NetUI Web App Project Model

Table of contents

1. Introduction

A NetUI enabled web application consists of the same resources as a Struts, servlet, or other J2EE webapp. The elements that make a NetUI web application different are the build steps for processing annotated Java files and the JARs / resources that comprise the NetUI webapp runtime. This document discusses several topics including possible web project layouts, the Ant tasks used to build Page Flows, the JARs / resources in a NetUI web application, and the files that must be added to source control in order to commit a NetUI-enabled web project into SCM.

2. Project Layout

J2EE web projects can be structured in a nearly limitless number of ways. Virtually all webapps have both source files and web addressable content. In addition, there are a variety of configuration files and deployment descriptors that are often stored in the `WEB-INF/` directory. A fundamental difference in how web projects are structured is where the web-addressable content and the source files live. One web project model stores the source files in a sub-directory the web addressable content; another stores source files as a peer to the web addressable content. When building Page Flows, the project layout affects the Ant calls used to build the annotated Java files. Both project layouts and the Ant used to build are discussed here.

2.1. Source Files peer to Web Content Root

The classic web project layout is described by Tomcat here and has the directories containing web-addressable content and web project source in peer directories. For example, the following directory structure uses this layout and stores the Ant build file in the project's root directory:

```
fooWebProject/
    build/
    src/
        Controller.java
    web/
        page1.jsp
        page2.jsp
        WEB-INF/
            web.xml
    build.xml
    build.properties
```

When using this layout, the source files in `src/` are often built into the `build/` directory

under `WEB-INF/classes`. Page Flows can be added to this project in either the `src/` or `web/` directory. When Page Flows are added to the `src/` directory, the following Ant can be used to build them into `build/WEB-INF/classes`:

```
<import file="../../beehive-imports.xml"/>
<import file="${beehive.home}/ant/beehive-tools.xml"/>
<property file="build.properties"/>

...

<build-pageflows srcdir="src/"
                 webcontentdir="web/"
                 destdir="build/WEB-INF/classes/"
                 tempdir="build/WEB-INF/.tmpbeansrc"
                 classpathref="webapp.classpath"/>
```

While unconventional, because a Page Flow is URL addressable and "owns" its JSPs it is sometimes useful to store Page Flow files in the `web/` directory. This makes it easier to visualize the Page Flow as both the pages and the controller source file. In this case, the Ant build changes slightly:

```
<import file="../../beehive-imports.xml"/>
<import file="${beehive.home}/ant/beehive-tools.xml"/>
<property file="build.properties"/>

...

<build-pageflows srcdir="web/"
                 webcontentdir="web/"
                 destdir="build/WEB-INF/classes/"
                 tempdir="build/WEB-INF/.tmpbeansrc"
                 classpathref="webapp.classpath"/>
```

Be careful of the dependencies between the `src/` and `web/` directories when adding Page Flows to the `web/` directory as building both source roots separately can be difficult they have circular dependencies on each other.

In both of the above project layouts, the `tempdir` is used as a destination for artifacts generated by the Beehive annotation processors including both resources and Java source files. These are then compiled by the annotation processor into the classes stored in `build/WEB-INF/classes`. This behavior can be changed by tweaking the build files to build into a different temporary directory or to create a JAR for the class files. Also, the `build/` directory is often deployed to an application container during development.

2.2. Source Files in the Web Content Root

An alternate web project layout stores Java sources in the `WEB-INF/src` sub-directory. This project layout might look like:

```
fooWebProject/
    page1.jsp
    page2.jsp
    WEB-INF/
        web.xml
        src/
            Controller.java
            build.xml
            build.properties
```

When building this type of web project, classes are often generated into the `WEB-INF/classes` directory and the webapp deployed from the `fooWebProject` directory. This is different from the previous project models which build and deploy an external `build/` directory. The Ant used to build Page Flows in this project structure might appear as:

```
    <import file="../../beehive-imports.xml"/>
    <import file="${beehive.home}/ant/beehive-tools.xml"/>
    <property file="build.properties"/>

    ...

        <build-pageflows srcdir="fooWebProject/"
                         tempdir="fooWebProject/WEB-INF/.tmpbeansrc"
                         classpathref="webapp.classpath"/>
```

The difference between this `<build-pageflows>` call and the previous examples is that the `webcontentdir` and `destdir` directories are implicitly set by only using the `srcdir` attribute. This causes the web project to build directly into the `fooWebProject/` directory and to generate classes into `fooWebProject/WEB-INF/classes`.

3. Creating a new NetUI Project

A new NetUI project can be created from a Beehive distribution by running two commands to first create a NetUI-enabled web project and then copy the Beehive runtime JARs into that project.

```
cp -r <beehive-root>/samples/netui-samples <project-directory>
ant -f <beehive-root>/ant/beehive-runtime.xml
-Dwebapp.dir=<project-directory> deploy.beehive.webapp.runtime
```

This command will create a webapp using the project layout described here. This webapp is

essentially a copy of the `<beehive-root>/samples/netui-blank` web application.

4. Runtime JARs / Resources

All web applications require runtime resources. Often, these are stored in a web project's `WEB-INF/lib` directory. In order to use NetUI in a J2EE web application, a variety of JARs must be stored in this directory.

4.1. JARs

Since NetUI is built atop Struts, the Struts JARs must be present in order for the web application to function. This table lists both the Struts and Beehive JARs; all of these JARs are available as part of the Beehive distribution.

Name	JAR file	Version	Required
Beehive Controls	beehive-controls.jar	*distribution*	Yes
Beehive NetUI	beehive-netui-core.jar	*distribution*	Yes for NetUI JSP tag support; no otherwise
Beehive NetUI	beehive-netui-tags.jar	*distribution*	No
Jakarta Commons Bean Utils	commons-beanutils.jar	1.6	Yes
Jakarta Commons Codec	commons-codec-1.3.jar	1.3	Yes
Jakarta Commons Collections	commons-collections.jar	2.1.1	Yes
Jakarta Commons Digester	commons-digester.jar	1.6	Yes
Jakarta Commons Discovery	commons-discovery-0.2.	0.2	Yes
Jakarta Commons EL	commons-el.jar	1.0	Yes
Jakarta Commons File Upload	commons-fileupload.jar	1.0	Yes
Jakarta Commons Logging	commons-logging.jar	1.0.4	Yes
Jakarta Commons ORO	jakarta-oro.jar	2.0.7	Yes

Jakarta Commons Validator	commons-validator.jar	1.1.4	Yes
JSTL 1.1	jstl.jar	1.1.0-D13	Yes for JSTL tag support; no otherwise
JSTL 1.1	standard.jar	1.1.0-D13	Yes for JSTL support; no otherwise
Log4J	log4j-1.2.8.jar	1.2.8	No
Struts	struts.jar	1.2.7	Yes

Note:

For the 1.0 release, the NetUI runtime *can not* be shared between multiple web applications; the runtime for every web application must be isolated inside of its own web application classloader. This is because in some cases, NetUI caches information in statics or class instances rather than in the `ServletContext`.

4.2. Other Resources

NetUI also uses several additional XML files used to configure various NetUI and Struts sub-systems. These are detailed in the table below.

Name	Location	Required
beehive-netui-validator-rules.xml	`<beehive-root>/samples/n`	Yes
validator-rules.xml	`<beehive-root>/samples/n`	Yes
beehive-netui-config.xml	See here for more information.	No (unless modified)

Also, the NetUI runtime requires a set of `web.xml` entries to register the Page Flow servlet, filters, and mappings. In any NetUI-enabled web project, be sure that these entries are present.

4.3. NetUI-enabled Web Projects and Source Control

When adding a NetUI-enabled web project to source control, all resources marked *Required* in the JAR table and the resources table should be checked into SCM. In addition, the optional resources may be required for certain features to function correctly. If a web project uses the Beehive System Controls, those JARs should also be checked into source control.

5. Building a Web Project

When a NetUI enabled web project builds, two processing steps happen to the Page Flow

annotated Java files. The first is annotation processing which produces a Struts module config file and the second is a Java class file for the Controller class. For example, given a Page Flow in some directory:

```
foo/
    Controller.java
    page1.jsp
    page2.jsp
```

in any of the project models above, the following artifacts will be produced by the build:

```
WEB-INF/classes/
             foo/
                 Controller.class
             _pageflow/
                 struts-config-foo.xml
```

By default, the Struts module config file is placed in the `WEB-INF/.pageflow-struts-generated` directory and the Java class file is placed in `WEB-INF/classes/`. In cases where these values need to change, the Beehive Ant build macros are documented here.

6. Deploying a Web Project

Once built, a Beehive web project can be deployed to a Servlet container just as with any other J2EE web application. On Tomcat, this can be done by copying the web project directory to `$CATALINA_HOME/webapps` or by using the Tomcat deployer to deploy the webapp. See your application container's documentation for details on how to deploy web applications.