Turbine2 FAQ

Turbine 2.x Frequently Asked Questions

This document serves as a catch-all for questions and problems commonly encountered in Turbine 2.x. Please be sure to read this document fully before asking repetitive questions on the turbine-user mailing list. It will save the entire Turbine community, including yourself, a little time. You are encouraged to improve this document as you see fit!

Q: Why do I get a java.lang.VerifyError when trying to set up the TDK 2.x?

A: This is because of a specific version of an XML related JAR that is incompatible with TDK 2.1 located somewhere in your classpath. The fixes that are known to work usually involve trying one (or more) of the following steps:

- See if there is a `XercesImpl.jar` in your `ANT_HOME\lib` directory, if so, try removing it (maybe move it into another directory for now) and running 'ant init' again.
- Try searching for anything jaxp-related jar files from your system classpath (which may even include your JDK installation directory).

People have had success after 'moving' out of the system classpath the following files:

- xercesImpl.jar
- sax.jar
- xsltc.jar
- xalan.jar
- dom.jar

Note: by 'moving' these files to allow the TDK to run, any other project that you are using Ant for *may* not work, so it's best to keep the moved JAR files into a directory for safe keeping.

Q: I would like to extend TurbineUser to include ... how do I go about that?

A: *There is a detailed how-to document available to get you started. It is possible to add additional attributes (fields) to the extended object which will be persisted to the database, as well as add new foreign key references with the TURBINE_USER table in your <code>project-schema.xml</code> file (cf. Tor que).

- The turbine-user list archives contains a large library of tips, suggestions and solutions concerning extending TurbineUser. This is probably the most frequently discussed topic in the list, so please be sure to educate yourself thoroughly.

Q: How do I capture uploaded files?

A: Read the documentation for the UploadService.

Q: What is with the 4 versions of Turbine currently available? Why? And Which do I use?

A: See the Turbine project page for the current version information.

Q: Despite what you people say, Intake does not work! How do I get it to do what you claim it does?

A: Please refer to the page dedicated to common issues and stumbing points with Intake: CommonIntakeProblems

Q: I created my own version of the login action. Why doesn't Turbine use it?

A: Turbine handles the login and logout actions differently than normal actions. The settings <code>action.login</code> and <code>action.logout</code> in TurbineResources.properties are used to tell Turbine which actions are login and logout.

Q: What are Fulcrum and Torque?

A: In Turbine 2.1, the code for all of the services (such as Intake, Velocity, Database, Logging, etc) were tightly coupled to Turbine. In order to allow this code to be reused in other projects Torque and Fulcrum were created. They are both still considered to be under the Turbine umbrella.

Torque is the database access layer. Turbine 2.2 uses this new decoupled version. More information on Torque can be found on the Torque project page.

Fulcrum contains all of the decoupled service code. Turbine 2.2 still uses the coupled services although you can switch to the Fulcrum version for *some* services. Fulcrum does not have a released version. It is suggested that you do not use the Fulcrum services at this time. More information can be found on the Fulcrum project page.

Q: Why do I have to use a comma with the $link tool?

A: Turbine's method of parsing URL parameters relies on everything being seperated by the "/" character. If you will notice the URL generated by $link.setPage("login.vm"), reads something like `http://www.servername.com/context/servlet/servletname/template/login.vm`. The output of $link.setPage("blah/login.vm") looks something like }}}http://www.servername.com/context/servlet/servletname/template/blah/login.vm}}}. When turbine parses the second URL, it thinks that you have requested a template named "blah".

The way to be able to request blah/login.vm is to replace the "/" character with a comma. This is the preferred way of using the $link pull tool. This does not mean that you **MUST** do it this way. Some people have complained that it is confusing to the people writing the view.

If you really don't like the comma, you can use a different version of the pull tool. There is another version supplied in the org.apache.turbine.util.template package called TemplateLink'*_WithSlash. This version of the pull tool enables you to use the slash anyway. When it generates the URL for you, it will replace the "/" character with the comma. Of course, this is inefficient since there is a search and replace going on everytime you generate a URL.

You are also free to create your own implementation of the $link pull tool. Simply override the setPage() method to replace whatever character you want to use as the seperator with the commas.

If you do decide to use a different version of the pull tool than the one configured by default, you will need to change the `tool.request.link` setting in your `TurbineResources.properties` file to the correct class name of the version that you want to use.

Q: Turbine only initializes during the first request to the application. How do I make it initialize on startup?

A: You can use the <load-on-startup> tag in your web.xml to accomplish this. Example:

```
<servlet>
  <servlet-name>turbine</servlet-name>
  <servlet-class>org.apache.turbine.Turbine</servlet-class>
  <init-param>
    <param-name>properties</param-name>
   <param-value>/WEB-INF/conf/TurbineResources.properties</param-value>
  </init-param>
  <load-on-startup/>
</servlet>
```

Q: I keep getting the following error message: log4j:WARN No appenders could be found for logger...

A: If you are using Tomcat and you do not get this message everytime you start the server, it is probably because Tomcat is attempting to reload the sessions that were serialized during the last shutdown. If this is the source of your problem, restarting Tomcat will ALWAYS fix the problem. It will seem to come back on every other restart.

To fix this, you can modify your startup script/batch file to delete *.ser from Tomcat's work directory.

Q: Can I use my IDE's debugger with Turbine?

A: Yes. The actual procedure of how to accomplish this will vary depending on the IDE and your servlet container though. Check the JakartaTurbineIDEDebugging page to see if someone has posted instruction for your environment. If not, please add the appropriate text to that page after you figure it out 🙂

Q: How can I test my Turbine actions?

A: Use Cactus! Using Cactus to perform in container testing is much simpler then trying to use Junit directly, manually start up Turbine, and then try and obtain access to all the resources that you need.

Write your Cactus test cases as if you were just writing code that was running inside of Turbine. The only extra step is to create your own RunData and Context objects. For example, I have an action called `DaughterboardDisplay`. To test it I would write:

```
 <snip>
 // Junit imports
 import junit.awtui.TestRunner;
 import junit.framework.Test;
 import junit.framework.TestSuite;

 // Cactus imports
 import org.apache.cactus.ServletTestCase;
 import org.apache.cactus.WebRequest;
 import org.apache.cactus.WebResponse;

 // Turbine imports
 import org.apache.turbine.util.RunData;
 import org.apache.turbine.services.TurbineServices;
 import org.apache.turbine.services.rundata.RunDataService;
 import org.apache.velocity.context.Context;
 import org.apache.turbine.services.velocity.TurbineVelocity;

 <snip>

 public class TestDaughterboardDisplay extends ServletTestCase {
    private DaughterboardDisplay daughterboardDisplay;
    private RunData data= null;
    private Context context = null;
 <snip>

    public void setUp()
        throws Exception {
        super.setUp();
        // Create the context objects to be used during testing.
        data = ((RunDataService) TurbineServices.getInstance().getService(RunDataService.SERVICE_NAME)).
getRunData(request, response, config);

        context = TurbineVelocity.getContext(data);

        // Create the Action
        daughterboardDisplay = (DaughterboardDisplay) ActionLoader.getInstance().getInstance
("DaughterboardDisplay");
    }

   /**
     * Tests if an daughterboard displays nicely.
     */
    public void testSimpleDisplayofDaughterboard () throws Exception
    {
        // Add parameters to the RunData that will be used to display a daughterboard
        data.getParameters ().setString ( "daughterboard_id", "636" );

        // Call the do* method
        daughterboardDisplay.doPerform(data, context);

        // Evaulate the results
        Daughterboard db = (Daughterboard)context.get("daughterboard");
        assertEquals ( "Loaded up the daughterboard into the context.",
        new Integer("636"), db.getDaughterboardId());
    }
 <snip>
```

Note: You must have a web.xml that automatically starts up Turbine when the container starts up. See the FAQ section "Turbine only initializes during the first request to the application. How do I make it initialize on startup?". Note: I have a TurbineTest_*'Case that I inherit from. In the setUp() method is where all the rundata stuff goes. Then I have it for all my test cases. Any other global stuff, like resetting the database using DbUnit goes there as well.

– EricPugh

Q: How can I start Turbine in Cactus?

A: If you are testing services, or don't want to use a custom web.xml to startup Turbine on load, or are testing different config files, you can manually start up Turbine like this:

```
/**
 * This setup will be running server side.  We startup Turbine and
 * get our test port from the properties.  This gets run before
 * each testXXX test.
 */
protected void setUp()
    throws Exception
{
    super.setUp();
    config.setInitParameter("properties",
            "/WEB-INF/conf/TurbineCacheTest.properties");
    turbine = new Turbine();
    turbine.init(config);
}

/**
 * After each testXXX test runs, shut down the Turbine servlet.
 */
protected void tearDown()
    throws Exception
{
    turbine.destroy();
    super.tearDown();
}
```

– EricPugh

Q: Why do I see jsessionid=xxxx on my URL when I access my application?

A: This should only be happening when you first access your application after a restart. If you are on a unix platform you can add a command to the startup.sh file to make lynx access your application.

The following example code was submitted by Jeff Painter:

```
<tdkroot>/bin/catalina.sh
elif [ "$1" = "start" ] ; then
.
.
...snipped...
#
# pause 15 seconds... wait for turbine to come up
#
/bin/sleep 15
#
# open first login page.. dump to /dev/null
#
lynx -source http://localhost:8080/app/servlet/app > /dev/null # elif[ "$1" = "stop" ] ; then ...end snip...
```

Q: How can I tune Turbine to perform better

A: There are quite a few things you can do to help Turbine run better:

- If you are running Tomcat as a windows NT Service, check out this link: http://forum.java.sun.com/thread.jsp?thread=290568&forum=33&message=1211179

Q: How can I setup Turbine once for my JUnit Test cases?

A: This can be accomplished using JUnit's TestSetup extension. In order to avoid copying the structure to each test case, create a superclass called BaseTurbineTest that you can perform callbacks to get the TestSetup.

- See Also: http://jakarta.apache.org/turbine/tdk/application-testing.html

Below is an example of a BaseTurbineTest class:

```
import junit.extensions.TestSetup;
import junit.framework.Test;
import junit.framework.TestCase;

import org.apache.turbine.util.TurbineConfig;

public class BaseTurbineTest extends TestCase
{
   protected static Test getTurbineSetup(Test test)
   {
      TestSetup testSetup = new TestSetup(test)
      {
         private TurbineConfig config = null;

         public void setUp() throws Exception
         {
            config = new TurbineConfig(".", "WEB-INF/conf/test/TurbineResources.properties");
            config.initialize();
         }

         public void tearDown() throws Exception
         {
            config.dispose();
         }
      };

      return testSetup;
   }
}
```

Once you've setup this class, creating new tests is very quick and easy. Now we can ensure the Turbine framework starts up once for the TestCase below "MyTest" and still run setUp() and tearDown() code for each test. Check out the example:

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

public class MyTest extends BaseTurbineTest
{
   public static Test suite()
   {
      return getTurbineSetup(new TestSuite(MyTest.class));
   }

   public void setUp()
   {
      System.out.println("Set up.");
   }

   public void tearDown()
   {
      System.out.println("Tear down.");
   }

   public void test1()
   {
   }

   public void test2()
   {
   }

   public void test3()
   {
   }
}
```