Deployment Environment

Eclipse code style code-style.xml

Deploying GWT Apps

- A good article about deploying GWT apps (http://developerlife.com/tutorials/?p=231)

Deployment from Eclipse

The eclipse plugin has the compile feature options built in.

1. Complile
    a. Right click on project > Debug As > Web Application OR
    b. Hit the Red"G" Box icon top left, and compile options will appear
2. Create a war, by zipping up the war directory and rename it [project name].war
    a. move libraries if need be, if there linked and not in the library folder
3. Copy the war file to the webapp folder for Tomcat deployment

Tomcat Setting for Connection Pool (JNDI) for SQL Server 2005 and Postgresql

- Copy jtds-1.2.5.jar (http://jtds.sourceforge.net/) to CATALINE_HOME/lib
- Copy postgresql-8.4-701.jdbc4.jar (http://jdbc.postgresql.org/) to CATALINE_HOME/lib

- Edit conf/context.xml like below:

```
<?xml version='1.0' encoding='utf-8'?>
<Context>
    <!-- definition of DB connection resource -->
<Resource name="jdbc/mmsql"
        auth="Container"
        type="javax.sql.DataSource"
        driverClassName="net.sourceforge.jtds.jdbc.Driver"
        url="jdbc:jtds:sqlserver://[host_url]/[dbname]"
        username="[username]"
        password="[password]"
        maxActive="8"
        maxIdle="4"
/>

<Resource name="jdbc/pgsql"
        auth="Container"
        type="javax.sql.DataSource"
        driverClassName="org.postgresql.Driver"
        url="jdbc:postgresql://[host_url]/[dbname]"
        username="[username]"
        password="[password]"
        maxActive="8"
        maxIdle="4"
/>

    <WatchedResource>WEB-INF/web.xml</WatchedResource>
</Context>
```

- Add the following code in <web-app> tag in webapps/[your_webapps]/WEB-INF/web.xml:

```
        <resource-ref>
                <description>MM DB connection</description>
                <res-ref-name>jdbc/mmsql</res-ref-name>
                <res-type>javax.sql.DataSource</res-type>
                <res-auth>Container</res-auth>
        </resource-ref>

        <resource-ref>
                <description>Spaital MM DB connection</description>
                <res-ref-name>jdbc/pgsql</res-ref-name>
                <res-type>javax.sql.DataSource</res-type>
                <res-auth>Container</res-auth>
        </resource-ref>
```

Configure Tomcat CGI

In order to retrieve features from the geoserver using XMLHttpRequests, you will need to use the proxy.cgi file that comes with OpenLayers. You can obtain this proxy and more information about it here. In this setup, we assume that you are configuring Tomcat 6 or higher, the setup of Tomcat 5 is slightly different.

Step 1:

The first file to edit is inside **CATALINA_HOME/conf** called *context.xml*. You will need to change:

<Context>

to

<Context privileged="true">

Save the changes.

Step 2:

The next file to edit is also in **CATALINA_HOME/conf** called *web.xml*. You will need to find the lines that discuss the _ Common Gateway Includes (CGI) processing servlet_, you can do this by searching for CGI. You will need to uncomment the section below and add the missing parts. Most of it is already filled in by default and is just commented out.

```
<servlet>
        <servlet-name>cgi</servlet-name>
        <servlet-class>org.apache.catalina.servlets.CGIServlet</servlet-class>
        <init-param>
          <param-name>debug</param-name>
          <param-value>0</param-value>
        </init-param>
        <init-param>
          <param-name>executable</param-name>
          <param-value>/usr/bin/python</param-value>
        </init-param>
        <init-param>
          <param-name>cgiPathPrefix</param-name>
          <param-value>WEB-INF/cgi</param-value>
        </init-param>
        <init-param>
        <param-name>passShellEnvironment</param-name>
        <param-value>true</param-value>
        </init-param>
         <load-on-startup>5</load-on-startup>
    </servlet>
```

Be sure to add the complete path to your python executable. Also, note the *cgiPRefixPath*. You will need to create a directory inside the **WEB_INF** folder called *cgi* and place the proxy.cgi file inside it. Further down in this file you will need to uncomment the following line:

```
<servlet-mapping>
        <servlet-name>cgi</servlet-name>
        <url-pattern>/cgi-bin/*</url-pattern>
</servlet-mapping>
```

Save the changes.

Step 3:

The final step involves setting the proxy host and creating the war file. If you name your war file **marketmaker.war** this will unzip and create the directory marketmaker which includes a subdirectory called **WEB-INF**. Inside the **WEB-INF** directory there should be a directory called **cgi** that contains the file *proxy.cgi*. In this case, inside your entry point class where you set the proxy host, you will need to specify the proxy host as **/marketmaker/cgi-bin/proxy.cgi?url=** so that the proxy host is set properly. If we had named the war file **client.war**, the proxy host would be set as **/client/cgi-bin/proxy.cgi?url=**. You can test your proxy by going to the link http://localhost:8080/marketmaker/cgi-bin/proxy.cgi if you named the war file **marketmaker.war** and left the default port for tomcat.