Adding Custom WebDriver Listeners

Selenium supports listeners. This page assumes you are familiar with the Selenium listener mechanism,

and the EventFiringDecorator class. If you are not familiar with the listener mechanism, please visit the JavaDoc or an example of how to use WebDriverListener.

VerisoftDriver listeners

The VerisoftDriver object has built-in listeners, which mainly write log entries. The listeners are initialized during the driver initialization process. There are listeners for web and for mobile (Appium listeners). You can check the class documentation or take a look at the class itself.

Adding a Custom Listener

The first step is to write a listener. Here is a simple example, which can also be found here:

```
1  @ToString
2  @NoArgsConstructor
3  @Slf4j
4  public final class ExampleListener implements WebDriverListener {
5
6      @Override
7      public void beforeAnyCall(Object target, Method method, Object[] args) {
8          log.info("************ Example Listener in Action ************");
9      }
10 }
```

This basic listener simply adds the log entry upon every WebDriver action

Next, we need to register the listener with the VerisoftDriver:

```
1  @Test
2  @ExtendWith(DriverInjectionExtension.class)
3  public void addListenerInline(VerisoftDriver driver) {
4      WebDriverListener listener = new ExampleListener();
5      driver.addListener(listener);
6      driver.get("https://www.google.com");
7  }
```

After creating our listener in line 4, we simply add the listener to the driver. That's it! The listener is added to the driver and will be fired once the `driver.get()` command is called. A working example can be found here.

Advanced: Adding a Custom Listener As Part of the Framework

Since each test method creates a new VerisoftDriver instance, the listener that we created in the previous example will not work once the next test method will be launched, and a new instance of the VerisoftDriver will be initialized. If we want a more in-depth robust way to add our listeners to all the tests, we need to create a custom driver injection extension, instead of the extension we are currently using.

First, we create a new CustomDriverInjectionExtension class which extends the original DriverInjectionExtension:

```
1  public class CustomDriverInjectionExtension extends DriverInjectionExtension {
```

Each method that is implemented in the DriverInjectionextension should be called from the CustomDriverInjectionExtension and invoke the superclass. For instance:

```
@Override
public boolean supportsParameter(ParameterContext parameterContext,
                 ExtensionContext extensionContext) throws ParameterResolutionException {
    return super.supportsParameter(parameterContext, extensionContext);
}
```

The following method will add the listener as part of the injection phase:

```
@Override
public Object resolveParameter(ParameterContext parameterContext, ExtensionContext extensionContext) throws Param
    WebDriver generatedDriver = ((WebDriver) super.resolveParameter(parameterContext, extensionContext));

    WebDriverListener listener = new ExampleListener();
    ((VerisoftDriver) generatedDriver).addListener(listener);

    return generatedDriver;
}
```

(Check out the full implementation of the class here)

and the test should extend the new extension:

```
@Test
@ExtendWith(CustomDriverInjectionExtension.class)
public void addListenerUsingExtension(VerisoftDriver driver) {
    driver.get("https://www.google.com");
}
```