Creating a wonder app to deploy as a servlet

Warning

This is legacy information. If you are using Eclipse 3.4 and WOLips 5640 or later, servlet deployment is significantly less complicated, and described in Servlet Deployment Setup.

Introduction

WO developers have been able to deploy their applications as a WAR bundle in a J2EE container since WO 5.1, and as an independent SSDD (Servlet Single Directory Deployment) bundles since WO 5.2. There is some documentation out there already for preparing your apps for this if you use XCode, but more and more developers are ditching that in favor of the vastly superior (in my opinion) Eclipse/WOLips IDE. For more information on XCode, start here. This article is all about how to do things in Eclipse, and then as a bonus, how to create a deploy a project wonder application so that it runs in tomcat. I had a fairly miserable time figuring all of this out, but there's really not that much to it once you have all of the information in one place. Read on...

Prerequisites

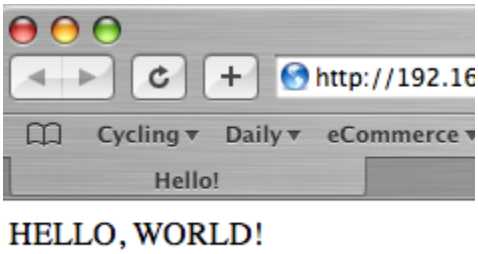
You really ought to know how to create a regular standalone webobjects application before you dive into this article. It helps if you understand the basic servlet container file structure, and you should also have the container of your choice installed. I'm using Tomcat 5.5.9. I'm also using WebObjects 5.3 and Java 1.5.

Getting Started

For an example of how this works, we're going to do the most boring little hello world app imaginable from scratch. Don't fret though, you can apply the same set of changes to any existing app and it should work. After getting Hello World working, I applied the changes described here to a fairly complex existing project full of Ajax and project wonder fanciness, and it worked just as expected.

Creating Hello World

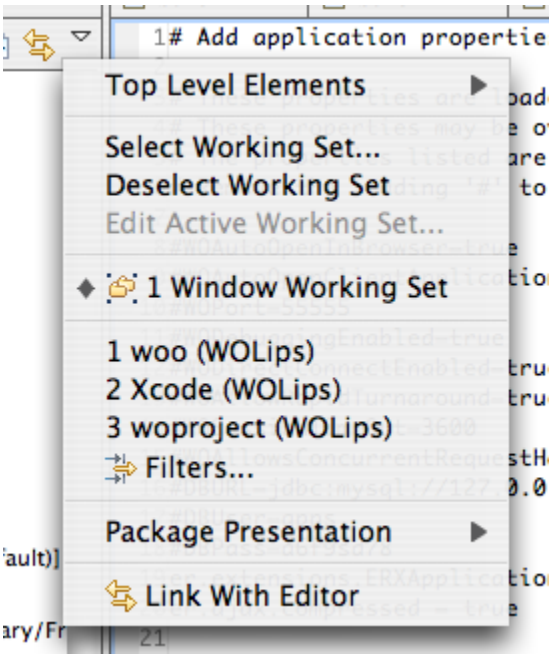
First create your hello world just like you would In this tutorial. I called my project JSPHelloWorld, and all I did after the WOLips wizard finished guiding me through the project was to edit the Main.html component to say HELLO, WORLD! About as boring as it gets, but go ahead and run your application as a WOLips WOApplication run configuration (the beauty of all this, as you'll see, is that you can develop your applications as you normally would, and in the end, you only have to think about the tomcat stuff when you're actually ready to deploy):



Now for the fun part...

There are really just a few steps you need to go through to make a plain WebObjects application deploy on Tomcat from here:

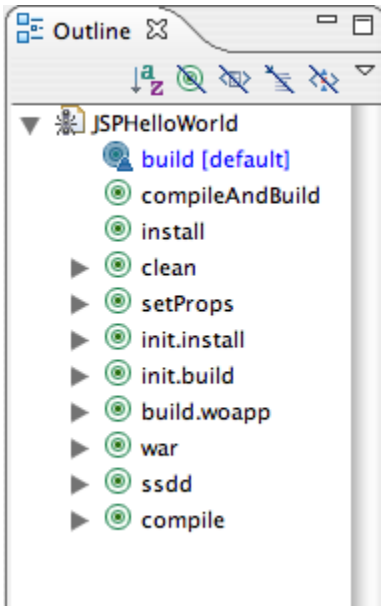
1) Open up the build.xml file for editing. It should be at the project level. If you don't see it, it's probably just hidden. To show it, click the down triangle at the top of the Package Explorer, and select the "Filters" menu:



Uncheck the item labeled build.xml (WOLips). Now you should see it in package explorer.

Note: If you are converting an existing project to deploy in a servlet container, you may have an older version of the WOLips build.xml file (this happened to me when I started playing with things, and it was all kinds of confusing). Lots of work has been done by the community in the last year to bring this up to speed. To get the recent good stuff, you can right click on the build.xml file and choose WOLips Ant Tools->Replace with latest build.xml. Take care to back up your original version somewhere first though, if you've made any modifications, because you will lose those changes.

If you are using the ant editor to edit the file, you will see a nice summary of your targets in the Outline window:



Click on the 'ssdd' target (If you are not in the and editor, you can just scroll down in the file until you see '<target name="ssdd" depends="build.woapp">').

The directions in the comments above the target are fairly self-explanatory, but here is the more verbose rundown:

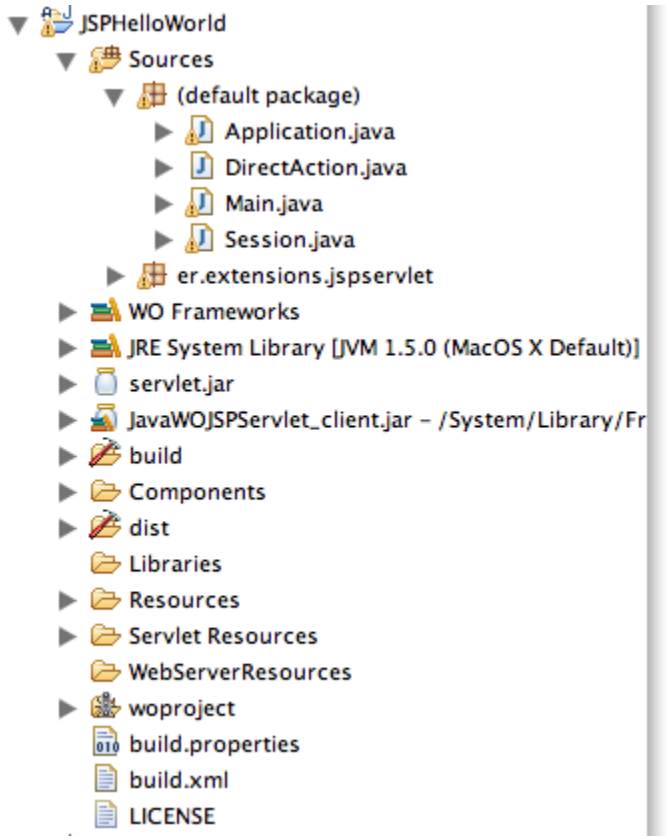
By default, the target tag for ssdd has an attribute that looks like

```
if="${never}"
```

. Just delete that, so that the target will execute next time you do an ant install.

A couple of targets up in the file, you will find the build.woapp target (<target name="build.woapp">). Within that tag, you will find several 'frameworks' tags. By default, these will have an embed="false" attribute. For each of them, change the "false" to "true". This is important because you are want a self contained thing to go in your servlet container, so everything you reference needs to be included. embed="true" makes the ant task copy all of the referenced frameworks out of /Library/Frameworks/ and /System/Library/Frameworks and into the folder that will ultimately move to your tomcat 'webapps' directory (by default, this folder will be created within your project, within a folder named 'dist'. More on this later).

At the root level of your project, you need to create a text file and name it LICENSE. Within this file, you are going to paste the contents of your WebObjects license key. For WebObjects 5.3, this can be found in the file at /System/Library/Frameworks/JavaWebObjects.framework/Resources/License.key. More on the whole license key thing here. Here's what my project directory looks like after I created the LICENSE file:



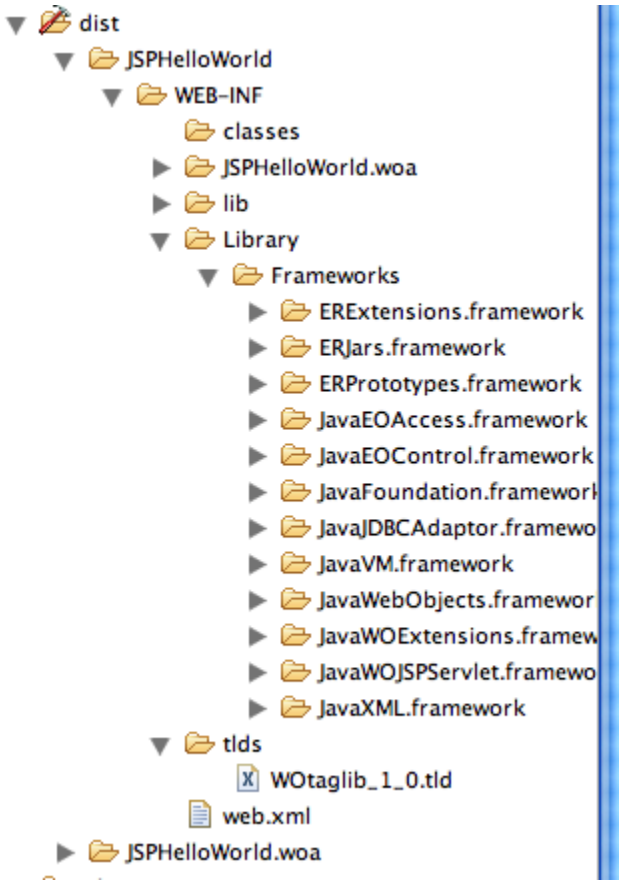
Now you need to modify your build path to include the framework that lets your webobjects application behave in the servlet world: JavaWOJSPServlet. It's in /System/Library/Frameworks. If you don't know how to modify your build path, go read this.

Now open up your build.properties file, which also resides at the root level of your project. If you don't see it, it's probably hidden from view. Open up the Filters window in package explorer (just like you did above to see the build.xml file), and uncheck build.properties (WOLips). Double click the build.properties file to edit it. There will be a line that looks like "webXML = ". Make that line read 'webXML = true' and then save the file.

Getting there....

That's the long and short of it if you don't want to include wonder in your app. All you have to do is do an ant install (right click on your project folder and choose WOLips Ant Tools->Install. A bunch of stuff will be compiled into the proper exploded WAR directory structure in a folder called 'dist' in your project. If you don't see it, it could be hidden. Open up the Filters window in package explorer again and make sure dist (WOLips) is unchecked. If you still don't see it, it could just be that package explorer hasn't picked up on it yet. Right click your project and choose 'Refresh' and it should appear.

Let's have a quick look at what's in 'dist':



There are two top-level folders. The first is the one you will copy into the webapps directory of your tomcat installation. Inside it is the WEB-INF folder containing an auto-generated web.xml file (more on this later), the bundled frameworks (because we set embed="true" in our build.xml), a tld file that you don't really need to know too much about, a lib directory that contains, as far as I can tell, every jar resource that you have installed in /Library/WebObjects /Extensions (does anyone know how to prevent the unnecessary stuff from being copied in? If so, please edit this article), and your .woa folder, which is an exact copy of the second folder under dist, JSPHelloWorld.woa, in this case.

If this were not a project wonder project (that is, if its Application and Session classes did not extend ERXApplication and ERXSession), we could copy over the JSPHelloWorld folder from dist into tomcat's webapps folder, restart tomcat, pull up a browser, type in http://localhost:8080/JSPHelloWorld /WebObjects/JSPHelloWorld.woa, and see the same boring hello world page we saw when we ran the application as a standalone WO app. If you really want to verify this, just make your Application and Session classes extend WOApplication and WOSession instead, rebuild the thing, deploy it into tomcat as described above, and have a go.

Now for Project Wonder

But SINCE we're trying to incorporate project wonder (and what sensible WebObjects developer tries to live life without project wonder?) we instead get this awesomely helpful message back:

Servlet WOServletAdaptor is currently unavailable

Man, that tells you a lot. Checking the logs (tomcat-root/logs/catalina.out) tells you little more that makes any sense. But I've done the desparate scrounging on Google, so I'll spare you the suspense: it's because Project Wonder does a lot of early intialization and patching of terrible WebObjects code, and the process of patching is broken when you run things in a servlet container. When you're running them as a standalone app, you have a static main method in your Application class that calls ERXApplication.main(), which in turn does all that cool pain saving patching. But the main method does not get called in a servlet container because things just operate differently in this world. So we have to do a few extra things. There's a pretty good thread on the whole business here.

Monkeying with the setup

Before I get started with what worked for me, I want to give a hat tip to Henrique Prange, who came up with a bunch of code that I cobbled together and modified only slightly to get things working. Without his guidance, I'd still be scratching my head and staring at my computer screen with bloodshot eyes wondering what is going on.

The gist of what we're about to do is that we're going to be writing a new ServletAdaptor, which is a subclass of WOServletAdaptor, which is a subclass of javax.servlet.http.HttpServlet. This is just a special kind of class that sits around and listens for HTTP requests, and hands them off to the right place when they come in (fairly similar to the WO HTTP adaptor). If you look at the documentation for WOServletAdaptor, it says it is not intended to be subclassed, and I think what they mean by that is that all of its methods except the constructor and one other special method are static. But we're going to subclass it anyway, and override the one non-static method (named init()) to setup the project wonder initialization stuff that usually happens when we call ERXApplication.main().

Henrique created a class called ERXServletAdaptor and it is submitted as a pending project wonder patch as we speak (or as I type, as the case may be). I could not get the code in that patch to run as is, so I have changed it just a bit. Here is my version of the class that works for me. Something very similar to this may soon be bundled with project wonder, but for now, you can just toss this class into your project's Sources folder:

ERXServletAdaptor.java

```
package er.extensions.jspservlet;

import java.lang.reflect.Method;

import javax.servlet.ServletContext;
import javax.servlet.ServletException;
import javax.servlet.UnavailableException;

import com.webobjects.jspservlet.WOServletAdaptor;

import er.extensions.appserver.ERXApplication;

/**
 * This class is just a wrapper around <code>WOServletAdaptor</code>.
 * <code>ERXServletAdaptor</code> must be used to make Wonder applications
 * compliant with WAR deployment.
 * <p>
 * This class is responsible to invoke the
 * {@link ERXApplication#setup(String[])} method before the application
 * initialization.
 *
 * @see WOServletAdaptor
 * @author <a href="mailto:hprange@moleque.com.br">Henrique Prange</a>
 */
public class ERXServletAdaptor extends WOServletAdaptor {

        /**
         * Overrides the <code>_appliationInit</code> and invoke the
         * {@link ERXApplication#setup(String[])} method before the application
         * initialization.
         *
         * @see WOServletAdaptor#_applicationInit(ServletContext)
         */

        /*
         *
         * @param servletContext
         *            The servlet context to get the application class
         * @throws UnavailableException
         *             If something wrong happens while trying to invoke the
         *             application setup method.
         */
        static void invokeApplicationSetupMethod(ServletContext servletContext) throws UnavailableException {
                ClassLoader classLoader = WOServletAdaptor.class.getClassLoader();

                try {
                        String applicationClassName = servletContext.getInitParameter("WOApplicationClass");

                        if (applicationClassName == null || "".equals(applicationClassName)) {
                                throw new UnavailableException("WOApplicationClass must be defined. Verify your
web.xml configuration.");
                        }

                        Class applicationClass = classLoader.loadClass(applicationClassName);

                        Method method = applicationClass.getMethod("setup", new Class[] { String[].class });
```

```
                        method.invoke(null, new Object[] { new String[0] });
                }
                catch (Exception e) {
                        e.printStackTrace();

                        throw new UnavailableException("Error initializing ERXServletAdaptor: " + e.
getMessage());
                }
        }

        public ERXServletAdaptor() throws ServletException {
                super();
        }

        @Override
        public void init() throws ServletException {
                ERXServletAdaptor.invokeApplicationSetupMethod(getServletContext());
                super.init();
        }

}
```

When you plop this into your application, it is not going to build. The reason is that it references classes that are bundled in jar files that aren't in your build path yet. The two files you need to add to your build path are

/System/Library/Frameworks/JavaWOJSPServlet.framework/Versions/A/WebServerResources/Java/JavaWOJSPServlet_client.jar

and

/System/Library/Frameworks/JavaWOJSPServlet.framework/Versions/A/Resources/Java/javawojspservlet.jar

I'd like to have a quiet, respectful, polite but stern word with whoever at Apple decided to put the first one in WebServerResources. Very confusing.

Now your project should build.

One last thing

That is pretty much the trick, but there is one last catch. Somehow, tomcat has to know to use your version of the ServletAdaptor, and not the WebObjects version that breaks with wonder. The container gets this information from the web.xml file that we saw magically generated for us in dist/JSPHelloWorld /WEB-INF after we ran our ant task. If you go into the dist folder and open up this file, you will see the following near the bottom:

```
  <!-- The WebObjects Servlet that interfaces between the Servlet container
       world and the WebObjects world. -->
  <servlet>
    <servlet-name>WOServletAdaptor</servlet-name>
    <servlet-class>com.webobjects.jspservlet.WOServletAdaptor</servlet-class>
    <load-on-startup>5</load-on-startup>
  </servlet>
```

For our little trick to work, we need to change the class to reference our class, so we change it to look like this:

```
  <!-- The WebObjects Servlet that interfaces between the Servlet container
       world and the WebObjects world. -->
  <servlet>
    <servlet-name>WOServletAdaptor</servlet-name>
    <servlet-class>er.extensions.jspservlet.ERXServletAdaptor</servlet-class>
    <load-on-startup>5</load-on-startup>
  </servlet>
```

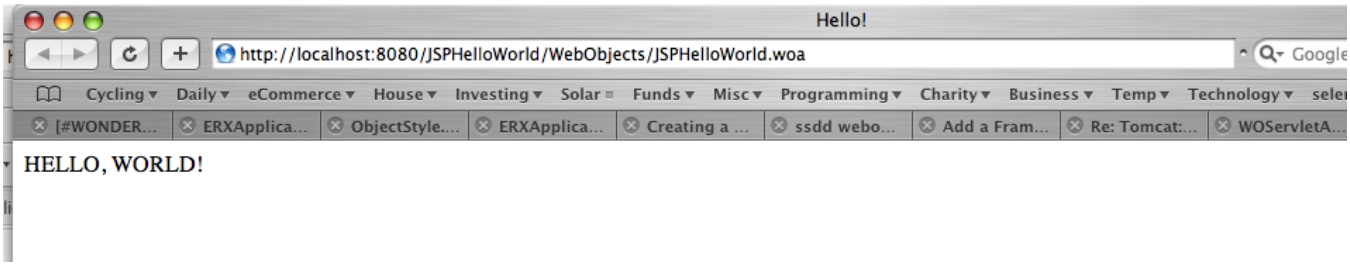
We can just edit the file by hand and save it, but next time we do an install, we're going to blow that change away and then have to remember all this minutia again to figure out what's going on. So I just modified the build.xml file to make the change for us whenever we do an install. Place this at the bottom of the build.woapp target, and next time you do a build, you'll be in business:

```
                <!-- fix the web.xml file to use a custom Servlet Adaptor that allows for Project Wonder
initialization -->

                <replaceregexp
                        file="${dest.dir}/${project.name}/WEB-INF/web.xml"
                        match="com.webobjects.jspservlet.WOServletAdaptor"
                        replace="er.extensions.jspservlet.ERXServletAdaptor"
                        byline="true" />
```

Done and done

That's it! Now, just do an ant install, copy the JSPHelloWorld folder into the webapps folder of your tomcat (or whatever) installation, restart tomcat, and load it up in your browser!



Now go make billions of dollars deploying WebObjects apps on Tomcat. I'm going to bed.