Testing-Debugging

CXF uses a standard maven build system and unit testing conventions. For those not familiar with that, here are some basic hints:

Unit testing

- We use JUnit as the testing API for writing tests.
- Tests live with the module in the src/test/java directory. In general, the src/test/java directory would have the same package layout as the src/main/java. This allows the tests to test the package protected methods.
- We use Easymock as our mock framework for producing more complex unit tests.
- To run all the tests in a module, just run "mvn test" in the module.
- To run a single test in a module, run "mvn test -Dtest=MyTest" where MyTest is the name of the test class (no package specified) that you want to run.

System/Integration testing

We have a special "systest" maven module that contains a large number of more complex test cases that involve full client/server interactions, server startups, etc... For the most part, they are just JUnit tests exactly like above, but test broader aspects. We have special utility classes (see below) to help these tests fork server process, test results, etc... Running the systests is exactly like running the unit tests, just in the systest module.

Usable testing utility classes

- org.apache.cxf.test.AbstractCXFTest - in the core module, we have a utility class that provides a few methods to help write tests. Things like creating a bus, doing xpath asserts on XML return values, etc...
- testutils module - top level, we have a testutils module that contains a BUNCH of utilities:
    - in org.apache.cxf.testutils.common, there are utility classes for forking servers, capturing the output, etc... The systests use this extensively.
    - The rest of testutils consist of a bunch of wsdls with generated code (wsdl2java) and sample server impls that implement those services. Other tests can then "share" those impls. This is to prevent having "HelloWorld" server impls all over the place.

Debugging

There are generally two ways of running a test in a debugger.

1. Within the IDE. If you setup eclipse properly, almost all the tests are runnable directly in eclipse. Right click on the test and select "Run As -> JUnit Test" (or "Debug As"). The test should just run.
2. Externally via jpda. This is much harder. By default, mvn will fork the unit tests into a JVM that doesn't have JPDA enabled. However, it can be done. First, set your MAVEN_OPTS like:

```
export JPDA_OPTS="-Xdebug -Xrunjdwp:transport=dt_socket,address=8000,
server=y,suspend=n"
export MAVEN_OPTS="-XX:MaxPermSize=192m -Xmx512M $JPDA_OPTS"
```

Then you would need to run the test like "mvn test -Dtest=MyTest -Dsurefire.fork.mode=never". When mvn starts, you can attach your external debugger to it and assign breakpoints and such.

Caveats: When debugging systests, they tend to fork background servers. If you need to debug the server side part of the interaction, you will need to disable that forking. Usually, the easiest way is to find the "setup" code for the test and look for a "launchServer(Server.class)" line and add a ", true" parameter to tell the server launcher to run the server "in process".