Errai configure maven and using Errai ui-binders

Author:	Sebastian Börebäck
Version	1.0 Initial draft
Date	2013-04-04

Contents

Description

This is a basic tutorial showing little about how to configure your maven, the pom file. And implementing Errai UI (Errais ui binders).
First you need an Errai project. See the basic setup tutorial.

Tutorial Background

So Lincoln Baxter III has a video showing how to implement Errai ui binders. In normal GWT uibinders are nice and awesome, but if you have a web designer that creates the web layout using normal html files, you need to convert them to uibinder friendly ui.xml files.
So with Errai UI is possible to take the html5 file from the web designer and use in you project. You just make a annotation and boom you can

manipulate the elements, add event handlers etc.

In this tutorial we will:

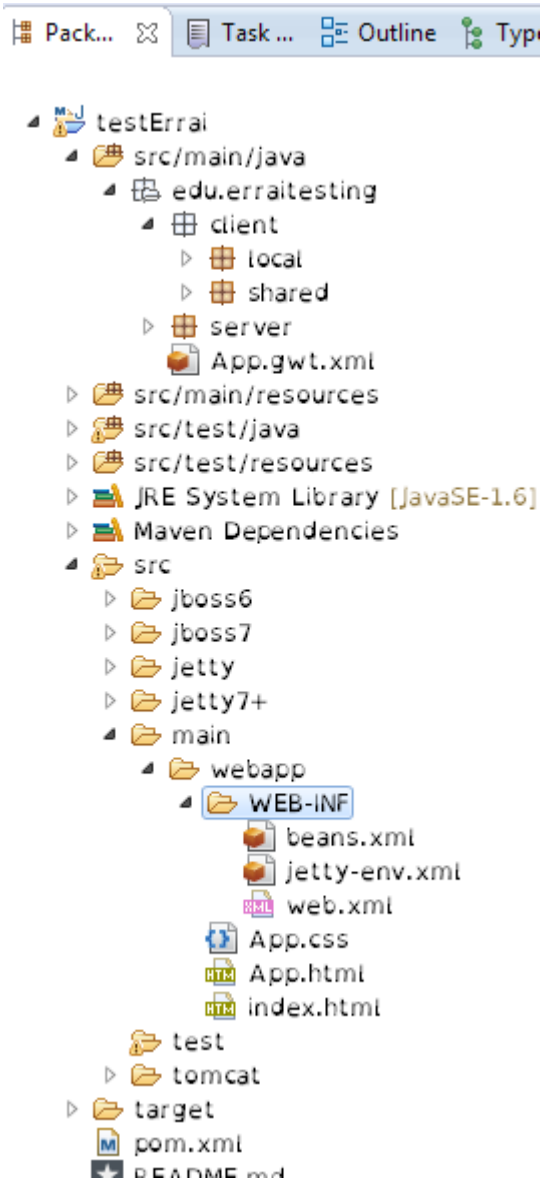
- Basic Edit and working with Errai project.
- This tutorial is to help you understand some basic maven settings, working with the pom.xml file.
- How to add Errai ui-binders to your project.
  You can read more about Errai uibinders
  If you can't see the video here is a link (happens in chrome)
- Deploy to Jboss as 7

All the tanks to Lincoln Baxter III for providing Errai UI

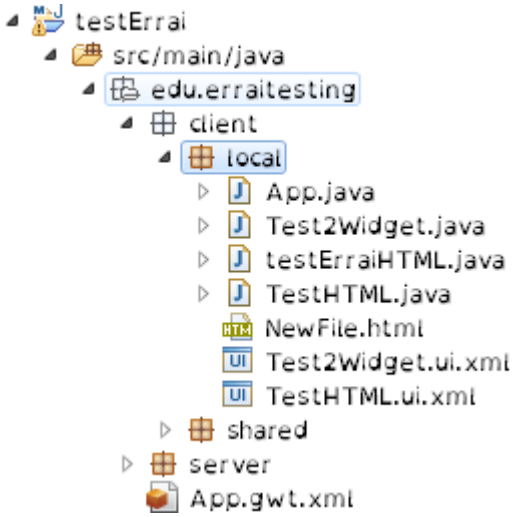
Let's begin

Where I left of I created a project using cdi-quickstart and we were able to run it (the basic setup tutorial).
With Errai/GWT the project is divided in to 3 different packages.

- Client
  this is everything on client side. The frontend.
  The code here is constrained to GWT. Because it will be converted to JavaScript.
  Basically what you do here is creating the GUI and linking it to the backend.
  To do the connection to backend, this is pure java. You use the shared package.
- Shared
  this is the "layer" between client (JavaScript) and server (java).
  It lives on client side.
  Here are your beans. What you send between server and client.
  Hence the annotation @portable
- Server
  this is the backend of your application.
  Here you have the database connection, security etc.
  What is important is that this is normal java code.
- "projectname".gw.xml
  Located in the root of the package.
  This file holds the modules that you use.
- Pom.xml
  this is your maven configurations.
  It holds which dependencies you are using and deployment settings.
- Web.xml
  this file basically holds you servlet settings.



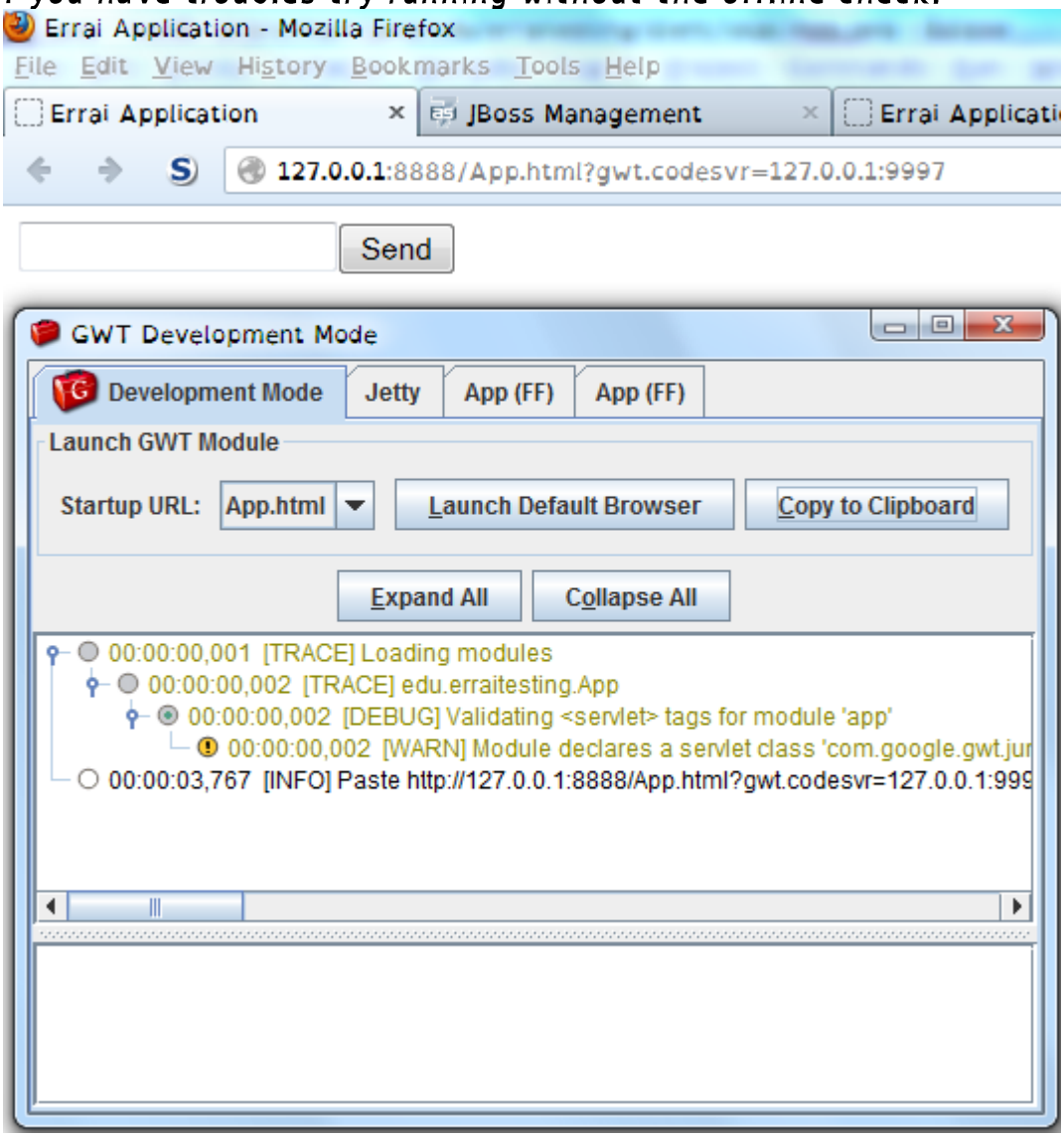
What does App.java do?



App.java lives on client side.
And has the **@EntryPoint** annotation.
This means it's the starting point for your Errai project. (ground zero, root).
@PostConstruct --> this means that it will run this before anything else.
And here you can connect the event handlers for the buttons, build your UI.

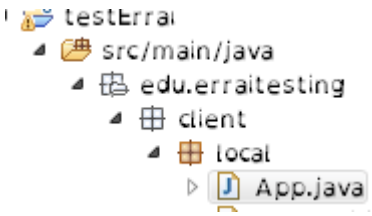
Run you project

Now run your project using the gwt:run configuration you created last time.
If you have troubles try running without the offline check.



This is what you should get a GWT development Mode.
And if you clicked Launch Default browser (Firefox in my case),
You should see a textbox and button saying send.
The URL should look something like the above one.
?gwt.codesvr=127.0.0.1:9997 this part of the url is the GWT development mode.
Remove it and you run in "install mode" (need to be compiled "install" before).
Keep it alive. (Don't shut the dev. Mode down).

Edit your App.java



In App.java (under client/local/) you can already see (from cdi-quickstart).
There is a HorizontalPanel which you add a bunch of element to textbox, button and label.
Then to connect everything you add the
RootPanel.get().add(horizontalPanel);
Add a new Label:
By in the @PostConstruct adding the yellow marked lines.
Return to your browser and refresh it.
you should instantly see the changes

In the PostConstruct constructor add:

```
@PostConstruct
public void buildUI() {
  Label helloLabel = new Label("Hello world");
  HorizontalPanel horizontalPanel = new HorizontalPanel();
  horizontalPanel.add(helloLabel);
  RootPanel.get().add(horizontalPanel);
```

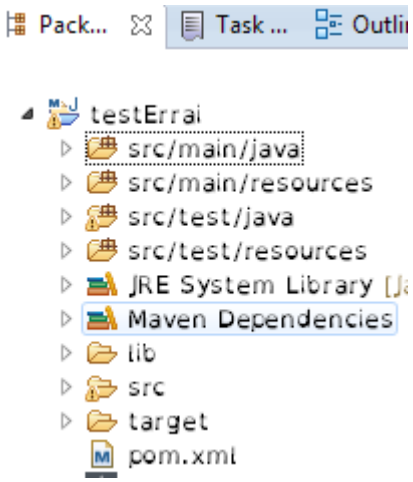
This will then add the

```
@PostConstruct
public void buildUI() {
    Label helloLabel = new Label();
    helloLabel.setText("Hello world");
    HorizontalPanel horizontalPanel = new HorizontalPanel();
    horizontalPanel.add(helloLabel);
    horizontalPanel.add(message);
    horizontalPanel.add(button);
    horizontalPanel.add(responseLabel);

    RootPanel.get().add(horizontalPanel);
```

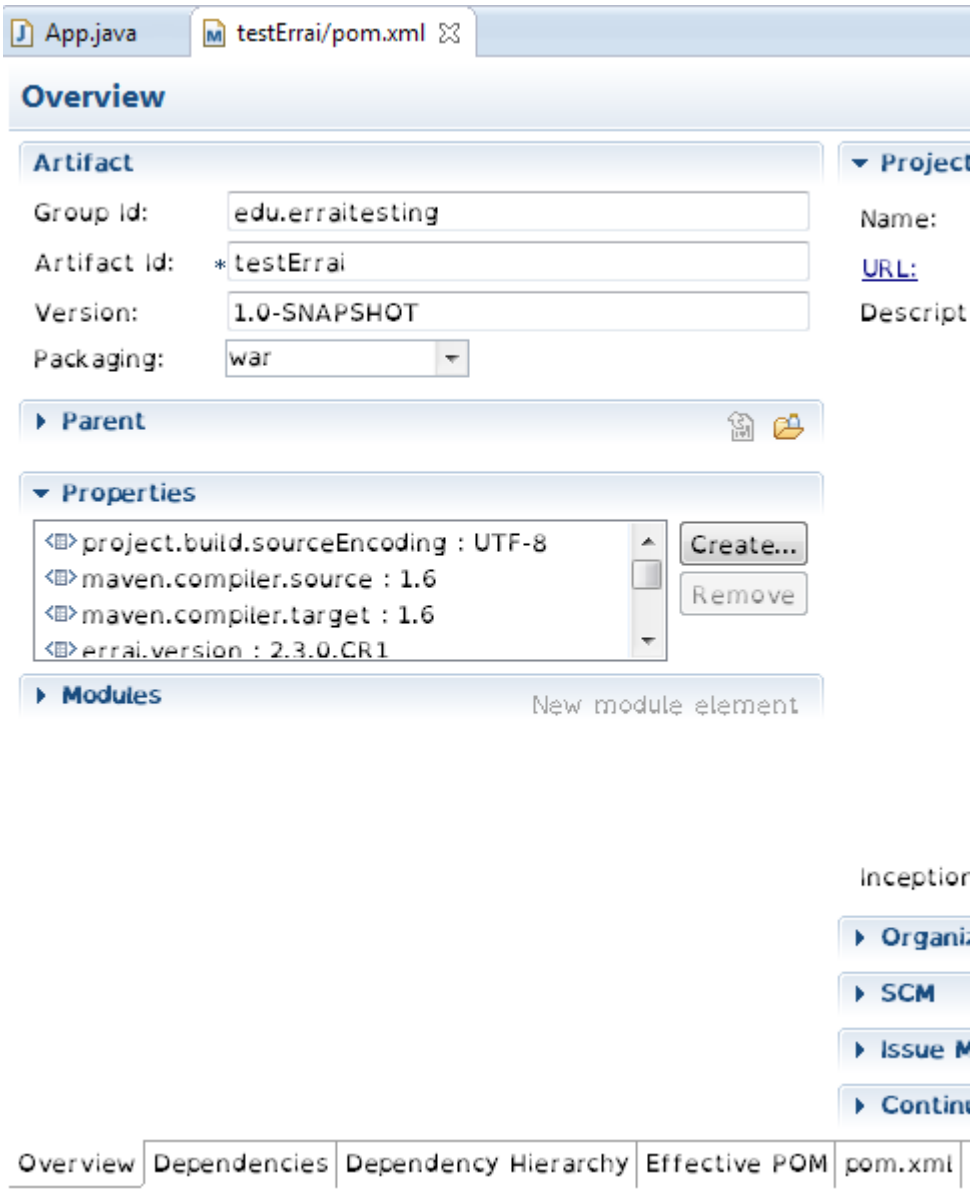
Result after adding the helloLabel

So that's normal for GWT...

Update maven dependencies



Open up your pom.xml file. This is the maven config file.



Click on pom.xml

App.java | testErrai/pom.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apac
  <modelVersion>4.0.0</modelVersion>
  <name>testErrai</name>
  <groupId>edu.erraitesting</groupId>
  <artifactId>testErrai</artifactId>
  <packaging>war</packaging>

  <version>1.0-SNAPSHOT</version>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <maven.compiler.source>1.6</maven.compiler.source>
    <maven.compiler.target>1.6</maven.compiler.target>
    <errai.version>2.3.0.CR1</errai.version>
    <gwt.version>2.4.0</gwt.version>
    <weld.version>1.1.6.Final</weld.version>
    <gwt.maven.version>2.4.0</gwt.maven.version>
    <mvel.version>2.1.Beta8</mvel.version>
    <jetty.version>6.1.25</jetty.version>
    <slf4j.version>1.6.1</slf4j.version>
    <uel.impl.version>2.1.2-b04</uel.impl.version>
  </properties>

  <dependencies>
    <dependency>
      <groupId>javax.servlet</groupId>
      <artifactId>servlet-api</artifactId>
      <version>2.5</version>
      <scope>provided</scope>
    </dependency>
```

So what do we see here:

- Name of the project
- Groupid (package)
- artifactID (project name)
- Packagging war - meaning its a Web application ARchive.
- Version.

Cool all this I know from last tutorial.
What more do we have:

- Under properties
  - errai.version - which tells you which Errai version you are using in the project.
  - gwt.version - which gwt version

These are variables that will affect your project.
So if you don't feel like using Errai.version 2.3.0.CR1 but instead 2.2.0.Final just change the text and save.

Now you need to install the project again.
If development mode is still running, close it.
And select/create a clean install maven run configuration. See last tutorial for more info.
This will now update the Errai version to your new selected one.
If you get:
[INFO] --
[INFO] BUILD SUCCESS
[INFO] --

Everything is shiny.
To double check versions and names: go to the http://search.maven.org/ here as I mentioned in the earlier tutorial, you can lookup dependencies etc.

If you changed the version to 2.2.0.Final change it back to 2.3.0.CR1.

Errai UI

Okay let's say you get a html5 file from a web designer with the new design for your webapp. And you want now to connect it to Errai.
No worries we have Errai UI!

What you need to do is:

- Update your maven dependencies in the pom.xml
- Update the web.xml file
- Update the "project".gwt.xml file
- Create a new html5 file and java file.
- Connect the new html5 page on the Rootpangel.get()

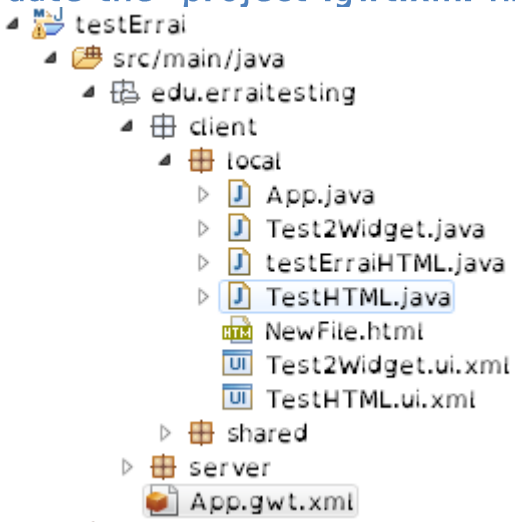
Update Maven dependencies, pom.xml

Open your pom.xml file.
Scroll now down to
<!-- CDI Integration Modules -->
Above this line add:

```
        <dependency>
            <groupId>org.jboss.errai</groupId>
            <artifactId>errai-ui</artifactId>
            <version>${errai.version}</version>
          </dependency>
            <dependency>
            <groupId>org.jboss.errai</groupId>
            <artifactId>errai-data-binding</artifactId>
            <version>${errai.version}</version>
    </dependency>
    <!-- CDI Integration Modules -->
```

Okay there is a second way to update your maven instead of running a complete clean install. Right click your project and select maven --> Update project...
Make sure that offline is unchecked.
This will update your dependencies as well.

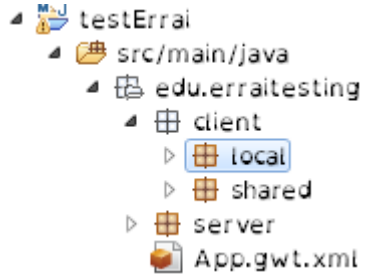
Update the "project".gwt.xml file



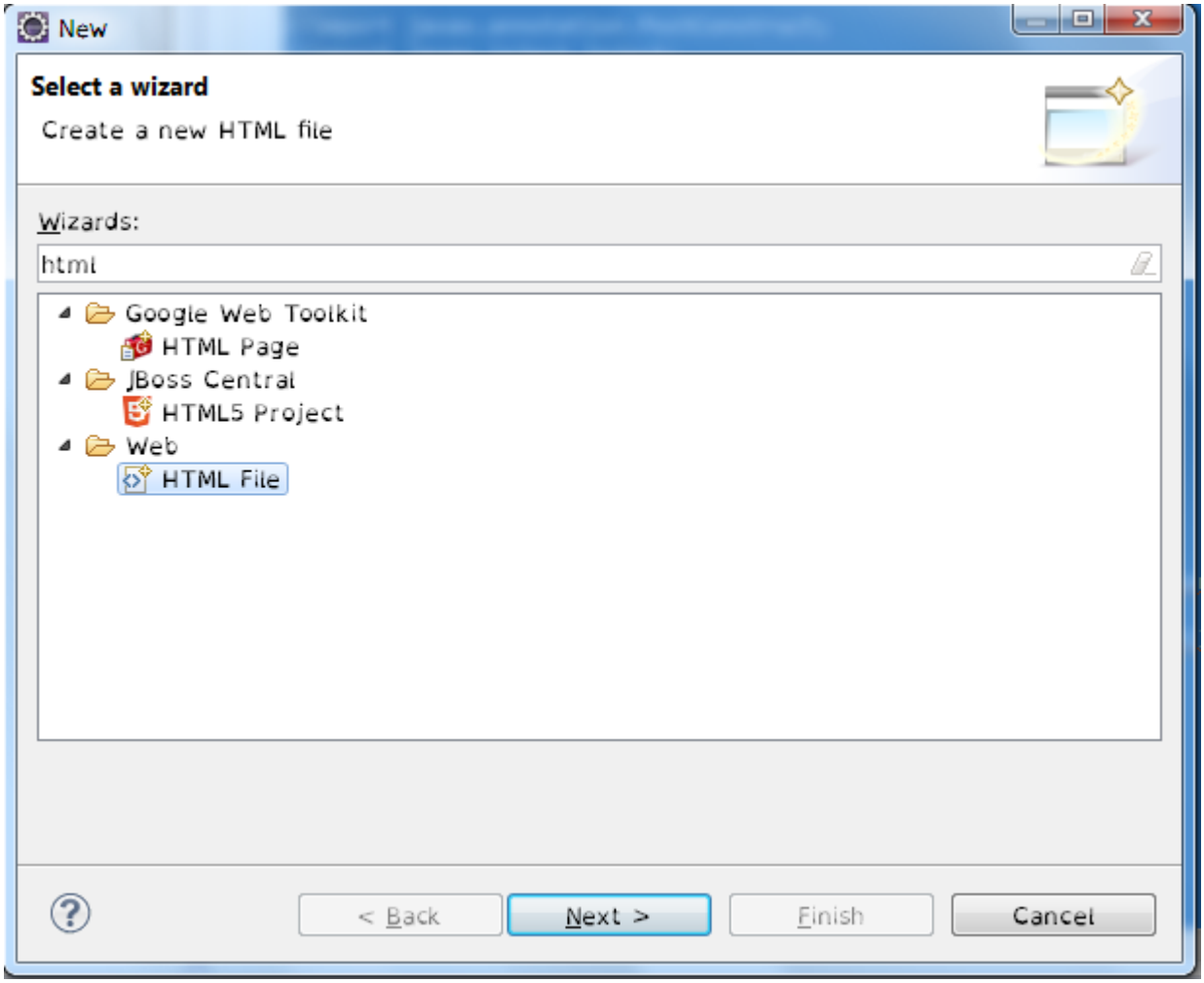
Under packagename update the "project".gwt.xml file. In my case App.gwt.xml
Add: <inherits name="org.jboss.errai.ui.UI" />
Extra: to add comments in xml files <!-- comment text -->

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE module PUBLIC "-//Google Inc.//DTD Google Web Toolkit
1.6//EN"
        "http://google-web-toolkit.googlecode.com/svn/releases/1.6/distro-
source/core/src/gwt-module.dtd">
<module rename-to="app">
    <inherits name="org.jboss.errai.common.ErraiCommon"/>
    <inherits name="org.jboss.errai.bus.ErraiBus"/>
    <inherits name="org.jboss.errai.ioc.Container"/>
    <inherits name="org.jboss.errai.enterprise.CDI"/>
        <!-- errai ui binder  -->
    <inherits name="org.jboss.errai.ui.UI" />
</module>
```

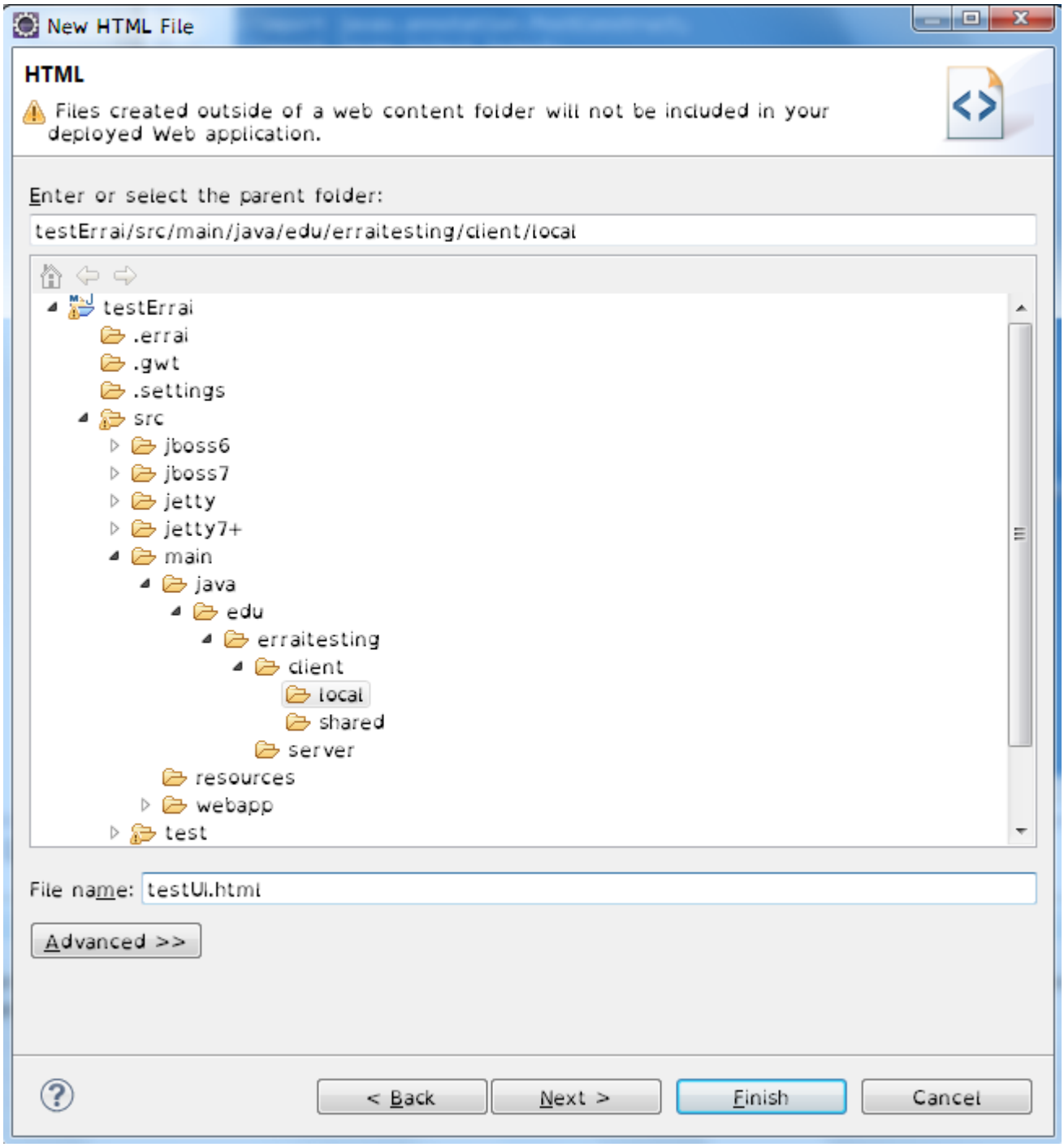
Create a new html5 file and java file.



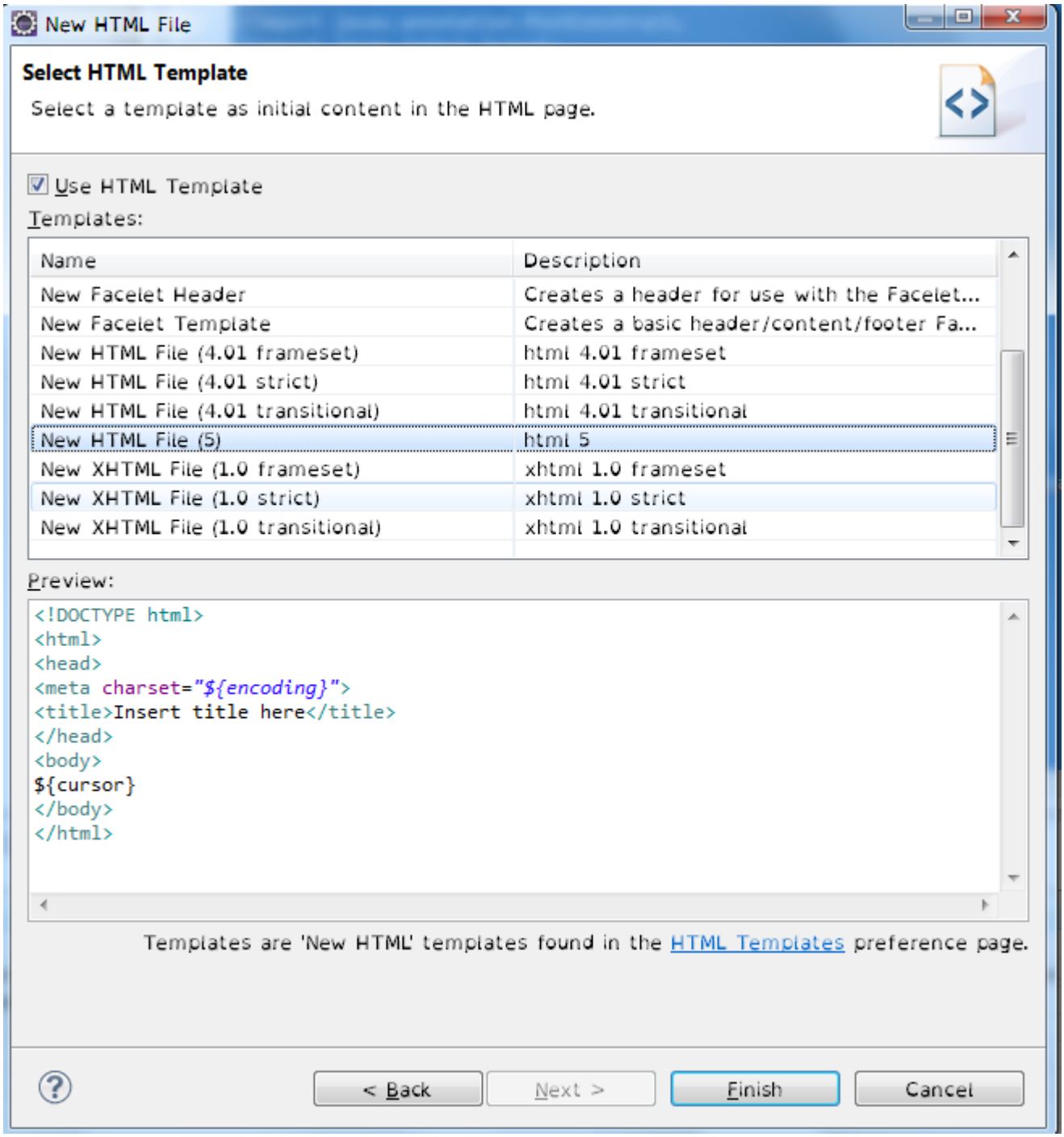
Right click Local and select New --> Other..



Write html and select HTML file --> Next



Now create the HTML file in your **client folder**, give it a name (in my case **testUI.html**).
Ignore the warnings/complains.
Click **Next.**
Ps. If clicked Finish then you created a html 4 file. And this is wrong.

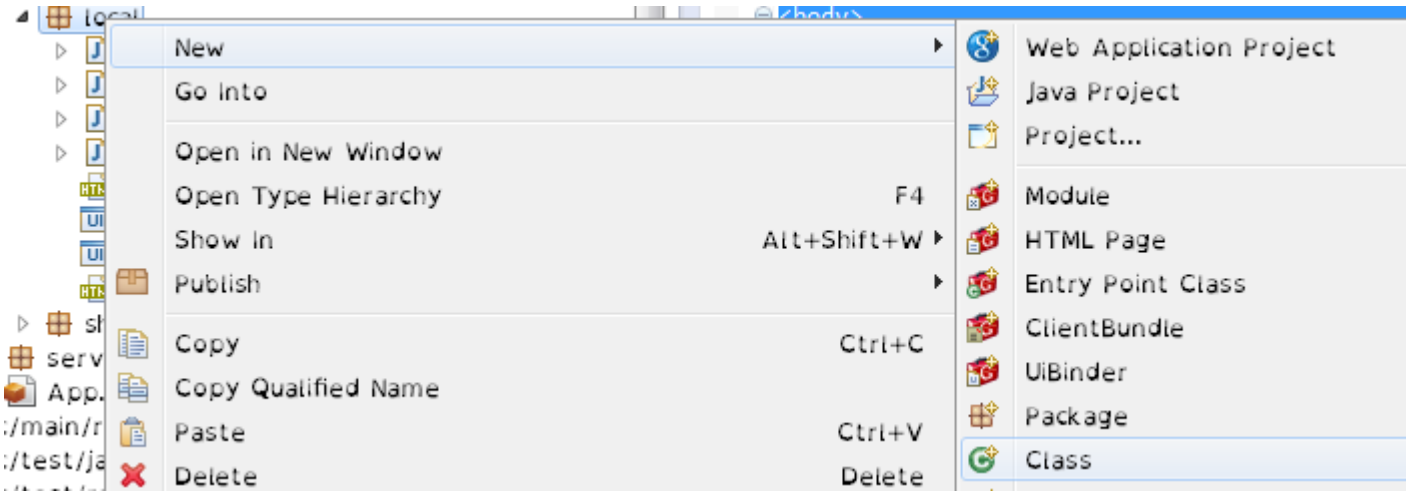


Select **New HTML File (5)** and click **Finish.**

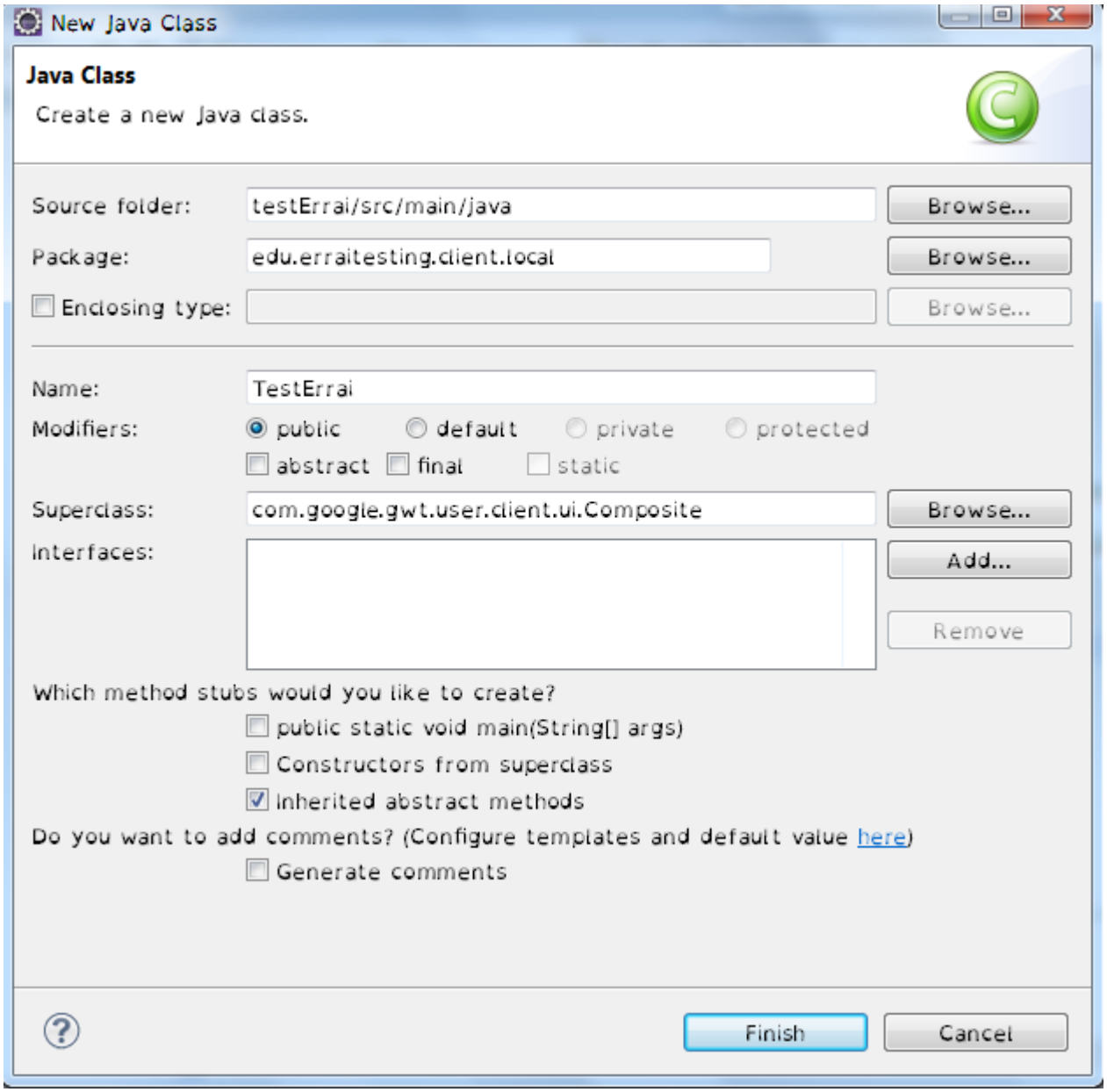
Now you should have a brand New HTML file in your client folder.
Better would be to import the file from your web designer.
So now create some things.
In your html file add some html code.
My html file looks like this:

```
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Test UI</title>
</head>
<body>
	<div> hello world</div>
	<div data-field="erraiStart"> second div
		<div data-field="labelhello"> hello world 2</div>
		<a data-field="buttonClickme" type="button">Click me</a>
	</div>
</body>
</html>
```

I added 3 dives and a button. I will come back to what data-field is.



Now we need to connect this HTML file to a java class.
Create a new java file in local (in the client package).



Add the com.google.gwt.user.client.ui.Composite as Superclass:
Click Finish.

On the newly created file TestErrai add
@Templated("testUI.html#erraiStart").
And import org.jboss.errai.ui.shared.api.annotations.Templated;
And add a @PostConstruct.

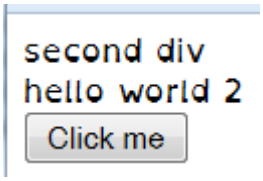
```
package edu.erraitesting.client.local;

import javax.annotation.PostConstruct;

import org.jboss.errai.ui.shared.api.annotations.Templated;
```

```
import com.google.gwt.user.client.ui.Composite;

@Templated("testUI.html#erraiStart")
public class TestErrai extends Composite {

	@PostConstruct
	public void setup()
	{

	}

}
```

So what I did now is say use the html file testUI.html and EntryPoint for this java class (which will be a JavaScript when it's compiled) at erraiStart the date-field I gave one of the dives before.
Now update the App.java so it can use the new java TestErrai.java:
Add a injection of the TestErrai class.
Comment/remove the old RootPanel.get().add(horizontalPanel);
And add the TestErrai to the Rootpanel,
RootPanel.get().add(testErraiInstance.get());

```
@EntryPoint
public class App {

	@Inject
	Instance<TestErrai> testErraiInstance;

 @PostConstruct
    public void buildUI() {

            RootPanel.get().add(testErraiInstance.get());
//        RootPanel.get().add(horizontalPanel);
```

Now rerun the project, you should see:



- The button and the 2 dives that was inside of the div with data-field erraiStart
- From the TestErrai.java/testUI.html

So now we managed to connect a html file with our java class.

Let's update something and make the button function.

First update the testUI.html.
Add data-filed="nameOfyourDatafield"
This will enable us to use these fields in the java class.

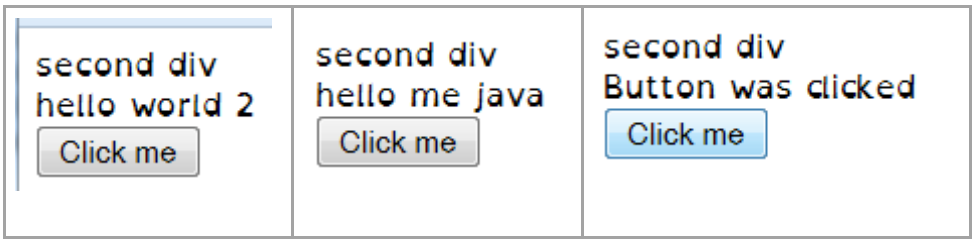
```
<div data-field="erraiStart"> second div
    <div data-field="labelhello"> hello world 2</div>
    <input data-field="button" type="button" value="Click me"/>
</div>
```

In the TestErrai.java edit it:

```
package edu.erraitesting.client.local;

import javax.annotation.PostConstruct;
import javax.inject.Inject;

import org.jboss.errai.ui.shared.api.annotations.DataField;
import org.jboss.errai.ui.shared.api.annotations.EventHandler;
import org.jboss.errai.ui.shared.api.annotations.Templated;

import com.google.gwt.event.dom.client.ClickEvent;
import com.google.gwt.user.client.ui.Button;
import com.google.gwt.user.client.ui.Composite;
import com.google.gwt.user.client.ui.Label;

@Templated("testUI.html#erraiStart")
public class TestErrai extends Composite {


    @Inject
    @DataField
    Label labelhello;

    @Inject
    @DataField
    Button buttonClickme;

    @PostConstruct
    public void setup()
    {
        this.labelhello.setText("hello me java");
    }

    @EventHandler("buttonClickme")
      public void doSomethingC1(ClickEvent e) {
        this.labelhello.setText("Button was clicked");
      }
```

}

So what changed is: the div that we created before that said "hello world 2" now will say "hello me java". Until you click the button then it will change to "Button was clicked".



With a real HTML5 file with CSS3 and everything you could do some great looking webpages.
Fast and easy, the sky is the limit.

One limitation is i18n is not available yet for Errai UI.
See this discussion for more info.