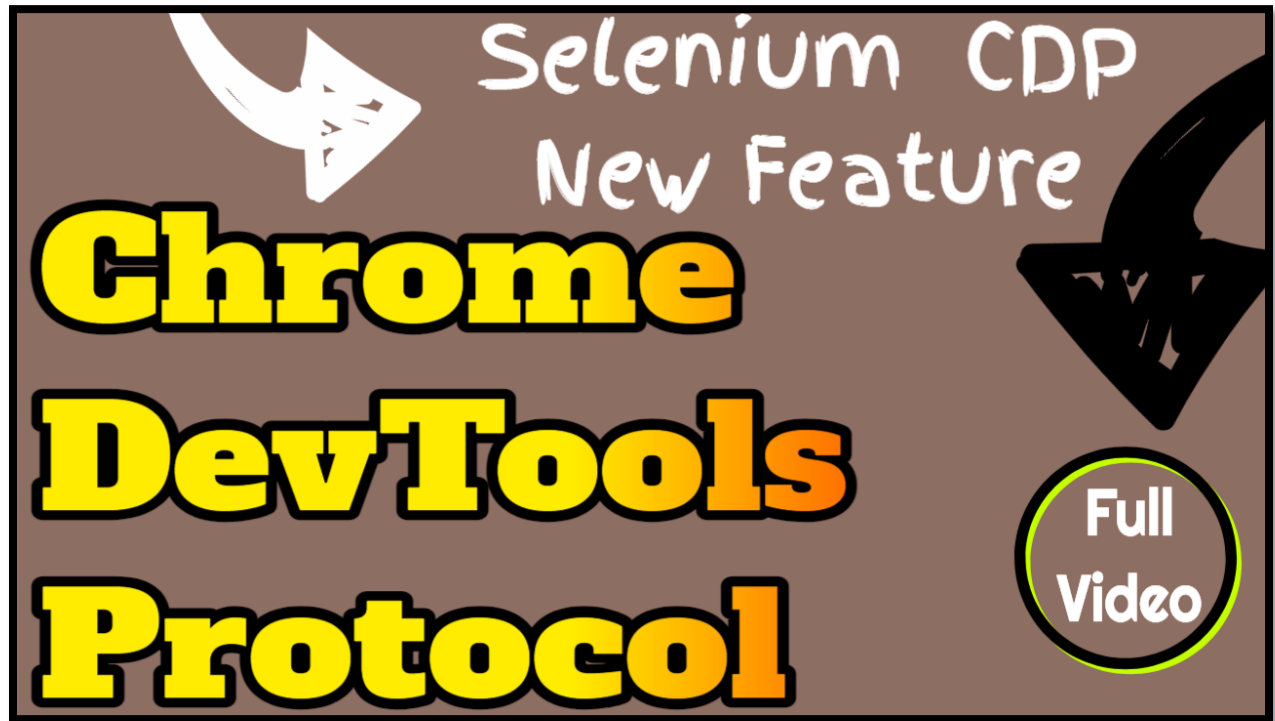


## Selenium 4 New Chrome DevTools Protocol



### Table of Contents

Selenium 4 Playlist .....	2
Introduction .....	2
View Console Logs.....	6
GeoLocation .....	11
Slow Down Internet Connection.....	15
Emulate Offline .....	20
Certificate Error Website .....	26
Contact Info.....	30

## Selenium 4 Playlist

[https://www.youtube.com/watch?v=BM3Gs0MzVdM&list=PLfp-cJ6BH8u\\_4AMzeLVizVfgn4SCywSTJ&index=13](https://www.youtube.com/watch?v=BM3Gs0MzVdM&list=PLfp-cJ6BH8u_4AMzeLVizVfgn4SCywSTJ&index=13)

## Introduction

Hello and Welcome, in this session, I will talk about CDP which is an acronym for Chrome Debugging Protocol. It's a new Selenium 4 API feature that's designed for debuggers. All browsers built on the Chromium platform has an option for Developer Tools. Therefore, another name for CDP can be called Chrome DevTools Protocol.

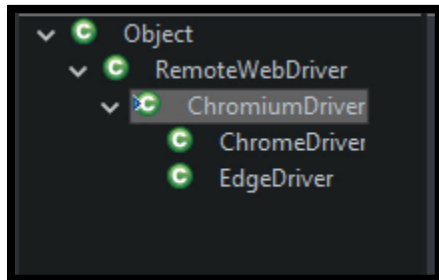
If I go to Chrome then click this vertical ellipsis, navigate to More Tools, and at this point, we see Developer Tools. DevTools is short for Developer Tools. Click and we see a few tabs. Elements is the most popular tab for Automation but we also have Console, Sources, Network, Performance, Memory, Application, and Security. It's the same panel if I right click a page and click Inspect.



Microsoft Edge also has these same tabs after accessing Developer Tools. We click this ellipsis, More Tools and here's Developer Tools.



Selenium 4 provides a way for us to take advantage of Chrome and Microsoft Edge's debugging protocol. Let's look behind the scenes at the classes and methods. Starting with selenium-chromium-driver, we see the ChromiumDriver class, open the Type Hierarchy. Notice, the ChromeDriver and EdgeDriver classes are both under the ChromiumDriver class. That's because ChromiumDriver is the parent. However, ChromiumDriver is a child to the parent RemoteWebDriver.



In a nutshell, EdgeDriver extends ChromiumDriver. ChromeDriver extends ChromiumDriver but ChromiumDriver extends RemoteWebDriver.

```

public class EdgeDriver extends ChromiumDriver {

    public EdgeDriver() { this(new EdgeOptions()); }

    public EdgeDriver(EdgeOptions options) {
        this(new EdgeDriverService.Builder().build(), options);
    }
  
```

```

public class ChromeDriver extends ChromiumDriver {

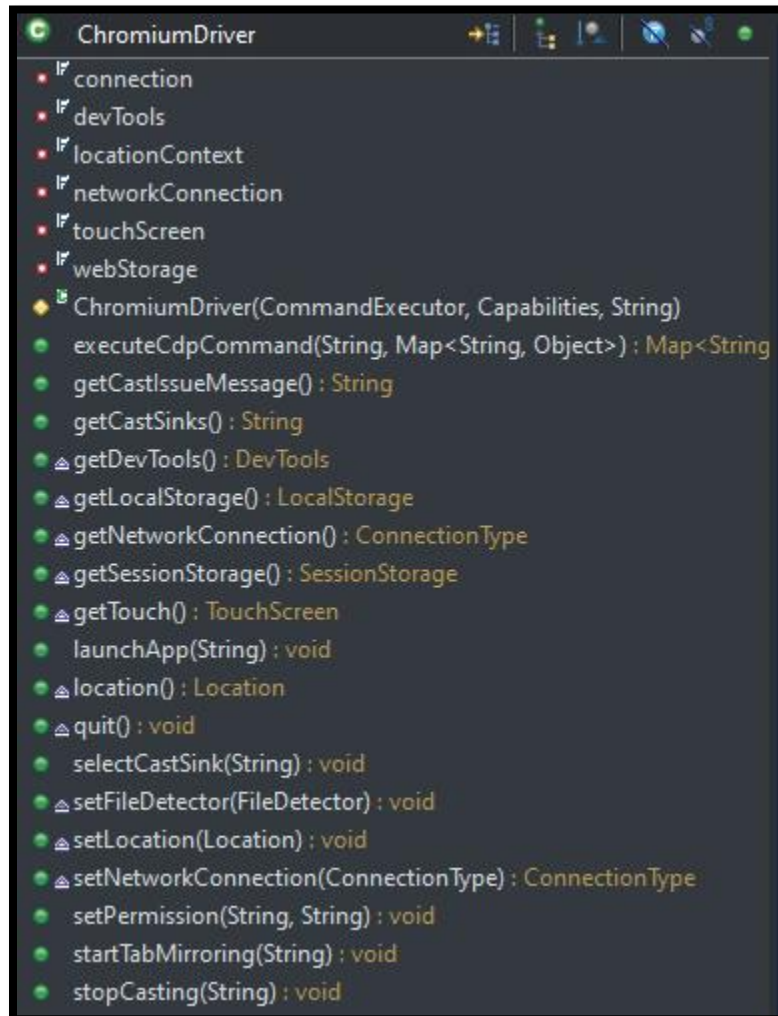
    /**
     * Creates a new ChromeDriver using the {@link ChromeDriverService}
     * server configuration.
     *
     * @see #ChromeDriver(ChromeDriverService, ChromeOptions)
     */
  
```

```

public class ChromiumDriver extends RemoteWebDriver
    implements HasDevTools, HasTouchScreen, LocationContext, NetworkContext {

    private final RemoteLocationContext locationContext;
    private final RemoteWebStorage webStorage;
    private final TouchScreen touchScreen;
  
```

There's a lot of methods in the ChromiumDriver class but 2 methods allow us to control Developer Tools in Chrome and Microsoft Edge.



Those 2 methods are executeCdpCommand and getDevTools. The executeCdpCommand allows us to directly execute a Chrome DevTool Protocol command by passing in a parameter for that command. getDevTools is a method that returns DevTools.

```
public Map<String, Object> executeCdpCommand(String commandName, Map<String, Object> parameters) {
    Require.nonNull("Command name", commandName);
    Require.nonNull("Parameters", parameters);
}
```

```
@Override
public DevTools getDevTools() {
    return devTools.orElseThrow() -> new WebDriverException("Unable to create DevTools connection");
}
```

DevTools is a class that has methods to handle developer options. We see close, send, addListener. Know what, let me go back to the Project Explorer and look at the methods for DevTools. Here are the methods within DevTools: addListener, clearListener, close, createSession, createSessionIfThereIsNotOne, getCdpSession, and send.



We see a lot of methods for DevTools because we can do a lot of things with CDP. I'm going to show you how to view the Console Logs when it comes to CDP, Mock a GeoLocation, and Enable the Network Speed. Thanks for watching and I'll see you in the next session starting with View Console Logs. If you are interested in more videos, feel free to subscribe to my YouTube channel and click the bell icon. Also, follow me on Twitter, connect with me on LinkedIn and Facebook.

## View Console Logs

There are 2 objectives for viewing a browser's Console panel. First, is to execute JavaScript. Second, is to view messages logged by the browser or logged by the developer. In this session, our focus will be to view messages logged by the browser. Most of the times, we view the Console Logs to get details about an error. With those details, we can identify the root cause of the problem.

Our AUT is this [lego page](#) and CTRL + SHIFT + I is a shortcut to the Developer Tools panel. In the Console tab, we already have some logs because the page has already been loaded. We can clear the logs, reload the page, then watch the logs back show up. DevTool failed to load SourceMap and HTTP error Status Code 404. The DevTools class in Selenium 4 provides a way to listen to these logs which has 4 Log Levels: Verbose, Info, Warnings, and Errors. Verbose is not checked for default. Therefore, the messages will not be wordy. It won't have many words in the message. Info means the logs will have important information about an event. Warnings indicate a strange condition might cause a problem with operations. Errors let us know a condition is likely to cause a problem with operations.

Now for our Test Script, chromedriver is setup and the window is maximized.

```
public class ViewConsoleLogs {  
  
    ChromeDriver driver;  
  
    @BeforeClass  
    public void setUp () {  
        WebDriverManager.chromedriver().setup();  
        driver = new ChromeDriver();  
        driver.manage().window().maximize();  
    }  
}
```

Start by writing @Test / public void viewConsoleLogs () {} Let's add some steps but write them as comments: First we // Get The DevTools and Create A Session. Then, // Send A Command To Enable The Logs. Next, we // Add A Listener For The Logs. Finally, we // Load The AUT



```
@Test
public void viewConsoleLogs () {
    // Get The DevTools And Create A Session

    // Send A Command To Enable The Logs

    // Add A Listener For The Logs

    // Load The AUT
}
```

To get the dev tools, we must cast the driver if WebDriver is our type. We cannot write `driver.getDevTools`. Notice, `getDevTools` is not available. We have to cast the driver by writing `((ChromeDriver) driver).getDevTools()`. If you want to directly write `driver.getDevTools`, we must change the type to `ChromeDriver`. Now, I'm going to start over and write `driver.getDevTools()`.

Do you see how `getDevTools` return `DevTools`? That's an indicator, this statement must be assigned to `DevTools` with an object reference of anyname but I'm going to write `devTools =` . Now, we have access to all of the methods I mentioned in the [Introduction](#). Before, we perform any more steps, we must take control of the Developer Tools panel in the browser by creating a session: `devTools.createSession()`. The purpose of this statement is to start a session in the Chrome browser.

```
@Test
public void viewConsoleLogs () {
    // Get The DevTools And Create A Session
    DevTools devTools = driver.getDevTools();
    devTools.createSession();
}
```

Next, is to enable logs in the Console. `devTools.send()`. Send is a method that has built in commands and accepts Command as a parameter. It allows us interact with the Developer Tools. `Log.enable()` allows us to listen to the logs but for here we see `Log.enable()` in the next part is one of those commands for the `send()` method. Log serves as a representation or model for a CDP domain.

```
// Send A Command To Enable The Logs
devTools.send(Log.enable());
```

Now that we have the logs enabled, this is the part we go and listen for logs by writing `devTools.addListener()`. Once again, we write Log. but this time it's an event for `entryAdded()`, `logEntry -> { }`. To use this Lambda expression (`->`), you must have Java 1.8 or higher.

```
// Add A Listener For The Logs
devTools.addListener(Log.entryAdded(), logEntry -> {
    System.out.println(logEntry.asSeleniumLogEntry());
});
```

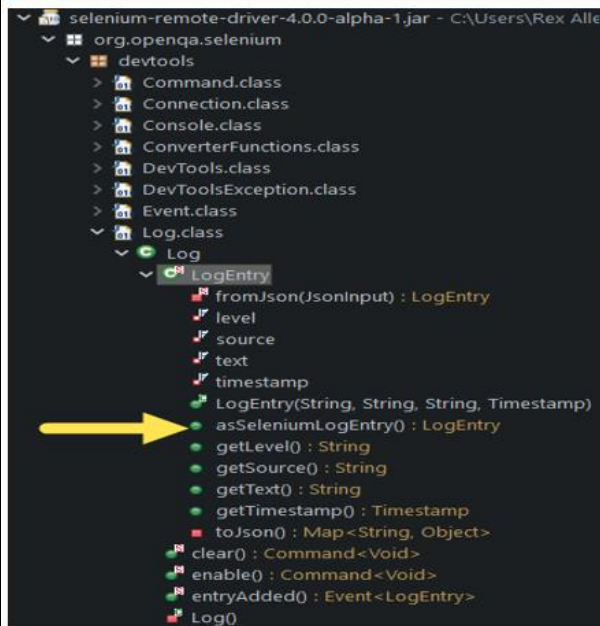
Let's print the log entries `sout(logEntry.asSeleniumLogEntry());` This is where I noticed a difference between Selenium 4 Alpha 1 and the current version. I changed my pom.xml file to use the version for Alpha . Alpha 1 has `asSeleniumLogEntry`.

In this screenshot, we see Alpha 6 does not have `asSeleniumLogEntry` for a method. Alpha 6 is not the most recent version. However, Alpha 6 and the most recent version added more methods although `asSeleniumLogEntry` is missing. Alpha 1 does not have methods `getArgs`, `getLineNumber`, `getNetworkRequestId`, `getStackTrace`, `getUri` and `getWorkerId`.

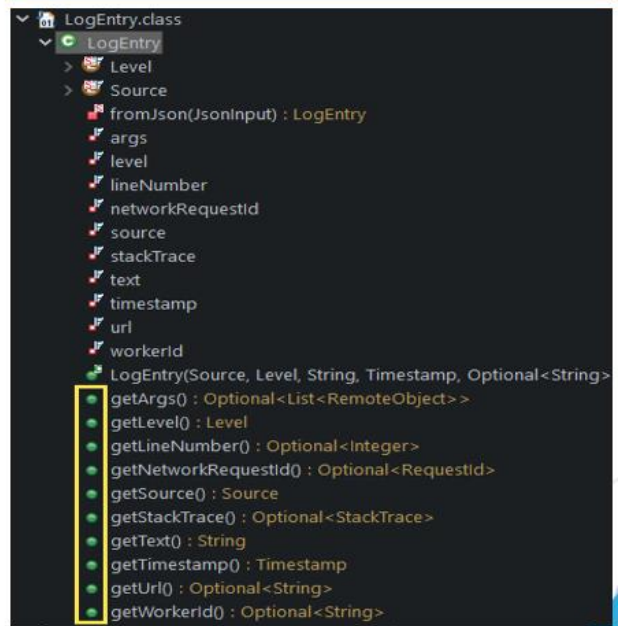


## Selenium 4 LogEntry Class

### Alpha 1



### Alpha 6



Here's my pom.xml file that shows Alpha 1.

```
<dependency>
  <groupId>org.seleniumhq.selenium</groupId>
  <artifactId>selenium-java</artifactId>
  <version>4.0.0-alpha-1</version>
</dependency>
```

I'm going to add some hyphens then print the next Log Entries. `sout("-----");`  
`sout("source = " + logEntry.getSource());`

Copy ("source = " + logEntry.getSource()) then get the level getLevel() and change source to level  
Change level to text and getLevel to getText then level to timestamp and getLevel to getTimestamp.

```
// Add A Listener For The Logs
devTools.addListener(Log.entryAdded(), logEntry -> {
    System.out.println(logEntry.asSeleniumLogEntry());
    System.out.println("-----");
    System.out.println("source = " + logEntry.getSource());
    System.out.println("level = " + logEntry.getLevel());
    System.out.println("text = " + logEntry.getText());
    System.out.println("timestamp = " + logEntry.getTimestamp());
});
```

Last but not least we are going to load the AUT to view the Console logs is driver.get(); The URL is <https://www.lego.com/404>. Let's Run. In the Console, we see some of the same information because that method asSeleniumLogEntry combines different log entries and we also printed individual log entries. For example, it has source as network, level = error, text = Failed to load resource and the timestamp.

```
Seen: {method=Log.entryAdded, params={entry={source=network, level=error,
text=Failed to load resource: the server responded with a status of 404 (),
timestamp=1.606829427178035E12, url=https://www.lego.com/en-us/404,
networkRequestId=62F0CB1492417E91F6660123A45C65ED}},
sessionId=180CE51405B2BC80F4ACCED20A964098}
[2020-12-01T07:30:27-0600] [SEVERE] Failed to load resource: the server responded
with a status of 404 ()
-----
source = network
level = error
text = Failed to load resource: the server responded with a status of 404 ()
timestamp = 1606829427178
```

That's it for capturing logs using Selenium 4 CDP and I'll see you in the next session. If you are interested in more videos, feel free to subscribe to my YouTube channel and click the bell icon. Also, follow me on Twitter, connect with me on LinkedIn. The transcript and code will get placed on GitHub.

## GeoLocation

In this session, I will show you how to mock a Geolocation using Selenium 4 CDP. Mock means imitate and Geolocation refers to the geographical location of a device connected to the internet. The device can be anything a phone, laptop, or even a watch. We use geolocation for events like finding the location of a place.

On [Selenium's site](#), we see Chrome DevTools, Emulate Geo Location, description and code to Emulate the Geo Location.

```
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.devtools.DevTools;

public void geolocationTest(){
    ChromeDriver driver = new ChromeDriver();
    Map coordinates = new HashMap()
    {{
        put("latitude", 50.2334);
        put("longitude", 0.2334);
        put("accuracy", 1);
    }};
    driver.executeCdpCommand("Emulation.setGeolocationOverride", coordinates);
    driver.get("<your site url>");
}
```

The description says “Some applications have different features and functionalities across different locations. But with the help of DevTools, we can easily emulate them.” Depending on the application, we can see different information because the application is in a different location. With this help of the Chrome DevTools Protocol, the application can have the same information after mocking the geolocation. We can mock, well this site says emulate. We can emulate the geolocation by making our browser be in the same location. As a result, the application will have the same features and functionalities we are trying to test. For example, I live in Texas but on my project, I work with people who live in India. If our application showed different information in India then I would change my location to be in India. For this test, I’m going to make my browser be located in the capital of India. In this pom.xml file, I changed my Selenium 4 version to alpha 7.

```
<dependency>
    <groupId>org.seleniumhq.selenium</groupId>
    <artifactId>selenium-java</artifactId>
    <version>4.0.0-alpha-7</version>
</dependency>
```

From scratch, I’m going to write @Test / public void mockGeoLocation () {} set up our browser by writing WebDriverManager.chromedriver().setup(); ChromeDriver is the type and driver = is the object

new ChromeDriver ();  
Maximize the window: driver.manage().window().maximize(); / driver.executeCdpCommand  
Notice, how executeCdpCommand has String commandName and Map for the parameters.

```
@Test
public void mockGeolocation () {
    WebDriverManager.chromedriver().setup();
    ChromeDriver driver = new ChromeDriver();
    driver.manage().window().maximize();
    driver.executeCdpCommand(
}
executeCdpCommand(String commandName, Map<String, Object>
```

In the sample code, the command is Emulation.setGeolocationOverride and coordinates for the Map. The coordinates are latitude, longitude, and accuracy. For more information, we can go to github. The url is <https://chromedevtools.github.io/devtools-protocol/>, search for Geo and we see Emulation.setGeolocationOverride with a description that says Overrides the Geolocation Position or Error. We see all 3 parameters: latitude, longitude, and accuracy.

Chrome DevTools Protocol
Geo

Emulation.clearGeolocationOverride

Clears the overridden Geolocation Position and Error.

Emulation.setGeolocationOverride

Overrides the Geolocation Position or Error. Omitting any of the parameters emulates position unavailable.

Page.clearGeolocationOverride

Clears the overridden Geolocation Position and Error.

Page.setGeolocationOverride

Overrides the Geolocation Position or Error. Omitting any of the parameters emulates position unavailable.

## Emulation.setGeolocationOverride #

Overrides the Geolocation Position or Error. Omitting any of the parameters emulates position unavailable.

### PARAMETERS

<b>latitude</b> optional	<b>number</b> Mock latitude
<b>longitude</b> optional	<b>number</b> Mock longitude
<b>accuracy</b> optional	<b>number</b> Mock accuracy

I'm going to set my geolocation to be at New Delhi. We see the coordinates are 28.6139 for latitude and 77.2090 for longitude. Latitude is always first. We see N for Latitude because the direction is North or South. E for Longitude because the direction is East or West. Our application under test is [Where Am I Right Now](#). I guess this site is accurate. The location is half a mile from my house. We see TX for Texas.

Now, let's go ahead and complete our script. 1<sup>st</sup> parameter is String commandName. We saw Emulation.setGeolocationOverride. The 2<sup>nd</sup> parameter was a Map and copy this code from Selenium's site and paste it to our code. Change latitude to 28.6139 and longitude is 77.2090, leave accuracy at 1. Now, I can pass in coordinates. Last step, is to load the Application Under Test:

```
driver.get("https://where-am-i.org/");
```

```
Map coordinates = new HashMap()
{{
    put("latitude", 28.6139);
    put("longitude", 77.2090);
    put("accuracy", 1);
}};
driver.executeCdpCommand(
    commandName: "Emulation.setGeolocationOverride", coordinates);
driver.get("https://where-am-i.org/");
```

Now, let's Run. We see New Delhi for Location. We see Latitude and Longitude. Also, here's the map.



### Your Location

Rajpath, Central Secretariat, New Delhi,  
110001, Delhi, India

### Your Latitude and Longitude

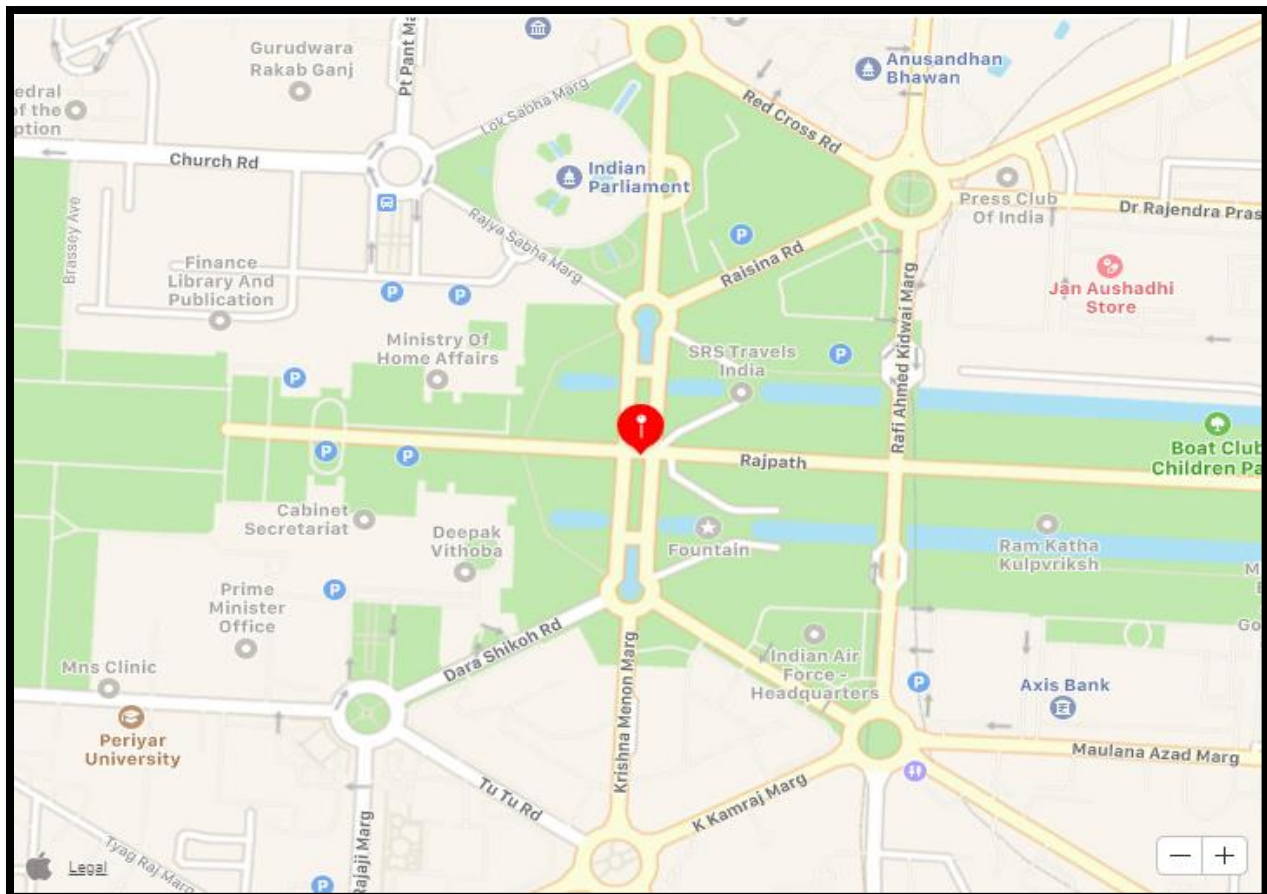
28.6139

77.209

### DMS (degrees, minutes, seconds)

N 28 ° 36 ' 50.04 "

E 77 ° 12 ' 32.4 "



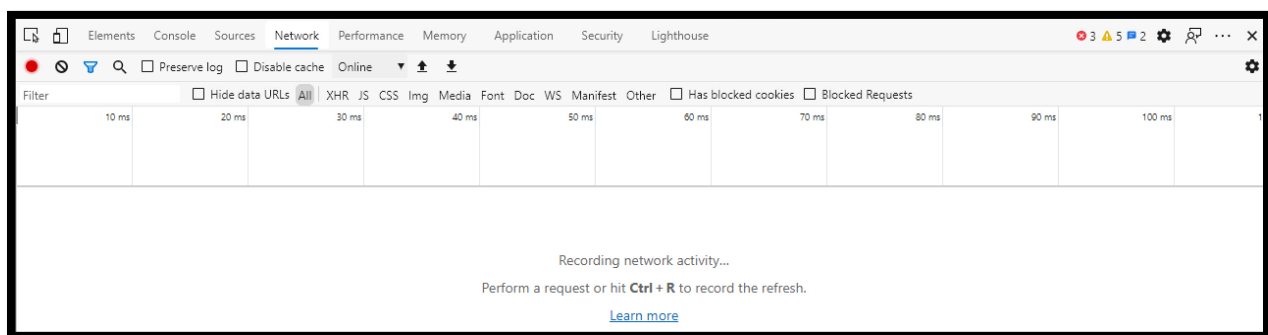
Thanks for watching and I'll see you in the next session for enabling the network.



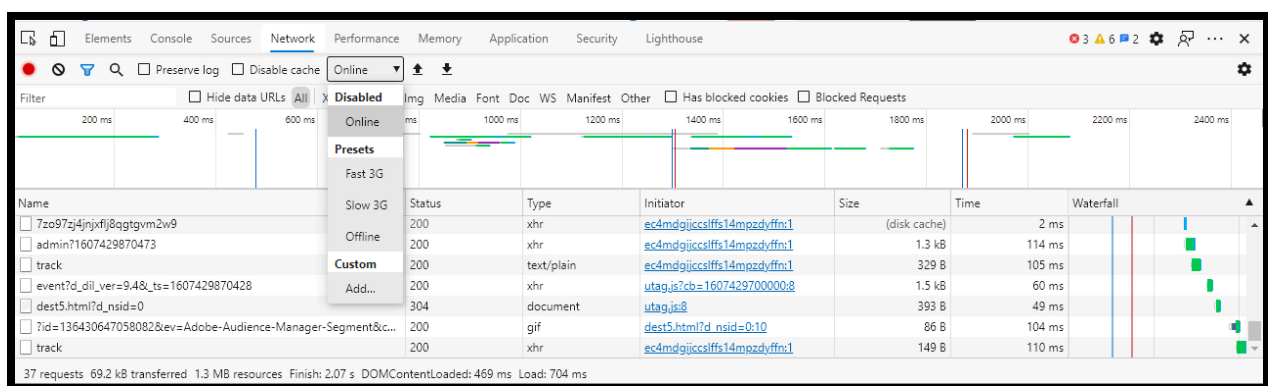
The Network panel is another way to debug a problem. It has logs to help us troubleshoot the problem. We mostly use the panel for 2 reasons. One reason is to inspect the network properties of a resource. Another reason is to make sure our resources are downloaded and uploaded as expected. When testing an application, it's easy to forget or not think about a user with a weak connection. Therefore, we are going to automate how to slow down our internet connection.

## Slow Down Internet Connection

In the Network panel, there's no available request. It's empty because we must open the panel before performing an action. That's how we preserve the network traffic data.



Refresh the page and all of the network activity shows up in the Network Log. Each row in the Network Log represents a resource. Do you see 69.2 kB transferred? That's the total download size. This Network Throttle dropdown is how we emulate different connection speeds. We see Fast 3G, Slow 3G, and Offline.



In this session, I will automate a 3G connection but next session I will take the network offline. Select any preset and the DevTools display a warning icon beside the Network tab.

For our Test Script, I have the setup method for EdgeDriver. Let me also add `driver.getDevTools()` assign it to `devTools =` . Also write `DevTools devTools; up top.`

```
public class EnableNetworks {  
  
    EdgeDriver driver;  
    DevTools devTools;  
  
    @BeforeMethod  
    public void setUp () {  
        WebDriverManager.edgedriver().setup();  
        driver = new EdgeDriver();  
        driver.manage().window().maximize();  
        devTools = driver.getDevTools();  
    }  
}
```

Enable the network to slow down by writing. `@Test public void enableSlowNetwork () { }` Start by creating a session `devTools.createSession()`; then we send a command to enable the network by writing `devTools.send(Network.enable(Optional.empty(), let's write this statement 2 more times. This complete statement makes it possible to deliver network tracking and events to the client. All of the parameters are Optional.empty because they are not required. In fact, the first 2 parameters are experiments.`

Next, is to emulate the Network Conditions. We can go to [github](#) for the method and parameters. Search for emulate. At the bottom is `Network.emulateNetworkConditions`. It activates emulation of network condition.

## Emulation.canEmulate

Tells whether emulation is supported.

## Emulation.setEmulatedMedia

Emulates the given media type or media feature for CSS media queries.

## Emulation.setEmulatedVisionDeficiency

Emulates the given vision deficiency.

## Input.emulateTouchFromMouseEvent

Emulates touch event from the mouse event parameters.

## Network.canEmulateNetworkConditions

Tells whether emulation of network conditions is supported.

## Network.emulateNetworkConditions

Activates emulation of network conditions.

The parameters are offline, latency, downloadThroughput, uploadThroughput, and connectionType. Connection Type is the only optional parameter.

### Network.emulateNetworkConditions #

Activates emulation of network conditions.

#### PARAMETERS

offline	<b>boolean</b>	True to emulate internet disconnection.
latency	<b>number</b>	Minimum latency from request sent to response headers received (ms).
downloadThroughput	<b>number</b>	Maximal aggregated download throughput (bytes/sec). -1 disables download throttling.
uploadThroughput	<b>number</b>	Maximal aggregated upload throughput (bytes/sec). -1 disables upload throttling.
connectionType	<b>ConnectionType</b>	Connection type if known.
optional		

The method we wrote for enabling the network is right here and we see all 3 parameters are optional: maxTotalBufferSize, maxResourceBufferSize, and maxPostDataSize. If I hover Experimental, the tool tip says "This may be changed, moved, or removed".

**Network.enable #**

Enables network tracking, network events will now be delivered to the client.

PARAMETERS

<b>maxTotalBufferSize</b> optional	<b>integer</b> Buffer size in bytes to use when preserving network payloads (XHRs, etc). <b>EXPERIMENTAL</b>
<b>maxResourceBufferSize</b> optional	<b>integer</b> Per-resource buffer size in bytes to use when preserving network payloads (XHRs, etc). <b>EXPERIMENTAL</b> <small>This may be changed, moved or removed</small>
<b>maxPostDataSize</b> optional	<b>integer</b> Longest post body size (in bytes) that would be included in requestWillBeSent notification

For our Test Script, let's complete it by writing `devTools.send(Network.emulateNetworkConditions())`. We want the network to stay online so we set `offline` to `false`, `latency` is 150, `downloadThroughput` is 2500, `uploadThroughput` is 2000. Next, is the Connection Type. `Optional.of(ConnectionType.)` We see different choices. We have `BLUETOOTH`, `2G`, `3G`, `4G`, and `WIFI`. Let's go ahead and select `3G`.

Let's also compare the time between this slow network and the normal way by loading an application. Know what before we compare, we must load the application for LinkedIn:  
`driver.get("https://www.linkedin.com")`. Let's also print the page title. `sout("Slow " + driver.getTitle())`.

```
@Test
public void enableSlowNetwork () {
    devTools.createSession();
    devTools.send(Network.enable(
        Optional.empty(),
        Optional.empty(),
        Optional.empty()));
    devTools.send(Network.emulateNetworkConditions(
        offline: false,
        latency: 150,
        downloadThroughput: 2500,
        uploadThroughput: 2000,
        Optional.of(ConnectionType.CELLULAR3G)));
    driver.get("https://www.linkedin.com");
    System.out.println("Slow " + driver.getTitle());
}
```

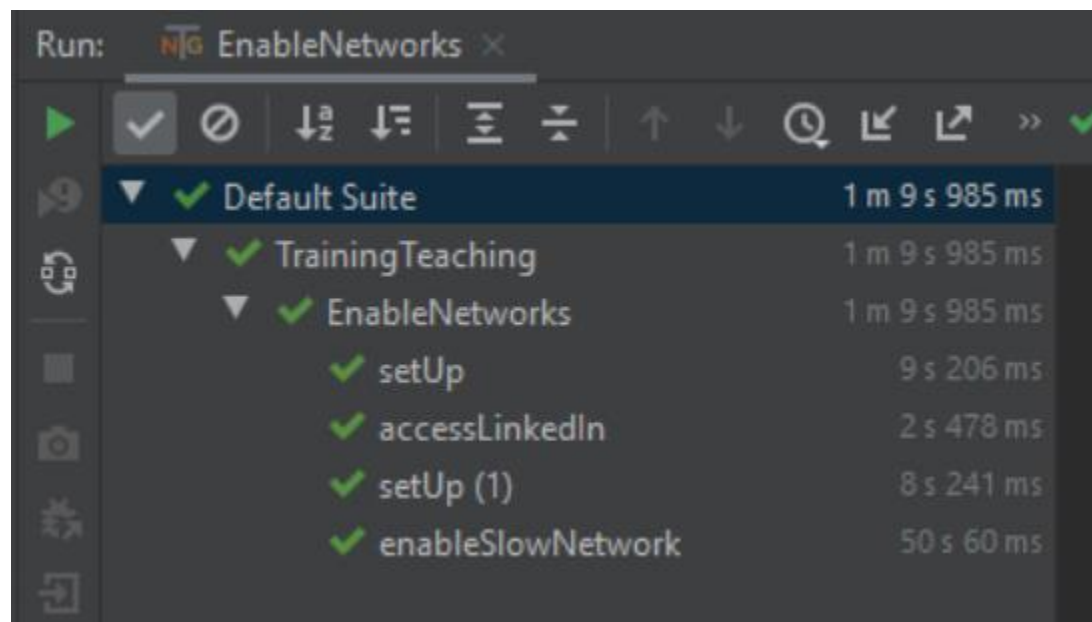
Now, let's go ahead and load LinkedIn the normal way.

```
@Test public void accessLinkedIn () { } driver.get("https://www.linkedin.com"); Also sout("Access " + driver.getTitle());
```

```
@Test
public void accessLinkedIn () {
    driver.get("https://www.linkedin.com");
    System.out.println("Access " + driver.getTitle());
}
```

Let's go ahead and run.

The Console shows accessLinkedIn loaded in 2 Seconds and 478 Milliseconds. enableSlowNetwork took longer to complete. It finished in 50 Seconds and 60 Milliseconds. Both Test Scripts show the Page Title. That's it for slowing down the Network to a slower connection.



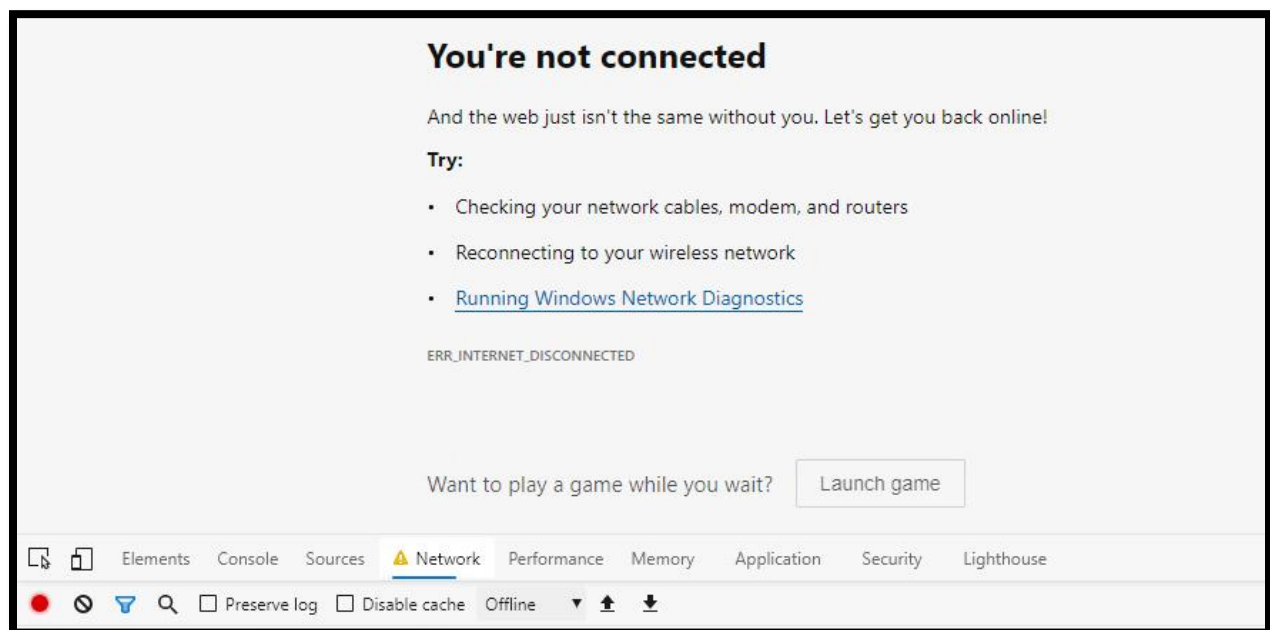
Run:	EnableNetworks
▶	✓
▼	✓
▼	✓
▼	✓
✓	setUp
✓	accessLinkedIn
✓	setUp (1)
✓	enableSlowNetwork

Next, I'm going to show you have to take the Network offline. If you are interested in more videos, consider subscribing to my [YouTube](#) channel and clicking the bell icon. Also, follow me on [Twitter](#), connect with me on [LinkedIn](#) and [Facebook](#). The transcript and code will get placed on [GitHub](#). Thanks for watching and I'll see you in the next session.

## Emulate Offline

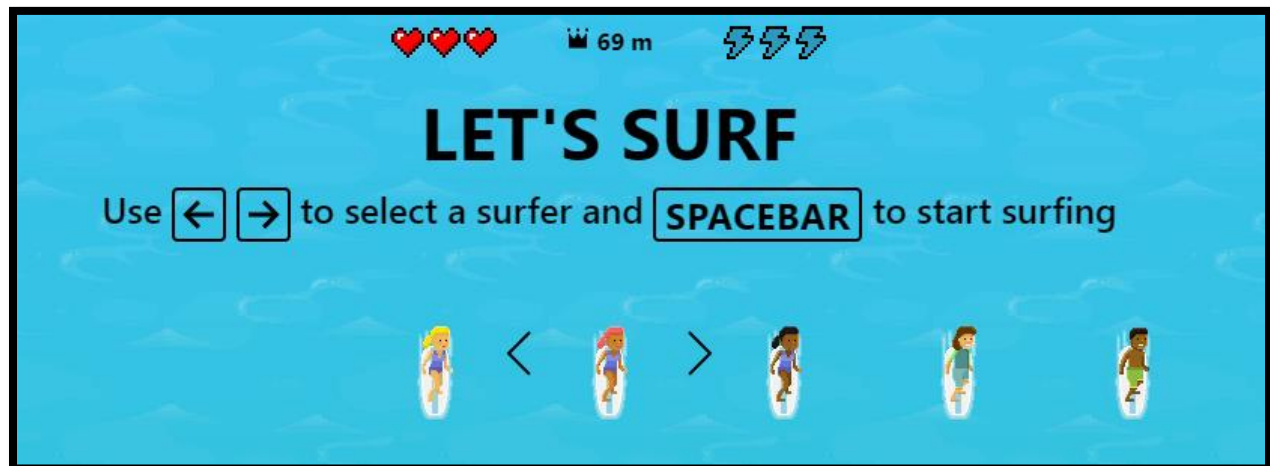
In this session, we are going to enable the network to be offline. It's very good when an application has functionality to continue working offline. There are many reasons for an application to be offline such as bad weather, a power outage, or even driving into a dead zone.

Here's our Application Under Test. Most of the times, I use Chrome but not this time. For some reason Chrome does not include the same offline functionality as Microsoft Edge. I'm going to right click, Inspect, and go to the Network tab. The Network Throttling drop down has different options. We want Offline. Refresh and notice the message says "You're not connected" and it also shows INTERNET\_DISCONNECTED. A lot of applications stop right here and do not let us continue. However, we can still perform an action while offline. Do you see Want to play a game while you wait and this Launch game button?



Inspect this launch game button and we see the id value is game-button. Now, at this point, we are going to do the same steps in our automation. In our test, we are going to set the test to be offline then click this Launch game button. It's a surf game.





For our Test Script, I already have EdgeDriver, DevTools, and the setup method for EdgeDriver. It's from the last session when we slowed down the network connection using 3G and accessed LinkedIn.

Now, let's write `@Test public void enableOfflineNetwork () {}`. We always begin by creating the session. `devTools.createSession()`. This will start a session in the Edge browser. Next, is to send a command so we can enable the network. `devTools.send(Network.enable())` The enable method has 3 parameters and all 3 of them are optional. Therefore, we write `Optional.empty(), Optional.empty(), Optional.empty()`.

```
@Test
public void enableOfflineNetwork () {
    devTools.createSession();
    devTools.send(Network.enable(
        Optional.empty(),
        Optional.empty(),
        Optional.empty()));
}
```

If I hover enable, the description shows "Enables network tracking, network events will now be delivered to the client". Those 3 empty parameters are `maxTotalBufferSize`, `maxResourceBufferSize`, and `maxPostDataSize`.

```
org.openqa.selenium.devtools.network.Network
@NotNull
@Contract("_,_,_->new")
public static org.openqa.selenium.devtools.Command<Void> enable(@NotNull
                                                                    @NotNull
                                                                    @NotNull
```

Enables network tracking, network events will now be delivered to the client.

Inferred annotations: Method enable: @org.jetbrains.annotations.NotNull  
 @org.jetbrains.annotations.Contract("\_,\_,\_->new")  
 Parameter maxTotalBufferSize: @org.jetbrains.annotations.NotNull  
 Parameter maxResourceBufferSize: @org.jetbrains.annotations.NotNull  
 Parameter maxPostDataSize: @org.jetbrains.annotations.NotNull

After enabling the network, we are going to send a command to emulate the network.  
 devTools.send(Network.emulateNetworkConditions()). Hover the method to see it activates an  
 emulation of the network conditions using 5 parameters: offline, latency, downloadThroughput,  
 uploadThroughput and connectionType. I don't see it here but Connection Type is optional.

```
org.openqa.selenium.devtools.network.Network
@NotNull
@Contract("_,_,_,_,_->new")
public static org.openqa.selenium.devtools.Command<Void> emulateNetworkCondition
```

Activates emulation of network conditions.

Inferred annotations: Method emulateNetworkConditions: @org.jetbrains.annotations.NotNull  
 @org.jetbrains.annotations.Contract("\_,\_,\_,\_,\_->new")  
 Parameter offline: @org.jetbrains.annotations.NotNull  
 Parameter latency: @org.jetbrains.annotations.NotNull  
 Parameter downloadThroughput: @org.jetbrains.annotations.NotNull  
 Parameter uploadThroughput: @org.jetbrains.annotations.NotNull  
 Parameter connectionType: @org.jetbrains.annotations.NotNull

Now, let's go ahead and continue with our test script by writing the value for offline and that value will  
 be true. This means yes, the network will be offline. 10 is the value for latency. Latency is sometimes  
 called lag. It's a delay in communication across the network. 100 for the downloadThroughput and 50  
 for the uploadThroughput. Throughput refers to the amount of data that transfers from the source to

the destination. For connection type, we write `Optional.of(ConnectionType.G)` and we see the different G connections but let's select WIFI.

```
devTools.send(Network.emulateNetworkConditions(
    offline: true,
    latency: 10,
    downloadThroughput: 100,
    uploadThroughput: 50,
    Optional.of(ConnectionType.WIFI)));
```

Now, let's add a listener and what we expect. `devTools.addListener()` At this point, I'm going to write some data inside the `addListener` some parameters and that starts with `(loadingFailed(),)` and we are going to select the example with lambda expression because there are 2 options. Now, we write `assertEquals()` and inside `assertEquals` we are going to write `loadingFailed` and select the one that do not have parenthesis dot `getErrorMessage()`. Remember when we saw Internet Disconnected after setting the network to offline then refreshing the page. That's what we expect `"net::ERR_INTERNET_DISCONNECTED")");`

```
devTools.addListener(loadingFailed(),
    loadingFailed -> assertEquals(loadingFailed.getErrorMessage(),
    expected: "net::ERR_INTERNET_DISCONNECTED"));
```

The last step is to load the page. Since it's offline, let's try { } to load the page by writing `driver.get("https://www.google.com")`.

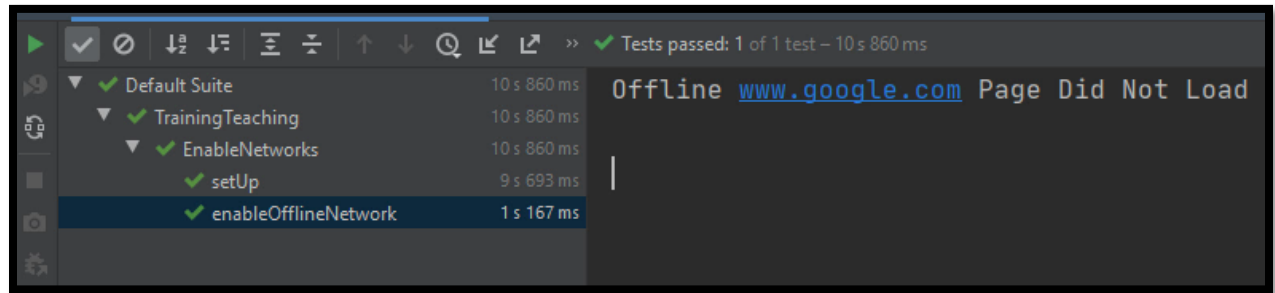
Next, is to catch ( ) the possible exception (`WebDriverException exc`) { } and click the button by writing `driver.find`. Do you see the extra `findElementBy` options? We have these options because our type is `EdgeDriver`. It's been available to us for a long time. The same with `ChromeDriver` and `FirefoxDriver`. However, they supposed to be deprecated for Selenium 4.

```
m findElement(By by)
m findElementByClassName(String using)
m findElementByCssSelector(String using)
m findElementById(String using)
m findElementByLinkText(String using)
m findElementByName(String using)
m findElementByPartialLinkText(String using)
m findElementByTagName(String using)
m findElementByXPath(String using)
m findElements(By by)
m findElementsByClassName(String using)
```

I spoke about this in the Introduction to Selenium 4. The first video. I'm going to select `driver.findElement(By.id("game-button")).click()`. Also, print the page title: `sout("Offline " + driver.getTitle() + "Page Did Not Load" );`

```
try {
    driver.get("https://www.google.com");
} catch (WebDriverException exc) {
    driver.findElement(By.id("game-button")).click();
    System.out.println("Offline " + driver.getTitle() + " Page Did Not Load");
}
```

That's it and let's run. It passed and we see the game. Now, for our Console, let's see what it shows `enableOfflineNetwork`. We see it shows Offline [www.google.com](https://www.google.com) and the Page Did Not Load.

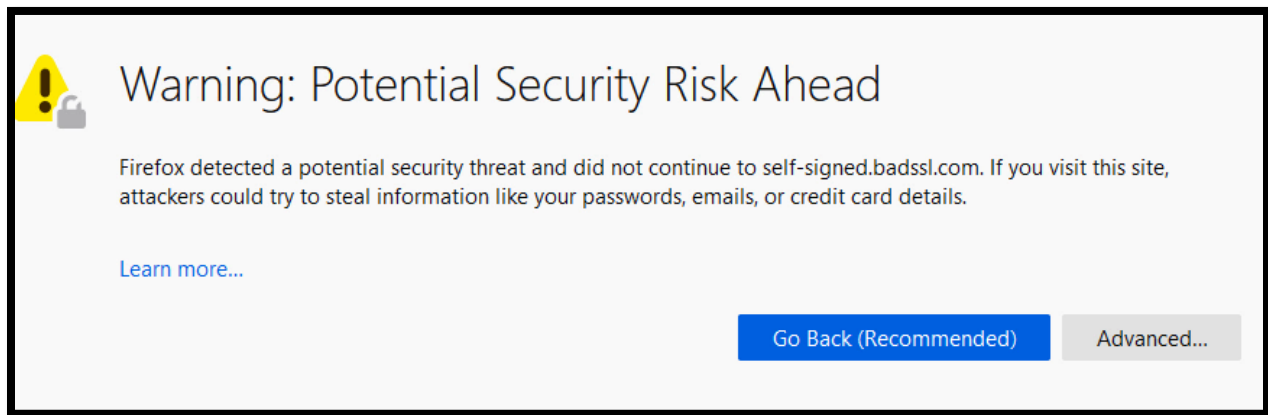


Thank You for watching and I'll see you in the next session. I create videos every week and if you are interested in videos that I create, feel free to subscribe to my [YouTube](#) channel and click the bell icon. Also follow me on [Twitter](#), connect with me on [LinkedIn](#) and [Facebook](#). I will make sure to place the transcript and code on [GitHub](#).

## Certificate Error Website

In this session, our focus will involve loading an untrusted website. Depending on your browser, there will be a different message but it all comes down a security issue. It's recommended we do not visit the site. However, there may be a scenario for us to test this type of application.

On [Firefox](#), the message says "Warning: Potential Security Risk Ahead". It's letting us know that an attacker could try to steal our information.



[Chrome](#) has a similar message about attacker might be trying to steal your information. With a heading that says "Your connection is not private". [Edge](#) has the same message as Chrome.





## Your connection is not private

Attackers might be trying to steal your information from **self-signed.badssl.com** (for example, passwords, messages, or credit cards). [Learn more](#)

NET::ERR\_CERT\_AUTHORITY\_INVALID

- ☐ Help improve security on the web for everyone by sending [URLs of some pages you visit, limited system information, and some page content](#) to Google. [Privacy policy](#)

Advanced

Back to safety



## Your connection isn't private

Attackers might be trying to steal your information from **self-signed.badssl.com** (for example, passwords, messages, or credit cards).

NET::ERR\_CERT\_AUTHORITY\_INVALID

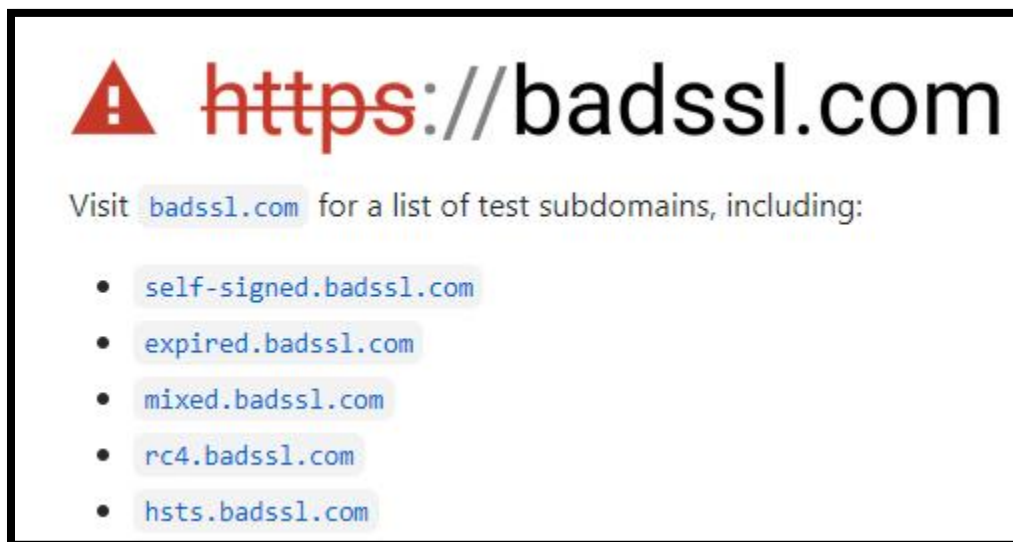
Advanced

Go back

Our automation Test Script will ignore these security warnings and load the page. If I click the Advanced button, we see a link that says “Proceed to the self-signed.badssl.com (unsafe)”. Click this link and the self-signed.badssl.com site load without a problem.



[GitHub](#) has more example sites just like this one. We see 5 sites that have subdomains that generate the same type of warning



For our Test, its already been set up for ChromeDriver and DevTools. The last statement assigns getDevTools to devTools.

```
public class UntrustedWebsite {

    ChromeDriver driver;
    DevTools devTools;

    @BeforeMethod
    public void setUp () {
        WebDriverManager.chromedriver().setup();
        driver = new ChromeDriver();
        driver.manage().window().maximize();
        devTools = driver.getDevTools();
    }
}
```

Let's start by creating a method with @Test public void loadBadWebsite () { } Next, we create a session with devTools.createSession(). Now, we are going to send a command to ignore the certificate error.

devTools.send(Security.) Security provides access for us to setIgnoreCertificateErrors(). This statement is similar to clicking the Advanced button and the unsafe hyperlink. We pass in a boolean value of (true). True is another way of saying we trust the website. So, it will successfully load the page. That's it, last part is to load the AUT. driver.get("https://self-signed.badssl.com/");

```
@Test
public void loadBadWebsite () {
    devTools.createSession();
    devTools.send(Security.setIgnoreCertificateErrors(true));
    driver.get("https://self-signed.badssl.com");
}
```

The key is ignoring the Certificate Error by passing in true. A negative Test Case is to pass in false which will not load the untrusted website. Now, I'm going to paste the code for what we just used to ignore and change true to false. Change the name from loadBadWebsite to doNotLoadBadWebsite.

```
●@Test
public void doNotLoadBadWebsite () {
    devTools.createSession();
    devTools.send(Security.setIgnoreCertificateErrors(false));
    driver.get("https://self-signed.badssl.com");
}
```

Let's Run. They both Passed. One browser shows self-signed.badssl.com and the other browser shows Your connection is not private. That's it.

Thanks for watching and I'll see you in the next session. If you are interested in more videos, feel free to subscribe to my [YouTube channel](#) and click the bell icon. Also, follow me on [Twitter](#), connect with me on [LinkedIn](#), and [Facebook](#). The transcript and code will get placed on [GitHub](#).

## Contact Info

- ✓ YouTube <https://www.youtube.com/c/RexJonesII/videos>
- ✓ Facebook <https://facebook.com/JonesRexII>
- ✓ Twitter <https://twitter.com/RexJonesII>
- ✓ GitHub <https://github.com/RexJonesII/Free-Videos>
- ✓ LinkedIn <https://www.linkedin.com/in/rexjones34/>