

Quarkus - Accessing application properties with Spring Boot properties API

If you prefer to use Spring Boot `@ConfigurationProperties` annotated class to access application properties instead of a Quarkus native `@ConfigProperties` or a MicroProfile `@ConfigProperty` approach, you can do that with this extension.



This technology is considered preview.

In *preview*, backward compatibility and presence in the ecosystem is not guaranteed. Specific improvements might require to change configuration or APIs and plans to become *stable* are under way. Feedback is welcome on our [mailing list](#) or as issues in our [GitHub issue tracker](#).

For a full list of possible extension statuses, check our [FAQ entry](#).

Prerequisites

To complete this guide, you need:

- less than 15 minutes
- an IDE
- JDK 1.8+ installed with `JAVA_HOME` configured appropriately
- Apache Maven 3.6.3

Solution

We recommend that you follow the instructions in the next sections and create the application step by step. However, you can go right to the completed example.

Clone the Git repository: `git clone https://github.com/quarkusio/quarkus-quickstarts.git`, or download an [archive](#).

The solution is located in the `spring-boot-properties-quickstart` directory.

Creating the Maven project

First, we need a new project. Create a new project with the following command:

```
mvn io.quarkus:quarkus-maven-plugin:1.4.2.Final:create \
  -DprojectId=org.acme \
  -DprojectArtifactId=spring-boot-properties-quickstart \
  -DclassName="org.acme.spring.boot.properties.GreetingResource" \
  -Dpath="/greeting" \
  -Dextensions="spring-boot-properties"
cd spring-boot-properties-quickstart
```

This command generates a Maven project with a REST endpoint and imports the `spring-boot-properties` extension.

GreetingController

The Quarkus Maven plugin automatically generated a `GreetingResource` JAX-RS resource in the `src/main/java/org/acme/spring/boot/properties/GreetingResource.java` file that looks like:

```
package org.acme.spring.boot.properties;

import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;

@Path("/hello")
public class GreetingResource {

    @GET
    @Produces(MediaType.TEXT_PLAIN)
    public String hello() {
        return "hello";
    }
}
```

Injecting properties

Create a new class `src/main/java/org/acme/spring/boot/properties/GreetingProperties.java` with a message field:

```

package org.acme.spring.boot.properties;

import
org.springframework.boot.context.properties.ConfigurationProperties
;

@ConfigurationProperties("greeting")
public class GreetingProperties {

    public String text;
}

```

Here `text` field is public, but it could also be a private field with getter and setter or just a public getter in an interface. Because `text` does not have a default value it is considered required and unless it is defined in a configuration file (`application.properties` by default) your application will fail to start. Define this property in your `src/main/resources/application.properties` file:

```

# Your configuration properties
greeting.text = hello

```

Now modify `GreetingResource` to start using the `GreetingProperties`:

```

package org.acme.spring.boot.properties;

import javax.inject.Inject;
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;

@Path("/greeting")
public class GreetingResource {

    @Inject
    GreetingProperties properties;

    @GET
    @Produces(MediaType.TEXT_PLAIN)
    public String hello() {
        return properties.text;
    }
}

```

Run the tests to verify that application still functions correctly.

Package and run the application

Run the application with: `./mvnw compile quarkus:dev`. Open your browser to <http://localhost:8080/greeting>.

Changing the configuration file is immediately reflected.

As usual, the application can be packaged using `./mvnw clean package` and executed using the `-runner.jar` file. You can also generate the native executable with `./mvnw clean package -Pnative`.

Default values

Now let's add a suffix for a greeting for which we'll set a default value.

Properties with default values can be configured in a configuration file just like any other property. However, the default value will be used if the property was not defined in a configuration file.

Go ahead and add the new field to the `GreetingProperties` class:

```
package org.acme.spring.boot.properties;

import
org.springframework.boot.context.properties.ConfigurationProperties
;

@ConfigurationProperties("greeting")
public class GreetingProperties {

    public String text;

    public String suffix = "!";
}
```

And update the `GreetingResource` and its test `GreetingResourceTest`:

```

package org.acme.spring.boot.properties;

import javax.inject.Inject;
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;

@Path("/greeting")
public class GreetingResource {

    @Inject
    GreetingProperties properties;

    @GET
    @Produces(MediaType.TEXT_PLAIN)
    public String hello() {
        return properties.text + properties.suffix;
    }
}

```

```

package org.acme.spring.boot.properties;

import io.quarkus.test.junit.QuarkusTest;
import org.junit.jupiter.api.Test;

import static io.restassured.RestAssured.given;
import static org.hamcrest.CoreMatchers.is;

@QuarkusTest
public class GreetingResourceTest {

    @Test
    public void testHelloEndpoint() {
        given()
            .when().get("/greeting")
            .then()
                .statusCode(200)
                .body(is("hello!"));
    }
}

```

Run the tests to verify the change.

Optional values

Properties with optional values are the middle-ground between standard and properties with default values. While a missing property in a configuration file will not cause your application to fail, it will nevertheless not have a value set. We use `java.util.Optional` type to define such properties.

Add an optional `name` property to the `GreetingProperties`:

```
package org.acme.spring.boot.properties;

import java.util.Optional;

import org.springframework.boot.context.properties.ConfigurationProperties
;

@ConfigurationProperties("greeting")
public class GreetingProperties {

    public String text;

    public String suffix = "!";

    public Optional<String> name;
}
```

And update the `GreetingResource` and its test `GreetingResourceTest`:

```

package org.acme.spring.boot.properties;

import javax.inject.Inject;
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;

@Path("/greeting")
public class GreetingResource {

    @Inject
    GreetingProperties properties;

    @GET
    @Produces(MediaType.TEXT_PLAIN)
    public String hello() {
        return properties.text + ", " + properties.name.orElse("
You") + properties.suffix;
    }
}

```

```

package org.acme.spring.boot.properties;

import io.quarkus.test.junit.QuarkusTest;
import org.junit.jupiter.api.Test;

import static io.restassured.RestAssured.given;
import static org.hamcrest.CoreMatchers.is;

@QuarkusTest
public class GreetingResourceTest {

    @Test
    public void testHelloEndpoint() {
        given()
            .when().get("/greeting")
            .then()
                .statusCode(200)
                .body(is("hello, You!"));
    }
}

```

Run the tests to verify the change.

Grouping properties

Now we have three properties in our `GreetingProperties` class. While `name` could be considered more of a runtime property (and maybe could be passed as an HTTP query parameter in the future), `text` and `suffix` are used to define a message template. Let's group these two properties in a separate inner class:

```
package org.acme.spring.boot.properties;

import java.util.Optional;

import org.springframework.boot.context.properties.ConfigurationProperties
;

@ConfigurationProperties("greeting")
public class GreetingProperties {

    public Message message;

    public Optional<String> name;

    public static class Message {

        public String text;

        public String suffix = "!";

    }

}
```

Here `Message` properties class is defined as an inner class, but it could also be a top level class.

Having such property groups brings more structure to your configuration. This is especially useful when the number of properties grows.

Because of the additional class, our property names have changed. Let's update the properties file and the `GreetingResource` class.

```
# Your configuration properties
greeting.message.text = hello
```



```

package org.acme.spring.boot.properties;

import javax.inject.Inject;
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;

@Path("/greeting")
public class GreetingResource {

    @Inject
    GreetingProperties properties;

    @GET
    @Produces(MediaType.TEXT_PLAIN)
    public String hello() {
        return properties.message.text + ", " + properties.name
        .orElse("You") + properties.message.suffix;
    }
}

```

More Spring guides

Quarkus has more Spring compatibility features. See the following guides for more details:

- [Quarkus - Extension for Spring DI](#)
- [Quarkus - Extension for Spring Web](#)
- [Quarkus - Extension for Spring Data JPA](#)
- [Quarkus - Extension for Spring Security](#)
- [Quarkus - Reading properties from Spring Cloud Config Server](#)