Creating a new Zcore-based Project

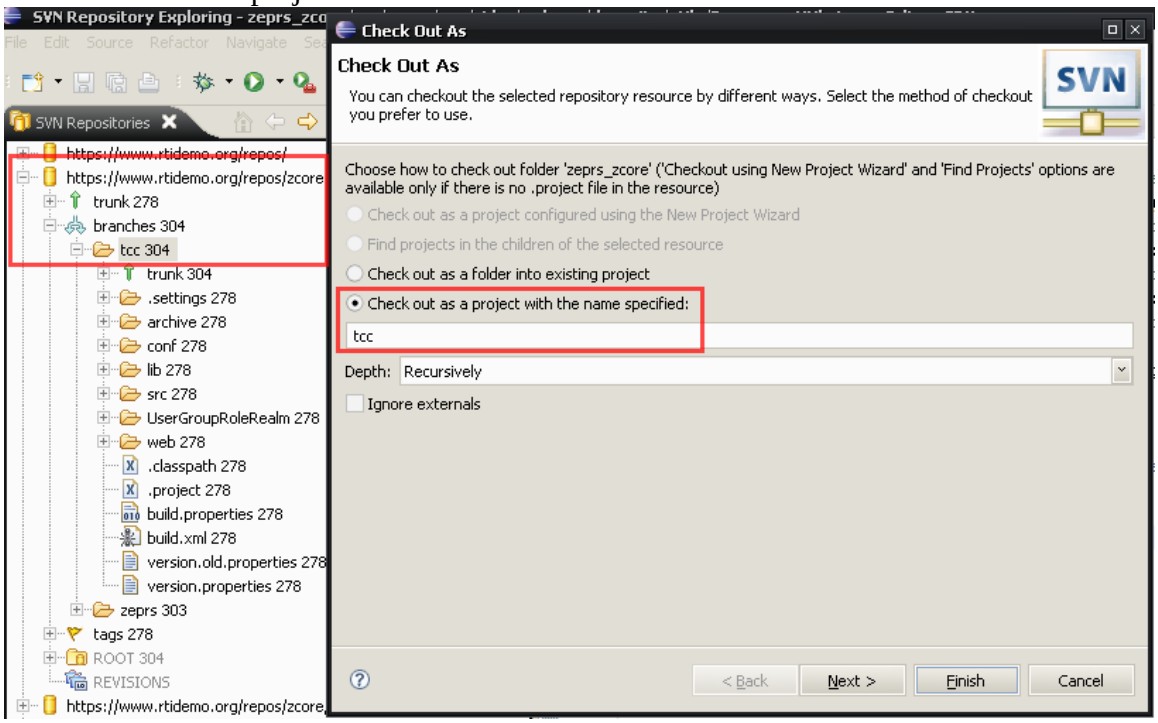
Chris Kelley
2/26/09

Creating a new Zcore-based Project 1
Eclipse Setup 1
Edit files while Eclipse is closed 2
Open the project in Eclipse 3
Copy templates and css files 21
Login and add a site 21
Importing forms 23
Creating new forms 24
Field names in Classes 24
Widget Tweaks 24
Set the correct path for the calendar widget 24
Pregnancy Dating widget 25
Modifications to the Patient Registration form 25
Extensions 25
Creating extensions 25
SessionPatientExtension 25
FormActionExtension 25
FormDAOExtension 26
Templates 26
Application Flow 26

Eclipse Setup

- Create a new branch of the zcore project in the subversion repository by copying the zcore trunk to a new branch.
  svn copy https://www.rtidemo.org/repos/zcore/trunk/
  https://www.rtidemo.org/repos/zcore/branches/tcc -m 'Creating tcc branch'

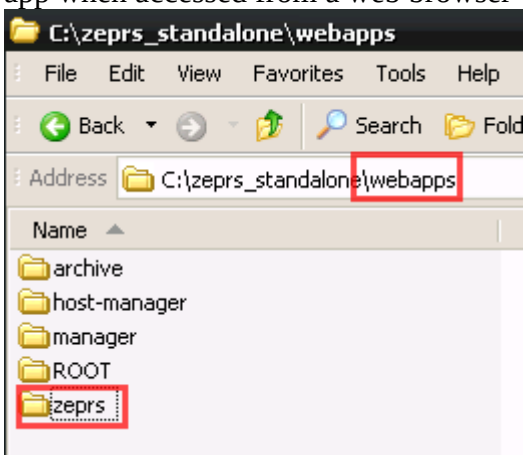
- Check out the new project.



- Once the project is checked out, close Eclipse. You will edit some files that should be edited without Eclipse running.

Edit files while Eclipse is closed

- Copy the directory that folds the "starter" Eclipse RCP app to a new directory. This is directory contains the standalone Eclipse RCP application as well as the zcore-based web application.
- Change the name of the webapp to the deployment name, i.e., the name the web app when accessed from a web browser – http://localhost/zeprs:



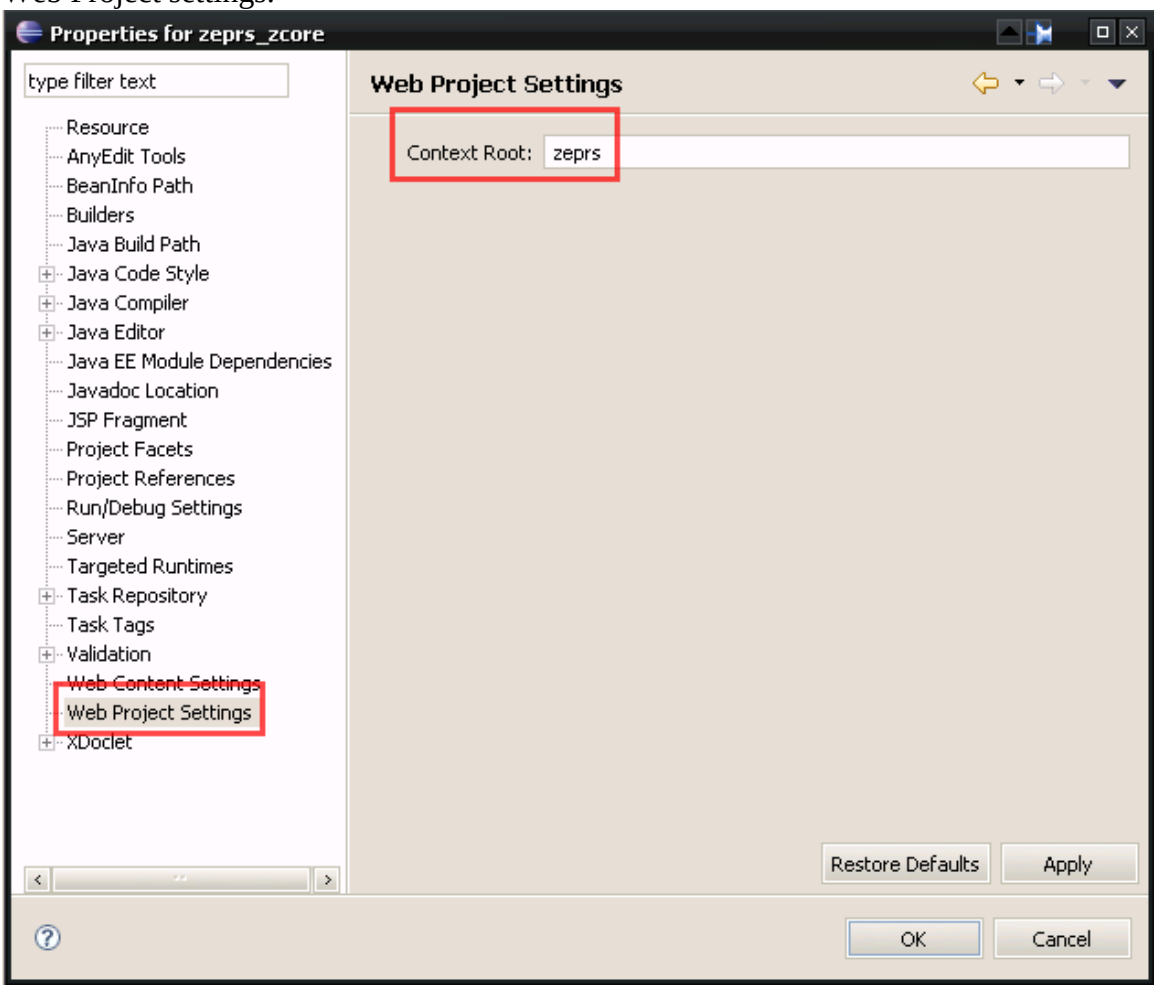
- Edit the .project file
  - Change the name of your project in a text editor. This will already be done if you checked out this branch via svn using Eclipse

    - Edit the path to the apache-work location. You probably just need to update the following highlighted text:
      <location>C:/zeprs_standalone/work/Catalina/localhost/zeprs/org</location>
- Edit the following files in the .settings/ directory:
    - org.eclipse.wst.common.component
        - change the deploy-name to your new project name
          <wb-module deploy-name="zeprs">
          This makes the correct name display when adding a web module.
        - <property name="context-root" value="zeprs"/>
    - org.eclipse.wst.common.project.facet.core.xml
        - change <runtime name="ZEPRS - standalone - Apache Tomcat v6.0"/>

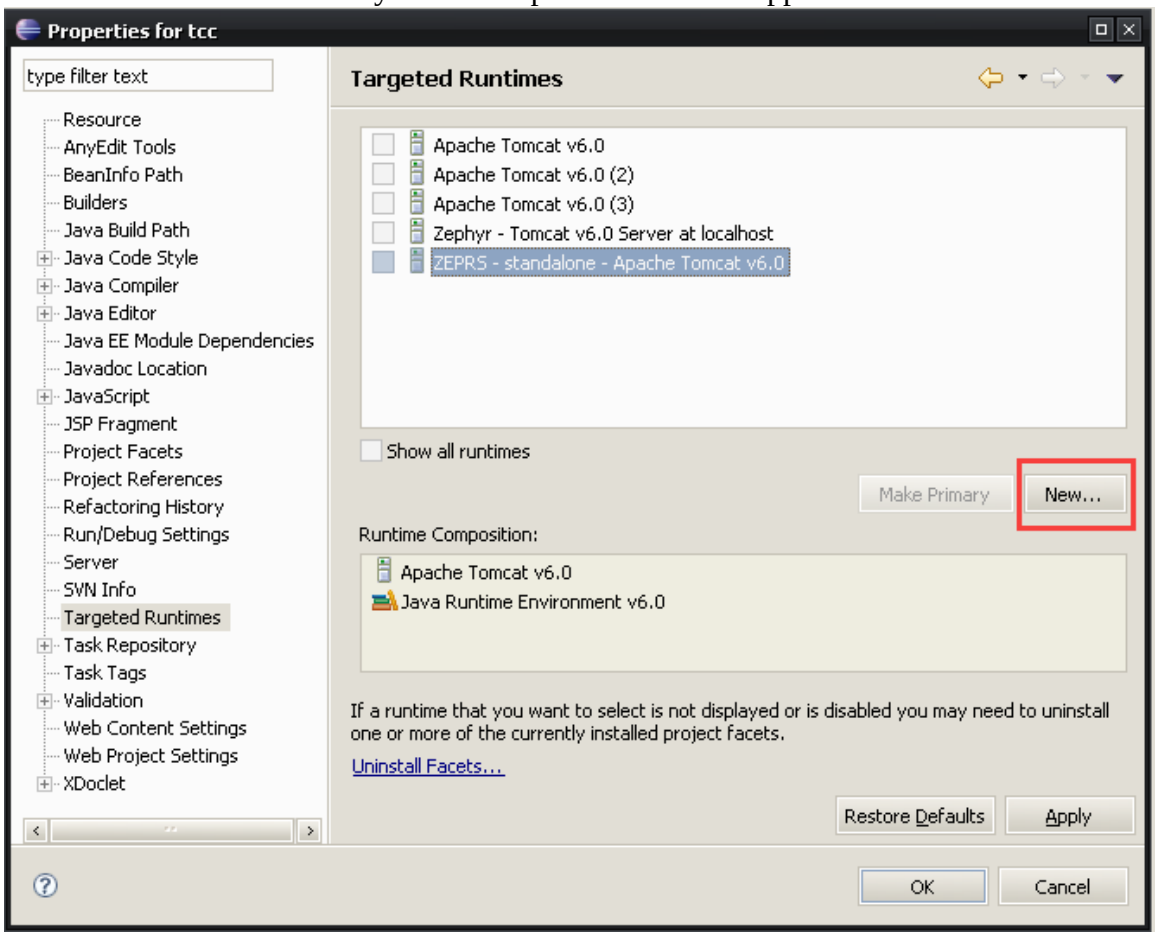
Open the project in Eclipse

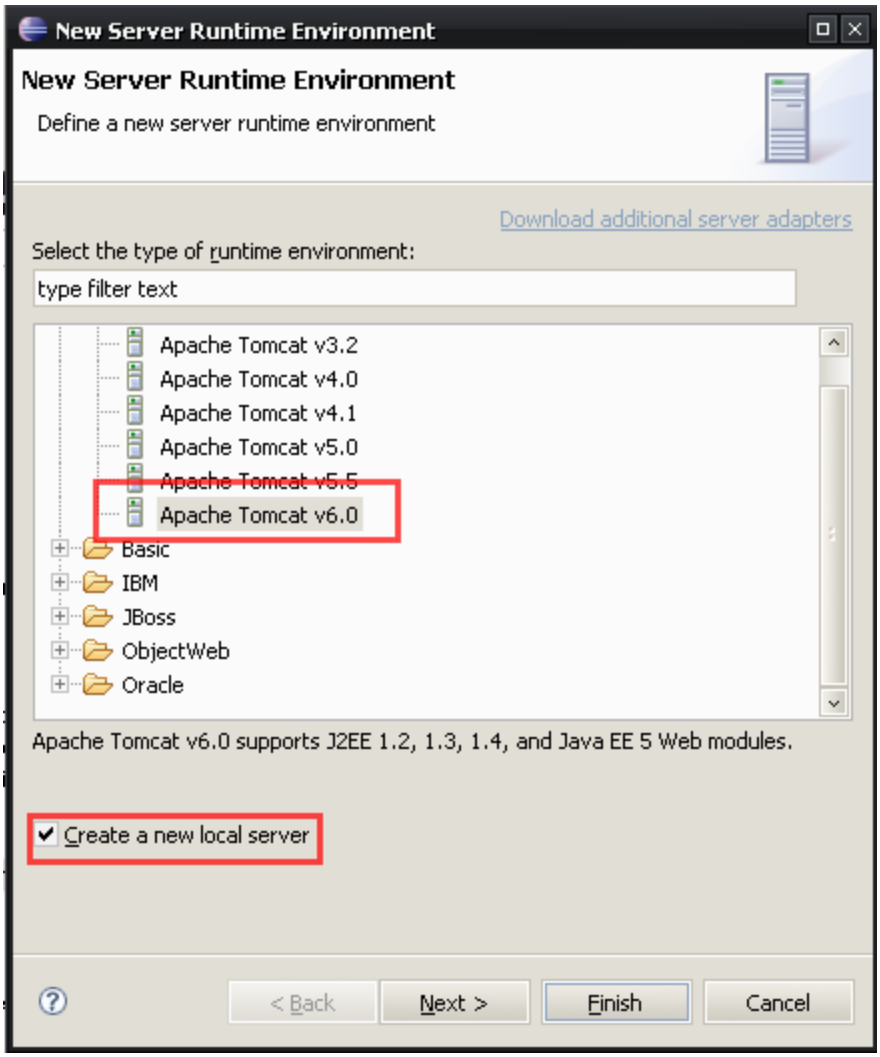
- Reset buildNum=0 and buildDate in version.properties and version.html
- Clear out all of the generated files from previous projects except for CreateReferral.java, PatientRegistration.java, UserInfo.java, and ReferralReasons.java
    - src\zeprs\org\cidrz\project\zeprs\valueobject\gen
    - src\zeprs\org\cidrz\project\zeprs\valueobject\report\gen – keep the files mentioned above, adding "Report" to the filename (PatientRegistration Report.java…)
- Clear out all of the XML files in src\zeprs\resources\xml\forms except for:
    - Activefields
    - Clinics
    - CreateReferral
    - Fields.xml
    - FormTypes.xml
    - PatientRegistration.xml
    - ReferralReasons
    - Sites
    - UserInfo
- Copy the application directory, which contains the tomcat instance, eclipse rcp code, database, and other useful files, to the deployment point (e.g. C:\ zeprs_standalone).
- Setup the Eclipse server settings.

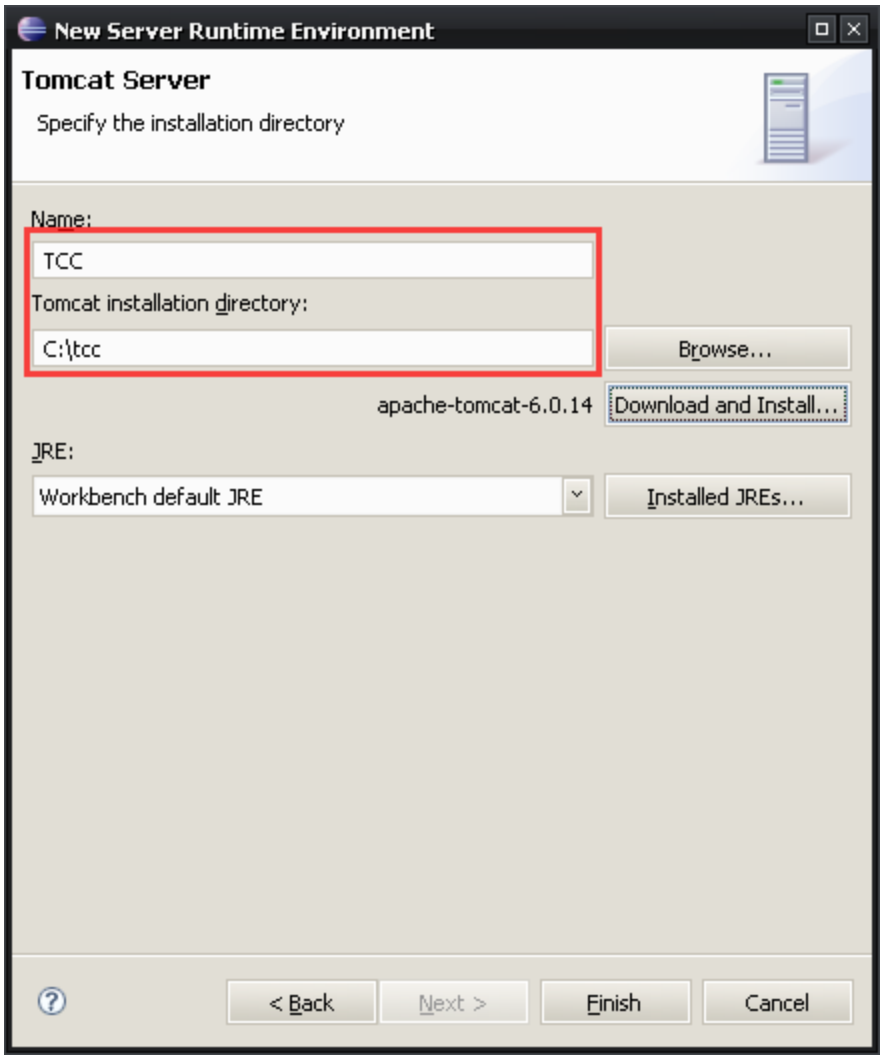
- Web Project settings:



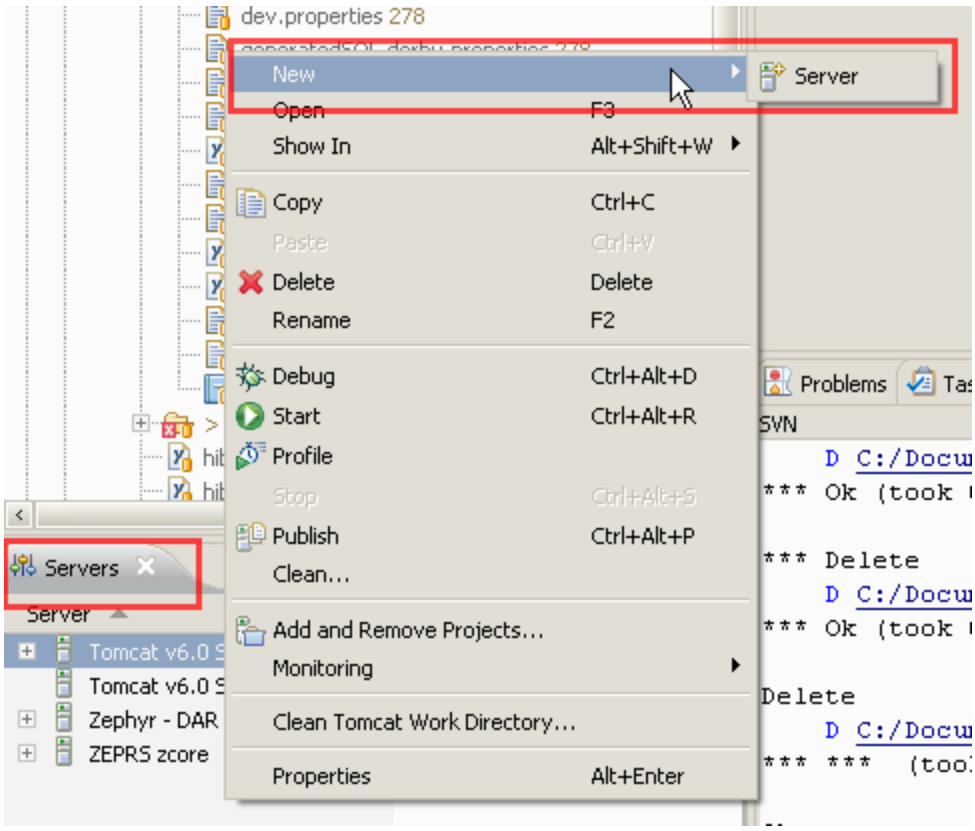
- Add a new server instance – you need to point to this new app.

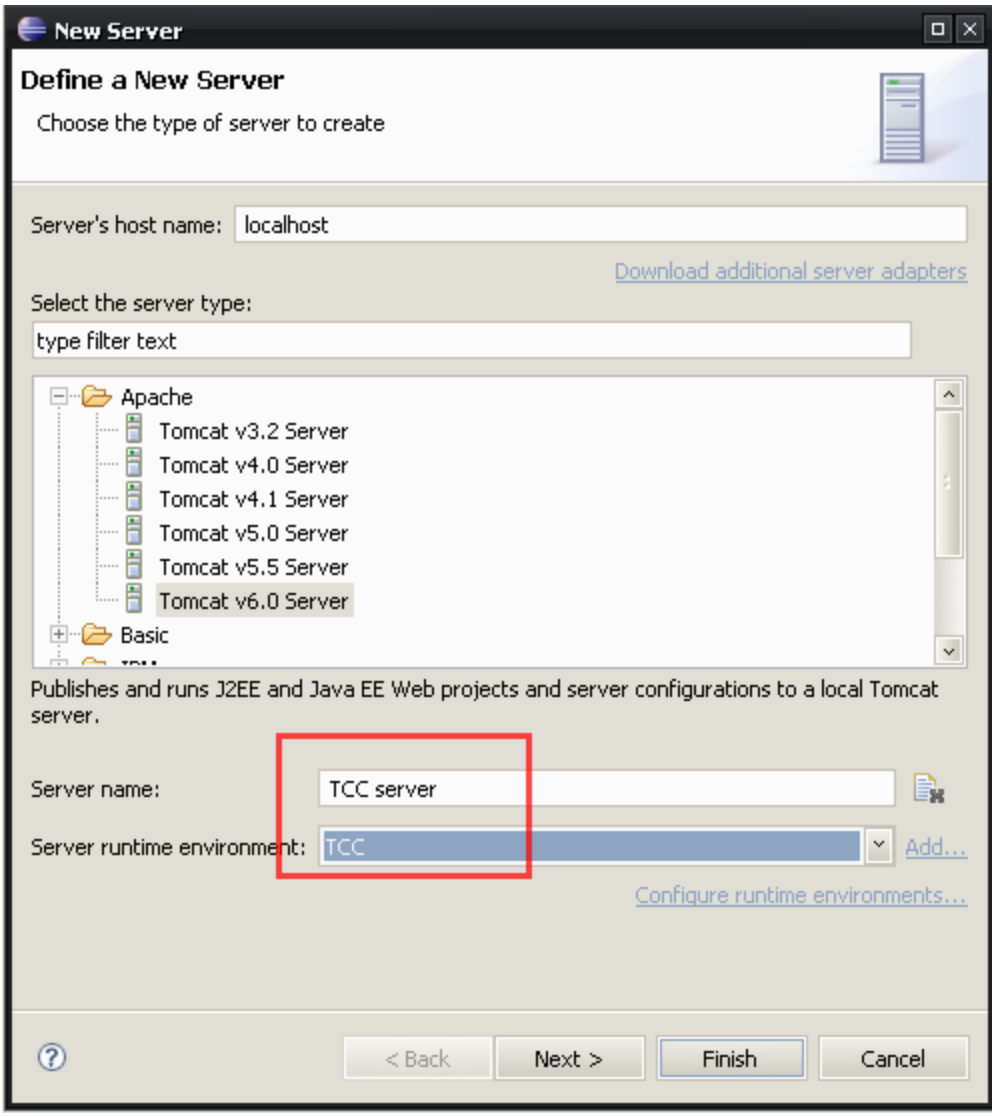



New Server Runtime Environment
New Server Runtime Environment
Define a new server runtime environment
Download additional server adapters
Select the type of runtime environment:
type filter text
Apache Tomcat v3.2
Apache Tomcat v4.0
Apache Tomcat v4.1
Apache Tomcat v5.0
Apache Tomcat v5.5
Apache Tomcat v6.0
Basic
IBM
JBoss
ObjectWeb
Oracle
Apache Tomcat v6.0 supports J2EE 1.2, 1.3, 1.4, and Java EE 5 Web modules.
Create a new local server
< Back
Next >
Finish
Cancel

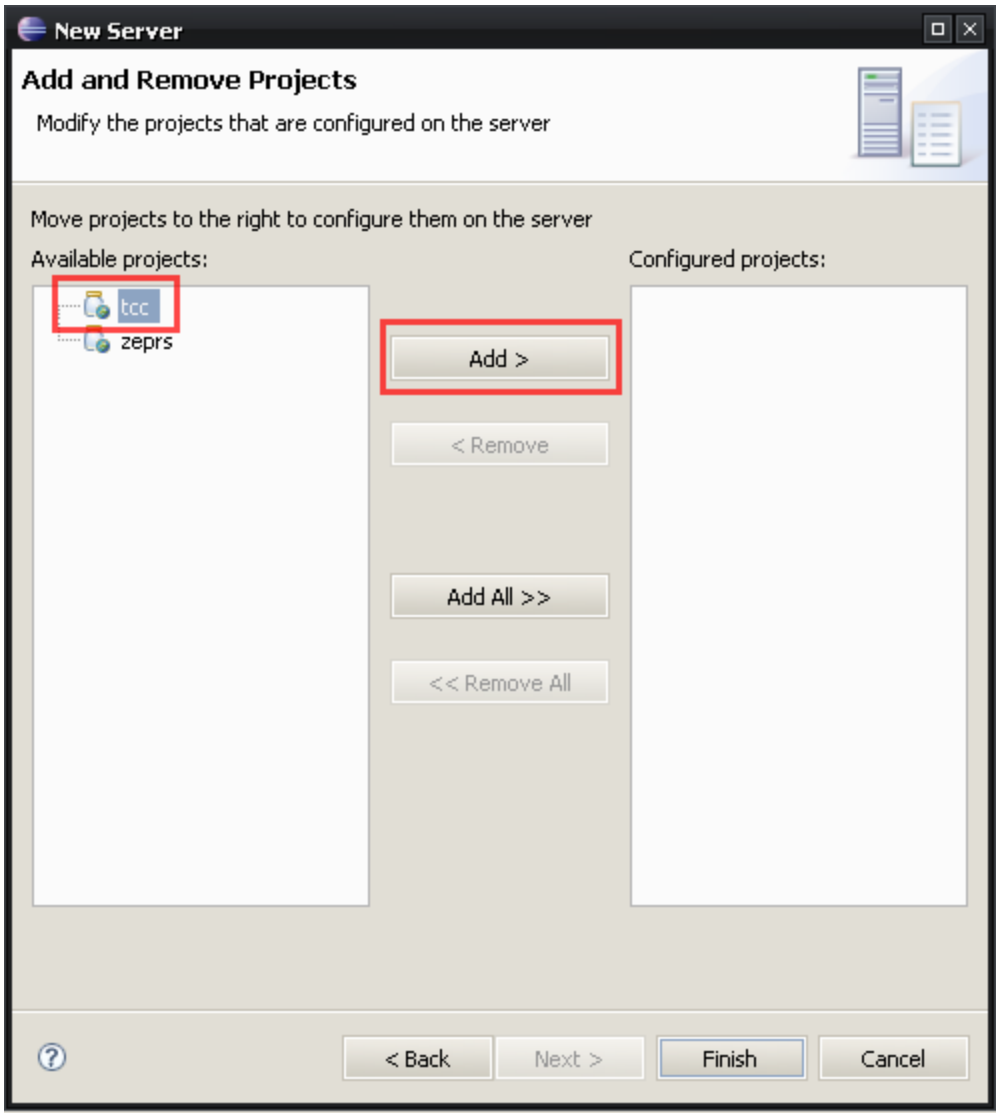

New Server Runtime Environment
Tomcat Server
Specify the installation directory
Name:
TCC
Tomcat installation directory:
C:\tcc
Browse...
apache-tomcat-6.0.14
Download and Install...
JRE:
Workbench default JRE
Installed JREs...
< Back
Next >
Finish
Cancel

- Create a new Server in the Servers view:

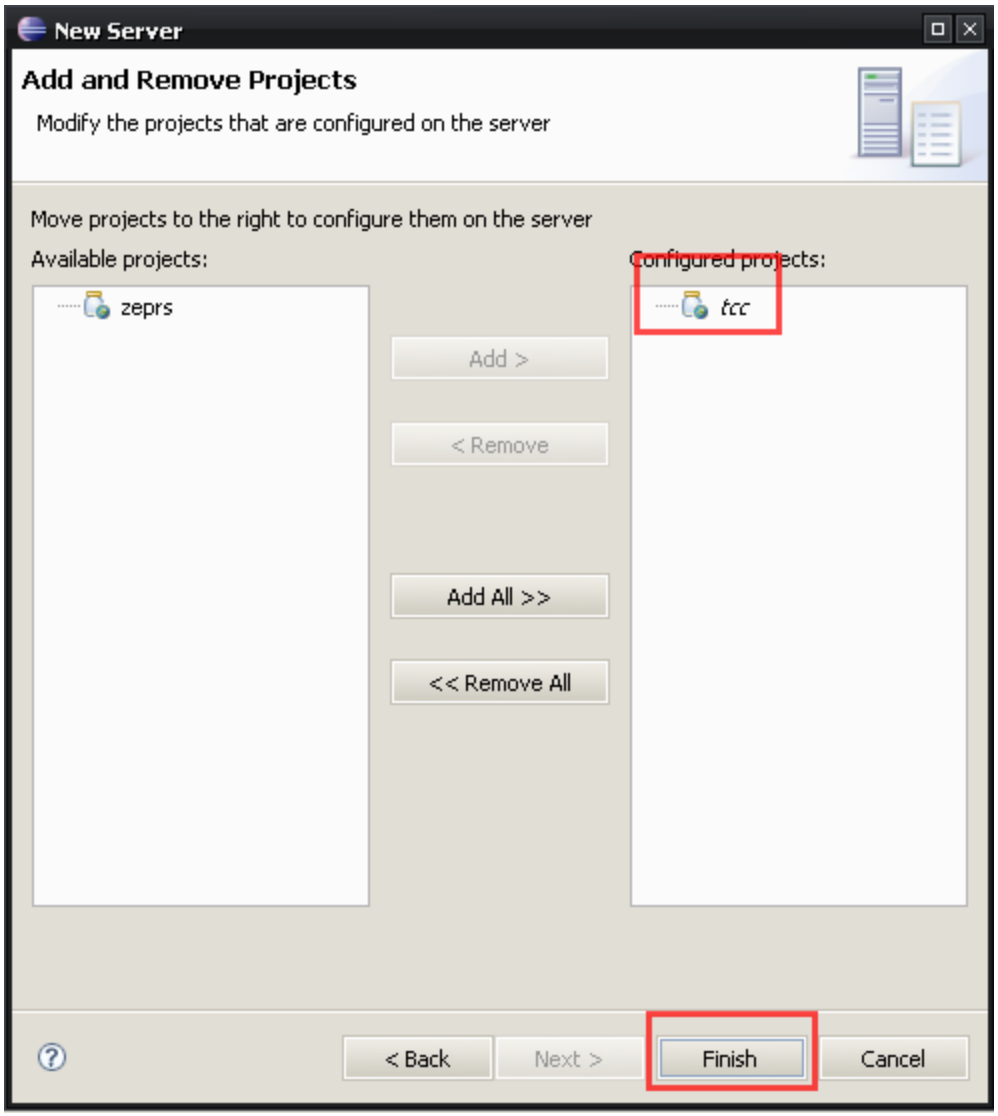




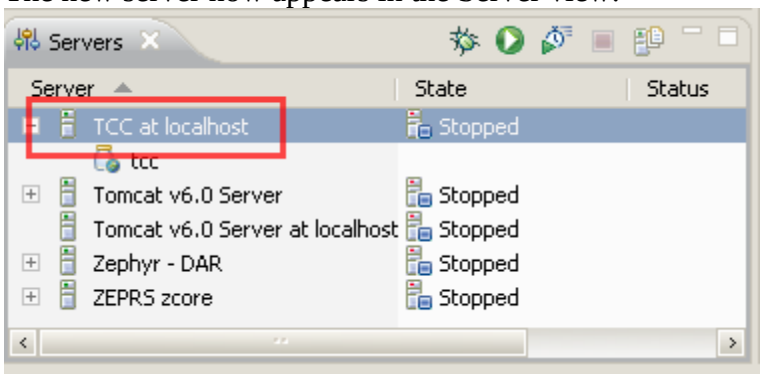
Click on the new server and click “Add”.



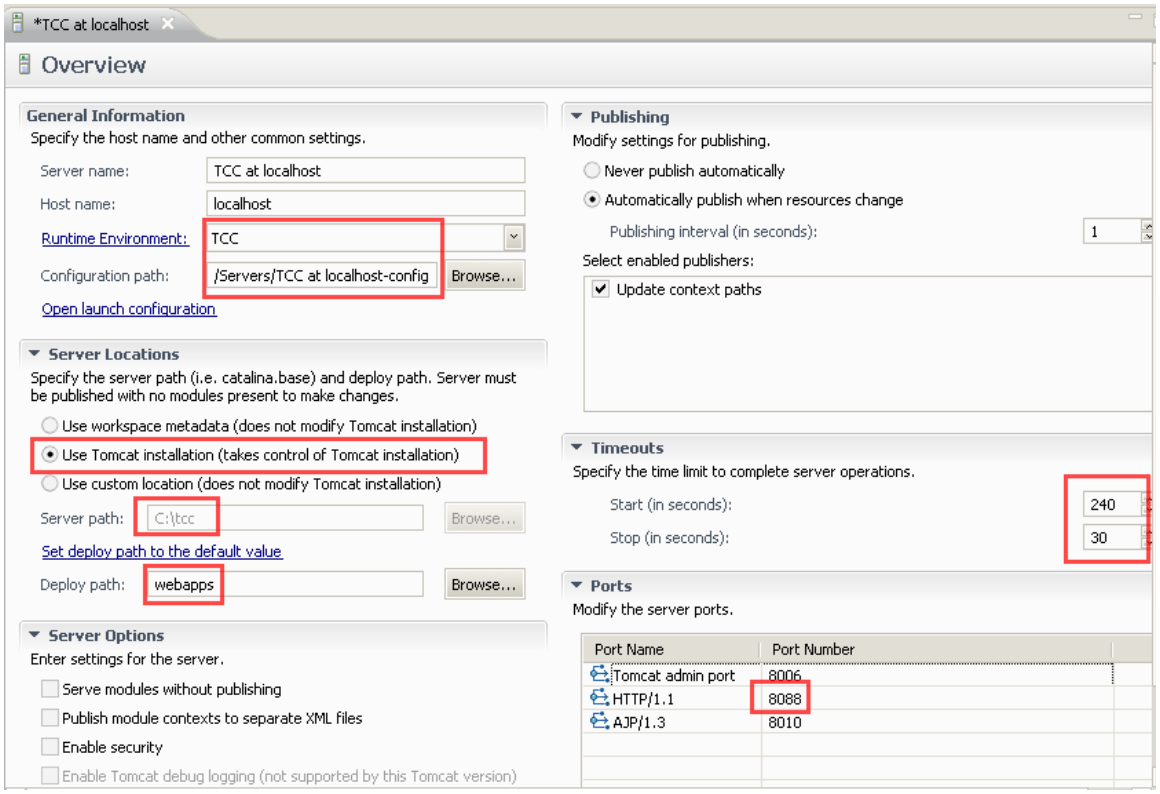
Click finish



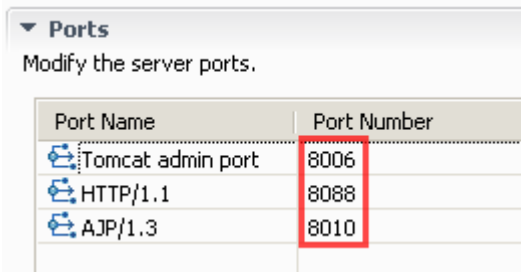
- The new server now appears in the Server view:



- Double-click on the new Server to edit its parameters:

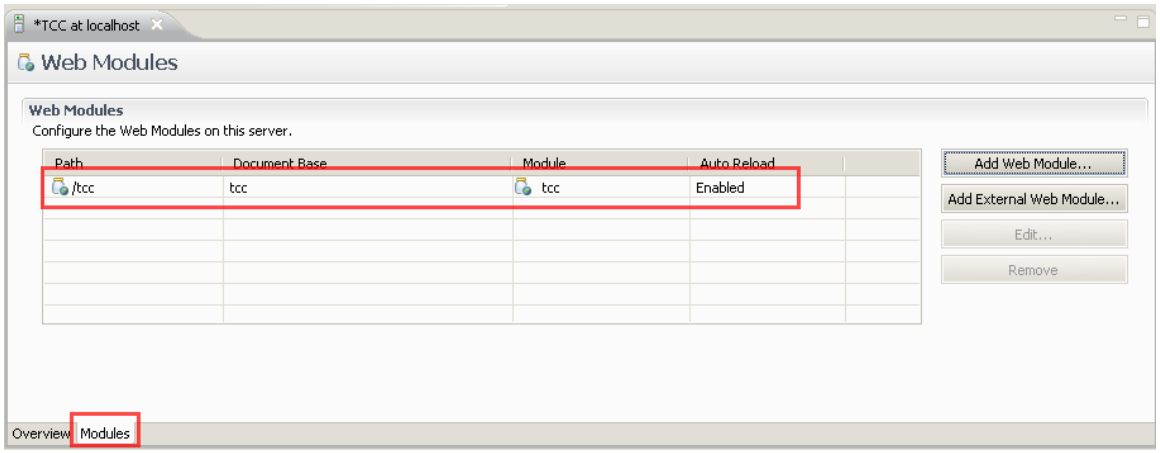


Please note that if you are also developing a separate instance of a zcore data center, change the Tomcat admin port and AJP port as well:

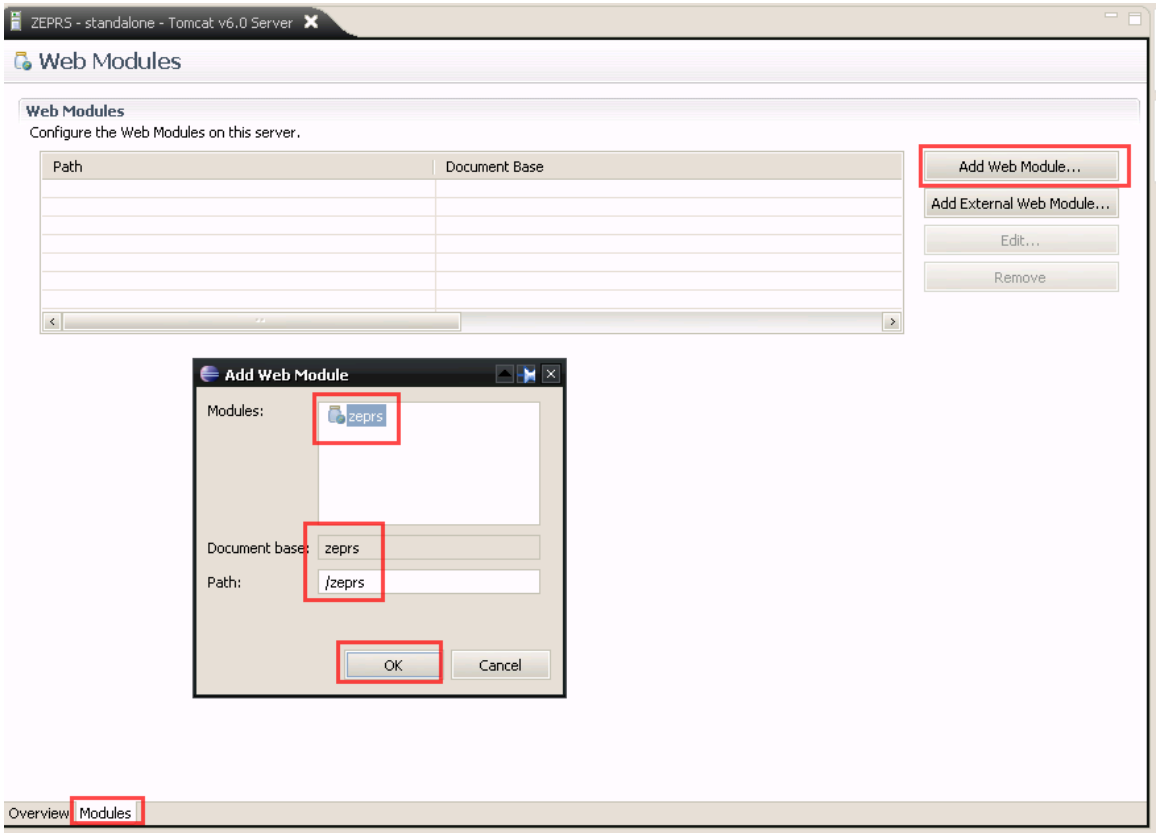


This will enable you to run both instances at the same time.

Click the Modules tab at the bottom of the window to confirm that your new module has been added:

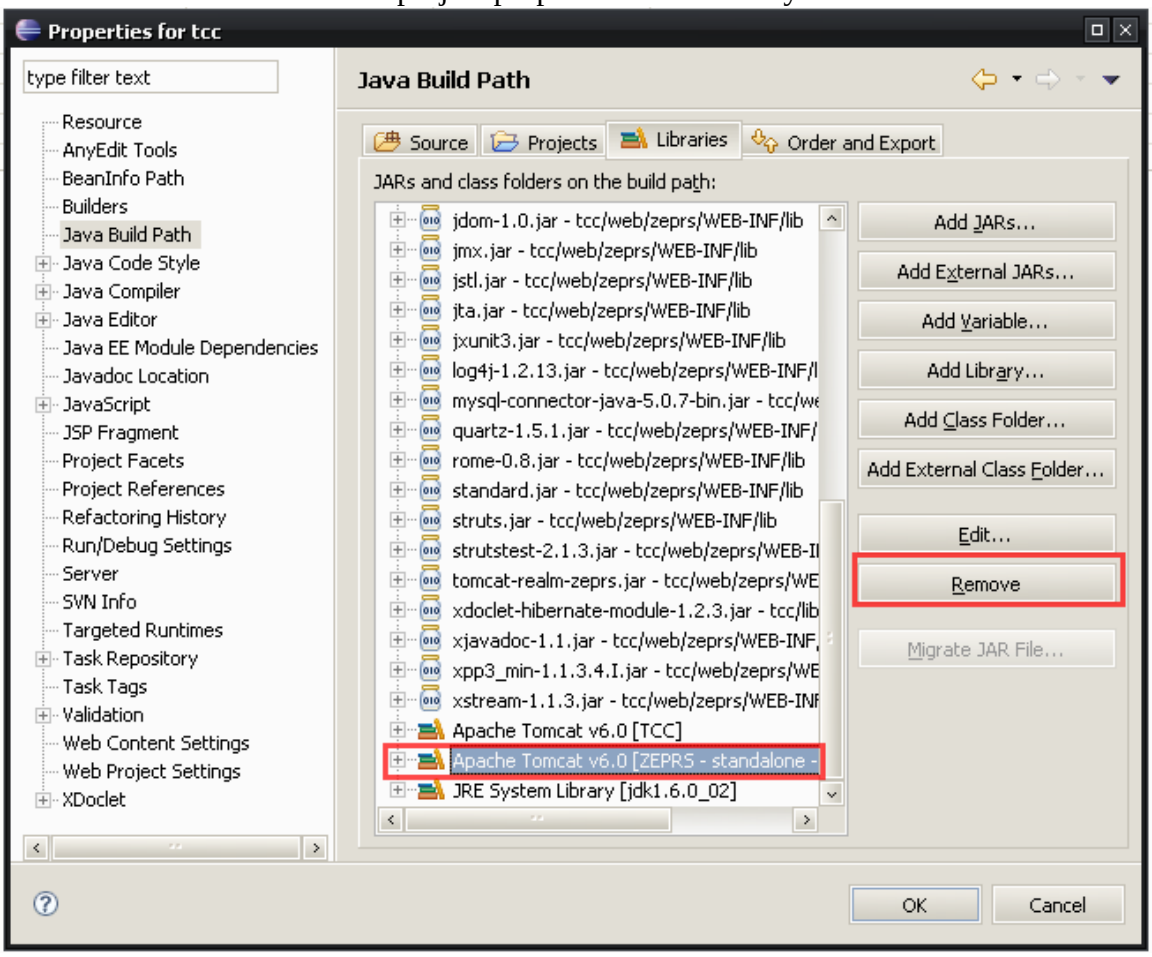


If it hasn't been added, click "Add Web Module":

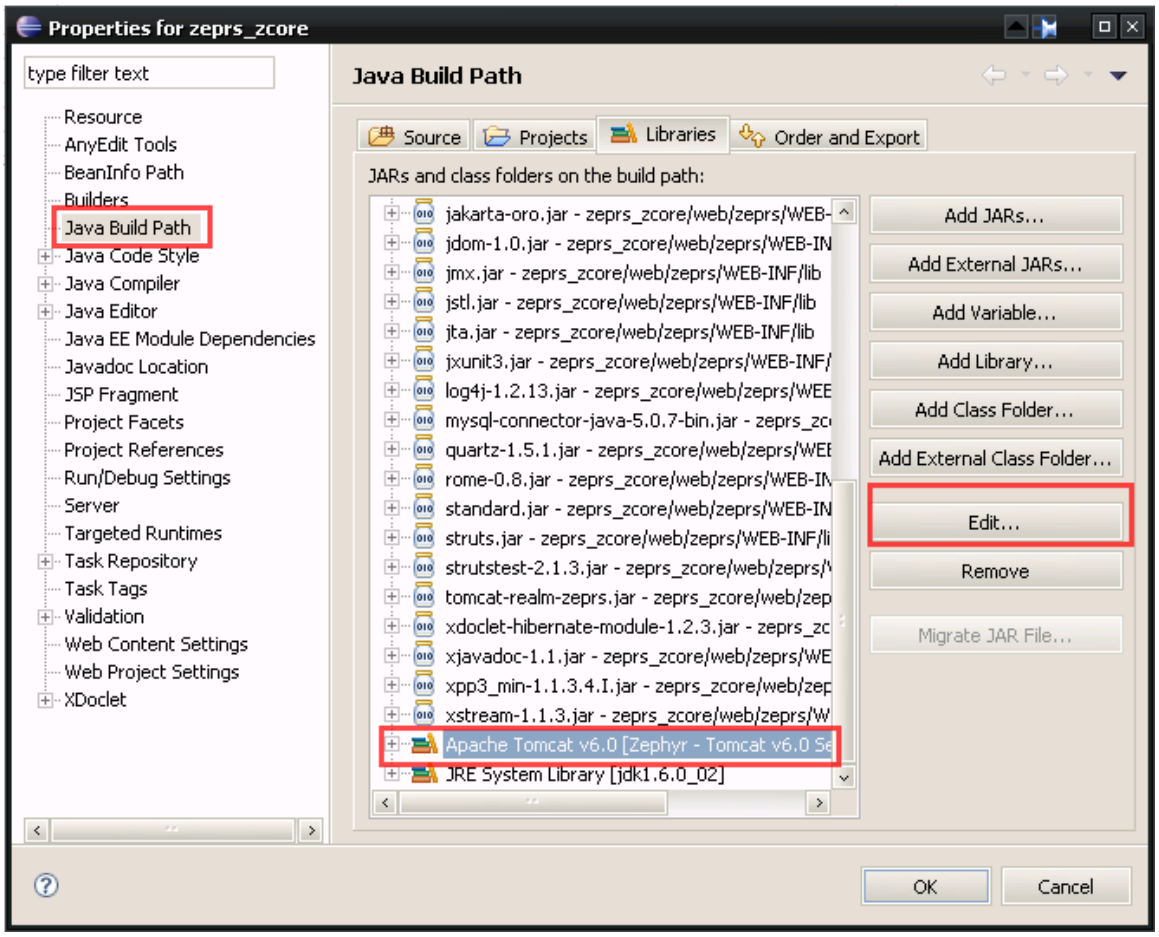


Press Save to save these config. Settings.

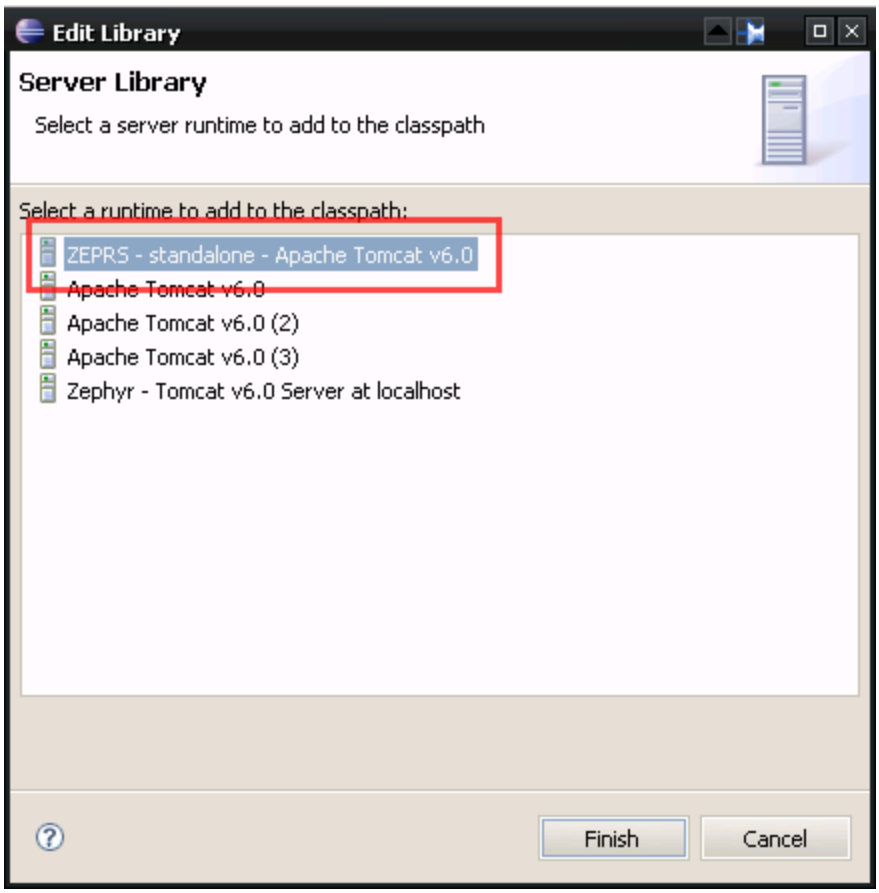
- Edit the Java Build Path in the project properties. Remove any incorrect servers.

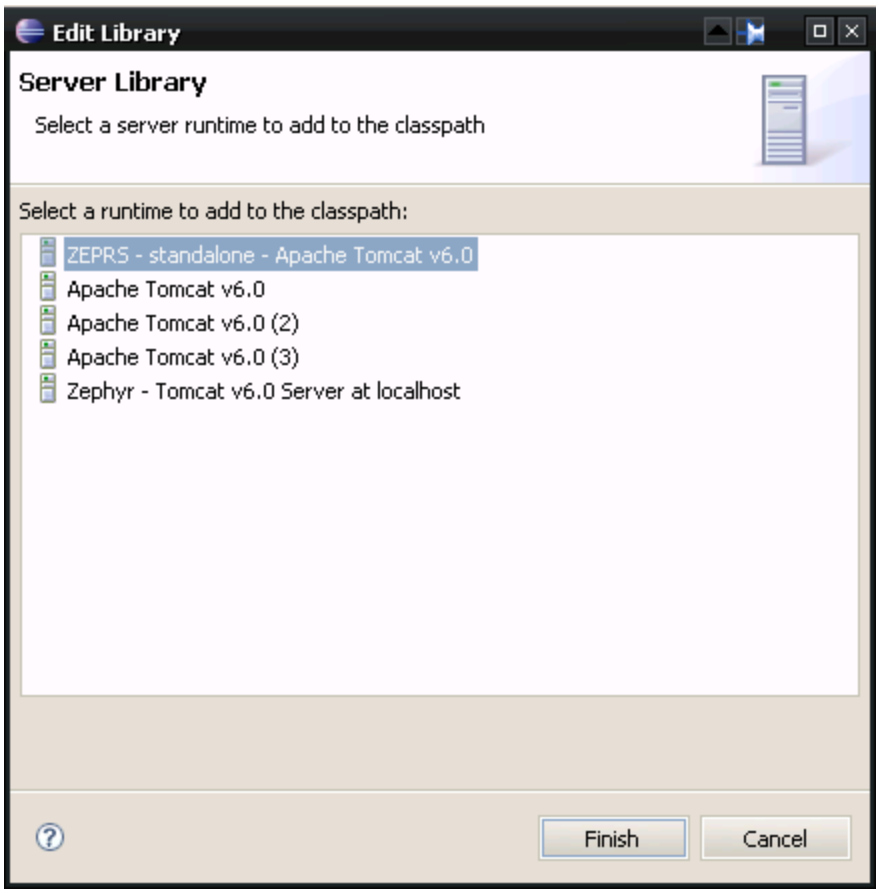


If your new server is not displayed, press Edit.



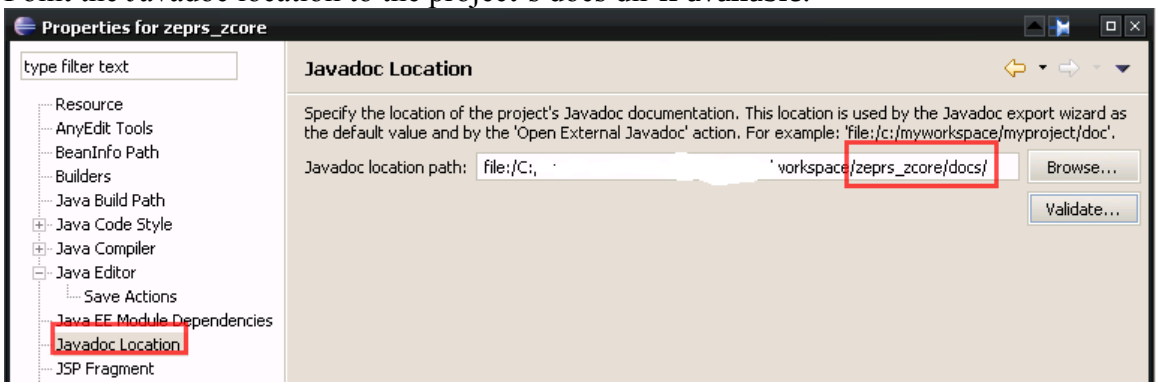
Choose the new server runtime you just setup:


Edit Library
Server Library
Select a server runtime to add to the classpath
Select a runtime to add to the classpath:
ZEPRS - standalone - Apache Tomcat v6.0
Apache Tomcat v6.0
Apache Tomcat v6.0 (2)
Apache Tomcat v6.0 (3)
Zephyr - Tomcat v6.0 Server at localhost
Finish
Cancel

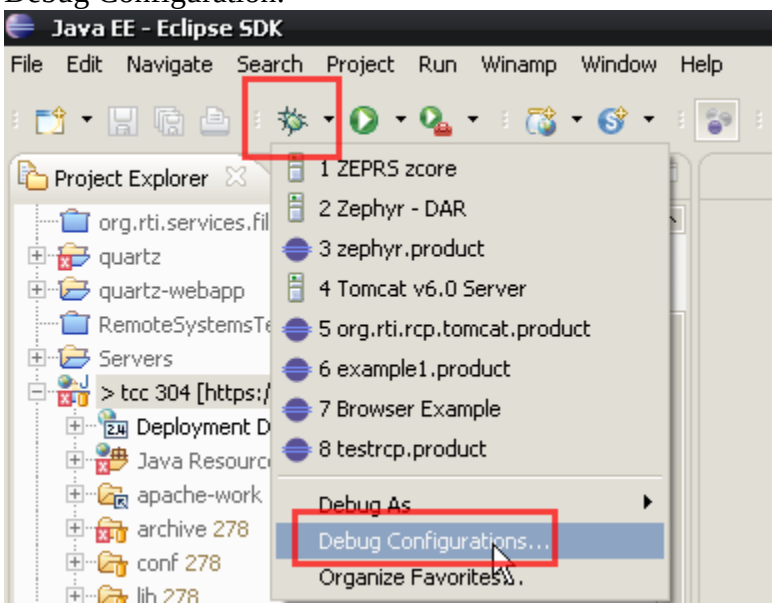


After you press Finish it will take a few minutes to compile the classes and display any errors about missing classes. The missing classes are likely due to the cleaning out the generated classes. This is OK. Your project probably will not call on the methods that use those classes.

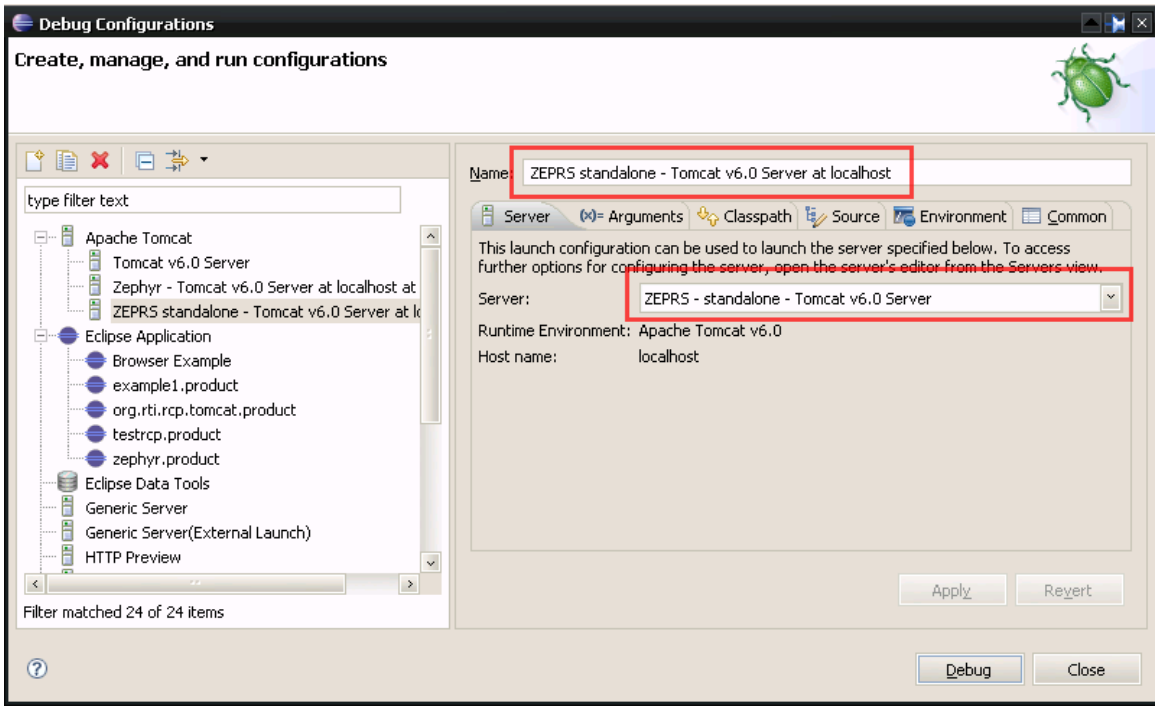
- Point the Javadoc location to the project's docs dir if available:



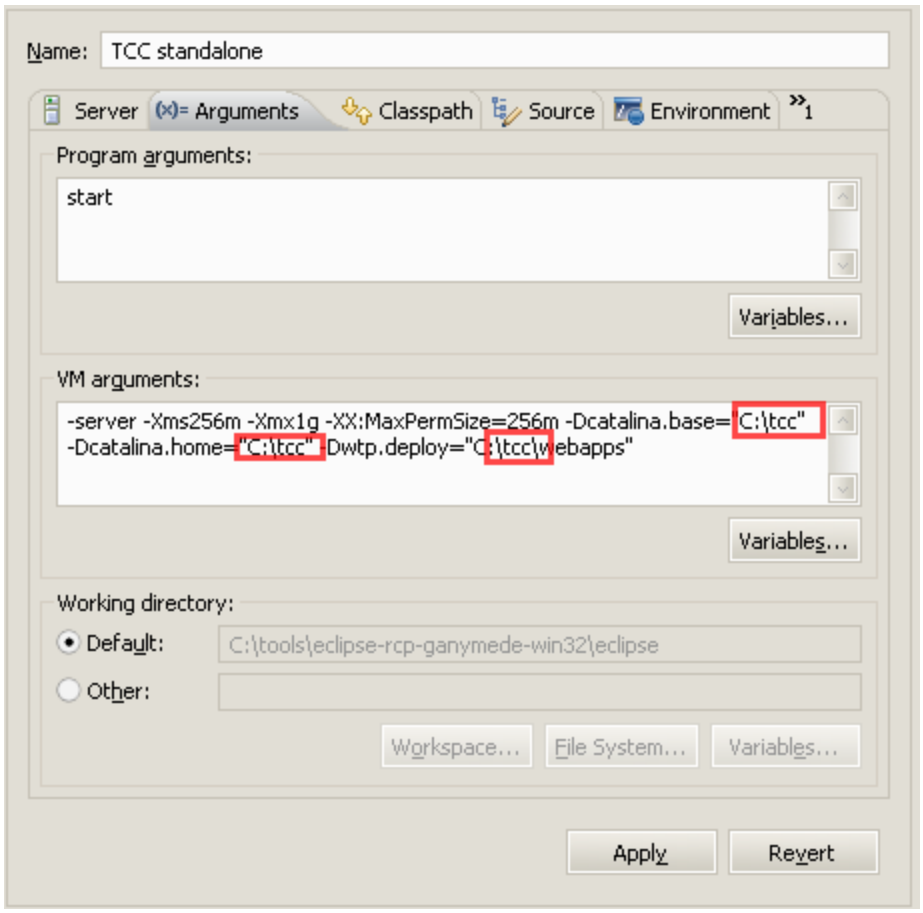
- Debug Configuration:



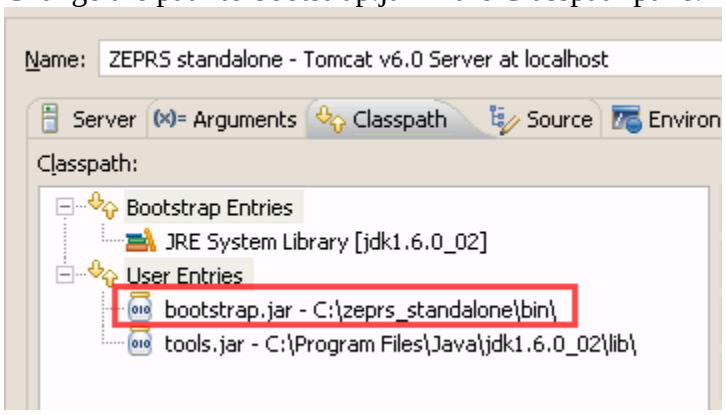
Select the server from the Server dropdown:



Change the path to the app in the Arguments pane:



Change the path to bootstrap.jar in the Classpath pane:



- Open web.xml and change the web-resource-name in the following section to the name of the deployed web app:

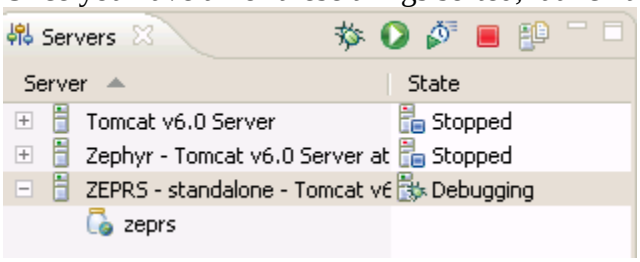
```
<security-constraint>
     <web-resource-collection>
          <web-resource-name>zeprs</web-resource-name>
          <url-pattern>*.do</url-pattern>
          <http-method>GET</http-method>
          <http-method>POST</http-method>
     </web-resource-collection>
     <auth-constraint>
          <role-name>*</role-name>
     </auth-constraint>
 </security-constraint>
```

Also here:

```
<security-constraint>
     <web-resource-collection>
          <web-resource-name>zeprs/admin</web-resource-name>
          <url-pattern>/admin/*</url-pattern>
          <http-method>GET</http-method>
          <http-method>POST</http-method>
     </web-resource-collection>
     <auth-constraint>
          <role-name>ALTER_PROGRAMS_AND_SCREEN_APPEARANCE</role-name>
     </auth-constraint>
 </security-constraint>
```

- Open src/resources/application.properties and edit the first 3 properties: app.name, app.template, and app.title.
- Open src/resources/dev.properties and edit source and install properties
- Open build.properties (use for building wars and other automation with ant) and edit the first 2 properties:
    - project=zeprs
    - install.dir = C:\\zeprs_standalone\\
- Open web/zeprs/META-INF/context.xml and change path and docBase in the second line:
  <Context path="/zeprs" docBase="zeprs"
- Edit web/zeprs/WEB-INF/pages/version.html. This file, which displays application data in the title bar, is normally built automatically when you run the build ant target; however, when you're starting out, it's nice to see the correct project name in the browser's title bar. Change the project name, build date, etc. - ZEPRS 1.0 - buildDate: 2008/10/29 20:15 - buildNum: 1
- Edit js/javascript.js and replace the project name in the first line:
  imgsrc = "/tcc/images/";

Once you have all of these things sorted, launch the app in Debug:



You should be able to login and run the site config. Most of the form-related functionality won't work yet, unless you branched from an instance w/ forms. Creating a new patient should work however.

Copy templates and css files

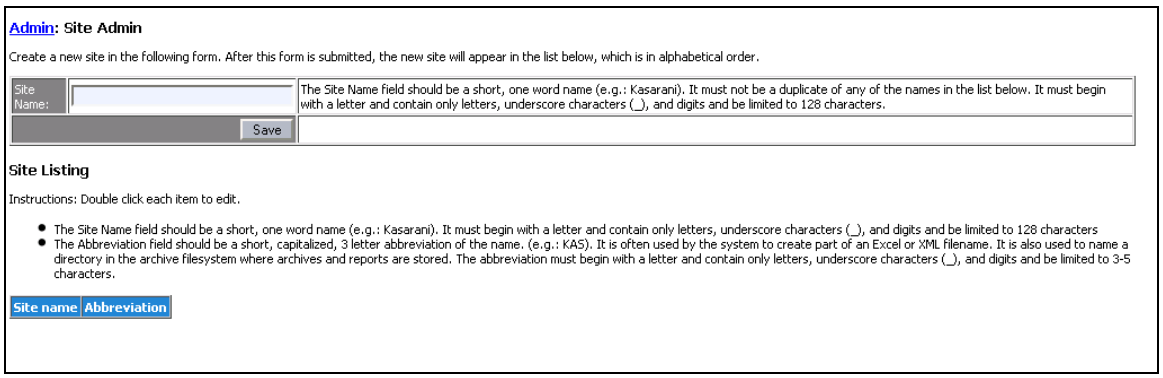
Create a new directory in web/zeprs/WEB-INF/templates using the same name you used for the app.template property. Copy all of the files in one of the other css dirs (zeprs is a good choice) into this new dir.

Create a new directory in web/zeprs/css using the same name you used for the app.template property. Copy all of the files in one of the other templates dirs (zeprs is a good choice) into this new dir.

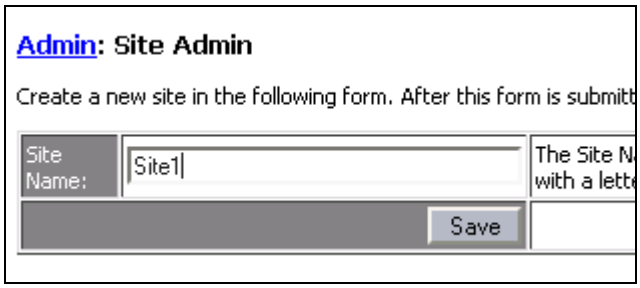
Login and add a site

If you have copied an instance that already has sites, this step may not be necessary.

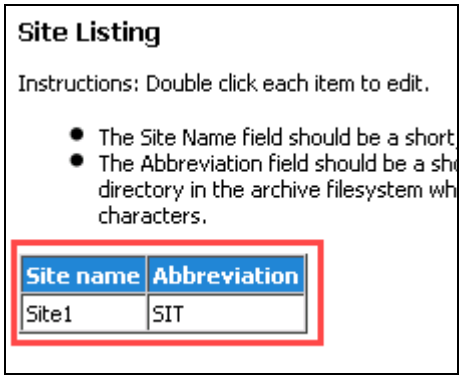
Login as zepadmin. The system will take you directly to the Site Admin page to add a site.



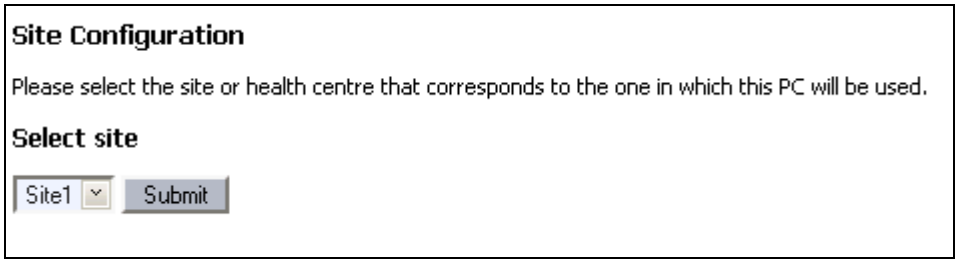
Enter a site name:



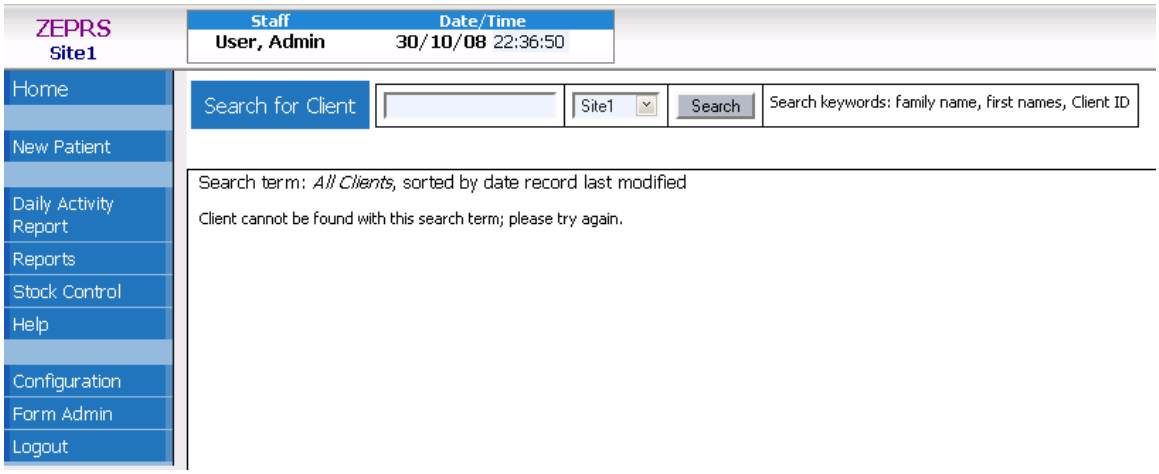
After you have saved the site, the system will display the site in the site list:



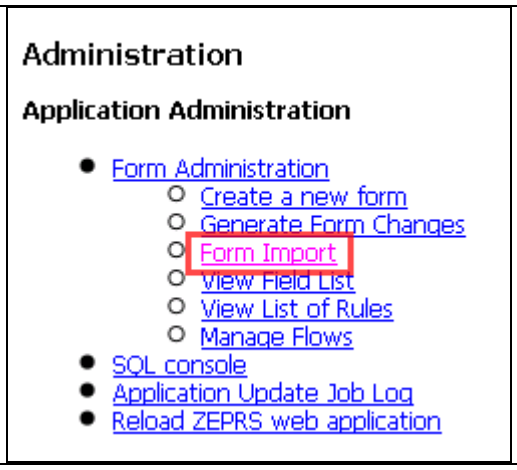
Click the ZEPRS link at the top right corner to configure this new site as your browser's site.



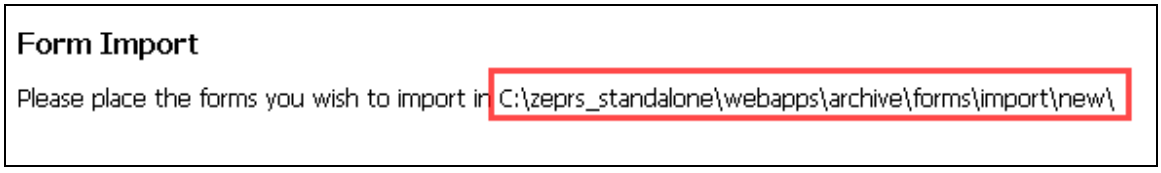
Once this is set, the home page will display:



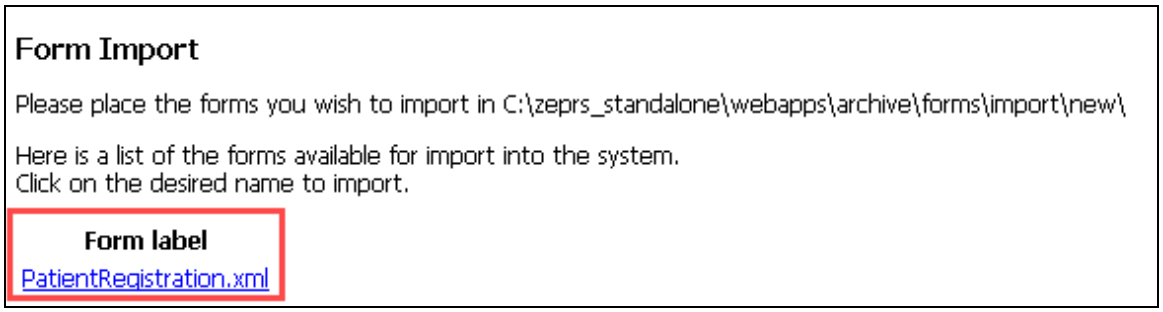
Importing forms

From the left nav strip click Admin and then click “Form Import” under “Form Administration.	

Note the location of where you must place the forms.



Place the PatientRegistration.xml form into the import/new directory and refresh the form import page. The system will now list the new form:



Click on the link to import the form.

Creating new forms

Consider appending the project name to the beginning of the table name when creating a new form. For instance, if the project name is tcc, name the table "tcc_form_name."

Field names in Classes

Each form has its own associated class, xml file, and entries in the struts-related files. When Dynasite generates the classes files for a form, it must create the field names for each form field. See DynaSiteGenerator. generateSource. If the form was imported – it checks for form.importId – the method concats the field id to "field." New forms – ones not imported – use the table name (StringManipulation.firstCharToLowerCase(formField.getStarSchemaName())).

In the same vein, when the app starts up, the system creates an identifier for each field (code in DynaSiteObjects. getFieldToPageItem) This identifier is used for the html object id <td id="${pageItem.form_field.identifier}" …") For new forms, the column name for the field is the identifier. In earlier version of zcore such as ZEPRS, the field name is a concatenation of "field" and the auto-generated field id. If a ZEPRS form is imported into a new instance, the import id – the original field id - is used instead for the field name. This keeps code for reports that use fields of the imported file working correctly; otherwise, one would need to rename the field names in the code that accesses the fields, because the field id would be different.

Todo: assignment of field name is too tightly bound to importId. Forms created in projects created after ZEPRS should use the column name, not the importId

Widget Tweaks

Set the correct path for the calendar widget

Change the following path: imgsrc = "/zeprs/images/";

Pregnancy Dating widget

You normally don't need to do anything to get this form to work. If you edit this form using the admin interface and change the numeric values (used for rules processing) for the "Dating Method" field, you'll need to update the switch statement in the javascript method processDatingMethod in pregnant.js

Modifications to the Patient Registration form

If you add fields to the Patient Registration form, you may wish to add some of the new fields to the patient table. You will also need to add these fields to FormDAO. createPatient's patientValues ArrayList. Also adjust SQL_CREATE_PATIENT in patientSQL.properties. Although the developer considered making an extension (see below) to the createPatient method, it seemed clearer to keep all of these values in one place. Plus, modifying the patient table may not be common to many other projects.

Extensions

In order to keep zcore from becoming too dependent on other project classes, an extension facility is available that can be used to add functionality to certain objects. The extensions are stored in org.cidrz.webapp.dynasite.utils.extensions package.

Creating extensions

Some guidelines:

- Name the new extension class similar to the class you wish to extend.
- Implement Extension class.
- The execute method takes a Connection and an Object. Cast your object to the object that you're passing within the method.

SessionPatientExtension

The updateSessionPatient in SessionPatientDAO calls SessionPatientExtension.execute. The execute method calls some pregnancy-related methods. If you are working on a project that does not need these methods, comment them out.

FormActionExtension

This class extends Action and handles form errors, persistence, dynaform value assignment, and other mapping issues.

The createForward method provides custom forwarding of servlets.

FormDAOExtension

This class implements Extension; however, the execute method is unimplemented.. The method updatePatientValues provides a useful post-processing hook after persisting a form. There may be several hooks useful for FormDAO.

Templates

Edit the templates that are in the template directory corresponding with app_name for your project – web/zeprs/WEB-INF/templates/app_name

- template-home – home page
- sidenav-full – side nav strip when viewing patient record

Application Flow

When the user clicks a link to a patient record, PatientHomeAction determines the most appropriate form or task list to which to send the user. Edit this class as needed to suit your app's flow.