Project WOnder Frameworks JUnit Testing

Out of Date Documentation

The content here is still useful, but it is not fully accurate and needs to be updated to correctly reflect latest Wonder

JUnit tests of the Project WOnder frameworks are included in the source. For each package, there is or can be a "Tests" directory next to the "Sources" directory. The test classes can be in the same package as the class being tested. For example, the location of the ERXArrayUtilities class in Project WOnder and the test class are below. For information on checking out the sources for Project WOnder, go to the at sourceforge project svn page. See the JUnit home page for more information on JUnit.

```
Frameworks/Core/ERExtensions/Sources/er/extensions/foundation/ERXArrayUtilities.java
Frameworks/Core/ERExtensions/Tests/er/extensions/foundation/ERXArrayUtilitiesTest.java
```

Note that this document is not describing any issues related to integrating these tests into an eclipse environment.

Building and Running Existing Tests

```
cd ~/<latestGreatest>/Wonder
ant tests.all
ant tests.run
( cd Frameworks/Core/ERExtensions ; ant all -Dinclude.tests=true )
```

The first ant command will do the same as "ant all" and will also build and install classes in the "Tests" directories. The names of test classes need to end in "Test". The second ant command will run all the JUnit tests found in the frameworks. The last command will run "ant all" on just the ERExtensions framework (after cd-ing into the proper directory) and will build and install it and its test classes.

Adding Tests to an Existing Class

Any method in a test class that matches the signature requirements will be run as a test. The method must be public, must return void and the name of the method must begin with "test." The method itself can do anything it wants. It can be suggested, though, that it can call methods such as "Assert. assertTrue" in the junit framework. See the JUnit javadoc for more details.

Adding a Test Class to a Package Already Having Tests

Every framework should have a class which subclasses the junit.framework.TestSuite class and which will load the test classes for that framework. For example:

```
% cat Frameworks/Core/ERExtensions/Tests/er/extensions/ERExtensionsTest.java

package er.extensions;

import junit.framework.Test;
import junit.framework.TestSuite;

/** Tests of the public API of the ERXExtensions framework.
 *
 * @author ray@ganymede.org, Ray Kiddy
 */
public class ERExtensionsTest extends TestSuite {

    public static Test suite() {
        TestSuite suite = new TestSuite("Tests for ERExtensions");
        //$JUnit-BEGIN$
        suite.addTestSuite(com.webobjects.foundation.NSArrayTest.class);
        suite.addTestSuite(com.webobjects.foundation.NSDictionaryTest.class);
        // ....
        suite.addTestSuite(er.extensions.foundation.ERXUtilitiesTest.class);
        suite.addTestSuite(er.extensions.statistics.ERXMetricsTest.class);
        //$JUnit-END$
        return suite;
    }
}
%
```

There may be better ways to have this class load the test classes, but this is what is being done now. If a TestSuite subclass does not exist for a framework, see the next section.

Adding Testability to a Package

There are a few things that need to be done so that a framework can include tests.

1) The "Tests" directory must be created at the top level of the project, next to the "Sources" directory. When test classes are found under that directory, the Project WOnder ant build system will automatically build those sources when the appropriate ant command is used.

Note that tests are assumed to be for framework projects only. One can obviously have junit tests in examples and applications, but they will not be run by the ant commands described above. (TODO: Can we point to other pages that have current information about WOUnitTest, Selenium tests and other harnesses?)

2) Every framework has a target for it in the "Build/build/build.xml" file in the Project WOnder sources. For example, one can see the following in this file:

```
     <target name="ERChronic.all">
        <antcall target="global.framework.${build.action}" >
            <param name="project.principal.class" value="" />
            <param name="project.name" value="ERChronic" />
            <param name="project.dir" value="Frameworks/Misc/ERChronic" />
            <param name="wo.external.root.bundles" value="${frameworks.wonder.core}" />
            <param name="test.className" value="er.chronic.ChronicTestSuite" />
       </antcall>
    </target>
```

The existence of the "test.className" parameter will cause the tests in this framework to be invoked by the appropriate ant command. As you may guess, the value of this parameter must be the fully-qualified class name of the TestSuite subclass for this framework.

Note that, at this time, the test classes must have the same build dependencies as the framework being tested. If the tests require different dependencies, it is not yet clear how that can best be handled. So, if you need to add a dependency for your tests, add that to the "<framework>.all" target. Hopefully it will not interfere with the building of the framework itself. (TODO: Add a parameter for "wo.external.test.bundles", same as "wo.external.root.bundles?) Also, if the antcall target above is "global.application.${build.action}", this will not work. As said above, building and running these tests works only for frameworks at this time.

See the "global.compile" target in the Build/build/generic.xml for the <wocompile> changes that cause test sources to be built.

3) Information for the framework must be added to the "tests.run" target in the Build/build/build.xml file. For example:

```
    <target name="tests.run">
        <antcall target="ERExtensions.all"><param name="build.action" value="test" /></antcall>
        <antcall target="ERChronic.all"><param name="build.action" value="test" /></antcall>
    </target>
```

4) For now, the runtime dependencies of any tests must be included in the <classpath> in the "global.framework.test" target in the Build/build/generic.xml file. The classpath includes jar files in ${build.root}, ${wo.local.frameworks}, ${wo.system.frameworks}, so this should work for most frameworks.

(TODO: Frameworks declare their build dependencies in the "<framework>.all" target. Can they also declare their test run dependencies? Where would be the best place to do this?)

If there are questions about this testing, please send mail to the Project WOnder mailing list and/or to kiddyr@users.sourceforge.net.

Adding Tests That Need Database Access

There are support methods in the er.extensions.ERExtensionsTest class that make it possible to use databases and to test against real data in the junit tests. If you add tests that can fetch and manipulate data, consider parameterizing your test class. This can be done by creating a "testAll" method in the test, and then having a static inner class that actually contains the tests. The "testAll" method can dynamically add methods to the existing TestSuite. See ERXEOAccessUtilitiesTest for an example of how this can be done.

In order to make the data access in the tests, add information to your ~/Library/wobuild.properties file:

```
    wonder.test.MySQL.url=jdbc:mysql://localhost/testingJunk
    wonder.test.MySQL.user=ray
    wonder.test.MySQL.pwd=aPassword
```

For now, the availableAdaptorNames() static method in the er.extensions.ERExtensionsTest class must also be changed. By default, the tests use only the "Memory" adaptor. It would be wonderful if one could run tests against the "Memory" adaptor and then against the "MySQL" adaptor, for instance. Right now, if one does this, tests fail in strange ways. This will be fixed when we can re-initialize EOF correctly. So, to use "MySQL", one would change the availableAdaptorNames() method from:

```
public static NSArray<String> availableAdaptorNames() { return new NSArray<String>(); }
```

to:

```
public static NSArray<String> availableAdaptorNames() { return new NSArray<String>("MySQL"); }
```

When there is something available in this array, the "Memory" adaptor will not be used by the test classes, so the tests will pass either way.