How to Use Appium Image Locator for Finding Elements and Image Recognition (https://bitbar.com/blog/how-to-use-appium-image-locator-for-finding-elements-and-image-recognition/)

It's always been a daunting task to perform automated mobile testing for graphics-heavy apps, games or even any OS-level popups. That is the main reason that Appium image recognition is one of the most popular approaches for enabling automated testing of these particular mobile apps.

The problem with regular test automation frameworks is that they may not be able to identify the graphics content of a mobile app with IDs, descriptions or object characters. Therefore, our own Appium image recognition library has been widely used among mobile developers and testers as a solution to that challenge.

The Appium 1.9.0 release has brought us a new image locator strategy especially for image recognition, `-image`. Today's we'll look at how this new approach works.

Download Our Appium Step-by-Step Tutorial to Properly Set Up Your Appium Testing Environment for Image Recognition (https://www2.smartbear.com/bitbar-beginner-guide-appium-mobile-app-testing.html)

What is an Image Element?

An image element is a base64-encoded image file that is used as a template to compare with the image captured from a device screen.

Without image comparison/matching, it would be impossible to automate the testing for graphics-heavy apps, let alone running those tests on hundreds of real mobile devices in parallel (https://bitbar.com/blog/parallelism-and-concurrency-in-mobile-app-testing/).

The reason being, none of the images have any identifying information in the UI tree, and their location changes on the device screen every time we load the view.

If the template image matches some region of the screen, then we can interact with the found image element by calling a small number of WebElement methods, such as:

- click
- isDisplayed
- getSize
- getLocation
- getElementRect

What about typing and sendKeys?

Typing can be done by using a keyboard image as a template and then tapping certain points of that screen region, using for example “getSize” and “getLocation” methods.

However, **sendKeys is not supported**, because Appium has no way of turning the match region into a driver-specific UI element object, which is needed for using other actions.

Get Started with Appium Image Recognition

First things first, the `-image` locator was a new feature in Appium 1.9.0, so make sure you have installed the latest version of Appium (https://github.com/appium/appium/releases).

Install

Appium does not come with OpenCV image comparison library, it needs to be installed manually using this command (https://github.com/justadudewhohacks/opencv4nodejs):

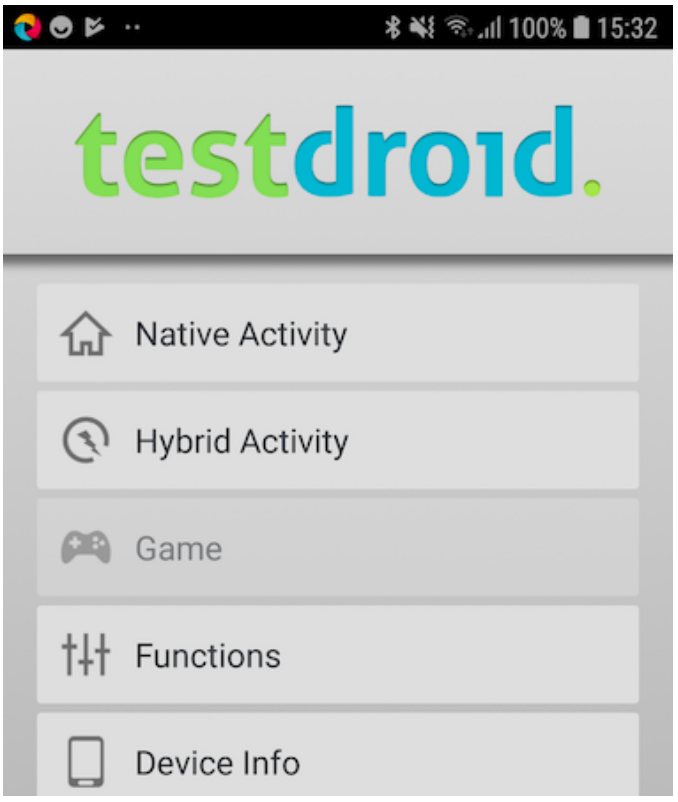
```
npm i -g opencv4nodejs
```

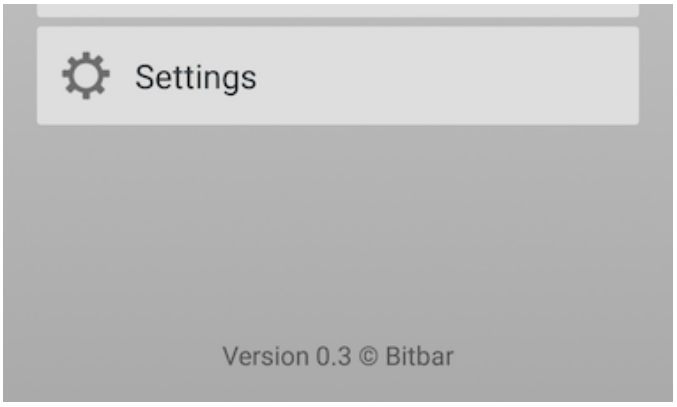
Usage

Reference images are located (in my case) in a folder called “queryimages” in the project root, and they are in PNG format. The reference images must be converted to Base64 encoded format.

```
protected String getRefImage(String refImgName) throws Exception {

File file = new File(System.getProperty("user.dir") + refImageFolderLocation + refImgName + ".png");
Path path = file.toPath();
return Base64.getEncoder().encodeToString(Files.readAllBytes(path));
}
```





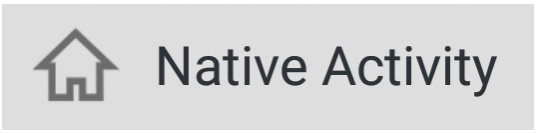
Find “logo” image using “MobileBy.image” or “findElementByImage” (Java, appium-java-client 7.2.0).


```
String logoImage = getRefImage ("logo");

wd.findElement(MobileBy.image(logoImage));

wd.findElementByImage(logoImage);
```

Find “native-button” and tap it. Will tap in the middle of the image element.




```
String nativeButtonImage = getRefImage ("native-button");

wd.findElement(MobileBy.image(nativeButtonImage)).click();
```

Tapping certain parts of the image element for example bottom left corner can be done like this.

```
MobileElement logo = wd.findElement(MobileBy.image(logoImage));
int xCenter = logo.getCenter().getX();
int yCenter = logo.getCenter().getY();
double xLeftCorner = xCenter * 0.1;
double yBottomCorner = yCenter + (yCenter * 0.4);

TouchAction action = new TouchAction(wd);
PointOption pointOption = new PointOption();
pointOption.withCoordinates((int)xLeftCorner,(int)yBottomCorner);
WaitOptions waitOptions = new WaitOptions();
waitOptions.withDuration(Duration.ofMillis(100));

action.press(pointOption)
.waitAction(waitOptions).release();
action.perform();
```

Modifying -image Locator Settings with Settings API

Some settings of finding elements by image can be modified by using Appium Settings API (http://appium.io/docs/en/advanced-concepts/settings/index.html).

- ImageMatchThreshold
  - How strict comparison is

    - Value from 0 to 1
    - Default 0.4
- FixImageFindScreenshotDims
    - Adjust the size of the screenshot to match screen dimensions
    - True or false
    - Default true
- FixImageTemplateSize
    - Resize reference image to be smaller than the size of the screenshot
    - True or false
    - Default false
- FixImageTemplateScale
    - Scale screenshot down to window size (for iOS), scales image by half (0.5)
    - True or false
    - Default false
- DefaultImageTemplateScale
    - Scale reference images to save storage space
    - Value e.g. 0.5, 10.0
    - Default 1.0 (no scaling)
- CheckForImageElementStaleness
    - Check that image element still exists on the screen before executing tap
    - True or false
    - Default true
- AutoUpdateImageElementPosition
    - Check that image element is still in the same position on the screen
    - True or false
    - Default false
- ImageElementTapStrategy
    - Choose between touch action strategies
    - W3cActions or touchActions
    - Default w3cActions
- GetMatchedImageResult
    - Store image matching result
    - True or false
    - Default false

Example of using the API with java-client:

```
driver.setSetting(Setting.IMAGE_MATCH_THRESHOLD, 0.3);
```

My thoughts on Appium –image locator strategy for recognition

I have been using a lot our own Appium image recognition library (https://github.com/bitbar/opencv-library) when doing automated Appium testing, for example, for mobile games, which can't be automated using normal locators strategies.

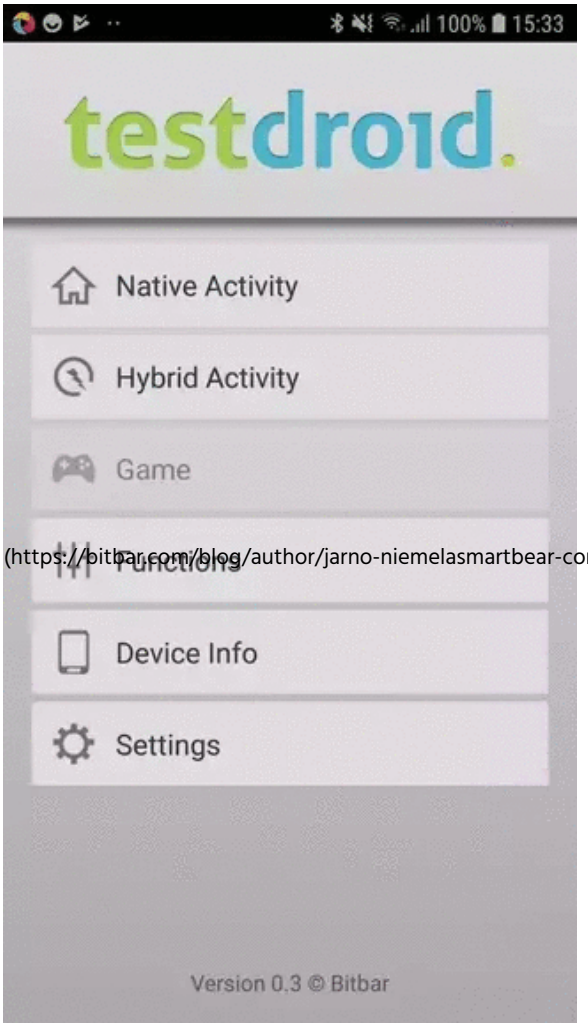
I had some doubts about how fast and usable the Appium `-image` locator can be for image recognition. But I have to say I was surprised.

Appium `-image` locator strategy worked at least as well as our own Appium image comparison library. The image recognition process was quite fast and even small images could be used as reference images reliably. I only tried it with really simple apps though.

But there were a couple of things that didn't work very well out of the box.

At first, I couldn't get the image comparison to work with iOS devices. No matter what image I was comparing to, it always failed. In the end, I used "FIX_IMAGE_FIND_SCREENSHOT_DIMENSIONS" setting and the image comparison started working.

Another thing was that the tap (click) action did not hit the correct spot on the screen of the iOS device. There is a setting for that



"FIX_IMAGE_TEMPLATE_SCALE", but that did not work. The coordinates for image elements were exactly two times too large. The solution for this was to make my own template where I would multiply those coordinate values by two.

If you have already built your own image library and Appium tests, check out how you can leverage our mobile app testing platform (https://bitbar.com/mobile-app-testing/) to get started with your Appium image recognition. Don't forget to check our other Appium tips (https://bitbar.com/blog/things-to-know-about-appium/)!

SHARE THIS ARTICLE

(https://www.linkedin.com/shareArticle?mini=true&url=https://bitbar.com/blog/how-to-use-appium-image-locator-for-finding-elements-and-image-recognition/&title=How to Use Appium Image Locator for Finding Elements and Image Recognition&summary=It always been a daunting task to perform automated mobile testing for graphics-heavy apps, games or even any OS-level popups. That is the main reason that Appium Image Locator for Finding Elements and Image Recognition is one of the most popular approaches for enabling automated testing of these particular mobile apps. The problem with regular test automation frameworks is that they [...]&source=Bitbar)

(http://www.facebook.com/sharer/sharer.php?u=https://bitbar.com/blog/how-to-use-appium-image-locator-for-finding-elements-and-image-recognition/)

(https://bitbar.com/blog/author/jarno-niemelasmartbear-com/)

Jarno Niemela
Test Automation Engineer

Categories

Have Another Byte

Uncategorized (https://bitbar.com/blog/category/uncategorized/)

The Importance of Synthetic Monitoring for Mobile Apps (https://bitbar.com/blog/the-importance-of-synthetic-monitoring-for-mobile-apps/)

READ MORE

Uncategorized (https://bitbar.com/blog/category/uncategorized/)

Updates to the BitBar Mobile App Testing Dashboard (https://bitbar.com/blog/updates-to-the-bitbar-mobile-app-testing-dashboard/)

READ MORE

Testing (https://bitbar.com/blog/category/testing/)

Ensuring Mobile Excellence with SmartBear Recap (https://bitbar.com/blog/ensuring-mobile-excellence-with-smartbear-recap/)

READ MORE

Testing (https://bitbar.com/blog/category/testing/)

How to Go from Manual to Automated Mobile Tests (https://bitbar.com/blog/manual-to-automated-mobile-tests/)

READ MORE

FIRST NAME

LAST NAME

COMPANY

WORK EMAIL

PHONE NUMBER

COUNTRY

WHAT'S THE MAIN PAIN POINT YOU WANT TO SOLVE?

CONTACT ME