**1. Overview**

In this quick tutorial, we'll show how to read a YAML properties file using the *@PropertySource* annotation in Spring Boot.

**2. *@PropertySource* and *YAML* Format**

Spring Boot has great support for externalized configuration. Also, it's possible to use different ways and formats to read the properties in the Spring Boot application out-of-the-box.

However, **by default, *@PropertySource* doesn't load YAML files**. This fact is explicitly mentioned in the [official documentation](https://docs.spring.io/spring-boot/docs/current/reference/html/spring-boot-features.html#boot-features-external-config-yaml-shortcomings).

So, if we want to use the *@PropertySource* annotation in our application, we need to stick with the standard *properties* files. **Or we can implement the missing puzzle piece ourselves!**

**3. Custom *PropertySourceFactory***

As of Spring 4.3, *@PropertySource* comes with the *factory* attribute. We can make use of it to **provide our custom implementation of the *PropertySourceFactory*, which will handle the YAML file processing**.

This is easier than it sounds! Let's see how to do this:

**public** **class** **YamlPropertySourceFactory** **implements** **PropertySourceFactory** {

@Override

**public** PropertySource<?> createPropertySource(String name, EncodedResource encodedResource)

**throws** IOException {

**YamlPropertiesFactoryBean** factory = **new** **YamlPropertiesFactoryBean**();

factory.setResources(encodedResource.getResource());

**Properties** properties = factory.getObject();

**return** **new** **PropertiesPropertySource**(encodedResource.getResource().getFilename(), properties);

}

}Copy

As we can see, it's enough to implement a single [*createPropertySource*](https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/core/io/support/PropertySourceFactory.html#createPropertySource-java.lang.String-org.springframework.core.io.support.EncodedResource-) method.

In our custom implementation, first, **we used the**[***YamlPropertiesFactoryBean***](https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/beans/factory/config/YamlPropertiesFactoryBean.html)**to convert the resources in YAML format to the *java.util.Properties* object**.

Then, we simply returned a new instance of the [*PropertiesPropertySource*](https://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/core/env/PropertiesPropertySource.html), which is a wrapper that allows Spring to read the parsed properties.

**4. *@PropertySource* and *YAML* in Action**

Let's now put all the pieces together and see how to use them in practice.

First, let's create a simple YAML file – *foo.yml*:

yaml:

name: foo

aliases:

- abc

- xyzCopy

Next, let's create a properties class with *@ConfigurationProperties* and use our custom *YamlPropertySourceFactory:*

@Configuration

@ConfigurationProperties(prefix = "yaml")

@PropertySource(value = "classpath:foo.yml", factory = YamlPropertySourceFactory.class)

**public** **class** **YamlFooProperties** {

**private** String name;

**private** List<String> aliases;

// standard getter and setters

}Copy

And finally, **let's verify that the properties are properly injected**:

@RunWith(SpringRunner.class)

@SpringBootTest

**public** **class** **YamlFooPropertiesIntegrationTest** {

@Autowired

**private** YamlFooProperties yamlFooProperties;

@Test

**public** **void** **whenFactoryProvidedThenYamlPropertiesInjected**() {

assertThat(yamlFooProperties.getName()).isEqualTo("foo");

assertThat(yamlFooProperties.getAliases()).containsExactly("abc", "xyz");

}

}Copy

**5. Conclusion**

To sum up, in this quick tutorial, we first showed how easy it is to create a custom *PropertySourceFactory*. After that, we presented how to pass this custom implementation to the *@PropertySource* using its *factory* attribute.

Consequently, **we were able to successfully load the YAML properties file into our Spring Boot application**.