Building the wonder source code with maven

Building instructions are in the file BUILD.txt at the top level of the svn checkout.

https://wonder.svn.sourceforge.net/svnroot/wonder/trunk/Wonder/BUILD.txt

These Maven built directory frameworks are slightly different than the ant built directory frameworks. For example, the Ajax framework has dependencies on json, and the ant built Ajax.framework includes a jabsorb-1.2.2.jar, whereas the Maven built Ajax.framework does not include this jabsorb-1.2.2.jar. This is, of course, because Maven dependencies are detailed in the pom.xml, and will be pulled in via standard Maven mechanisms. The frameworks are not interchangeable--at least without manipulating dependencies. There are also a different number of frameworks.

Currently (February 2009) the wonder frameworks, applications, etc, can be built for WebObjects 5.3.x or 5.4.x; the choice is made by using a "profile" which has the value of wo53 or wo54, e.g.:

```
 mvn clean install -P wo54
```

This installs NSJarBundle frameworks for WebObjects 5.4.x into your ~/.m2/repository. The frameworks for which the profile, or classifier, was relevant, will appear in your local repository with the profile suffix:

```
./repository/wonder/core/ERExtensions/5.0.0-SNAPSHOT/ERExtensions-5.0.0-SN
APSHOT-wo54.jar
```

Note: currently the only frameworks that make use of the classifier, having differing classes and resources for them, are ERExtensions and WOOgnl.

So, in your project for both ERExtensions and WOOgnl you'll need to define these as dependencies in the following form:

```
<dependency>
  <artifactId>Foo</artifactId>
  <groupId>wonder.core</groupId>
  <classifier>bar</classifier>
  <version>baz</version>
</dependency>
```

Where *Foo* is either ERExtensions or WOOgnl, *bar* is either wo53, wo54, or wo55 etc, and *baz* is either 5.0.0-SNAPSHOT or some fixed release version.

For all other wonder frameworks you either leave out the classifier definition or leave it blank. Here's a complete example showing the two frameworks that require a classifier in the dependency declaration together with another that doesn't.

```
<dependencies>
  <dependency>
    <artifactId>ERExtensions</artifactId>
    <groupId>wonder.core</groupId>
    <classifier>wo54</classifier>
    <version>5.0.0-SNAPSHOT</version>
  </dependency>
  <dependency>
    <artifactId>WOOgnl</artifactId>
    <groupId>wonder.core</groupId>
    <classifier>wo54</classifier>
    <version>5.0.0-SNAPSHOT</version>
  </dependency>
  <dependency>
    <artifactId>Ajax</artifactId>
    <groupId>wonder.ajax</groupId>
    <version>5.0.0-SNAPSHOT</version>
  </dependency>
</dependencies>
```

Useful Information: declaring dependencies

It is important to understand how dependencies are uniquely resolved when defining your dependencies.

If you've utilised a dependency management section in your pom (or uppermost parent pom) then, according to Maven's Dependency Management, the minimal set of information for matching a dependency reference against a dependencyManagement section is actually **groupId, artifactId, type, classifier**. In many cases, these dependencies will refer to jar artifacts with no classifier. This allows us to shorthand the identity set to **groupId, artifactId**, since the default for the type field is jar, and the default classifier is null.

Thus, for those wonder dependencies that require a classifier, the set of information required is: **group, artifactId, classifier** which accepts the default type of jar. For those that have no need for a classifier, you'll only need **group, artifactId**.

If you've not used a dependency management section then you'll additionally need to specify the **version** of the dependency in order for it to be resolved.

You'll notice that the qualifier / classifier is part of the basic artifact coordinate system that the wonder team has chosen to follow: <major version>.<minor version>.<incremental version>-<qualifier>

For ERExtensions, this is: 5 . 0 . 0-SNAPSHOT - wo54;

Don't trip yourself up

Java, of course, can work with anything you choose to place on your classpath. However, it's surely wise to avoid tripping yourself right?! If you're using maven to determine your classpath then I (ldeck) advise – though nothing will stop the unwilling 🙂 – not additionally using the standard WOLips ant-based dependency management in conjunction with mavens.

That is to say, you need to understand why this would be a bad idea. Ant-based builds from wonder include their transitive dependencies in their final build; maven's do not because maven is able to resolve dependencies during your build whereas the ant-based builds won't (unless they add ivy or something similar to the mix). WOLips ant-based dependency management has quite entirely different means for resolving its dependencies and as such unless you know what your doing you should avoid mixing and matching systems.

See the Quick Start Guide for further information on setting up your environment and getting started. Also see WOLips Tutorials > Maven and other guides for creating applications and frameworks from within eclipse, with or without wonder support, utilizing the maven standard project layout

See also the WOProject-maven2 for various other related wiki pages.