

# How to easily use application.properties property file

Verisoft framework utilizes the Spring library to make it super easy to use and add properties.

## General Instructions

In order to be able to use the built-in properties (like external URL) you will need to do the following:

1. Extend your test with `SpringContext.class`
2. Define a `ContextConfiguration` annotation sent to `EnvConfig.class`
3. `@Autowired` your property

In addition, in case you want to extend and add properties of your own you will need to do the following:

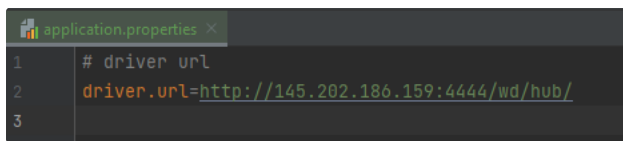
1. Add property to the property file
2. Create a class that loads the property
3. `@Autowired` the class in your test class

Some more information about the Spring framework and the dependency injection mechanism can be found [here](#).

## Using the external URL property

Built into the framework there is support for external use of the driver URL.

Take a look at the [src/test/application.properties](#) file. You can see there is a property named `driver.url`:



```
application.properties x
1 # driver url
2 driver.url=http://145.202.186.159:4444/wd/hub/
3
```

this is a definition of the driver we want to use. **Note! If you just want to use your local driver set this value to null, or simply do not put the `@DriverURL` annotation in your tests.**

Next, here is the code:

```
1 @Slf4j
2 @ExtendWith(SpringExtension.class)
3 @ContextConfiguration(classes = {EnvConfig.class})
4 public class PropertyFileLoaderTest extends BaseTest{
5
6     @Autowired
7     @DriverUrl
8     private URL url;
9
10    @DriverCapabilities
11    private DesiredCapabilities capabilities = new DesiredCapabilities();
12    {
13        ChromeOptions options = new ChromeOptions();
14        options.addArguments("--no-sandbox");
15        options.addArguments("--headless");
16
17        capabilities.setBrowserName("chrome");
```

```

18     options.merge(capabilities);
19 }
20
21 @Test
22 @DisplayName("Injected URL")
23 public void injectedURL(VerisoftDriver driver) throws InterruptedException {
24     driver.get("https://www.wikipedia.org/");
25     Report.info("Loaded page from a url found in application.properties");
26 }
27 }
28

```

Specifically, you can see in lines 2-4:

`@ExtendWith(SpringExtension.class)` - This tells JUnit 5 that spring is involved in the process

`@ContextConfiguration(classes = {EnvConfig.class})` - This tells JUnit 5 where to look for the root configuration file. This root config file will locate all the other config files. Specifically, it will locate the `SeleniumConfig.class` which is in charge of loading the URL property definition.

Next, we will define the URL in our code. Lines 6-8 do exactly that:

`@Autowired` - This tells Spring that the next variable should be located using the Spring framework

`@DriverUrl` - This tells the Verisoft framework that the URL field below should be injected into the VerisoftDriver instance.

`private URL url` - The actual URL to be used

Next, we will demonstrate [how to define your own property file](#).