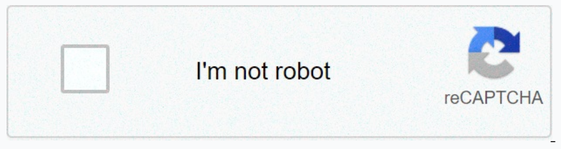

Continue

Capabilities for android appium

In a previous article, we discussed how Desired Capabilities in Selenium help qa teams test their web applications in the desired testing environment on the Selenium Grid.On the same line, Desired Capabilities in Appium help QAs instruct Appium servers about the mobile platform they want to use as a testing environment. Let's understand this in detail with relevant examples. What Capabilities Do Appium Want? The desired capability in Appium refers to a combination of key value pairs encoded in json objects. This key value pair is defined by QAs to request an Appium server for the desired test automation session. Let's consider an example of Desired Capabilities in Appium represented as a JSON object. { platformName: Android, platformVersion: 10.0, deviceName: Samsung Galaxy Note10, automationName: Appium, app: path for the app under test } Using the Desired Capabilities above, the user instructs the driver to start a test automation session for the app on the path mentioned on the Samsung Galaxy Note 10 with Android version 10.0 using Appium.As Appium supports Android and iOS, it has a unique set of Capabilities for both platforms. The table below mentions commonly used capabilities for Android and iOS.General Capabilities in AndroidCapabilities Description ValuesappPackageRepresents java packages of Android Applications desired to be testedcom.example.android.myApp, com.android.settingsappActivityRepresents the desired activity name to launch from the package. This needs to be preceded by '.'. Play Activity, . SettingsappWaitPackageRepresents java package Android application to wait for forcom.example.android.myApp, com.android.settingsbrowserNameRepresents mobile web browser name to be automated. The value must be an empty string if automating the 'Safari' app for iOS One may refer to this link to see the full suite of Desired Capabilities for Android.General Capabilities in iOSCapabilities Description ValuesudidUnique identifier of the connected physical devicee.g 1ae203187fc012gautoAcceptAlertsAccept all iOS alerts automatically if they appear. The default value is FALSETRUE or FALSESafariInitialUrlInitial URL to be loaded. The default URL is redirected to the local.g home page. this link to see a complete set of iOS capabilities. Market trends continue to grow rapidly. Therefore, teams need to ensure that their apps are ready to serve users operating on the latest mobile devices. They need to test their mobile app extensively across devices operating across multiple platforms (Android and iOS). Teams also need to release faster to maintain an edge over competitors. In such a competitive landscape, platforms like BrowserStack are essential to and speed up the testing cycle. BrowserStack's real cloud device provides real Android and iOS devices to teams to run manual and automated testing. One only has to choose to choose to test and configure the Ability to automate appropriate tests. Try Automatic App Testing for FreeOne can easily generate the Appium Desired Capabilities required for iOS and Android devices using Browserstack's Capability Generator. Simply select the OS and device you want to test along with the preferred programming language. Because configuring tests with the ideal Desired Capability forms the basis of most automated app tests, understanding the basics of Appium's Desired Capabilities is essential. This article aims to help foster that understanding, thus giving devs and QAs the tools they need to create an ideal customer experience. Happy early 2019! This is another post from Appium contributor Jonah Stiennon. In the past, I've labeled his contributions to Appium Pro a guest post, but until now, Jonah has officially joined Cloud Grey as a partner! More is to come about Jonah then, but in the meantime get ready to see more of him in this newsletter, on Appium GitHub, and around the world consult with you all. Thanks to Appium @AnnaWyrwal, there are two desired new capabilities available in the latest Appium beta release (see our previous post on how to install Appium beta). Under the right circumstances, this can increase the time it takes to start an Android session. skipDeviceInitialization Available for all Android platforms, the desired skipDeviceInitialization capability can be forwarded with the correct boolean value to bypass installing the io.appium.settings application on the device. This particular app is installed by The Appium Android Driver at the beginning of each test and is used to manage certain settings on the device, such as: wifi/data settings disable the local IME setting animation settings Without the io.appium.settings app, the Android Appium driver cannot automate this function, but if the previous device installed this app by Appium, it does not need to install it anymore. If you know that your device is already in the right state then you can set skipDeviceInitialization to true and pass the time it takes to reinstall. Appium already checks whether the settings app is already installed on the device, but with this capability enabled it even passes the check to see if the app is installed. This is most useful for tests that follow previous successful tests, or tests that run on the same emulator or device over and over again. skipServerInstallation The desired skipServerInstallation capability applies only when using the UiAutomator2 automation method. How the UIAutomator2 driver works, it installs a dedicated server to the device, which listens to test commands from Appium execute it. By default, Appium installs this server to the device at the beginning of each test session. If the skipServerInstallation value is set to true, you can skip the time that it takes to install this server. Of course, without a server on an Appium device can't automate it, automate it, if you know that the server has been installed during a previous trial, you can safely skip this step. WARNING: Be sure to disable this capability if you have updated the version of Appium that you are using, because you want to install the latest version of the UIAutomator2 server. This capability is also most useful for tests that follow previous successful tests, or tests that run on the same emulator or device over and over again. Use appPackage and appActivity to launch existing apps Another major time saver in terms of Android testing using the desired capabilities of appPackage and appActivity instead of app capabilities. We need to tell Appium which apps to test. Usually we use application capabilities, which can be the path to .apk, .apks, or .zip files stored on your computer's file system or stored on public websites. If the app being tested is known to be installed on the device (most likely from a previous trial), the app package name and primary activity can be forwarded instead (using the desired appPackage and appActivity capabilities). Passing the time to download large files or install them on the Android operating system leads to huge savings. As previously discussed capabilities, make sure you install the app again if you're testing a newer version. There's nothing more frustrating than accidentally running a test on an older version of your app! Honorary Mention: ignoreUnimportantViews IgnoreUnimportantViews' desired capabilities are nothing new, and are beyond the scope of this post, but it's worth mentioning as another way to potentially speed up Android automation tests, especially if your test focuses on finding many elements using the XPath finder. Set this to true to speed up Appium's ability to find elements in Android apps. (To learn more about these capabilities and other capabilities used to speed up test execution, see Part 6 of this series to make your Appium testing fast and reliable.) Below is an example of the desired skipDeviceInitialization, skipServerInstallation, and appPackage capabilities used to speed up Android testing. The first test runs without this ability to ensure that the io.appium.settings and UiAutomator2-server applications are installed. The second test runs without bothering to check or install these dependencies or the application being tested on the device. With my not-so-rigorous analysis, skipDeviceInitialization and skipServerInstallation are stored about 1 second per test and using appPackage saved ten seconds per test (mostly because the sample code downloads the app from the internet). import org.junit.Assert; import org.junit.Test; Import import org.openqa.selenium.support.ui.WebDriverWait; import java.io.IOException; import java.net.URL; import io.appium.java_client. AppiumDriver; public class Edition049_Faster_Android_Capabilities { private String APP = = personal AppiumDriver drivers; Private WebDriverWait awaits; public void testInstallingDependencies() throws ioexception cap { DesiredCapabilities = DesiredCapabilities new(); caps.setCapability(platformName, Android); caps.setCapability(deviceName, Android Emulator); caps.setCapability(automationName, UiAutomator2); caps.setCapability(app, APP); driver=AppiumDriver new(new URL(, caps); wait=New WebDriverWait(driver,10); driver.quit(); } test public voidSkippingInstallOfDependencies() throws ioexception cap { DesiredCapabilities = new DesiredCapabilities(); caps.setCapability(platformName, Android); caps.setCapability(deviceName, Android Emulator); caps.setCapability(automationName, UiAutomator2); caps.setCapability(appPackage, io.cloudgrey.the_app); caps.setCapability(appActivity, io.cloudgrey.the_app. Main Activities); caps.setCapability(skipDeviceInitialization, true); caps.setCapability(skipServerInstallation, true); driver=New AppiumDriver (new URL(, caps); wait = New WebDriverWait(driver, 10); driver.quit(); } } }