# Bean Definition Profiles

## Outline

Made available since Spring f/w ver. 3.1, bean definition profiles make feasible multiple beans to be defined for a single ID, for a corresponding bean for the activated profile to stay in operation over the runtime. With bean definition profile, all you need to do is to configure the profile as appropriate to apply the concerned bean to Spring Container.

Keep in mind you need to activate the profile before configuration, in order to avoid Exception(NoSuchBeanDefinitionException).

## Description

The section refers to how to configure and activate profiles.

### How to configure Profiles

You have a couple configuration options for profile, XML and Annotation.

#### Configuring XML Profile

To configure XML Profile, you can take to either the typical configuration of XML Bean or configuration of beans newly made available. You can either divide XML into two or keep an XML as-is when attempting to configure for a single Bean ID.

##### Typical XML Configuration

You need to secure the Bean ID that is unique to the XML configuration. When changing configuration of a Bean, you need to establish another Bean ID or configure the existing Bean ID to fit the configuration changed.

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:jdbc="http://www.springframework.org/schema/jdbc"
    xsi:schemaLocation="...">

    <bean id="transferService" class="com.bank.service.internal.DefaultTransferService">
        <constructor-arg ref="accountRepository"/>
        <constructor-arg ref="feePolicy"/>
    </bean>

    <bean id="accountRepository" class="com.bank.repository.internal.JdbcAccountRepository">
        <constructor-arg ref="dataSource"/>
    </bean>

    <bean id="feePolicy" class="com.bank.service.internal.ZeroFeePolicy"/>

    <jdbc:embedded-database id="dataSource">
        <jdbc:script location="classpath:com/bank/config/sql/schema.sql"/>
        <jdbc:script location="classpath:com/bank/config/sql/test-data.sql"/>
    </jdbc:embedded-database>
</beans>
```

**1. Bean Profiles - Dividing XML into two**

<transfer-service-config.xml>

Refer to the following code example for how to configure XML for “accountRepository” bean using "dataSource" Bean. Here, all you need to do is to keep the same Bean ID with the previous setting as the Bean ID "dataSource" is loaded, regardless of the Profile, when Spring Container is in operation:

```
<beans ...>
    <bean id="transferService" ... />

    <bean id="accountRepository" class="com.bank.repository.internal.JdbcAccountRepository">
        <constructor-arg ref="dataSource"/>
    </bean>

    <bean id="feePolicy" ... />
</beans>
```

<standalone-datasource-config.xml>

Refer to the following code example for how to configure XML defining “dataSource” bean used for the developmental phase. Here, the Profile is entitled "dev" and Embedded DB is defined as DataSource. Activate the Profile "dev" to work the Bean:

```
<beans profile="dev">
    <jdbc:embedded-database id="dataSource">
        <jdbc:script location="classpath:com/bank/config/sql/schema.sql"/>
        <jdbc:script location="classpath:com/bank/config/sql/test-data.sql"/>
    </jdbc:embedded-database>
</beans>
```

<jndi-datasource-config.xml>

Refer to the following code example for how to configure XML defining “dataSource” bean used for the operational phase. Here, the Profile is entitled "production" and JDNI is defined as DataSource. Activate the Profile "production" to work the Bean:

```
<beans profile="production">
    <jee:jndi-lookup id="dataSource" jndi-name="java:comp/env/jdbc/datasource"/>
</beans>
```

**2. Bean Profiles - Single-file XML configuration**

You may define multiple times the element <beans> within a single XML file that matches the concerned Profile.

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:jdbc="http://www.springframework.org/schema/jdbc"
    xmlns:jee="http://www.springframework.org/schema/jee"
    xsi:schemaLocation="...">

    <bean id="transferService" class="com.bank.service.internal.DefaultTransferService">
        <constructor-arg ref="accountRepository"/>
        <constructor-arg ref="feePolicy"/>
    </bean>

    <bean id="accountRepository" class="com.bank.repository.internal.JdbcAccountRepository">
        <constructor-arg ref="dataSource"/>
    </bean>
```

```
    <bean id="feePolicy" class="com.bank.service.internal.ZeroFeePolicy"/>
    <beans profile="dev">
        <jdbc:embedded-database id="dataSource">
            <jdbc:script location="classpath:com/bank/config/sql/schema.sql"/>
            <jdbc:script location="classpath:com/bank/config/sql/test-data.sql"/>
        </jdbc:embedded-database>
    </beans>
    <beans profile="production">
        <jee:jndi-lookup id="dataSource" jndi-name="java:comp/env/jdbc/datasource"/>
    </beans>
</beans>
```

## Configuring Annotation Profile

You can configure Annotation Profile using @Profile. With the class containing the string @Bean, Beans are registered when the class contains strings of both @Configuration and @Profile(title of “Profile”).

The section describes how to work the typical @Configuration class and configure Beans via @Profile.

### Typical @Configuration Class

Add @Configuration immediately before the class begins to register the title of the pre-defined method, along with @Bean, as the desired Bean ID.

```
@Configuration
public class TransferServiceConfig {

    @Bean
    public TransferService transferService() {
        return new DefaultTransferService(accountRepository(), feePolicy());
    }

    @Bean
    public AccountRepository accountRepository() {
        return new JdbcAccountRepository(dataSource());
    }

    @Bean
    public FeePolicy feePolicy() {
        return new ZeroFeePolicy();
    }

    @Bean
    public DataSource dataSource() {
        return new EmbeddedDatabaseBuilder()
            .setType(EmbeddedDatabaseType.HSQL)
            .addScript("classpath:com/bank/config/sql/schema.sql")
            .addScript("classpath:com/bank/config/sql/test-data.sql")
            .build();
    }
}
```

### Configuring @Profile

Refer to the following code example for configuration of Bean Profile XML as Annotation(@Profile). Keep in mind @Profile works the same as Beans profile of XML and @Bean matches the configuration of Bean for XML.

Note the following Java code example assumes the Profile entitled "dev" for Bean “dataSource”:

```
@Configuration
@Profile("dev")
public class StandaloneDataConfig {
	@Bean
	public DataSource dataSource() {
		return new EmbeddedDatabaseBuilder()
			.setType(EmbeddedDatabaseType.HSQL)
			.addScript("classpath:com/bank/config/sql/schema.sql")
			.addScript("classpath:com/bank/config/sql/test-data.sql")
			.build();
	}
}
```

Refer to the following Java code example assumes the Profile entitled "production" for Bean “dataSource”:

```
@Configuration
@Profile("production")
public class JndiDataConfig {

	@Bean
	public DataSource dataSource() throws Exception {
		Context ctx = new InitialContext();
		return (DataSource) ctx.lookup("java:comp/env/jdbc/datasource");
	}
}
```

## Activating Profile

You can activate the duly configured Profile by either declaration or Java coding:

### 1. How to activate Profile by declaration (web.xml, environment variables, properties, etc.)

You can activate the duly configured Profile by way of environment variables, properties, etc. when executing web.xml or JVM. Using web.xml is preferred to Java coding when the war file serves as an execution file or organized broadcasting management.

<Activating Profile using web.xml>

```
<servlet>
	<servlet-name>dispatcher</servlet-name>
	<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
	<init-param>
		<param-name>spring.profiles.active</param-name>
		<param-value>production</param-value>
	</init-param>
  </servlet>
```

<Activate the Profile using Environment Variables when executing JVM>

```
java -Dspring.profiles.active="production"
```

### 2. How to activate Profile by Java coding

You may also activate the Profile via the interface entitled 'Environment' If you have Spring 3.1 or newer.

<Activating Profile by Java coding when an XML Profile exists>
Activate the Bean whose Profile is entitled "dev". Activate the Profile, working the method "setActiveProfiles" via the Environment loaded from GenericXmlApplicationContext and load the concerned configuration xml.

Refer to the following code example to activate the Bean whose Profile is entitled "production", skipping the Profile entitled "Dev".

```
GenericXmlApplicationContext ctx = new GenericXmlApplicationContext();
ctx.getEnvironment().setActiveProfiles("production");
ctx.load("classpath:/com/bank/config/xml/*-config.xml");
ctx.refresh();
```

<Activating Profile by Java coding when @Profile Profile exists>
Activate the Profile working the method "setActiveProfiles". Scan the entirety of @Configuration class within the package “com.bank.config.code” to activate the Beans whose Profiles are entitled "dev" and skip the Beans whose Profiles are entitled "production".

```
AnnotationConfigApplicationContext ctx = new AnnotationConfigApplicationContext();
ctx.getEnvironment().setActiveProfiles("dev");
ctx.scan("com.bank.config.code"); // find and register all @Configuration classes within
ctx.refresh();
```

### 3. How to activate Profile using annotation @ActiveProfile for JUnit4 Test

Another option to activate Profile is to add the string @ActiveProfile to the test class for JUnit4. Keep in mind the string should follow @ContextConfiguration, and be followed by the desired title of Profile.

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(loader=AnnotationConfigContextLoader.class)
@ActiveProfiles("annotationProfile")
public class SpringAnnotationProfileTest {
}
```

## References


- Spring Framework - Reference Document / 3.2 Bean Definition Profiles
- Spring team blog - XML profiles
- Spring team blog - introducing @Profile