# Using Hibernate Entity and Dao in Plugin

Sometimes, you may want to have your own Hibernate entity and DAO to access extra database tables in your plugin. It is very simple to achieve that by adding the following files and classes to your plugin.
Similar to development in Spring + Hibernate, an application context file is required for your plugin. In my sample plugin, I created a productApplicationContext.xml as below

**Application context file (productApplicationContext.xml)**

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:p="http://www.springframework.org/schema/p"
       xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema
/beans/spring-beans-4.1.xsd">

    <bean id="productSessionFactory" class="org.springframework.orm.hibernate4.LocalSessionFactoryBean">
        <property name="dataSource" ref="setupDataSource"/>
        <property name="mappingResources">
            <list>
                <value>/org/joget/sample/product/model/Product.hbm.xml</value>
             </list>
        </property>
        <property name="hibernateProperties">
            <props>
                <prop key="hibernate.hbm2ddl.auto">update</prop>
                <prop key="hibernate.show_sql">false</prop>
                <prop key="hibernate.format_sql">false</prop>
            </props>
        </property>
    </bean>

    <bean id="productDao" class="org.joget.product.dao.ProductDaoImpl">
        <property name="sessionFactory" ref="productSessionFactory" />
    </bean>

</beans>
```

In the application context, I created 2 beans. Bean "productSessionFactory" is to initialize a session factory with the hibernate mapping file. Bean "productDao" is to initialize the DAO object of my sample plugin.

Next, we need a Hibernate mapping file. In my sample plugin, it is /org/joget/sample/product/model/Product.hbm.xml. It mapped the POJO "org.joget.product.model.Product" with "hibernate_product" table.

**Hibernate mapping file (Product.hbm.xml)**

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN" "http://hibernate.sourceforge.
net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
    <class entity-name="Product" name="org.joget.product.model.Product" table="hibernate_product">
        <id column="id" name="id" type="string"/>
        <property column="name" name="name" type="string"/>
        <property column="description" name="description" type="string"/>
    </class>
</hibernate-mapping>
```

You need a utility class to initialize your application context and retrieve the bean object.

Utility class

```
package org.joget.product;

import org.joget.apps.app.service.AppUtil;
import org.springframework.context.support.AbstractApplicationContext;
import org.springframework.context.support.GenericXmlApplicationContext;

public class AppContext {

    private static AppContext instance;
    private final GenericXmlApplicationContext springApplicationContext;

    public synchronized static AppContext getInstance() {
        if (instance == null) {
            instance = new AppContext();
        }
        return instance;
    }

    private AppContext() {
        this.springApplicationContext = new GenericXmlApplicationContext();
        this.springApplicationContext.setValidating(false);
        this.springApplicationContext.setClassLoader(this.getClass().getClassLoader());
        this.springApplicationContext.setParent(AppUtil.getApplicationContext());
        this.springApplicationContext.load("/productApplicationContext.xml");
        this.springApplicationContext.refresh();
    }

    public AbstractApplicationContext getAppContext() {
        return springApplicationContext;
    }

}
```

Sample DAO implementation

DAO Implementation

```
package org.joget.product.dao;

import java.util.Collection;
import org.joget.apps.app.service.AppUtil;
import org.joget.commons.spring.model.AbstractSpringDao;
import org.joget.commons.util.LogUtil;
import org.joget.product.model.Product;
import org.springframework.transaction.TransactionStatus;
import org.springframework.transaction.support.TransactionCallback;
import org.springframework.transaction.support.TransactionTemplate;

public class ProductDaoImpl extends AbstractSpringDao implements ProductDao {

    @Override
    public Boolean addProduct(Product product) {
        try {
            TransactionTemplate transactionTemplate = (TransactionTemplate) AppUtil.getApplicationContext().
getBean("transactionTemplate");
            Boolean result = (Boolean)transactionTemplate.execute(new TransactionCallback() {
                @Override
                public Object doInTransaction(TransactionStatus ts) {
                    save("Product", product);
                    return true;
                }
            });
            return result;
```

```
        } catch (Exception e) {
            LogUtil.error(ProductDaoImpl.class.getName(), e, "Add Product Error!");
            return false;
        }
    }

    @Override
    public Boolean updateProduct(Product product) {
        try {
            TransactionTemplate transactionTemplate = (TransactionTemplate) AppUtil.getApplicationContext().
getBean("transactionTemplate");
            Boolean result = (Boolean)transactionTemplate.execute(new TransactionCallback<Object>() {
                @Override
                public Object doInTransaction(TransactionStatus ts) {
                    merge("Product", product);
                    return true;
                }
            });
            return result;
        } catch (Exception e) {
            LogUtil.error(ProductDaoImpl.class.getName(), e, "Update Product Error!");
            return false;
        }
    }

    @Override
    public Boolean deleteProduct(String id) {
        try {
            TransactionTemplate transactionTemplate = (TransactionTemplate) AppUtil.getApplicationContext().
getBean("transactionTemplate");
            Boolean result = (Boolean)transactionTemplate.execute(new TransactionCallback() {
                @Override
                public Object doInTransaction(TransactionStatus ts) {
                    Product product = getProduct(id);
                    if (product != null) {
                        delete("Product", product);
                        return true;
                    } else {
                        return false;
                    }
                }
            });
            return result;
        } catch (Exception e) {
            LogUtil.error(ProductDaoImpl.class.getName(), e, "Delete Product Error!");
            return false;
        }
    }

    @Override
    public Product getProduct(String id) {
        try {
            TransactionTemplate transactionTemplate = (TransactionTemplate) AppUtil.getApplicationContext().
getBean("transactionTemplate");
            Product product = (Product)transactionTemplate.execute(new TransactionCallback() {
                @Override
                public Object doInTransaction(TransactionStatus ts) {
                    return (Product) find("Product", id);
                }
            });
            return product;
        } catch (Exception e) {
            LogUtil.error(ProductDaoImpl.class.getName(), e, "Get Product Error!");
            return null;
        }
    }

    @Override
    public Collection<Product> getProducts() {
        try {
            TransactionTemplate transactionTemplate = (TransactionTemplate) AppUtil.getApplicationContext().
```

```
getBean("transactionTemplate");
            Collection products = (Collection)transactionTemplate.execute(new TransactionCallback<Object>() {
                @Override
                public Object doInTransaction(TransactionStatus ts) {
                    return find("Product", "", null, null, null, null, null);
                }
            });
            return products;
        } catch (Exception e) {
            LogUtil.error(ProductDaoImpl.class.getName(), e, "Get Products Error!");
            return null;
        }
    }

}
```

After you have implemented your POJO and DAO class, you should be able to use your DAO in your plugin as follows.

Please refer to the attached sample plugin for the POJO and DAO implementation.


POJO and DAO usage

```
ProductDao productdao = (ProductDao) AppContext.getInstance().getAppContext().getBean("productDao");

Product p = new Product();
p.setId("001");
p.setName("Product A");
p.setDescription("Product A Descpription");

productdao.addProduct(p);
```

In this sample plugin, you are able to add, delete and list product by the following JSON API.

To add

```
http://localhost:8080/jw/web/json/plugin/org.joget.product.ProductApi/service?
_action=add&name=Product_A&desc=Product_Descpription
```

To delete

```
http://localhost:8080/jw/web/json/plugin/org.joget.product.ProductApi/service?_action=delete&id=001
```

To list all products

```
http://localhost:8080/jw/web/json/plugin/org.joget.product.ProductApi/service?_action=list
```