

# **Geomajas end user guide**

**Geomajas Developers and Geosparc**

---

# **Geomajas end user guide**

by Geomajas Developers and Geosparc

v1.7.2

Copyright © 2010 Geosparc nv

---

---

---

---

## Table of Contents

1. Introduction .....	1
2. Configuration .....	2
Dependencies .....	2
Hibernate layer configuration .....	2
Transaction configuration .....	5
3. How-to .....	7

---

## List of Examples

2.1. Hibernate layer dependency .....	2
2.2. Hibernate entity .....	3
2.3. Hibernate roads layer definition .....	3
2.4. Hibernate roads layer info .....	4
2.5. Hibernate roads layer feature info .....	5
2.6. Hibernate transaction configuration .....	6

---

# Chapter 1. Introduction

---

# Chapter 2. Configuration

## Dependencies

The Hibernate layer is based on the popular Hibernate O/R mapping framework. It uses a special spatial extension of Hibernate, unsurprisingly called Hibernate Spatial. The Hibernate Spatial project has its project website at <http://www.hibernate.org> [<http://www.hibernate.org>]. The spatial extensions or dialects (in Hibernate language) allow the definition of spatial types and the execution of spatial queries in a database independent way.

You need to include the following dependencies to make this work. This needs to include the hibernate spatial provider, in this example, PostGis.

### Example 2.1. Hibernate layer dependency

```
<dependency>
  <groupId>org.geomajas.plugin</groupId>
  <artifactId>geomajas-layer-hibernate</artifactId>
  <version>${geomajas-layer-hibernate-version}</version>
</dependency>
<dependency>
  <groupId>org.hibernate.spatial</groupId>
  <artifactId>hibernate-spatial-postgis</artifactId>
  <version>1.0</version>
</dependency>
<dependency>
  <groupId>org.postgis</groupId>
  <artifactId>postgis-jdbc</artifactId>
  <version>1.1.6</version>
</dependency>
<dependency>
  <groupId>postgresql</groupId>
  <artifactId>postgresql</artifactId>
  <version>8.1-407.jdbc3</version>
</dependency>
```

## Hibernate layer configuration

A Hibernate layer cannot be defined by configuration only. As in every O/R model, there has to be a mapping between Java classes and tables in the database. In the most simple case there is a one-to-one mapping between a single class and a single spatial table.

The following listing shows the (partial) definition of a Hibernate annotated Java class `Road.java` that is mapped to a table `roads` in the database.

### Example 2.2. Hibernate entity

```

@Entity
@Table(name = "roads")
public class Road {

    @Id
    @GeneratedValue(strategy = javax.persistence.GenerationType.IDENTITY)
    @Column(name = "gid")
    private Long gid;

    private Long id;

    private String type;

    private Float length;

    private Long use;

    private Short wid;

    private Date date;

    private String url;

    @Type(type = "org.hibernate.spatial.GeometryUserType")
    @Column(name = "the_geom")
    private Geometry geometry;

```

The field annotations describe the relation between the fields of the class and the columns in the table. A special annotation `@Type` with type `org.hibernate.spatial.GeometryUserType` is used to map the geometry field to the `the_geom` spatial column.

Once the Java class mapping is finished, the actual layer configuration can be made. An example configuration that matches the `Road.java` class is shown below:

### Example 2.3. Hibernate roads layer definition

```

<bean name="roads" class="org.geomajas.layer.hibernate.HibernateLayer">
  <property name="layerInfo" ref="roadsInfo" />
  <property name="featureModel">
    <bean class="org.geomajas.layer.hibernate.HibernateFeatureModel">
      <property name="sessionFactory" ref="appSessionFactory" />
    </bean>
  </property>
  <property name="sessionFactory" ref="appSessionFactory" />
  <property name="dateFormat" ref="appDateFormat" />
</bean>

```

The first property `layerInfo` is the reference to the `VectorLayerInfo` object. While it can be defined inline, it has been defined as an outer bean for clarity here.

The `featureModel` property refers to the internal feature accessor face of the layer. This property will probably be removed as it has no additional configuration parameters for the moment.

The `sessionFactory` property refers to the Hibernate session factory. This is the same factory that has to be defined by the transaction configuration.

The `dateFormat` property determines how the layer will convert date values to strings and vice versa.

As already mentioned, the bulk part of the layer's metadata is defined through the `VectorLayerInfo` object. An example definition of this object is given below:

#### Example 2.4. Hibernate roads layer info

```
<bean name="roadsInfo" class="org.geomajas.configuration.VectorLayerInfo">
  <property name="layerType" value="MULTILINESTRING" />
  <property name="crs" value="EPSG:900913" />
  <property name="maxExtent">
    <bean class="org.geomajas.geometry.Bbox">
      <property name="x" value="505500" />
      <property name="y" value="6588000" />
      <property name="width" value="2000" />
      <property name="height" value="2000" />
    </bean>
  </property>
  <property name="featureInfo" ref="roadsFeatureInfo" />
  <property name="namedStyleInfos">
    <list>
      <ref bean="roadsStyleInfo" />
    </list>
  </property>
</bean>
```

The feature metadata can be found in the `FeatureInfo` object. This object contains the complete feature type description (id, attributes and geometry) as well as the validation rules for the attributes. An example definition of this object is given below:

**Example 2.5. Hibernate roads layer feature info**

```
<bean class="org.geomajas.configuration.FeatureInfo" name="roadsFeatureInfo">
  <property name="dataSourceName" value="mypackage.server.Road" />
  <property name="identifier">
    <bean class="org.geomajas.configuration.PrimitiveAttributeInfo">
      <property name="label" value="gid" />
      <property name="name" value="gid" />
      <property name="type" value="LONG" />
    </bean>
  </property>
  <property name="geometryType">
    <bean class="org.geomajas.configuration.GeometryAttributeInfo">
      <property name="name" value="geometry" />
      <property name="editable" value="true" />
    </bean>
  </property>
  <property name="attributes">
    <list>
      <bean class="org.geomajas.configuration.PrimitiveAttributeInfo">
        <property name="label" value="Id" />
        <property name="name" value="id" />
        <property name="type" value="LONG" />
      </bean>
      <bean class="org.geomajas.configuration.PrimitiveAttributeInfo">
        <property name="label" value="Length" />
        <property name="name" value="length" />
        <property name="editable" value="true" />
        <property name="identifying" value="true" />
        <property name="type" value="FLOAT" />
      </bean>
    </list>
  </property>
</bean>
```

## Transaction configuration

Transaction configuration for Hibernate layers:

**Example 2.6. Hibernate transaction configuration**

```
<!-- 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/simple" />
  <property name="username" value="simple" />
  <property name="password" value="simple" />
</bean>

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

<!-- enable the configuration of transactional behavior based on annotations -
<tx:annotation-driven transaction-manager="transactionManager" />

<bean id="transactionManager" class="org.springframework.orm.hibernate3.Hibern
  <property name="sessionFactory" ref="appSessionFactory" />
</bean>
```

Starting from the top, the following are defined:

- The data source: this specifies the connection pool type and the connection properties of the database (PostGis in this case)
- The session factory: this is Hibernate's primary singleton and used by the Hibernate layer to access the session/connection
- A tag to enable annotation-based transactional behavior, internally used by Geomajas to decide which commands need transaction support
- The platform transaction manager for Hibernate

---

# Chapter 3. How-to