Spring Classic Configuration

Using Spring Data JPA in a classical Spring environment provides you with a lot of flexibility but also requires quite a bit of configuration.

Dependencies

To use Spring Data JPA in your project, you need to add a dependency to spring-data-jpa, your preferred JPA implementation and a JDBC driver

```
<dependency>
    <groupId>org.springframework.data</groupId>
    <artifactId>spring-data-jpa</artifactId>
    <version>${version.springdata}</version>
</dependency>
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-core</artifactId>
    <version>${version.hibernate}</version>
</dependency>
<dependency>
    <groupId>org.postgresql</groupId>
    <artifactId>postgresql</artifactId>
    <version>${version.postgresql}</version>
</dependency>
```

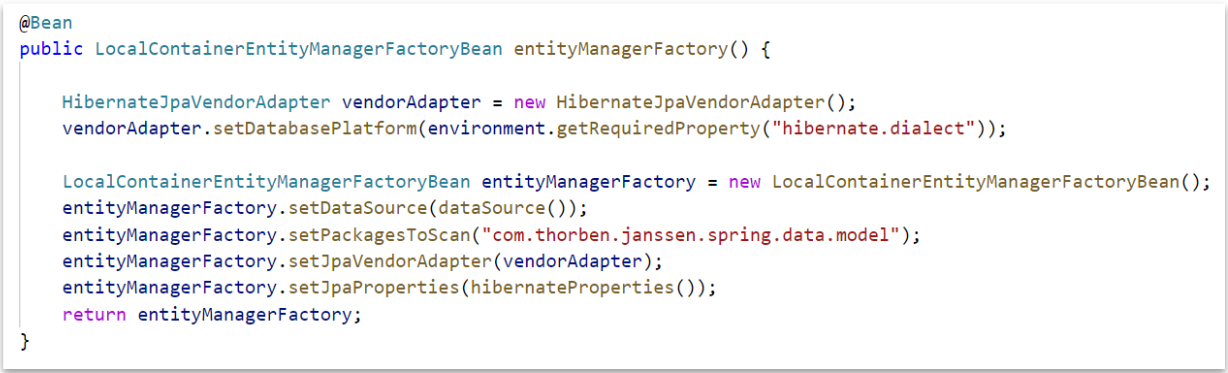
Configuration

There are 4 things you need to configure:

- EntityManagerFactoryBean
- DataSource
- TransactionManager
- JPA repositories

EntityManagerFactoryBean

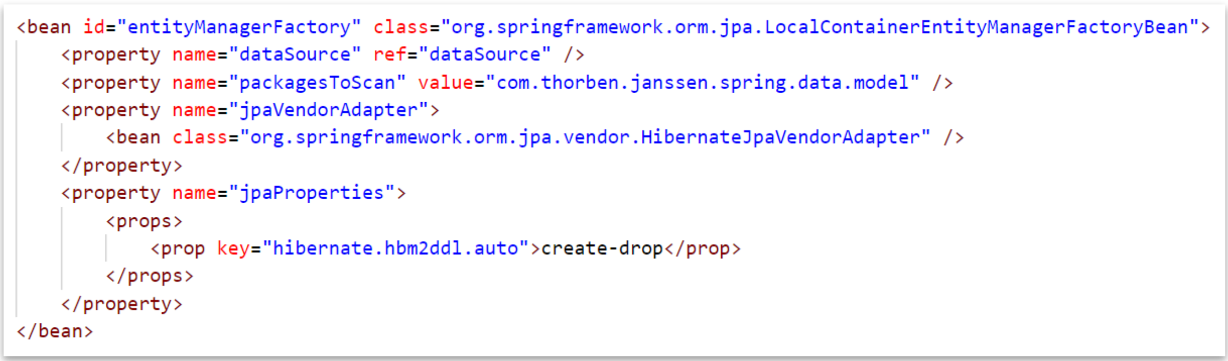
You can configure your *LocalEntityManagerFactoryBean* or the more flexible *LocalContainerEntityMangerFactoryBean* in your Java code or in an external XML file. Both read the *persistence.xml* file from the classpath, but also work entirely without one.



```java
@Bean
public LocalContainerEntityManagerFactoryBean entityManagerFactory() {

    HibernateJpaVendorAdapter vendorAdapter = new HibernateJpaVendorAdapter();
    vendorAdapter.setDatabasePlatform(environment.getRequiredProperty("hibernate.dialect"));

    LocalContainerEntityManagerFactoryBean entityManagerFactory = new LocalContainerEntityManagerFactoryBean();
    entityManagerFactory.setDataSource(dataSource());
    entityManagerFactory.setPackagesToScan("com.thorben.janssen.spring.data.model");
    entityManagerFactory.setJpaVendorAdapter(vendorAdapter);
    entityManagerFactory.setJpaProperties(hibernateProperties());
    return entityManagerFactory;
}
```



```xml
<bean id="entityManagerFactory" class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">
    <property name="dataSource" ref="dataSource" />
    <property name="packagesToScan" value="com.thorben.janssen.spring.data.model" />
    <property name="jpaVendorAdapter">
        <bean class="org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter" />
    </property>
    <property name="jpaProperties">
        <props>
            <prop key="hibernate.hbm2ddl.auto">create-drop</prop>
        </props>
    </property>
</bean>
```

You can reference the name of the *datasource* that shall be used with the created *EntityManagerFactory*.

The *packagesToScan* property defines the name of the packages in which you want to scan for entity classes.

The *jpaVendorAdapter* integrates Spring Data JPA with your preferred JPA implementation.

And you can also set additional JPA properties in the *jpaProperties* property, which are the same as the ones you set in the properties element of your persistence.xml file.

DataSource

The configuration of the *EntityManagerFactoryBean* referenced a data source. It configures the connection to your database.

```xml
<bean id="dataSource" class="com.zaxxer.hikari.HikariDataSource">
    <property name="jdbcUrl" value="jdbc:postgresql://localhost:5432/springDataJpaCourse" />
    <property name="username" value="postgres" />
    <property name="password" value="postgres" />
</bean>
```

```java
@Bean
public DataSource dataSource() {
    HikariDataSource ds = new HikariDataSource();
    ds.setMaximumPoolSize(10);
    ds.setJdbcUrl(environment.getProperty("datasource.url"));
    ds.setUsername(environment.getProperty("datasource.username"));
    ds.setPassword(environment.getProperty("datasource.password"));
    return ds;
}
```

TransactionManager

By configuring the transaction manager, you make sure that the operations performed by Spring Data JPA become part of the overall transaction.

```xml
<bean id="transactionManager" class="org.springframework.orm.jpa.JpaTransactionManager">
    <property name="entityManagerFactory" ref="entityManagerFactory" />
</bean>
```

```java
@Bean
public DataSource dataSource() {
    DriverManagerDataSource dataSource = new DriverManagerDataSource();
    dataSource.setUrl(environment.getProperty("datasource.url"));
    dataSource.setUsername(environment.getProperty("datasource.username"));
    dataSource.setPassword(environment.getProperty("datasource.password"));
    return dataSource;
}
```

JPA Repositories

Repositories help you avoid almost all of the boilerplate code required by JPA. The implementation of these repositories gets generated by Spring Data JPA based on your repository definitions.

This requires you to configure the *basePackage*. Spring Data JPA will scan all packages that start with the provided String for repository definitions and provide their required implementation at runtime.

```
<jpa:repositories base-package="com.thorben.janssen.spring.data.repository" />
```

```
@Configuration
@PropertySource({"classpath:application.properties" })
@EnableTransactionManagement
@EnableJpaRepositories("com.thorben.janssen.spring.data.repository")
public class Config {

    private Environment environment;

    public Config(Environment environment) {
        this.environment = environment;
    }
```