



TouchTest™ Appcelerator iOS Tutorial

SOASTA TouchTest™ Appcelerator iOS Tutorial

©2015, SOASTA, Inc. All rights reserved.

The names of actual companies and products mentioned herein may be the trademarks of their respective companies.

This document is for informational purposes only. SOASTA makes no warranties, express or implied, as to the information contained within this document.

Table of Contents

Why Mobile App Testing?	1
CloudTest® Basics	1
What Does Touch Test Record?	2
Adding TouchTest™ to an Appcelerator App	4
Appcelerator and iOS Developer Prerequisites	4
TouchTest Prerequisites	4
Importing the Sample Project	6
Using the MakeAppTouchable Utility	9
Static vs. Dynamic Instrumentation	9
Applying MATT to an IPA or APP (Dynamic Instrumentation).....	9
Making an APP file TouchTestable	10
Making an IPA file TouchTestable	13
Making a Titanium Project (Static Instrumentation)	14
Inspecting Changes to the Titanium Project (Static)	16
Deploy the TouchTestable App	18
Install from Titanium Studio (Static Instrumentation).....	18
Install from the Command Line (Dynamic Instrumentation)	20
Registering Your Device to Use TouchTest™	22
Approving a Mobile Device (Administrator Only).....	27
Associating Mobile Apps with a Device	28

Recording a TouchTest™ Scenario.....	30
Create a Simple TouchTest™ Clip using KitchenSink	31
Adding an Interval Delay between Each Action.....	35
Create a Composition	37
Playing a Composition	39
Result Details	40
Navigating Result Details.....	42
Identifying and Analyzing Common Errors	44
Network or Communication Errors	44
App Action and Other Errors	45
Advanced Clip Editing	46
Planning a Complex Test in KitchenSink	46
Increasing Test Clip Complexity	47
Inspecting App Action Details	49
App Action Elements and Properties.....	50
Adding a Text Validation.....	52
Adding an Output	53
Adding a Label Validation	55
Analyzing Results.....	57
Adding TouchTest™ IDs to an iOS App in Titanium.....	60
Supported Titanium Components.....	61
Identifying the Component to Modify	62

Re-Record the KitchenSink Clip with TouchTestIDs 63



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.

There is no need to jailbreak the iOS device and the device can be untethered.

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 the 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 touch gestures including pan, pinch, zoom and scroll.

Each gesture you perform on a TouchTest-enabled device is precisely, and automatically, 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 containers (e.g. transactions, groups, etc.).

TouchTest™ clips are recorded directly from the mobile app and added to the Clip Editor 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 TouchTestable. This tutorial uses the Appcelerator sample app, KitchenSink. However, you can substitute any Appcelerator app and easily follow along.

Appcelerator and iOS Developer Prerequisites

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

- Titanium Studio is installed with Titanium SDK version 2.1.3 or above.

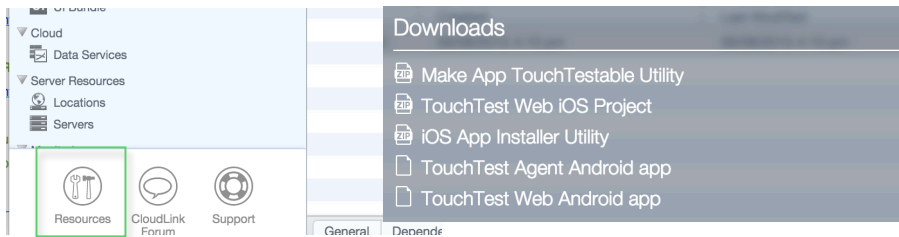
The remaining prerequisites are common to any Appcelerator project that utilizes the iOS Developer Program to install an iOS app on a device:

- The developer is enrolled in the iOS Developer Program (so that the Appcelerator app can be pushed to the iOS Device via iTunes).
- The Unique Device Identifier (UDID) of the iOS Device that will be used to test must be registered at the Apple Provisioning Portal.
- iTunes 10.x or greater must be installed on the desktop client that is running Titanium Studio.

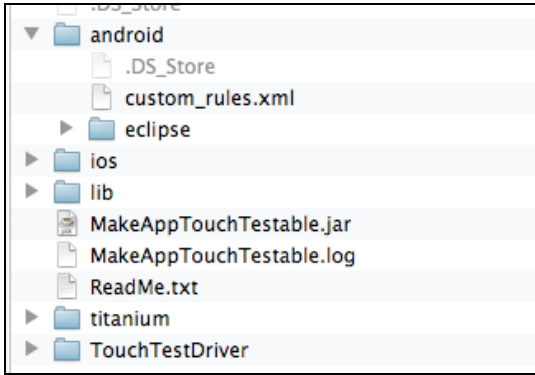
TouchTest Prerequisites

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

- Download the MakeAppTouchTestable utility from Resources > Downloads.



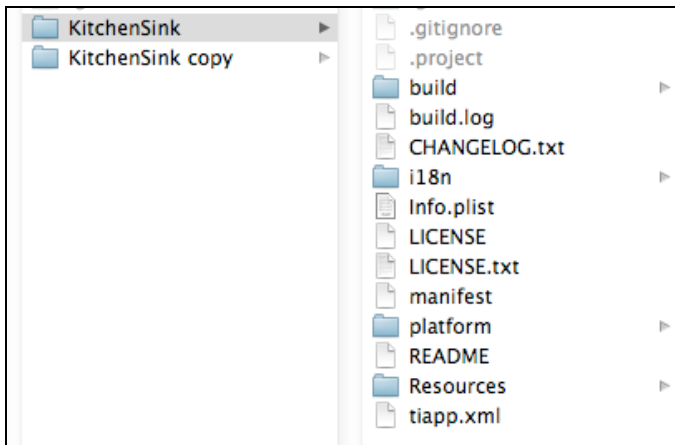
- Unarchive the ZIP file into your working directory. It is not necessary to open any of the contents shown below.



This archive contains the following:

- MakeAppTouchable.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.
- The TouchTestDriver folder. The contents of this folder will be automatically copied to your project.
- The titanium folder contains files expressly used by Titanium Studio.
- You will need a mobile app to follow the tutorial, such as the example [KitchenSink](#) project.

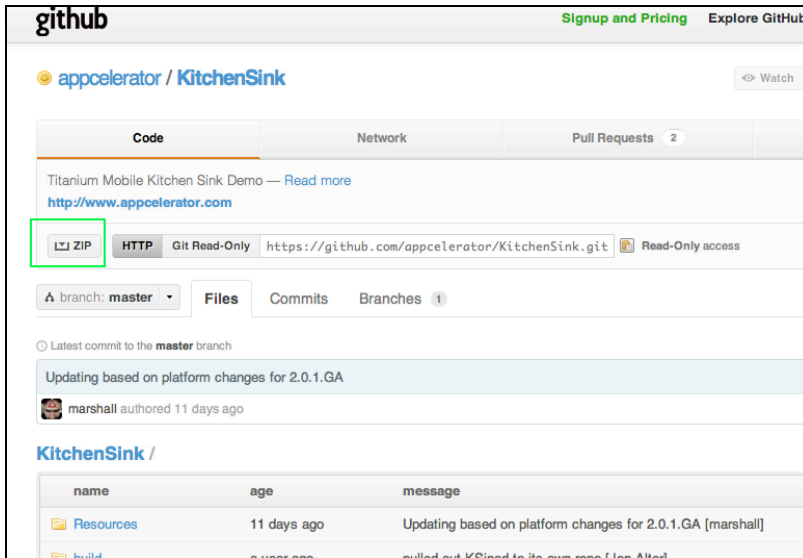
Whichever project is used, note the project's components prior to running the utility. The unarchived KitchenSink project folders are shown below.



Importing the Sample Project

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 for Mac app, or:



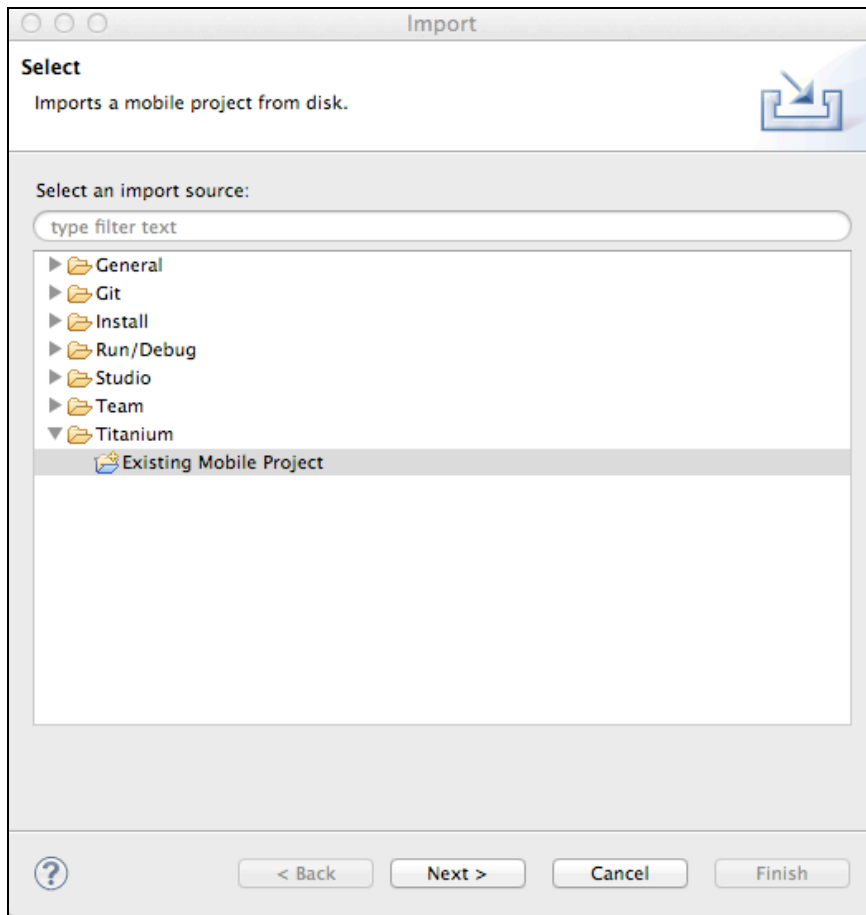
- By using the git command from the command line:

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

Note: The git command must be installed on the given node and in the path to succeed. If git is not in your path, modify the command above accordingly.

1. Once you have the project (or some other one) imported, launch Titanium Studio and login.
2. Click File > Import. The Import box appears.

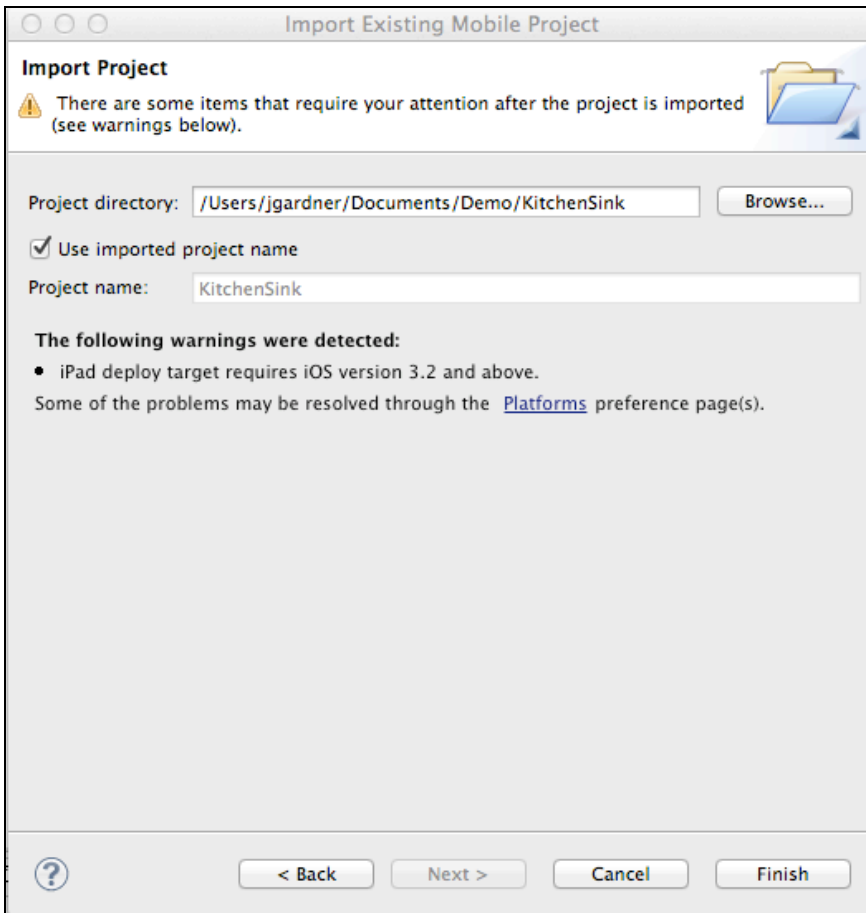
3. Select Titanium > Existing Mobile Project and click next.



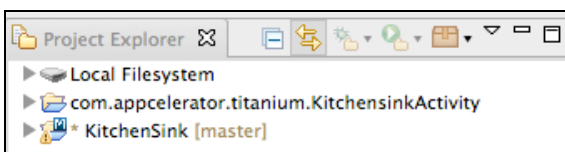
4. Click "Next" to proceed to the Import Existing Mobile Project box.

5. Identify the folder to use as the Project directory.

6. Use the default project name or uncheck the Use imported project name to assign it a new name.



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



7. Exit Titanium Studio if it is running.

Using the MakeAppTouchable Utility

TouchTest™ includes the MakeAppTouchable (MATT) utility, which will automatically add the necessary components to an existing Xcode project file, APP bundle folder, or IPA and make it TouchTestable. The MATT utility will also create the Mobile App entry in the CloudTest® repository.

Note: The CloudTest user specified to run the MakeAppTouchable 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 an existing IPA or APP bundle folder.

This method requires SOASTA 51 (TouchTest 7040.11 or later) and can be applied to iOS versions 6 and 7 only.

- Static instrumentation occurs when MATT instruments a project file (i.e. an Xcode project or in this case to a Titanium project folder).

Static instrumentation is available in all TouchTest releases and for all supported iOS versions.

Applying MATT to an IPA or APP (Dynamic Instrumentation)

Before you can use dynamic instrumentation, you must have either an APP or IPA file with which to work. Obtain the required APP bundle folder or IPA file path prior to attempting the following steps.

TIP: Refer to MakeAppTouchable help (via the Command line using: `java -jar MakeAppTouchable.jar -help`) for more information.

MATT users can dynamically instrument a compiled iOS file by specifying one of the two following flags:

- `-ipa` – The full path to the compiled IPA file
- `-appbundle` – The full path to the compiled APP file

TIP: Provide the additional MATT signing, provisioning, or entitlements file parameters (listed below) whenever they are required by iOS. If you're not the developer, discuss the optional parameters with your developer.

Optional parameters for IPA/APP

The following optional parameters are also available for dynamic instrumentation of IPA and APP files (in some cases their use may be necessary to succeed).

<pre>-signingidentity <signingidentityname></pre>	<p>The name of the signing identity to be used for codesigning the application (e.g. "iOS Distribution: Developer Name").</p> <p>IMPORTANT: MATT can only sign your app with a Distribution profile. This can be an Ad-Hoc Distribution or an Enterprise Distribution profile.</p>
<pre>-provisioningprofile <profilepath></pre>	<p>Path of the Provisioning profile to be used for building IPA file.</p>
<pre>-entitlementsfile <entitlementsfilepath></pre>	<p>Path of the entitlements file to be used for codesigning the application. In the vast majority of cases, the entitlements file is not required.</p>

Making an APP file TouchTestable

The dynamic instrumentation method for APP uses the MATT parameter - appbundle.

1. To dynamically instrument an IPA using the MakeAppTouchTestable utility, run:

```
java -jar MakeAppTouchTestable.jar -appbundle <compiled APP bundle folder> -url <CloudTest URL> -username <CloudTest user name> -password <CloudTest password>
```

where:

- <compiled APP bundle folder> is the full path to the compiled iOS APP bundle file

Here is a complete example using -appbundle:

```
sh MakeAppTouchTestable MakeAppTouchTestable/bin/MakeAppTouchTestable -appbundle ~/Documents/Demo/KitchenSink/build/iphone/build/Debug-
```

```
iphonesimulator/KitchenSink.app -url http://10.0.1.44/concerto -
username bob@acme.com -password secret
```

MakeAppTouchTestable will configure your app, and create a new Mobile App object in the CloudTest repository. The Mobile App object created will have the auto-created URL Scheme in its Launch URL field.

TIP: In this basic example, we do not use MATT's `launchurl` flag to create a launch URL. In which case, MakeAppTouchTestable will auto-generate the URL for us. If the flag is used, be sure to avoid spaces and underscores in the launch URL, as they will cause an error.

If a mobile app object by the name KitchenSink didn't already exist on the CloudTest instance's tenant then the following appears:

```
Processing App...
```

```
App processed successfully.
```

```
TouchTest enabled app is now available at:
/Users/<username>/Documents/Demo/KitchenSink/build/iphone/build/Debug-
iphonesimulator/KitchenSink_TouchTest.app
```

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

If a mobile app object by the name KitchenSink did already exist on the CloudTest instance's tenant then the following appears:

```
Processing App...
```

```
App processed successfully.
```

```
TouchTest enabled app is now available at:
/Users/jgardner/Documents/Demo/KitchenSink/build/iphone/build/Debug-
iphonesimulator/KitchenSink_TouchTest.app
```

```
Already Have A Mobile App Object By This Name. Please Change The Name And
Try Again.
```

```
You should manually create an object corresponding to your Application using
Application ID: touchtest-18284700-303c-428c-9c2d-14aba175dc76://
```

A number of MATT flags are provided to manage the launch URL requirement, including `overwriteapp`, `donotcreateapp`, and `launchurl`. When managing your launch URL(s), it is important to consider whether other users are also using the same mobile app object in CloudTest.

Making an IPA file TouchTestable

The dynamic instrumentation method for IPA uses the MATT parameter `-ipa` and requires the `signingidentity` and `provisioningprofile` flags. Omitting these flags will result in an `.app` file being created rather than the expected TouchTestable `.ipa`.

1. To instrument an IPA using the `MakeAppTouchTestable` utility, run:

```
sh MakeAppTouchTestable MakeAppTouchTestable/bin/MakeAppTouchTestable
-ipa <compiled IPA> -url <CloudTest URL> -username <CloudTest user
name> -password <CloudTest password> -signingidentity "iPhone
Distribution: <Distribution Name>" -provisioningprofile
"/Users/<username>/Documents/Name.mobileprovision"
```

where:

- `<compiled IPA>` is the full path to the compiled iOS `.ipa` file
- `<iPhone Distribution: <Distribution Name>` must be a valid iOS Distribution signing identity
- The provisioning profile file path must refer to a valid provisioning profile

Note: One and only one of the project file, IPA, or app file can be specified when using the MATT utility.

where:

- `<compiled IPA >` is the full path to the compiled iOS `.ipa` file

Here is a complete example using `-ipa`:

```
sh MakeAppTouchTestable MakeAppTouchTestable/bin/MakeAppTouchTestable
-ipa ~/Documents/Demo/KitchenSink/build/iphone/build/Debug-
iphoneos/KitchenSink.ipa -url http://10.0.1.44/concerto -username
bob@acme.com -password secret -signingidentity "iPhone Distribution:
Bob Doe" -provisioningprofile
"/Users/username/Library/MobileDevice/Provisioning Profiles/20454FCB-
7B8C-4996-BDD3-E660DB279261.mobileprovision"
```

If a mobile app object by the name `KitchenSink` didn't already exist on the CloudTest instance's tenant then the following appears:

```
Processing App...
```

App processed successfully.

TouchTest enabled app is now available at:
/Users/jgardner/Documents/Demo/KitchenSink/build/iphone/build/Debug-iphonios/KitchenSink_TouchTest.ipa

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

If a mobile app object by the name KitchenSink did already exist on the CloudTest instance's tenant then the following appears:

Processing App...

App processed successfully.

TouchTest enabled app is now available at:
/Users/jgardner/Documents/Demo/KitchenSink/build/iphone/build/Debug-iphonesimulator/KitchenSink_TouchTest.app

Already Have A Mobile App Object By This Name. Please Change The Name And Try Again.

You should manually create an object corresponding to your Application using Application ID: touchtest-25df433e-d852-4475-bcf3-eeadfb9e5e33://

A number of MATT flags are provided to manage the launch URL requirement, including `overwriteapp`, `donotcreateapp`, and `launchurl`. When managing your launch URL(s), it is important to consider whether other users are also using the same mobile app object in CloudTest.

Making a Titanium Project (Static Instrumentation)

The static instrumentation method for project uses the MATT parameter `project` on the Titanium project directory.

1. In Terminal, navigate to the `MakeAppTouchTestable` folder you created above.

- For example, `cd ~/Documents/Demo/MakeAppTouchTestable`.
- From the `MakeAppTouchTestable` folder, run

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

where:

- `<Titanium project directory>` is the path to the project folder representing your project (for example, the folder where you downloaded KitchenSink). It is not necessary to specify beyond the folder.

Here is a complete example:

```
sh MakeAppTouchTestable MakeAppTouchTestable/bin/MakeAppTouchTestable
-project ~/Documents/KitchenSink -url http://215.77.85.116/concerto -
username bob@acme.com -password secret
```

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.

If a mobile app object by the name KitchenSink didn't already exist on the CloudTest instance's tenant then the following appears:

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

Or, if the mobile app object already exists in the CloudTest instance:

```
Already Have A Mobile App Object By This Name. Please Change The Name And
Try Again.
```

```
You should manually create an object corresponding to your Application using
Application ID: touchtest-627312b8-425a-405b-a83b-fa011d076584://
```

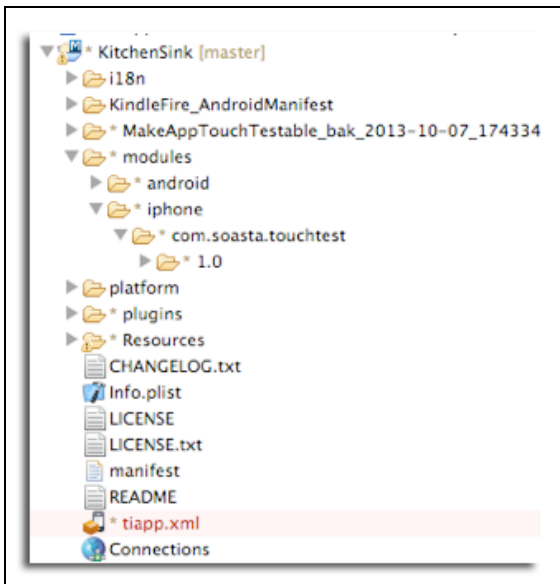
As noted elsewhere, the launch URL of the app on the device must match the launch URL in the CloudTest instance, Mobile App.

A number of MATT flags are provided to manage the launch URL requirement, including `overwriteapp`, `donotcreateapp`, and `launchurl`. When managing your launch URL(s), it is important to consider whether other users are also using the same mobile app object in CloudTest.

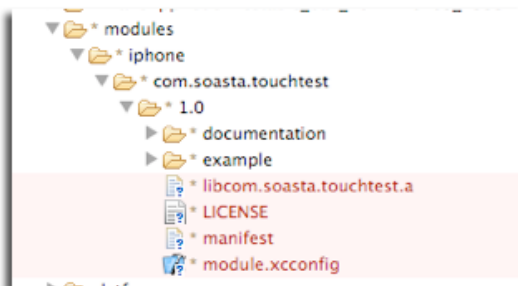
Inspecting Changes to the Titanium Project (Static)

The MakeAppTouchable utility makes some additions and changes related to implementing those changes in your Titanium project. The following additions are made.

- Expand the project folder and note the asterisks (*) indicating changes to the project. These include the new backup folder, the modules folder, and tiapp.xml file modifications that should now be saved.



- TouchTest Module files are added to the modules folder.



- In addition, MakeAppTouchTestable adds the following code to implement the module. In the tiapp.xml file:
 - An iOS plist section has been added to implement the URL scheme:

```
</plugins>
<ios>
  <plist>
    <dict>
      <key>CFBundleURLTypes</key>
      <array>
        <dict>
          <key>CFBundleTypeRole</key>
          <string>Editor</string>
          <key>CFBundleURLName</key>
          <string>com.appcelerator.kitchensink</string>
          <key>CFBundleURLSchemes</key>
          <array>
            <string>touchtest-ad3716ca-9308-4737-9cc9-d9ab89a1f308</string>
          </array>
        </dict>
      </array>
    </dict>
  </plist>
</ios>
```

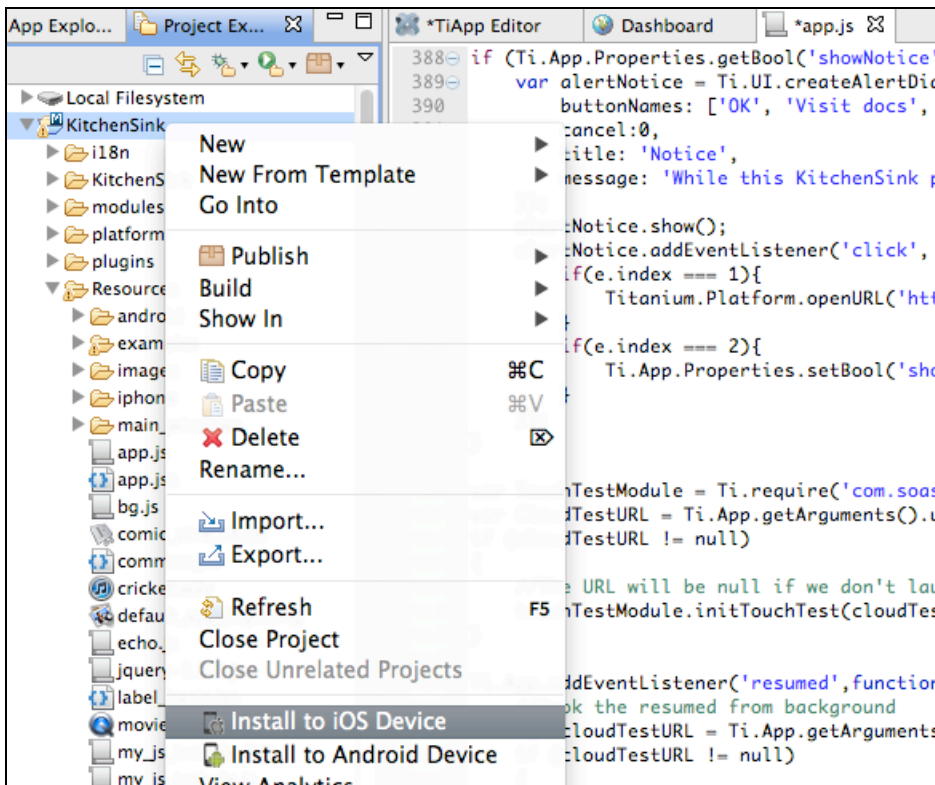
Deploy the TouchTestable App

Choose either the Dynamic or Static method for making the mobile app TouchTestable.

Install from Titanium Studio (Static Instrumentation)

Once the project has been made TouchTestable using the MATT, project flag, you are ready to deploy it to the device or simulator.

1. Select the Project node for the app (i.e. KitchenSink) and right-click to select Install to iOS Device.

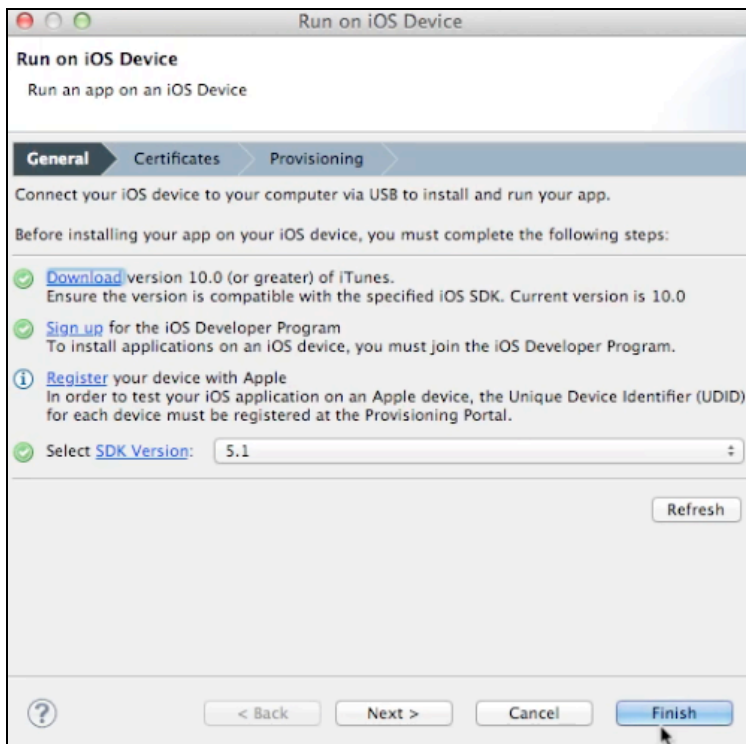


2. The Run on iOS Device box appears.



Note that if the logged in Titanium Studio user is not enrolled in the iOS Developer Program, this is indicated in the box (shown below).

3. Ensure that the iOS device is connected to the desktop client running Titanium Studio.



4. Click Finish to build the project.

```
[DEBUG] copying: /Users/eboudrant/Library/Application Support/Titanium/mobilesdk/osx/2.1.0/iphone/header
[DEBUG] copying: /Users/eboudrant/Library/Application Support/Titanium/mobilesdk/osx/2.1.0/iphone/header
[DEBUG] copying: /Users/eboudrant/Library/Application Support/Titanium/mobilesdk/osx/2.1.0/iphone/header
[DEBUG] copying: /Users/eboudrant/Library/Application Support/Titanium/mobilesdk/osx/2.1.0/iphone/header
[DEBUG] copying: /Users/eboudrant/Library/Application Support/Titanium/mobilesdk/osx/2.1.0/iphone/header
[DEBUG] copying: /Users/eboudrant/Library/Application Support/Titanium/mobilesdk/osx/2.1.0/iphone/header
[DEBUG] copying: /Users/eboudrant/Library/Application Support/Titanium/mobilesdk/osx/2.1.0/iphone/header
[DEBUG] copying: /Users/eboudrant/Library/Application Support/Titanium/mobilesdk/osx/2.1.0/iphone/header
[DEBUG] copying: /Users/eboudrant/Library/Application Support/Titanium/mobilesdk/osx/2.1.0/iphone/header
[DEBUG] copying: /Users/eboudrant/Library/Application Support/Titanium/mobilesdk/osx/2.1.0/iphone/header
[DEBUG] Detecting modules in /Users/eboudrant/Documents/github/Persistence/modules
[DEBUG] Detecting modules in /Users/eboudrant/Library/Application Support/Titanium/modules
[DEBUG] Detected module for android: ti.cloudpush 2.0.1 @ /Users/eboudrant/Library/Application Support/Ti
[DEBUG] Detected module for android: ti.cloudpush 2.0.0 @ /Users/eboudrant/Library/Application Support/Ti
[DEBUG] Detected module for commonjs: ti.cloud 2.0.1 @ /Users/eboudrant/Library/Application Support/Tit
[DEBUG] Detected module for commonjs: ti.cloud 2.0.0 @ /Users/eboudrant/Library/Application Support/Tit
[DEBUG] Detected module for iphone: com.soasta.touchtest 1.0 @ /Users/eboudrant/Library/Application Supp
[DEBUG] Looking for Titanium Module id: com.soasta.touchtest, version: 1.0, platform: iphone
[INFO] Performing clean build
[INFO] Installing application in iTunes ... one moment
[DEBUG] executing command: /Applications/Xcode.app/Contents/Developer/Platforms/iPhoneOS.platform/Devel
```

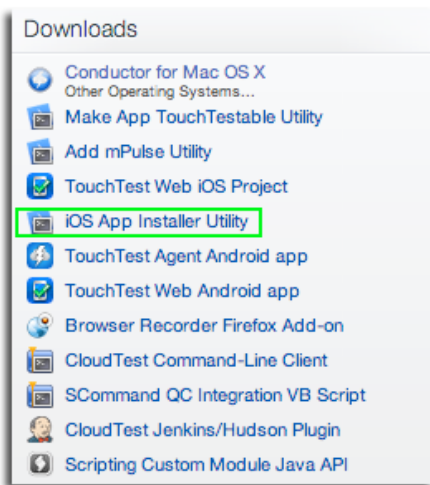
TIP: Be sure to sync via iTunes, ensuring that the device's Apps tab has KitchenSink set to "Install." This must be done each and every time that you use the Install to iOS Device command.

Install from the Command Line (Dynamic Instrumentation)

Using the newly TouchTestable APP or IPA you can now easily:

- Install the TouchTestable app to an iOS device or simulator.

SOASTA provides the iOSAppInstaller Utility for this purpose. You can download the latest iOSAppInstaller Utility from the Central > Welcome page, Downloads section of your own server instance.



Unzip the utility at this time if you have yet to do so and note the contents of the resulting iOSAppInstaller folder.

Unzip the utility at this time if you have yet to do so and note the contents of the resulting iOSAppInstaller folder.

- For a Simulator, use:

```
./bin/ios_sim_launcher --appbundle <path to KitchenSink_TouchTest.app file>
```

- For an iPhone or iPad, use:

```
./bin/ios_app_installer --ipa <path to KitchenSink_TouchTest.ipa file>
```

Registering Your Device to Use TouchTest™

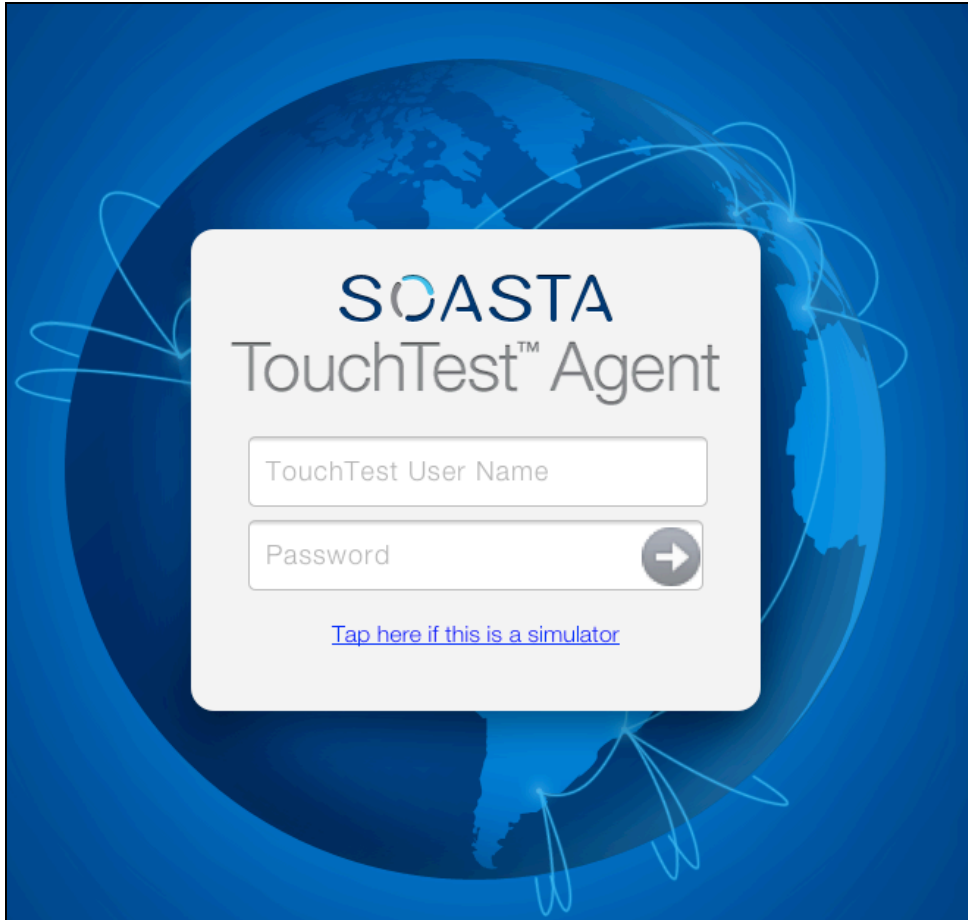
The TouchTest™ Agent is responsible for launching the apps that are being tested. It is a web application that is served from the CloudTest server and runs in mobile Safari on iOS devices. To get started, browse to the TouchTest Agent URL on the mobile device and perform the one-time registration steps that will enable your device for use with TouchTest.

Note: If you clear your cookies on the given mobile device after registration, you may need to register your device again so that TouchTest™ can recognize it. This does not consume an additional license.

1. On the mobile device, launch Safari and point it to:

http://<CloudTest URL>/concerto/touchtest

The following screen appears.



TIP: If you're using a Simulator that is running an iOS version prior to iOS 7, use the "Tap here if this is a simulator" link (shown above) to proceed. This link will appear below the Login button in all configurations that require it. This link doesn't appear in iOS 7 or after.

2. Login using your SOASTA CloudTest user name and password.

If the device is not registered, the Register Device page below appears.

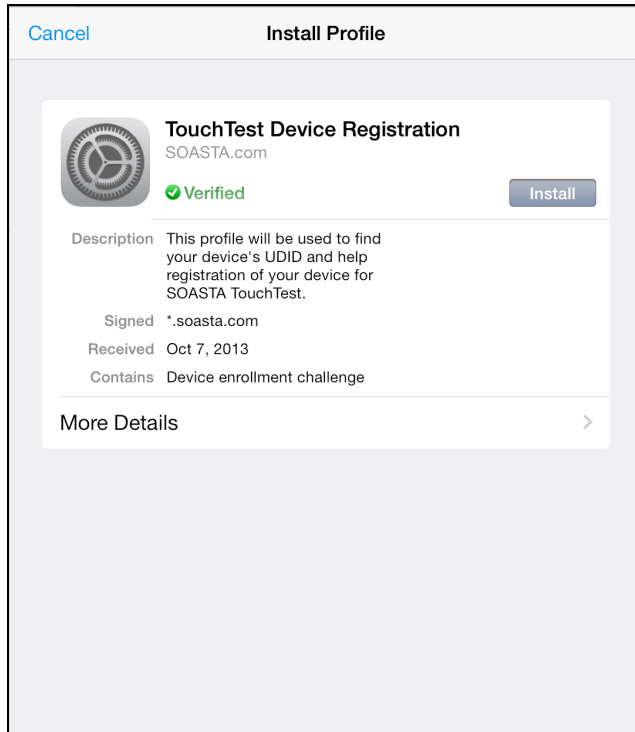
Note: If you clear your cookies, you may need to register your device again so that TouchTest™ can recognize it. This does not consume an additional license.



The Unique Device Identifier (UDID) will be used to register the mobile device for use with TouchTest™ in iOS7 or after, for earlier iOS versions, the registration will also use a profile.

3. Click the Register Device button to continue.

- a. First, the Install Profile screen appears. Click the Install button to proceed.



- b. An alert appears to indicate that mobile device settings will be changed. Click Install Now to proceed.
- c. If a passcode is in effect on the mobile device, an additional prompt will appear to authorize the profile installation.

4. When prompted, give the TouchTest Agent a name. For example , in our screenshots the device is *SOASTA Demo iPad*. 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.
5. Once this name is entered, click Submit for administrator approval.

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

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.

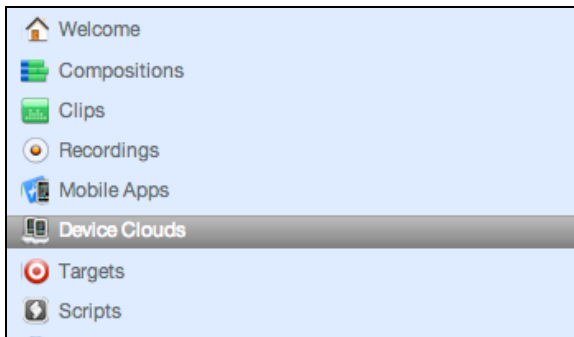
If the Mobile Device Administrator approves your device, the "Connected" page will appear the first time TouchTest™ is launched in Safari on the approved device. On subsequent launches click Login to Connect and Logout to Disconnect.



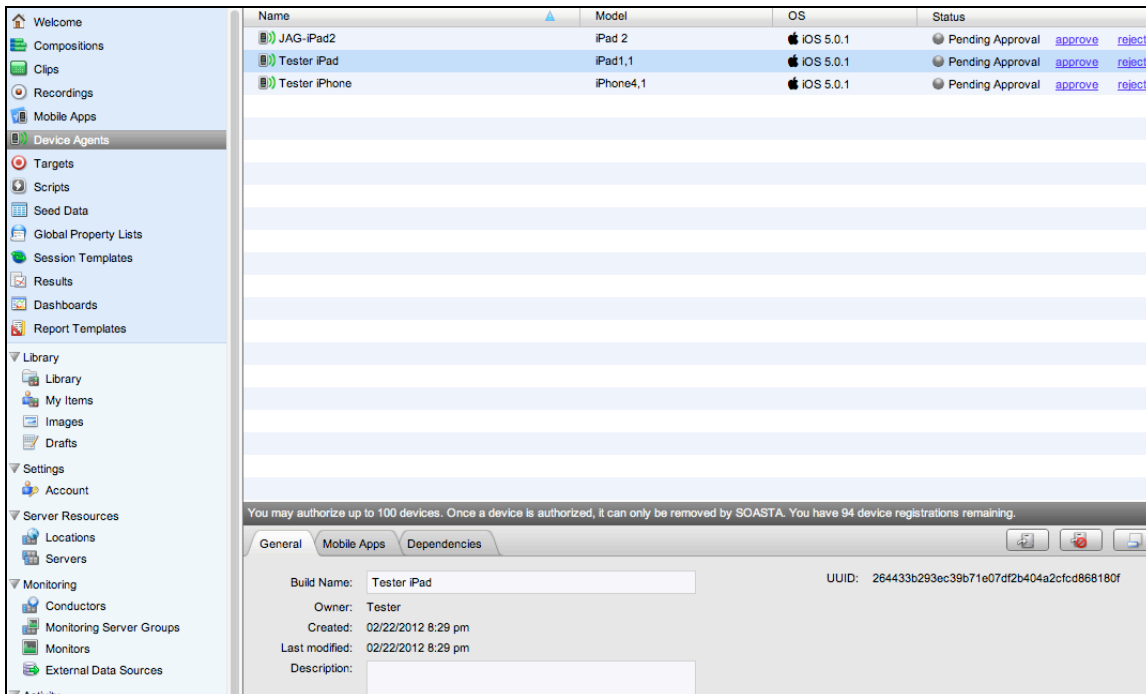
Approving a Mobile Device (Administrator Only)

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 Agents 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.

A screenshot of the TouchTest Mobile Device Administrator interface. The left sidebar shows the navigation menu with 'Device Agents' highlighted. The main area displays a table of device agents in a queue. Below the table, there is a detailed view of a selected device.

Name	Model	OS	Status
JAG-iPad2	iPad 2	iOS 5.0.1	Pending Approval approve reject
Tester iPad	iPad1,1	iOS 5.0.1	Pending Approval approve reject
Tester iPhone	iPhone4,1	iOS 5.0.1	Pending Approval approve reject

You may authorize up to 100 devices. Once a device is authorized, it can only be removed by SOASTA. You have 94 device registrations remaining.

General Mobile Apps Dependencies

Build Name: Tester iPad UID: 264433b293ec39b71e07df2b404a2cfd868180f

Owner: Tester

Created: 02/22/2012 8:29 pm

Last modified: 02/22/2012 8:29 pm

Description:

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

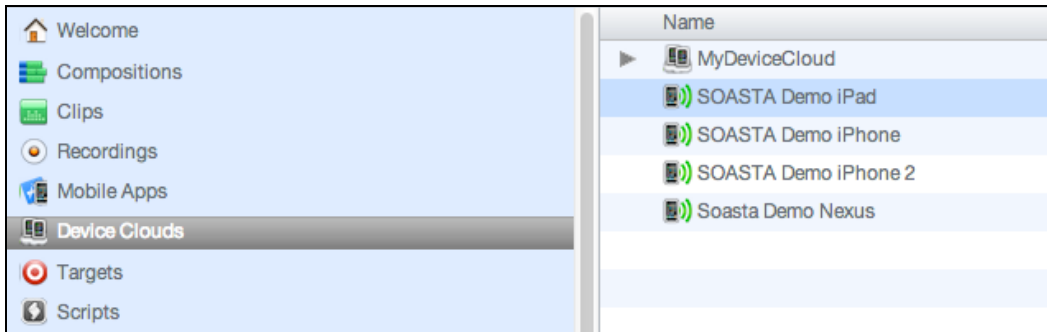
3. Click Approve to complete adding a device and Reject to deny its access.

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.

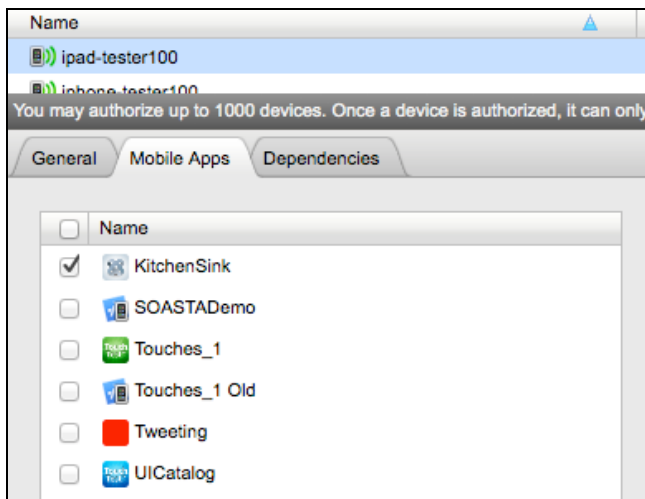
Associating Mobile Apps with a Device

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. Locate and check the Mobile App(s) that you want to authorize this device to access. For example, KitchenSink.

4. Click Save on the lower panel toolbar.

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.

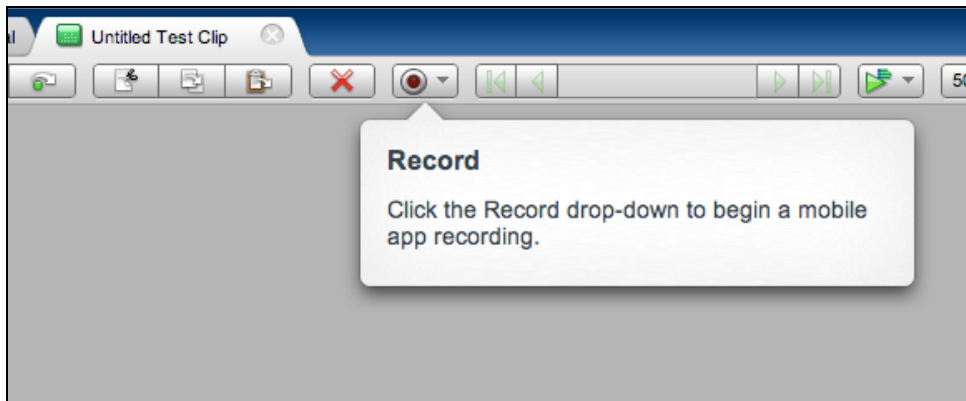
These and the following additional configuration steps are described in the remainder of this tutorial.

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

Create a Simple TouchTest™ Clip using KitchenSink

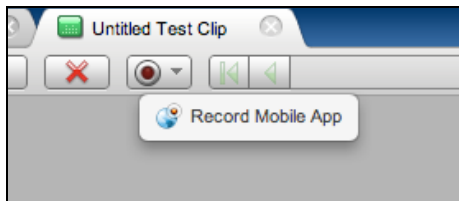
Create a new clip that will be used to perform mobile app recording and serve as the basis for your first test composition.

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

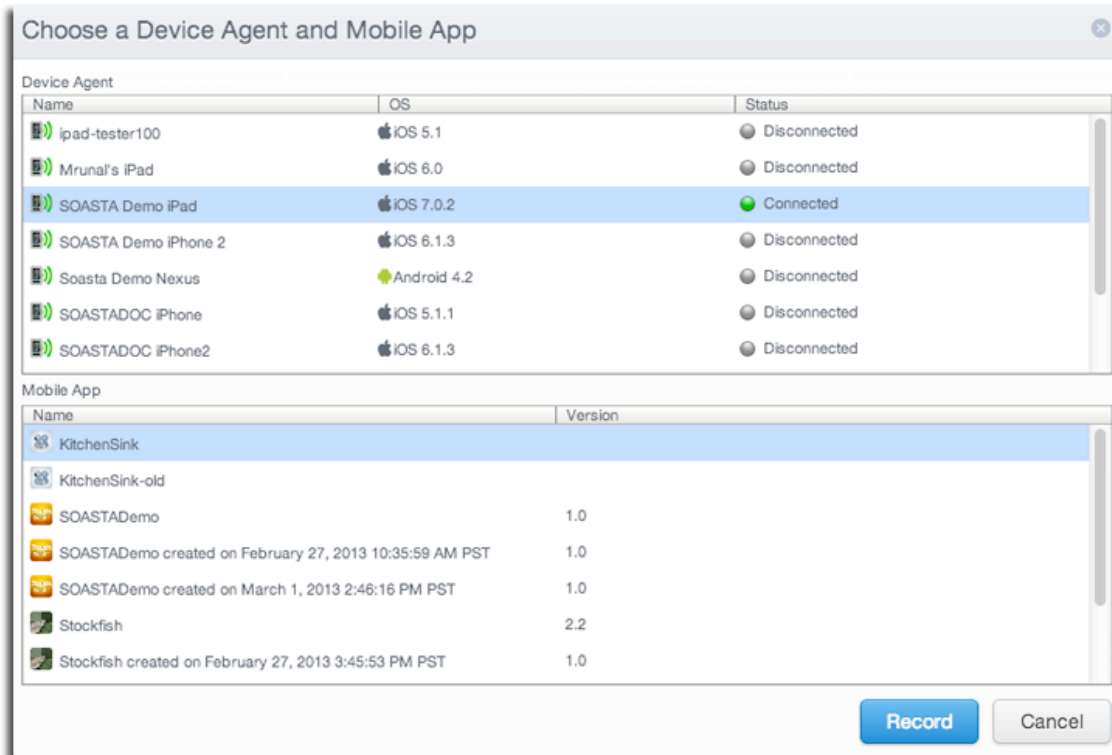


A new Untitled Test Clip opens in a Clip Editor tab. A Record pop-up identifies the Record drop-down.

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 launches in Safari on your mobile device.

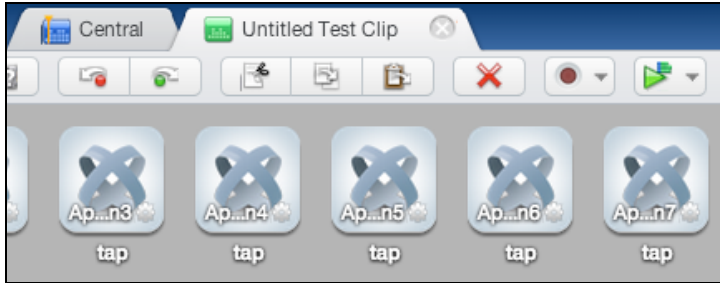


Once successfully logged on, its Status will be *Connected*.

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 Button
 - On the Button screen, tap the button that says "Click Me" and then tap it three more times.
 - Return to Controls (top left)
 - Tap Text Field
 - Tap Events
 - Tap Focus
 - When the text field has focus, enter "Test" and tap the Return key

- Tap Test Field (top left) and Controls (also top left) to return to the original view.

For each app action you perform, the Clip Editor adds an app action to the clip.



- Once the relevant interactions have been recorded, click the Record button again to stop the recording (the Record button will appear as above after recording is stopped).
- 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.

 A screenshot of the Clip Editor List view. The toolbar at the top includes a 'List' button. Below the toolbar is a table with the following columns: Name, Operation, Parameter 1, and Parameter 2. The table contains 13 rows of app actions.

Name	Operation	Parameter 1	Parameter 2
App Action1	tap	text=Button	{"tapCount":"1","duration":"0.1...0,26
App Action2	tap	text=Click Me	{"tapCount":"1","duration":"0.0...0,28
App Action3	tap	text=I am Enabled	{"tapCount":"1","duration":"0.1...0,32
App Action4	tap	text=I am red	{"tapCount":"1","duration":"0.0...0,29
App Action5	tap	text=White text	{"tapCount":"1","duration":"0.0...0,29
App Action6	tap	text=Controls[2]	{"tapCount":"1","duration":"0.0...0,19
App Action7	tap	text=Text Field	{"tapCount":"1","duration":"0.1...0,25
App Action8	tap	text=Events	{"tapCount":"1","duration":"0.1...0,23
App Action9	tap	text=Focus	{"tapCount":"1","duration":"0.1...0,22
App Action10	type	classname=TITextField	Test
App Action11	keybdRtrn	classname=TITextField	
App Action12	tap	text=Text Field[2]	{"tapCount":"1","duration":"0.1...0,23
App Action13	tap	text=Controls[2]	{"tapCount":"1","duration":"0.0...0,15

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

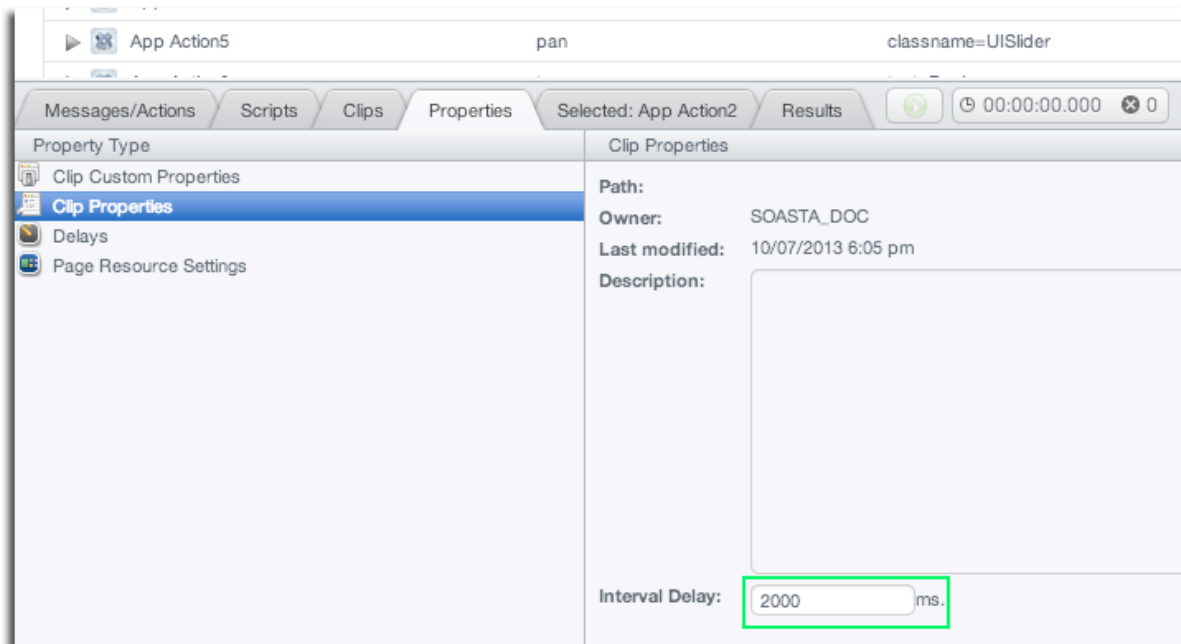
TIP: It is possible to proceed to create and play a composition at any point after Recording is completed.

Adding an Interval Delay between Each Action

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

Imposing delays using the Interval Delay setting can make the test more viewable during test playback.

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.

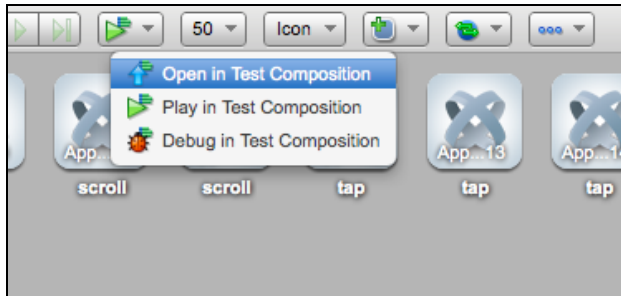


4. Click Save on the Clip Editor toolbar.

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.

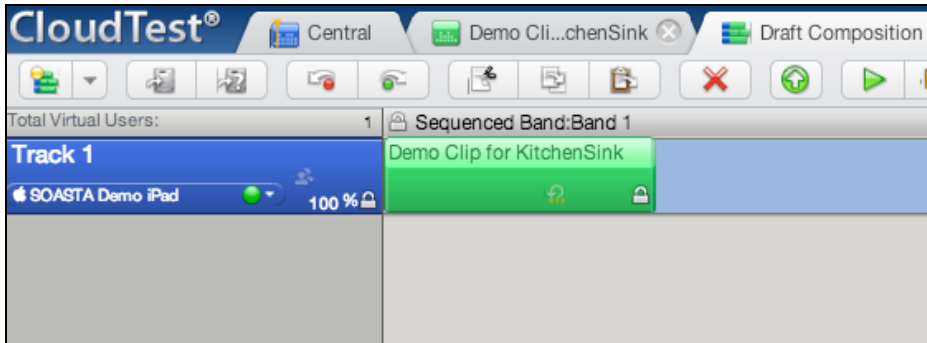
- **Play in Test Composition**

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's Location dropdown (i.e. SOASTA Demo iPad is selected in that field above).

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

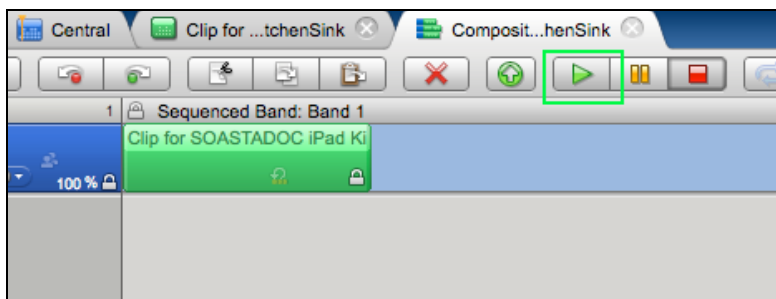
2. Click Save on the Composition Editor toolbar and give the test composition a name or accept the default name.

For example, *Demo Composition for KitchenSink*.

Playing a Composition

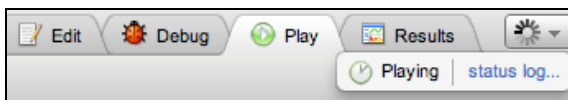
Perform these additional steps while the test composition is open in the Composition Editor.

1. Ensure that the TouchTest Agent status is “Connected” on the mobile device via Safari.
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.



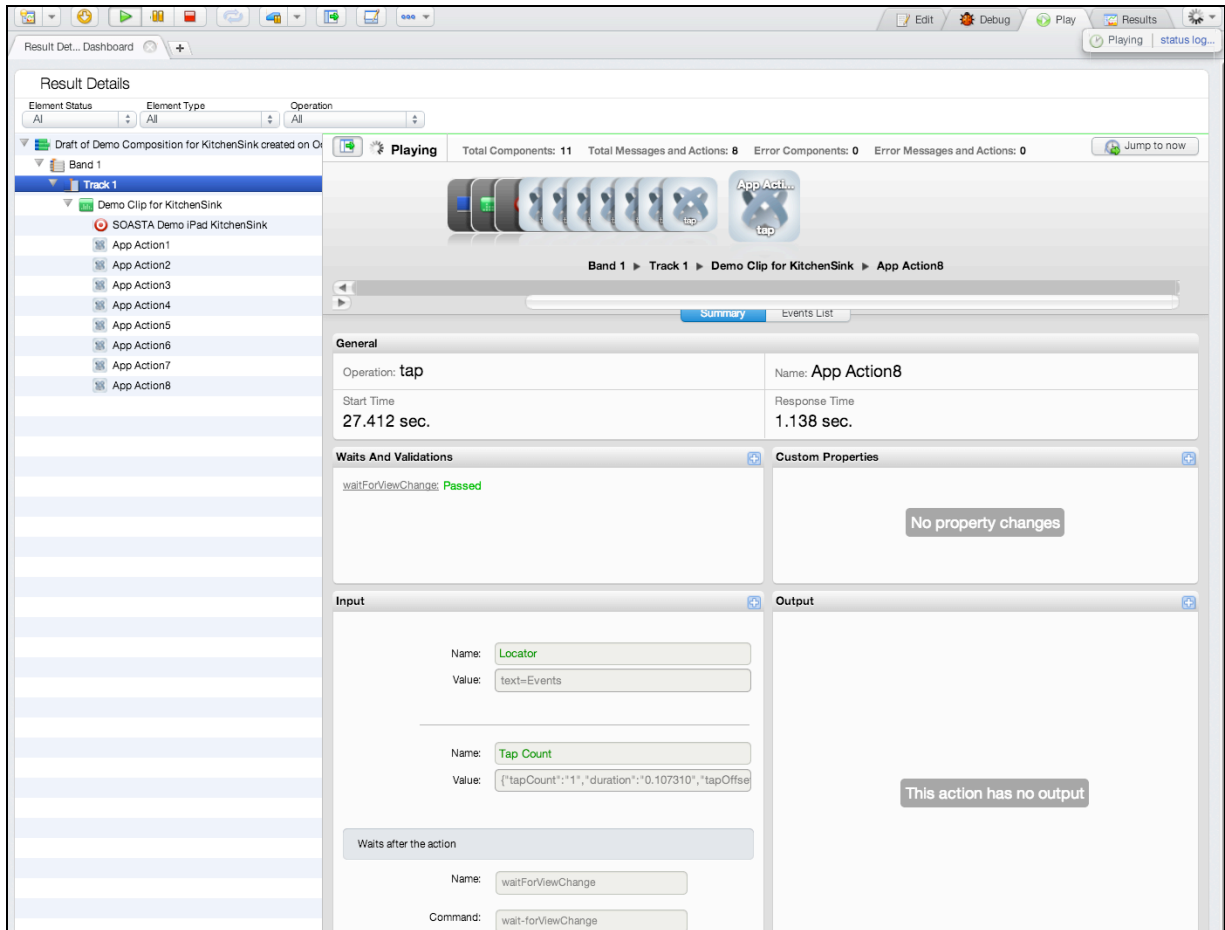
While the test runs, the Composition Editor automatically switches to the Play tab, and by default, the Result Details dashboard displays.

The Composition Editor’s Status Indicator changes to “Playing.” Meanwhile, the mobile app is launched on the specified mobile device(s) and the actions in the test clip are repeated precisely as it was recorded.



Result Details

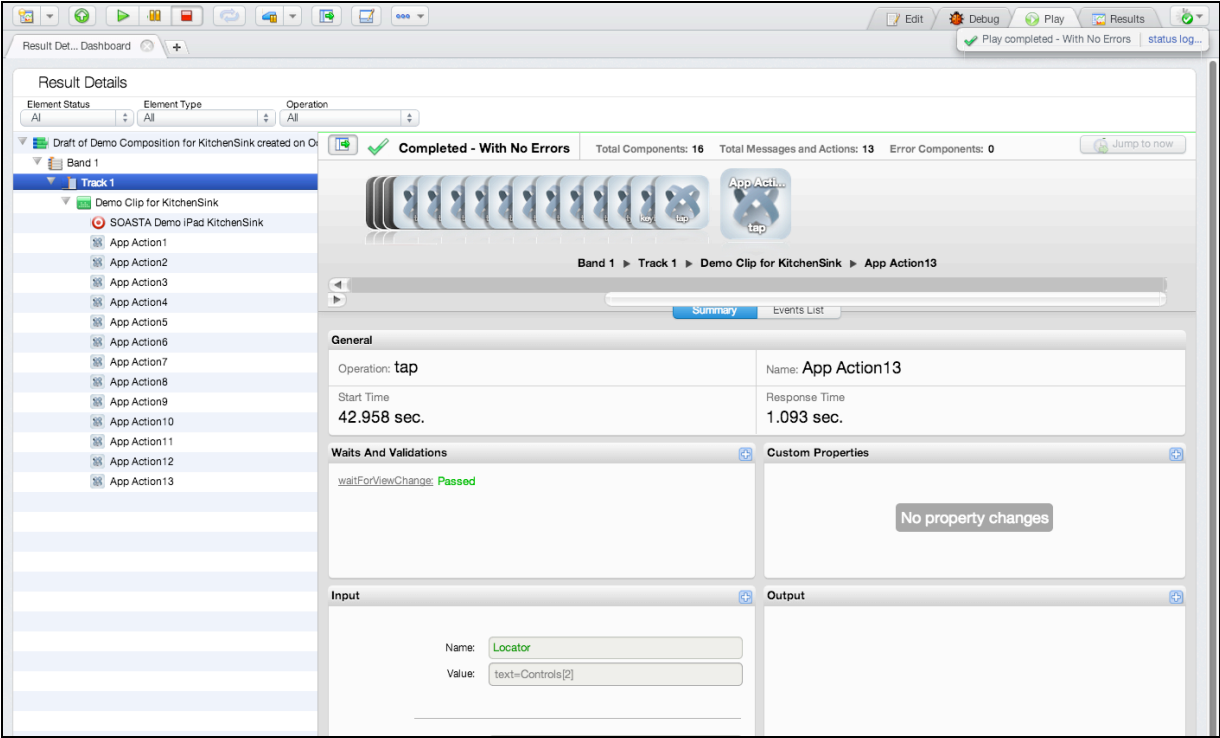
The Result Details dashboard helps to discover the cause of errors in your test, if any.



While play continues results are posted in the Composition Editor, Play tab, Result Details widget.

If the composition fails for any reason, the Result Details dashboard clearly indicates that and presents details as to why.

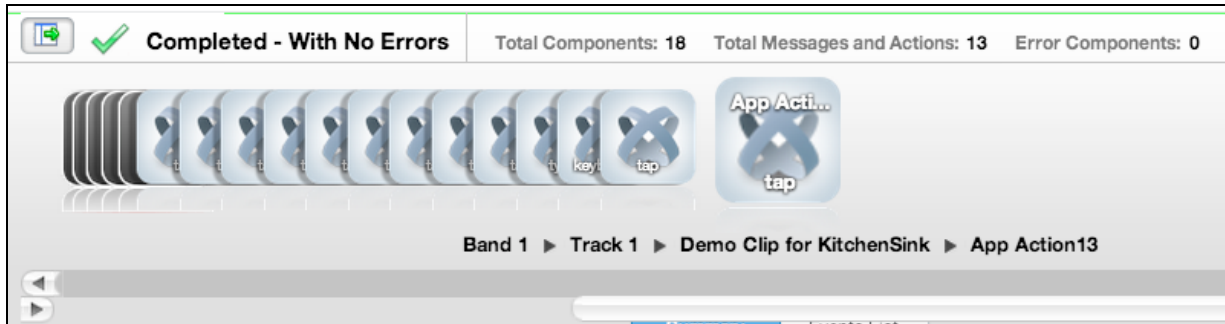
For a simple composition such as this one, using the one clip you recorded above, success is the most likely outcome.



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.



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 in the Summary panel below.

- Locate and select App Action2, which has the locator, *text=Click Me* (depending how closely you followed the suggestion above this locator might be in a different action, whatever the case try and locate it).

The screenshot displays the 'App Action2' configuration page in a testing tool. At the top, there is a breadcrumb trail: 'Band 1 > Track 1 > Demo Clip for KitchenSink > App Action2'. Below this, there are two tabs: 'Summary' (selected) and 'Events List'. The main content is divided into several sections:

- General:** A table with two columns. The left column contains 'Operation: tap' and 'Start Time: 8.369 sec.'. The right column contains 'Name: App Action2' and 'Response Time: 610 ms.'.
- Waits And Validations:** A section with a plus icon. It contains a single entry: 'builtin-waitForGestureComplete: Passed'.
- Custom Properties:** A section with a plus icon. It contains a message: 'No property changes'.
- Input:** A section with a plus icon. It contains two input fields:
 - Name: Locator (highlighted in green), Value: text=Click Me
 - Name: Tap Count (highlighted in green), Value: {"tapCount": "1", "duration": "0.090079", "tapOffse
- Output:** A section with a plus icon. It contains a message: 'This action has no output'.

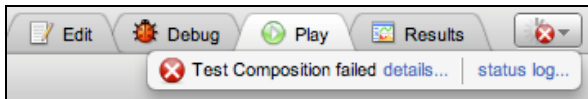
Identifying and Analyzing Common Errors

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).

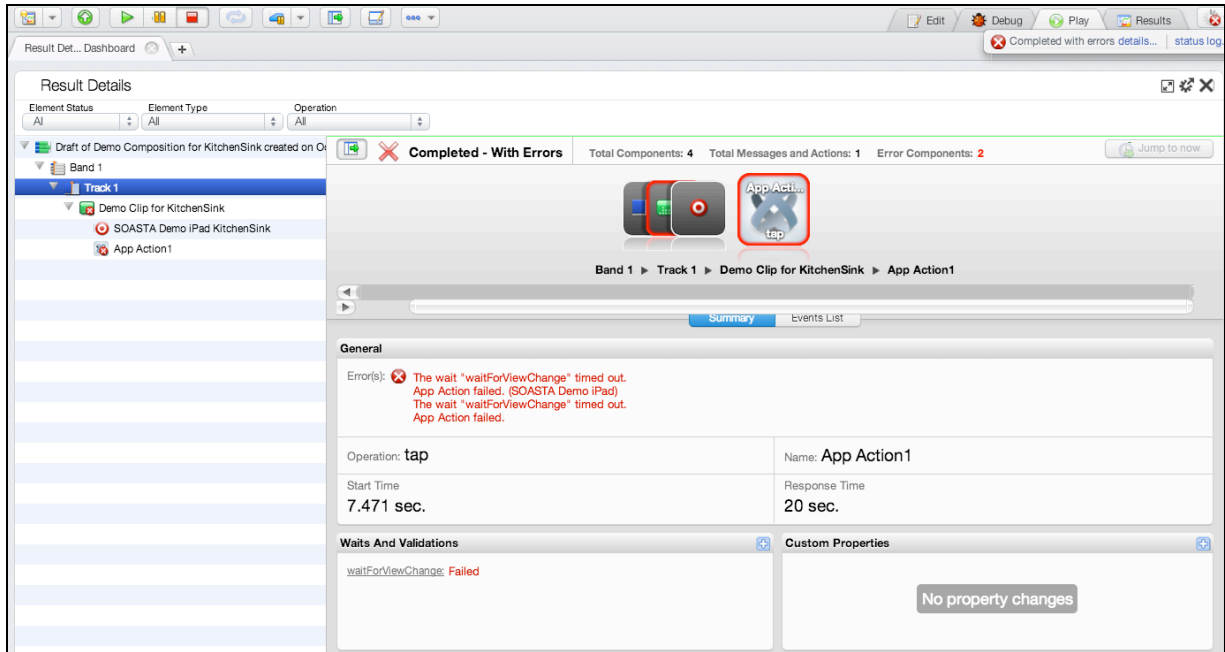


- Click Details to display additional information in a dialog box.

In some cases, the TouchTest Agent may have been started but is no longer responding (such as was the case with the Device Agent timeout error shown below). In such cases, refresh the Safari page for the TouchTest Agent, or 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 when the error item is selected in Result Details.

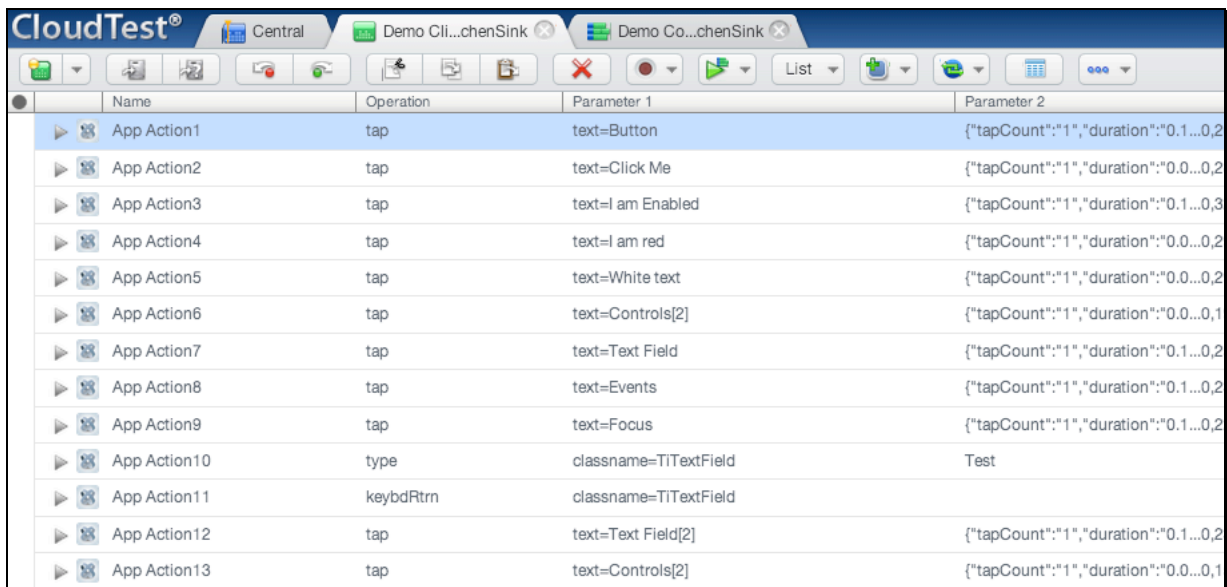
Note: In the case of the above error, it was created intentionally by changing the locator in App Action1.

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.

The List view is useful while clip editing, because it shows all the parameters (8th column to the right) and their corresponding inputs in one tabular view.



The screenshot shows the CloudTest interface with a table of test actions. The table has five columns: Name, Operation, Parameter 1, and Parameter 2. The actions are listed from App Action1 to App Action13. The operations are 'tap' or 'type'. The parameters are text-based or JSON objects.

Name	Operation	Parameter 1	Parameter 2
App Action1	tap	text=Button	{"tapCount": "1", "duration": "0.1...0.2"}
App Action2	tap	text=Click Me	{"tapCount": "1", "duration": "0.0...0.2"}
App Action3	tap	text=I am Enabled	{"tapCount": "1", "duration": "0.1...0.3"}
App Action4	tap	text=I am red	{"tapCount": "1", "duration": "0.0...0.2"}
App Action5	tap	text=White text	{"tapCount": "1", "duration": "0.0...0.2"}
App Action6	tap	text=Controls[2]	{"tapCount": "1", "duration": "0.0...0.1"}
App Action7	tap	text=Text Field	{"tapCount": "1", "duration": "0.1...0.2"}
App Action8	tap	text=Events	{"tapCount": "1", "duration": "0.1...0.2"}
App Action9	tap	text=Focus	{"tapCount": "1", "duration": "0.1...0.2"}
App Action10	type	classname=UITextField	Test
App Action11	keybdRtrn	classname=UITextField	
App Action12	tap	text=Text Field[2]	{"tapCount": "1", "duration": "0.1...0.2"}
App Action13	tap	text=Controls[2]	{"tapCount": "1", "duration": "0.0...0.1"}

TIP: In the above, note that the assigned locators for App Action2, App Action3, and App Action4, all of which correspond to the button taps we performed, use the text=*

When additional parameters are present they are displayed to the right of the Parameter 1 column.

Planning a Complex Test in KitchenSink

Your first clip using the sample app (or your own app) was probably relatively simple. CloudTest clips of great complexity can also be devised.

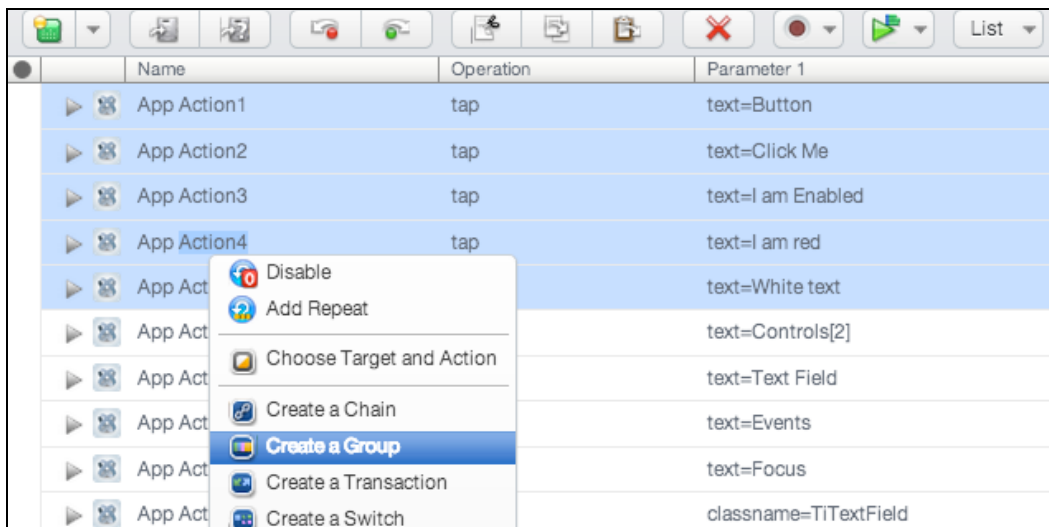
For example, in the sample clip shown below, we recorded actions from Buttons and Text Fields.

In the remaining sections of this tutorial, we will add outputs, validations, and then finally, we will customize a locator using a technique called "TouchTestID". TouchTest Driver is already in place in projects where MakeAppTouchTestable has been applied and provides TouchTestID support.

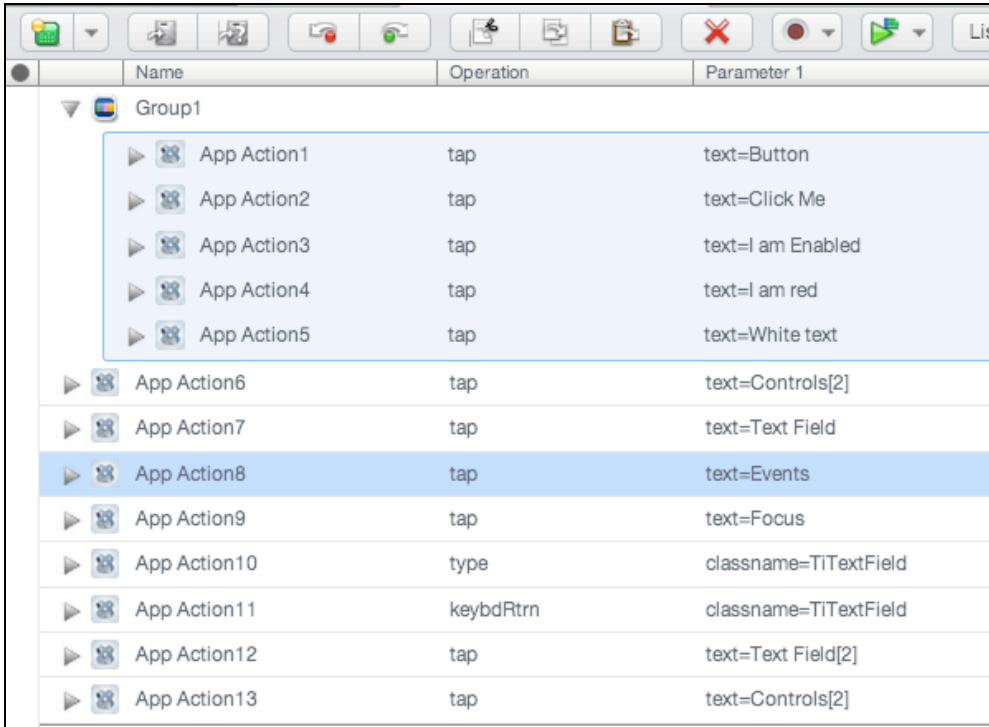
Increasing Test Clip Complexity

1. Open the clip created in the steps above in the Clip Editor.

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



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



The screenshot shows a software interface with a toolbar at the top and a table below. The table has three columns: 'Name', 'Operation', and 'Parameter 1'. A group named 'Group1' is expanded, showing a list of 13 app actions. The first five actions (App Action1 to App Action5) are highlighted with a blue background, indicating they are selected. The other actions (App Action6 to App Action13) are not highlighted.

Name	Operation	Parameter 1
Group1		
▶ App Action1	tap	text=Button
▶ App Action2	tap	text=Click Me
▶ App Action3	tap	text=I am Enabled
▶ App Action4	tap	text=I am red
▶ App Action5	tap	text=White text
▶ App Action6	tap	text=Controls[2]
▶ App Action7	tap	text=Text Field
▶ App Action8	tap	text=Events
▶ App Action9	tap	text=Focus
▶ App Action10	type	classname=UITextField
▶ App Action11	keybdRtrn	classname=UITextField
▶ App Action12	tap	text=Text Field[2]
▶ App Action13	tap	text=Controls[2]

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 the recorded AppAction2 is open in the lower panel. The type of app action, *type*, represents the user name entered on the SOASTA Demo app login page.

1. In the Clip Editor, locate and double-click the app action you already located above (the one with the "Click Me" button).

In our sample clip, this is *Selected: AppAction2*. This app action has five Inputs: Locator, Tap Count, Touch Count, Duration, and Tap Offset.

The screenshot displays the SOASTA Clip Editor interface. At the top, a list of app actions is shown, with 'App Action2' selected. Below this, the 'Selected: App Action2' tab is active, showing the details for this specific action. The interface is divided into several panes:

- Messages/Actions:** Shows a list of app actions with columns for name, type, text, and a JSON snippet.
- Scripts:** Shows the script for the selected action.
- Clips:** Shows the clip details for the selected action.
- Properties:** Shows the properties of the selected action, including its type and inputs.
- Results:** Shows the results of the action, including a timer and a counter.

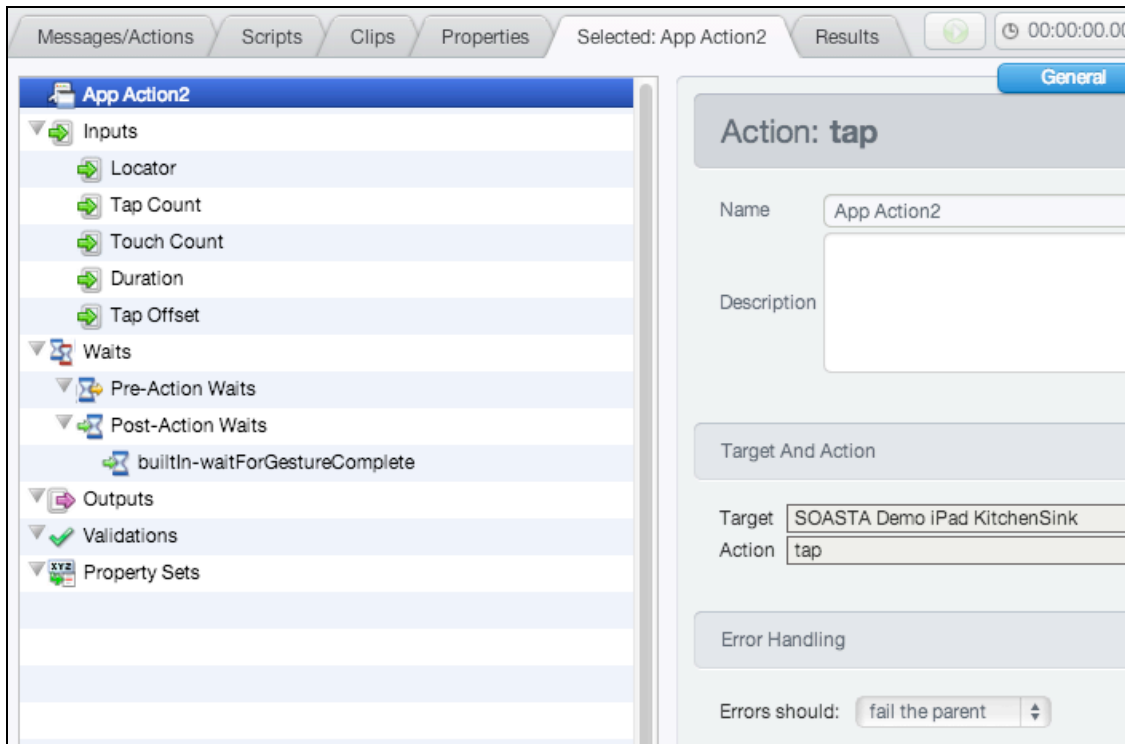
The 'Properties' pane is expanded to show the 'Inputs' section, which includes:

- Locator: text=Click Me
- Tap Count: 1
- Touch Count: 1
- Duration: 0.090079
- Tap Offset: 163.000000,28.500000

App Action Elements and Properties

While the *Selected: Action* tab is 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 the given 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.


TIP: These settings are also found inline in the Clip Editor, List view whenever an app action is expanded. In the sections below, we will add complexity to app actions using both methods.

-  **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 has 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*

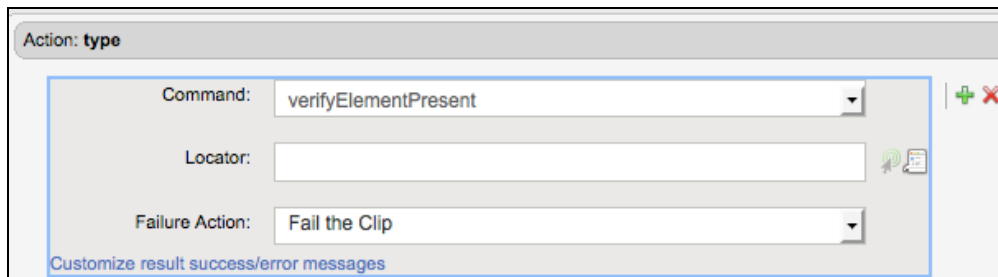
Global properties are defined within the Central > Global Properties List and are “global” within the entire SOASTA CloudTest environment—and can be used across compositions.

Adding a Text Validation

Next, we will add a validation on AppAction1. 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 App Action1 of our sample clip we were on the main Controls page, and the first action we took was to tap Button.
2. If it's not already open in the lower panel, open App Action1 now by double-clicking.
3. Click Validations in the list and then click the green Plus sign  in the Validations panel on the right.

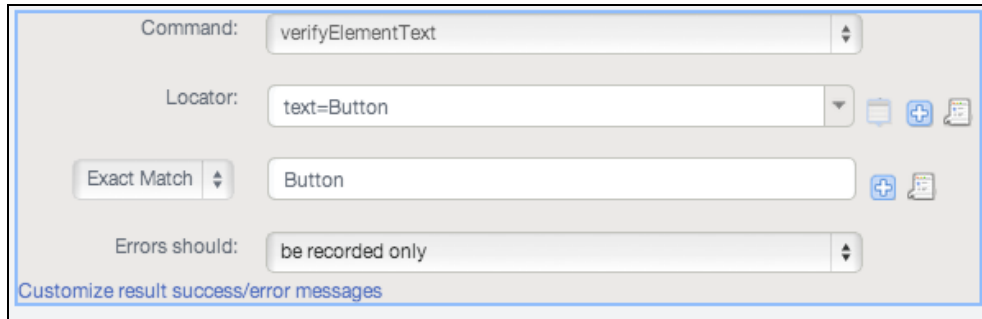
An Action: type form is added to the right panel.



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=Button*.
6. In the Match field, accept *Exact Match* and enter the expected text. For example, *Button*.



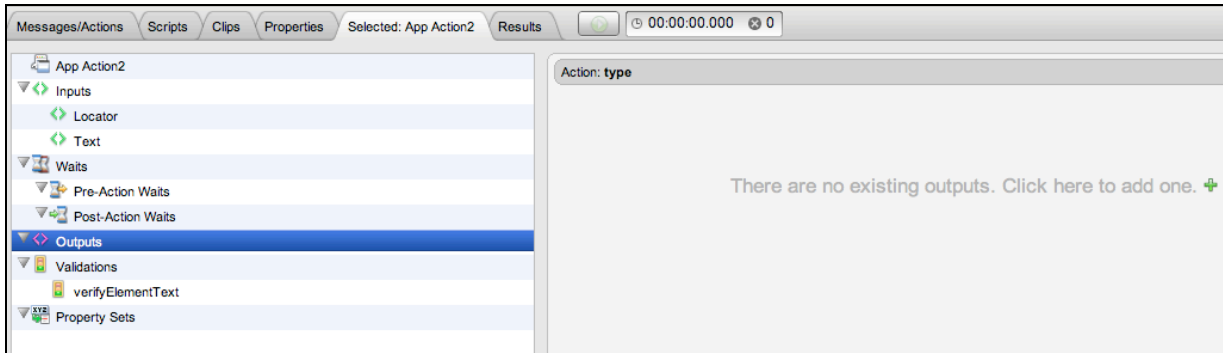
7. **TIP:** If you're not sure which app action pertains to an action you took while recording, try adding an Output, captureScreenshot, and then play the composition. The screenshot is shown in the Result Details, Outputs section. See Adding an Output (next heading) if you're not sure how to do this.

8. Leave the default *Fail the Clip* set in the Failure action field.
9. Click Save on the Clip Editor toolbar.

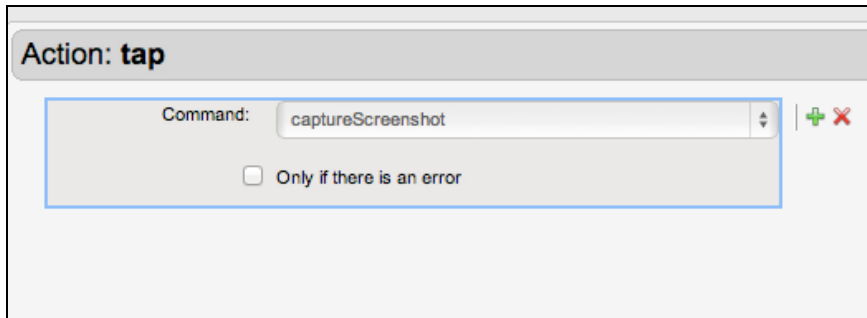
Adding an Output

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. The default output is *captureScreenshot*.

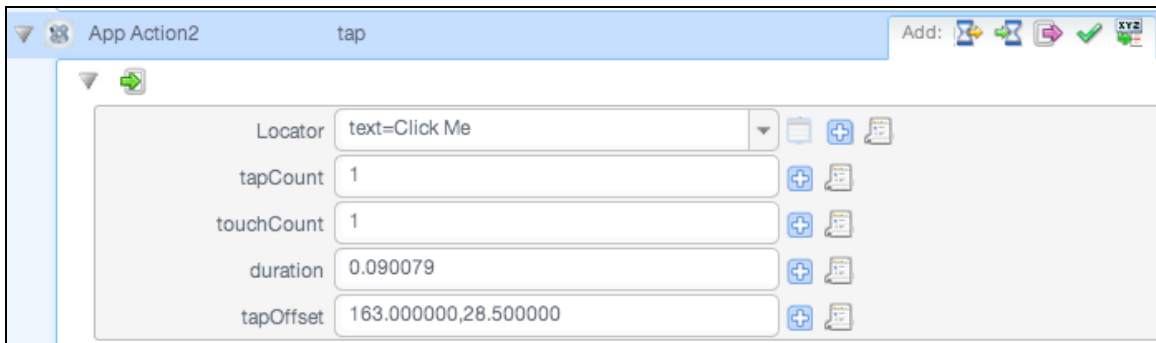


3. Check *Only if there is an error* if the output is desired only in the event this AppAction produces an error. Otherwise, leave it unchecked. For our example, we will leave it unchecked to ensure we get an output in every eventuality.
4. Click the Green Plus (+) icon and add a second Output.
5. In the Command list, scroll to the bottom and set to outputXMLHierarchy.
6. Click Save on the Clip Editor toolbar.

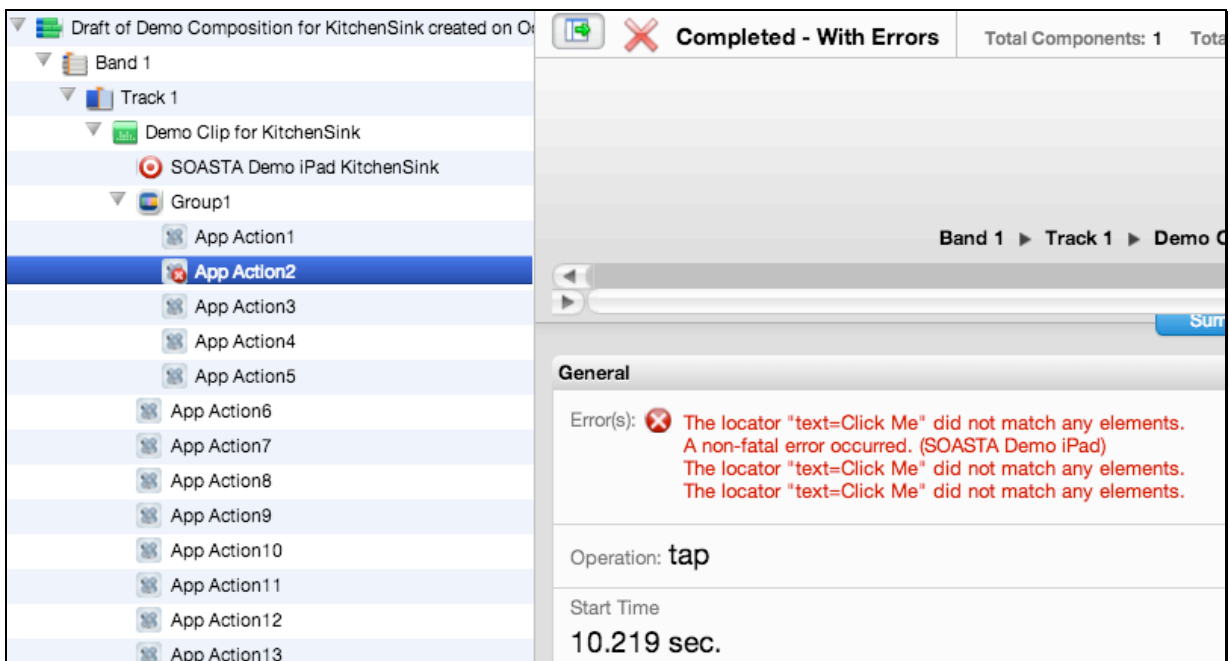
Adding a Label Validation

Finally, we will add a label validation on App Action2. In the subsequent sections, we will then alter this label using a TouchTestID.

1. Select and expand App Action2 in the Clip Editor, List view.



Note: The default locator assigned to App Action2 by TouchTest uses the button label text, Click Me. This is fine in most cases, however, when we perform the tap action the button's label changes to "I am Enabled." Therefore, validating that label text using the locator "text=Click Me" will fail (shown below).



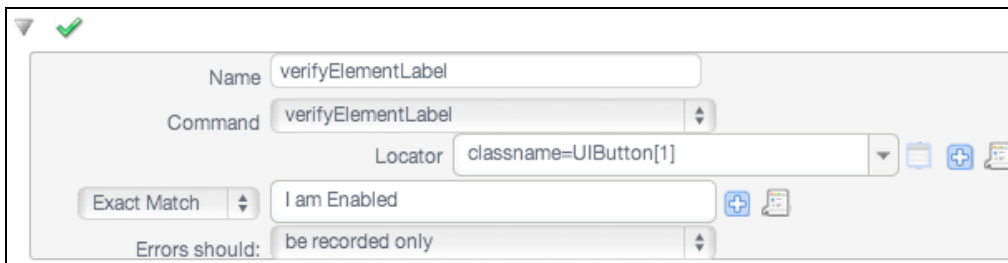
2. Inspect the locator array for App Action2 by selecting the Locator drop down. Note that it includes `classname=UIButton[1]`. We will use this as the

specified locator in the label validation, instead of the text value that is present only before we perform this action.

3. Hover your mouse over the expanded App Action2 and then click the green checkmark to add a validation form into the workspace.



4. In the newly added Validations form, change verifyElementPresent to verifyElementLabel.
5. In the Locator field, enter the locator, classname=UIButton[1].



6. Click Save on the Clip Editor toolbar.
7. Return to the Composition Editor tab once again and click Play.

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.

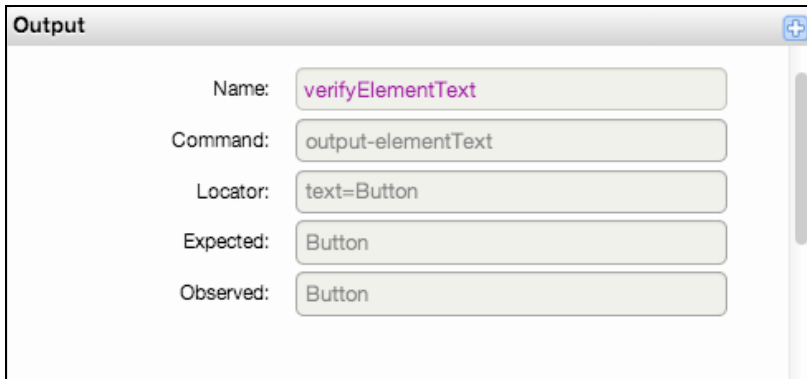
- Expand the Navigation Tree, including Group1, until App Action1 is in display and then select it as shown below.

The screenshot displays the test results interface. On the left, a navigation tree is expanded to show 'App Action1' selected. The right panel shows the test results for 'App Action1', including the operation type 'tap', the start time '3.921 sec.', and the status of 'Waits And Validations' (both 'verifyElementText' and 'waitForViewChange' are 'Passed').

Section	Item	Status
General	Operation	tap
General	Start Time	3.921 sec.
Waits And Validations	verifyElementText	Passed
Waits And Validations	waitForViewChange	Passed

1. Note that for App Action 1 (selected on the left) that the text validation passed. This is noted on the Summary tab, Waits and Validations section.

2. In the Waits and Validations section, click `verifyElementText` and note that details in the Outputs section are highlighted and scrolled for your inspection.



3. Scroll down, if necessary, to view the `captureScreenshot` output defined for App Action1 in our example.

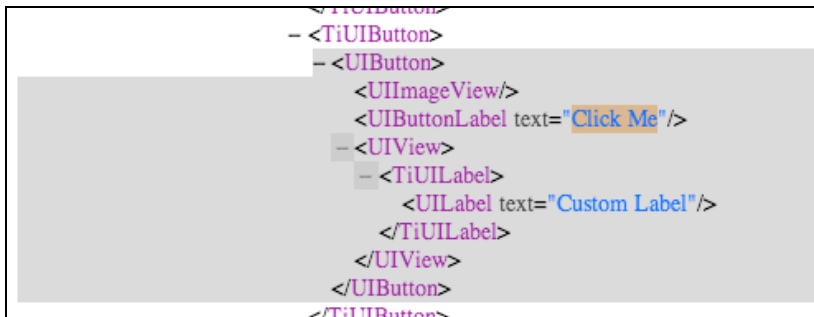
4. Double-click the screenshot to get a closer look.

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

5. Scroll down further, if necessary, and locate `outputXMLHierarchy`.

6. Expand the Output section.

7. Use the browser's Search in page function to locate the button text "Click Me."



Note that TouchTest has recorded the `UIButtonLabel` text as equal to "Click Me" but that if you search for the other values (I am Enabled or I am Disabled) they are not found.

8. 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	
	10	1074	Info	App Action: send	Performing App Action. Band: "Band 1" Track: "Track 1" Clip: "Demo Clip for KitchenSink" Group: "Group1" Target: "SOASTA Demo iPad KitchenSink"
	11	1075	Verbose	Transport: appbeg	Performing App Action "App Action1" for Destination "SOASTA Demo iPad KitchenSink", operation "tap". Band: "Band 1" Track: "Track 1" Clip: "Demo Clip for KitchenSink" Group: "Group1" Target: "SOASTA Demo iPad KitchenSink" ▶Details:
	12	6267	Verbose	Transport: append	App Action "App Action1" completed. Band: "Band 1" Track: "Track 1" Clip: "Demo Clip for KitchenSink" Group: "Group1" Target: "SOASTA Demo iPad KitchenSink" ▶Details:
	13	6267	Info	Validation: vstart	Starting validation "verifyElementText". Band: "Band 1" Track: "Track 1" Clip: "Demo Clip for KitchenSink" Group: "Group1" Target: "SOASTA Demo iPad KitchenSink"
	14	6270	Verbose	Validation: vcpass	Validation of response body passed. Band: "Band 1" Track: "Track 1" Clip: "Demo Clip for KitchenSink" Group: "Group1" Target: "SOASTA Demo iPad KitchenSink" ▶Details:
	15	6270	Info	Validation: vpass	Validation verifyElementText passed. Band: "Band 1" Track: "Track 1" Clip: "Demo Clip for KitchenSink" Group: "Group1" Target: "SOASTA Demo iPad KitchenSink"
	16	6270	Info	App Action: sent	App Action completed. Band: "Band 1" Track: "Track 1" Clip: "Demo Clip for KitchenSink" Group: "Group1" Target: "SOASTA Demo iPad KitchenSink"

9. Return to the Navigation tree and select App Action2.

10. Note that the label validation on App Action2 passes using these settings discussed above.

Output

Validations

Name:

Command:

Locator:

Expected:

Observed:

Adding TouchTest™ IDs to an iOS App in Titanium

A TouchTestID is a reliable locator that is added to project source code by developers and is then used by testers as a convenient hook.

This technique is useful in testing situations, most especially where locators are recorded successfully by TouchTest, but which in reality have an unreliable, or variable value.

For example, in an app that has a navigation bar on top the element on that bar might record a different locator `classname=UILabel[x]` where each time `x` has a different integer value (i.e., 24, 27, 50). In such a case, normal recording techniques will prove insufficient.

This tutorial presents a basic scenario for using TouchTestIDs with Titanium Studio projects in iOS.

In the following section, we will add a TouchTestID to the Titanium project that will change the locators recorded earlier in our example clip. Once we have done so, we will then reinstall KitchenSink to the device once again using the Add to iOS Device command, and then re-record the test clip using the same scenario as before.

Refer to the [SOASTA TouchTest Developer's Guide](#), Adding TouchTestIDs to an iOS app in Xcode section for a brief Xcode example.

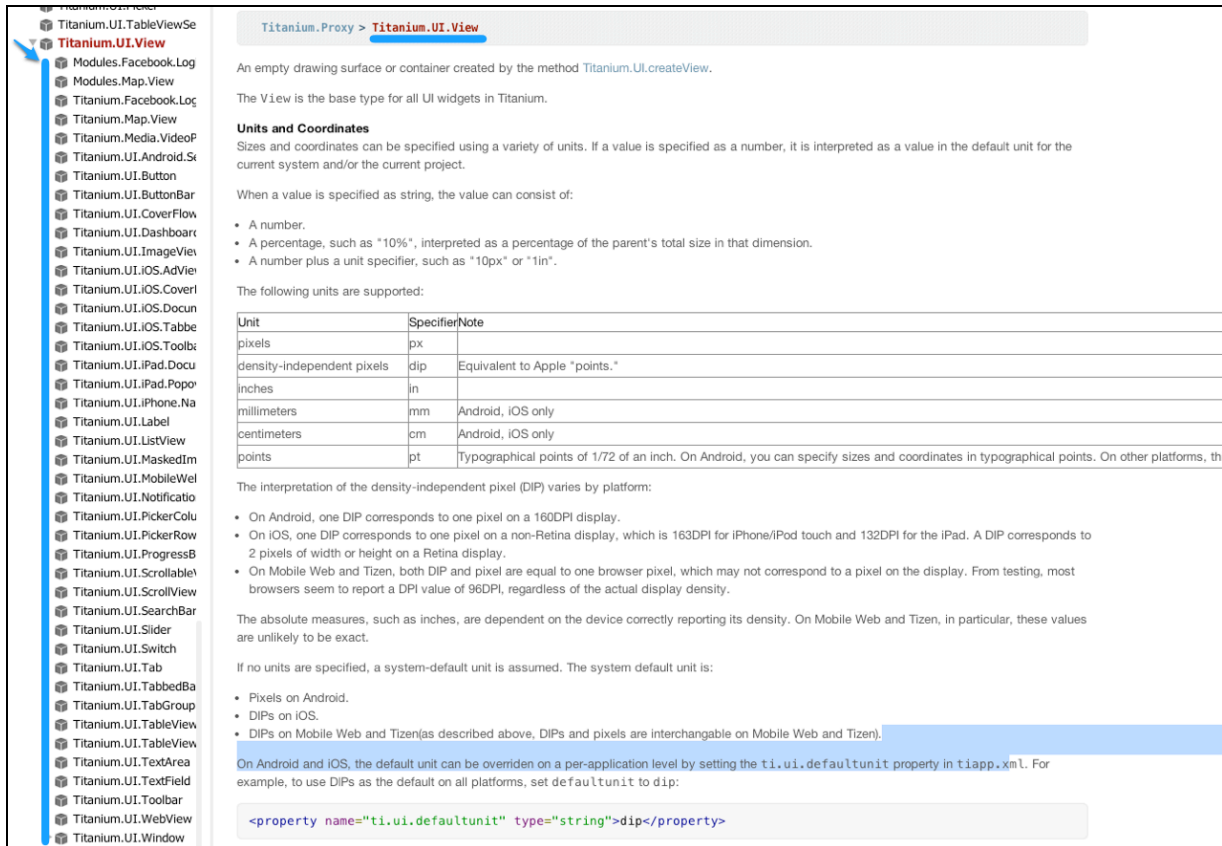
Refer to the [SOASTA TouchTest Advanced Tutorial](#) for a longer example implementation of TouchTestIDs in an iOS environment, as well as additional examples using accessors such as outputs, validations, and waits to enhance mobile testing. Additionally, an App Action References is included in this document.

The TouchTestID scenario included in the Advanced Tutorial also appears in the , [Zero To Test 3 – TouchTestIDs](#) instructional video.

Supported Titanium Components

TouchTestIDs are supported on ALL the Titanium UI components that inherit from `Titanium.UI.View`.

Components that do not inherit from `Titanium.UI.View` are not currently supported.



Titanium.Proxy > **Titanium.UI.View**

An empty drawing surface or container created by the method `Titanium.UI.createView`.

The `View` is the base type for all UI widgets in Titanium.

Units and Coordinates

Sizes and coordinates can be specified using a variety of units. If a value is specified as a number, it is interpreted as a value in the default unit for the current system and/or the current project.

When a value is specified as string, the value can consist of:

- A number.
- A percentage, such as "10%", interpreted as a percentage of the parent's total size in that dimension.
- A number plus a unit specifier, such as "10px" or "1in".

The following units are supported:

Unit	Specifier	Note
pixels	px	
density-independent pixels	dip	Equivalent to Apple "points."
inches	in	
millimeters	mm	Android, iOS only
centimeters	cm	Android, iOS only
points	pt	Typographical points of 1/72 of an inch. On Android, you can specify sizes and coordinates in typographical points. On other platforms, the interpretation of the density-independent pixel (DIP) varies by platform:

- On Android, one DIP corresponds to one pixel on a 160DPI display.
- On iOS, one DIP corresponds to one pixel on a non-Retina display, which is 163DPI for iPhone/iPod touch and 132DPI for the iPad. A DIP corresponds to 2 pixels of width or height on a Retina display.
- On Mobile Web and Tizen, both DIP and pixel are equal to one browser pixel, which may not correspond to a pixel on the display. From testing, most browsers seem to report a DPI value of 96DPI, regardless of the actual display density.

The absolute measures, such as inches, are dependent on the device correctly reporting its density. On Mobile Web and Tizen, in particular, these values are unlikely to be exact.

If no units are specified, a system-default unit is assumed. The system default unit is:

- Pixels on Android.
- DIPs on iOS.
- DIPs on Mobile Web and Tizen (as described above. DIPs and pixels are interchangeable on Mobile Web and Tizen).

On Android and iOS, the default unit can be overridden on a per-application level by setting the `ti.ui.defaultunit` property in `tiapp.xml`. For example, to use DIPs as the default on all platforms, set `defaultunit` to `dip`:

```
<property name="ti.ui.defaultunit" type="string">dip</property>
```

Refer to the complete list of components that inherit from `Titanium.UI.View` [here](#):

The `touchTestId` is added to the project using the JSON-style parameter from the factory methods: `Titanium.UI.createXxx(...)`.

For example,

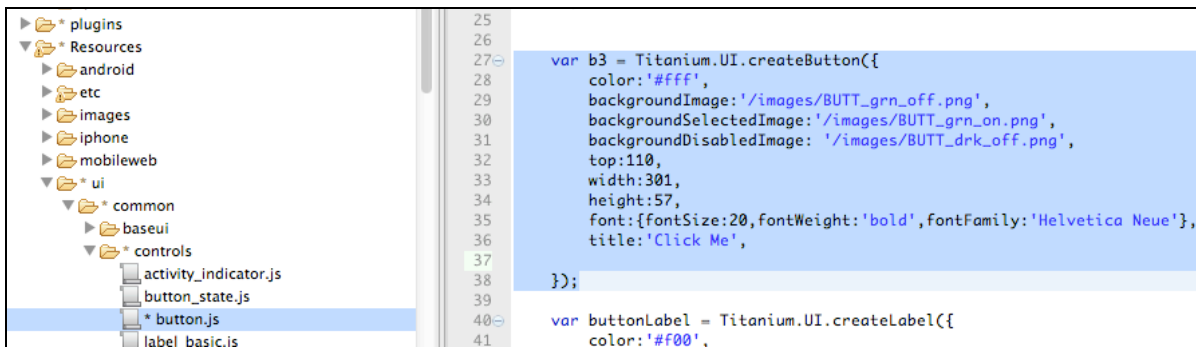
```
var button = Titanium.UI.createButton({
  title: 'Hello',
  top: 10,
  width: 100,
  height: 50,
  touchTestId: 'MyButtonId'
});
```

See [Titanium.UI.createButton](#).

Identifying the Component to Modify

Use the following steps to determine which component in your project to modify by adding TouchTestIDs.

1. Identify the source file that has the button that is used in AppAction 2, 3, and 4. In this case, that file is `button.js`, which can be found in the project at `master/Resources/ui/common/controls/button.js`.
2. Open the `button.js` file.
3. Locate the "Click Me" string using Find.



4. Next, add the following new line after the line `title: 'Click Me'`,
- ```
touchtestid: 'TouchTestButton'
```
5. Your modification should look like this:

```
var b3 = Titanium.UI.createButton({
 color: '#fff',
 backgroundImage: '/images/BUTT_grn_off.png',
 backgroundSelectedImage: '/images/BUTT_grn_on.png',
 backgroundDisabledImage: '/images/BUTT_drk_off.png',
 top: 110,
 width: 301,
 height: 57,
 font: {fontSize: 20, fontWeight: 'bold', fontFamily: 'Helvetica Neue'},
 title: 'Click Me',
 touchtestid: 'TouchTestButton'
});
```

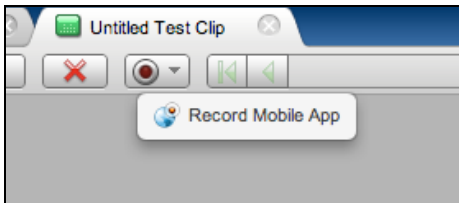
6. Save `button.js`, refresh the Titanium project, and then right-click the master node and choose Add to iOS Device, ensuring that you follow through all the steps including the iTunes sync.

## Re-Record the KitchenSink Clip with TouchTestIDs

Now that we've successfully added TouchTestIDs to the KitchenSink project, and have that up and running (on either a mobile device or a simulator), let's return to the Clip Editor and record a new version of the Fool's Mate chess game.

**TIP:** Don't miss the TouchTestID video tutorial, [Zero To Test 3 – TouchTestIDs](#) video.

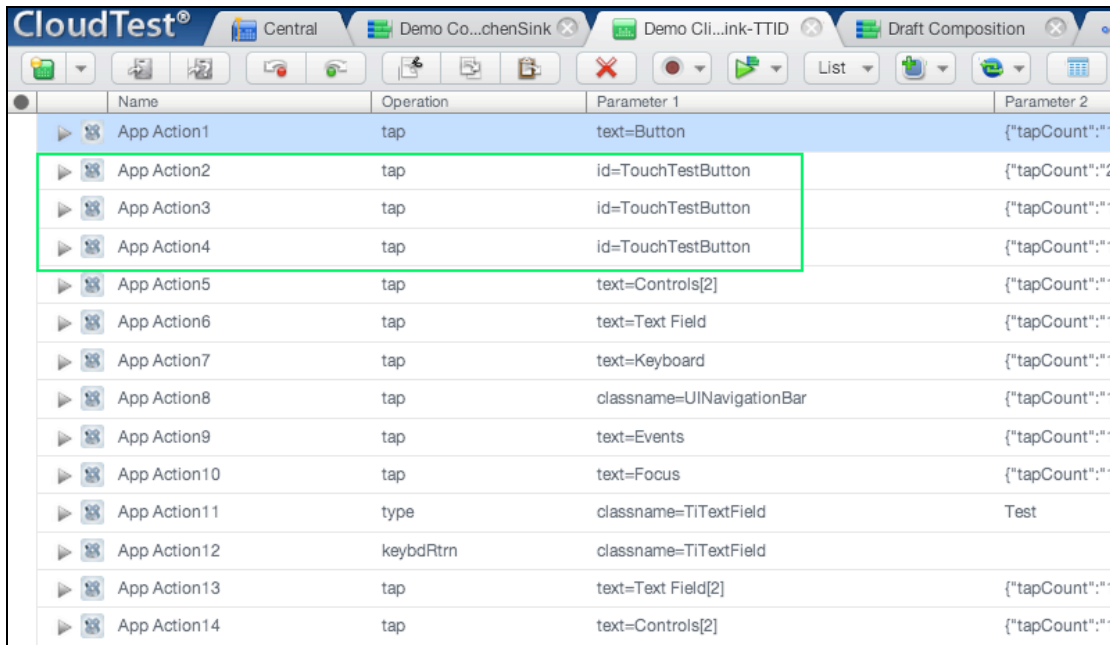
1. Click the Record button and select Record Mobile App.



2. The Choose a Device Agent and Mobile App wizard appears. Select the device agent and app just as you did while recording the first time.
3. Perform the same 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 Button
  - On the Button screen, tap the button that says, Click Me, and then tap that same button three more times (pause slightly on the first tap until the text "I am Enabled" appears).
  - Return to Controls (top left)
  - Tap Text Field
  - Tap Events
  - Tap Focus
  - When the text field has focus, enter "Test" and tap the Return key
  - Tap Test Field (top left) and Controls (also top left) to return to the original view.

4. Once the relevant interactions have been recorded, click the Record button again to stop the recording (the Record button will appear as above after recording is stopped).

For each app action you perform, the Clip Editor adds an app action to the clip. Note that in the Clip Editor List view (shown below) that App Action 2 through 4 now have `id=TouchTestButton` as their value.



| Name         | Operation | Parameter 1                      | Parameter 2      |
|--------------|-----------|----------------------------------|------------------|
| App Action1  | tap       | text=Button                      | {"tapCount": ""} |
| App Action2  | tap       | id=TouchTestButton               | {"tapCount": ""} |
| App Action3  | tap       | id=TouchTestButton               | {"tapCount": ""} |
| App Action4  | tap       | id=TouchTestButton               | {"tapCount": ""} |
| App Action5  | tap       | text=Controls[2]                 | {"tapCount": ""} |
| App Action6  | tap       | text=Text Field                  | {"tapCount": ""} |
| App Action7  | tap       | text=Keyboard                    | {"tapCount": ""} |
| App Action8  | tap       | classname=UINavigationController | {"tapCount": ""} |
| App Action9  | tap       | text=Events                      | {"tapCount": ""} |
| App Action10 | tap       | text=Focus                       | {"tapCount": ""} |
| App Action11 | type      | classname=UITextField            | Test             |
| App Action12 | keybdRtrn | classname=UITextField            |                  |
| App Action13 | tap       | text=Text Field[2]               | {"tapCount": ""} |
| App Action14 | tap       | text=Controls[2]                 | {"tapCount": ""} |

5. Now that we've added more reliable locators using TouchTestId, let's revisit three of the outputs and validations that we used before. Do all of the following:
  - a. Add the Output, `outputXMLHierarchy`, to App Action1
  - b. Add the validation, `verifyElementText`, to App Action2 using `id=TouchTestButton` and I am Enabled as the Exact Match
  - c. Add the Output, `captureScreenshot`, to App Action2

6. Next, let's add some additional text validations:
  - a. Add the validation, `verifyElementText`, to App Action3 using `id=TouchTestButton` as the locator and "I am red" as the exact match value
  - b. Add the validation, `verifyElementText`. to App Action4 using the locator `id=TouchTestButton` and "White text" as the exact match value
7. Save the test clip.
8. Using the Play in Test Composition command on the Clip Editor toolbar, run this clip in a new Draft Composition.
9. In our example test composition, all of this succeeded right off the bat.
  - In Result Details, we selected App Action1 and used the browser's Search in page function to locate the button text "Click Me" in the Result Details, Output section and found that the XML Hierarchy showed the addition of our TouchTestId.

```

- <UIButton>
 - <UIButton touchTestId="TouchTestButton">
 <UIImageView/>
 <UIButtonLabel text="Click Me"/>
 - <UIView>
 - <UILabel>
 <UILabel text="Custom Label"/>
 </UILabel>
 </UIView>
</UIButton>

```

- We selected App Action2 and used the Result Details, Summary tab to inspect the details for the text validation "I am Enabled"

**Output**

Name:

Command:

Locator:

Expected:

Observed:

- We selected App Action3 and used the Result Details, Summary tab to inspect the details for the text validation "I am red"

| Output    |                    |
|-----------|--------------------|
| Name:     | verifyElementText  |
| Command:  | output-elementText |
| Locator:  | id=TouchTestButton |
| Expected: | I am red           |
| Observed: | I am red           |

- We selected App Action4 and used the Result Details, Summary tab to inspect the details for the text validation "White text"

| Output    |                    |
|-----------|--------------------|
| Name:     | verifyElementText  |
| Command:  | output-elementText |
| Locator:  | id=TouchTestButton |
| Expected: | White text         |
| Observed: | White text         |

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