

In the article about [Actions Class in Selenium](#), we discussed that Selenium provides a feature to handle keyboard events where user gestures can emulate. For example, suppose you search **"India"** on google search engine. For this, you will type **"India "** and press the Enter key, so that you see the result queries. Say, you want to automate this google search using the Selenium test, ***how will you simulate pressing the ENTER key using selenium code?*** The provision of the capability to simulate such keyboard actions is by the ***keyboard events generated using the Actions class of Selenium WebDriver***. In this tutorial, we will cover the details of all the Keyboard events supported by Selenium. Additionally, we will also see how Actions class fit for the need of simulating the keyboard events in Selenium.

What are keyboard events in Selenium?

Why is Actions class needed to perform Keyboard actions using Selenium WebDriver?

What is Actions Class in Selenium WebDriver?

What are the different methods provided by the Actions class for Keyboard Events?

How to handle contiguous Keyboard Actions using Actions Class?

What are keyboard events in Selenium?

A Keyboard Event describes a user's interaction with the keyboard. When a user presses single or multiple keys, keyboard events generate. Selenium provides various ways to automate these Keyboard Events, a few of which are:

[Automate keyboard events using the sendKeys\(\) method of WebElement class.](#)

[Automate keyboard events using Robot class.](#)

[And Automate keyboard events using Actions class.](#)

We already discussed the first two ways of handling the keyboard events using the ***"sendKeys()"*** method of ***WebDriver's WebElement class*** and ***"Robot class"*** in the articles given by corresponding links. In this article, we will specifically cover the details of the ***"Actions"*** class in ***Selenium WebDriver***. Before going deep to understand the concepts of the ***"Actions"*** class, let's first understand why specifically ***"Actions"*** class is needed to handle those Keyboard Events?

Why is Actions class needed to perform Keyboard actions using Selenium WebDriver?

When we interact with a web application, there will be various scenarios when the user performs the following actions:

Type in capital/Camel case letters: Wherever user need to type a word or letter in caps, he/she will press the **"SHIFT"** key and will type the necessary characters, and whatever characters will type while pressing the **"SHIFT"** key, will type as a capital letter.

Copy & Paste Text: When we need to copy some text from one text box to another, we select the text by pressing **"CTRL+A"** they copy the text using **"CTRL+C"** and paste the text in the new text box by simply clicking in the text box and pressing keys **"CTRL+V"**.

These are very common user actions, which we perform on an almost daily basis. Now, as we discussed, **Selenium WebDriver** provides two ways to send any keyboard event to a web element:

*sendKeys() method of WebElement class.
Actions class*

Now let's try to understand in detail that if we want to automate the scenario as mentioned above of typing the letters in the capital (*with SHIFT key pressed*), using the **sendKeys()** method of the **WebElement** class.

Consider the following scenario for quickly understanding the behavior:

*First, navigate to **"https://demoqa.com/text-box."***

*Secondly, enter the Full name: **"Mr.Peter Haynes"**.*

*Thirdly, enter the Email: **"PeterHaynes@toolsqa.com."***

*After that, Enter the Current Address: **"43 School Lane London EC71 9GO"**.*

Fifthly, click on the Current Address text box and Copy the Current Address.

After that, paste the copied address in the Permanent Address text box.

Finally, validate that the text in the Current Address and Permanent Address is the same.

Let's try to automate the above scenario using the **sendKeys()** method of the **WebElement** class:

```
package automationFramework;

import static org.junit.Assert.assertEquals;

import org.openqa.selenium.By;
import org.openqa.selenium.Keys;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;

public class KeyboardEvents {

    public static void main(String[] args) {

        // Initialize ChromeDriver
        // Here we assume that the ChromeDriver path has been set in the System Global va
        WebDriver driver=new ChromeDriver();

        //Navigate to the demo site
        driver.get("https://demoqa.com/text-box");

        // Enter the Full Name
        WebElement fullName = driver.findElement(By.id("userName"));
```

```
fullName.sendKeys("Mr.Peter Haynes");

//Enter the Email
WebElement email=driver.findElement(By.id("userEmail"));
email.sendKeys("PeterHaynes@toolsqa.com");

// Enter the Current Address
WebElement currentAddress=driver.findElement(By.id("currentAddress"));
currentAddress.sendKeys("43 School Lane London EC71 9G0");

// Copy the Current Address
currentAddress.sendKeys(Keys.CONTROL);
currentAddress.sendKeys("A");
currentAddress.sendKeys(Keys.CONTROL);
currentAddress.sendKeys("C");

//Press the TAB Key to Switch Focus to Permanent Address
currentAddress.sendKeys(Keys.TAB);

//Paste the Address in the Permanent Address field
WebElement permanentAddress=driver.findElement(By.id("permanentAddress"));
permanentAddress.sendKeys(Keys.CONTROL);
permanentAddress.sendKeys("V");

//Compare Text of current Address and Permanent Address
assertEquals(currentAddress.getAttribute("value"),permanentAddress.getAttribute("value"));

driver.close();

}
```

In the above code snippet, though the **sendKeys()** method of *WebElement* allows using the **Control Key**, it cannot do the copy and paste action, as it fails to combine the key sequences.

As we can see from the following screenshot, instead of pasting the content of Current Address in the Permanent Address text field, it just pasted the character **"V"**.

← → ↻ demoqa.com/text-box

Chrome is being controlled by automated test software.

TOOLSQA

Text Box

≡ Elements

↑

Text Box

Check Box

Radio Button

Web Tables

Buttons

Links

Full Name

Mr.Peter Haynes

Email

PeterHaynes@toolsqa.com

Current Address

43 School Lane London EC71 9GOAC

Permanent Address

V|

Submit

As is clear from the above screenshot, that the text of the Current Address text field did not copy to the Permanent Address text field, so it leads to failure of the assertion when the comparison of the text of these two fields happens. So, when we execute the above test, it fails with the following error message:

```
Exception in thread "main" org.junit.ComparisonFailure: expected:<[43 School Lane London EC71 9GOAC]> but was:<[V]>  
    at org.junit.Assert.assertEquals(Assert.java:117)  
    at org.junit.Assert.assertEquals(Assert.java:146)  
    at automationFramework.KeyboardEvents.main(KeyboardEvents.java:54)
```

So, this is where the **sendKeys() method of the WebElement** class fails. In other words, it fails when we need to combine special keys such as **"SHIFT"**, **"CONTROL"**, etc. with the different key sequences, which we all know, is a prevalent scenario when we as a user are using any of the web applications.

Therefore, this is where the **Actions class of Selenium WebDriver** comes into the picture, which provides various methods to specifically handle operations of these meta keyboard keys, which

need to press while performing operations on the other keyboard keys. Let's see how we can handle such keyboard actions, using the *Actions class of Selenium WebDriver*.

What is Actions Class in Selenium WebDriver?

As we discussed above, ***Selenium WebDriver*** provides a class named ***"Actions"***, which provides various methods that can help in automating and simulating the ***Keyboard and Mouse actions***. The below figure shows the exhaustive list of methods offered by *Selenium WebDriver*, and the highlighted ones are the most used methods for simulating the Keyboard actions:


```

23 //Create object of the Actions class
24 Actions actions = new Actions(driver);
25 actions.
26 ● build() : Action - Actions
27 ● click() : Actions - Actions
28 ● click(WebElement target) : Actions - Actions
29 ● clickAndHold() : Actions - Actions
30 ● clickAndHold(WebElement target) : Actions - Actions
31 ● contextClick() : Actions - Actions
32 ● contextClick(WebElement target) : Actions - Actions
33 ● doubleClick() : Actions - Actions
34 ● doubleClick(WebElement target) : Actions - Actions
35 ● dragAndDrop(WebElement source, WebElement target) : Actions - Actions
36 ● dragAndDropBy(WebElement source, int xOffset, int yOffset) : Actions - Actions
37 ● equals(Object obj) : boolean - Object
38 ● getClass() : Class<?> - Object
39 ● hashCode() : int - Object
40 ● keyDown(CharSequence key) : Actions - Actions
41 ● keyDown(WebElement target, CharSequence key) : Actions - Actions
42 ● keyUp(CharSequence key) : Actions - Actions
43 ● keyUp(WebElement target, CharSequence key) : Actions - Actions
44 ● moveByOffset(int xOffset, int yOffset) : Actions - Actions
45 ● moveToElement(WebElement target) : Actions - Actions
46 ● moveToElement(WebElement target, int xOffset, int yOffset) : Actions - Actions
47 ● notify() : void - Object
48 ● notifyAll() : void - Object
49 ● pause(Duration duration) : Actions - Actions
50 ● pause(long pause) : Actions - Actions
51 ● perform() : void - Actions
52 ● release() : Actions - Actions
53 ● release(WebElement target) : Actions - Actions
54 ● sendKeys(CharSequence... keys) : Actions - Actions
55 ● sendKeys(WebElement target, CharSequence... keys) : Actions - Actions
56 ● tick(Action action) : Actions - Actions
57 ● tick(Interaction... actions) : Actions - Actions
58 ● toString() : String - Object
59 ● wait() : void - Object
60 ● wait(long timeout) : void - Object
61 ● wait(long timeout, int nanos) : void - Object
62 }
63 }
64 }
65 }
66 }
67 }

```

Console

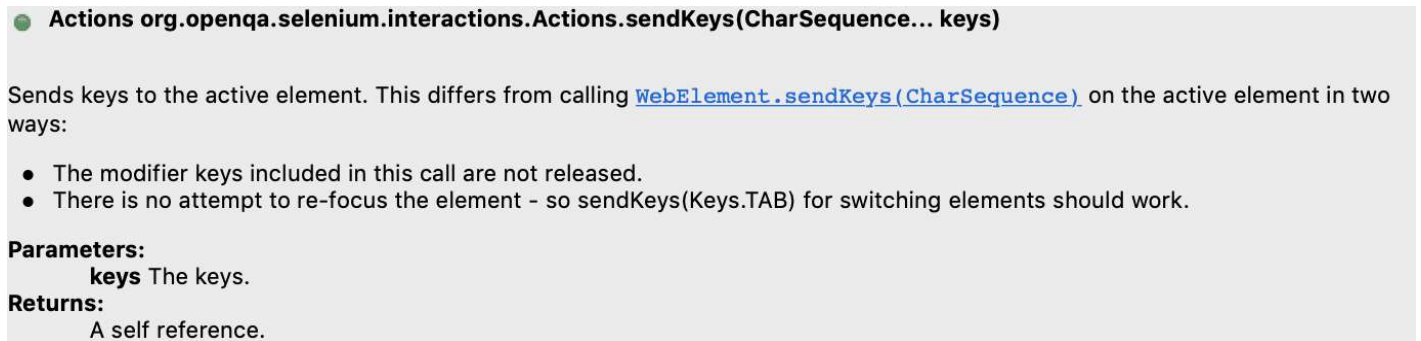
Let's understand the Keyboard specific methods provided by the **Actions** Class:

What are the different methods provided by the Actions class for Keyboard Events?

As highlighted in the above screenshot, the Actions class majorly provide the following three methods for simulating the Keyboard events:

sendKeys(): This method sends a series of keystrokes to a given web element. This method has two overloaded formats:

sendKeys(CharSequence... KeysToSend): The following screenshot shows the syntactical details of this method:



Actions org.openqa.selenium.interactions.Actions.sendKeys(CharSequence... keys)

Sends keys to the active element. This differs from calling [WebElement.sendKeys\(CharSequence\)](#) on the active element in two ways:

- The modifier keys included in this call are not released.
- There is no attempt to re-focus the element - so sendKeys(Keys.TAB) for switching elements should work.

Parameters:
 keys The keys.

Returns:
 A self reference.

This method sends a sequence of keys to a currently focused web element, i.e., if we want to send specific characters to a web element, that element must be first focussed, then only the mentioned characters will go to that web element.

sendKeys(WebElement element, CharSequence... KeysToSend): The following screenshot shows the syntactical details of this method:



Actions org.openqa.selenium.interactions.Actions.sendKeys(WebElement target, CharSequence... keys)

Equivalent to calling: `Actions.click(element).sendKeys(keysToSend)`. This method is different from [WebElement.sendKeys\(CharSequence\)](#) - see [sendKeys\(CharSequence\)](#) for details how.

Parameters:
 target element to focus on.
 keys The keys.

Returns:
 A self reference.

Throws:
 [IllegalArgumentException](#) - if keys is null

This implementation of the sendKeys() method sends a sequence of characters/keys to a specific web element, which passes as the first parameter to the method. This method first focuses on the target web element and then performs the same action as sendKeys(CharSequence keys).

keyDown(): This method simulates a keyboard action when a specific keyboard key needs to press. So, whenever you need to press a key and then perform specific other actions, we can use the keyDown() method to keep the key pressed. E.g., say a user has to type some characters in Capital. Then to simulate user behavior, where the user presses the **SHIFT** key and then presses the set of characters that need to type in Capital. This method is also available in the following two overloaded variants:

keyDown(CharSequence key): The following screenshot shows the syntactical details of this method:

● Actions org.openqa.selenium.interactions.Actions.keyDown(CharSequence key)

Performs a modifier key press. Does not release the modifier key - subsequent interactions may assume it's kept pressed. Note that the modifier key is **never** released implicitly - either `keyUp(theKey)` or `sendKeys(Keys.NULL)` must be called to release the modifier.

Parameters:

key Either [Keys.SHIFT](#), [Keys.ALT](#) or [Keys.CONTROL](#). If the provided key is none of those, [IllegalArgumentException](#) is thrown.

Returns:

A self reference.

*This method presses the specified key on the currently focussed Web Element. This method generally presses the "**Modifier keys**" such as SHIFT, CTRL, etc. If you want to press the keyboard key on a specified web element, then that web element first needs to be focussed explicitly, and then this method needs to be invoked.*

keyDown(WebElement element, CharSequence key): The following screenshot shows the syntactical details of this method:

● Actions org.openqa.selenium.interactions.Actions.keyDown(WebElement target, CharSequence key)

Performs a modifier key press after focusing on an element. Equivalent to: `Actions.click(element).sendKeys(theKey);`

Parameters:

key Either [Keys.SHIFT](#), [Keys.ALT](#) or [Keys.CONTROL](#). If the provided key is none of those, [IllegalArgumentException](#) is thrown.

target WebElement to perform the action

Returns:

A self reference.

This method first focusses on the web element, which has been passed as a parameter to the method and presses the mentioned key on that Web Element.

. **keyUp()**: We use this method majorly in collaboration with the **keyDown()** method. The keyboard key which presses using the **keyDown()** method, doesn't get released automatically, so the same need to be explicitly released using the **keyUp()** method. So, similar to the **keyDown()** method, this method has two overloaded variants:

keyUp(CharSequence key): The following screenshot shows the syntactical details of this method:

● Actions org.openqa.selenium.interactions.Actions.keyUp(CharSequence key)

Performs a modifier key release. Releasing a non-depressed modifier key will yield undefined behaviour.

Parameters:

key Either [Keys.SHIFT](#), [Keys.ALT](#) or [Keys.CONTROL](#).

Returns:

A self reference.

This method releases the specified key on the currently focussed Web Element. If you want to release the keyboard key on a specified web element, then that web element first needs to be focussed explicitly, and then this method needs to be invoked.

keyUp(WebElement element, CharSequence key): The following screenshot shows the syntactical details of this method:

● **Actions org.openqa.selenium.interactions.Actions.keyUp(WebElement target, CharSequence key)**

Performs a modifier key release after focusing on an element. Equivalent to: `Actions.click(element).sendKeys(theKey);`

Parameters:

key Either `Keys.SHIFT`, `Keys.ALT` or `Keys.CONTROL`.

target WebElement to perform the action on

Returns:

A self reference.

This method first focusses on the web element, which gets passed as a parameter to the method. Then, it releases the mentioned key on that Web Element.

Conclusively, we are clear about all the keyboard specific methods provided by the *Actions* class. Subsequently, let's see how we can automate the user as mentioned above scenario using the methods provided by the *Actions* class of *Selenium WebDriver*.

Let's modify the above-written code-snippet to use the methods of *Actions* class, instead of using the methods of *WebElement* class:

```
package automationFramework;

import static org.junit.Assert.assertEquals;

import org.openqa.selenium.By;
import org.openqa.selenium.Keys;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.interactions.Actions;

public class KeyboardEventsUsingActions {

    public static void main(String[] args) {

        // Initialize ChromeDriver
        // Here we assume that the ChromeDriver path has been set in the System G
        WebDriver driver=new ChromeDriver();

        //Navigate to the demo site
        driver.get("https://demoqa.com/text-box");

        //Create object of the Actions class
        Actions actions = new Actions(driver);

        // Enter the Full Name
        WebElement fullName = driver.findElement(By.id("userName"));
        fullName.sendKeys("Mr.Peter Haynes");

        //Enter the Email
        WebElement email=driver.findElement(By.id("userEmail"));
```

```

WebElement email=driver.findElement(By.id("currentEmail"));
email.sendKeys("PeterHaynes@toolsqa.com");

// Enter the Current Address
WebElement currentAddress=driver.findElement(By.id("currentAddress"));

currentAddress.sendKeys("43 School Lane London EC71 9G0");

// Select the Current Address using CTRL + A
actions.keyDown(Keys.CONTROL);
actions.sendKeys("a");
actions.keyUp(Keys.CONTROL);
actions.build().perform();

// Copy the Current Address using CTRL + C
actions.keyDown(Keys.CONTROL);
actions.sendKeys("c");
actions.keyUp(Keys.CONTROL);
actions.build().perform();

//Press the TAB Key to Switch Focus to Permanent Address
actions.sendKeys(Keys.TAB);
actions.build().perform();

//Paste the Address in the Permanent Address field using CTRL + V
actions.keyDown(Keys.CONTROL);
actions.sendKeys("v");
actions.keyUp(Keys.CONTROL);
actions.build().perform();

//Compare Text of current Address and Permanent Address
WebElement permanentAddress=driver.findElement(By.id("permanentAddress"));
assertEquals(currentAddress.getAttribute("value"),permanentAddress.getAttribute("value"));

driver.close();

}

}

```

When we run the above code snippet, we get a sample output, as shown below:



Text Box

Elements

Text Box

Check Box

Radio Button

Web Tables

Buttons

Links

Full Name

Mr.Peter Haynes

Email

PeterHaynes@toolsqa.com

Current Address

43 School Lane London EC71 9GO

Permanent Address

43 School Lane London EC71 9GO

Submit

As we can see in the above screenshot, copying of the address from the **"Current Address "** field to the **"Permanent Address "** field was successful. Few more point, which we should explicitly focus on the above code:

*Whichever **META** key (Eg **CONTROL** in the above use case) we press using the **keyDown()** method, it must be released using the **keyUp()** method. Otherwise, it will remain pressed and can cause side-effects on the next line of code.*

*All the commands of the **"Actions"** class perform/ execute their operations when we invoke the **"build()"** and **"perform()"** methods. So, each of the expected actions/commands should follow by these methods.*

The above code will work on the Windows platform only, as the **CTRL+C, etc.**, is only the Windows-specific operation. We can update the platform-specific keys when we need to run the same program on other platforms.

How to handle contiguous Keyboard Actions using Actions Class?

As noticed in the above sections, all the methods of the **Actions** class briefed above returns an **object of the Actions class** only. So, this gives us the flexibility of using the "**Chaining of Methods**", where we can club all the method invocations specific to one operation in one line of code only.

Let's modify the above-written code to trim it down further. We will use the *Chaing of Methods* and will handle the various contiguous Keyboard operations in a single go:

```
package automationFramework;

import static org.junit.Assert.assertEquals;

import org.openqa.selenium.By;
import org.openqa.selenium.Keys;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.interactions.Actions;

public class KeyboardEventsUsingActions {

    public static void main(String[] args) {

        // Initialize ChromeDriver
        // Here we assume that the ChromeDriver path has been set in the System G
        WebDriver driver=new ChromeDriver();

        //Navigate to the demo site
        driver.get("https://demoqa.com/text-box");

        //Create object of the Actions class
        Actions actions = new Actions(driver);

        // Enter the Full Name
        WebElement fullName = driver.findElement(By.id("userName"));
        fullName.sendKeys("Mr.Peter Haynes");

        //Enter the Email
        WebElement email=driver.findElement(By.id("userEmail"));
        email.sendKeys("PeterHaynes@toolsqa.com");

        // Enter the Current Address
        WebElement currentAddress=driver.findElement(By.id("currentAddress"));

        currentAddress.sendKeys("43 School Lane London EC71 9G0");

        // Select the Current Address
        actions.keyDown(Keys.CONTROL).sendKeys("a").keyUp(Keys.CONTROL).build().perform()

        // Copy the Current Address
        actions.keyDown(Keys.CONTROL).sendKeys("c").keyUp(Keys.CONTROL).build().perform()
```

```

//Press the TAB Key to Switch Focus to Permanent Address
actions.sendKeys(Keys.TAB).build().perform();

//Paste the Address in the Permanent Address field
actions.keyDown(Keys.CONTROL).sendKeys("v").keyUp(Keys.CONTROL).build().perform()

//Compare Text of current Address and Permanent Address
WebElement permanentAddress=driver.findElement(By.id("permanentAddress"));
assertEquals(currentAddress.getAttribute("value"),permanentAddress.getAttribute("value"));

driver.close();

}

}

```

The above code-snippet will perform precisely the same functionality as was being briefed in the previous section. The only difference between the two is that this code looks more compact and easy to read. So, this way, we can combine various methods of the Actions class. Additionally, we can simulate user behavior for different Keyboard actions.

Key Takeaways

Keyboard events are the events that any of the Keyboard keys generate.

Additionally, they are a must to simulate the user behavior while automating a web application using Selenium WebDriver

The Actions Class of Selenium WebDriver provides - sendKeys(),keyUp(),keyDown() methods to handle various keyboard actions

The modifier key is never released implicitly after the keyDown() method - either we should call the keyUp(theKey) or sendKeys(Keys.NULL) to release the modifier.