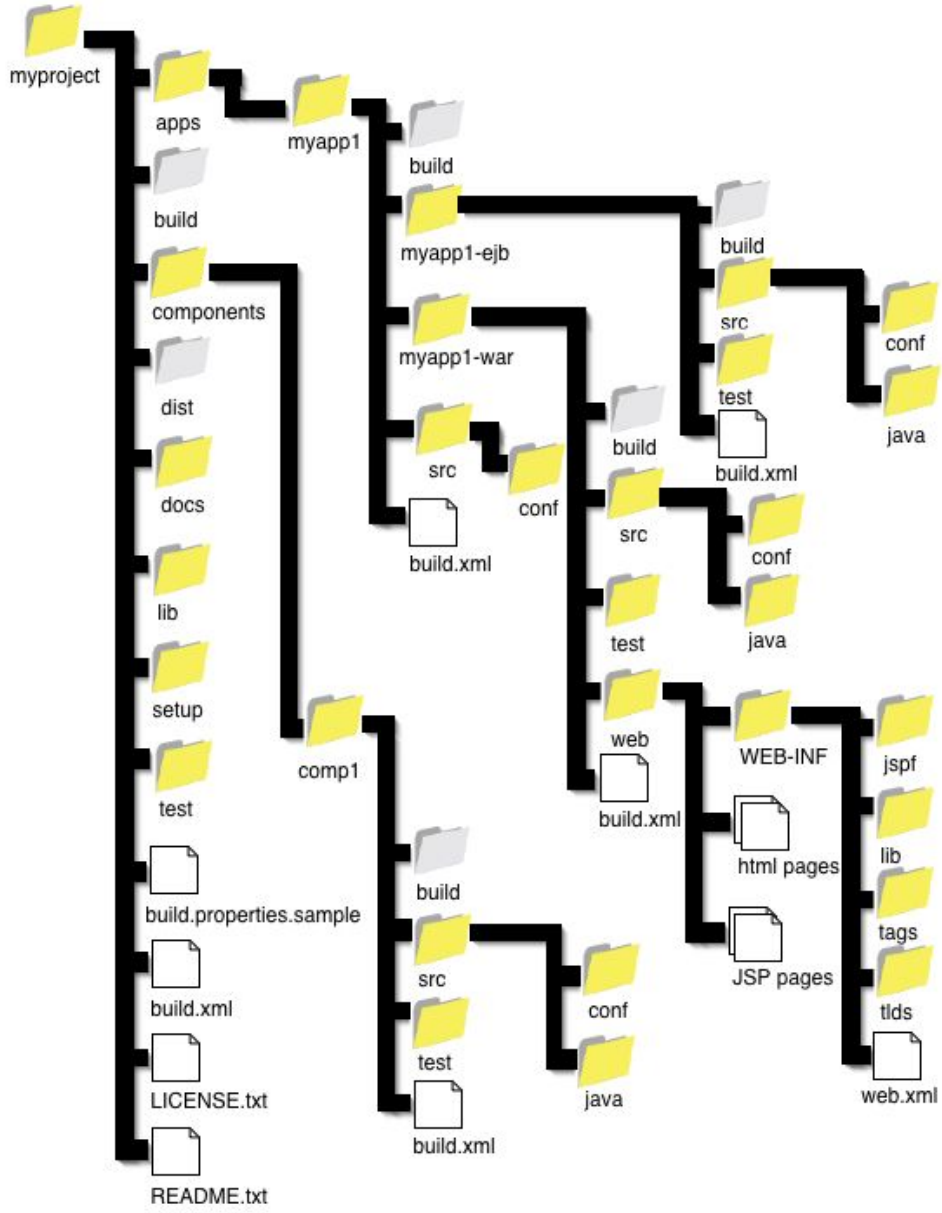


Figure 1.1: J2EE Applications: Recommended Directory Structure

The directories and their contents as shown in Figure 1.1: J2EE Applications: Recommended Directory Structure are explained in Table 1.1: J2EE Applications: Recommended Directory Contents.

Table 1.1: J2EE Applications: Recommended Directory Contents

Directory Name	Directory Contents
apps/	Base directory for applications.
build/	Created by the build tasks and used to hold project-wide build items such as compiled source, assembled modules, or files generated by the Javadoc tool. When deploying a Web module, consider building an unzipped view of the component to enable deployment as a directory structure directly to an application server.
components/	Components specific to more than one application in a project. Components used in only one application should be placed as part of that application. When creating components that are reusable across many applications, consider making those components part of another project or repository. Refer to Strategy for Reusable J2EE Components and Modules for more information on reusable components.
conf/	Configuration or other set-up files, such as Web service configuration files needed by a component or module during the build process. Includes files that are placed in a module's META-INF directory. Also includes configuration files that may require processing prior to being placed in the final module.
dist/	Created by the top-level ant dist task, structures under this directory represent the unzipped versions of the binary created by the project.
docs/	Contains all the documentation for a project, including HTML files, installation and setup files, etc.
lib/	Holds specific versions of components of external libraries used by an application. If you have multiple binaries in a project, it may be difficult for users with low bandwidth to download the application. Consider including ant targets that download the correct versions of dependent binaries at build time.
web/	Contains the static content of the resulting WAR file.
setup/	Contains files that are relevant to the environment for a project or application. This directory may contain database SQL files, ant files containing shared ant tasks, or any other files that are used to configure a container for a project or application.
web/WEB-INF/	Contains the web.xml deployment descriptor and static configuration files such as faces-config.xml. May also include vendor-specific runtime deployment descriptors, such as sun-web.xml. Generally, this directory contains files that are copied rather than changed during a build. Dynamic configuration files should be placed in the conf/ directory.
test/	The top-level test/ directory contains project-wide tests. Each individual component or module should also include a unit test, which should be placed in the src/test directory for each component or module.

Specific files recommended for J2EE applications are shown in Figure 1.1: J2EE Applications: Recommended Directory Structure and are described in Table 1.2: Function of Recommended Files.

Table 1.2: Function of Recommended Files

File Name	Function
`build.properties.sample`	Contains sample properties that are copied to a `build.properties` file in the user's home directory. These properties enable a project to be set up specific to a user's environment.
`build.xml`	Contains top-level build tasks. Usually, this file calls the `build.xml` files contained in the `apps/`, `components/`, and `test/` subfolders.
`LICENSE.txt`	The license for the project.
`README.txt`	Contains basic instructions on how to build the project.

Refer to the Java Adventure Builder Reference application at http://java.sun.com/blueprints/code/index.html#java_adventure for an example that applies this directory structure to an application.

1.1 Some Variations on the Structure

These conventions do not cover all possible directories, files, or artifacts that might be part of a project. In particular, optional files are not covered. For example, a project might rely on a `setup.xml` file, which is an `ant` file that sets up Java Naming and Directory Interface (J.N.D.I.) API references, databases, database pools, or other resources needed to deploy the project.

Some artifacts may not be present in every application, but may be necessary for other applications. These artifacts should be considered on a case-by-case basis. Some of those cases that are slight variations and some of the corner cases are discussed in the following paragraphs.

For example, you may want to include a `docs/` directory at locations other than those shown in Figure 1.1: J2EE Applications: Recommended Directory Structure. An EJB module or a Web module may be sufficiently complex so that developers feel it warrants having a its own `docs/` directory to describe how to use it, understand it, or run its tests. Or a project may contain multiple applications, and developers may include a separate `docs/` directory for each application in addition to the top level `docs/` directory.

Another issue to consider is that sometimes particular artifacts can be placed in the `setup/` directory within an application module. Setup directories may not be present in every project, and sometimes a project will have multiple `setup/` directories. There could be a top level `setup/` directory, an individual `setup/` directory for each application, and/or a `setup/` directory for individual component modules, if needed.

1.2 Extended Cases

The conventions put forth in Figure 1.1: J2EE Applications: Recommended Directory Structure can also be extended to support cases not shown. For example, you can extend these conventions to the case where an application has code that is shared by several modules within just the application. If you were to write a set of utility classes to be used by more than one module and that utility is just local to that application, where would you put it? In this case, treat this utility as another module. First, create a directory under `myapp1`, at the same level as `myapp1-ejb`, and named something like `myapp1-utility`. Next, put the utility class inside that directory. When you build your application, jar these files up and include that JAR as a module within your EAR file.

1.2.1 J2EE Application with a Single Web Application

Another example of extending the conventions is to consider how to construct a simple Web application, containing only an EAR file composed of just one WAR file. The Web application follows the Web application project structure as described in Strategy for Web Applications.

1.2.2 J2EE Project With Multiple Web Applications

Another example of extending the conventions is to consider how to construct a Web application that is an EAR file composed of multiple WAR files. What would that case look like inside this larger project structure? Each Web application follows the Web application project structure. If individual Web applications require specific environment configuration, these applications may provide a `setup/` directory specific to that application. The same would be true for a `docs/`, `lib/`, and `test/` directories. For further description on this topic, read Strategy for Web Applications.

2.0 Strategy for Web Applications

Figure 2.1: Web Applications: Recommended Directory Structure shows a recommended structure for Web projects that use J2EE Web technologies. This is useful for projects that only build a WAR file and may not plan to include it in an EAR file, or for projects that build WAR files as modules to be included in an EAR file, as shown in Strategy for J2EE Applications.

Figure 2.1: Web Applications: Recommended Directory Structure shows the simple case for an application with a single WAR file. For more complex projects containing more than one WAR file, refer to Strategy for Reusable J2EE Components and Modules.

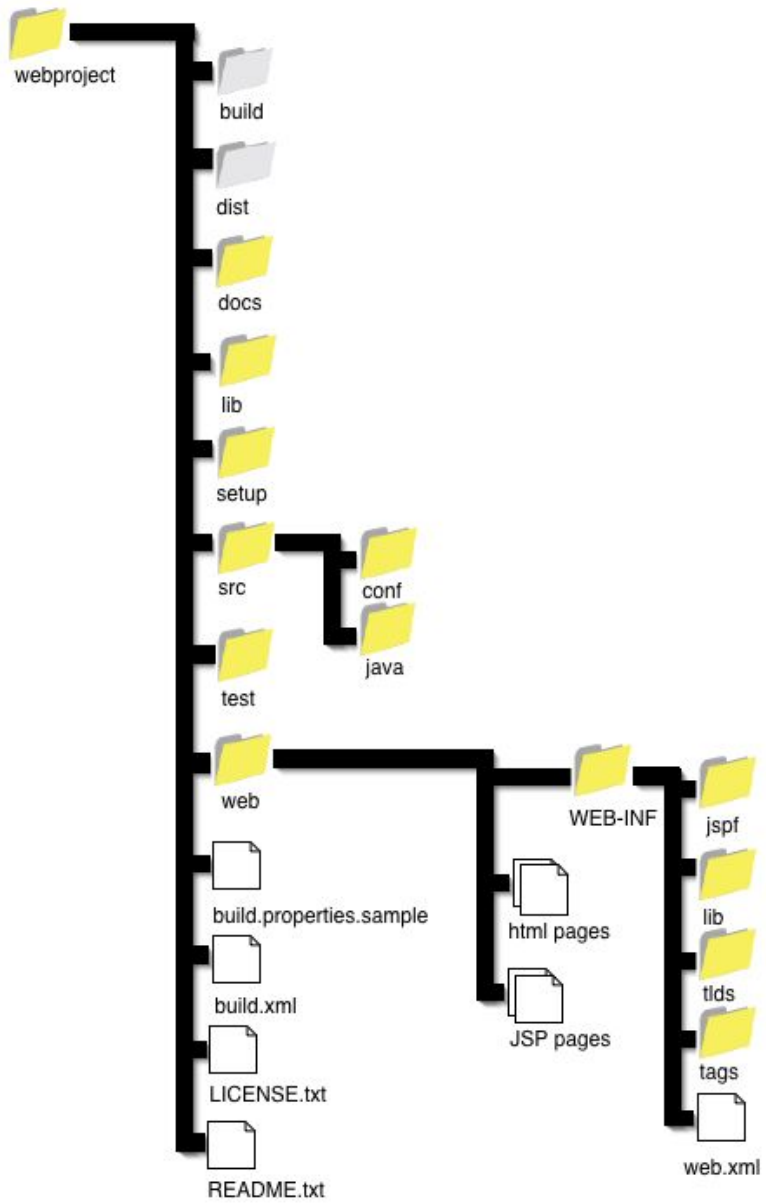


Figure 2.1: Web Applications: Recommended Directory Structure

Table 2.1: Web Applications: Recommended Directory Contents explains the various directories.
Table 1.2: Function of Recommended Files explains the recommended files within these directories.

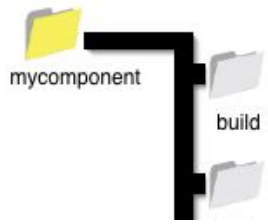
Table 2.1: Web Applications: Recommended Directory Contents

Directory Name	Directory Contents
build/	Created by the build tasks and used to hold project-wide build items such as compiled source, assembled modules, or files generated by the Javadoc tool. When deploying a Web module, consider building an unzipped view of the component to enable deployment as a directory structure directly to an application server.
conf/	Configuration or other set-up files, such as Web service configuration files needed by a component or module during the build process. Includes files that are placed in a module's META-INF directory. Also includes configuration files that may require processing prior to being placed in the final module.
dist/	Created by the top-level ant dist task, structures under this directory represent the unzipped versions of the binary created by the project.
docs/	Contains all the documentation for a project, including HTML files, installation and setup files, etc.
lib/	Holds specific versions of components of external libraries used by an application. If you have multiple binaries in a project, it may be difficult for users with low bandwidth to download the application. Consider including ant targets that download the correct versions of dependent binaries at build time.
setup/	Contains files that are relevant to the environment for a project or application. This directory may contain database SQL files, ant files containing shared tasks, or any other files that are used to configure a container for a project or application.
web/	Contains the static content of the resulting WAR file.
web/WEB-INF/	Contains the web.xml deployment descriptor and static configuration files, such as faces-config.xml. May also include vendor-specific runtime deployment descriptors, such as sun-web.xml. Generally, this directory contains files that are copied rather than changed during a build. Dynamic configuration files should be placed in the conf/ directory.
test/	The top-level test/ directory contains project-wide tests. Each individual component or module should also include a unit test, which should be placed in the src/test directory for each component or module.

3.0 Strategy for Reusable J2EE Components and Modules

Figure 3.1: Reusable J2EE Components: Recommended Directory Structure shows the recommended directory structure for developing a component or module that is delivered as a JAR file to another application. These conventions will help while designing and developing a reuseable component. Examples of such reusable components and modules include a Web framework (such as struts), a set of enterprise beans, a set of JavaServer Pages (JSP) tags, or an applet.

Figure 3.1: Reusable J2EE Components: Recommended Directory Structure



The directory structure is shown in Table 3.1: Reusable J2EE Components: Recommended Directory Contents, and the files within the directories are shown in Table 1.2: Function of Recommended Files.

Table 3.1: Reusable J2EE Components: Recommended Directory Contents

Directory Name	Directory Contents
`build/`	Created by the build tasks and used to hold project-wide build items such as compiled source, assembled modules, or files generated by the Javadoc tool. When deploying a Web module, consider building an unzipped view of the component to enable deployment as a directory structure directly to an application server.
`conf/`	Configuration or other set-up files, such as Web service configuration files, needed by a component or module during the build process. Includes files that are placed in a module's `META-INF` directory. Also includes configuration files that may require processing prior to being placed in the final module.
`dist/`	Created by the top-level `ant dist` task, structures under this directory represent the unzipped versions of the binary created by the project.
`docs/`	Contains all the documentation for a project, including HTML files, installation and setup files, etc.
`lib/`	Holds specific versions of components of external libraries used by an application. If you have multiple binaries in a project, it may be difficult for users with low bandwidth to download the application. Consider including `ant` targets that download the correct versions of dependent binaries at build time.
`test/`	The top-level `test/` directory contains project-wide tests. Each individual component or module should also include a unit test, which should be placed in the `src/test` directory for each component or module.

Refer to the Java Adventure Builder Reference application at http://java.sun.com/blueprints/code/index.html#java_adventure for an example that uses this form of the directory structure.

4.0 Related Links

- Packaging and Deployment of J2EE Applications: http://java.sun.com/blueprints/guidelines/designing_enterprise_applications_2e/deployment/deployment.html.
- Naming Conventions: http://java.sun.com/blueprints/code/namingconventions.html.
- Apache Jakarta Guidelines for Web Applications: http://jakarta.apache.org/tomcat/tomcat-4.1-doc/appdev/source.html.

5.0 Feedback

We welcome your feedback on this document and its contents. You can use either of the following methods for providing feedback to us:

- The Java Blueprints Developer Forum at http://forums.java.sun.com/forum.jsp?forum=121
- The Java Blueprints Feedback page at http://java.sun.com/blueprints/feedback.html

by Oracle, does not represent a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. It is intended for information purposes only, and may not be incorporated into any contract.