SOASTA

TouchTest™ Appcelerator Android Tutorial

SOASTA TouchTest™ for Appcelerator Android Tutorial

Table of Contents

SOASTA

Why Mobile App Testing?

CloudTest 's new TouchTest™ technology delivers, for the first time, complete functional test automation for continuous multi-touch, gesture-based mobile applications. TouchTest™ technology delivers fast, precision functional testing while increasing the stability of automated tests across releases.

CloudTest® controls mobile devices through a lightweight software agent, SOASTA TouchTest Agent, and accesses them using their IP address. Devices can be dedicated to testing in the lab, used as part of a short external test, or crowd-sourced as part of a high volume, globally distributed test.

Support is provided for recording user actions on:

- Any iOS 5.0 or later device including iPhone, iPad, iPod Touch, as well as Simulators. There is no need to jailbreak the Android Device and the device can be untethered.

Beta Support is provided for recording user actions on:

- Any Android 2.2 device or later.

CloudTest® Basics

SOASTA provides fast, effective performance, load and functional testing of any modern Web application, Web service, or mobile application in a lab, staging or production environment using a unique multi-track user interface. The CloudTest® platform can utilize both public and private cloud resources to assure any web or mobile application won't fail under peak user traffic.

The CloudTest® Central tab lists all primary features, organized by sections. Central's first section—highlighted in light blue—contains the Welcome page, and the primary test building tools.

The **Composition** is the test itself as presented in the **Composition Editor**, and contains one or more **Clips** arranged on **Tracks** and governed by user-specified sequence and tempo. The Composition Editor is a player, debugger, as well as the dashboard where results are analyzed.

The **Clip** is the basic building block of a test as presented in the **Clip Editor** and has a **Target** such as HTTP traffic for a site, or a browser UI (web site); or in the case of TouchTest™, a mobile app. A clip can contain messages, actions, scripts, as well as

delays and checkpoints—all of which can be organized into containers (chains, groups, pages, transactions, if-then-else, and switch)—and parameterized as required.

TouchTest™ clips are recorded directly from the mobile app and added to the Clip Editor (as described in this tutorial) as you perform them on the mobile device. A clip can be thought of as a visual script that is composed of a series of timed or sequenced events, which correspond to gestures performed on the mobile device. It can contain messages, browser or app actions, and scripts, as well as delays and checkpoints—all of which can be organized into containers (i.e. groups, chains, transactions, etc.).

What Does Touch Test Record?

TouchTest™ records the details of actual gestures and events that iOS invokes on th e app that is tested. These gestures and events are represented within the Clip Editor as App Actions. Precision recording captures and plays back all continuous to uch gestures including pan, pinch, zoom and scroll.

Each gesture you perform on a TouchTest-enabled device is precisely, and automati cally, added to the test clip as an App Action. Like any clip element within CloudTest ®, App Actions have inputs and outputs, as well as properties, waits, and validations that can be parameterized. Additionally, an App Action can be added to any contain ers (e.g. transactions, groups, etc.).

TouchTest™ clips are recorded directly from the mobile app and added to the Clip as you perform them on the mobile device. A clip can be thought of as a visual script that is composed of a series of timed or sequenced events, which correspond to gestures performed on the mobile device. Planning a TouchTest™

As a general guideline, your test should account for all the factors of the mobile app(s) you want to test, and include one, or, as many viable test cases as it will take to arrive at a good mobile app test case.

Once a test case is determined for a given mobile app it can be easily captured. The test designer can move quickly from recording to defining validations and other test details for those captured app actions. In the context of mobile testing, planning will also take into account the following factors:

- **The types of app actions to perform**

The test designer will consider the types of app actions that make up a test case for the given mobile app. These app actions should then be performed during recording.

- **The timing of app actions**

In addition to TouchTest's built-in detection of the duration of gestures, CloudTest® provides an additional set of Waits, which allow the tester to gain control of the pace within a test.

- **The validation of tests**

Verifying the behavior of a mobile app is another important step in successful testing. After each app action is recorded, the test designer can add as many validations as needed by picking from among built-in Verify commands.

- **The number of devices and their locations**

Typically, a single test clip defines a single test case that can be run on multiple tracks or devices. However, tests of great complexity can be quickly devised by introducing multiple test cases, multiple devices, and multiple repeats—possibly in tandem with geographic location. Complex mobile app tests can be easily built utilizing one or all of these capabilities.

Adding TouchTest™ to an Appcelerator App

This section describes the steps necessary to make an existing Appcelerator project for Android TouchTestable. This tutorial uses the Appcelerator sample app, KitchenSink. However, you can substitute any Appcelerator app and easily follow along. Please refer to the basic TouchTest Android Tutorial for the latest Android-specific prerequisites.

Appcelerator Prerequisites for Android

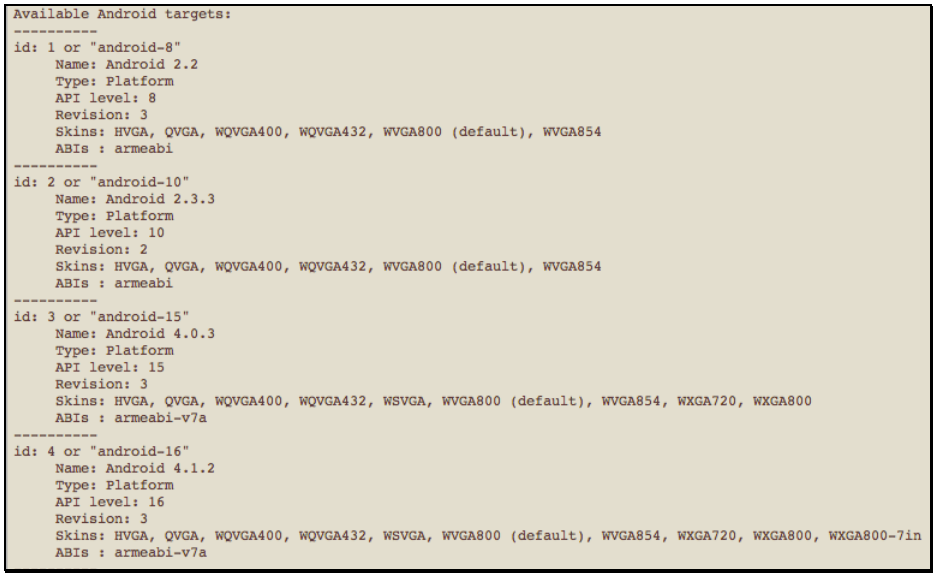
The following steps assume a minimal familiarity with Appcelerator and the following prerequisites:

- Titanium Studio is installed with the Android SDK (the minimum supported Android version is 2.3.3. (Gingerbread).

TIP: If you're unsure of the installed Android SDK(s) on your system, use the `android-sdks/tools/android` command to list them. For example, from your android-sdks folder (or if it's in your path) use:

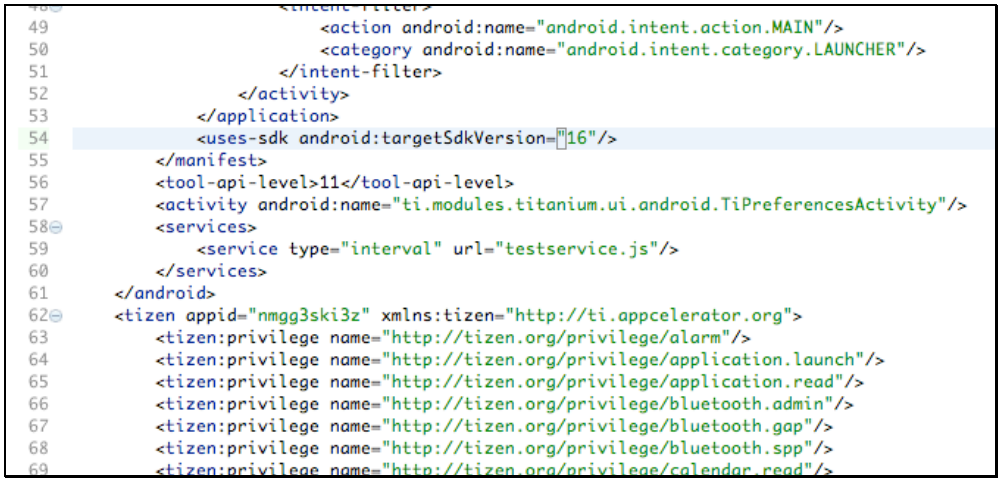
```
android list target
```

The android command with the list target flag gets a list of the installed Android SDK(s) on your system.



```
Available Android targets:
----------
id: 1 or "android-8"
     Name: Android 2.2
     Type: Platform
     API level: 8
     Revision: 3
     Skins: HVGA, QVGA, WQVGA400, WQVGA432, WVGA800 (default), WVGA854
     ABIs : armeabi
----------
id: 2 or "android-10"
     Name: Android 2.3.3
     Type: Platform
     API level: 10
     Revision: 2
     Skins: HVGA, QVGA, WQVGA400, WQVGA432, WVGA800 (default), WVGA854
     ABIs : armeabi
----------
id: 3 or "android-15"
     Name: Android 4.0.3
     Type: Platform
     API level: 15
     Revision: 3
     Skins: HVGA, QVGA, WQVGA400, WQVGA432, WSVGA, WVGA800 (default), WVGA854, WXGA720, WXGA800
     ABIs : armeabi-v7a
----------
id: 4 or "android-16"
     Name: Android 4.1.2
     Type: Platform
     API level: 16
     Revision: 3
     Skins: HVGA, QVGA, WQVGA400, WQVGA432, WSVGA, WVGA800 (default), WVGA854, WXGA720, WXGA800, WXGA800-7in
     ABIs : armeabi-v7a
```

Then, ensure that your project's tiapp.xml is using the SDK version id number that you intend for it to use and that the SDK version is among those installed.



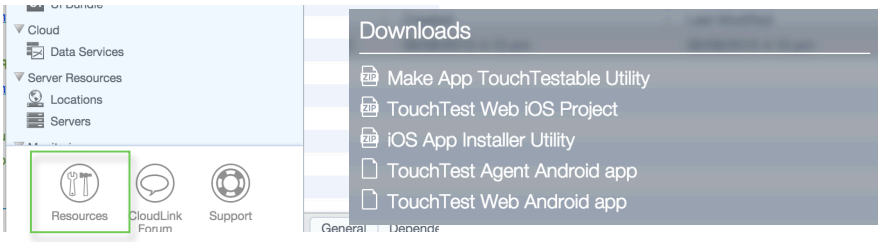
- Titanium SDK version 2.1.3 or greater is installed.
- An Appcelerator application or sample app, such as KitchenSink, is available to be made TouchTestable.
- The developer should note the Titanium Studio SDK path before proceeding with the steps under "Using the MakeAppTouchTestable Utility" below.

  This is either ~/Library/Application Support/Titanium/mobilesdk/osx or ~/Library/Application Support/Titanium/mobilesdk/osx with the specific SDK folder appended at the end.
- The developer should also note the Android SDK location for use with the MATT `androidsdk` flag where required (for signing, etc.).

TouchTest Prerequisites

The following prerequisites are required to make an Android project TouchTestable:

- Download and unarchive the MakeAppTouchTestable utility from Resources > Downloads.

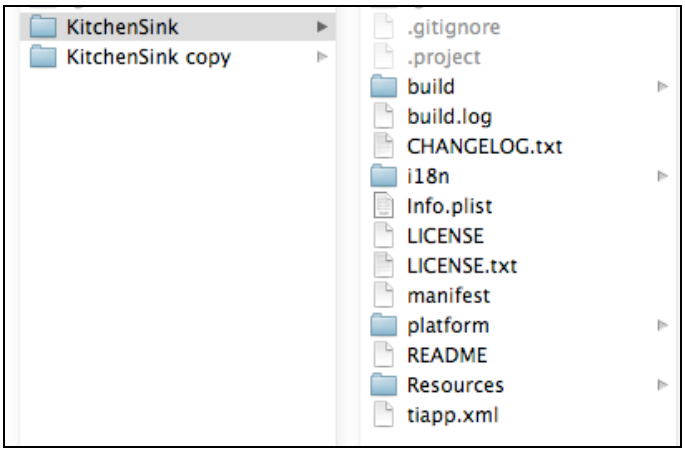


- You will need a mobile app to follow the tutorial, or clone the example Appcelerator KitchenSink project using GitHub.

  --Or—

- You can also download the KitchenSink project archive and unzip it into your repository folder.

Once downloaded, unarchive the project's components prior to running the utility. The unarchived project folders are shown below.



This archive contains the following:

- MakeAppTouchTestable.jar utility is the script that will make the necessary modifications to a project, APP bundle folder, or IPA, as well as to create a mobile app object in your CloudTest® instance.

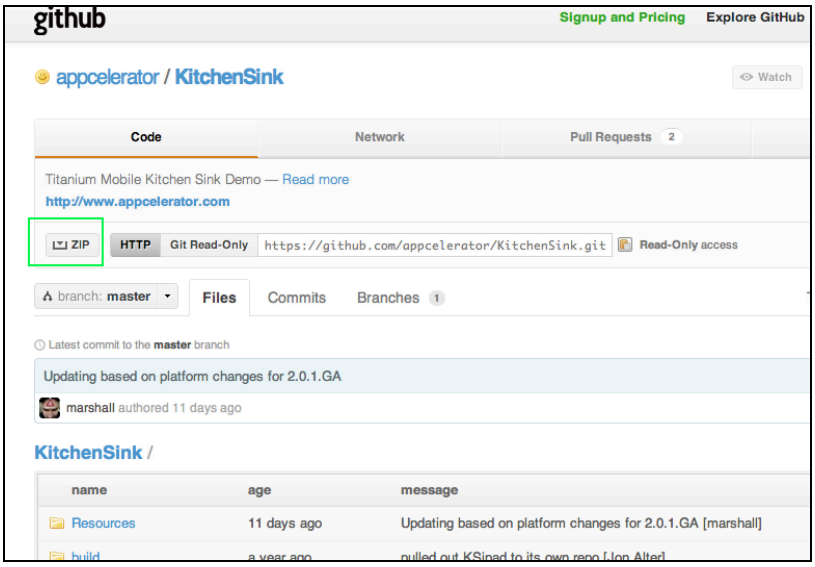
TIP: Consider duplicating the project, as shown above, using one sample project to inspect the changes that the utility will make while the other sample project remains untouched.

Importing and Preparing the Project for Android

Use the following steps to import the sample KitchenSink app used in this tutorial into Titanium Studio.

- Download the KitchenSink project archive into your working folder and unzip it.

- This can be done using the GitHub app, or:



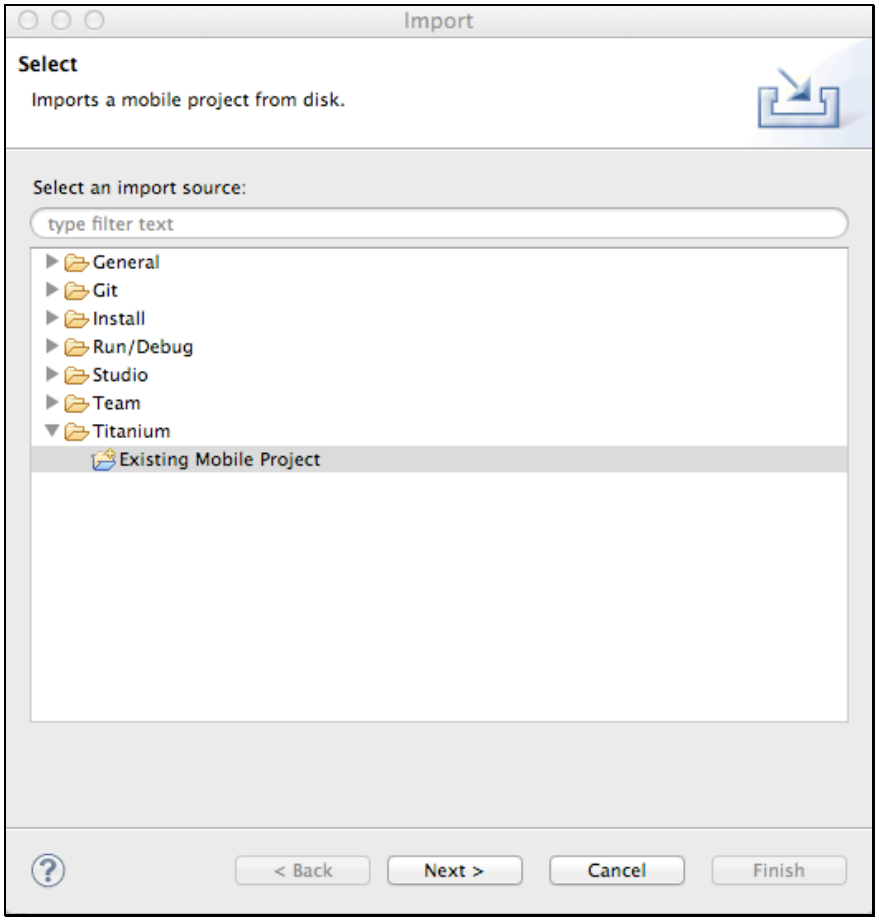
- By using the git command from the command line:

```
git clone https://github.com/appcelerator/KitchenSink
```

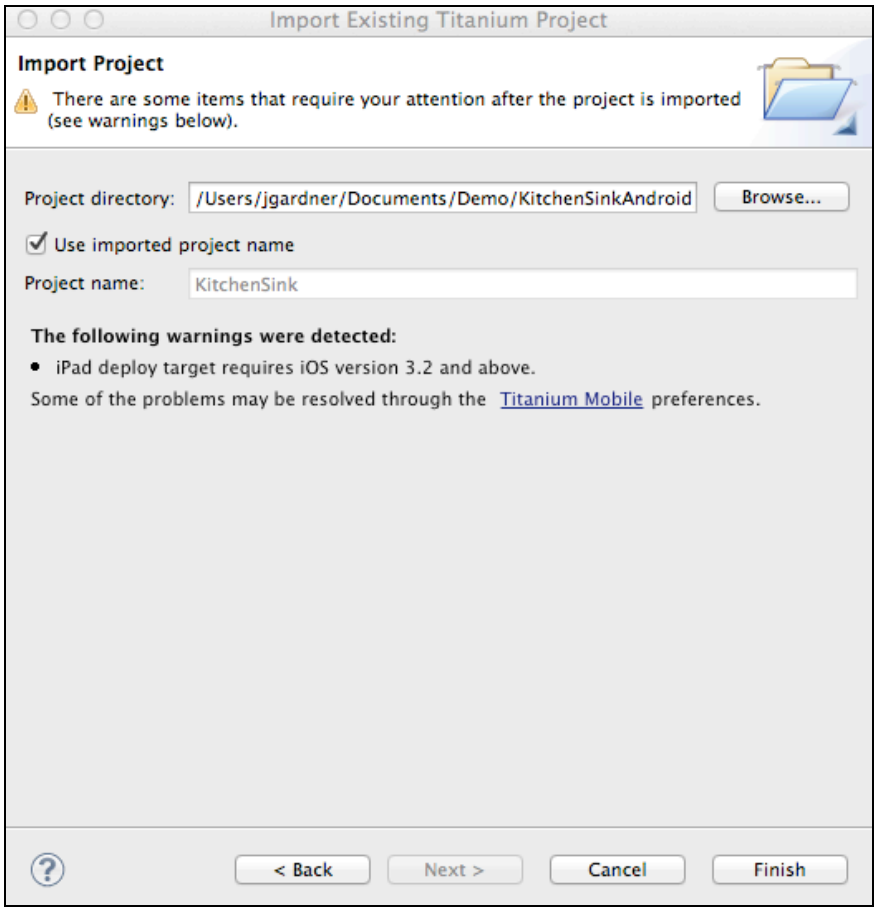
1. Once you have the project (or some other one) imported, launch Titanium Studio and login.

2. Click File > Import.

The Import box appears. Choose Titanium > Existing Mobile Project.

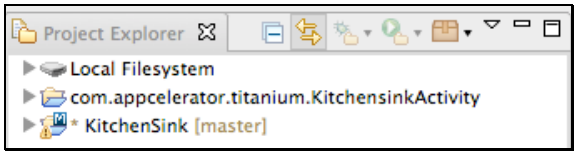


3. Click Next, and identify the folder to use as the Project directory.



TIP: If you duplicated the sample project you can also import it here (by assigning a different project name.

The KitchenSink project is created and appears in the Project Explorer list in Titanium Studio.



4. Exit Titanium Studio before proceeding with the next section.

Using the MakeAppTouchTestable Utility

As noted in the prerequisites above, TouchTest™ uses the MakeAppTouchTestable Utility, which is downloaded from the CloudTest, Welcome page, to modify the Android project or the compiled APK.

> **Note:** The CloudTest user specified to run the MakeAppTouchTestable utility must be a user with Mobile Device Administrator rights or a CloudTest Lite user.

Static vs. Dynamic Instrumentation

The MATT utility supports two instrumentation methods: static and dynamic.

- Dynamic instrumentation occurs when MATT instruments a compiled file (i.e. an APK file). This method requires that you compile your Android project first to create an APK, after which it can be instrumented using SOASTA 51.07 or later (TouchTest 7040.58). Dynamic instrumentation is available for all supported Android versions.

- Static instrumentation occurs when MATT instruments an Android project. Static instrumentation is available in all TouchTest releases and for all supported Android versions.

Making an APK TouchTestable (Dynamic)

This section presumes that the APK file has already been compiled. This can be done using any method common to your organization (using Ant, for example). Do so at this time (without applying the MATT command) to proceed using the following steps.

1. On the command line, navigate to the MakeAppTouchTestable folder you created above.

- For example, in Windows Command Prompt,
  `cd C:\Documents\MakeAppTouchTestable`

- For example in Mac OS X Terminal,
  `cd ~/Documents/Demo/MakeAppTouchTestable`

2. Next, run the utility on the KitchenSink APK using your own modified version of the MakeAppTouchTestable command below

   For Mac OS X:

```
sh MakeAppTouchTestable/bin/MakeAppTouchTestable -apk <KitchenSink
APK>  -url <CloudTest URL> -username <CloudTest user name> -password
<CloudTest password>
```

For Windows:

```
C:\Users\<user>\MakeAppTouchTestable>sh
MakeAppTouchTestable/bin/MakeAppTouchTestable
-apk <KitchenSink APK>  -url <CloudTest URL> -username <CloudTest user
name> -password <CloudTest password>
```

TIP: Copy the above command into a text file to build your own command.

Where:

- `<KitchenSink APK>` is the path to the KitchenSink (or other) APK file. As we noted above, our example path under Mac OS X was:

  `~/Documents/Demo/KitchenSink`

- `<CloudTest URL>` is the CloudTest Lite or CloudTest instance in use.

Here is a complete Mac OS X example:

```
sh MakeAppTouchTestable/bin/MakeAppTouchTestable -apk
~/Documents/Demo/KitchenSink/build/android/bin/KitchenSink.apk
-androidsdk ~/Development/android-sdk-macosx -url
http://10.0.1.9/concerto -username bob@acme.com -password secret
```

Here is a complete Windows example:

```
C:\Users\<user>\MakeAppTouchTestable>sh
MakeAppTouchTestable/bin/MakeAppTouchTestable
-apk "C:\My Documents\Demo\KitchenSink\build\android\bin\KitchenSink.apk"
-androidsdk "C:\My Documents\Development\android-sdk" -url
http://10.0.1.9/concerto -username bob@acme.com -password secret
```

- Optionally, you can manually specify an additional `launchurl` flag, being sure to specify the correct URL syntax (shown below).

  This flag is used in the CloudTest repository to represent your mobile app and in the compiled app. For Eclipse projects, this setting originates in the AndroidManifest.xml. Whether you create the TouchTestable Android app using the project or apk MATT parameter—this launchurl must match for testing to succeed.

For example,

```
-launchURL “KitchenSink://key1=value1&key2=value2&key3=value3”
```

MakeAppTouchTestable will configure your project, and create a new Mobile App object in the CloudTest server repository. The Mobile App object created will have the auto-created URL Scheme in its Launch URL field. The following text output appears in Terminal:

```
Mobile App Object "KitchenSink" representing your Application "KitchenSink"
has been created in CloudTest Repository.
```

The Mobile App object created will have the auto-created scheme found in tiapp.xml unless otherwise specified.

You will see a message similar to the following:

```
Will create the launch url: touchtest-e4eedd67-4ea9-495a-be57-
2d34eaafc510://
```

Using Additional MATT APK Flags

You can use MATT to add keystore, keypass, and storepass flags to sign the dynamically instrumented APK file.

Use the following MATT optional APK parameters

- -keystore <keystorepath> - Path of the keystore to be used to sign the APK file..
- -storepass <keystorepassword> - Password of the keystore to be used to sign the APK file.
- -keypass <privatekeypassword> - Password of the private key (if different than the keystore password) to be used to sign the APK file.

For more about using additional MATT parameters, from the command line, use:

java –jar MakeAppTouchTestable.jar - help

Making a Titanium Project TouchTestable (Static)

In the remainder of this tutorial, we will add TouchTest™ to the sample project *KitchenSink*. You can use your own project instead.

- Next, in Terminal, navigate to the MakeAppTouchTestable folder you created above.

- For example, `cd Downloads/MakeAppTouchTestable`.
- From the MakeAppTouchTestable folder, run

```
sh MakeAppTouchTestable/bin/MakeAppTouchTestable -project <Android
project directory> - -url <CloudTest URL> -username <CloudTest user
name> -password <CloudTest password>
```

where:

- `<Android project file>` is the path to the root folder of your project

Here is a complete example:

```
sh MakeAppTouchTestable/bin/MakeAppTouchTestable -project
~/Documents/Demo/KitchenSink
-url http://10.0.1.6/concerto -username SOASTA_DOC -password secret
```

- Optionally, add the `–launchurl` flag, being sure to specify the correct URL syntax. This setting originates in the AndroidManifest.xml file in Eclipse projects and tiapp.xml in Titanium projects. The launch URL in the compiled app and in the CloudTest, Mobile App, launch URL field must match for testing to occur.

  For example,

```
-launchurl KitchenSink://key1=value1&key2=value2&key3=value3
```

MakeAppTouchTestable will configure your project, and create a new Mobile App object in the CloudTest server repository. The Mobile App object created will have the auto-created URL Scheme in its Launch URL field. The following text output appears in Terminal:

```
Mobile App Object "KitchenSink" representing your Application "KitchenSink"
has been created in CloudTest Repository.
```

The Mobile App object created will have the auto-created scheme found in AndroidManifest.xml unless otherwise specified. You will see a message similar to the following:

```
Will create the launch url: touchtest-e4eedd67-4ea9-495a-be57-
2d34eaafc510://
```

Using the Optional –titaniumsdk Flag (Dynamic/Static)

In most cases, the MakeAppTouchTestable utility will correctly detect the location of the Titanium SDK on Android. However, for those exceptions that may occur—such as for non-standard locations—the -titaniumsdk flag is provided. This flag is applicable only to Android projects.

To use this flag, specify the path of the Titanium SDK. For example:

```
/Library/Application Support/Titanium/mobilesdk/osx/
```

or

```
~/Library/Application Support/Titanium/mobilesdk/osx/
```

with the flavor or SDK appended at the end of the path. For example:

```
 "/Library/Application Support/Titanium/mobilesdk/osx/2.1.3.GA"
```

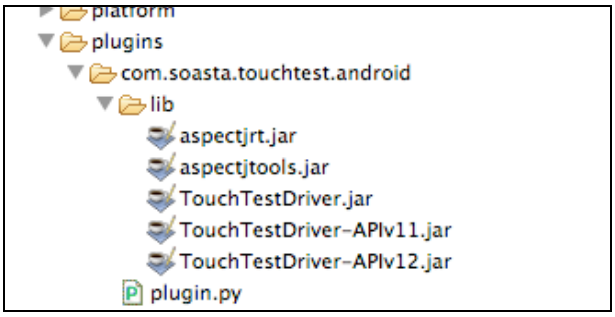
Putting it all together we get:

```
-titaniumsdk "/Library/Application Support/Titanium/mobilesdk/osx/2.1.3.GA2"
```

Inspecting Changes to the Titanium Project (Static)

The MakeAppTouchTestable utility changes your Titanium project in the following ways:

- Plugin files are added to the plugins folder, in the com.soasta.touchtest.android folder.



- The remaining changes made by MakeAppTouchTestable to the project are located in the tiapp.xml file:
    - A plugin statement has been added to use the plugin to utilize the plugin files (shown above):

```
    <plugins>
        <plugin>com.soasta.touchtest.android</plugin>
    </plugins>
</ti:app>
```

    - An intent-filter section has been added

```
        <supports-screens android:anyDensity="false"/>
        <application>
            <activity android:alwaysRetainTaskState="true" android:configChanges="keyboar
                <intent-filter>
                    <action android:name="android.intent.action.MAIN"/>
                    <category android:name="android.intent.category.LAUNCHER"/>
                </intent-filter>
                <intent-filter>
                <data android:scheme="touchtest-302e5eb4-872d-4c16-8099-db0a9ef0aa15"/>
                <action android:name="android.intent.action.VIEW"/>
                <category android:name="android.intent.category.BROWSABLE"/>
                <category android:name="android.intent.category.DEFAULT"/>
            </intent-filter>
        </activity>
                    <service android:enabled="true" android:exported="false" android:name
```

    - A new TouchTest Service has been added

```
tent-filter>
ta android:scheme="touchtest-302e5eb4-872d-4c16-8099-db0a9ef0aa15"/>
tion android:name="android.intent.action.VIEW"/>
tegory android:name="android.intent.category.BROWSABLE"/>
tegory android:name="android.intent.category.DEFAULT"/>
t-filter>

 <service android:enabled="true" android:exported="false" android:name="com.soasta.android.touchtest.TouchTestService"/>
on>
ermission android:name="android.permission.ACCESS_WIFI_STATE"/>
ermission android:name="android.permission.INTERNET"/>
ermission android:name="android.permission.READ_PHONE_STATE"/>
ermission android:name="android.permission.BLUETOOTH"/>
```

- And, finally, a new uses-permission section has been added.

```
          <service android:enabled="true" android:exported="false" androi
/application>
   <uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
   <uses-permission android:name="android.permission.INTERNET"/>
   <uses-permission android:name="android.permission.READ_PHONE_STATE"/>
   <uses-permission android:name="android.permission.BLUETOOTH"/>
ifest>
```

Once you have verified these changes you are ready to deploy the sample app to your Android device or emulator.

Deploy the TouchTestable App

Create the TouchTestable APK—whether using dynamic or static instrumentation techniques—before using adb to install it to the device or simulator using the following step.

1. From the command line, execute the adb command using your own paths:

```
~/android-sdks/platform-tools/adb install -r
~/Shared/Jenkins/Home/jobs/KitchenSinkFunctionalTests/bin/KitchenSink-
debug_TouchTest.apk
```

When all of the conditions and steps above are completed, the APK is pushed onto the Android Device.

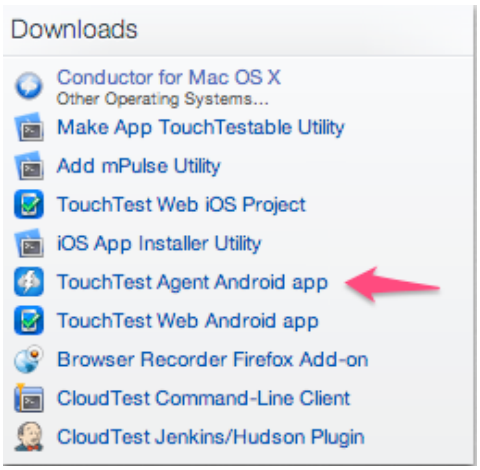
```
873 KB/s (2083295 bytes in 2.330s)

      pkg: /data/local/tmp/KitchenSink-debug_TouchTest.apk

Success
```

Install the TouchTest Agent on a Device

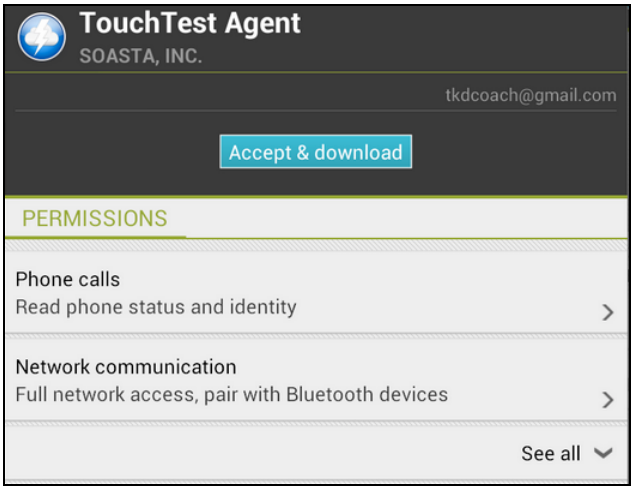
The TouchTest™ Agent is responsible for launching the apps that are being tested on a given device. This Android app runs in Android devices running 2.3.3 or later.

To get started, download the TouchTest Agent onto the mobile device and then perform the one-time registration steps that will enable your device for use with TouchTest.

1. Using your Android device, download the TouchTest Agent from the CloudTest, Welcome page.

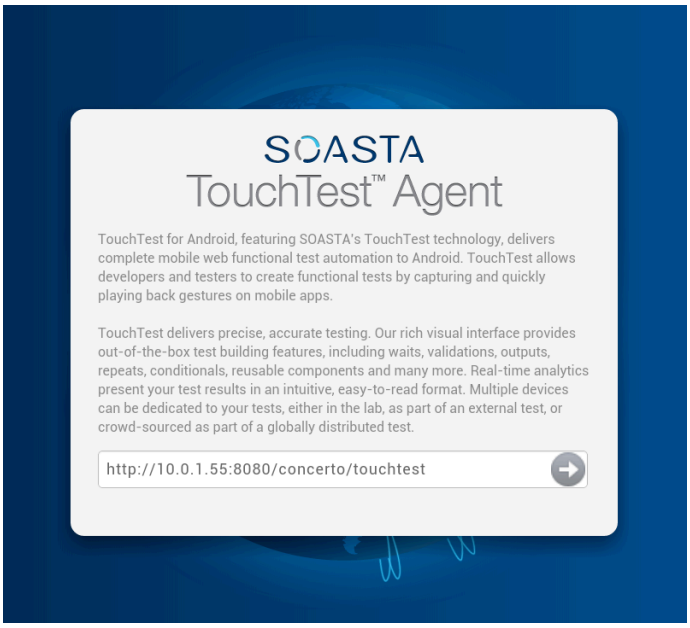


2. Tap Install. When the installer appears, tap Install to proceed. The Play Store installer is shown below.



Note: You can also download the TouchTest Agent on the /touchtest URL of the CloudTest Environment.

3. Once install completes, tap Open to launch the TouchTest Agent app on the Android device. When you do so, the app launches and the TouchTest Agent splash screen displays.



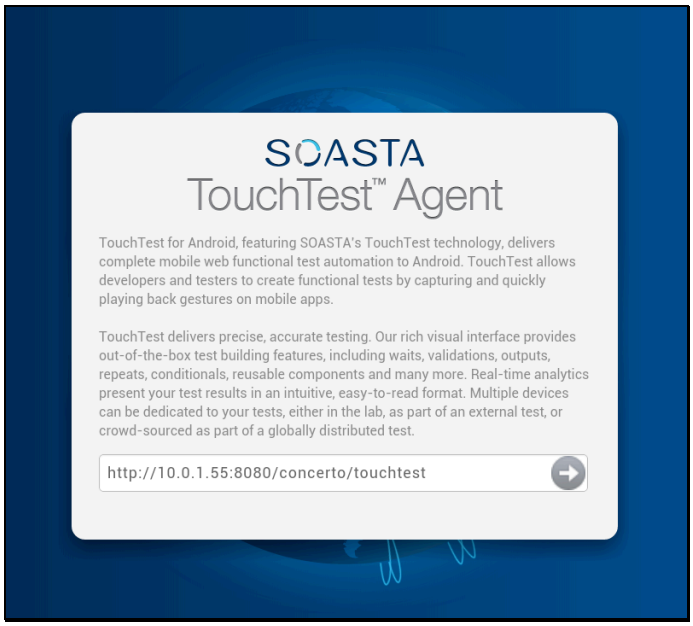
4. Enter the CloudTest URL. For example, http://10.0.1.6/concerto/touchtest or on another server using the same syntax, http://<CloudTest server>/concerto/touchtest.

5. Tap Go. The TouchTest Agent opens the provided URL and a spinner appears as the app auto-registers itself.

6. Once the spinner disappears, enter your CloudTest user name and password and tap Login.

7. When prompted, give the TouchTest Agent a name. For example *Soasta Demo Nexus.* Note that this name will be used throughout the product to refer to this device. Once entered the device name can only be changed by an Administrator.

8. Click Submit for administrator approval. Note that if you're a CloudTest Lite user—you can approve this device yourself using the instructions in the next section.



Once the request for Administrator approval has been made, the TouchTest Agent will continue to poll CloudTest for approval.

Note: It is not necessary to keep the TouchTest Agent running while this approval is pending. The TouchTest Agent will resume polling for its approval once restarted.

If the Mobile Device Administrator approves your device, the Connected page will appear the first time TouchTest is launched on the approved device. On subsequent launches click Login to Connect and Logout to Disconnect.

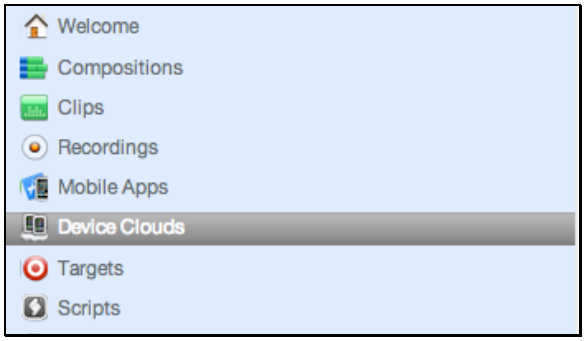
If the Administrator rejects your device, that status will display.

Approving a Mobile Device (Mobile Admin or CloudTest Lite user)

The TouchTest™ Mobile Device Administrator has the responsibility to approve or reject the devices attempting to join testing. Administrators will use the following steps to approve/reject the devices attempting to join.

1. Login as the user with mobile device administrative rights.

2. Click Central > Device Clouds,



When you do so, the Device Clouds list displays those devices in queue by name. Additionally, the model (iOS device type), OS (iOS version), and current status of the TouchTest agents are displayed in the list columns.

Those devices that have the status Pending Approval need administrative attention:

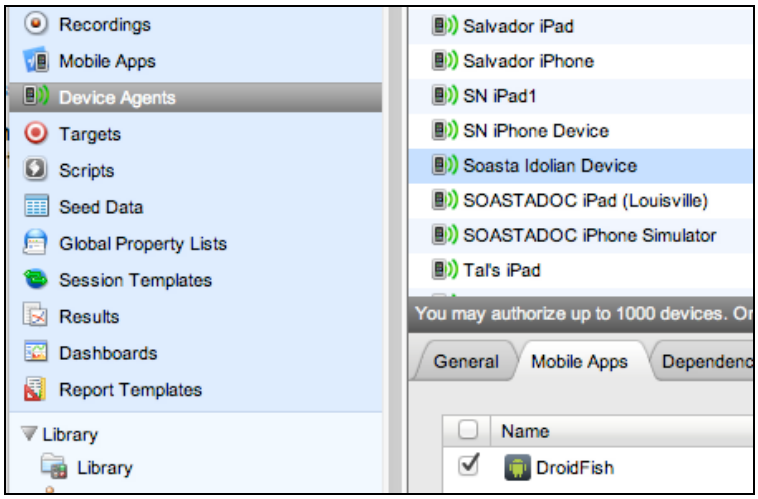
3. Click Approve to complete adding a device and Reject to deny its access.

Note: CloudTest Lite/TouchTest Lite users may approve a single device only and that device cannot be removed. Approval should only be performed on the intended single device.

Associating Mobile Apps with a Device

Once a device is approved, use the following steps to assign one or more mobile apps to that device.

1. In Central > Device Clouds, select the mobile device.



2. In the lower panel, click the Mobile Apps tab. If necessary, use the Maximize button to increase the workspace.

3. Check the Mobile App(s) that you want to authorize this device to access and then click Save on the lower panel toolbar. For example, KitchenSink.

Recording a TouchTest™ Scenario

Once the TouchTest Agent profile is installed and device access is approved you are ready to record and playback your TouchTest.

- Create a new test clip within CloudTest®
- Click Record within the Clip Editor and then choose Mobile App Recording and specify the Device Agent and the mobile app whose actions you want to record.

These and the following additional configuration steps are described in the remainder of this tutorial.

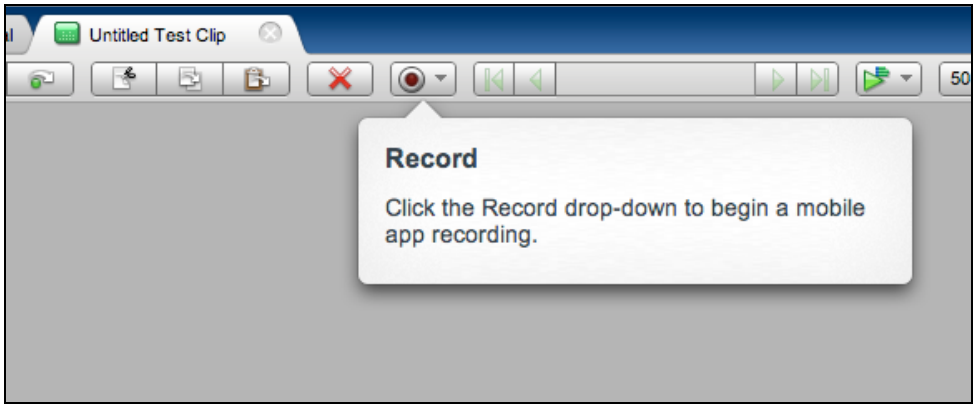
- Composing a TouchTest™ Clip
- Playing back a TouchTest™Composition
- Analyzing TouchTest™ Composition's Results

Create a Simple TouchTest™ Clip using KitchenSink

Create a new clip that will be used to perform mobile app recording and serve as the basis for your first test composition.

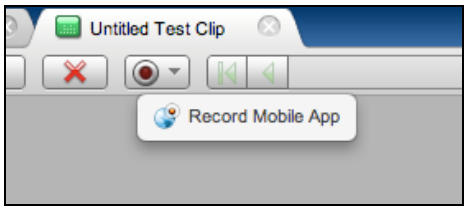
1. Start the TouchTest Agent app on your Android device.

1. Login to CloudTest on your desktop computer and select Central > Clips, and then click New on the Central toolbar.

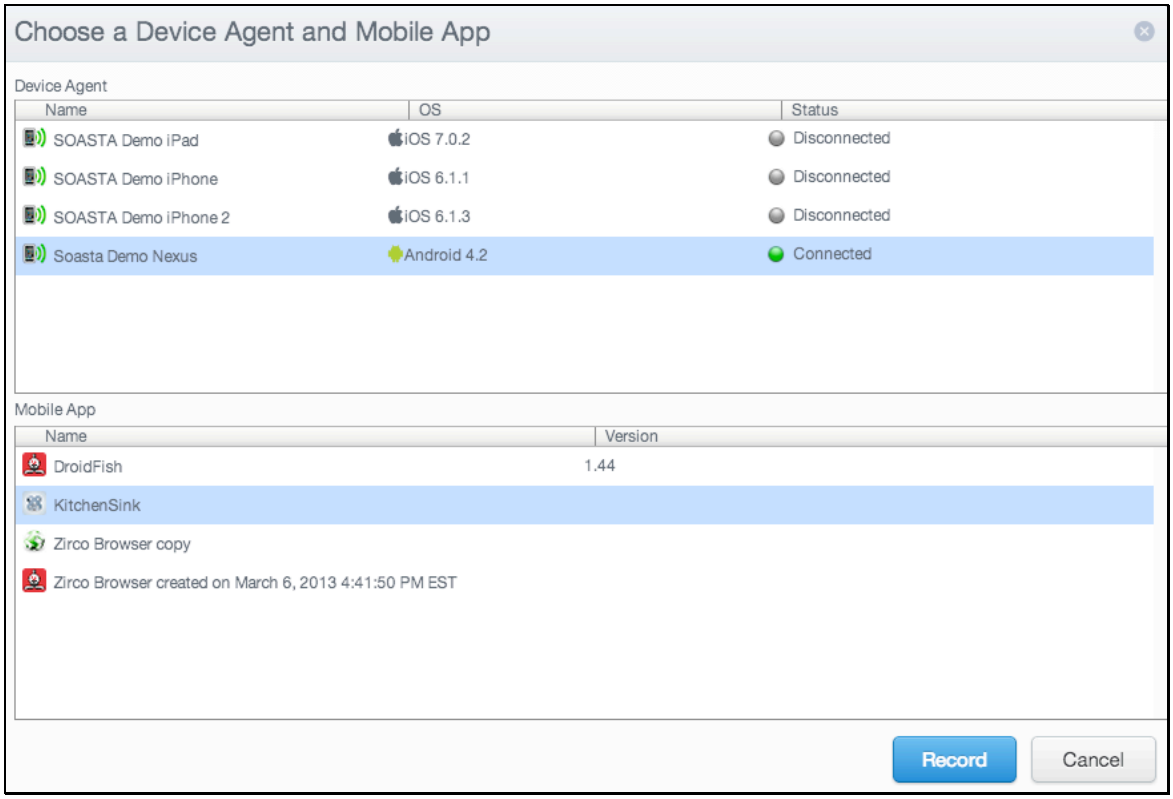


A new Untitled Test Clip opens in a Clip Editor tab. A Record pop-up identifies the Record drop-down.

2. Once ready, click the Record drop-down and then select Record Mobile App.



The Choose a Device Agent and Mobile App wizard appears.



Note: If the Mobile Device Administrator has completed the steps above to associate one or more mobile apps with the device, those will appear in the Mobile App list whenever that device is selected. If you are the Mobile Device Admin you should perform those steps yourself (as noted above in Approving a Mobile Device).

3. Select the TouchTest Agent that you created above and also select the mobile app.

4. Click the Record button in the wizard once your selection is made. TouchTest Agent will launch the selected app on the selected device.

5. The TouchTest Agent app launches on your mobile device.

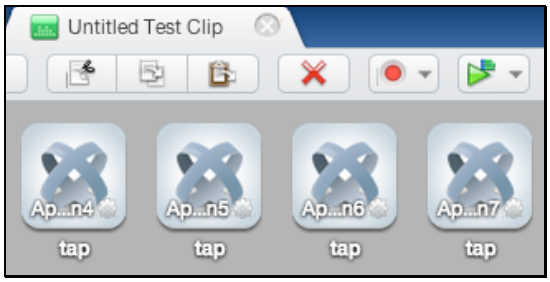
Once successfully logged on, its Status will be *Connected*.

6. The mobile app launches to its initial screen.

7. Perform the planned mobile app user interactions on your mobile device. In the case of our example, record the following in KitchenSink after the Controls view appears:

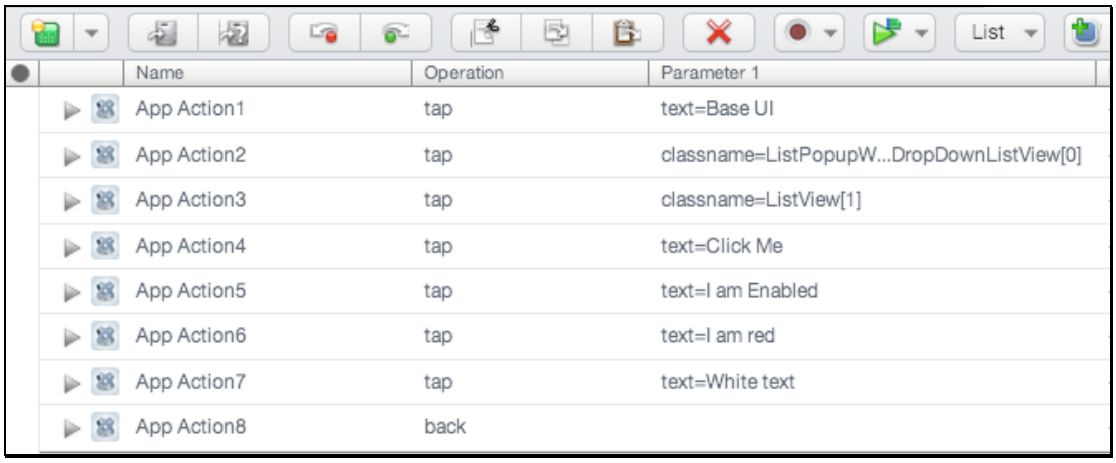
- Tap the BASE UI menu (top left)
- Select Controls from the menu
- Tap Button
- On the Button screen, tap the button that says "Click Me" and then tap it three more times; once each for "I am enabled", "I am red", and finally for "White text."
- When the button says "I am disabled" for the second time (and this time doesn't quickly change its state to "I am enabled"), then tap the Android device's Back icon (bottom left)

For each app action you perform, the Clip Editor adds an app action to the clip. There were eight actions recorded in our clip.



- Click the Record button again to stop the recording (the "red" Record button above is still in Record mode).

- Switch the Clip Editor to List view using the Icon/List button on the Clip Editor toolbar in order to see more inline details about the app actions that were added.



Note: App Action4 through App Action7 above are actions that originated in the button clicks during our recording session.

- Click Save on the Clip Editor toolbar. In our example, we gave the clip the name, *Demo Clip for KitchenSink-Android*.

TIP: It is possible to proceed to create and play a composition at any point after Recording is completed.

Adding an Interval Delay between Each Action (Optional)

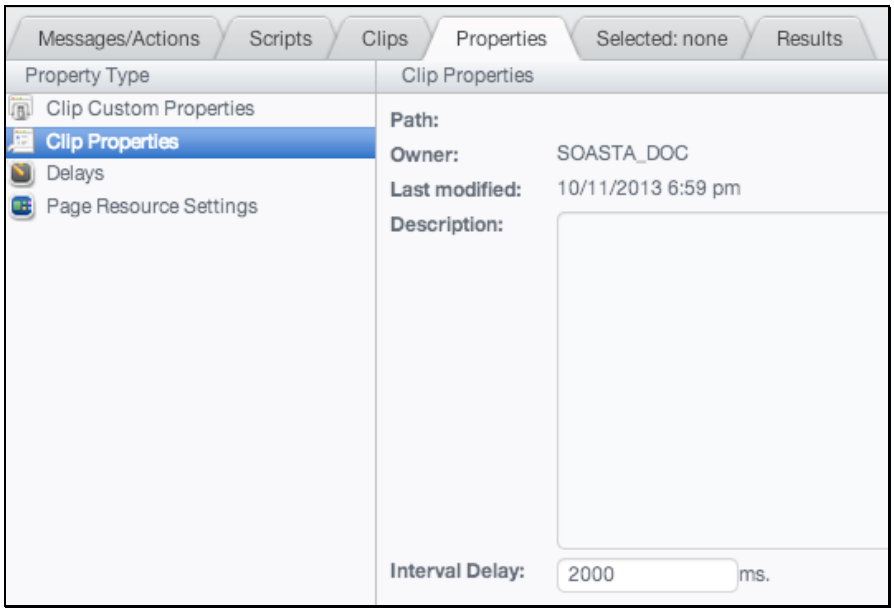
In the following steps, we will add an interval delay to the test clip. This type of delay will stretch out the time between all the recorded app actions.

Imposing delays, either using the Interval Delay setting or by inserting Delay clip elements, can make the test more viewable during test playback (when viewing the test as it plays is most desirable).

1. Click the Properties tab in the minimized sub-panel and then select the Clip tab at the top of the pane (the Clip tab may already be visible if properties are already open from the prior exercise).

2. In the Property Type list, click Clip Properties.

3. In the Clip Properties panel on the right, enter an Interval Delay in the given field. For example, *2000* ms. Entering *2000* adds a two second gap between each app action in the given test clip.



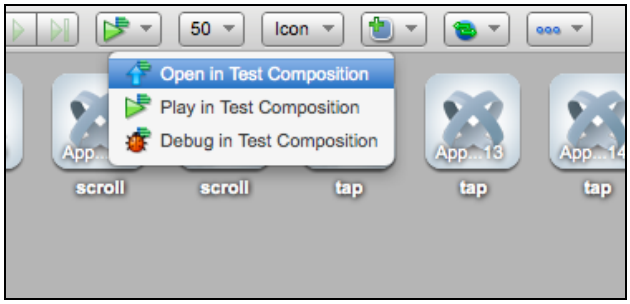
4. Click Save on the Clip Editor toolbar.

TIP: It is possible to proceed to Create a Composition at any point after Recording is completed.

Create a Composition

With your test clip open in the Clip Editor, you are ready to create and play a new test composition using this test clip.

1. To create a new composition from your test clip, click the Open in Test Composition drop-down in the upper-right corner of the Clip Editor toolbar and choose from among:



- **Open in Test Composition**

  Choose Open in Test Composition to add this clip to a new draft composition where additional composition parameters can be set in the Composition Editor, Edit tab before proceeding to play.

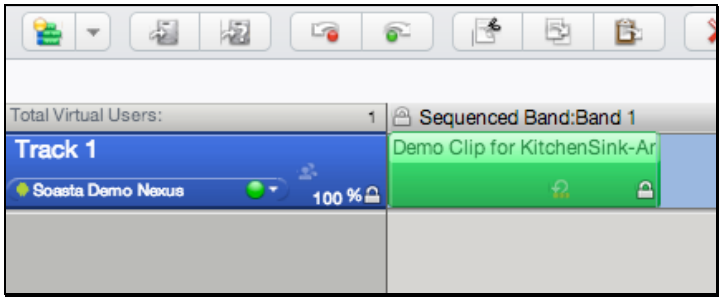
- **Play in Test Composition**

  Choose Play in Test Composition to add this clip to a new draft composition where it will immediately be played in the Composition, Play tab before proceeding to edit parameters or play.

- **Debug in Test Composition**

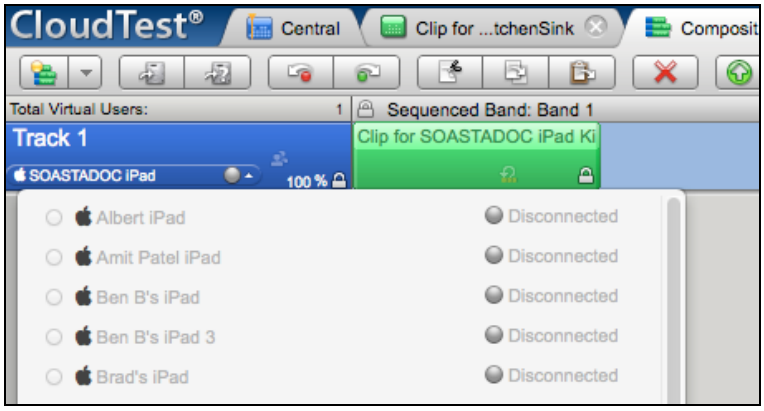
  Choose Debug in Test Composition to add this clip to a new draft composition where it can be debugged in the Composition, Debugging tab before proceeding to edit parameters or play based on debug actions.

When you choose Open in Test Composition, the Composition Editor appears with a draft test composition including the new test clip in Track 1.



When opened from the Clip Editor, the device that was used to record the clip is automatically selected in the track (i.e. SOASTA Demo Nexus is selected in the Track's dropdown location above).

TIP: To playback this test composition on another device, click the drop-down and make another selection. This preference can also be set at the clip level.



2. Click Save on the Composition Editor toolbar and give the test composition a name or accept the default name.

   For example, *Composition for KitchenSink-Android*.

Playing a Composition

Perform these additional steps while the test composition is open in the Composition Editor.

1. Ensure that the TouchTest Agent app's status is “Connected” on the mobile device.

2. In the Composition Editor, click Play to run the test composition. The mobile app actions performed when the clip was created are played back on the device.

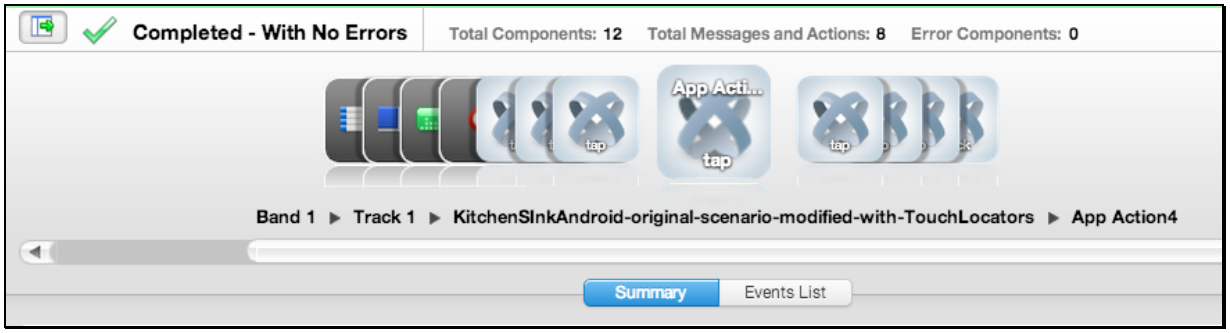
The Composition Editor’s Status Indicator changes to “Playing.” While the test runs, the Composition Editor automatically switches to the Play tab, and by default, and the runtime results are shown on the Result Details dashboard.

Meanwhile, on the device, the mobile app is launched and the actions in the test clip are repeated precisely as they were recorded.

Navigating Result Details

Click to expand the nodes in the Navigation Tree on the left as they appear.

Result Details uses a Cover Flow (top panel to the right) to display the test composition’s stream as it occurs.

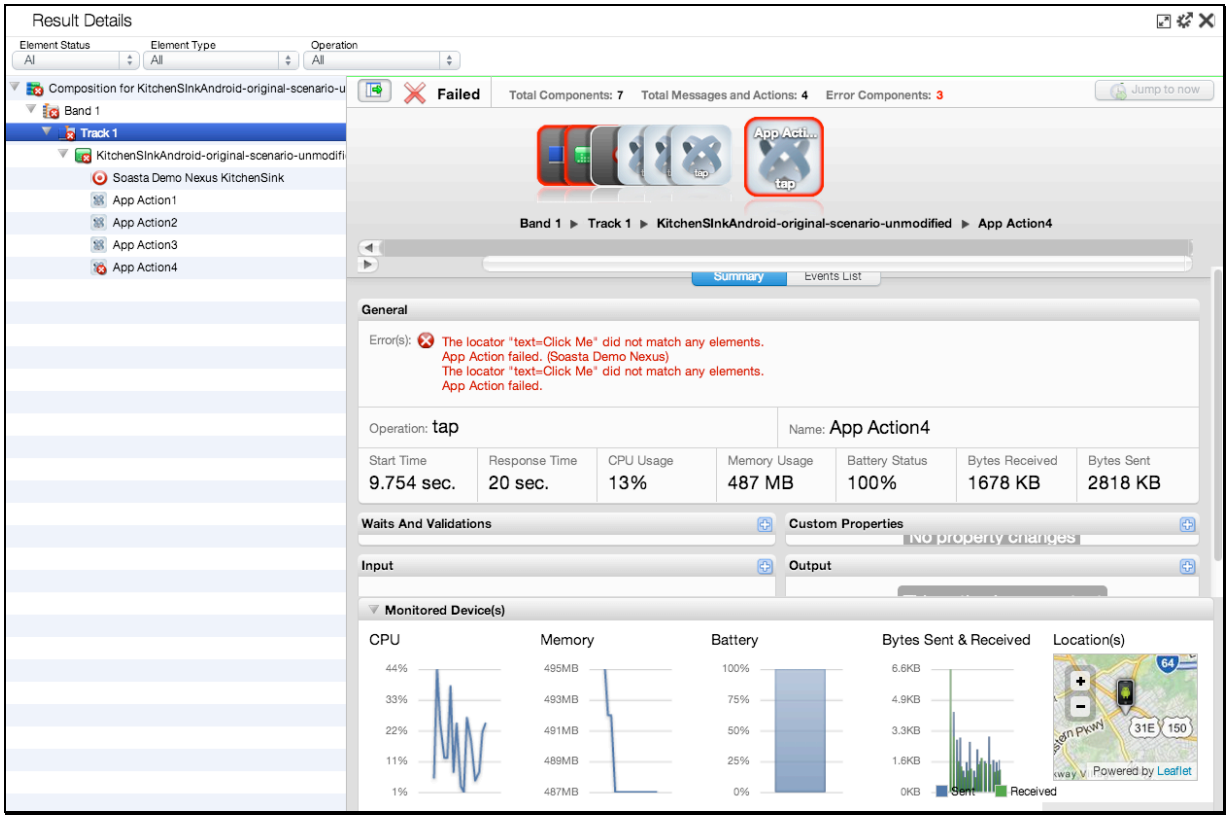


TIP: You'll need to continue following this section to get to the "Completed – With No Errors" status for the draft composition that is shown above.

This stream is also shown in the tree as elements are executed during play. As play continues, the focus is set to the last executed element. The current container is expanded while the prior containers are closed. Clicking an element during play will halt this auto-focus-to-the-last-executed behavior.

- Click any object in the Cover Flow at the top to center it and display its details and play statistics in the panes below. Use the scrollbar to browse the flow. Select any item to show its low-level details.

The Result Details dashboard helps to discover the cause of errors in your test, if any. While play continues the Composition Editor switches to the Play tab, and displays the runtime results in the Result Details Dashboard.



If the composition fails for any reason, as was the case in our initial playback of this test, then the Result Details dashboard clearly indicates that and presents details as to why.

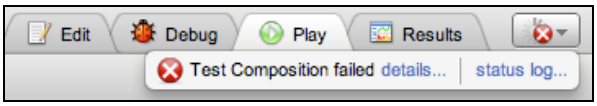
Identifying and Analyzing Common Errors

Despite the successful results above, in some cases your test may not succeed initially. As test advocates, we are often more interested in such failures. The way we approach them is first to identify where they occurred and then to analyze what occurred.

Network or Communication Errors

Initial errors in a simple test like the one above are also often simple network or configuration errors having to do with test staging. They frequently are related the state of the Device Agent (e.g. if the device agent is not connected when you click Play).

For example, if the Device Agent is not connected or is not responding the Composition Editor's Status Indicator will indicate "Test Composition failed" (shown below).



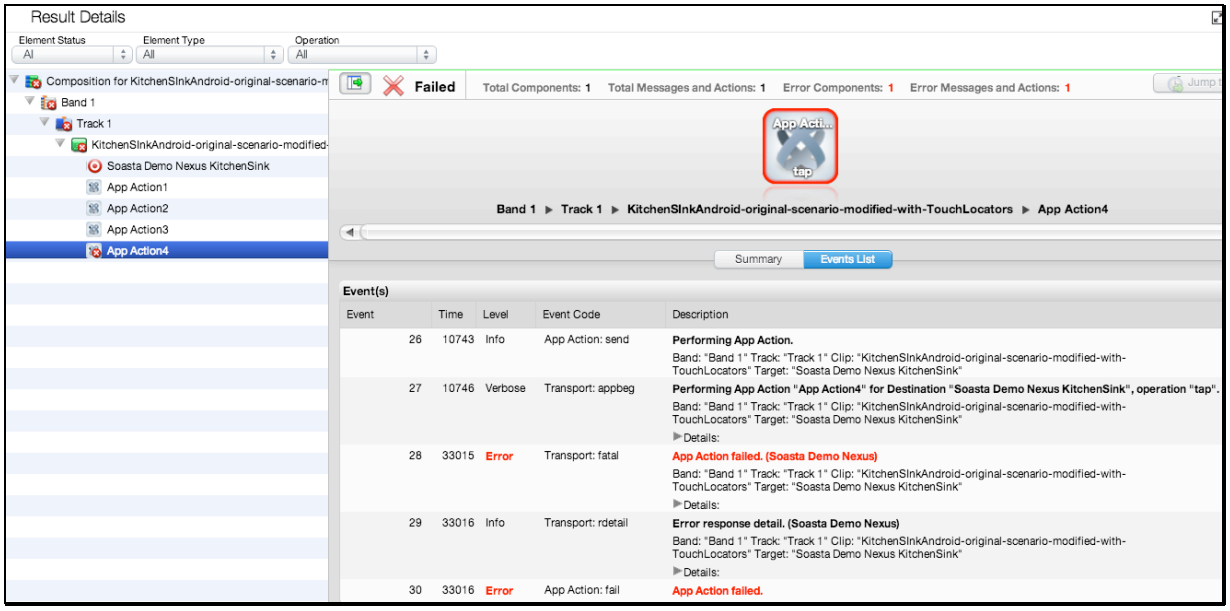
- Click Details to display additional information in a dialog box.

In some cases, the TouchTest Agent may have been started but is no longer responding. In such cases Logout and re-login to the TouchTest Agent.

App Action and Other Errors

CloudTest reports all failures and marks test successful or unsuccessful by the Failure Actions set within it. Failure Actions are set stringently by default to fail the test for any error and show that failure in red.

Result Details clearly indicates the type of test failure that has occurred in a given case. The red "X" in the Navigation Tree easily distinguishes failures on specific app actions you recorded, and whenever the error item is selected in Result Details.

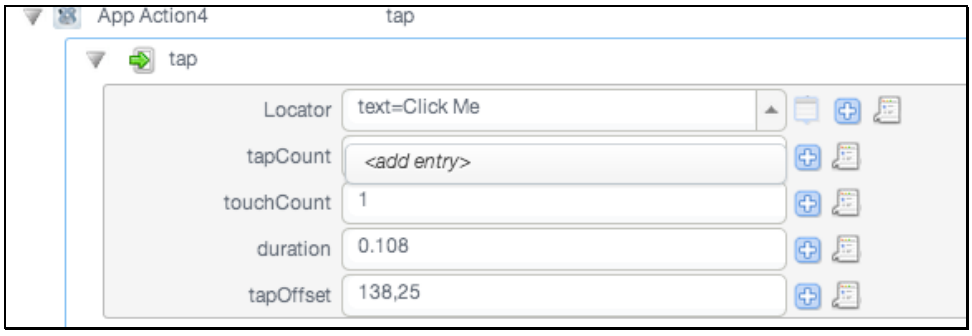


You can get an events view of the selection by clicking the Summary tab (shown above).

The error in App Action4 in our example clip shown above resulted because the locator recorded for *text=Click Me* failed. In such a case, double-click the result item, App Action4, in order to return to that same action in the Clip Editor.

While inspecting the failed action in the Clip Editor, List view, click the Locator drop-down (first icon to the right of the Locator entry field) to see if any additional locators were found. Try the next one in the order listed and replay the test.

In our test clip there were no additional locators captured. As a result, we decided to use the Touch Locator tool to find additional locators.



We used the following steps to quickly repair App Action4 (with TouchTest Agent app on top on the Android device):

1. Leave App Action4 selected and expanded in List view.

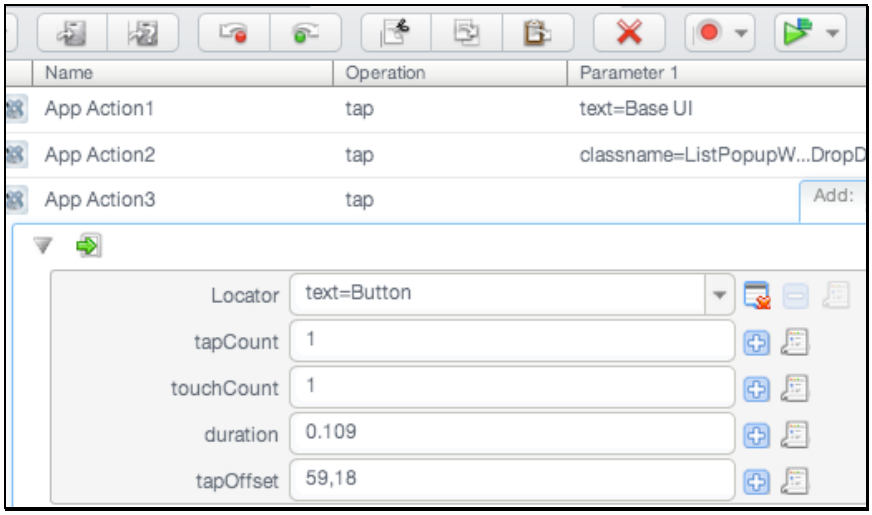
2. Invoke record mode a second time.

3. Navigate to the Controls list as before

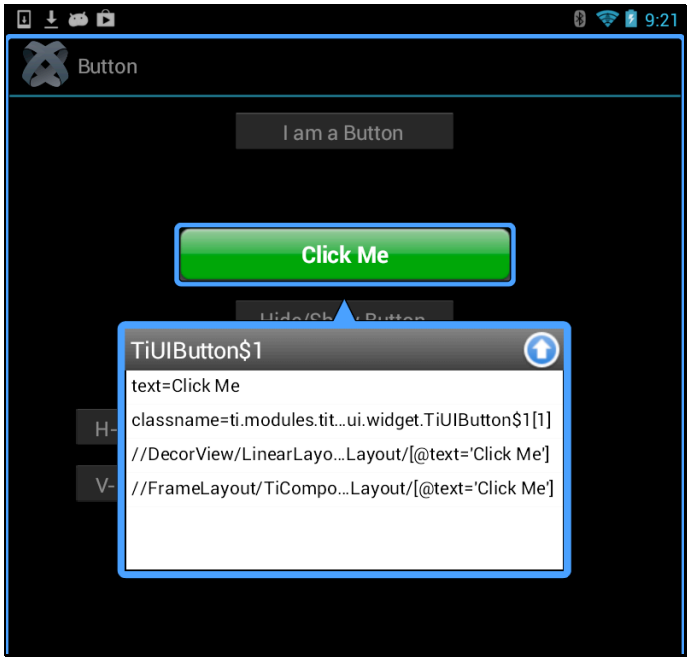
4. Once on the correct view (e.g. with the Click Me button text shown after navigating from Basic UI to Controls and then Buttons), click the Touch Locator icon.

   A blue border surrounds the page on the device and the in the Clip Editor the Touch Locator icon is active.

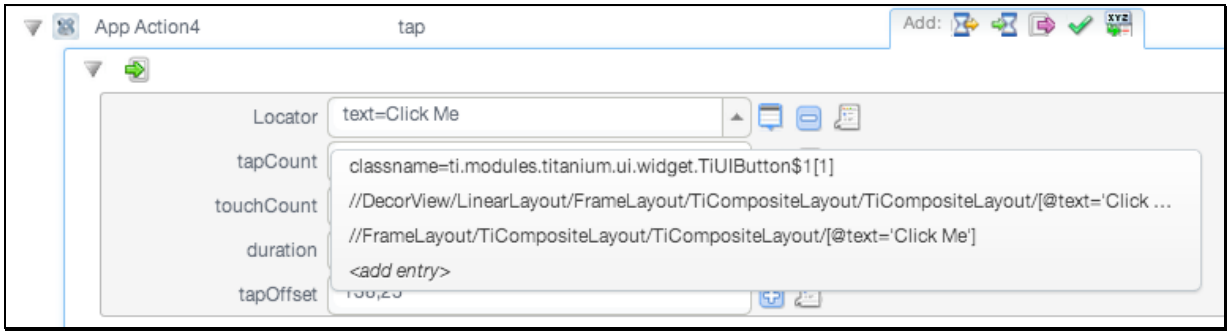
TIP: If you find that the mode is stuck, try toggling the Touch Locator icon for the given action off and then on again. For more about the Touch Locator feature, see Touch Locator for Mobile Apps.



5. Long press the "Click Me" button until the blue border collapses around it and the list of locators shown below appears.



6. Click the Up Arrow icon (in the Locators box above) to include this list of locators into App Action4.



Select the first locator (e.g. other than *text=Click me*), save the clip, and then play the Draft Composition (in the Composition Editor tab) again.

7. Repeat the Touch Locator procedure for any additional failures.

   —Or—

   In this case, since we know that a number of taps are on that same button in different states, we can save some time and copy the Locator text from App Action4 into all of the remaining App Action locator fields that use that button (e.g. App Action 4 through App Action7 all were on this button):

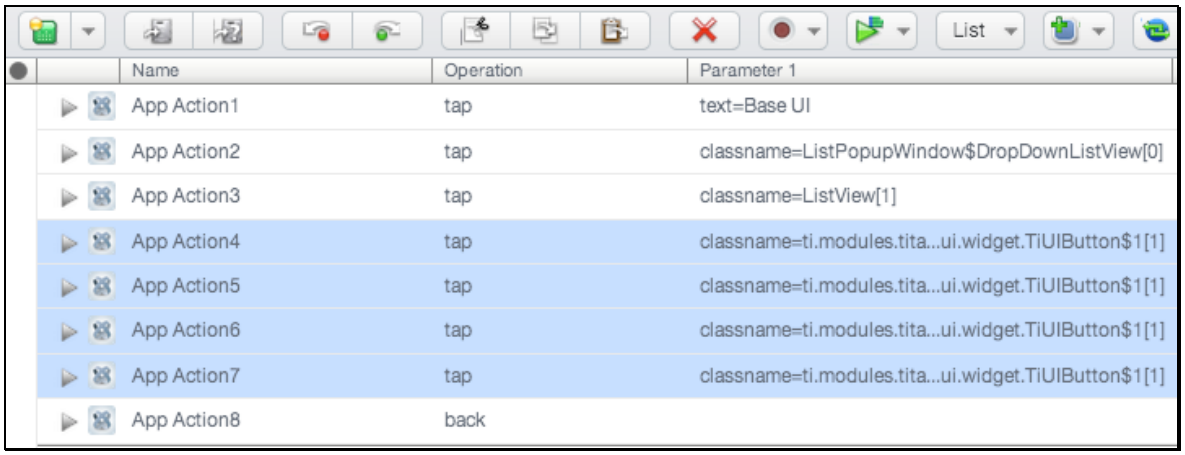
   ```
   classname=ti.modules.titanium.ui.widget.TiUIButton$1[1]
   ```

If you do opt for the copy and paste method, click the drop-down and use "*<add entry>*" so that the locator is added to the list of locators. Otherwise *text=Click Me* and the other original locators will be lost.

8. In either case, exit Recording mode in the Clip Editor when done.

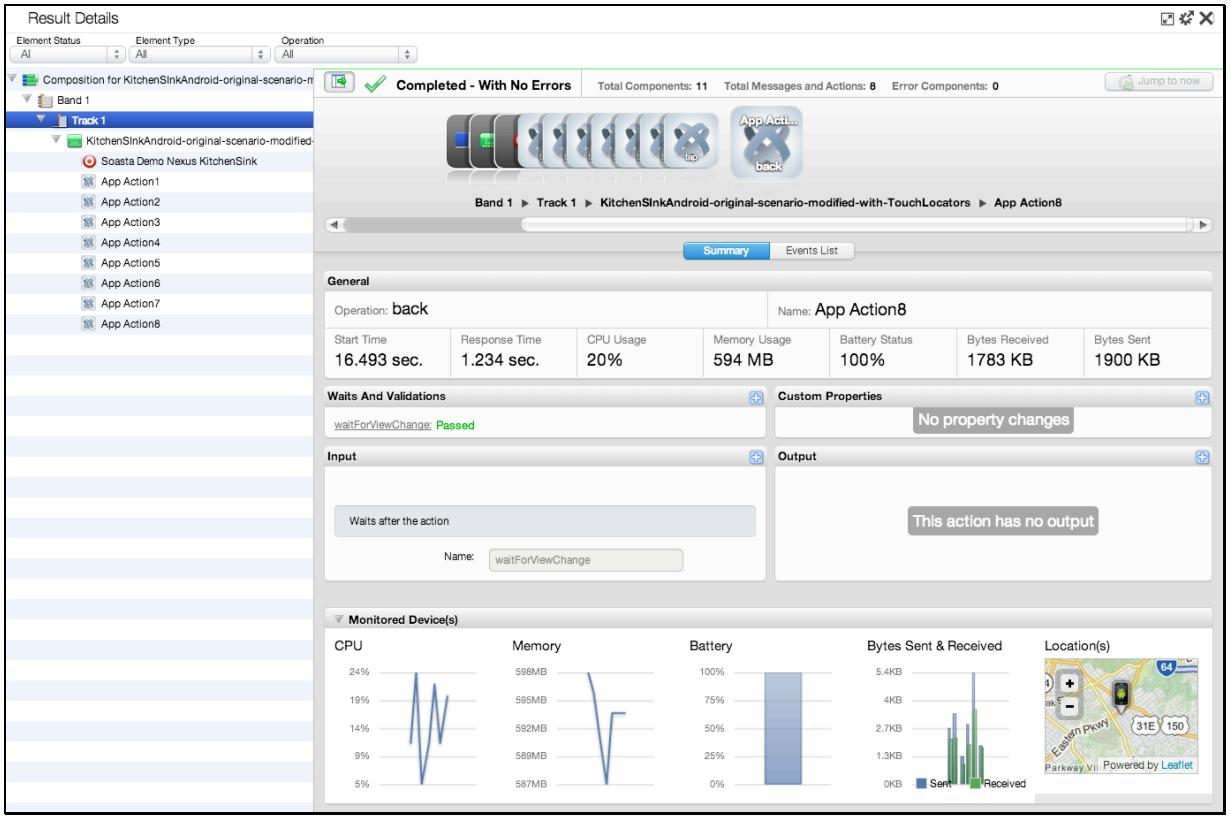
TIP: While working in Touch Locator mode, **delete any new app actions** that are inadvertently recorded. You can recognize them by their out of order action numbers (e.g. their names are not sequential to the action you selected before entering Touch Locator mode).

Our edits to the original KitchenSink clip looked like this:



9. Save the test clip before playing the composition again.

In our example clip, the modifications made by using the Touch Locator procedure were successful.



Despite the fact that this strategy proved successful in fixing up our initial clip, the locator that we used isn't exactly intuitive, and in situations where the value of a button, or some other control, is *variable* this strategy (e.g. `*.TiUIButton$1[1]` isn't always "`[1]`" then this technique may still prove unsuccessful in some mobile apps.

In the subsequent sections, we will add accessors such as outputs and validations. After we do that, we will finalize our clip by going back to the Titanium project and modify some code to add TouchTestIDs that will be more user-friendly, as well as more accurate for many types of elements.

Advanced Clip Editing

Now that we've played this simple test composition successfully, and discussed how errors are presented when they occur, let's return to the test clip to inspect the clip elements and do some additional parameterization.

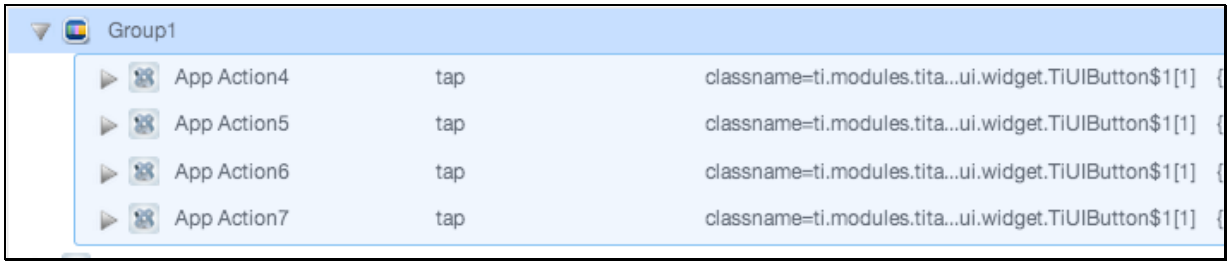
1. Click the Clip Editor tab if it's still open, or right-click the test clip in the Composition Editor and choose Open in New Tab.

2. To add groups or other containers, first identify and select the relevant actions, then right-click and choose the relevant command (i.e., Add to Group). In addition making a test more readable, items in a group can share properties such as repeats, as well as appear in container analytics.

   In the example below, we identified and selected all the app actions created via the steps we performed in KitchenSink's Buttons section, then using the right-click Action menu we chose the Create a Group command.

TIP: In the above shot, note that the locators TouchTest assigned for App Action4 through 7 use the Touch Locator value we added to the clip in the steps above.

Afte this command is applied, the selected actions are placed into a new group, Group 1.



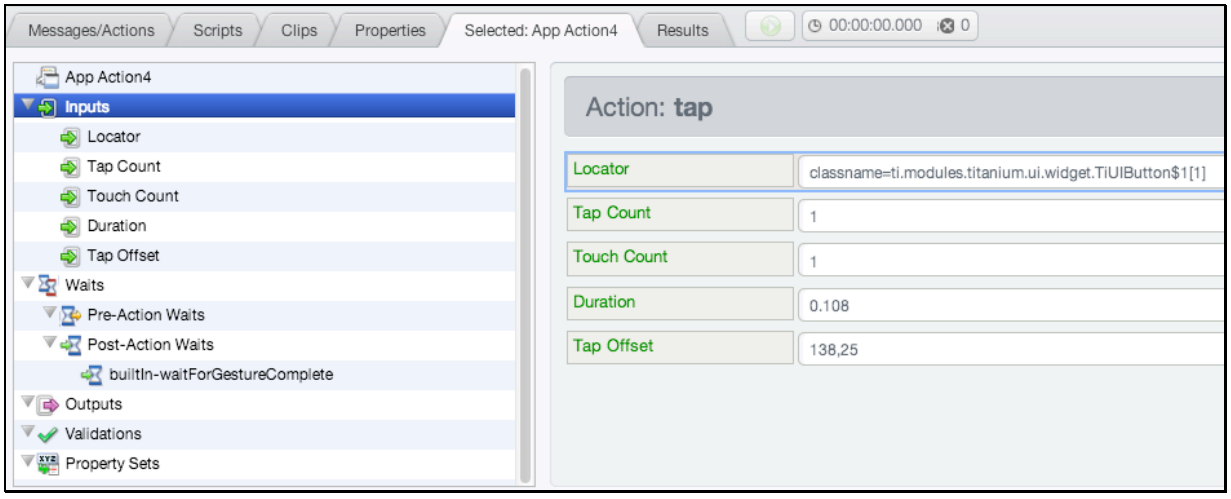
TIP: You can repeat this command for each selection. It is also possible to record each section, halt recording, organize those app actions into a group, and then restart recording from that point.

Inspecting App Action Details

Examine elements and properties for any App Action by selecting it in the workspace above and then double-clicking it. When you do so, the <*Selected: Action Name*> tab displays the contents of the selection in its own pane.

In the test clip below, AppAction4 is open in the lower panel.

4. Locate and double-click the first app action in the Group1. In our sample clip, this is *Selected: AppAction4*. This app action has five Inputs: Locator, Tap Count, Touch Count, Duration, and Tap Offset.

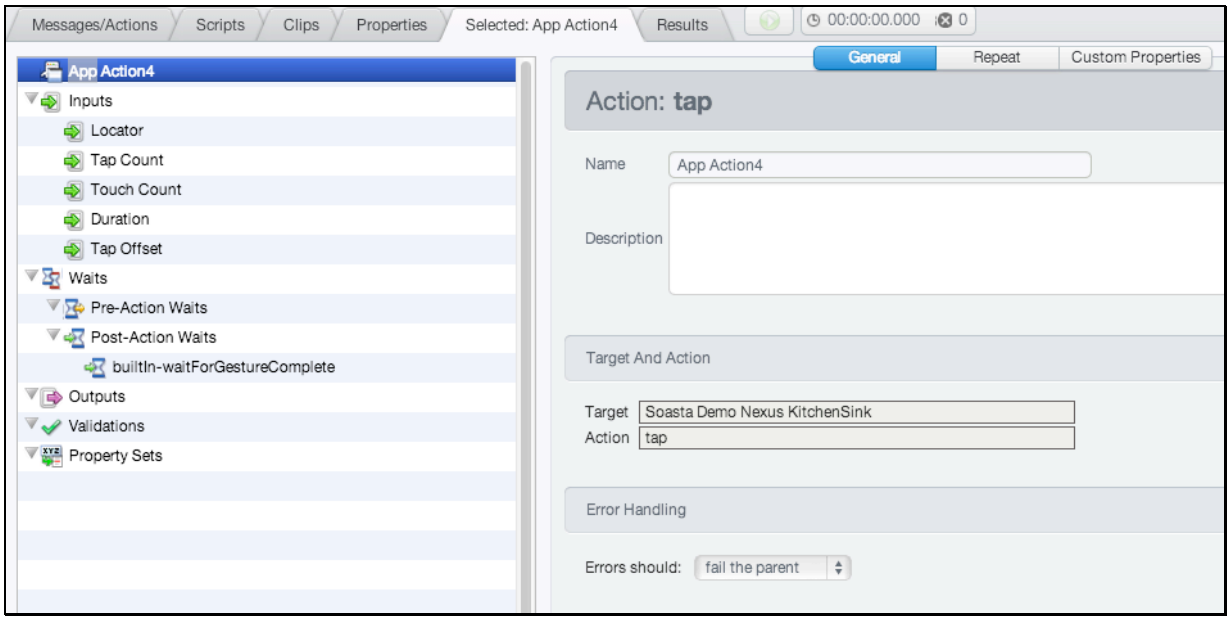


App Action Elements and Properties

While the *Selected: Action* tab active, the Action level properties are shown in the tree on the left.

1. Select the top-level node in the tree (as shown below)

- General, Repeat, and Custom Properties (for the action only; not for the entire clip) tabs appear on the right. Note that Error Handling here is set to *Errors should fail the parent* by default.



- Other settings, including Waits, Inputs, Outputs, Validations, and Property Sets can be set by clicking that node in the tree and then performing the desired action on the right.

1. In the *Selected: AppAction2* tab (or for any selected app action), familiarize yourself with the available elements and properties.

    - **Inputs** (Locator, Scale, Precision, Content Offset)

      Locators are unique characteristics that identify a specific element or object on a mobile device. Locators come in many forms, including links, IDs such as those defined within CSS, and XPath expressions.

    - **Waits** (Pre-Action Waits , Post-Action Waits)

Waits are commands that tell CloudTest not to execute an Action until a condition is met (pre-action waits), or to not continue processing the outputs, validations and property sets of the Action until a condition is met (post-action waits).

-  **Outputs**
  Outputs specify what is to be shown in the Result Viewer for a given Action. Typical outputs include “captureScreenshot”, “outputElementText”, and “outputInnerHTML”. A single Action can have an unlimited number of outputs.

-  **Validations**

  Validations verify some content or event occurred as expected and have a corresponding Failure Action. App Action validations can range from simple true/false conditions to more complex conditions. A single App Action can have an unlimited number of validations. Any validation failures will be exposed in the Results Dashboard.

-  **Property Sets**

  Property Sets give you the ability to take text or data from the app you are testing and store it in a custom property for use in a subsequent action or message.

  SOASTA CloudTest includes three property sets, all of which have relevance for refining and editing a selected App Action.

  - *Custom Properties*

    Custom Properties are user-defined properties that are available to all clip elements, including Actions. Custom properties can be thought of as backdoors that allow access to portions of the object model more easily.

  - *System Properties*

    System Properties are available to all clip elements, including Actions. SOASTA CloudTest defines system properties. For example, a test clip has system properties such as name, repeat timing, label, and more.

  - *Global Properties*

Global properties are defined within the Central > Global Properties List and are “global” within the entire SOASTA CloudTest environment—and can be used across compositions.

Adding a Text Validation

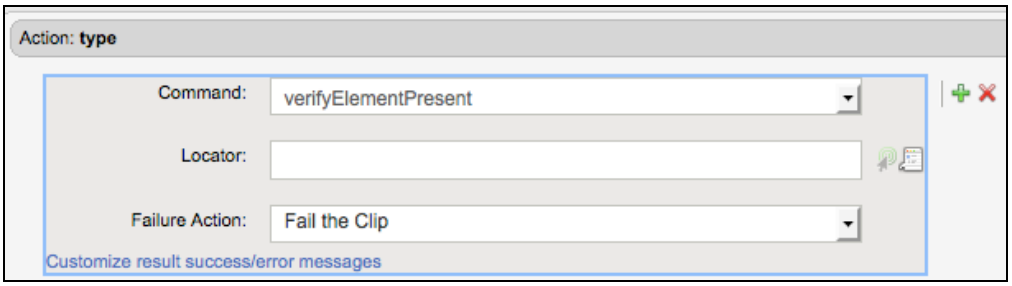
Next, we will add a validation on AppAction2 (or the app action whose operation is *type* and whose locator is *placeholder=Username* in the clip you recorded from the SOASTA Demo app). The response to this and other actions will contain information worth validating in many cases. The remaining steps demonstrate how to do simple validation in CloudTest.

1. Note the content of the given app action. For example in AppAction2 of our sample clip, Locator had the value *text=Slider[1].*

2. If it’s not already open in the lower panel, open it now by double-clicking.

3. Click Validations in the list and then click the green Plus sign  in the Validations panel on the right.

An Action: type form is added to the right panel.



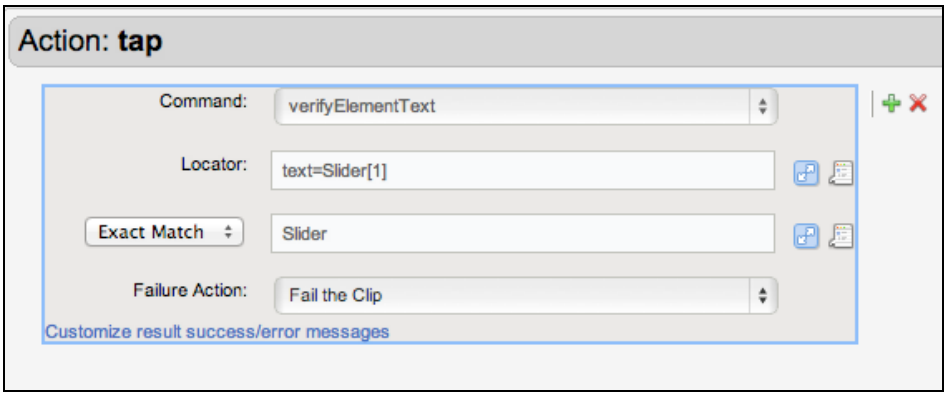
4. In the Command list, a list of commands is presented.

   Click the drop-down and select *verifyElementText*. This Command verifies that the specified text is in the rendered field.

5. In the Locator field, enter the field name shown above. For example, *text=Slider[1]*.

6. In the Match field, accept *Exact Match* and enter the username. For example, *soastadoc*.

7. Leave the default *Fail the Clip* set in the Failure action field.

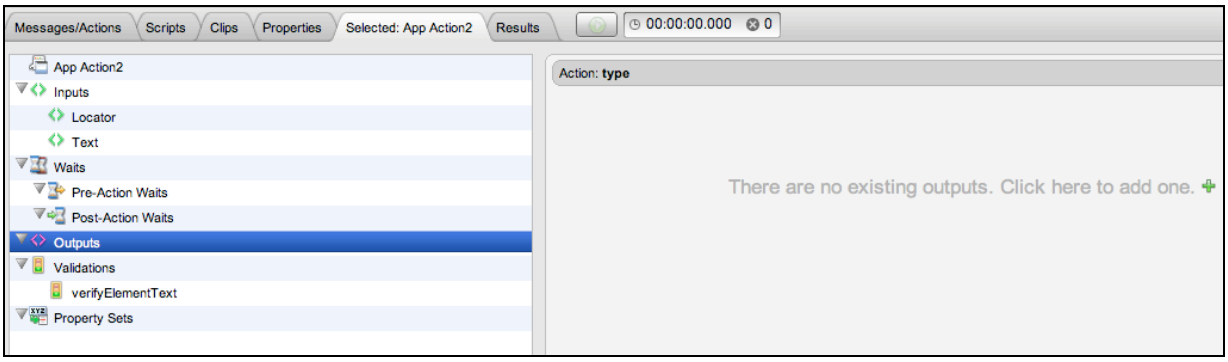


8. Click Save on the Clip Editor toolbar.

Adding an Output

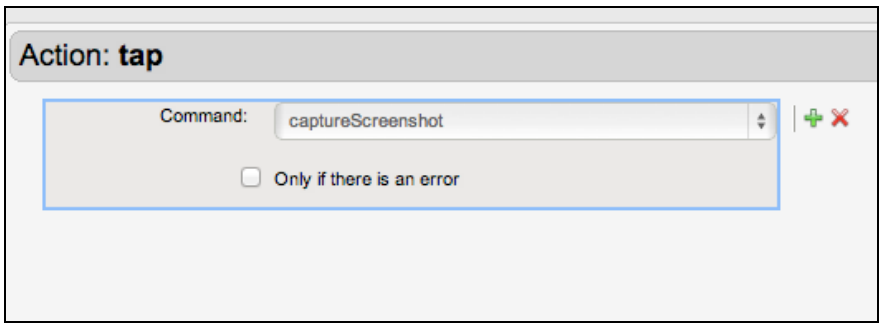
In the following steps, we will add an output to an App Action in the test clip we created above. This output will capture a screenshot of the test clip element as it is executed during runtime and this screenshot will be integrated into the test results.

1. Once again, select the Action to display its properties in the sub-pane below.



1. Click Outputs in the list on the left, and then in the *Action: type* field just to the right, click the green Plus sign again. A new output is added to the Outputs list and the details are shown in the Element Info field.

2. Click the Command list drop-down, and then scroll to the top of the list and select *captureScreenshot*.



3. Check *Only if there is an error* if the output is desired only in the event this AppAction produces an error. Otherwise, leave it unchecked. For our example, we will leave it unchecked to ensure we get an output in every eventuality.

4. Click Save on the Clip Editor toolbar.

5. Return to the Composition Editor tab once again and click Play a second time.

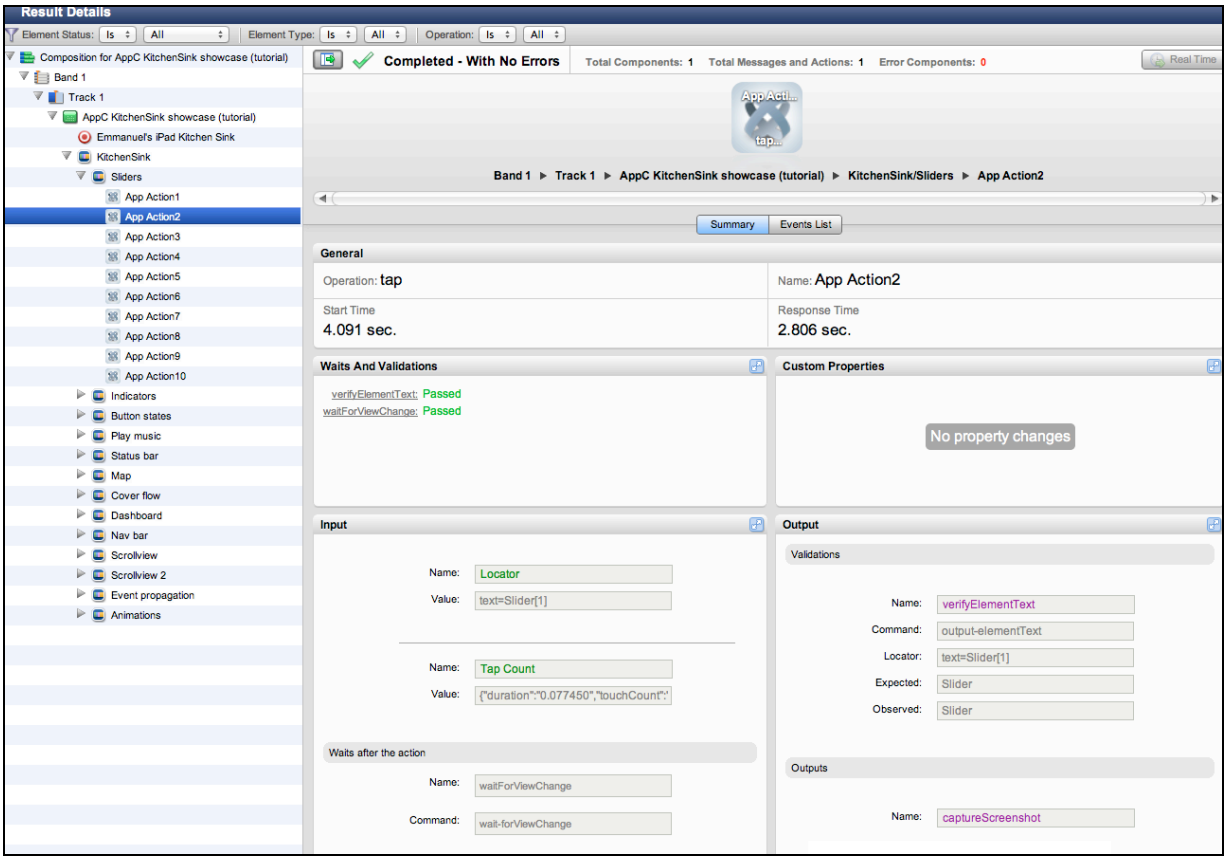
   In the following section, we'll inspect results for the validations and output that were set on AppAction2 above.

Analyzing Results

Result Details has several methods for navigating through the test results. Click the checkboxes in the cover flow to quickly display messages (or Actions) only (within a single band, track, test clip, or chain.

In the best-case scenario, the parameters added in Advanced Clip Editing work without a hitch. We can easily verify the status of our parameters in Result Details.

- Expand the Navigation Tree until AppAction2 is in display and then select it as shown below.

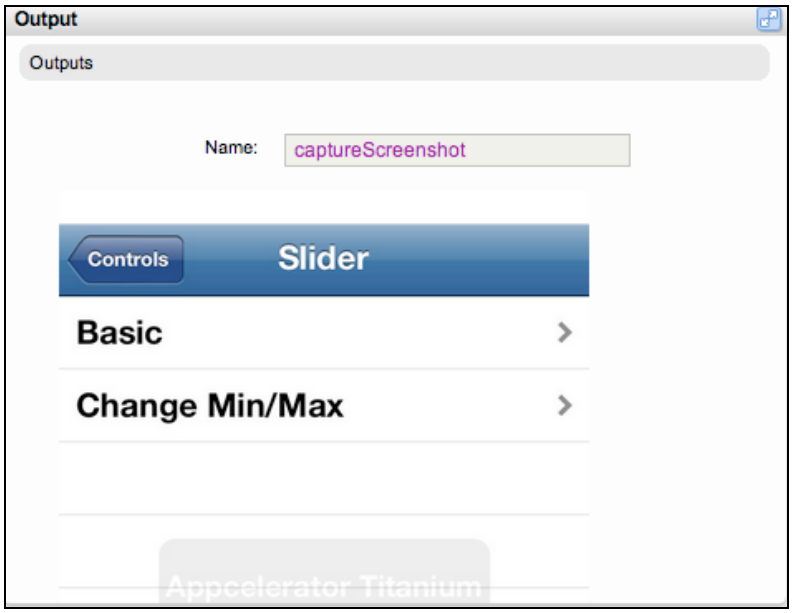


1. To inspect the latest result for the test, open the result in the Composition Editor > Results tab (if it is not already open.

2. Select the clip element that had the validation. For example, AppAction2 (as shown above).

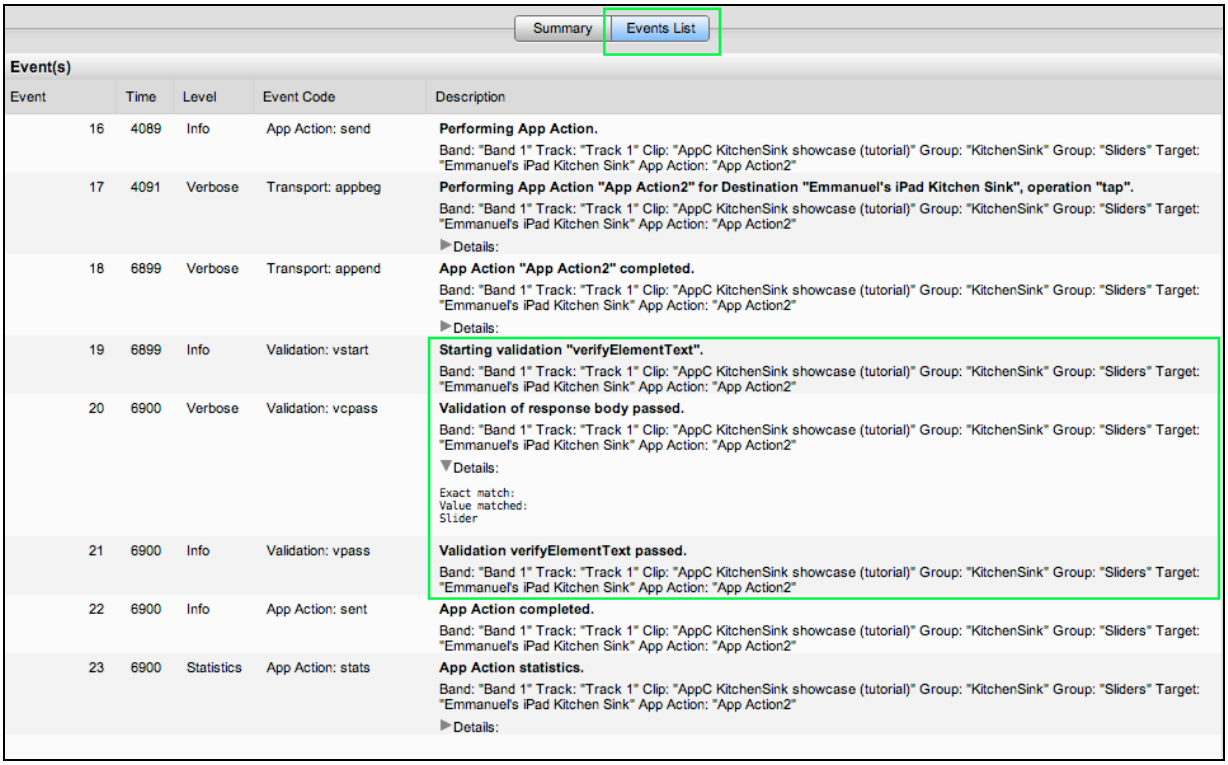
3. Inspect the information on the Summary tab for the selection. In the result above, both the validations on AppAction2 passed in the result shown.

4. Scroll down, if necessary, to view the *captureScreenshot* output defined for AppAction2 in our example.



Note: Since we didn't check *Only if there is an error* in the Output form so a shot of the success is included in this result for the given app action.

5. Click the Events list tab for the given selection to view action-related events, including validations. Click the Details arrow to inspect any event's details.


Summary
Events List
Event(s)
Event
Time
Level
Event Code
Description
16 4089 Info App Action: send
Performing App Action.
Band: "Band 1" Track: "Track 1" Clip: "AppC KitchenSink showcase (tutorial)" Group: "KitchenSink" Group: "Sliders" Target: "Emmanuel's iPad Kitchen Sink" App Action: "App Action2"
17 4091 Verbose Transport: appbeg
Performing App Action "App Action2" for Destination "Emmanuel's iPad Kitchen Sink", operation "tap".
Band: "Band 1" Track: "Track 1" Clip: "AppC KitchenSink showcase (tutorial)" Group: "KitchenSink" Group: "Sliders" Target: "Emmanuel's iPad Kitchen Sink" App Action: "App Action2"
Details:
18 6899 Verbose Transport: append
App Action "App Action2" completed.
Band: "Band 1" Track: "Track 1" Clip: "AppC KitchenSink showcase (tutorial)" Group: "KitchenSink" Group: "Sliders" Target: "Emmanuel's iPad Kitchen Sink" App Action: "App Action2"
Details:
19 6899 Info Validation: vstart
Starting validation "verifyElementText".
Band: "Band 1" Track: "Track 1" Clip: "AppC KitchenSink showcase (tutorial)" Group: "KitchenSink" Group: "Sliders" Target: "Emmanuel's iPad Kitchen Sink" App Action: "App Action2"
20 6900 Verbose Validation: vcpass
Validation of response body passed.
Band: "Band 1" Track: "Track 1" Clip: "AppC KitchenSink showcase (tutorial)" Group: "KitchenSink" Group: "Sliders" Target: "Emmanuel's iPad Kitchen Sink" App Action: "App Action2"
Details:
Exact match:
Value matched:
Slider
21 6900 Info Validation: vpass
Validation verifyElementText passed.
Band: "Band 1" Track: "Track 1" Clip: "AppC KitchenSink showcase (tutorial)" Group: "KitchenSink" Group: "Sliders" Target: "Emmanuel's iPad Kitchen Sink" App Action: "App Action2"
22 6900 Info App Action: sent
App Action completed.
Band: "Band 1" Track: "Track 1" Clip: "AppC KitchenSink showcase (tutorial)" Group: "KitchenSink" Group: "Sliders" Target: "Emmanuel's iPad Kitchen Sink" App Action: "App Action2"
23 6900 Statistics App Action: stats
App Action statistics.
Band: "Band 1" Track: "Track 1" Clip: "AppC KitchenSink showcase (tutorial)" Group: "KitchenSink" Group: "Sliders" Target: "Emmanuel's iPad Kitchen Sink" App Action: "App Action2"
Details:

SOASTA, Inc.
444 Castro St.
Mountain View, CA 94041
866.344.8766
http://www.soasta.com