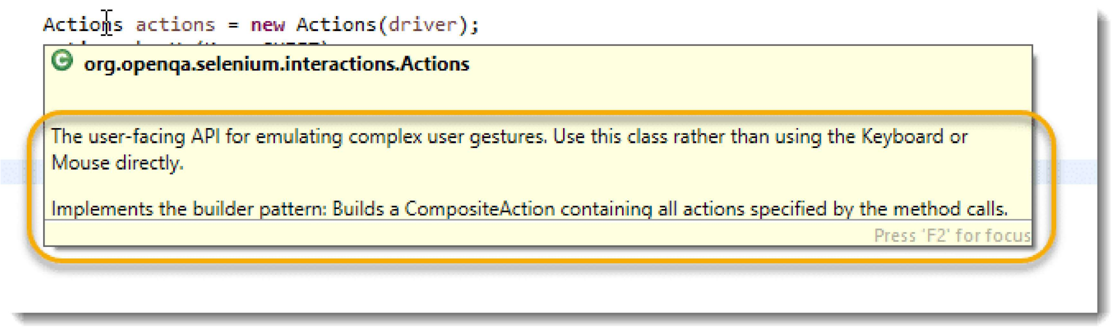
Most user interactions like clicking on a button, entering text in textbox can be done using the ***WebDriver Element Commands.*** For e.g. Commands like ***WebElement.click()*** and ***WebElement.sendKeys()*** are used to click on buttons and enter text in text boxes. Submitting a form can be done using the ***WebElement.submit()*** command. Even interactions with drop-down are enabled using the ***Select*** class which exposes commands like ***selectByValue(), deselectAll()*** to select and deselect options.

However, there are complex interactions like ***Drag-n-Drop*** and ***Double-click*** which cannot be done by simple ***WebElement*** commands. To handle those types of advance actions we have the ***Actions*** class in Selenium.

In this tutorial, we are going to cover Actions class and its usage.

What is Actions class in Selenium?

Let us start by taking a look at the information shown by intellisense for the ***Actions*** class. You can see this by hovering over ***Actions*** class in any IDE, a pop up menu should open up as shown below.



```
Actions actions = new Actions(driver);
```

org.openqa.selenium.interactions.Actions

The user-facing API for emulating complex user gestures. Use this class rather than using the Keyboard or Mouse directly.

Implements the builder pattern: Builds a CompositeAction containing all actions specified by the method calls.

Press 'F2' for focus

The user-facing API for emulating complex user gestures. Use this class rather than using the Keyboard or Mouse directly.

Implements the builder pattern: Builds a CompositeAction containing all actions specified by the method calls.

The class description clearly states that we can perform complex user interaction using the ***Actions*** class.

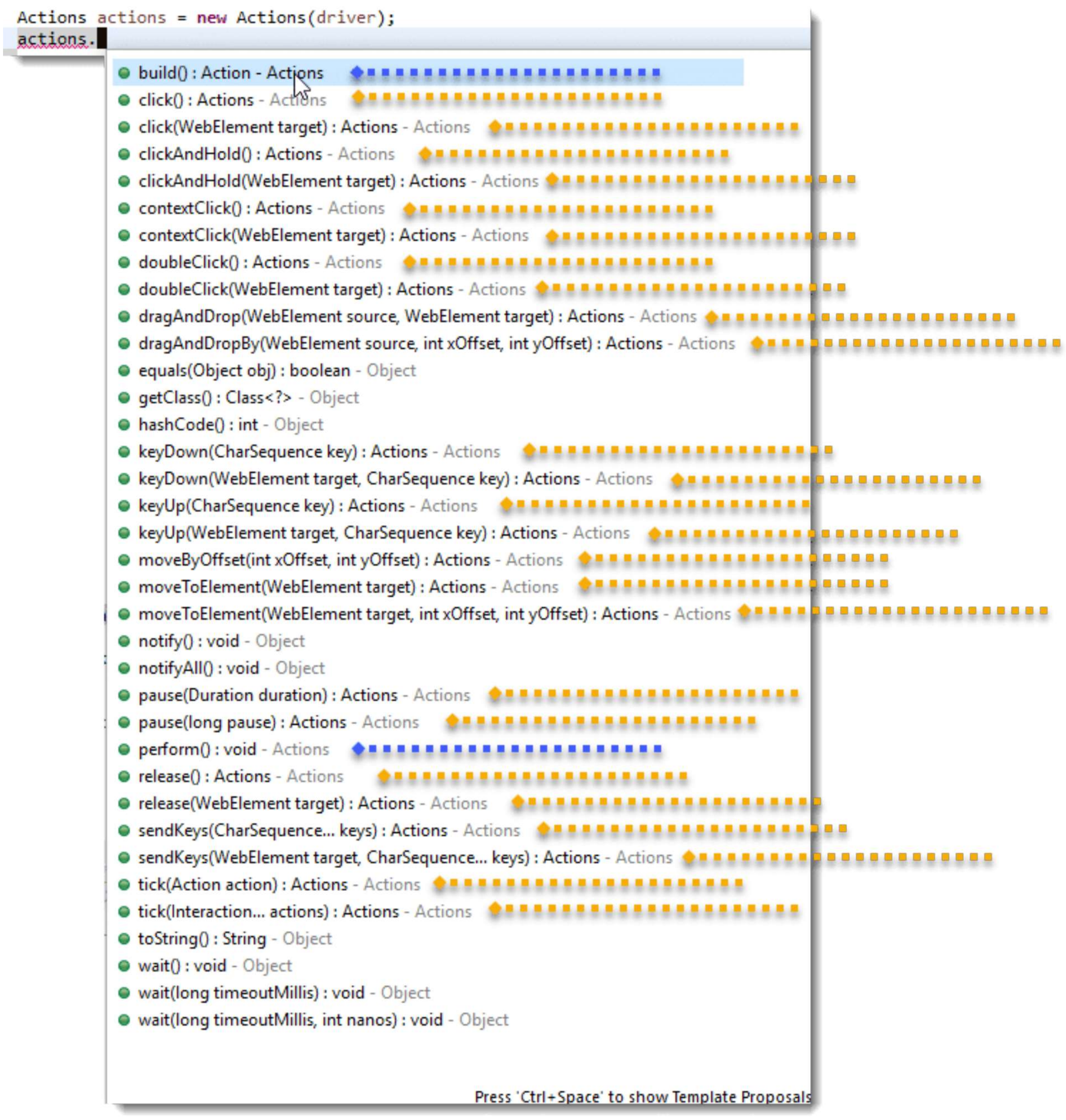
Actions Class

Actions class is a collection of individual ***Action*** that you want to perform. For e.g. we may want to perform a mouse click on an element. In this case we are looking at two different ***Action***

. Moving the mouse pointer to the element
. Clicking on the element

Individual action mentioned above are represented by a class called ***Action,*** we will talk about it later. Collection of such ***Action*** is represented by the ***Actions*** class.

There is a huge collection of methods available in ***Actions*** class. The below screenshot shows the list of methods available.



```
Actions actions = new Actions(driver);
actions.
```

- build() : Action - Actions
- click() : Actions - Actions
- click(WebElement target) : Actions - Actions
- clickAndHold() : Actions - Actions
- clickAndHold(WebElement target) : Actions - Actions
- contextClick() : Actions - Actions
- contextClick(WebElement target) : Actions - Actions
- doubleClick() : Actions - Actions
- doubleClick(WebElement target) : Actions - Actions
- dragAndDrop(WebElement source, WebElement target) : Actions - Actions
- dragAndDropBy(WebElement source, int xOffset, int yOffset) : Actions - Actions
- equals(Object obj) : boolean - Object
- getClass() : Class<?> - Object
- hashCode() : int - Object
- keyDown(CharSequence key) : Actions - Actions
- keyDown(WebElement target, CharSequence key) : Actions - Actions
- keyUp(CharSequence key) : Actions - Actions
- keyUp(WebElement target, CharSequence key) : Actions - Actions
- moveByOffset(int xOffset, int yOffset) : Actions - Actions
- moveToElement(WebElement target) : Actions - Actions
- moveToElement(WebElement target, int xOffset, int yOffset) : Actions - Actions
- notify() : void - Object
- notifyAll() : void - Object
- pause(Duration duration) : Actions - Actions
- pause(long pause) : Actions - Actions
- perform() : void - Actions
- release() : Actions - Actions
- release(WebElement target) : Actions - Actions
- sendKeys(CharSequence... keys) : Actions - Actions
- sendKeys(WebElement target, CharSequence... keys) : Actions - Actions
- tick(Action action) : Actions - Actions
- tick(Interaction... actions) : Actions - Actions
- toString() : String - Object
- wait() : void - Object
- wait(long timeoutMillis) : void - Object
- wait(long timeoutMillis, int nanos) : void - Object

Press 'Ctrl+Space' to show Template Proposals

Also, an important thing to bring here is that there is one another class which is called ***Action Class*** and it is different from Actions class. Because maybe you have noticed the top blue line in the above screenshot, the build method returns Action class. But then what is Action class and how does it different with Actions Class. Let's have a look.

What is Action Class?

Did I mention Action Class, actually it is not a class but an ***Interface.***

It is only used to represent the ***single user interaction to perform*** the series of action items build by Actions class.

What is the difference between Actions Class and Action Class in Selenium?

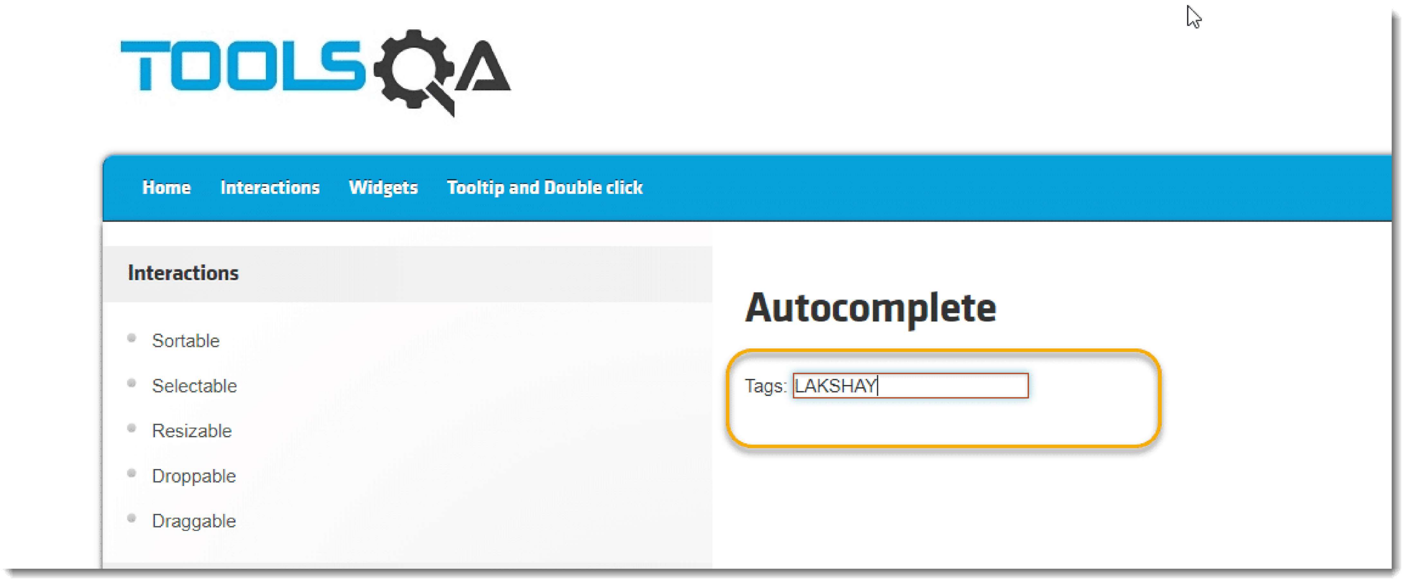
With the above explanations of Actions Class & Action Class, we can now conclude that ***Actions is a class*** that is based on a builder design pattern. This is a user-facing API for emulating complex user gestures.

Whereas ***Action is an Interface*** which represents a single user-interaction action. It contains one of the most widely used methods perform().

How to Use Actions class in Selenium?

Let's understand the working of Actions class with a simple example:

Consider the scenario where it is required to enter Upper Case letters in the text box, let's take text box on ToolsQA's demo site ***http://demoqa.com/auto-complete/***



Manually, it is done by pressing the ***Shift key and then typing the text*** which needs to be entered in Uppercase keeping Shift key pressed and then release the Shift key. In short ***Shift + Alphabet Key*** are pressed together.

Now, to emulate the same action through automation script, Actions class method is used:

1.Import package:

Actions class & Action class reside in ***org.openqa.selenium.Interactions*** package of ***WebDriver API.*** To consume these, import their packages:

import org.openqa.selenium.interactions.Actions;

import org.openqa.selenium.interactions.Action;

2. Instantiate Actions class:

Actions class object is needed to invoke to use its methods. So, let's instantiate Actions class, and as the Class signature says, it needs the WebDriver object to initiate its class.

Actions actions = new Actions(webdriver object);

3. Generate actions sequence: Complex action is a sequence of multiple actions like in this case sequence of steps are:

Pressing Shift Key
Sending desired text
Releasing Shift key

For these actions, Actions class provides methods like:

Pressing Shift Key : Actions Class Method => keyDown
Sending desired text : Actions Class Method => sendKeys
Releasing Shift key : Actions Class Method => keyUp

The keyDown method performs a modifier key press after focusing on an element, whereas keyUp method releases a modifier key pressed.

A modifier key is a key that modifies the action of another key when the two are pressed together like Shift, Control & Alt.

Generate a sequence of these actions but these actions are performed on a webElement. So, let's find the web-element and generate the sequence:

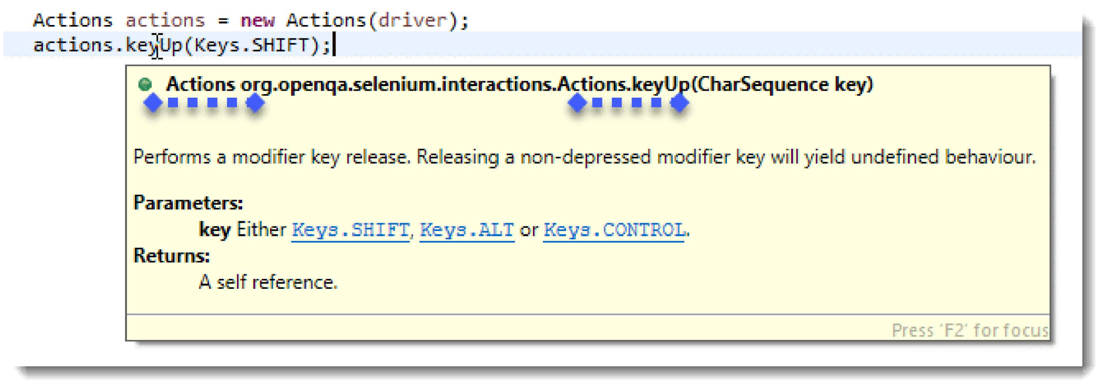
WebElement element = driver.findElement(By strategy to identify element);

actions.keyDown(element, Keys.SHIFT);

*actions.sendKeys("**TextToBeConvertAndSendInUpperCase**");*

actions.keyUp(Keys.SHIFT);

An important thing to note here is that, if you hover over any action class method, you will notice that it returns the Actions class object.



```
Actions actions = new Actions(driver);
actions.keyUp(Keys.SHIFT);
```

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.

Press 'F2' for focus

This is the beauty of the builder pattern. Which means that all actions can be clubbed together as below:

*actions.keyDown(element, Keys.SHIFT).sendKeys("**TextToBeConvertAndSendInUpperCase**").keyUp(Keys.SHIFT);*

4. Build the actions sequence:

Now, build this sequence using the ***build() method*** of Actions class and get the composite action. Build method generates a composite action containing all actions so far which are ready to be performed.

Action action = actions.build();

Notice that the build method returns the object type of ***Action.*** It is basically representing Composite Action which we built from a sequence of multiple actions. So, the second part of the Actions class description will get clear now, i.e. *Actions class implements the builder pattern* i.e. it Builds a CompositeAction containing all actions specified by the method calls.

5. Perform actions sequence: And finally, perform the actions sequence using ***perform()*** method of Action Interface.

action.perform();

And this is done, once the execution passes this point, you will notice the action on the browser.

Same steps need to follow to leverage Actions class methods for performing different complex combinations of user gestures.

Catch in Actions Class:

In the above screenshot of Actions Class, where I highlighted all the different methods, take a look at the bottom blue line. Notice the Perform method is available in the Actions class as well. It means that it can be used directly as well without making the use of Action Interface like below:

actions.keyDown(element,Keys.SHIFT).sendKeys(TextToBeConvertAndSendInUpperCase).keyUp(Keys.SHIFT).perform();

I know you must be wondering what happened to Build step. Again, this is the charm of the builder pattern, build method is called inside the perform method automatically.

Methods in Actions class of Selenium

There are a lot of methods in this class which can be categorized into two main categories:

Keyboard Events
Mouse Events

Different Methods for performing Keyboard Events:

keyDown(modifier key): Performs a modifier key press.
sendKeys(keys to send): Sends keys to the active web element.
keyUp(modifier key): Performs a modifier key release.

Different Methods for performing Mouse Events:

click(): *Clicks at the current mouse location.*
doubleClick(): *Performs a double-click at the current mouse location.*

contextClick() : *Performs a context-click at middle of the given element.*
clickAndHold(): *Clicks (without releasing) in the middle of the given element.*
dragAndDrop(source, target): *Click-and-hold at the location of the source element, moves to the location of the target element*
dragAndDropBy(source, xOffset, yOffset): *Click-and-hold at the location of the source element, moves by a given offset*
moveByOffset(x-offset, y-offset): *Moves the mouse from its current position (or 0,0) by the given offset*
moveToElement(toElement): *Moves the mouse to the middle of the element*
release(): *Releases the depressed left mouse button at the current mouse location*

To see the complete list of all methods

visit ***https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/interactions/Actions.html***