EJB3 Spatial Tutorial

Spatial EJB3 is a quick investigation to see if it is possible to integrate the Java 5 annotation approach, Illustration 1,to mark a property of an object as spatial and to delegate to the EJB3 persistence model to store and retrieve this data.

```
@Type(type = "org.postgis.hibernate.GeometryType")
@Column(name="location", columnDefinition="Geometry")
public Geometry getLocation() {
    return location;
}
```

Illustration 1: Geometry Annotation

The project utilises JBoss and PostGIS, future iterations will look to remove the dependency on JBoss and Hibernate to incorporate other Application Servers.

The tutorial will be a step by step guide to getting started with EJB3 and Spatial Annotations, it assumes that the user has little knowledge of Java 5 annotations. Please contribute back to this document with your comments.

The document will walk through the steps to allow users to post co-ordinates specify their positions to a JBoss server and for this data to be stored and retrieved from a PostGIS database. Interestingly the tutorial will use Java 5 annotations to mark the Java POJOs as web services, the security required, and Python will be the client application.

If you have downloaded the associated source code, then running ant deploy will build and deploy the application and configuration files to JBoss. (Note: edit paths to match your system).

This document assumes you have already installed PostGIS and PostgreSQL 8.1.x!

Installing JBoss

'JBoss Application Server is the #1 most widely used Java application server on the market. Hundreds of professional open source developers have contributed to the JBoss Application Server over the years and community contributors are not only welcome but encouraged. In fact all JBoss employed contributors to the JBoss Application Server were hired from the community and each of them contributed to an open source project in one way or another.'

Key benefits of Enterprise Java to the geo-spatial community include;

- Geo-spatial feature persistence using Java Annotations
- Clustering of applications to provide high availability services
- Security, new and integrated using Java Authentication and Authorisation Service (JAAS)
- Transactional Support
- EJB3 is simple!

Install the JBoss Application Server using the web start installer available at http://labs.jboss.com/portal/jbossas/download this tutorial was written using the 4.0.4 version of Jboss. Select the EJB3 option as in Illustration 2: JEMS Installer.

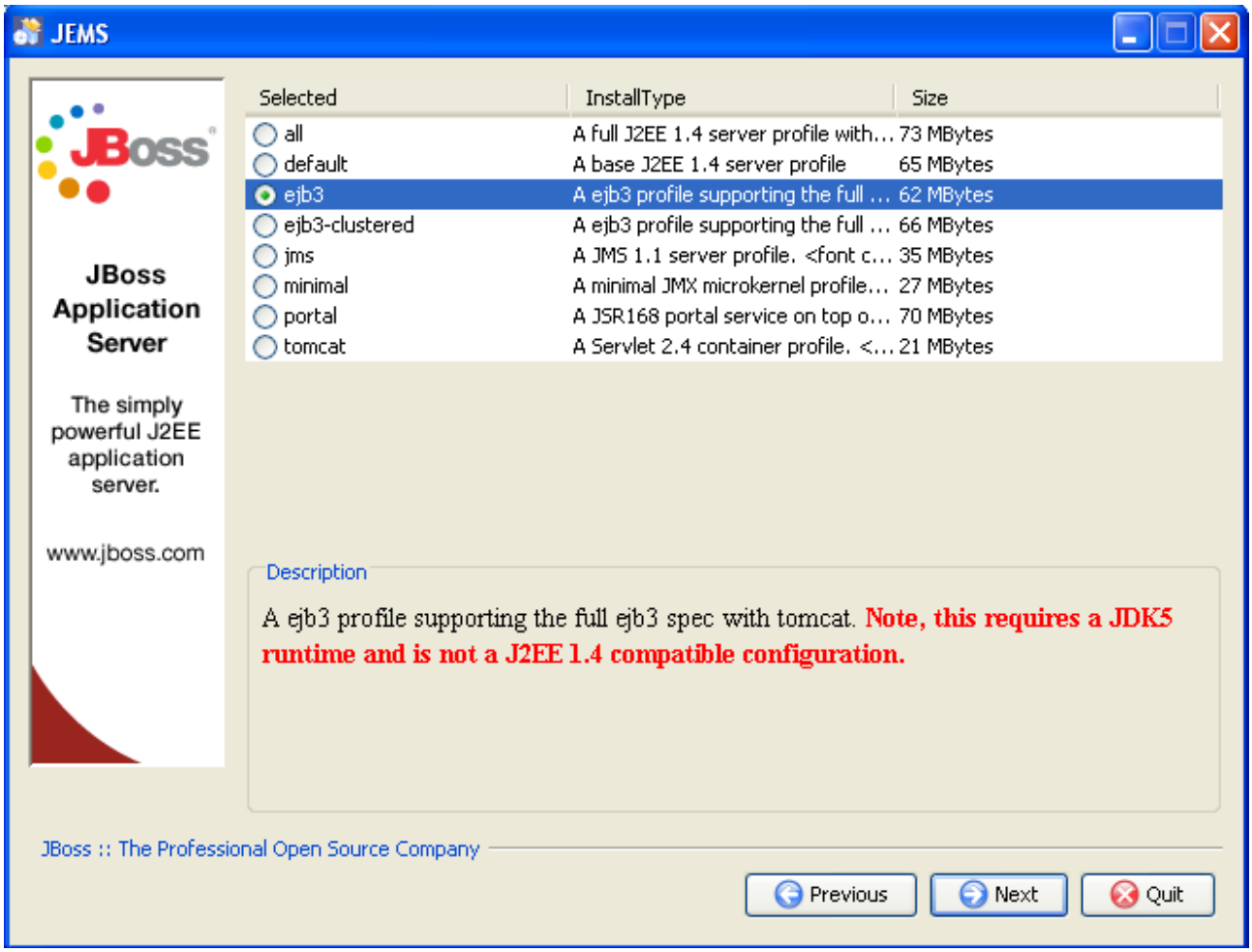

Illustration 2: JEMS Installer

Configuration Steps;

- Copy the posgresql and postgis jdbc drivers from postgres_home\jdbc to jboss_home\server\default\lib
- Define an enterprise data source in jboss_home\server\default\deploy by copying the following XML into geodata-ds,xml (naming convention mydatasourcename-ds.xml)

```
<?xml version="1.0" encoding="UTF-8"?>

<datasources>
  <local-tx-datasource>
  <jndi-name>GeoDataDS</jndi-name>
  <connection-url>jdbc:postgresql://127.0.0.1:5432/geotest</connection-url>
 <driver-class>org.postgis.DriverWrapper</driver-class>
  <user-name>geo</user-name>
 <password>geo</password>
 <metadata>
   <type-mapping>PostgreSQL 8.1</type-mapping>
 </metadata>
 </local-tx-datasource>
</datasources>
```

Configure PostGIS

This section assumes the user is familiar with setting up a PostGIS database, if not there is a useful tutorial here (http://www.bostongis.com/?content_name=postgis_tut01)

- Create a Postgis database called geotest (or rename the database in the datasource above)

```
CREATE TABLE people
(
  id serial NOT NULL,
  name text,
  surname text,
  "location" geometry,
  ingested TIMESTAMP,
  CONSTRAINT people_pkey PRIMARY KEY (id),
  CONSTRAINT enforce_dims_location CHECK (ndims("location") = 2),
  CONSTRAINT enforce_srid_location CHECK (srid("location") = -1)
)
WITHOUT OIDS;
ALTER TABLE people OWNER TO geo;

CREATE INDEX people_location_index ON people USING gist ("location");
```

Java Application

The tutorial code will use a JMS queue to do the actual data ingestion into PostGIS, however with the amount of data we are loading it doesn't justify this approach; Queues however are scalable and will therefore be demonstrated.

JMS queue

Create an XML file called ingest-service.xml in the JBOSS_HOME\server\default\deploy

```
<?xml version="1.0" encoding="UTF-8"?>
<server>
 <mbean code="org.jboss.mq.server.jmx.Queue"
  name="jboss.mq.destination:service=Queue,name=ingestQueue">
  <depends optional-attribute-name="DestinationManager">jboss.mq:service=DestinationManager</depends>
 </mbean>
</server>
```

The name of this JMS queue is ingestQueue, and we are going to see how easy it is to do place and read messages off this queue using EJB3.

The two steps to this integration are to write the Message Driven Bean (MDB) that is going to receive the data, and the session bean that is going to handle the request to ingest the data.

Message Driven Bean

We are going to create a MDB called IngestMDB using Java 5 annotations. To configure a java bean in Java 5 we use the following annotations, which can be overridden using a standard ejb-jar.xml if required.

```
@MessageDriven(activationConfig={
    @ActivationConfigProperty(
```

```
        propertyName="destinationType",
        propertyValue="javax.jms.Queue"),
    @ActivationConfigProperty(
        propertyName="destination",
        propertyValue="queue/ingestQueue"),
    @ActivationConfigProperty(
        propertyName="acknowledgeMode",
        propertyValue="Auto-acknowledge")
})
public class IngestMDB implements MessageListener {
....
```

In the annotations we have specified the queue we are listening to, and the acknowledge mode of the MDB.

To ingest the data, we are going to parse the message, and create a geometry entity which will be ingested into PostGIS.

Using annotations we can get the object we need to persist the sent data in one line;

```
@PersistenceContext(unitName="People")
private EntityManager entityManager;
```

Where the persistence context is defined in the associated persistence.xml (see source).

POJO Entity Bean

A POJO is a plain old Java object, very simple bean implementing standard Java 'getters' and 'setters'. The whole aim of this exercise it to be able to persist and retrieve geometry types, and fortunately thanks to annotations, and the great work in the PostGIS JDBC driver it is easy.

```
@Entity
@Table(name="people")
@NamedQuery(name="findPerson",
            query="SELECT DISTINCT OBJECT(p) FROM PersonEntity p WHERE ((p.name
= :name) AND (p.surname = :surname)) ORDER BY p.date")
public class PersonEntity {
```

The entity and the reference table are all defined by annotations, and we use an EJB-QL query to retrieve objects, JBoss / Hibernate annotations all support native SQL queries.

Question: Do the users want to do spatial queries in PostGIS space, or object space using the Java Topology Suite?

Having written a custom GeometryType for Hibernate (in the source) we can mark up our location column as a Geometry

```
@Type(type = "org.postgis.hibernate.GeometryType")
@Column(name="location", columnDefinition="Geometry")
public Geometry getLocation() {
```

It is as simple as that, to persist the data we use the entity manager;

```
entityManager.persist(person);
```

SOAP Annotations Java Bean

We want python, or another client to be able to talk to the server, and we certainly don't want to be doing this all by hand :-)

We can use JBossWS which uses standard annotations to mark out beans as SOAP services;

```
@Stateless
@WebService(
              name = "EndpointInterface",
              targetNamespace = "http://org.postgis/ejb/UserBean",
              serviceName = "PeopleFinder")
@SOAPBinding(style = SOAPBinding.Style.RPC)
public class UserBean implements UserBeanRemote{
```

If JBoss is running, look at http://localhost:8080/jbossws/ to see your service.

Security can be added by updating JBossWS to the latest version

http://today.java.net/pub/n/JBossws-1.0.1.GA

and annotating the class with (for example)

```
@SecurityDomain(value = "JBossWS")
@RunAs(value = "friend")
```

Python Web Service Client (SOAPpy)

Following the instructions here;

http://www.diveintopython.org/soap_web_services/install.html

and noting that fpconst has moved to here!

http://cheeseshop.python.org/pypi/fpconst/0.7.2

we can execute the following code to interact with our web service

```
from SOAPpy import SOAPProxy
name = 'Joe'
surname = 'Bloggs'
lat = 52.0
lon = 0.0
url = 'http://localhost:8080/ingest/UserBean'
namespace = 'http://org.postgis/ejb/UserBean'
server = SOAPProxy(url, namespace)
server.ingest(name=name, surname=surname, lat=lat, lon=lon)
p = server.findPerson(name='Joe', surname='Bloggs')

print p
```

Executing the following SQL confirms this has persisted to PostGIS;

```
SELECT id, name, surname, AsText(location), ingested FROM people;
```

Part 2 of the tutorial will cover Entity Bean Spatial queries, and security in more depth.

Contact:

Norman Barker, norman.barker@gmail.com