

Getting started with Geomajas

Geomajas Developers and Geosparc

Getting started with Geomajas

by Geomajas Developers and Geosparc

1.11.0-SNAPSHOT

Copyright © 2010-2012 Geosparc nv

Table of Contents

1. Introduction	1
1. About this document	1
2. About this project	1
3. License information	1
4. Author information	1
2. Starting a new GWT based Geomajas project	3
1. Prerequisites / Command line	3
1.1. Creating the template project	3
1.2. Testing the template project	5
2. Eclipse	6
2.1. Running/debugging with the Google Plug-in for Eclipse (embedded Jetty option)	6
2.2. Running/debugging with the Google Plug-in for Eclipse	11
3. IntelliJ IDEA	11
4. NetBeans	14
5. Using a Hibernate layer	14
6. How to continue	14
3. Real world example: marine application	16
1. Introduction	16
2. Preliminary note	16
3. Setting up a Geomajas based project	16
4. Showing your own shape files in Geomajas	17
5. Show your own PostGIS data in Geomajas	20
6. Show your own WMS map using Geomajas	23
7. Show locations of vessels using SOAP	24
8. Show information about a lock using FacililyXML	28
9. User management	30

List of Figures

2.1. Choose the correct archetype	4
2.2. Screenshot when building the Geomajas GWT archetype	5
2.3. Import project as Maven project	7
2.4. Eclipse project properties dialog, Google Web Toolkit	8
2.5. Eclipse project properties dialog, Google Web Application	8
2.6. Debug configurations dialog	9
2.7. JettyRunner as main class	9
2.8. Running Jetty	10
2.9. Running the GWT application	10
2.10. Classpath of GWT plug-in (no-server mode)	11
2.11. Open project using pom	12
2.12. IDEA GWT run configuration	12
2.13. run gwt:i18 target	13
2.14. Project structure for simple GWT project	13
3.1. Update your project as a GWT project in eclipse after each mvn eclipse:eclipseA	17
3.2. Add vhaLayer to the list of layers	18
3.3. Add vhaLayer to the list of treeNodes	18
3.4. Add ClientVectorLayerInfo bean	19
3.5. Add a reference to layerVha.xml in your web.xml	19
3.6. Create a symbol for MULTIPOINT or POINT layers	20
3.7. Add appDataSource to applicationContext.xml	21
3.8. Add appSessionFactory to applicationContext.xml	21
3.9. Add hibernateEcdisLayer to the list of layers	21
3.10. Add hibernateEcdisLayer to the list of treeNodes	22
3.11. Add ClientVectorLayerInfo bean	22
3.12. Enable PostgisDialect	22
3.13. Add a reference to layerEcdis.xml in your web.xml	23
3.14. Add ClientVectorLayerInfo bean	23
3.15. Show vessels using an Image marker.png	25
3.16. Resize the images depending on the zoom level	26
3.17. Controller that shows extra information about a clicked vessel	27
3.18. Controller that shows extra information about a clicked vessel	29
3.19. Security pom.xml	30
3.20. Security inherit	30

List of Examples

2.1. Create project using GWT Maven archetype	3
2.2. Create project using GWT Maven archetype	3
2.3. Creating a build from your project	5
2.4. Create a build, then run it	5
2.5. Run the template application in development mode	6
3.1. Create project using GWT Maven archetype	17
3.2. Use ogr2ogr to convert Ecdis files to PostGIS	20

Chapter 1. Introduction

1. About this document

Documentation for developer who want to use and extend the Geomajas GIS framework.

2. About this project

Geomajas is a free and open source GIS application framework for building rich internet applications. It has sophisticated capabilities for displaying and managing geospatial information. The modular design makes it easily extendable. The stateless client-server architecture guarantees endless scalability. The focus of Geomajas is to provide a platform for server-side integration of geospatial data, allowing multiple users to control and manage the data from within their own browsers. In essence, Geomajas provides a set of powerful building blocks, from which the most complex GIS applications can easily be built. Key features include:

- Modular architecture
- Clearly defined API
- Integrated client-server architecture
- Built-in security
- Advanced geometry and attribute editing with validation
- Custom attribute definitions including object relations
- Advanced querying capabilities (searching, filters, style, ...)

See <http://www.geomajas.org/>.

3. License information

Copyright © 2009-2010 Geosparc nv.

Licensed under the GNU Affero General Public License. You may obtain a copy of the License at <http://www.gnu.org/licenses/>

This program is distributed in the hope that it will be useful, but *without any warranty*; without even the implied warranty of *merchantability* or *fitness for a particular purpose*. See the GNU Affero General Public License for more details.

The project also depends on various other open source projects which have their respective licenses.

From the Geomajas source (possibly specific module), the dependencies can be displayed using the "mvn dependency:tree" command.

For the dependencies of the Geomajas back-end, we only allow dependencies which are freely distributable for commercial purposes (this for example excludes GPL and AGPL licensed dependencies).

4. Author information

This framework and documentation was written by the Geomajas Developers. If you have questions, found a bug or have enhancements, please contact us through the user fora at <http://www.geomajas.org/>

List of contributors for this manual:

- Pieter De Graef
- Jan De Moerloose
- Joachim Van der Auwera
- Frank Wynants

Chapter 2. Starting a new GWT based Geomajas project

Geomajas uses the Apache Maven project management tool for its build and documentation process. Thanks to Maven, the easiest way to start using Geomajas is by creating a new project using the Maven archetype. This will create a simple working project that you can use as starting point.

1. Prerequisites / Command line

As the simple project is created using the Maven archetype, you will need to install Maven on your system, which can be downloaded from <http://maven.apache.org/>. We recommend using the latest stable version (2.2.1 at the time of writing). Installing Maven is quite simple: just unzip the distribution file in the directory of your choice and make some environment changes so you can access the executable. More information for your specific OS can be found at the bottom of <http://maven.apache.org/download.html>.

1.1. Creating the template project

At this point it is assumed that Maven has been successfully installed. Using Maven, you can now create a template project, called the Geomajas GWT Application Archetype.

1. *Step1*: Go to the folder you want to create this application in, and execute the following command:

Example 2.1. Create project using GWT Maven archetype

```
mvn archetype:generate -DarchetypeCatalog=http://files.geomajas.org/archetype
```

This will create the template project, using the latest stable release. If you want to use the latest snapshot, use the following command instead:

Example 2.2. Create project using GWT Maven archetype

```
mvn archetype:generate -DarchetypeCatalog=http://files.geomajas.org/archetype
```

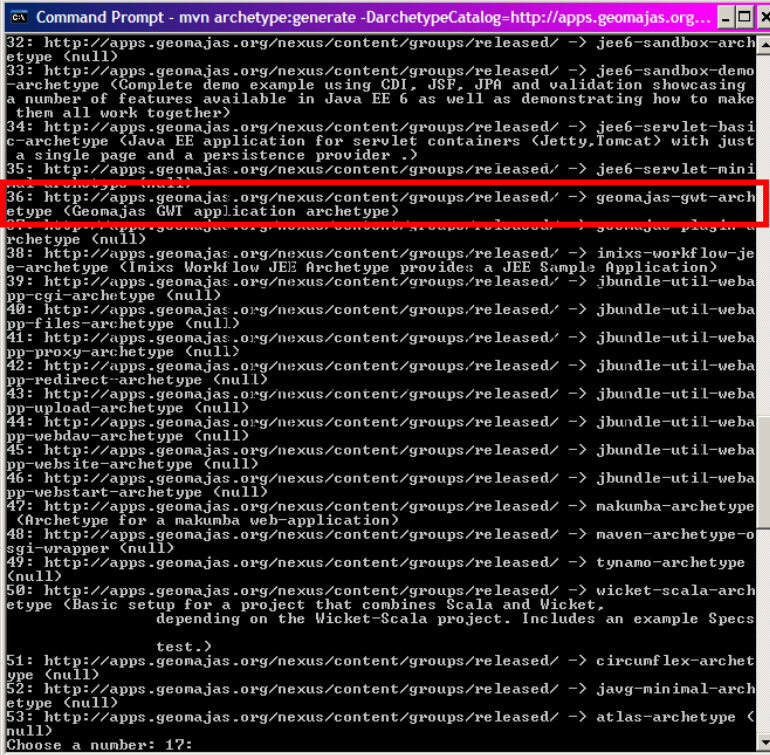
Maven will now prompt the user for input.

Note

If anything fails with these maven commands, you need to check the output to check the problem. One typical problem is the need to define a proxy to allow maven to access sites in the outside world. See the <http://maven.apache.org/guides/mini/guide-proxies.html> for details.

2. *Step2*: Maven will display the full list of available archetypes. Make sure you select the correct one: **geomajas-gwt-archetype**. In the image below, the correct number would be 36:

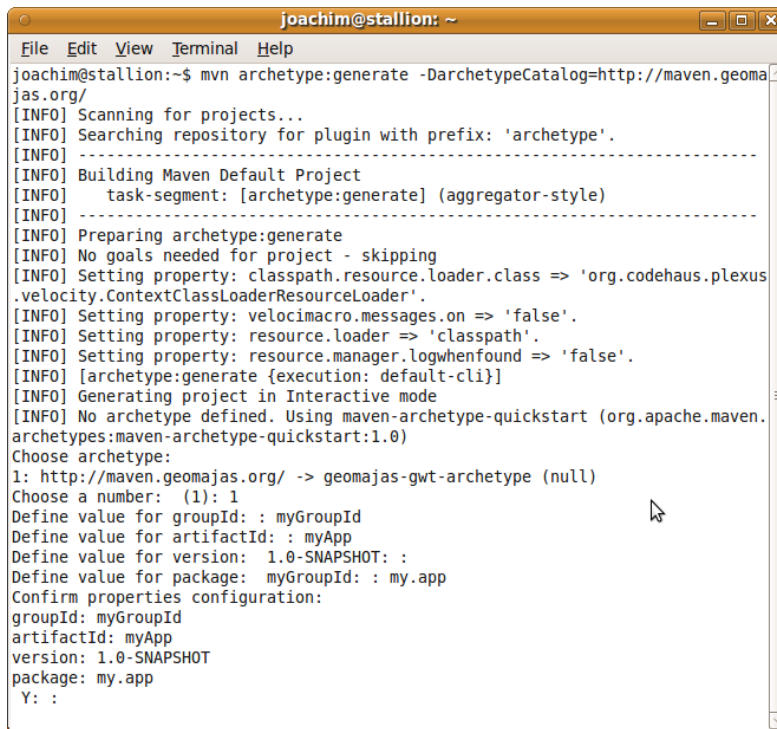
Figure 2.1. Choose the correct archetype



```
Command Prompt - mvn archetype:generate -DarchetypeCatalog=http://apps.geomajas.org...
32: http://apps.geomajas.org/nexus/content/groups/released/ -> jee6-sandbox-archetype (null)
33: http://apps.geomajas.org/nexus/content/groups/released/ -> jee6-sandbox-demo-archetype (Complete demo example using CDI, JSF, JPA and validation showcasing a number of features available in Java EE 6 as well as demonstrating how to make them all work together)
34: http://apps.geomajas.org/nexus/content/groups/released/ -> jee6-servlet-basic-archetype (Java EE application for servlet containers (Jetty, Tomcat) with just a single page and a persistence provider.)
35: http://apps.geomajas.org/nexus/content/groups/released/ -> jee6-servlet-mini-archetype (null)
36: http://apps.geomajas.org/nexus/content/groups/released/ -> geomajas-gwt-archetype (Geomajas GWT application archetype)
37: http://apps.geomajas.org/nexus/content/groups/released/ -> geomajas-plugin-archetype (null)
38: http://apps.geomajas.org/nexus/content/groups/released/ -> inixs-workflow-jee-archetype (Inixs Workflow JEE Archetype provides a JEE Sample Application)
39: http://apps.geomajas.org/nexus/content/groups/released/ -> jbundle-util-webapp-cgi-archetype (null)
40: http://apps.geomajas.org/nexus/content/groups/released/ -> jbundle-util-webapp-files-archetype (null)
41: http://apps.geomajas.org/nexus/content/groups/released/ -> jbundle-util-webapp-proxy-archetype (null)
42: http://apps.geomajas.org/nexus/content/groups/released/ -> jbundle-util-webapp-redirect-archetype (null)
43: http://apps.geomajas.org/nexus/content/groups/released/ -> jbundle-util-webapp-upload-archetype (null)
44: http://apps.geomajas.org/nexus/content/groups/released/ -> jbundle-util-webapp-wehdav-archetype (null)
45: http://apps.geomajas.org/nexus/content/groups/released/ -> jbundle-util-webapp-website-archetype (null)
46: http://apps.geomajas.org/nexus/content/groups/released/ -> jbundle-util-webapp-website-archetype (null)
47: http://apps.geomajas.org/nexus/content/groups/released/ -> makumba-archetype (Archetype for a makumba web-application)
48: http://apps.geomajas.org/nexus/content/groups/released/ -> naven-archetype-origi-wrapper (null)
49: http://apps.geomajas.org/nexus/content/groups/released/ -> tynamo-archetype (null)
50: http://apps.geomajas.org/nexus/content/groups/released/ -> wicket-scala-archetype (Basic setup for a project that combines Scala and Wicket, depending on the Wicket-Scala project. Includes an example Specs test.)
51: http://apps.geomajas.org/nexus/content/groups/released/ -> circumflex-archetype (null)
52: http://apps.geomajas.org/nexus/content/groups/released/ -> javg-minimal-archetype (null)
53: http://apps.geomajas.org/nexus/content/groups/released/ -> atlas-archetype (null)
Choose a number: 17:
```

3. *Step3*: Next maven asks for the **groupId**. Often the package name is used. (foo.bar)
4. *Step4*: Next maven asks for the **artifactId**. This represents the name for your application. (i.e. my-app)
5. *Step5*: Next maven asks for the first version for your application. 1.0-SNAPSHOT is a good first version.
6. *Step6*: Next maven asks for the base package wherein to place Java files. By default this is the same as the groupId. Just hit "enter" to continue.

Figure 2.2. Screenshot when building the Geomajas GWT archetype



```
joachim@stallion: ~  
File Edit View Terminal Help  
joachim@stallion:~$ mvn archetype:generate -DarchetypeCatalog=http://maven.geomajas.org/  
[INFO] Scanning for projects...  
[INFO] Searching repository for plugin with prefix: 'archetype'.  
[INFO] -----  
[INFO] Building Maven Default Project  
[INFO]   task-segment: [archetype:generate] (aggregator-style)  
[INFO] -----  
[INFO] Preparing archetype:generate  
[INFO] No goals needed for project - skipping  
[INFO] Setting property: classpath.resource.loader.class => 'org.codehaus.plexus.velocity.ContextClassLoaderResourceLoader'.  
[INFO] Setting property: velocimacro.messages.on => 'false'.  
[INFO] Setting property: resource.loader => 'classpath'.  
[INFO] Setting property: resource.manager.logwhenfound => 'false'.  
[INFO] [archetype:generate {execution: default-cli}]  
[INFO] Generating project in Interactive mode  
[INFO] No archetype defined. Using maven-archetype-quickstart (org.apache.maven.archetypes:maven-archetype-quickstart:1.0)  
Choose archetype:  
1: http://maven.geomajas.org/ -> geomajas-gwt-archetype (null)  
Choose a number: (1): 1  
Define value for groupId: : myGroupId  
Define value for artifactId: : myApp  
Define value for version: : 1.0-SNAPSHOT: :  
Define value for package: : myGroupId: : my.app  
Confirm properties configuration:  
groupId: myGroupId  
artifactId: myApp  
version: 1.0-SNAPSHOT  
package: my.app  
Y: :
```

Tip

You have now create a Geomajas template project! It is best to continue by testing if everything went well.

1.2. Testing the template project

From the project root, you can immediately compile, test (using jetty as servlet container), or run the application in development mode using the following commands.

1.2.1. Creating a full build

You now have several options to actually test your application. First of all, you could compile it, and create a build. This can be done using the following command:

Example 2.3. Creating a build from your project

```
mvn install
```

The "install" target will create a .war file for the project in the target directory. This web archive can be dropped into a Java application container such as Tomcat.

1.2.2. Running the compiled template application (full speed)

Secondly, if you don't have a Java application container ready or want a quick test, then you can use mvn to run the application for you, using the following commands:

Example 2.4. Create a build, then run it

```
mvn jetty:run
```

The "jetty:run" variant will immediately start a jetty server and start the application. This way, you can test your application at full speed (as when deployed). The application can be accessed at <http://localhost:8080/>.

1.2.3. Running the template application in development mode

Another option is to start up the application in GWT development mode, using the following command lines:

Example 2.5. Run the template application in development mode

```
mvn gwt:run
```

The "gwt:run" option allows you to start the application in GWT development mode. A console will appear which allows starting your application (from the browser). Amongst other things, this allows you to see the messages GWT generates and see the output of the "GWT.log" commands.

2. Eclipse

The combination of Eclipse, maven and GWT is not quite trivial, especially for complex multi-module projects like Geomajas. There are 2 approaches possible for integrating eclipse with maven:

- Eclipse plug-in for maven, avoiding the use of the maven command-line interface: m2eclipse (<http://m2eclipse.sonatype.org/>) is the most mature project here
- Maven plug-in to generate eclipse project configurations: maven-eclipse-plugin (<http://maven.apache.org/plugins/maven-eclipse-plugin/>)

It is clear that an IDE integrated solution like m2eclipse offers considerable advantages over manually generating Eclipse project configurations:

- direct import of maven projects
- support for maven properties and filtering
- In-place editing of poms
- full dependency support

For a functional Geomajas setup, the following Eclipse plug-ins should be installed on a fresh Galileo download (<http://www.eclipse.org/downloads/>):

- m2eclipse: update site <http://m2eclipse.sonatype.org/sites/m2e>
- m2eclipse extras (especially WTP extension): <http://m2eclipse.sonatype.org/sites/m2e-extras>
- checkstyle: update site <http://eclipse-cs.sf.net/update/>
- SVN team provider: update site <http://download.eclipse.org/releases/helios>, choose Collaboration ->Subversive SVN Team Provider (Incubation)
- Google's GWT Eclipse plug-in: <http://dl.google.com/eclipse/plugin/3.6> (Plugin and SDK)

2.1. Running/debugging with the Google Plug-in for Eclipse (embedded Jetty option)

There is a classpath issue with the Google Plug-in for Eclipse (GPE) that prevents us from using it in a reliable way when there are multiple versions of artifacts in the maven dependency tree: <http://code.google.com/p/google-web-toolkit/issues/detail?id=5033> [<http://code.google.com/p/google-web-toolkit/issues/detail?id=5033>]

On top of that, GPE forces the use of the built-in jetty launcher, which has problems with loading libraries from the maven repository. Recent development by Google points in the direction of better maven support, but as far as we know a stable solution which does not require explicit user interaction is not available. (see <http://googlewebtoolkit.blogspot.com/2010/08/how-to-use-google-plugin-for-eclipse.html>).

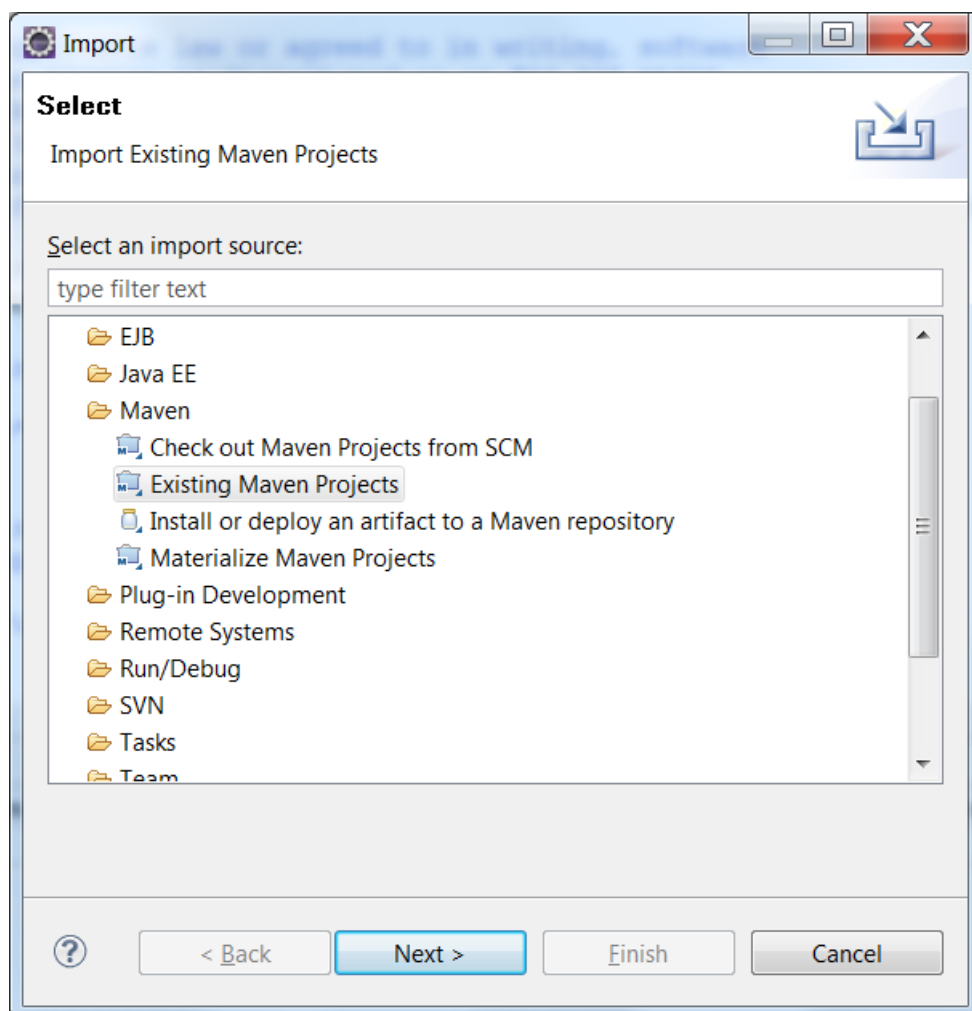
In view of these problems, the following workaround seems to be the most reliable way of using the GWT plugin for us:

- Run the GWT plug-in with the no-server option (this can be done straight from the project since GWT 2.1)
- Run an embedded Jetty server to replace the GWT server

The following series of steps have to be performed to achieve a working project.

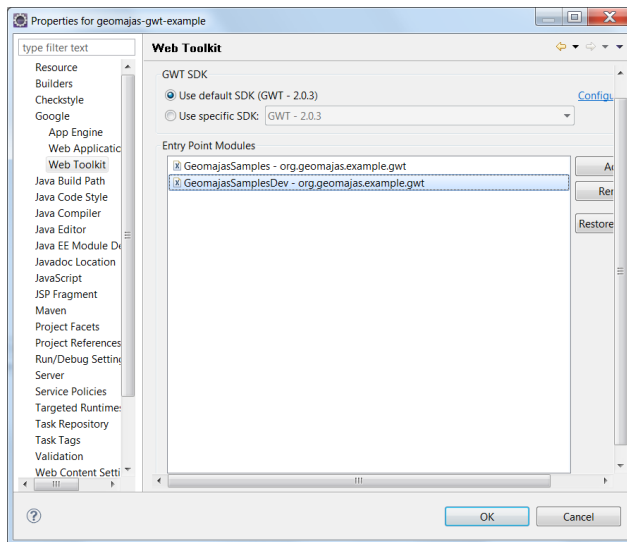
- Import the project as a maven project

Figure 2.3. Import project as Maven project



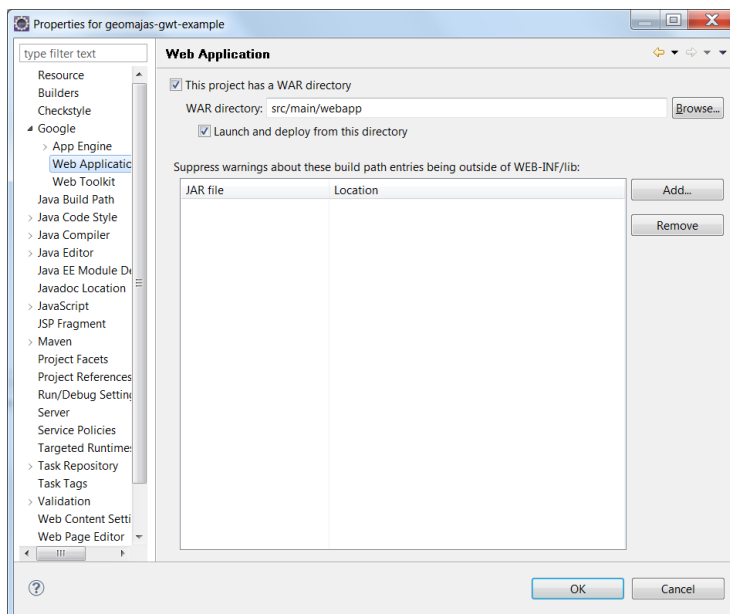
- After the project has been imported and the workspace has been built, you should now manually mark the project as a GWT project in the project properties dialog. Open the Google -> Web Toolkit section and mark the checkbox. If the eclipse GWT version differs from the project version, the "Use specific SDK" checkbox will be enabled:

Figure 2.4. Eclipse project properties dialog, Google Web Toolkit



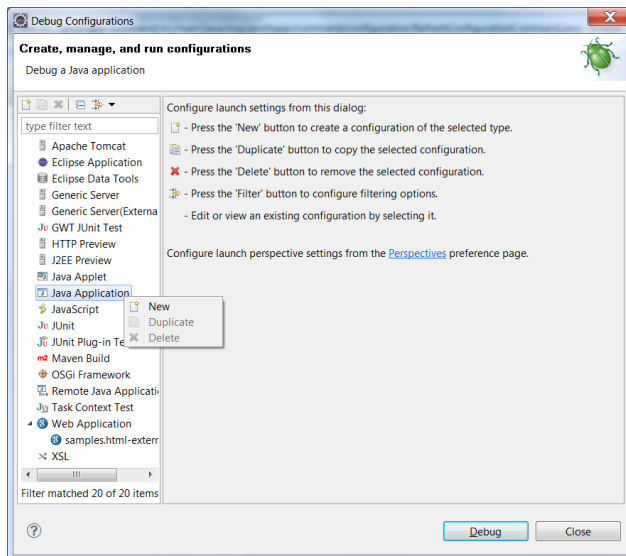
- In the Google -> Web Application section, the WAR directory should be changed to the default maven war sources directory (src/main/webapp)

Figure 2.5. Eclipse project properties dialog, Google Web Application



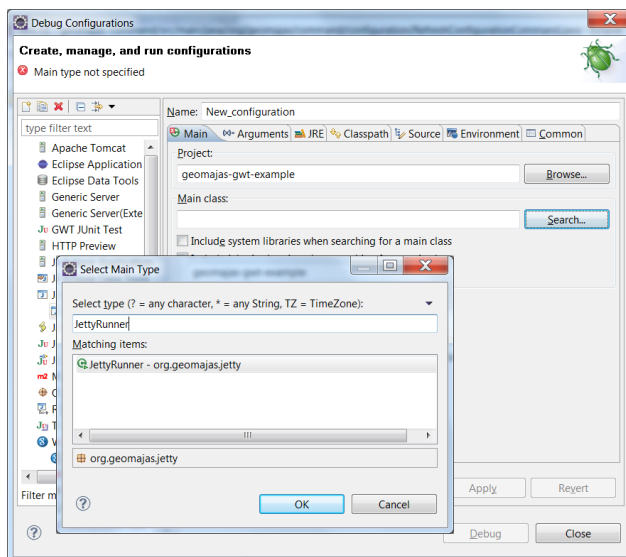
- Open the debug configurations dialog and create a new Java application:

Figure 2.6. Debug configurations dialog



- Search for JettyRunner as the main class. JettyRunner is a specially prepared main class that starts up Jetty with the correct parameters (port 8888 and src/main/webapp as webapp directory)

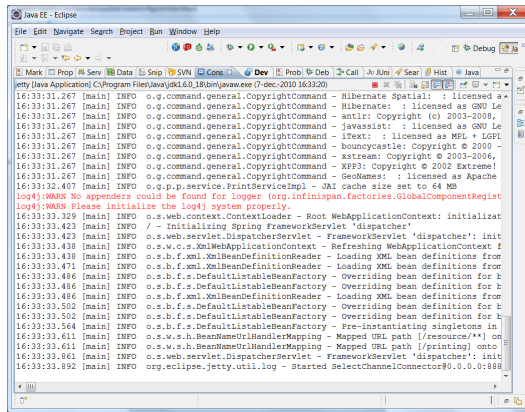
Figure 2.7. JettyRunner as main class



- Run the new Jetty server configuration (in debug mode):

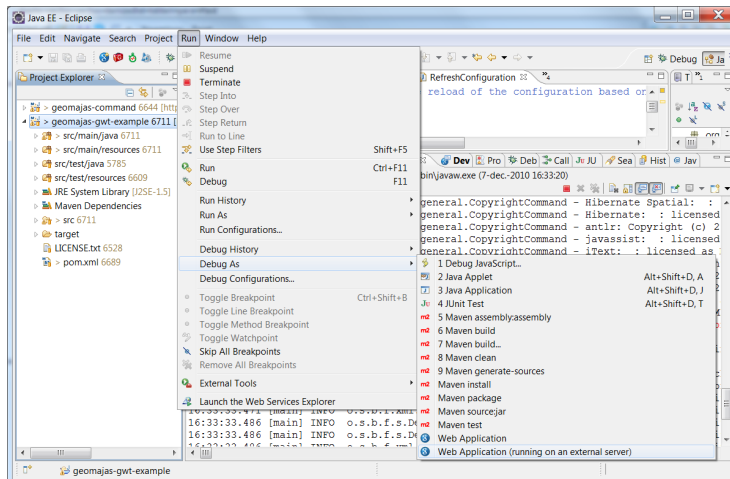
Starting a new GWT based Geomajas project

Figure 2.8. Running Jetty



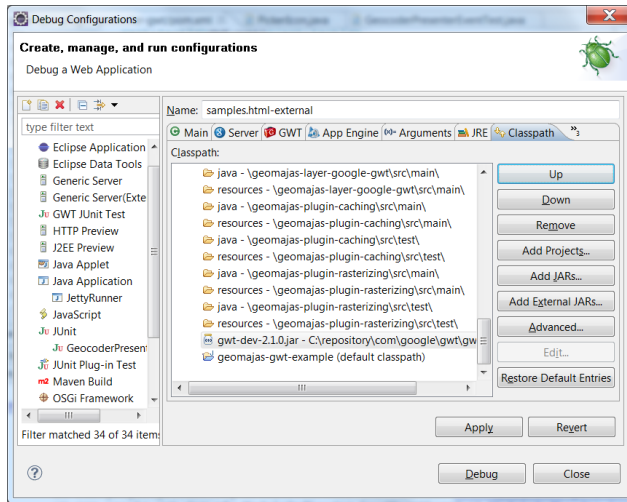
- Now, run the GWT plug-in in no-server mode by right-clicking on the project and selecting Debug As -> Web Application (running on an external server):

Figure 2.9. Running the GWT application



- Click ok on the dialog screen
- You can now add breakpoints and debug your application as if it was a normal Java application.
- If you have multiple versions of GWT in your project workspace, make sure that the gwt-dev jar is in front of the default classpath (reported GWT issue):

Figure 2.10. Classpath of GWT plug-in (no-server mode)



2.2. Running/debugging with the Google Plug-in for Eclipse

If you want to run your application directly with GPE, some extra actions are needed to avoid classpath problems. You will have to change your web.xml by adding a special context listener that allows Spring component scanning and circumvents a GeoTools problem with the builtin Jetty classloader:

```
<listener>  
<listener-class>org.geomajas.servlet.PrepareScanningContextListener</listener-class>  
</listener>
```

This solves most of the classpath problems but does not cure the problem of having multiple artifact versions in the classpath! This is usually only a problem when you have several Geomajas projects opened in your workspace. If this is the case, run the GWT plug-in with an embedded Jetty server as explained in the previous chapter and you should be fine. Note that this should be the first listener in your web.xml files, before the spring listeners.

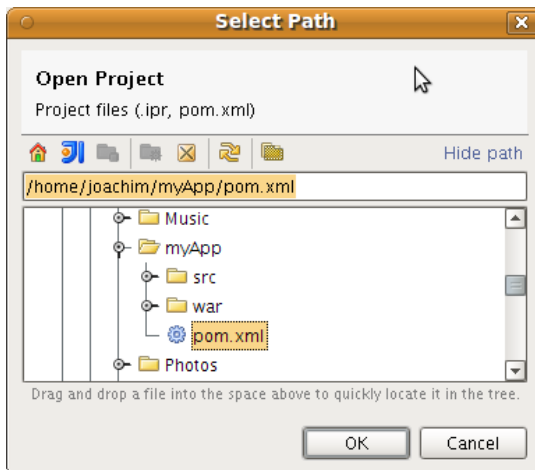
The following steps are needed to run GPE directly:

- Follow the configuration steps of the indirect mode, right up to the Jetty part
- Run the project as a GWT Web application by right-clicking on the project and selecting Run as -> Web Application.
- For debugging, debug the project as a GWT Web application by right-clicking on the project and selecting Debug as -> Web Application.

3. IntelliJ IDEA

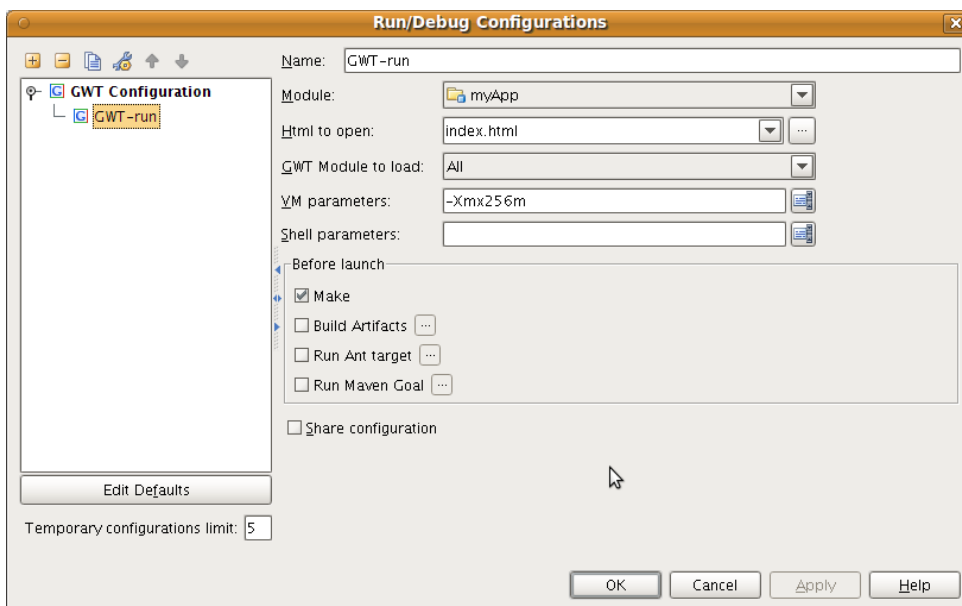
The setup in IntelliJ IDEA is quite straightforward and does not require running a separate Maven command. Just open the project from IDEA by selecting the pom in the root directory.

Figure 2.11. Open project using pom



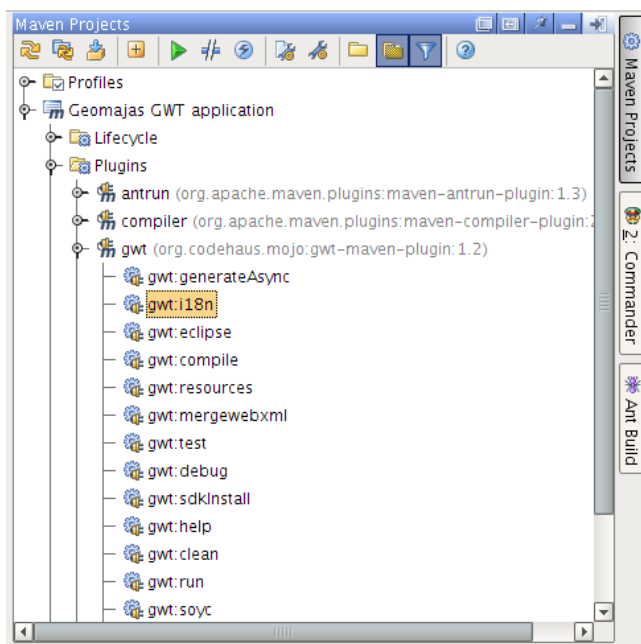
IDEA will recognize this as a GWT project and assign the correct facet but as always you will have to make your own run configuration (which is fortunately trivial). You will need version 9.0 or later for the GWT 2.0 support.

Figure 2.12. IDEA GWT run configuration



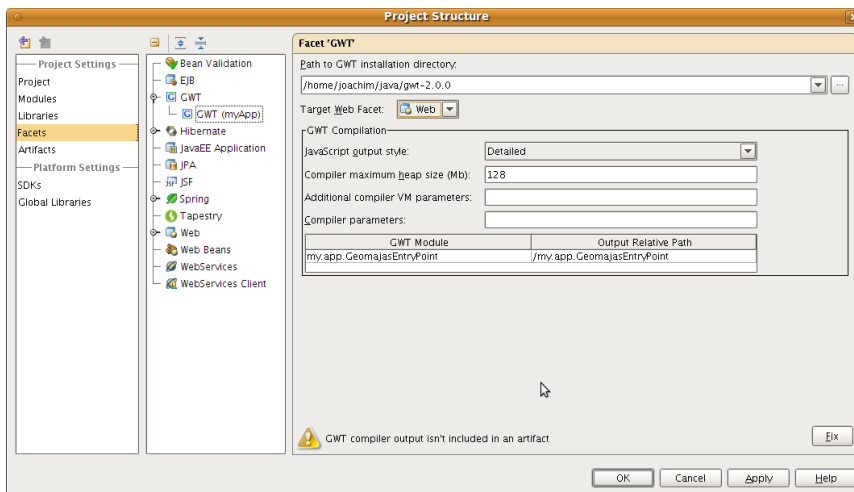
Before being able to use this configuration, you need to invoke the `gwt:i18n` Maven target to assure the files which are used for internationalisation are available (otherwise, you will get compilation errors). You can do this from the "Maven projects" tab.

Figure 2.13. run gwt:i18n target



Some additional settings have to be done in the "project structure" dialog. Apart from specifying the GWT installation directory, there is a specific project setting which has to be done manually, which is setting the target Web facet to "Web". The project structure for the simple GWT project should look as follows:

Figure 2.14. Project structure for simple GWT project



After this, you should be able to run the project. Any changes in the source code will be automatically detected, and debugging is possible.

Note

When creating GWT run configurations, it is recommended to increase the amount of memory given to the process. You can do this by entering a higher -Xmx value in the VM parameters field, for example "-Xmx768m".

4. NetBeans

You can both create the project from the archetype or open directly the Maven project in NetBeans. See <http://wiki.netbeans.org/MavenBestPractices> for more details.

5. Using a Hibernate layer

The template project can easily be modified to use a layer stored in PostGIS. You should do the following:

- In the pom.xml, uncomment block marked with "uncomment if you want to use Hibernate with postgis, for another db you will need similar dependencies".
- In the web.xml file (src/main/webapp/WEB-INF/web.xml) uncomment the block marked with "To use Roads layer stored in PostGIS through hibernate layer, uncomment the following".

To create the database, you can use the following command which will create the database (named "app") with some sample data (just a couple of roads):

```
psql -d app <src/main/webapp/WEB-INF/example/road.sql
```

6. How to continue

The archetype generates a dependency on the geomajas-dep project to manage the version of the Geomajas dependencies. This project exists for the sole purpose of keeping track of the latest released versions. It is quite likely that a new version of geomajas-dep has been released since the latest release of the archetype. Therefore, we recommend you update this dependency to the latest version. You can check the latest version using this URL: <http://repo.geomajas.org/nexus/index.html#nexus-search;quick~geomajas-dep>.

The most important configuration files in the project are the following:

- main configuration : `src/main/webapp/WEB-INF/applicationContext.xml`
- map configuration : `src/main/webapp/WEB-INF/mapMain.xml`
- overview map configuration: `src/main/webapp/WEB-INF/mapOverview.xml`
- countries layer configuration : `src/main/webapp/WEB-INF/clientLayerCountries.xml` and `src/main/webapp/WEB-INF/layerCountries.xml`
- OpenStreetMap layer configuration : `src/main/webapp/WEB-INF/clientLayerOsm.xml` and `src/main/webapp/WEB-INF/layerOsm.xml`
- GWT configuration file : `src/main/java/Application.gwt.xml`
- web.xml: `src/main/webapp/WEB-INF/WEB-INF/web.xml`

More details about the Geomajas configuration are found in the developer's guide [<http://files.geomajas.org/maven/trunk/geomajas/docbook-devuserguide/html/master.html#part-configuration>].

Reference which may be interesting to read:

- GWT project page: <http://code.google.com/webtoolkit/>.
- SmartGWT showcase: <http://www.smartclient.com/smartgwt/showcase/>.

- DZone's GWT refcardz: <http://refcardz.dzone.com/refcardz/gwt-style-configuration-and-js>.
- spring documentation: <http://www.springsource.org/documentation>.
- DZone's spring configuration refcardz: <http://refcardz.dzone.com/refcardz/spring-configuration>.
- maven project: <http://maven.apache.org/>.
- Maven by example book: <http://www.sonatype.com/books/mvnex-book/reference/public-book.html>.
- maven reference book: <http://www.sonatype.com/books/mvnref-book/reference/public-book.html>.
- DZone's maven 2 refcardz: <http://refcardz.dzone.com/refcardz/apache-maven-2>.

Chapter 3. Real world example: marine application

Warning

This chapter is out of date. It was originally written for an older, no longer supported version of Geomajas. It was mostly updated to comply with current conventions but is likely to be inaccurate or even wrong in some places.

1. Introduction

In this chapter a step-by-step guide is given of how to create a web-application based on Geomajas from scratch. The application is based on marine information and has the following features :

- Maps

The following maps will be used in the web-application

 - OpenStreetMaps as a background
 - Shape files
 - ECDIS map. The source of these maps are simple feature data files. These will be converted using OGR2OGR to a postGIS database.
 - AGIV maps. This is a WMS server (<http://gditestbed.agiv.be/blog/2010/01/default.aspx>)
- SOAP
 - Using SOAP we will request the current positions of vessels in Belgium and Holland. Each vessel will be shown on a the map with a marker. When such a marker is clicked extra information about the selected vessel is shown.
- Facility XML
 - The facility XML contains information about locks. When a lock is clicked (in one of the shape file layers) the corresponding XML is requested from the server and translated to HTML using XSL. After this the resulting HTML will be displayed to the user.
- User management
 - Two different users will be available. One user has complete access to everything. The second user only can view some layers.

2. Preliminary note

When using Eclipse and the GWT plug-in we removed the "generateAsync" feature from the pom.xml. Our code already contained manually generated async files, and this caused weird problems/conflicts.

Note that when you are in need of help you can always post questions on the Geomajas mailing list (check this page [<http://geomajas.org/gis-development>] for instructions on how to join the mailing list) or on the Geomajas forum [<http://geomajas.org/forum>].

3. Setting up a Geomajas based project

To create a new project based on Geomajas you must execute the following steps:

Example 3.1. Create project using GWT Maven archetype

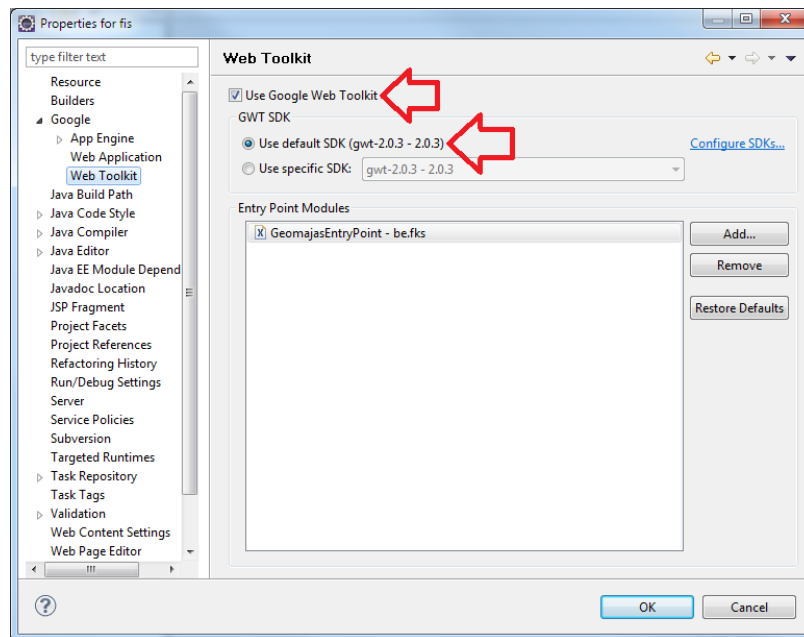
```
mvn archetype:generate -DarchetypeCatalog=http://files.geomajas.org/archetype-c
```

Once this is executed you can , among others, use the following commands :

- `mvn install`
- `mvn jetty:run`
- `mvn gwt:run`
- `mvn eclipse:eclipse`

If you are using eclipse you want to start by using the `mvn eclipse:eclipse` command. This will make your project eclipse compliant. Once this command is finished go to eclipse, right-click on the project, properties, google and check that the project is using GWT. Note that every time you do an `mvn eclipse:eclipse` you will have to retake the steps. Most times you have to switch off/switch on the GWT marker in eclipse.

Figure 3.1. Update your project as a GWT project in eclipse after each `mvn eclipse:eclipse`



Now you can already try and run the project. Either from within eclipse or from the command line using maven: `mvn gwt:run`. You will see that you now have a basic Geomajas project already showing an OpenStreetMaps layer and a vector layer based on shape files.

4. Showing your own shape files in Geomajas

In this section we are going to explain how to add a layer to your Geomajas showing some of your own shape files. The layer we will be creating has the name `VhaLayer`.


To show your own shape files in Geomajas follow these steps :

1. Place your shape files in `src/main/resources/be/fks/shapes` (note that you can configure the exact location of these).
2. Now modify `src/main/webapps/WEB-INF/applicationContext.xml` Note that you can use the existing road shape file layer as a reference to help you out.

- a. Add Layer

Figure 3.2. Add vhaLayer to the list of layers


```
<property name="layers">
  <list>
    <ref bean="osmLayer" />
    <ref bean="vhaLayer" />
    <ref bean="vlSluisLayer" />
    <ref bean="wmsAgivLayer"/>
    <ref bean="hibernateEcdisLayer"/>
  </list>
</property>
```



- b. Add treeNode

Figure 3.3. Add vhaLayer to the list of treeNode

```
<property name="treeNode">
  <bean class="org.geomajas.configuration.client.ClientLayerTreeNodeInfo">
    <property name="label" value="Layers" />
    <property name="layers">
      <list>
        <ref bean="osmLayer" />
        <ref bean="vhaLayer" />
        <ref bean="vlSluisLayer" />
        <ref bean="wmsAgivLayer"/>
        <ref bean="hibernateEcdisLayer"/>
      </list>
    </property>
    <property name="expanded" value="true" />
  </bean>
</property>
```



- c. Add org.geomajas.configuration.client.ClientVectorLayerInfo bean

Figure 3.4. Add ClientVectorLayerInfo bean

```
<bean class="org.geomajas.configuration.client.ClientVectorLayerInfo" id="
  <property name="serverLayerId" value="vha" />
  <property name="label" value="VHA Bevaarbaar" />
  <property name="visible" value="false" />
  <property name="viewScaleMin" value="0" />
  <property name="viewScaleMax" value="10" />
  <property name="namedStyleInfo" ref="vhaStyleInfo" />
  <property name="snappingRules">
    <list>
      <bean class="org.geomajas.configuration.SnappingRuleInfo">
        <property name="distance" value="10" />
        <property name="type" value="NEAREST_POINT" />
        <property name="layerId" value="vhaLayer" />
      </bean>
    </list>
  </property>
</bean>
```

3. Modify src/main/webapp/WEB-INF/web.xml (please remove the "be/fks/shapeinmem/" from the displayed text).

Figure 3.5. Add a reference to layerVha.xml in your web.xml

```
<!DOCTYPE web-app PUBLIC
"-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/dtd/web-app_2_3.dtd" >
<web-app>
  <display-name>Geomajas application</display-name>
  <context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>
      be/fks/shapeinmem/applicationContext.xml
      be/fks/shapeinmem/layerVha.xml
      be/fks/shapeinmem/layerVlSluis.xml
      be/fks/shapeinmem/layerOsm.xml
      be/fks/shapeinmem/layerWmsAgiv.xml
      be/fks/shapeinmem/layerEcdis.xml
      be/fks/shapeinmem/security.xml
```

4. Create src/main/webapp/WEB-INF/layerVha.xml. Note that you can use the existing layerRoads.xml as a reference to help you out.

Use a tool like uDig [<http://udig.refractor.net/>] to help you. With uDig you can view shape files, PostGIS data,... and you can view the available fields.

If you get really stuck you can download an example layerVha.xml right here [<files/realworldexample/layerVha.xml>].

5. Note that if your layer is a MULTIPOINT or a POINT layer you must define a symbol.

Figure 3.6. Create a symbol for MULTIPOINT or POINT layers

```
<property name="featureStyles">
  <list>
    <bean class="org.geomajas.configuration.FeatureStyleInfo">
      <property name="name" value="VlSluis" />
      <property name="fillColor" value="#00FF00" />
      <property name="fillOpacity" value="1" />
      <property name="strokeColor" value="#0000FF" />
      <property name="strokeOpacity" value="1" />
      <property name="strokeWidth" value="4" />
      <property name="symbol">
        <bean class="org.geomajas.configuration.SymbolInfo">
          <property name="rect">
            <bean class="org.geomajas.configuration.RectInfo">
              <property name="v" value="12" />
              <property name="h" value="12" />
            </bean>
          </property>
        </bean>
      </property>
    </bean>
  </list>
</property>
```

5. Show your own PostGIS data in Geomajas

In this section we are going to explain how to add a layer to your Geomajas showing some of your own shape files. The layer we will be creating has the name `EcdisLayer`.

Ecdis files are provided as simple feature files. The first step we are going to do is convert these files to PostGIS. After this we will configure Geomajas to display our created PostGIS data using hibernate.

1. Download Ecdis files (not available here).
2. Be sure you have a PostGIS database available.
3. Install the ogr2ogr tool (this is included in FWTools). You can find this tool using Google.
4. Use ogr2ogr in the following manner. Note that the csv files included in the zip must be visible to ogr2ogr.

Example 3.2. Use ogr2ogr to convert Ecdis files to PostGIS

```
ogr2ogr.exe -f "PostgreSQL" "PG:dbname=ecdis user=postgres password=postgres port=5432" Y:\1R5EK012.000 -append
```

5. Check your PostGIS database (remember that you can use a tool like uDig [<http://udig.refractor.net/>] for this).
6. Now modify `src/main/webapp/WEB-INF/applicationContext.xml`
 - a. Add `appDataSource`

Figure 3.7. Add appDataSource to applicationContext.xml

```
<!-- DataSource Property -->
<bean id="appDataSource" class="org.apache.commons.dbcp.BasicDataSource">
  <property name="driverClassName" value="org.postgresql.Driver" />
  <property name="url" value="jdbc:postgresql://127.0.0.1:5432/ecdis" />
  <property name="username" value="postgres" />
  <property name="password" value="postgres" />
</bean>
```

- b. Add appSessionFactory


Figure 3.8. Add appSessionFactory to applicationContext.xml

```
<!-- Hibernate SessionFactory -->
<bean id="appSessionFactory" class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
  <property name="dataSource" ref="appDataSource" />
  <property name="configLocation">
    <value>classpath:/be/fks/hibernate/cfg/hibernate.cfg.xml</value>
  </property>
  <property name="configurationClass">
    <value>org.hibernate.cfg.AnnotationConfiguration</value>
  </property>
</bean>
```

- c. Add layer

Figure 3.9. Add hibernateEcdisLayer to the list of layers


```
<property name="layers">
  <list>
    <ref bean="osmLayer" />
    <ref bean="vhaLayer" />
    <ref bean="vlSluisLayer" />
    <ref bean="wmsAgivLayer"/>
    <ref bean="hibernateEcdisLayer"/>
  </list>
</property>
```



- d. Add treeNode

Figure 3.10. Add hibernateEcdisLayer to the list of treeNode

```
<property name="treeNode">
  <bean class="org.geomajas.configuration.client.ClientLayerTreeNodeInfo"
    <property name="label" value="Layers" />
    <property name="layers">
      <list>
        <ref bean="osmLayer" />
        <ref bean="vhaLayer" />
        <ref bean="vlSluisLayer" />
        <ref bean="vmsAgivLayer"/>
        <ref bean="hibernateEcdisLayer"/>
      </list>
    </property>
    <property name="expanded" value="true" />
  </bean>
</property>
```



- e. Add org.geomajas.configuration.client.ClientVectorLayerInfo bean

Figure 3.11. Add ClientVectorLayerInfo bean

```
<bean class="org.geomajas.configuration.client.ClientVectorLayerInfo" id="ecdis"
  <property name="serverLayerId" value="ecdis" />
  <property name="label" value="Ecdis Hibernate" />
  <property name="visible" value="false" />
  <property name="viewScaleMin" value="0" />
  <property name="viewScaleMax" value="100000" />
  <property name="namedStyleInfo" ref="ecdisStyleInfo" />
</bean>
```

7. Modify src/main/webapp/WEB-INF/web.xml (please remove the "be/fks/shapeinmem/" from the displayed text)

Figure 3.12. Enable PostgisDialect

```
<context-param>
  <param-name>preloadClasses</param-name>
  <param-value>
    org.geomajas.command.general.LogCommand
    org.geomajas.spring.GeoMajasBeanNameGenerator
    org.hibernate.validator.engine.ValidatorImpl
    org.geotools.data.shapefile.ShapefileDataStoreFactory
    org.opengis.filter.FilterFactory
    org.geomajas.layer.hibernate.filter.ExtendedFilterFactory
    org.springframework.transaction.config.TxNameSpaceHandler
    org.hibernate.spatial.postgis.PostgisDialect
    org.geomajas.plugin.staticsecurity.security.AuthenticationTokenService
    org.geotools.referencing.crs.EPSGCRSAuthorityFactory
    org.geomajas.gwt.server.GeoMajasServiceImpl
  </param-value>
</context-param>
```

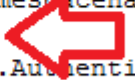


Figure 3.13. Add a reference to layerEcdis.xml in your web.xml

```
<web-app>
  <display-name>Geomajas application</display-name>
  <context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>
      be/fks/shapeinmem/applicationContext.xml
      be/fks/shapeinmem/layerVha.xml
      be/fks/shapeinmem/layerVlSluis.xml
      be/fks/shapeinmem/layerOsm.xml
      be/fks/shapeinmem/layerWmsAgiv.xml
      be/fks/shapeinmem/layerEcdis.xml
      be/fks/shapeinmem/security.xml
    </param-value>
  </context-param>
</web-app>
```

8. Create `src/main/webapp/WEB-INF/layerEcdis.xml`. Base yourself on an example layer file using Hibernate.

Use a tool like uDig [<http://udig.refractions.net/>] to help you. With uDig you can view shape files, PostGIS data,... and you can view all available fields.

If you get really stuck you can download an example `layerEcdis.xml` right here [<files/realworldexample/layerEcdis.xml>].

6. Show your own WMS map using Geomajas

Note that the WMS used in this example is a password protected WMS. Due to privacy reasons the login details are excluded from this example. Please use your own or a public WMS server to test out this part of the guide.

1. Modify `src/main/webapp/WEB-INF/applicationContext.xml`
 - a. Add `wmsAgivLayer` just as you did in the previous 2 examples.
 - b. Add `treeNode` just as you did in the previous 2 examples.
 - c. Add `org.geomajas.configuration.client.ClientRasterLayerInfo` bean

Figure 3.14. Add ClientVectorLayerInfo bean

```
<bean class="org.geomajas.configuration.client.ClientRasterLayerInfo" id="wmsAgivLayer">
  <property name="serverLayerId" value="agiv" />
  <property name="label" value="Agiv WMS" />
  <property name="visible" value="false" />
  <property name="viewScaleMin" value="0" />
  <property name="viewScaleMax" value="100000" />
  <property name="style" value="1" />
</bean>
```

2. Modify `src/main/webapp/WEB-INF/web.xml`.

Add a reference to `layerWmsAgiv.xml` in your `web.xml` just like you did in the previous 2 examples.

3. Create `src/main/resources/be/fks/shapeinmem/layerWmsAgiv.xml`

- a. `baseWmsUrl` : The link to the capabilities file but without all the extra parameters
- b. `dataSourceName` : This must contain a name of the layer (names of the layers can be found in the capabilities file of the WMS server)
- c. `org.geomajas.geometry.Bbox` : This can be derived from the capabilities file of the WMS server
- d. Now you can also add extra parameters. In our example we set `transparent` to `true` and do some other things. Note that the WMS must understand the parameters you try to use (possible parameters can be derived from the capabilities file).

You can download an example `layerWmsAgiv.xml` file right here [[realworldexample/layerWmsAgiv.xml](#)]. Note that as said before the login details are excluded from this example due to privacy reasons. This renders this example file useless. Please use your own, or a public WMS server to test out this part of the guide.

7. Show locations of vessels using SOAP

Geomajas doesn't have any native SOAP support implemented. So what you need to is create a normal GWT servlet which does the SOAP handling and sends information back to the client about the locations of the vessels. Finally at the Geomajas client side you can display these vessels.

The following steps should be taken :

1. Create a servlet requesting the needed information using SOAP. Note that if you are using any special external jars you need to add those to the class path in eclipse, and to the maven class path.
2. At the client side display the location of the vessels using an image.


First we load the positions of the vessels and put all the requested information inside a `ListVesselPos`. Initially we give the images a size of 0,0. This is because in Geomajas images resize when the user zooms. In the next step we will create size the images depending on the zoom level

Figure 3.15. Show vessels using an Image marker.png

```
//SOAP positions
service.getPositions(new AsyncCallback<List<VesselPosition>>()
{
    public void onFailure(Throwable caught)
    {
        GWT.log(null, caught);
    }

    public void onSuccess(List<VesselPosition> vesselPositions)
    {
        if(vesselPositions != null)
        {
            GeomajasEntryPoint.this.vesselPositions = vesselPositions;

            imgVesselPos.clear();
            GWT.log(Geomajas.getIsomorphicDir() + "marker.png");
            for(int i=0;i<vesselPositions.size();i++)
            {
                float lat = vesselPositions.get(i).getLat();
                float lng = vesselPositions.get(i).getLng();

                Image img = new Image("schip_" + i);
                img.setHref(Geomajas.getIsomorphicDir() + "marker.png");
                img.setBounds(new Bbox((double)lng, (double)lat, 0, 0));

                imgVesselPos.add(img);

                map.registerWorldPaintable(img);
            }
        }
    }
});
```

3. In `OnMapModelChange()` (called every time a user zooms) we resize the images to the correct size depending on the current zoom level.

Figure 3.16. Resize the images depending on the zoom level

```
//Zoom listener
//recalculate the size of the icons
map.getMapModel().getMapView().addMapViewChangedHandler(new MapViewChangedHa
{
    public void onMapViewChanged(MapViewChangedEvent event)
    {
        markerSize = 30 / event.getScale();

        for(int i=0;i<imgVesselPos.size();i++)
        {
            Bbox bbox = ((Bbox)imgVesselPos.get(i).getOriginalLocation());
            double lng = bbox.getX() + bbox.getWidth() / 2;
            double lat = bbox.getY();
            map.unregisterWorldPaintable(imgVesselPos.get(i));
            imgVesselPos.get(i).setBounds(new Bbox(lng - markerSize / 2, lat, m
            map.registerWorldPaintable(imgVesselPos.get(i));
        }
    }
});
```

4. Finally we create a controller. This controller shows some extra information to the user when the user clicks on a vessel.

In reality the controller is on the map (because you can't add a controller to an image). When clicked a check is performed to see if a vessel is at the location of the click.

Figure 3.17. Controller that shows extra information about a clicked vessel

```
//custom controller
final GraphicsController possrvController = new AbstractGraphicsController(m
{

    public void onMouseUp(MouseUpEvent event)
    {
        GWT.log("vesselPositions.size = " + vesselPositions.size());

        double clickX = getWorldPosition(event).getX();
        double clickY = getWorldPosition(event).getY();

        for(int i=0;i<vesselPositions.size();i++)
        {
            double vesselX = vesselPositions.get(i).getLng();
            double vesselY = vesselPositions.get(i).getLat();

            vesselX += markerSize / 2; //pak het midden
            vesselY += markerSize / 2; //pak het midden

            double difX = Math.abs(clickX - vesselX);
            double difY = Math.abs(clickY - vesselY);

            if((difX < (markerSize / 2)) && (difY < (markerSize / 2)))
            {
                SC.say("U heeft een schip aangeklikt : <UL><LI><b>Naam:</b> " +
                    + "</LI><LI><b>VoyageNo:</b> " + vesselPositions.get(i).ge
                break;
            }
        }
    }
};

// Add custom controller to toolbar

toolbar.addModalButton(new ToolbarModalAction("[ISOMORPHIC]/marker.png", "Sc
{
    @Override
    public void onSelect(ClickEvent event)
    {
        map.setController(possrvController);
    }

    @Override
    public void onDeselect(ClickEvent event)
    {
        map.setController(null);
    }
});
```

8. Show information about a lock using FacilityXML

FacilityXML documents are XML documents containing information about locks in Belgium. The goal was that when the user clicked on a lock in a certain layer the correct FacilityXML document was requested from the server and shown to the user in a nicely formatted way. The formatting is done using an XSL transformation.

Since the requesting of the XML and the XSL transformations isn't part of Geomajas this will not be discussed here. What will be show however is how you can add a listener to a layer and use data from the selected feature.

Figure 3.18. Controller that shows extra information about a clicked vessel

```
if(map.getMapModel().getVectorLayer("vlSluisLayer") != null)
{
    map.getMapModel().getVectorLayer("vlSluisLayer").addFeatureSelectionHandler
    {

        public void onFeatureSelected(FeatureSelectedEvent event)
        {
            String locode = "" + event.getFeature().getAttributeValue("LOCODE");
            String id = locode.substring(5);
            service.transformFacilityXML(id, new AsyncCallback<String>()
            {

                public void onFailure(Throwable caught)
                {
                    GWT.log(null, caught);
                    SC.warn("Server error: " + caught.getMessage());
                }

                public void onSuccess(String facilityHtml)
                {
                    if(facilityHtml == null)
                    {
                        SC.warn("Geen Facility-XML gevonden voor de gekozen sluis.")
                    }
                    else
                    {
                        final Window winModal = new Window();
                        HTMLPane htmlPane = new HTMLPane();
                        htmlPane.setContents(facilityHtml);
                        winModal.addItem(htmlPane);
                        winModal.setWidth("860px");
                        winModal.setHeight("80%");
                        winModal.setTitle("Facility XML");
                        winModal.setShowMinimizeButton(false);
                        winModal.setIsModal(true);
                        winModal.setShowModalMask(true);
                        winModal.centerInPage();
                        winModal.show();
                    }
                }
            });
        }

        public void onFeatureDeselected(FeatureDeselectedEvent event)
        {
        }
    });
}
```

9. User management

1. Create `src/main/resources/be/fks/shapeinmem/security.xml` (download an example right here [files/realworldexample/security.xml]). This is the file inside which you configure the users and what they are allowed to access.
2. Add a reference to `security.xml` to the `web.xml` just like you would do for a normal layer.
3. Add dependencies in `pom.xml`

Figure 3.19. Security pom.xml

```
<dependency>
  <groupId>org.geomajas.plugin</groupId>
  <artifactId>geomajas-plugin-staticsecurity</artifactId>
  <version>${geomajas-staticsecurity-version}</version>
</dependency>
<dependency>
  <groupId>org.geomajas.plugin</groupId>
  <artifactId>geomajas-plugin-staticsecurity</artifactId>
  <version>${geomajas-staticsecurity-version}</version>
  <classifier>sources</classifier>
</dependency>
<dependency>
  <groupId>org.geomajas.plugin</groupId>
  <artifactId>geomajas-plugin-staticsecurity-gwt</artifactId>
  <version>${geomajas-staticsecurity-version}</version>
</dependency>
<dependency>
  <groupId>org.geomajas.plugin</groupId>
  <artifactId>geomajas-plugin-staticsecurity-gwt</artifactId>
  <version>${geomajas-staticsecurity-version}</version>
  <classifier>sources</classifier>
</dependency>
```

4. Inherit security in the `gwt.xml` of the project.

Figure 3.20. Security inherit

```
<inherits name="org.geomajas.plugin.staticsecurity.StaticSecurity"/>
```

5. To assure that a security token is obtained when needed, you should register a token request handler in the `GwtCommandDispatcher`. A good candidate is using the `StaticSecurityTokenRequestHandler`.