Mobile Application Testing with Camera Image Injection

Does your mobile app have the ability to take images and then process or store them within the app (e.g., scanning a bank check, taking a social media selfie)? Camera Image Injection – also known as *camera mocking* – is a Sauce Labs Real Device Cloud (RDC) feature that simulates taking a picture through your mobile device app, allowing you to test the app's camera-based functionality and deliver the best possible user experience.

To mimic camera behavior when testing your app, you'll provide the app with an image that mocks the use of the camera. During a Sauce Labs live test, you'll be prompted to upload a photo that will be fed to your app, rather than using your device camera to take the photo. For automated tests, you'll pass an image to the image injection endpoint. Image Injection intercepts the camera input and replaces the camera output with the image file via the camera APIs. When the app requests an image from the device's camera, we inject your uploaded image into the response (the app doesn't know the difference).

See the following sections for more information:

What You'll Need

You'll need to upload your app to Sauce Labs prior to testing. For instructions, see Uploading your Application to Real Devices using the User Interface.

Using Camera Image Injection

Camera Image Injection is a core feature built into our RDC functionality and available for use with public and private devices. Your mobile app accesses the camera and instead of getting back the picture of the device camera, it'll retrieve your uploaded image for your test. You employ the built-in device camera in your live and automated testing and perform test cases that require taking images with any of the device cameras.

System Requirements

See the topics under Mobile Application Testing for RDC system requirements.

Key Specs

Supported

- All iOS and Android devices available in the RDC
- Front-facing and rear-facing system device cameras
- Image file sizes up to 5MB
- JPG, JPEG, PNG image file formats.
- For Android devices, there are multiple ways to capture an image, as described in the Android Camera API developer documentation. We support the following:
    - ACTION_IMAGE_CAPTURE Intent: opens the system camera and notifies the calling app gets when the image is taken
    - camera2 API: everything is configured and handled from within the app
    - Camera API (deprecated) (partially supported): As with camera2, everything is handled in the app itself. QR Code readers often use Camera#setPreviewCallback. We pass the injected image to this method, but the rest of this deprecated API is not supported. UI Elements will not likely display the injected image.
- For iOS devices, the camera can be configured with different outputs. We support the following:
    - AVCapturePhotoOutput: for capturing still images. The results are received via the AVCapturePhotoCaptureDelegate and the method captureOutput:didFinishProcessingPhoto:error: The other methods in this delegate are either deprecated or handle live photos, which we don't support.
    - AVCaptureMetadataOutput: for reading QR-Codes. The QR Codes are passed to the app via captureOutput:didOutputMetadataObjects:fromConnection:. We are detecting the AVMetadataMachineReadableCodeObject and QR Codes are part of that.

Not Supported

- Ephemeral apps (i.e., app with temporary messages that disappear after a certain timeframe)
- Testing with Emulators, Simulators

Common Use Cases & Examples

Here are some common use cases ideal for implementing Camera Image Injection in your tests:

Scanning a Check for a Mobile Banking App Deposit

Many mobile banking apps allow customers to deposit checks using their smartphone. The customer takes and uploads an image of their physical paper check, and the image is then submitted to the bank for processing.

Using a QR Code to Link to an Embedded URL

QR codes are often used as a way to bridge print media to digital. Users take a photo with a QR code reader app, the app scans the code and directs them to an embedded URL.

For use cases that involve scanning barcodes or QR codes, your own application in testing must do the actual image processing. Camera Image Injection passes your uploaded image directly to your app as if it came from the device camera; it does not do any processing.

Taking a Selfie for a User Profile Photo

This could be taking a selfie or uploading a picture for apps that require a user profile photo. You can use Camera Image Injection to test image formats and sizes.

Taking an Image to Store or Send via Mobile App

Whether it’s a social media app or photo sharing, this use case can encompass many different scenarios. In its simplest form, it could be taking pictures from the front or back camera to send and/or archive within the app.

Live Testing with Camera Image Injection

1. Launch a test session by logging into Sauce Labs and going to **Live** > **Mobile App**.
2. Click the checkbox next to **Enable Image Injection** in the top toolbar. It's important that you do this *before* selecting a mobile device.

   NOTE: This checkbox will populate once your CSM grants you access to image injection.
3. Select the mobile device you wish to test your app on, either by clicking directly on a device or the Launch button on the device. This will start the test session with your app.
4. When you reach the stage in your test session where you'd like to take an image with the app camera, go to the right toolbar and click the **Camera** icon to open Camera Image Injection.
5. Click **Choose Image** to upload your image. Be sure to do this *before* opening your app camera. If successful, you'll see a notification at the top: "The image has been loaded successfully."
6. Open the camera inside of your app.
7. The device will show your uploaded image in the app as if the image was taken by the device camera. The image will stay in the device's memory for your test session; if you would go back to the camera during your test session, it will still be displayed. To use a different image, you'd need to upload that and reopen the device camera.

Video: Running iOS Mobile Device Tests

This video illustrates how to use image injection in an iOS live test:

Your browser does not support the HTML5 video element

Video: Running QR Code Tests

When injecting an image with a QR Code or barcode, the image size in your preview may exceed the boundaries of the target scanner area, which would prevent your app from reading the code. In this scenario, you'd need to add padding to your uploaded image so that when it's scaled to full-screen, the QR Code will fit inside the scanning area limits and can be processed.

Your browser does not support the HTML5 video element

Automated Appium Testing with Camera Image Injection

In your automated test script, you'll need to input the desired capabilities specific to Camera Image Injection (see below code snippets). The code will execute your image upload and opening of the device camera.

1. First, you'll need add the camera instrumentation desired capability command, `sauceLabsImageInjectionEnabled`, to your test script. This capability enables image injection functionality.

Webdriver.io example

```
exports.config = {
  //...
  capabilities: [
    {
      deviceName: 'Samsung Galaxy S10',
      platformName: 'Android',
      platformVersion: '10',
      automationName: 'UiAutomator2',
      // Enable image-injection on RDC
      sauceLabsImageInjectionEnabled: true,
    },
  ]
  //...
}
```

Java example

```
var desiredCapabilities = new DesiredCapabilities();
desiredCapabilities.setCapability("deviceName", "Samsung Galaxy S10");
desiredCapabilities.setCapability("platformVersionName", "10");
...
// Enable image-injection on RDC
desiredCapabilities.setCapability("sauceLabsImageInjectionEnabled", true);
```

2. Next, add the image injection method to your test script by providing a file path to your image. When you execute your test, these code snippets will call the endpoint and pass the image to the app for further processing or for other use.

   You can change the image by sending the custom command with a different image. Note that your image file path must be converted to base64 encoding.

Webdriver.io example

```
const {readFileSync} = require('fs');
const {join} = require('path');

// Read the file from your project and transform it to a base64 string
const qrCodeImage = readFileSync(join(process.cwd(), 'assets/qr-code.png'), 'base64');

// Provide the transformed image to the device
driver.execute(`sauce:inject-image=${qrCodeImage}`);
```

Java example

```
import java.util.Base64;
import static org.apache.commons.io.IOUtils.toByteArray;

// Read the file from the classpath and transform it to a base64 string
String qrCodeImage = Base64.getEncoder().encodeToString(
  toByteArray(getClass().getResourceAsStream("qr-code.png")));

// Provide the transformed image to the device
((JavascriptExecutor)driver).executeScript("sauce:inject-image=" + qrCodeImage);
```

3. But you only really start image injection when you put the second code snippet into your code in one or more places to call the end point and pass the image.

Troubleshooting

Here are some common errors you may see in the course of testing with Camera Image Injection and how to resolve them.

Image injection failed

This error is displayed when you attempt to inject your image before the app fully loads during your initial test session startup. You must wait until your app has fully loaded prior to injecting your image.

Image injection is not enabled for the application

This error is displayed due to one or more of these reasons:

- **Enable Image Injection** checkbox is not checked; this needs to be checked
- For Android tests, the debuggable flag (`android:debuggable="true"`) is missing from your application's manifest file

Additional Resources

- Sauce Labs Camera Image Injection examples on GitHub
- Sample Mobile App you can use to try out Camera Image Injection
- Android Camera API developer documentation
- For support in beta, please reach out to your CSM/SE or Sauce Labs Support