Appium cheat sheet(Commonly Used Commands)

Author: Diksha Gupta(Sr. Automation Engineer)
Linkedin Profile: https://www.linkedin.com/in/diksha-gupta-3a538865/

Appium is a test automation tool used for testing Mobile Applications. It's an open source tool which supports various platforms such as iOS, Android, and Windows. It allows users to test various types of Mobile Applications such as:

1. Native Apps: These are native to a particular platform and are written using Android, IOS, or Windows SDKs.

2. Web Apps: These are web apps and are accessed via native browsers like Chrome, Safari, Firefox, etc.

3. Hybrid Apps: These are combinations of both Native and Web apps.

Here are some commonly used important commands:

Here aim is to collate the important commands which are exposed to us by Appium server and Java client. This neat cheat sheet you can refer to see all useful Appium commands.

1. driver.hideKeyboard()—This method is used to hide the on-screen keyboard on a mobile device. When an application requires user input, such as filling out a form or typing in a search query, the on-screen keyboard appears to allow the user to input text. However, once the user has finished typing and wants to interact with other parts of the application, the keyboard can get in the way and cover up important information. This is where this method comes in.

Here is an example on how to use this command

```
driver.findElement(By.name("city")).sendKeys("Delhi");
driver.hideKeyboard();
```

2. driver.getContextHandles()—This method returns a list of all available execution contexts on the device or emulator. Execution contexts in Appium refer to the different environments in which an application can run, such as the native app context or the web view context.When testing a mobile application, it’s important to be able to switch between these contexts to interact with different elements and

perform different types of actions. For example, when testing a hybrid application that contains both native and web view components, you may need to switch between the native app context and the web view context to perform different actions.

Here is an example on how to use this command

```
driver.getContextHandles();
```

3. driver.context("<context-handle-view name>")—This method allows you to switch between different execution contexts in the application being tested. When testing a mobile application, you may need to switch between the contexts to interact with different elements and perform different types of actions. For example, if you're testing a hybrid application that contains both native and web view components, you may need to switch between the native app context and the web view context to perform different actions.

The driver.context() method takes a context handle as its argument and switches the current execution context to the one specified. Context handles are identifiers that represent the different contexts available on the device.

Here is an example on how to use this command

```
driver.context("WEB_VIEW");
```

4. driver.setClipboardText("Text to copy") -This method allows you to set the text contents of the device's clipboard. This method is useful when testing applications that interact with the clipboard, such as those that involve copying and pasting text.

The driver.setClipboardText() method takes a string as its argument, which represents the text to be copied to the clipboard. Once the text has been set in the clipboard, it can be pasted into other applications or fields using the device's standard paste functionality

Here is an example on how to use this command

```
driver.setClipboardText("Diksha Gupta");
```

5. driver.getClipboardText()—This method allows you to retrieve the text contents of the device's clipboard. This method is useful when testing applications that interact with the clipboard, such as those that involve copying and pasting text.

The driver.getClipboardText() method does not take any arguments and returns a string representing the text currently stored in the clipboard. Once the text has been retrieved from the clipboard, it can be used to perform actions that involve pasting the text into other applications or fields.

Here is an example on how to use this command

```
driver.getClipboardText();
```

6. driver.pressKey(new KeyEvent(AndroidKey.HOME))—This method allows you to simulate pressing the Home button on an Android device. This method is useful when testing applications that involve navigating between different applications or returning to the device's home screen.

Here is an example on how to use this command

```
driver.pressKey(new KeyEvent(AndroidKey.HOME))
```


7. driver.pressKey(new KeyEvent(AndroidKey.BACK))—This method allows you to simulate pressing the Back button on an Android device. This method is useful when testing applications that involve navigating through different screens and returning to previous screens.

Here is an example on how to use this command

```
driver.pressKey(new KeyEvent(AndroidKey.BACK))
```

8. driver.pressKey(new KeyEvent(AndroidKey.ENTER))—This command is used in Appium to simulate pressing the "Enter" key on an Android device's keyboard.

Here is an example on how to use this command

```
driver.pressKey(new KeyEvent(AndroidKey.ENTER))
```

9. driver.rotate(<object>)—This command is used in Appium to rotate the screen orientation of an Android or iOS device during a test execution**.**

> Note—rotate() method works only on devices that support screen rotation. If the device doesn't support screen rotation, this method will not have any effect. Additionally, some devices may not support all possible screen orientations, and in such cases, the device may rotate to the closest possible orientation.

Here is an example on how to use this command

```
DeviceRotation rotation = new DeviceRoation(0,0,90)
driver.rotate(rotation );
```

10. Supported Locators

- **Accessibility ID:** This allows finding the elements using the element Accessibility labels.

  Here is an example on how to use this:


```
driver.fingElement(AppiumBy.accessibilityId("accessibility_id");
```

- **Id** : This allows finding the elements using the element unique ID

  Here is an example on how to use this:

```
driver.fingElement(AppiumBy.id("id");
```

- **Class Name:** This will use the element class name to locate it

  Here is an example on how to use this:

```
driver. fingElement(AppiumBy.className("className");
```

- **XPath:** XPath locator strategy will scan the complete XML hierarchy tree of the application screen for the locator you will try to find the element.

  Here is an example on how to use this:

```
driver. fingElement(AppiumBy.xpath("//div[@id='div']");
```

Conclusion:

Appium mobile Testing has become indispensable with the ever-increasing demands of fast and reliable testing that spans over various platforms, devices, and versions. Appium offers feasibility, flexibility, and cost-friendliness above other testing methods, which enables both development and testing teams to deliver great user experiences.

Please feel free to share.

Thank you for reading. Happy Learning !! DG