Testing iOS applications using Appium and JUnit in Java

Overview

In this tutorial, we will create a JUnit Test Case in Java, using the Appium library for automation of iOS applications.

Description

The following automated test is taken from the tutorials for iOS provided by Appium.

> **Please note**
>
> This example is found in the public Github repository in https://github.com/appium/tutorial/tree/master/projects/java_ios. It also provides examples for other languages.

Requirement

- Appium must be running in the machine with iOS SDK.

```
appium
```

The class implementing the automated tests needs to be updated in order to properly set up the IP of the Appium server along with the required iOS version.

AppiumTest.java

```
package appium.tutorial.android.util;

import appium.tutorial.android.page.HomePage;
import com.saucelabs.common.SauceOnDemandAuthentication;
import com.saucelabs.common.SauceOnDemandSessionIdProvider;
import com.saucelabs.junit.SauceOnDemandTestWatcher;
import com.saucelabs.saucerest.SauceREST;
import io.appium.java_client.android.AndroidDriver;
import org.apache.commons.logging.LogFactory;
import org.junit.After;
import org.junit.Before;
import org.junit.Rule;
import org.junit.rules.TestRule;
import org.junit.rules.TestWatcher;
import org.junit.runner.Description;
import org.openqa.selenium.By;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.remote.DesiredCapabilities;

import java.io.File;
import java.net.URL;
import java.nio.file.Paths;
import java.util.Date;
import java.util.concurrent.TimeUnit;

import static appium.tutorial.android.util.Helpers.driver;

public class AppiumTest implements SauceOnDemandSessionIdProvider {

    static {
        // Disable annoying cookie warnings.
        // WARNING: Invalid cookie header
```

```
        LogFactory.getFactory().setAttribute("org.apache.commons.logging.Log", "org.apache.commons.logging.impl.
NoOpLog");
    }

    /** Page object references. Allows using 'home' instead of 'HomePage' **/
    protected HomePage home;

    /** wait wraps Helpers.wait **/
    public static WebElement wait(By locator) {
        return Helpers.wait(locator);
    }

    private boolean runOnSauce = System.getProperty("sauce") != null;

    /** Authenticate to Sauce with environment variables SAUCE_USER_NAME and SAUCE_API_KEY **/
    private SauceOnDemandAuthentication auth = new SauceOnDemandAuthentication();

    /** Report pass/fail to Sauce Labs **/
    // false to silence Sauce connect messages.
    public @Rule
    SauceOnDemandTestWatcher reportToSauce = new SauceOnDemandTestWatcher(this, auth, false);

    @Rule
    public TestRule printTests = new TestWatcher() {
        protected void starting(Description description) {
            System.out.print("  test: " + description.getMethodName());
        }

        protected void finished(Description description) {
            final String session = getSessionId();

            if (session != null) {
                System.out.println(" " + "https://saucelabs.com/tests/" + session);
            } else {
                System.out.println();
            }
        }
    };

    private String sessionId;

    /** Keep the same date prefix to identify job sets. **/
    private static Date date = new Date();

    /** Run before each test **/
    @Before
    public void setUp() throws Exception {
        DesiredCapabilities capabilities = new DesiredCapabilities();
        capabilities.setCapability("appium-version", "1.1.0");
        capabilities.setCapability("platformName", "Android");
        capabilities.setCapability("deviceName", "Android");
        capabilities.setCapability("platformVersion", "7.1");

        // Set job name on Sauce Labs
        capabilities.setCapability("name", "Java Android tutorial " + date);
        String userDir = System.getProperty("user.dir");

        URL serverAddress;
        String localApp = "api.apk";
        if (runOnSauce) {
            String user = auth.getUsername();
            String key = auth.getAccessKey();

            // Upload app to Sauce Labs
            SauceREST rest = new SauceREST(user, key);

            rest.uploadFile(new File(userDir, localApp), localApp);

            capabilities.setCapability("app", "sauce-storage:" + localApp);
            serverAddress = new URL("http://" + user + ":" + key + "@ondemand.saucelabs.com:80/wd/hub");
            driver = new AndroidDriver(serverAddress, capabilities);
```

```
        } else {
            String appPath = Paths.get(userDir, localApp).toAbsolutePath().toString();
            capabilities.setCapability("app", appPath);
            serverAddress = new URL("http://127.0.0.1:4723/wd/hub");
            driver = new AndroidDriver(serverAddress, capabilities);
        }

        sessionId = driver.getSessionId().toString();

        driver.manage().timeouts().implicitlyWait(30, TimeUnit.SECONDS);
        Helpers.init(driver, serverAddress);
    }

    /** Run after each test **/
    @After
    public void tearDown() throws Exception {
        if (driver != null) driver.quit();
    }

    /** If we're not on Sauce then return null otherwise SauceOnDemandTestWatcher will error. **/
    public String getSessionId() {
        return runOnSauce ? sessionId : null;
    }
}
```

The sample project contains two classes with the automated Tests. Below is one of them:

AutomatingASimpleActionTest.java

```
package appium.tutorial.ios;

import appium.tutorial.ios.util.AppiumTest;
import org.openqa.selenium.WebElement;

import java.util.ArrayList;
import java.util.List;

import static appium.tutorial.ios.util.Helpers.*;

public class AutomatingASimpleActionTest extends AppiumTest {

    @org.junit.Test
    public void one() throws Exception {
        text("Various uses of UIButton").click();
        text_exact("Buttons");
    }

    @org.junit.Test
    public void two() throws Exception {
        wait(for_text("Various uses of UIButton")).click();
        wait(for_text_exact("Buttons"));
    }

    @org.junit.Test
    public void three() throws Exception {
        WebElement cell_1 = wait(for_text(2));
        String page_title = cell_1.getAttribute("name").split(",")[0];

        cell_1.click();
        wait(for_text_exact(page_title));
    }

    @org.junit.Test
    public void four() throws Exception {
        List<String> cell_names = new ArrayList<String>();

        for (WebElement cell : tags("TableCell")) {
            cell_names.add(cell.getAttribute("name"));
        }

        for (String name : cell_names) {
            wait(for_text_exact(name)).click();
            wait(for_text_exact(name.split(",")[0]));
            back();
        }
    }
}
```

Actually, the above class and others, such as Helpers.java, had to be changed due to updates introduced by Apple in the automation API with XCUITest.

Helpers.java

```
package appium.tutorial.ios.util;

import io.appium.java_client.AppiumDriver;
import io.appium.java_client.MobileElement;
import org.openqa.selenium.By;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.remote.RemoteWebElement;
import org.openqa.selenium.support.ui.ExpectedConditions;
import org.openqa.selenium.support.ui.WebDriverWait;
```

```
import java.util.ArrayList;
import java.util.List;

public abstract class Helpers {

  private static AppiumDriver driver;
  private static WebDriverWait driverWait;

  /**
   * Initialize the webdriver. Must be called before using any helper methods. *
   */
  public static void init(AppiumDriver webDriver) {
    driver = webDriver;
    int timeoutInSeconds = 60;
    // must wait at least 60 seconds for running on Sauce.
    // waiting for 30 seconds works locally however it fails on Sauce.
    driverWait = new WebDriverWait(webDriver, timeoutInSeconds);
  }

  /**
   * Wrap WebElement in MobileElement *
   */
  private static MobileElement w(WebElement element) {
    return new MobileElement((RemoteWebElement) element, driver);
  }

  /**
   * Wrap WebElement in MobileElement *
   */
  private static List<MobileElement> w(List<WebElement> elements) {
    List list = new ArrayList(elements.size());
    for (WebElement element : elements) {
      list.add(w(element));
    }

    return list;
  }

  /**
   * Return an element by locator *
   */
  public static MobileElement element(By locator) {
    return w(driver.findElement(locator));
  }

  /**
   * Return a list of elements by locator *
   */
  public static List<MobileElement> elements(By locator) {
    return w(driver.findElements(locator));
  }

  /**
   * Press the back button *
   */
  public static void back() {
    driver.navigate().back();
  }

  /**
   * Return a list of elements by tag name *
   */
  public static List<MobileElement> tags(String tagName) {
    return elements(for_tags(tagName));
  }

  /**
   * Return a tag name locator *
   */
  public static By for_tags(String tagName) {
    return By.className(tagName);
```

```
}

/**
 * Return a static text element by xpath index *
 */
public static MobileElement text(int xpathIndex) {
  return element(for_text(xpathIndex));
}

/**
 * Return a static text locator by xpath index *
 */
public static By for_text(int xpathIndex) {
  //return By.xpath("//UIAStaticText[" + xpathIndex + "]");
  return By.xpath("//XCUIElementTypeStaticText[" + xpathIndex + "]");
}

/**
 * Return a static text element that contains text *
 */
public static MobileElement text(String text) {
  return element(for_text(text));
}

/**
 * Return a static text locator that contains text *
 */
public static By for_text(String text) {
  String up = text.toUpperCase();
  String down = text.toLowerCase();

  //return By.xpath("//UIAStaticText[@visible=\"true\" and (contains(translate(@name,\"" + up
  return By.xpath("//XCUIElementTypeStaticText[@visible=\"true\" and (contains(translate(@name,\"" + up
      + "\",\"" + down + "\"), \"" + down + "\") or contains(translate(@hint,\"" + up
      + "\",\"" + down + "\"), \"" + down + "\") or contains(translate(@label,\"" + up
      + "\",\"" + down + "\"), \"" + down + "\") or contains(translate(@value,\"" + up
      + "\",\"" + down + "\"), \"" + down + "\"))]");
}

/**
 * Return a static text element by exact text *
 */
public static MobileElement text_exact(String text) {
  return element(for_text_exact(text));
}

/**
 * Return a static text locator by exact text *
 */
public static By for_text_exact(String text) {

  //XCUIElementTypeStaticText
  return By.xpath("//XCUIElementTypeStaticText[@visible=\"true\" and (@name=\"" + text
  //return By.xpath("//UIAStaticText[@visible=\"true\" and (@name=\"" + text
      + "\" or @hint=\"" + text + "\" or @label=\"" + text
      + "\" or @value=\"" + text + "\")]");
}

/**
 * Wait 30 seconds for locator to find an element *
 */
public static MobileElement wait(By locator) {
  return w(driverWait.until(ExpectedConditions.visibilityOfElementLocated(locator)));
}

/**
 * Wait 60 seconds for locator to find all elements *
 */
public static List<MobileElement> waitAll(By locator) {
  return w(driverWait.until(ExpectedConditions.visibilityOfAllElementsLocatedBy(locator)));
}
```

```
}
```

Tests can be run using Maven.

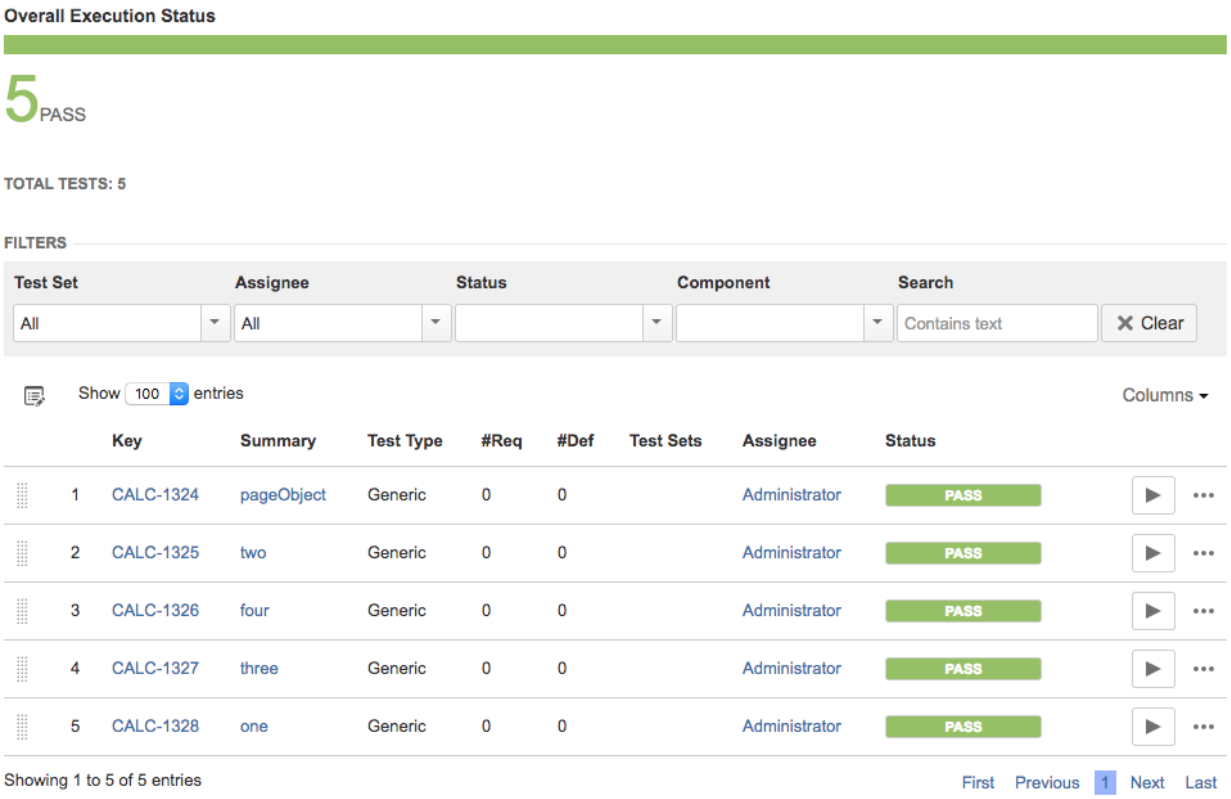
```
mvn clean test
```

Since the previous command generates multiple JUnit XML files, we may need to merge them into a single XML file so it can be submitted into a Test Execution more easily. That can be achieved by using the junit-merge utility.

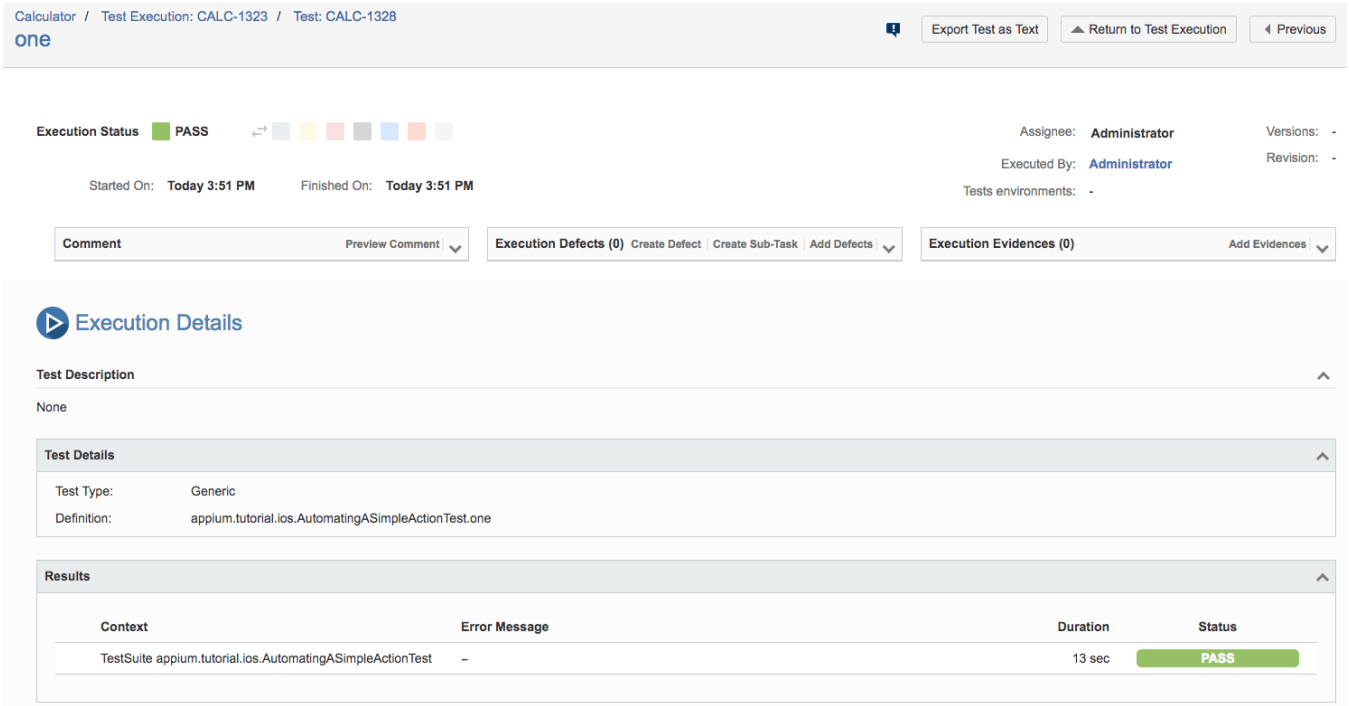
```
junit-merge  -o results.xml -d target/surefire-reports/
```

After successfully running the Test cases and generating the aggregated JUnit XML report (e.g., results.xml), it can be imported to Xray (either by the REST API or through the **Import Execution Results** action within the Test Execution).

Each JUnit's Test Case is mapped to a Generic Test in Jira, and the **Generic Test Definition** field contains the name of the package, the class and the method name that implements the Test Case. The summary of each Test issue is filled out with the name of the method corresponding to the JUnit Test.



The Execution Details of the Generic Test contains information about the Test Suite, which in this case corresponds to the Test Case class, including its namespace.



References

- https://github.com/appium/tutorial/tree/master/projects/java_ios
- http://appium.io
- https://www.npmjs.com/package/junit-merge