# Exercise II: Appium Mobile

## Mobile gestures

Interactions performed on touch-enabled devices such as smartphones and tablets. These gestures include **taps**, **swipes**, **pinches**, and **long presses**, allowing users to interact with the device's user interface. By translating physical touch into digital commands, gestures play a significant role in enhancing the user experience. Understanding and automating these gestures is essential for testing mobile applications to ensure they behave as expected.

**Common Mobile Gestures**:

**Tap:** A quick touch on the screen, similar to a mouse click.

**Long Press:** Touching and holding an element for an extended period.

**Swipe:** A quick movement across the screen, often used to scroll or switch views.

**Scroll:** Moving up or down to view content that extends beyond the screen's current view.

**Drag and Drop:** Selecting an object by pressing and holding it, moving it to a desired location, and then releasing it.

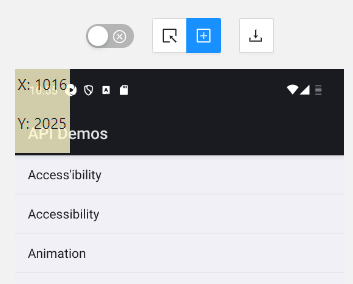
**Zoom In/Out:** A two-finger gesture used to change the zoom level of the screen.

## Appium Inspector: Interaction Mode in the Screenshot Panel

You're already familiar with the Screenshot panel, and also used Element Mode, so today we're going to focus on the Coordinates Mode. This mode is important for executing actions on the device screen directly from the Inspector.

**Coordinates Mode:** When you switch to this mode, the top left corner of the screenshot will show a coordinates overlay. This overlay updates as you hover over the screenshot, displaying the coordinates of your cursor on the device screen.

**Switching Modes:** You can switch between Element Mode and Coordinates Mode using the Interaction Mode buttons in the Screenshot panel header.



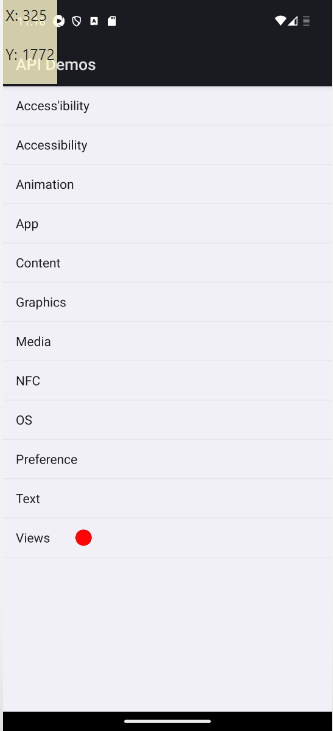
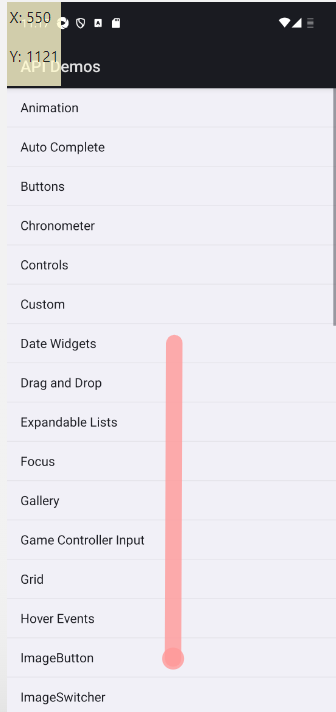
**Performing Actions:** In Coordinates Mode, you can perform various actions directly on the device screen. For example, you can tap on a button or swipe across the screen to test scrolling and other gestures.

## Scroll

The app for testing (**ApiDemos-debug.apk**) is included in the resources for this exercise. So, start the prerequisites needed to run the Appium Inspector, start Appium Inspector itself and let's do some scrolling.

We will perform the following actions:

* Open the App
* Tap on Views
* Scroll down
* Tap on Lists

## Scroll Test

**Set Up and Launch Appium Inspector:**

* Start your Appium server: Ensure the Appium server is running to allow connections.
* Open Appium Inspector: Connect it to your running Appium server.
* Load your testing app: Install and launch your testing app (e.g., ApiDemos-debug.apk) on a virtual or   
  real device.

**Inspecting Elements:**

Identify Elements: Use Appium Inspector to navigate through the app and find elements. Use Appium Inspector to locate "Views" and "Lists" and get their Accessibility ID. Switch to Coordinates Mode to swipe.

**ScrollToText Method:** This method leverages the UiScrollable selector to scroll through a list until the specified text ("Lists") is found.

The **UiScrollable** selector is part of the Android UI Automator framework, a powerful tool used for automated UI testing of Android applications. It is particularly useful when you need to perform scrolling actions in your test scripts. Here’s why we use it and how it works:

#### Why Use UiScrollable:

1. **Accuracy**: The boilerplate code generated by Appium Inspector for gestures like scroll and swipe often lacks the precision needed for consistent test results. The **UiScrollable** selector provides a more reliable method to scroll to specific elements by their text.
2. **Ease of Use**: Using **UiScrollable** is straightforward and simplifies the process of writing test scripts that involve scrolling. It abstracts away the complexities of handling different screen sizes and densities.
3. **Direct Targeting**: It allows you to target elements directly by their text, making your test scripts cleaner and easier to maintain.

#### How UiScrollable Works:

The **UiScrollable** selector scrolls through a scrollable layout until it finds an element that matches the specified criteria. Here's a breakdown of the key components:

1. **Creating UiScrollable Instance**:

No direct instantiation in C#, used via Appium's MobileBy

1. **Scrolling to Text**:

driver.FindElement(MobileBy.AndroidUIAutomator(

"new UiScrollable(new UiSelector().scrollable(true)).scrollIntoView(new UiSelector().text(\"Lists\"))"));

This method scrolls through the scrollable layout until the element with the specified text ("Lists") is found and brought into view.

1. **Example Code in C#**:

private void ScrollToText(string text)

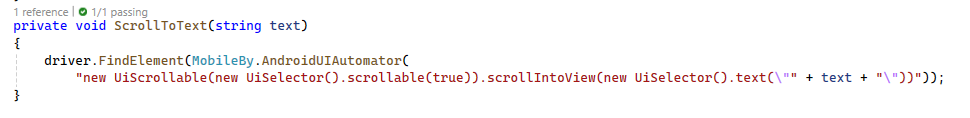
{

\_driver.FindElement(MobileBy.AndroidUIAutomator(

$"new UiScrollable(new UiSelector().scrollable(true)).scrollIntoView(new UiSelector().text(\"{text}\"))"));

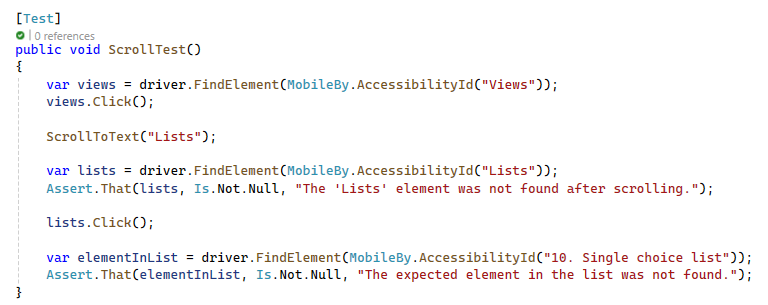
}

* Create a new **UiScrollable** instance: The **UiScrollable** selector is initialized with a **UiSelector** that specifies the view is scrollable.
* Scroll into view: The **scrollIntoView** method is called with a **UiSelector** that looks for the specified text.
* This causes the screen to scroll until the text is found or the end of the scrollable area is reached.



**ScrollTest Method:**

* Tap on "Views": The test starts by finding the "Views" element using its Accessibility ID ("Views") and clicking on it.
* Scroll Down: Calls the **ScrollToText** method to scroll down the list until the "Lists" item is visible.
* Tap on "Lists": Finds the "Lists" item using its Accessibility ID and clicks on it.
* Verify Element: After clicking on "Lists", the test verifies that the expected element ("10. Single choice list") is found in the resulting view.



## Swipe

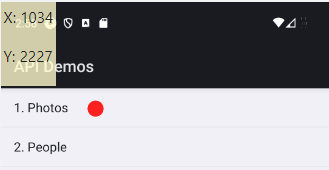
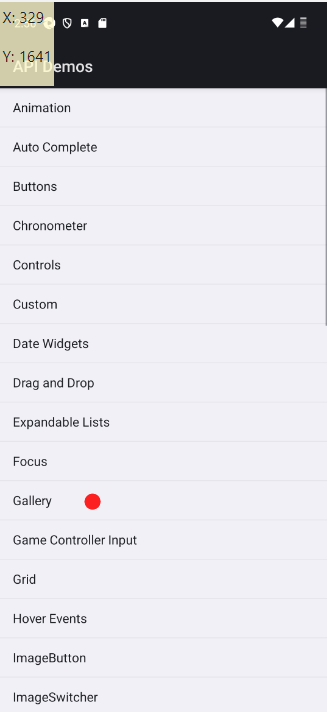
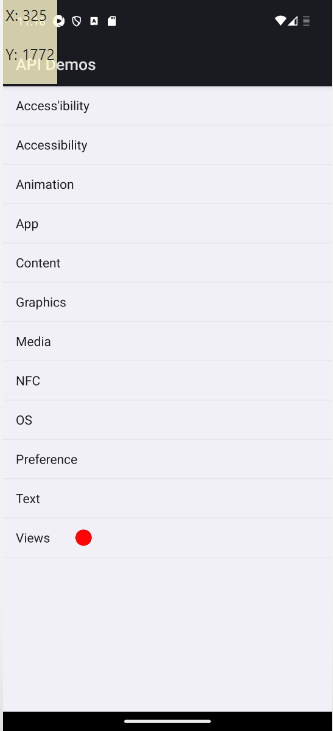
**Difference between Scroll and Swipe**: In summary, **scrolling** is a continuous action used for navigating through large amounts of content, whereas **swiping** is a quick gesture used to switch views or trigger specific actions.

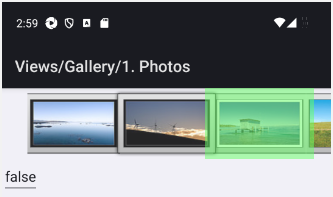
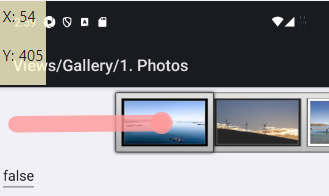
|  |  |  |
| --- | --- | --- |
| **Feature** | **Scroll** | **Swipe** |
| **Purpose** | Move content within a scrollable area | Quickly move between views or perform actions |
| **Interaction** | Continuous movement with one or more fingers | Quick, short flick of the finger |
| **Use Case** | Browsing long content (articles, lists) | Navigating pages, switching tabs, swipe-to-delete |
| **Behavior** | Content moves in drag direction and stops when drag ends; may bounce at edges | Triggers predefined responses, like changing views or executing actions |
| **Direction** | Up, down, left, right | Left, right, up, down |
| **Speed** | Usually slower and controlled | Faster and more immediate |
| **Examples** | Reading an article, viewing a list | Switching between carousel images, dismissing notifications |

The app for testing (ApiDemos-debug.apk) is included in the resources for this exercise. So, start the prerequisites needed to run the Appium Inspector, start Appium Inspector itself and let's do some swiping.

We will perform the following actions:

* Open the App
* Tap on Views
* Tap on Gallery
* Tap on Photos
* Swipe Left to see the third picture
* Assert the third picture is visible





## Swipe Test

**Set Up and Launch Appium Inspector:**

* Start your Appium server: Ensure the Appium server is running to allow connections.
* Open Appium Inspector: Connect it to your running Appium server.
* Load your testing app: Install and launch your testing app (e.g., ApiDemos-debug.apk) on a virtual or   
  real device.

**Inspecting Elements:**

Identify Elements: Use Appium Inspector to navigate through the app and find elements Use Appium Inspector to locate "Views", "Gallery", "Photos" and get their Accessibility ID. Also locate the first picture from the photos. Switch to Coordinates Mode to swipe left in a way that the third picture becomes visible.

**SwipeTest Method:**

* Tap on 'Views': Uses the accessibility ID obtained from Appium Inspector to find and click the "Views" element.
* Navigate to 'Gallery' and 'Photos': Finds and clicks on the "Gallery" and "Photos" elements using their accessibility IDs.
* Locate the image element: Uses By.ClassName to find the first image in the gallery.
* **Perform swipe using Selenium Actions:** Creates an action sequence to simulate a swipe gesture by clicking and holding the image, moving it by an offset, and then releasing it.

The W3C Actions API offers a simpler and accessible way to perform user gestures such as:

* Click and hold
* Release
* Move to an element
* Drag and drop
* Double click
* Context click (right click)
* Key down/up

**Steps**:

* Click and hold a specific element (pic1).
* Move by a specified offset.
* Release the hold.

**Ease of Use**: Simplifies common interactions like clicks and drags, without needing to define each step explicitly.

#### How It Works:

**Creating an Actions Instance:**

* + An instance of the Actions class is created by passing the WebDriver instance to its constructor.

Actions actions = new Actions(driver);

**Building an Action Sequence:**

* + Various methods are called on the Actions instance to build a sequence of actions.
  + Each method call returns the same Actions instance, allowing for method chaining.

actions.MoveToElement(element).Click().Perform();

**Performing the Actions:**

* + The perform method executes all the actions that were defined in the sequence.

actions.Perform();

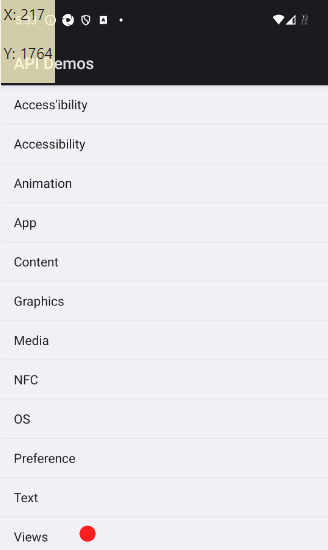
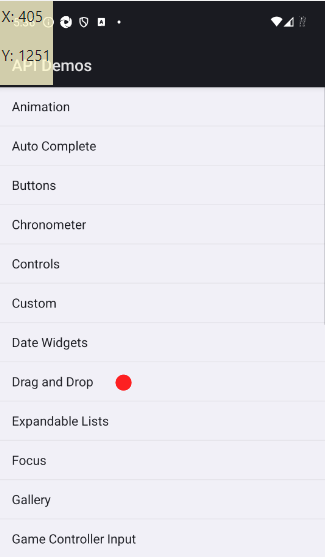
* **SwipeTest Method:**
* Finds the "Views" element and clicks it.
* Navigates to the "Gallery" and "1. Photos" sections by finding the elements and clicking them.
* Locates the first image in the gallery.
* Uses the Actions class to build a swipe action:
* ClickAndHold(pic1): Clicks and holds on the first image.
* MoveByOffset(-200, 0): Moves the pointer horizontally by -200 pixels.
* Release(): Releases the pointer, completing the swipe action.
* Executes the action using swipe.Perform().
* Verifies that the third image is found after the swipe action.

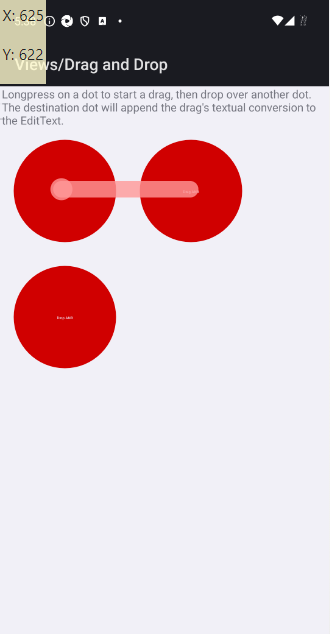
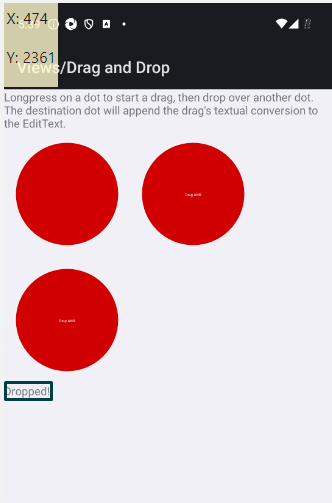


## Drag and Drop

The drag and drop gesture involves selecting an object by pressing and holding it (dragging), moving it to a desired location, and then releasing it (dropping). This gesture is used for rearranging items, moving files, or performing specific actions like deleting an icon by dragging it to a trash bin.

The app for testing (ApiDemos-debug.apk) is included in the resources for this exercise. So, start the prerequisites needed to run the Appium Inspector, start Appium Inspector itself and let's do some drag and dropping.

We will perform the following actions:

* Open the App
* Tap on Views
* Tap on Drag and Drop
* Tap and hold the first red dot.
* Drag and Drop the first red dot over the second red dot
* Assert "Dropped!" text.

**Note**: Don't mind the fourth red dot that appears while dragging! Some developer had a sense of humor. 😊

## Drag and Drop Test

**Set Up and Launch Appium Inspector**

* Start your Appium server: Ensure the Appium server is running to allow connections.
* Open Appium Inspector: Connect it to your running Appium server.
* Load your testing app: Install and launch your testing app (e.g., ApiDemos-debug.apk) on a virtual or real device.
* Inspecting Elements

**Identify Elements**

* Use Appium Inspector to navigate through the app and find elements. Use Appium Inspector to locate "Views" and "Drag and Drop" and get their Accessibility IDs. Also, locate the draggable dot and the drop target dot.
* Switch to Coordinates Mode.

**Drag and Drop Test Method**

Use the accessibility ID obtained from Appium Inspector to find and click the "Views" element.

Find and clicks on the "Drag and Drop" element using its accessibility ID.

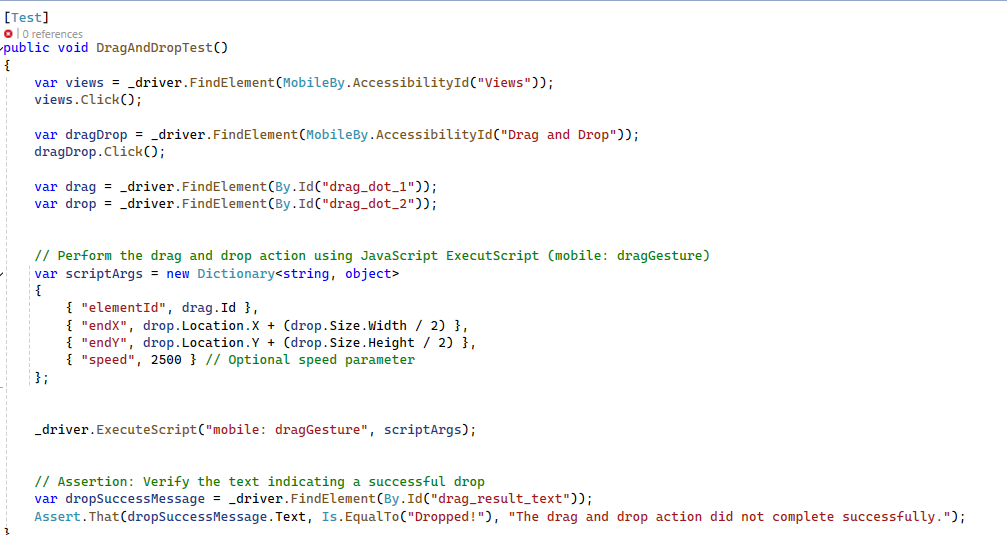
Use By.Id to find the draggable dot ("drag\_dot\_1") and the drop target dot

Perform drag and drop using a JavaScript method.

Create a dictionary to specify the drag and drop action.

Execute the drag and drop action using the ExecuteScript method.

Verify the text indicating a successful drop.

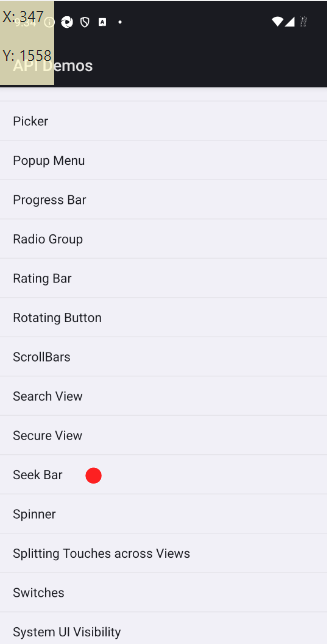
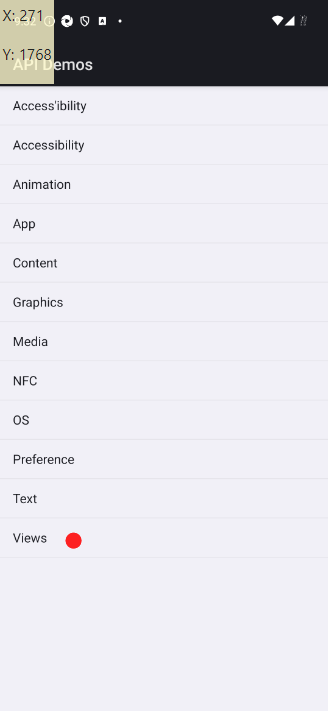


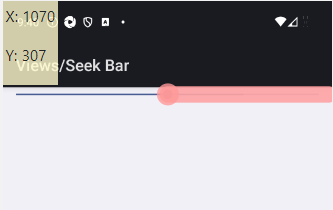
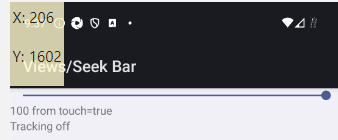
## Sliding

The sliding of the seek bar gesture involves interacting with a slider by pressing and holding a specific point on the slider, moving it horizontally to adjust a value, and then releasing it. This gesture is commonly used in user interfaces to control settings such as volume, brightness, or progress.

The app for testing (ApiDemos-debug.apk) is included in the resources for this exercise. So, start the prerequisites needed to run the Appium Inspector, start Appium Inspector itself, and let's do some seek bar sliding.

* We will perform the following actions:
* Open the App
* Tap on Views
* Scroll own to Seek Bar Option
* Tap on Seek Bar Option
* Use Appium Inspector to get the precise coordinates for the sliding action.
* Move Seek Bar with Inspector's Coordinates.
* Find the element displaying the slider position text and assert that it shows the expected value.



## Seek Bar Test

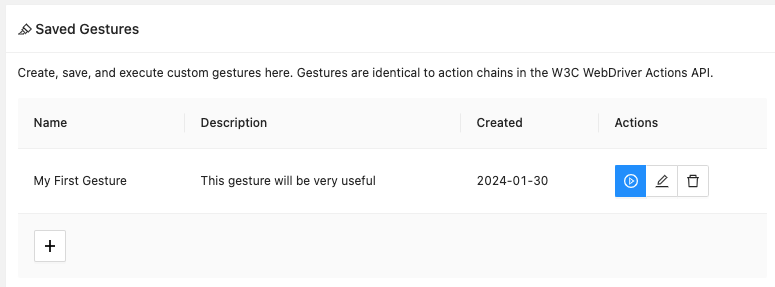
* Use the accessibility ID obtained from Appium Inspector to find and click the "Views" element.
* Use the ScrollToText method to scroll down the list until the "Seek Bar" item is visible.
* Use the accessibility ID to find and click the "Seek Bar" element.
* Use the MoveSeekBarWithInspectorCoordinates method to slide the seek bar from the start to the end coordinates.
* Find the element displaying the slider position text and assert that it shows the expected value.



## Gestures in Appium Inspector

The Gestures tab lets you create, save, and run custom gestures. When you switch to this tab, you'll see the Saved Gestures screen by default.

### 2.1. Gestures Tab

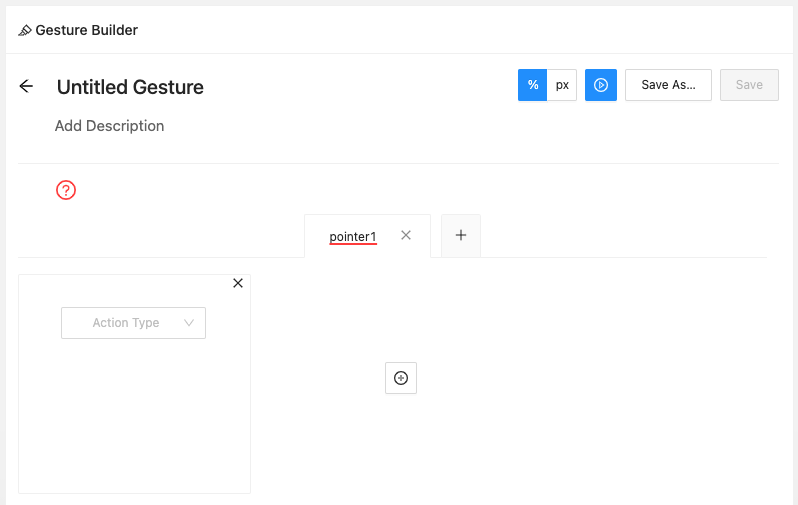


* The list of saved gestures is empty at first.
* You can add new gestures by pressing the "+" button at the bottom left, which opens the Gesture Builder screen.
* Once saved, clicking on a gesture shows its actions as an overlay on the screenshot.
* There are three buttons for each gesture:
  + Execute the gesture
  + Edit the gesture (opens it in the Gesture Builder)
  + Delete the gesture

### 2.2. Gesture Builder

The Gesture Builder opens when you create a new gesture or edit an existing one.

Most of the Screenshot panel header buttons are disabled while the Gesture Builder is open.



* The Gesture Builder screen has four main parts:
  + Header
  + Gesture timeline
  + Pointer tabs
  + Pointer actions

#### 2.2.1. Header

The header shows general gesture info, settings, and actions.



* The gesture title and description can be clicked and edited.
* You can set the X/Y coordinate units for Move pointer actions to either relative (percentage) or   
  absolute (pixels).
* The Play button runs the gesture.
* The Save As button saves the gesture with the specified title.
* The Save button, available when editing, updates the existing gesture with current details.

#### 2.2.2. Gesture Timeline

A new gesture timeline starts with a red circle and a question mark below the description, indicating one   
undefined action.

Gesture Timeline Empty

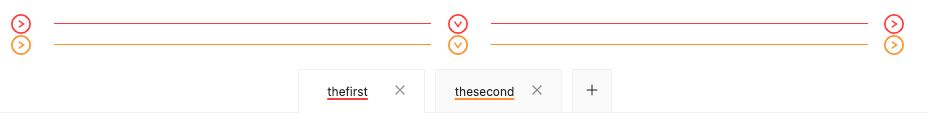
The timeline updates as you add or modify pointer actions.

Gesture Timeline Full

* Example timeline for a swipe gesture:
  + Move to location
  + Pointer down
  + Move to location
  + Pointer up

#### 2.2.3. Pointer Tabs

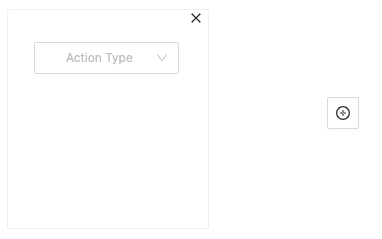
The pointer tabs show the current number of pointers used simultaneously in the gesture.



* A new gesture starts with one pointer, but you can have up to 5 pointers.
* Each pointer has a unique color and appears as a separate row in the gesture timeline.
* The title of each pointer can be clicked and edited.

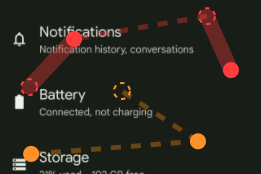
#### 2.2.4. Pointer Actions

Pointer actions define what each pointer does. Actions are shown as rectangular cards, and you can add or remove cards as needed.



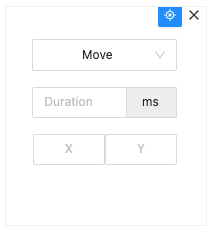
* Action Type dropdown includes:
  + move
  + pointer down
  + pointer up
  + pause

Valid action combinations show the gesture behavior on the screenshot.



##### 2.2.4.1. Move

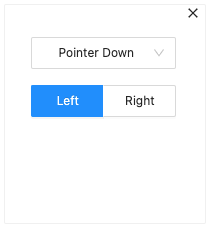
The Move action usually starts most gestures.



* Requires duration (in milliseconds) and X/Y coordinates for the target location.
* The aim button in the top right of the card allows setting X/Y coordinates by clicking the screenshot.
* X and Y values automatically change based on the selected units.

##### 2.2.4.2. Pointer Down/Pointer Up

Both of these actions only require specifying whether the left or right pointer should be used. For pointer types where this does not apply (e.g. touch), the default left pointer can be used.

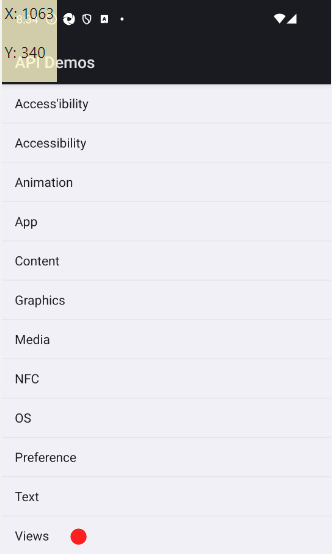
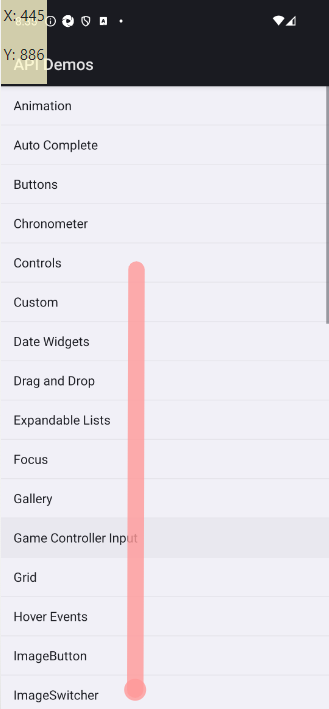
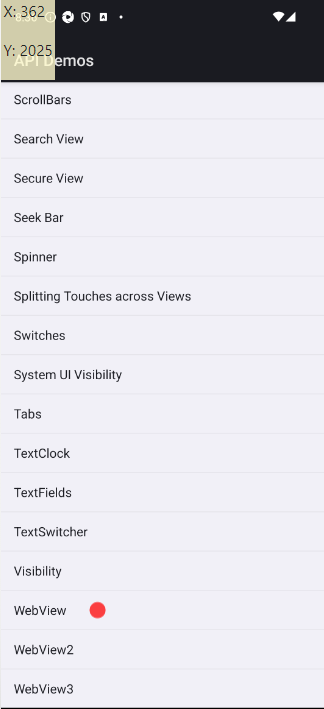


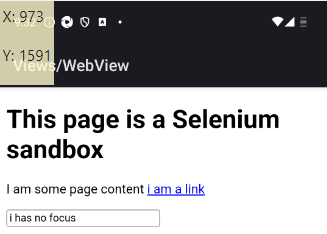
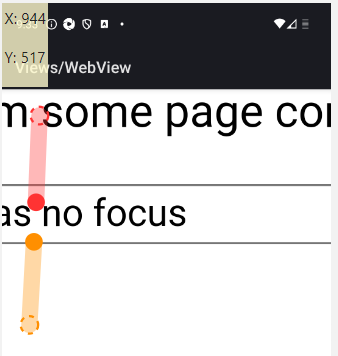
##### 2.2.4.3. Pause

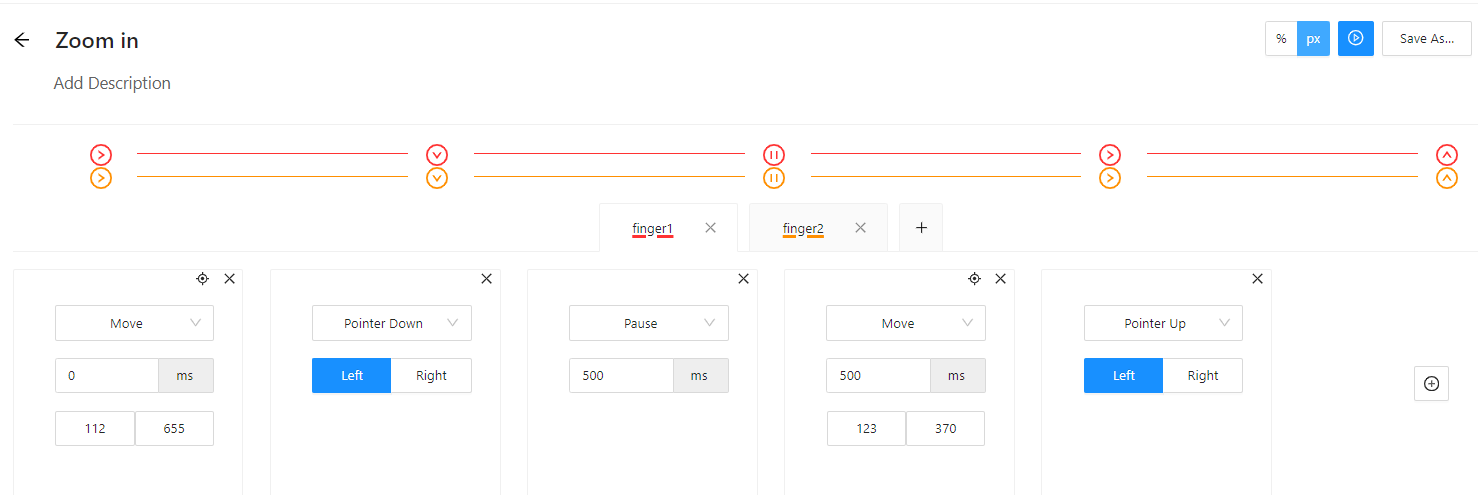
Requires only the pause time (in milliseconds).



## Zoom In and Zoom Out



## Zoom In and Zoom Out Test

**Set Up and Launch Appium Inspector**

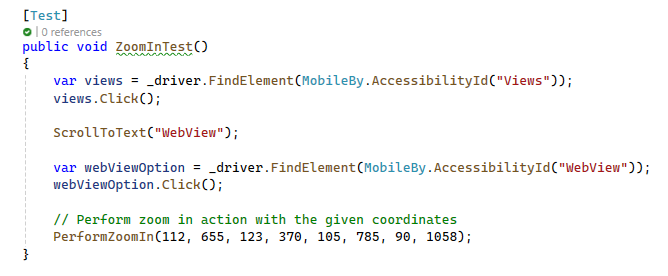
* Ensure the Appium server is running to allow connections.
* Connect it to your running Appium server.
* Install and launch your testing app (e.g., ApiDemos-debug.apk) on a virtual or real device.

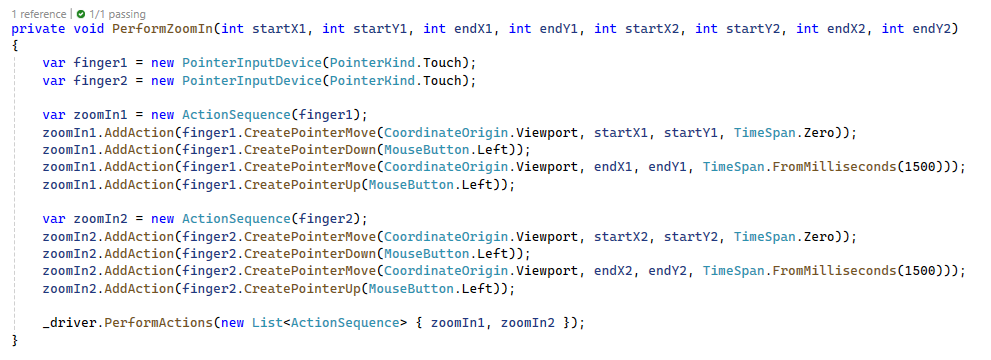
**Inspecting Elements**

* Use Appium Inspector to navigate through the app and find elements.
* Use Appium Inspector to locate "Views" and "WebView" and get their Accessibility IDs.
* Create a Zoom In Gesture in Appium Inspector.
* Use the coordinates from it when writing your tests.

**Zoom In Test Method**

* Uses the accessibility ID obtained from Appium Inspector to find and click the "Views" element.
* Scrolls down the list until the "WebView" item is visible and clicks it.
* Uses the specific coordinates obtained from Appium Inspector to perform the zoom-in gesture at the desired location.
* Moves the fingers apart to zoom in.
* Uses a slower speed for the gesture.





**Zoom Out Test Method**

* Uses the specific coordinates obtained from Appium Inspector to perform the zoom-out gesture at the desired location.
* Moves the fingers together to zoom out.
* Uses a slower speed for the gesture.

