# TouchTest™ Android Tutorial

SOASTA TouchTest™ Android Tutorial

## Table of Contents

# Why Mobile App Testing?

TouchTest® Mobile, featuring 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.

TouchTest controls mobile devices through a lightweight web service called TouchTest™ Agent. Devices can be dedicated to testing in the lab, used as part of an external test, or crowd-sourced as part of a high volume, globally distributed test.

TouchTest support is provided for recording user actions within any Android SDK version 2.3.3 or greater.

## *TouchTest® Basics*

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

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, and the dashboard where results are analyzed.

The **Clip** is the basic building block of a test as built in the **Clip Editor** and has a **Target** such as a web site, or in the case of TouchTest™, a mobile app. 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.)—and parameterized as required.

TouchTest clips are recorded directly from the mobile app and added to the Clip as you perform them on the mobile device.

## What Does Touch Test Record?

TouchTest™ records the details of actual gestures and events that invoked 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 TouchTest®, 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™ for Android

TouchTest™ for Android brings the innovative functional test automation capabilities of TouchTest™ to Android mobile apps on your Android device or emulator.

When combined with Android, TouchTest™ technology delivers precision functional testing that increases the stability of automated tests across releases.

TouchTest® can launch Android mobile apps that are under test using the Android mobile app, TouchTest™ Agent. Devices can be dedicated to testing in the lab, used as part of an external test, or crowd-sourced as part of a high volume, globally distributed test.

Using SOASTA TouchTest™ Driver, which is compiled into the app under test, support is provided for recording, playback and validations of user actions within any Android device. There is no need to jailbreak the Android device and the device can be untethered.

## TouchTest™ Hybrid

TouchTest™ Hybrid extends TouchTest mobile app support to include the recording and playback of Android hybrid apps. Hybrid mobile apps in Android are those mobile apps that render a web page as a portion of its content.

TouchTest Hybrid extends the TouchTest native mobile app support to every Android mobile app on your device.

When you record a TouchTest clip in a hybrid mobile app, the new App Action type, *webClick* appears in the test clip.

## Getting Started

The tutorial is written for TouchTest or TouchTest Lite users who are either Android developers and/or testers, and who want to learn to use TouchTest and TouchTest Hybrid techniques to test their own mobile apps. The preferred method for deploying TouchTest is Gradle because it simplifies integrating TouchTest.

Steps are provided for using the TouchTest Gradle Plugin (with Android Studio). Additionally, for Eclipse users, the MakeAppTouchTestable utility is provided. Eclipse and MATT instructions are found in **Appendix III** at the end of this tutorial.

To use TouchTest techniques **for test creation** on a ready-to-test, proceed with Recording a TouchTest Scenario below.

- For test creation examples, we'll first record a basic TouchTest test clip using each mobile app, then add that test clip to a test composition and play it to see initial results; after which we'll refine each test clip by adding outputs and validations, and then finally, we'll analyze the results of a fully articulated test composition.

If you are a tester with a device where the mobile app is installed, but you still need to register a TouchTest Agent on it, begin with the Preparing an Android Device section.

## Prerequisites

TouchTest™ recording is performed by any TouchTest user by accessing or deploying the following TouchTest® components on the desktop and on a Android device running 2.3.3 or later:

- The TouchTest® (or TouchTest® Lite instance) where the user has rights and will login to start recording.
- The TouchTest™ Agent; a per mobile device agent pointed at the same TouchTest®/Lite instance installed on the Android device. The TouchTest Agent app will be installed from the TouchTest, Resources page onto the given Android device. Download and registration instructions are included in the following sections.

**Note:** If you're using the Appcelerator Titanium Studio—refer to the TouchTest™ for Appcelerator Android Tutorial instead of this tutorial.

- The mobile app to test. For this tutorial, two sample apps are presented, but you can also use your Android native or hybrid mobile app. The deployed mobile app and its corresponding entry in TouchTest's Central > Mobile Apps list must share the same launch URL in order for testing to succeed as discussed in the MakeAppTouchTestable sections below.

### *Configuring the Android Device*

The Android Device must first be in Developer mode. Note that the Developer Options section is not shown by default in Android device Settings. A secret handshake is required.

Use the following steps to enable Developer mode on the device:

1. Tap *Settings*
2. Tap *About phone*.

3. In the *About phone* section. Tap the list item labeled *Build Number* seven distinct times.
4. After 3 taps, each tap shows a new toast message encouraging you to keep going. After the last tap, the message "You are now a developer" appears.

Subsequently, your Android device should also have the following Settings:

- In the device settings, tap Developer Options and check the USB Debugging box.
- In the device settings, tap "Security" and then tap to check the Unknown sources box.

Optionally, you can also enable Android's built-in equivalent of TouchTest's (iOS only) Head's Up Display by using the following device steps:

1. On the mobile device or simulator, go to Settings, Developer options.
2. Enable Show touches.
3. Enable Pointer location.

Refer to TouchTest Head's Up Display for ideas about how to use this optional feature using Android's own built-in HUD-like display.

**Note:** TouchTest Agent must be installed on the device. Instructions to install it are included below.

## *Build Prerequisites*

With Gradle, no additional utility downloads or separate build steps are necessary.

**Note:** Periodically, plugin updates will be released, increasing the Gradle Plugin number. The plugin number in your build.gradle file must match the latest Gradle Plugin version.

This is because Gradle downloads the necessary JAR files, builds and the Android app, and creates the corresponding mobile app object in the Central > Mobile Apps list.

**Note:** Eclipse developers can find additional Eclipse-related instructions in the Appendix III section at the end of this tutorial. Gradle automates most of the Eclipse steps that are quite involved.

## Using the TouchTest Gradle Plugin

With the TouchTest Gradle Plugin, Android Studio users have a simple, effective means to quickly get TouchTest integrated into existing Android apps.

### *Revise Your Gradle Build Script*

Make the following modifications to add the TouchTest Gradle Plugin to your project. Because Gradle will automatically download the necessary JAR files, once these changes are made, no separate download or build steps are required, Gradle will do them automatically.

**Note:** Periodically, plugin updates will be released, increasing the Gradle Plugin number. The plugin number in your build.gradle file must match the latest Gradle Plugin version.

In Android Studio, Central Repository, make the following project changes.

1. In the example below, add the build.gradle line(s) that follow after the comment: `//Add the following line`

```
buildscript {
   repositories {
       jcenter()
   }
   dependencies {
       classpath 'com.android.tools.build:gradle:1.2.3'

       //Add the following line
       classpath 'com.soasta.touchtest:touchtest-plugin:1.2'

   }
}

apply plugin: 'com.android.application'

//Add the following line
apply plugin: 'com.soasta.touchtest'
```

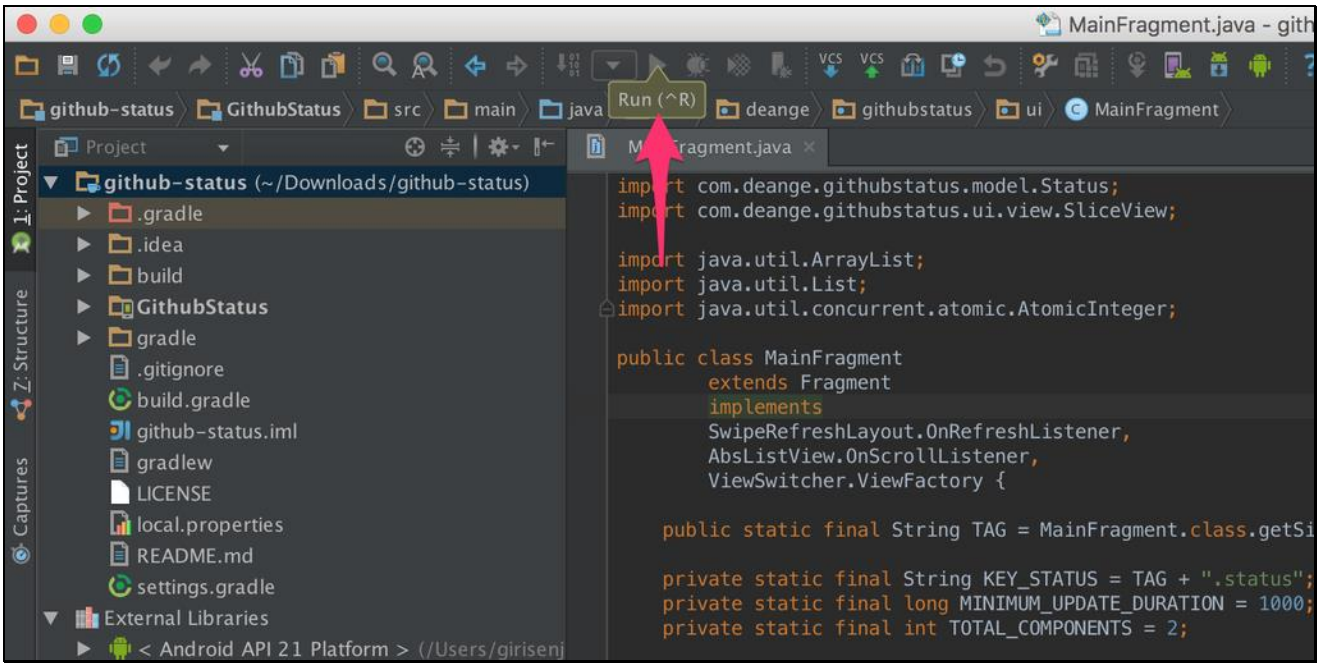
```
repositories {
 jcenter()
}

configurations {
 soasta
}

dependencies {
   compile fileTree(dir: 'libs', include: ['*.jar'])
   compile 'com.android.support:appcompat-v7:20.+'
   soasta  'com.soasta.touchtest:touchtestdriver:1.0'
}
```

2. Create a new file called **touchtest.properties** and then add the following properties:

- `TouchTestURL=http://67.111.67.24:8080/concerto`
  Concerto url of CloudTest server
- `TouchTestUserName=admin`
  CloudTest user name
- `TouchTestPassword=password`
  CloudTest user password
- `TouchTestTenant=SOASTA`
  Tenant in CloudTest environment
- `OverwriteMobileApp=true`
  If a Mobile App already exists, it gets overwritten when true, left alone otherwise
- `AppLaunchURL=touchtest-gradle-example://`
  This launch url will be used when present, a system generated launch url would be used otherwise

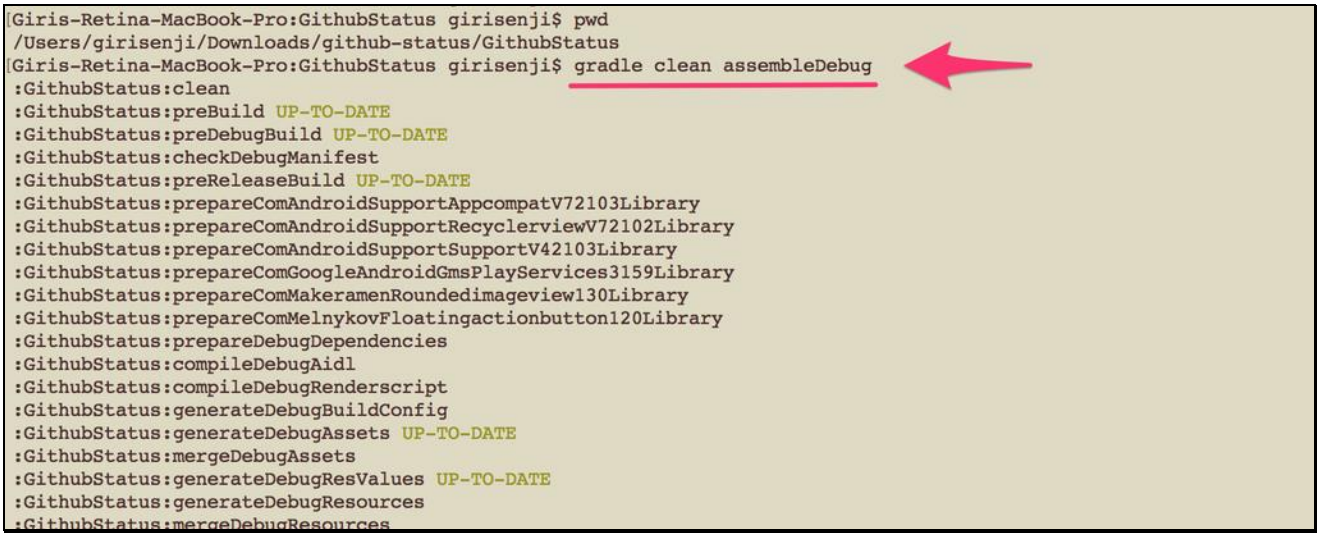
3. Build an Android app by doing one of the following:

- Run a debug build in Android Studio.




Or, in a Terminal window, do:

gradle clean assembleDebug




- gradle clean installDebug

After performing either of the above build methods, the example *application-debug.apk* is Touch Testable.

Next, do the following to install the Android app to a device:

- `gradle clean installDebug`

Your *application-debug.apk* is now testable using TouchTest.

### Empty project's build.gradle

```
apply plugin: 'com.android.application'

android {
    compileSdkVersion 22
    buildToolsVersion "23.0.0 rc3"

    defaultConfig {
        applicationId "com.soasta.myapplication"
        minSdkVersion 19
        targetSdkVersion 22
        versionCode 1
        versionName "1.0"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
        }
    }
}

dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    compile 'com.android.support:appcompat-v7:22.2.1'
}
```

## Typical project's build.gradle

build.gradle — soasta/.../touchtest-plugin | build.gradle — sampleApps/android/touchtest-example | build.

```
buildscript {
    repositories {
        jcenter()
    }
    configurations.all {
      // check for updates every build
      resolutionStrategy.cacheChangingModulesFor 0, 'seconds'
    }
    dependencies {
        classpath 'com.android.tools.build:gradle:1.2.3'
    }
}

apply plugin: 'com.android.application'

repositories {
  jcenter()
}

configurations {
  compile
  runtime
}

android {
    compileSdkVersion 22
    buildToolsVersion "23.0.0 rc3"

    defaultConfig {
        applicationId "com.soasta.gradlesample"
        minSdkVersion 19
        targetSdkVersion 22
        versionCode 1
        versionName "1.0"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
        }
    }
}

dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    compile 'com.android.support:appcompat-v7:22.2.1'
}
```

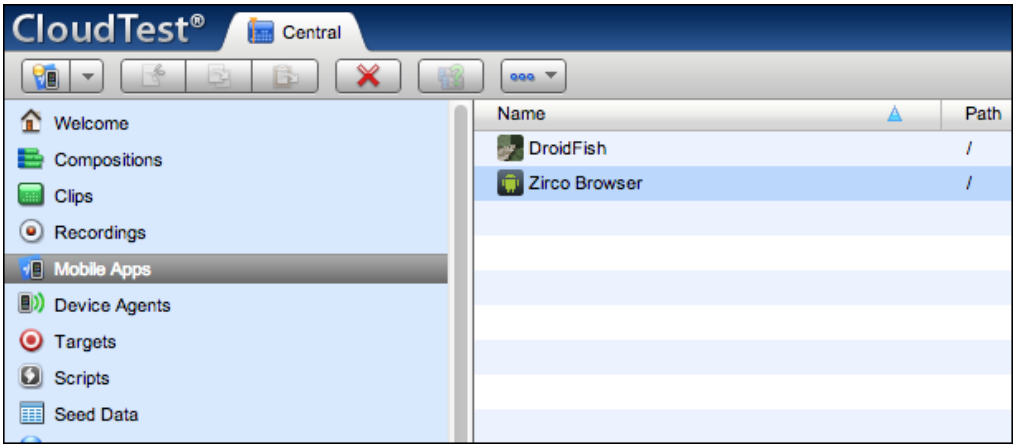
## *Inspecting the Mobile App in TouchTest® (Native or Hybrid)*

Once Gradle (or for Appendix III users, Eclipse) creates the Central, Mobile App object, you can locate an entry for the given mobile app in the Central > Mobile Apps list.

**TIP:** This mobile app object appears by name in the *Choose Device Agent* and *Mobile App* box whenever end-users start a mobile app recording. Selecting which mobile app to launch, on which test devices, is a crucial end-user step. You will need to be familiar with its uses in order to record and perform all of the subsequent test-building steps.

1. Verify that your Mobile App has been added by logging into TouchTest® and looking for its entry in the Central > Mobile Apps list.

- For example, in the screenshot below both DroidFish and Zirco Browser apps appear are listed.

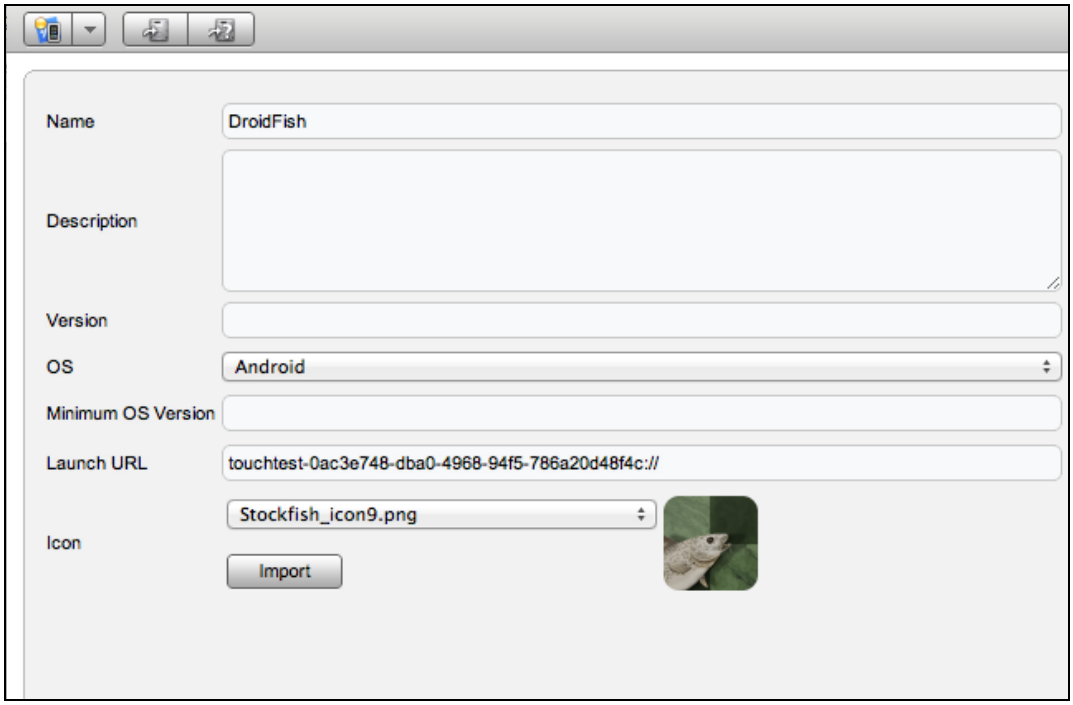


- Double-click the Mobile App to inspect its details.

  The Mobile App detail form appears.

All of the fields shown were populated from the project, with the exception of Supported Device Type and Minimum OS Version.

- The DroidFish mobile app detail is shown below



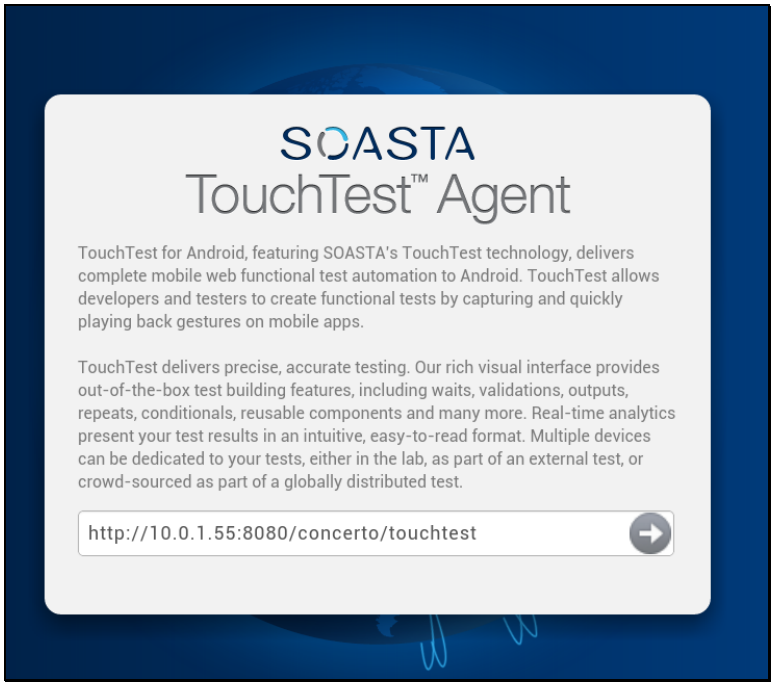
## Install TouchTest Agent & Register Device to Use TouchTest

Once the mobile app is built, installed, and the corresponding mobile app exists in SOASTA Central, use the following steps to install an agent. 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. From your Android device, download the TouchTest Agent Android app from the Resources page. Tap on the downloaded TouchTestAgent.apk file (usually found in the device's notification tray) to start Install.


2. When the installer appears, tap Install to proceed.
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.



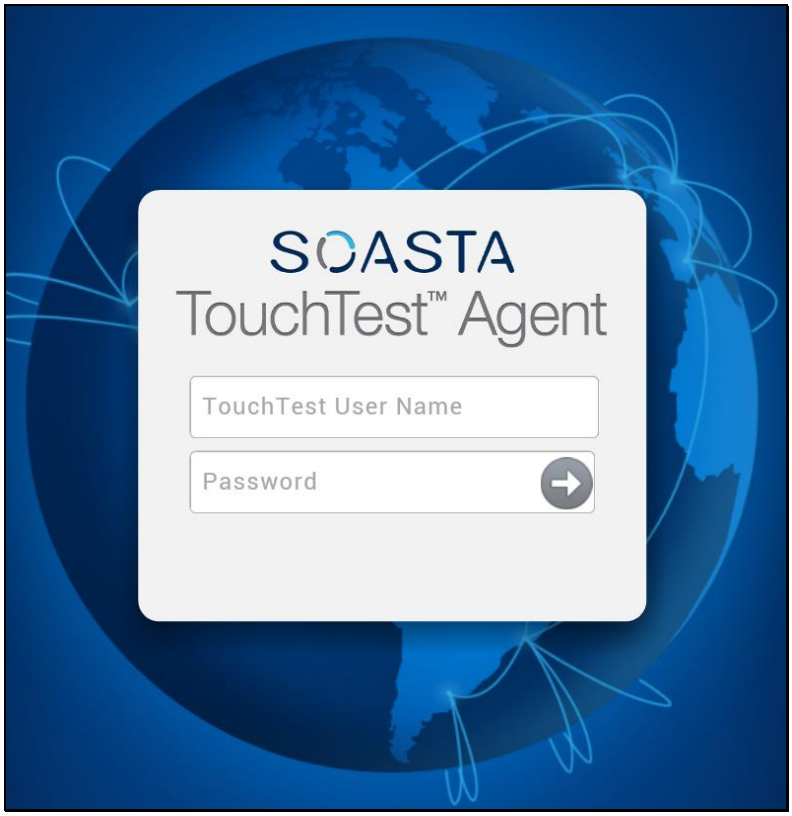

4. Enter the TouchTest URL for TouchTest, which is your own server with `/concerto/touchtest` appended:

   `http://<TouchTest server>/concerto/touchtest`.

   For example, a user with an Apple router running TouchTest Lite might connect to:

   `http://10.0.1.9/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 TouchTest user name and password and tap Login. If the spinner hangs, tap the screen to clear it.
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. The TouchTest Agent page will appear with the status Connected for the first time. On subsequent launches click Login to Connect and Logout to Disconnect.

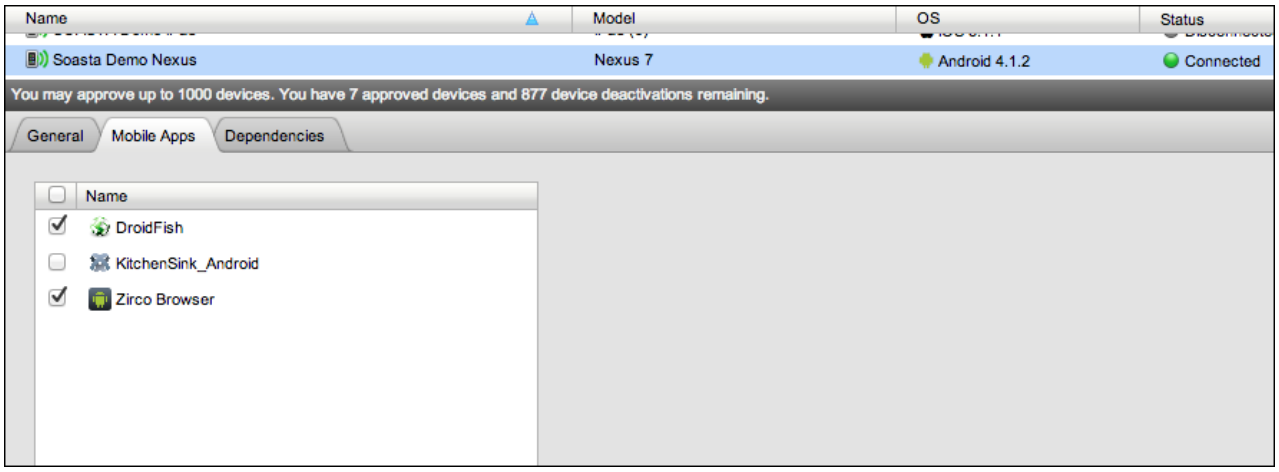

**Notes:** You can also download the TouchTest Agent on the /touchtest URL of the TouchTest Environment.

If these steps have already been performed on a given device, proceed to Recording a TouchTest Scenario below.

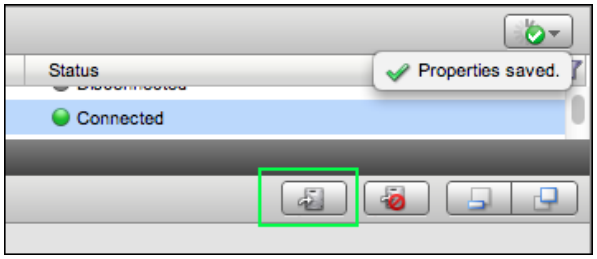
## Associating Mobile Apps with a Specific Device

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. For example, *Soasta Demo Nexus* as shown below.




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. For example, DroidFish and/or Zirco Browser.
4. Click Save on the lower panel toolbar before proceeding.



## Recording a TouchTest Scenario (All Users)

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

- Create a new test clip within TouchTest® for your native or hybrid app.
- Click Record within the Clip Editor and then choose Mobile App Recording, specify the Device Agent, and then the native or hybrid mobile app whose actions you want to record.
- Perform the app actions on the mobile device to capture them in the new test clip.

These steps are described in the remainder of this tutorial.

### *Create a TouchTest™ Clip*

Prepare your device for recording and create a new clip that will be used to perform mobile app recording and serve as the basis for your TouchTest.

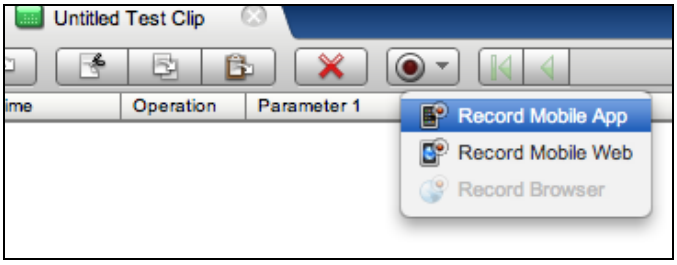
1. Open the TouchTest Agent on your mobile device, and click Login (your previously entered username and password should be auto-populated).


Once successfully logged on, the device Status will be *Connected*.

- Login to TouchTest 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.

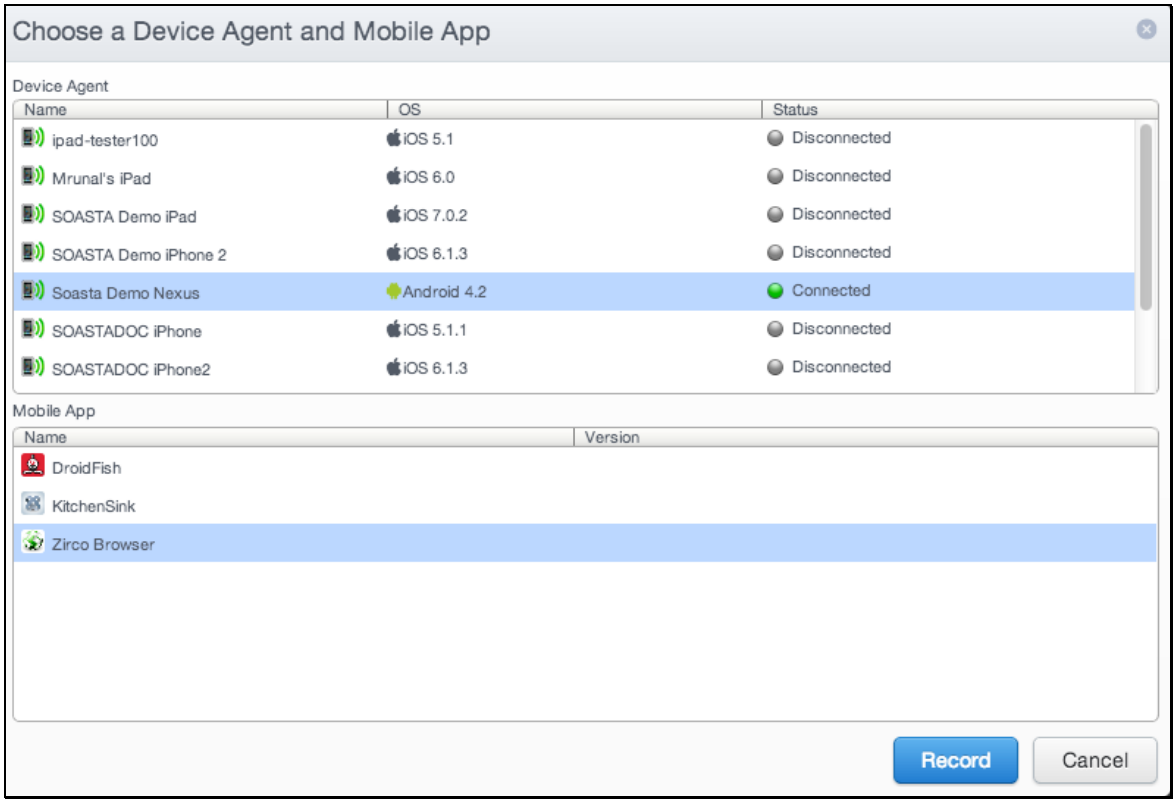
- Once ready, click the Record button and then select Record Mobile App.



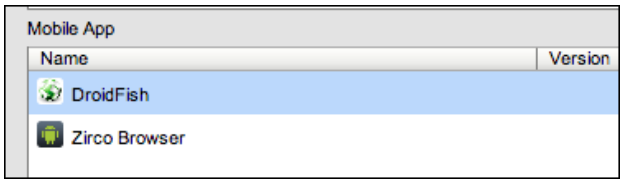
The Choose a Device Agent and Mobile App box appears.

2. Select the TouchTest Agent that you created above and also select the mobile app you'd like to test.

   Note that in the shot below mobile apps are listed for the select device agent (i.e. Soasta Demo Nexus and Zirco Browser).




Or, alternately, *Soasta Demo Nexus* and Droidfish.




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

**Note:** If the Mobile Device Administrator (or TouchTest Lite user) has completed the steps above to associate one or more mobile apps with the device, those apps will appear in the Mobile App list whenever that device is selected. If no mobile app has been defined for the selected device agent, the Mobile App list will be empty (shown below).

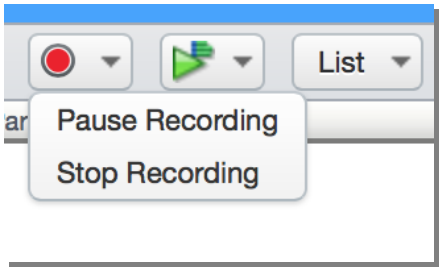
**TIP:** For developers and admins both, ensure that the Make App TouchTestable steps have been applied to your app and that the device agent is associated with a mobile app. You can also click the help link in the box to access documentation on this topic.

### Pause Recording

You have the ability to pause at any moment during a recording to:

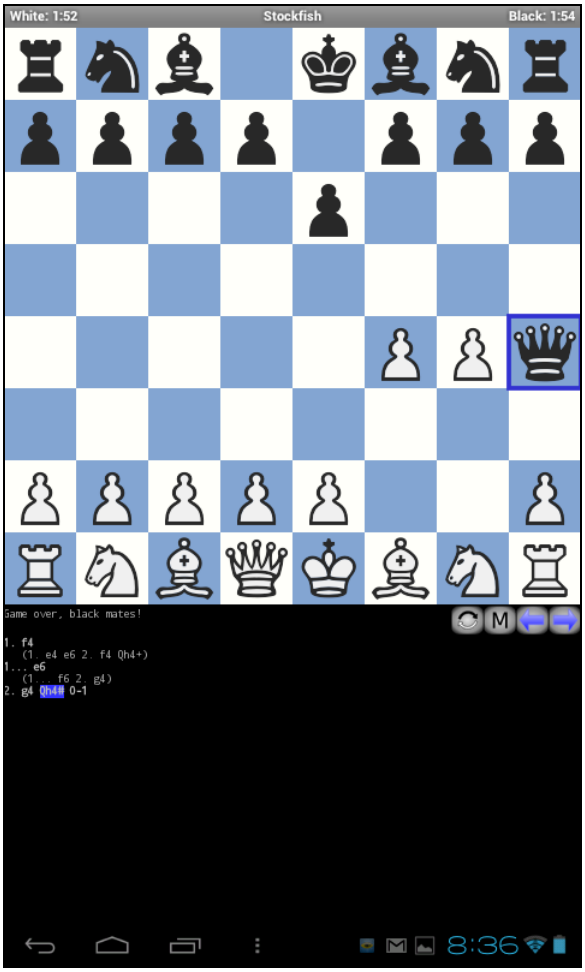
- View your app
- Eliminate unwanted actions
- Save time getting your app to a state where you want it to record actions
- Add more actions to a clip
- Correct locators in the middle of a clip
- Add more waits, outputs, or validations to an existing clip using touch locator
- Record screenshots to use in validations for all cases on a specific page

To pause at any moment, click the **Pause Recording** option.



## Recording a Droidfish Scenario (Native)

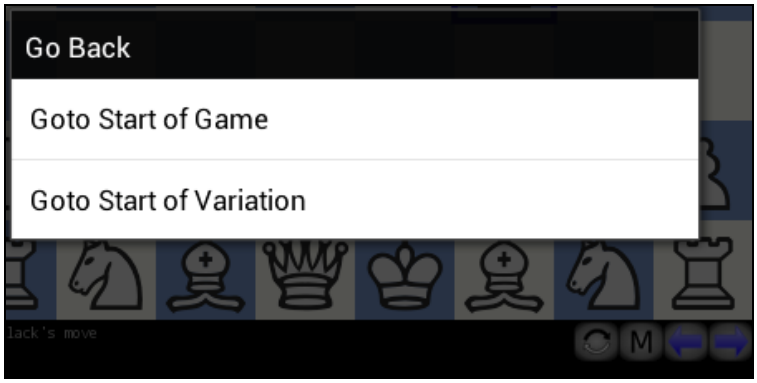
DroidFish is a typical chess application with menu options, the board pieces and settings, a clock for each player, and so forth.



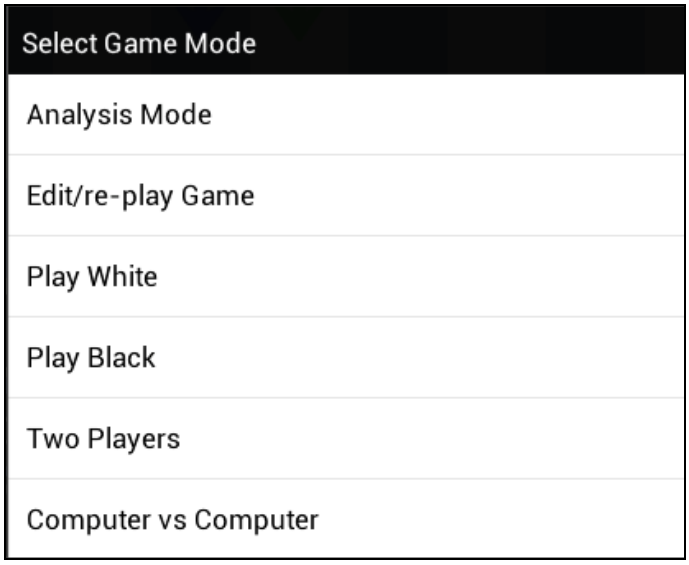

There are a number of verification possibilities:

- Verify that buttons are present
- Verify start of game that 32 pieces are visible
- Verify move list updates
- Verify clock
- Verify check mate position

1. Perform some or all of the following menu actions on your mobile device to get to the beginning of a game.
    - Select and long press Game (3rd button from left at bottom of the chess board), Goto Start of Game.




    - Enter Game Mode, Select Two Players.

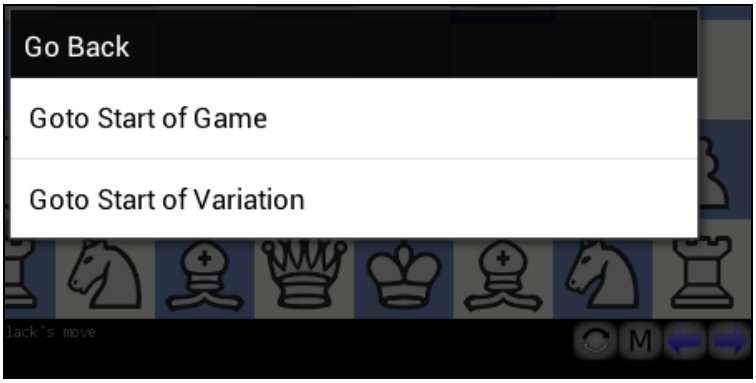


2. Perform the following initial moves known as Fool's Mate:
    - White pawn to F4
    - Black pawn to E6

- White pawn to G4


- Black Queen to H4


5. After reaching the checkmate position shown above, Select Game, Goto Start of Game (e.g. using the 3rd button from the left below the chess board).

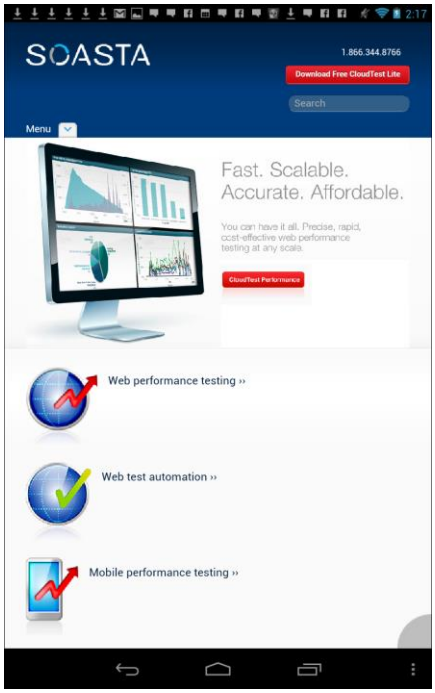


While you perform the mobile app actions, the Clip Editor adds an app action to the clip. The Info Window streams with the latest app action's General tab shown as actions are added.


6. In the desktop browser, click the Record button to stop recording. The recorded clip displays the recorded actions.

## Recording a Zirco Browser Scenario (Hybrid)

Zirco Browser is an Android hybrid app, meaning that it renders a web page as all or part of its functionality. With Zirco Browser running on the device as a result of launching it from the Clip Editor, we will now navigate to the URL for the SOASTA web site: http://www.soasta.com/



1. Perform the planned mobile app user interactions on your mobile device. For each action you perform in the browser, TouchTest Web adds an action to the clip.

   For example, in our demo clip, which is used in the remainder of this hybrid example, we tapped the following sequence:

   a. Long press the Location field in Zirco Browser until the about:text is selected
   b. With the previous text selected, enter `www.soasta.com`
   c. Hit the space bar so that the shortcut menu goes away
   d. Tap the Go Button (the right arrow at the end of the Location field).
   e. On SOASTA home, tap the Menu link.
   f. Tap Solutions and then Mobile Performance testing on the sub-menu.
   g. Click the SOASTA logo to return to home.
   h. On the home page, tap Web performance testing.
   i. Click the SOASTA logo to return to home a second time.

In the screenshot below, the Clip Editor is in Icon view and is also in Record mode while connected to Zirco Browser.

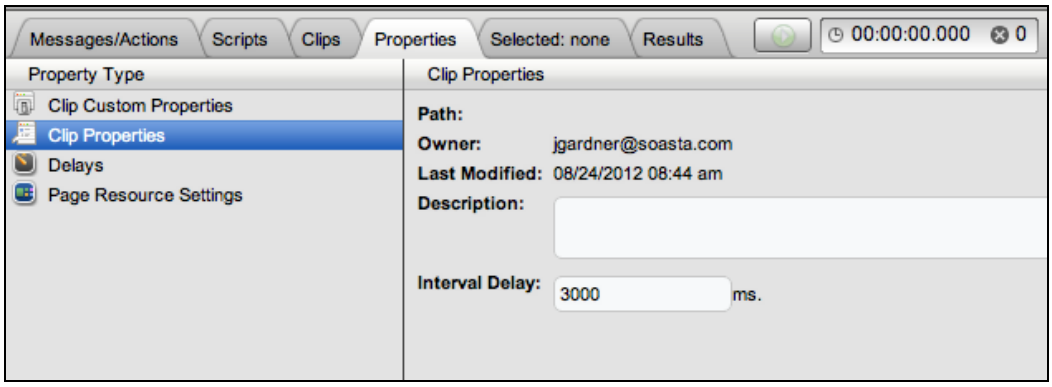

2. Once the relevant interactions have been recorded, click the Record button again to stop the recording.

### Adding an Interval Delay between Each Action (All Users)

In the following optional 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 the editing phase, as well as 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.




Click Save on the Clip Editor toolbar. When the Save Test Clip box appears, accept the default name, which takes the form "`Clip for <Device Name> <Mobile App Name>`.

## Create a Composition (All Users)

With any test clip open in the Clip Editor, you are ready to create and play a simple, new test composition using this test clip. These steps are applicable to both DroidFish and Zirco Browser examples.

1. To create a new composition from your test clip, click the Use in Test Composition drop-down in the upper-right corner of the Clip Editor toolbar, and select Play in Test Composition (as shown below).

**TIP:** Take note of the *Use in Test Composition* commands and their purposes.


The Droidfish clip is shown above; however, these steps apply equally to Zirco Browser.

- **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.

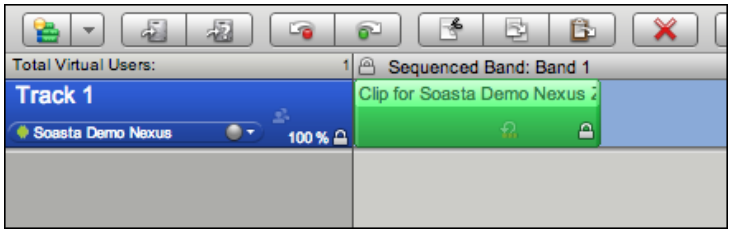
- **Debug in Test Composition**

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.

## *Playing a Composition*

If you used Play in Test Composition as suggested above, then your test clip is added to a new draft test composition, which opens in a new Composition Editor tab and immediately begins to play. If you clicked Open in Test Composition, you have a few more clicks to go.

- Ensure that the TouchTest Agent status is still "Connected" on the mobile device.
- In the Composition Editor, click Play to run the test composition a second time.




The Composition Editor's Status Indicator changes to "Playing," and the mobile app is launched on the specified mobile device(s) precisely as it was recorded.

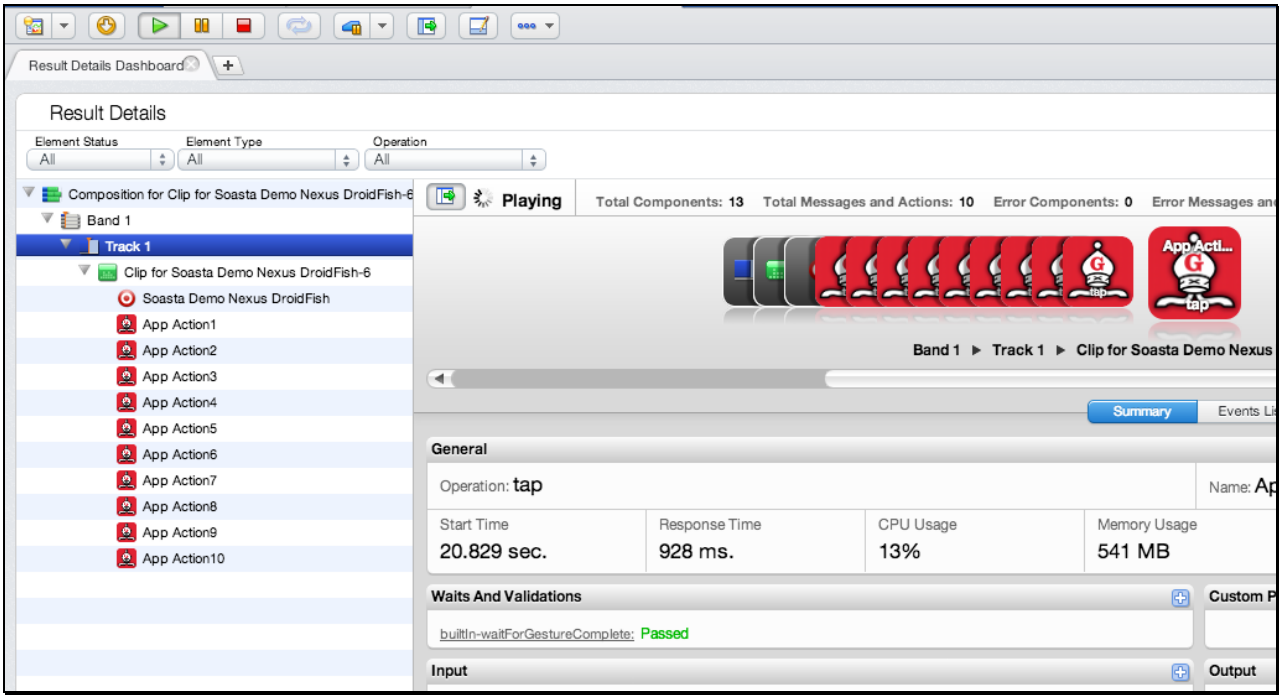

While the test runs, the Composition Editor automatically switches to the Play tab, and by default, the Result Details dashboard displays.

## *Result Details (Droidfish)*

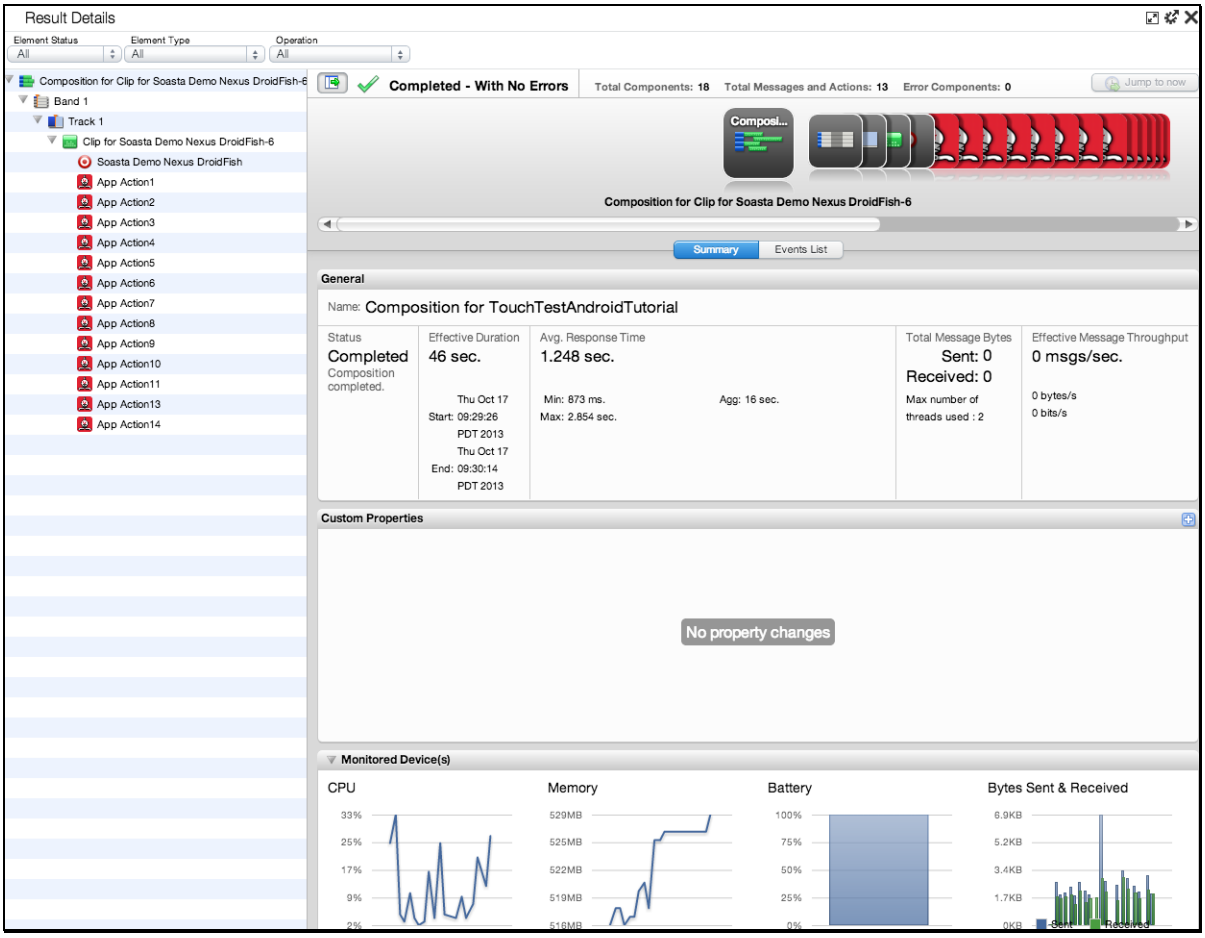
The Result Details dashboard helps to discover the cause of errors in your test, if any.

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

In the show below, a DroidFish test plays in the Composition Editor, Play tab, Result Details dashboard.

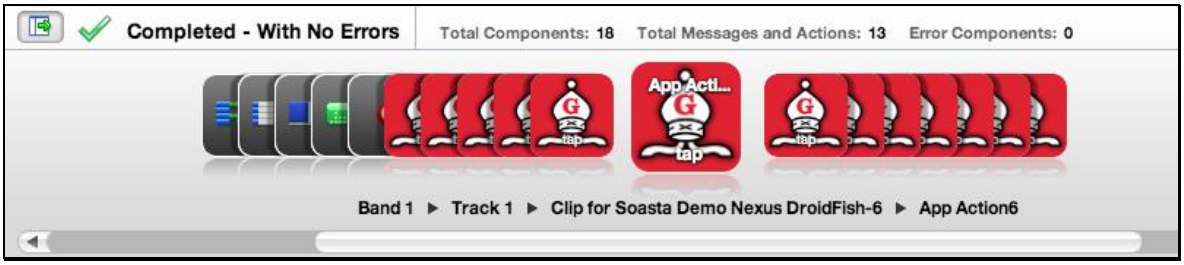


Once play completes, the final results are displayed in the Results tab (also in the Result Details widget). If the test passed on all points, the status "Completed – With No Errors" is clearly posted in the Result Details dashboard.




The mobile app actions performed when the clip was created are played back on the device. Click to expand the nodes in the Navigation Tree on the left as they appear.

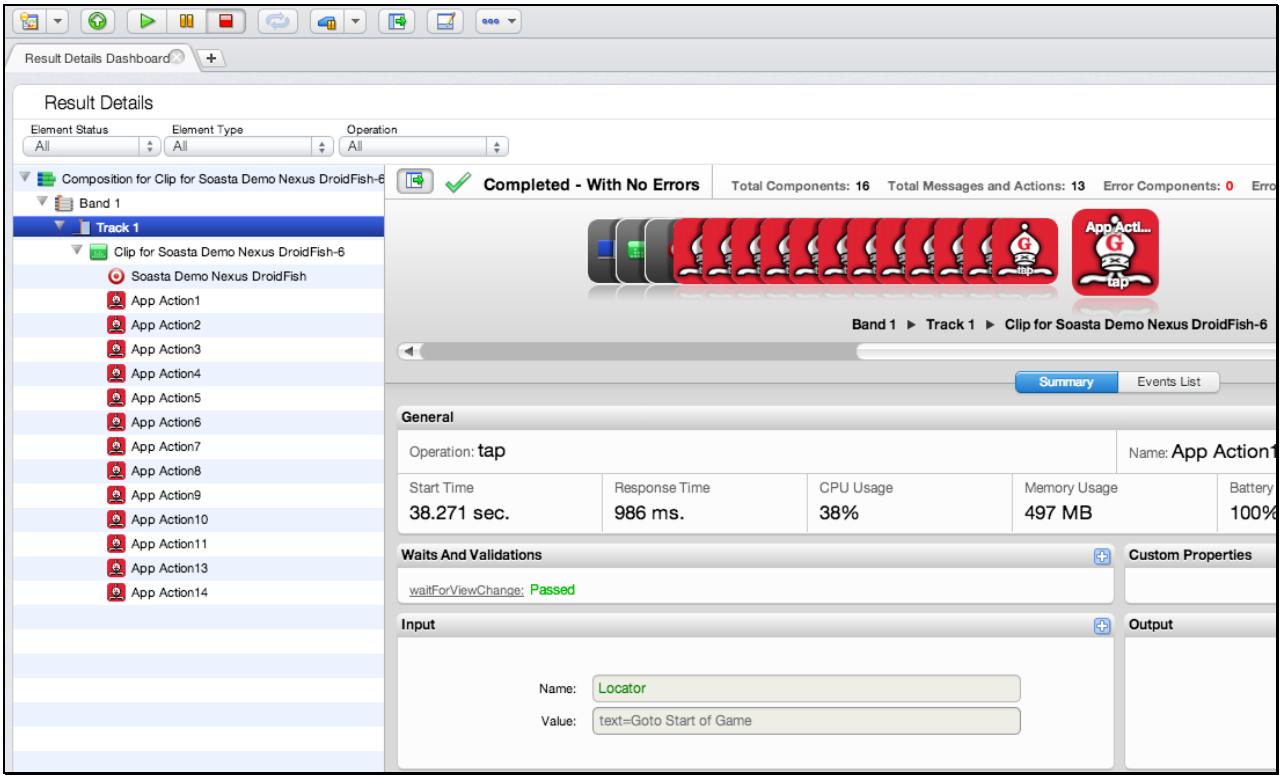
Result Details uses a Cover Flow (top panel to the right) to display the test composition's stream as it occurs.



This stream is also shown in the Navigation Tree (on the left) as elements are executed. As play continues, the focus is set to the last executed element unless user interaction prevents it. 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. To resume auto-focus once interrupted, click Jump to Now in the upper right of the dashboard.

- Click any object in the Cover Flow at the top to center it and display its details and play statistics in the panes below.



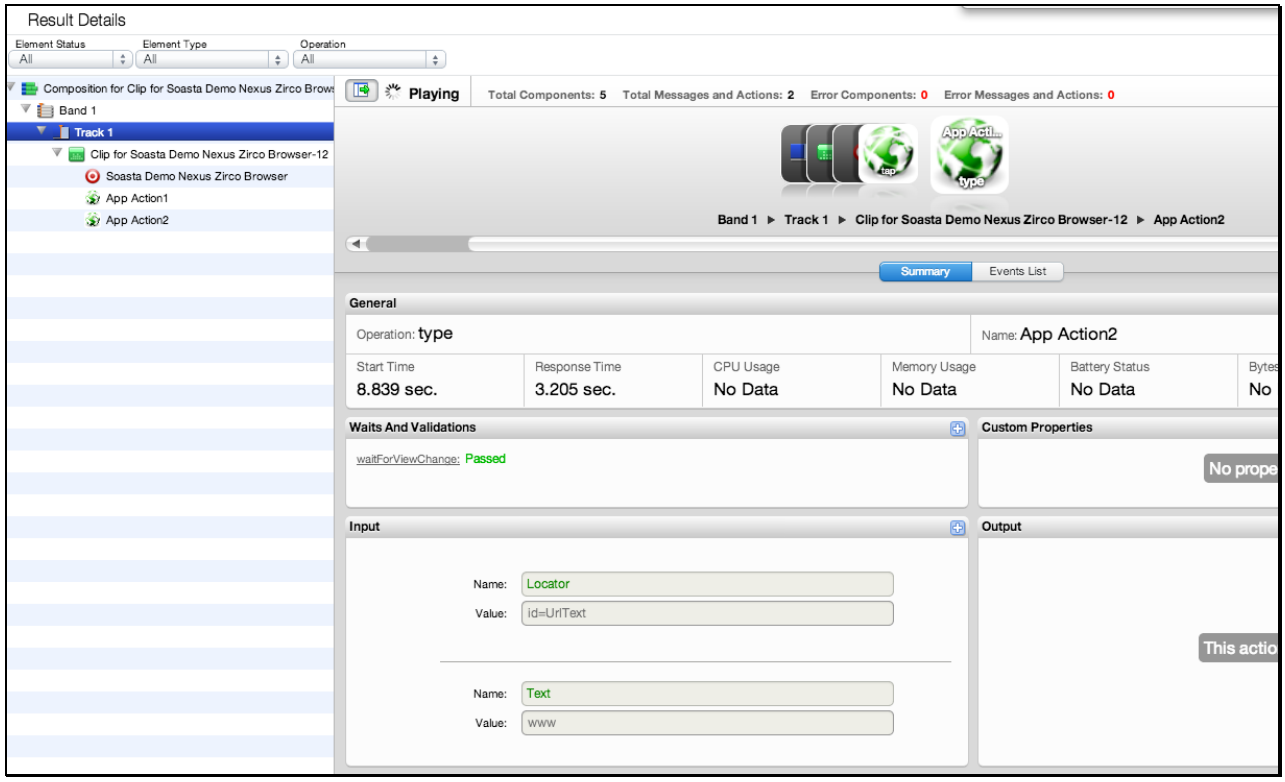

- Use the scrollbar to browse the flow. Select any item to show its low-level details.

## *Result Details (Zirco Browser)*

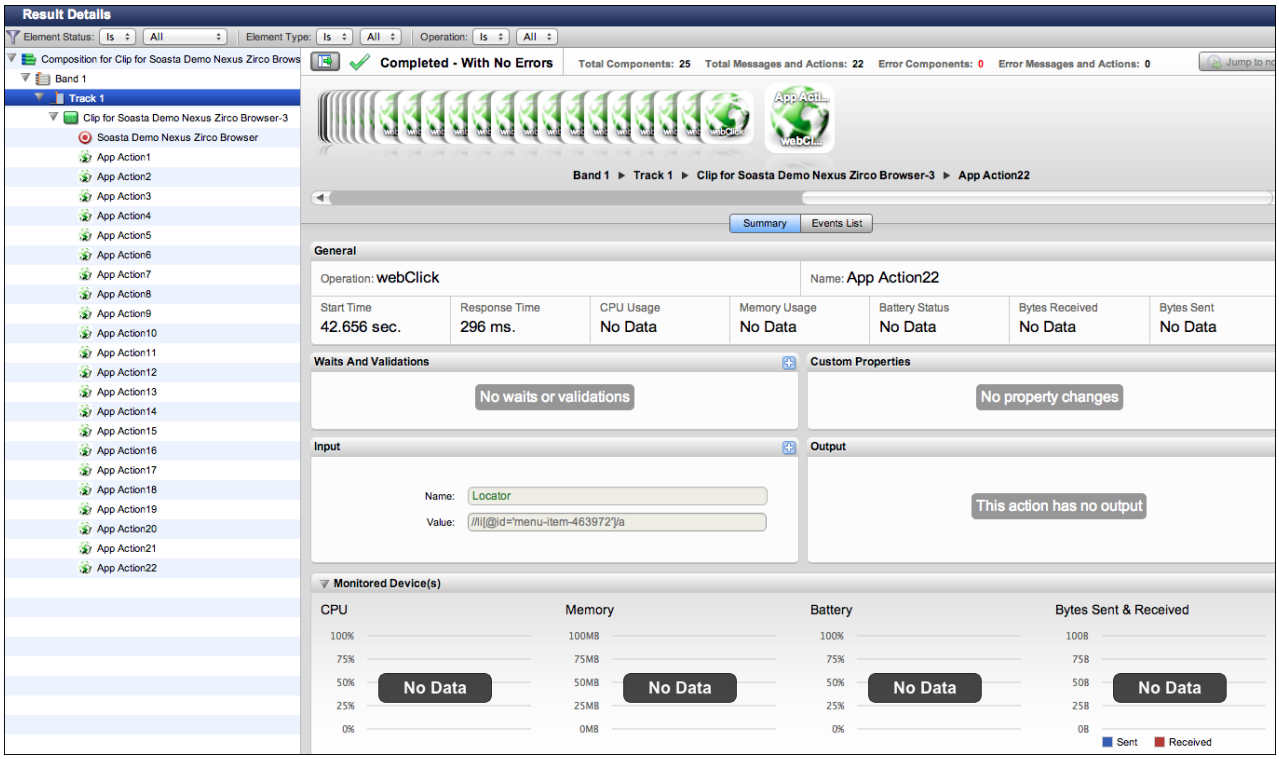
The Result Details dashboard helps to discover the cause of errors in your test, if any.

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

In the shot below, a Zirco Browser test plays in the Composition Editor, Play tab, Result Details dashboard.

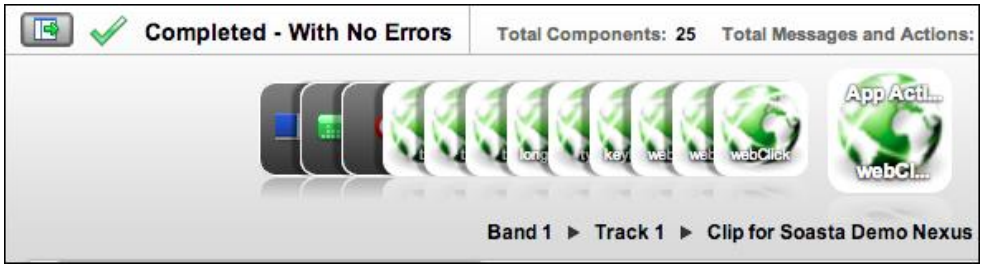


Once play completes, the final results are displayed in the Results tab (also in the Result Details widget). If the test passed on all points, the status "Completed – With No Errors" is clearly posted in the Result Details dashboard.




The mobile app actions performed when the clip was created are played back on the device. Click to expand the nodes in the Navigation Tree on the left as they appear.

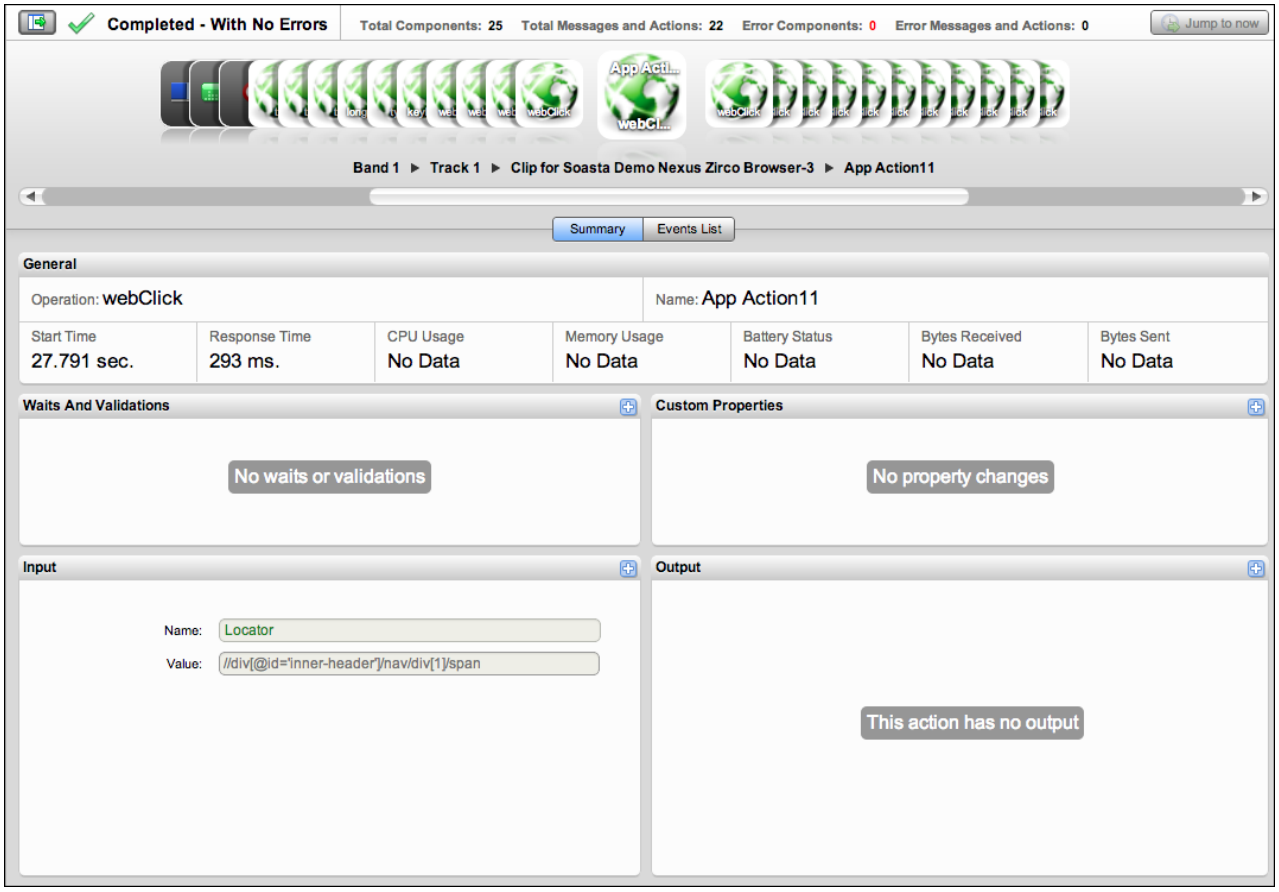
Result Details uses a Cover Flow (top panel to the right) to display the test composition's stream as it occurs.




This stream is also shown in the Navigation Tree (on the left) as elements are executed. As play continues, the focus is set to the last executed element unless user interaction prevents it. 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. To resume auto-focus once interrupted, click Jump to Now in the upper right of the dashboard.

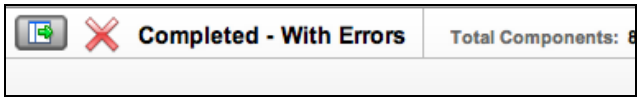
- Click any object in the Cover Flow at the top to center it and display its details and play statistics in the panes below. For example, in the shot below, App Action11 is selected in the Cover Flow and its details are in display in the lower panels.




- Use the scrollbar to browse the flow. Select any item to show its low-level details.

## *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.

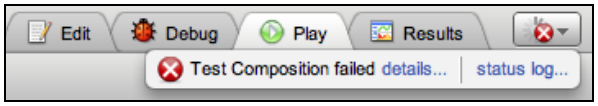



The way we approach them is first to identify where they occurred and then to analyze what occurred.

### Network or Communication Errors

Initial errors in a simple test like the one above are most often only simple network or configuration errors having to do with test staging.

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.

Composition-wide errors such as these are clearly indicated in the General section in the initial view of Result Details. They frequently are related the state of the Device Agent (e.g. if the device agent is not connected when you click Play).

In some cases, the TouchTest Agent may have been started but is no longer responding (or the device auto-lock may have been invoked). In such cases Logout and re-login, or wake up the device if it is in auto-lock mode.

### App Action and Other Errors (All Users)

TouchTest 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 to 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 Result Details dashboard easily distinguishes failures on specific app actions you recorded.

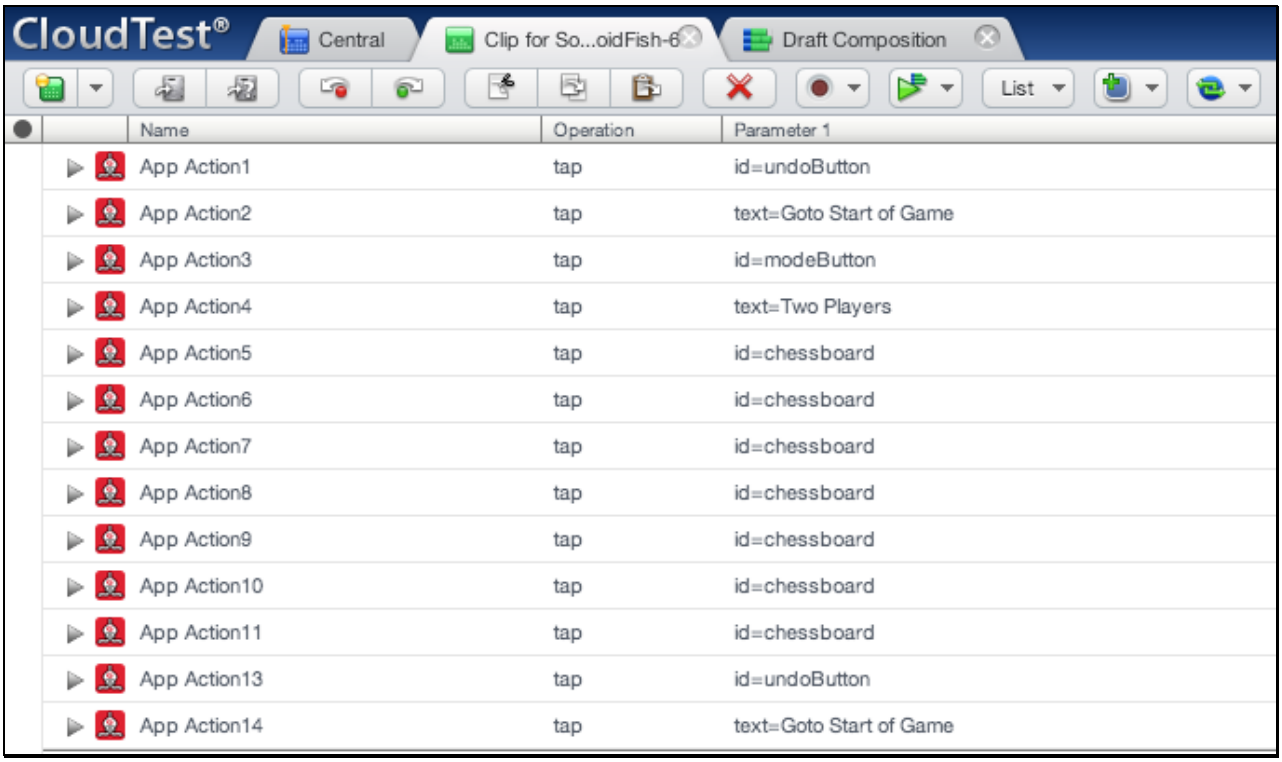
In the remainder of this tutorial advanced editing and test analysis is shown separately for Droidfish (Native) and Zirco Browser (Hybrid), we will add outputs, and then verify, or validate, that an app action's value matches what we expect to find. This validation becomes the basis of creating real-world functional tests.

## Advanced Clip Editing (Droidfish)

Now that we've played this simple Droidfish test composition successfully, and learned how TouchTest will check the success or failure of a given composition, let's return to the test clip to inspect the clip elements and do some additional parameterization.

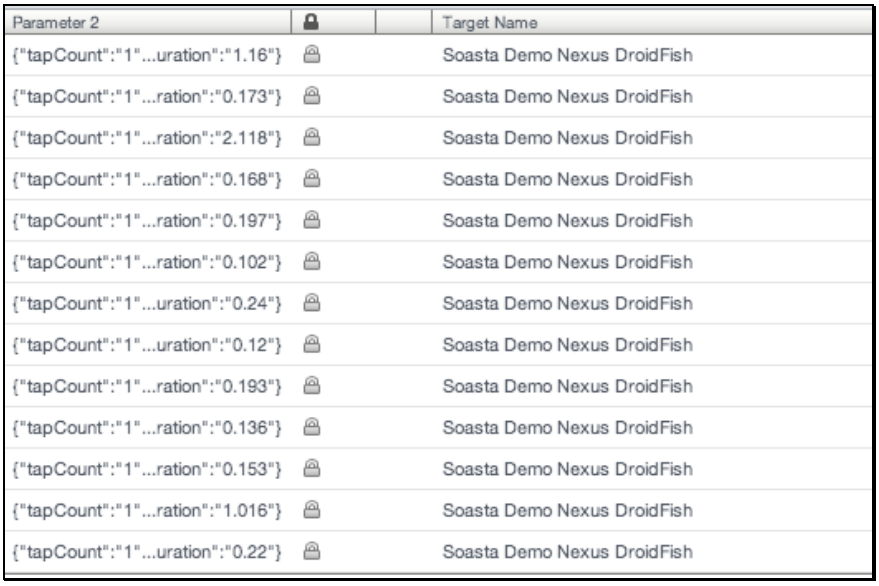
- Click the Clip Editor tab if it's still open, or right-click the test clip in the Composition Editor and choose Open in New Tab.
- Once the Clip Editor tab is in view, click the drop down Icon/List button on the toolbar and then select List.


The List view is useful while clip editing, because it shows all the parameters and their corresponding inputs in one tabular view.




| Name | Operation | Parameter 1 |
|---|---|---|
| App Action1 | tap | id=undoButton |
| App Action2 | tap | text=Goto Start of Game |
| App Action3 | tap | id=modeButton |
| App Action4 | tap | text=Two Players |
| App Action5 | tap | id=chessboard |
| App Action6 | tap | id=chessboard |
| App Action7 | tap | id=chessboard |
| App Action8 | tap | id=chessboard |
| App Action9 | tap | id=chessboard |
| App Action10 | tap | id=chessboard |
| App Action11 | tap | id=chessboard |
| App Action13 | tap | id=undoButton |
| App Action14 | tap | text=Goto Start of Game |

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



| Parameter 2 | | Target Name |
|---|---|---|
| {"tapCount":"1"...uration":"1.16"} | | Soasta Demo Nexus DroidFish |
| {"tapCount":"1"...ration":"0.173"} | | Soasta Demo Nexus DroidFish |
| {"tapCount":"1"...ration":"2.118"} | | Soasta Demo Nexus DroidFish |
| {"tapCount":"1"...ration":"0.168"} | | Soasta Demo Nexus DroidFish |
| {"tapCount":"1"...ration":"0.197"} | | Soasta Demo Nexus DroidFish |
| {"tapCount":"1"...ration":"0.102"} | | Soasta Demo Nexus DroidFish |
| {"tapCount":"1"...uration":"0.24"} | | Soasta Demo Nexus DroidFish |
| {"tapCount":"1"...uration":"0.12"} | | Soasta Demo Nexus DroidFish |
| {"tapCount":"1"...ration":"0.193"} | | Soasta Demo Nexus DroidFish |
| {"tapCount":"1"...ration":"0.136"} | | Soasta Demo Nexus DroidFish |
| {"tapCount":"1"...ration":"0.153"} | | Soasta Demo Nexus DroidFish |
| {"tapCount":"1"...ration":"1.016"} | | Soasta Demo Nexus DroidFish |
| {"tapCount":"1"...uration":"0.22"} | | Soasta Demo Nexus DroidFish |

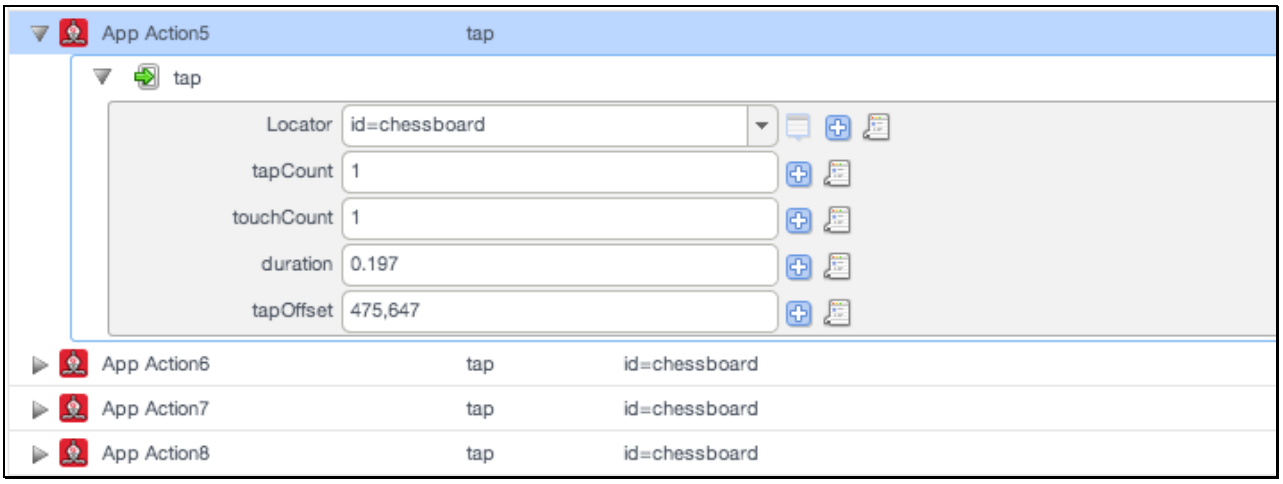
## Inspecting App Action Details (Droidfish)

Examine elements and properties for any App Action by selecting it in the workspace above and then click its Gear icon to Show Info. When you do so, the Info Window appears.

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.

- Locate App Action5 and expand it by clicking the arrow. Note that when you hover the mouse over the expanded app action the Add toolbar appears on the expanded row. This toolbar shows icons for Pre-Action Waits, Post-Action Waits, Outputs, Validations, and Property Sets. Any click on one of these will add the relevant form to the given app action.

4. Expand the tap action (if not already expanded) to inspect the Inputs of this action.




Because all the moves in DroidFish use the *id-chessboard* in this example, this means that although the test plays successfully (even though the *tapOffset* values distinguish one from another) this doesn't result in a very helpful test in terms of human readability.

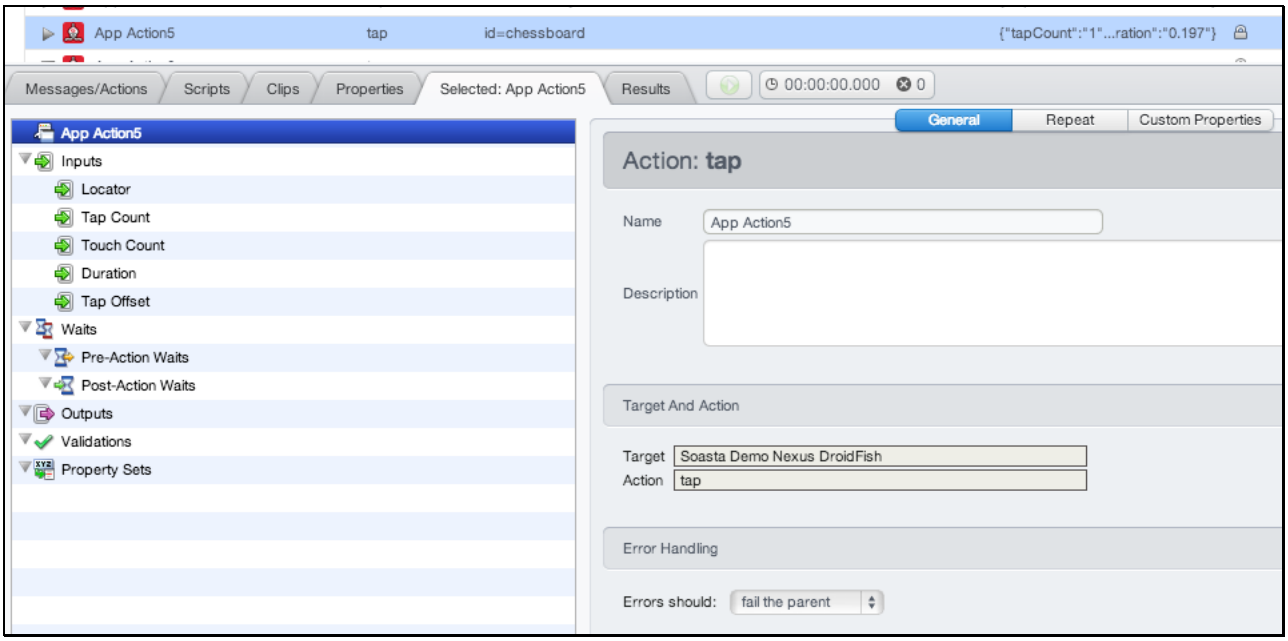
### App Action Properties (Droidfish)

Additional parameters, such as Custom Properties, can be set by double-clicking an app action to open it in the Clip Editor lower panel. 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 that were seen in the expanded list view steps above can also be set in the lower panel by clicking that node in the tree and then performing the desired action on the right.

1. In the *Selected: App Action5* 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 TouchTest 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, however, as a general rule they are used sparingly.

-  **Validations**

  Validations verify some 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 TouchTest 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 TouchTest 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 TouchTest environment—and can be used across compositions.

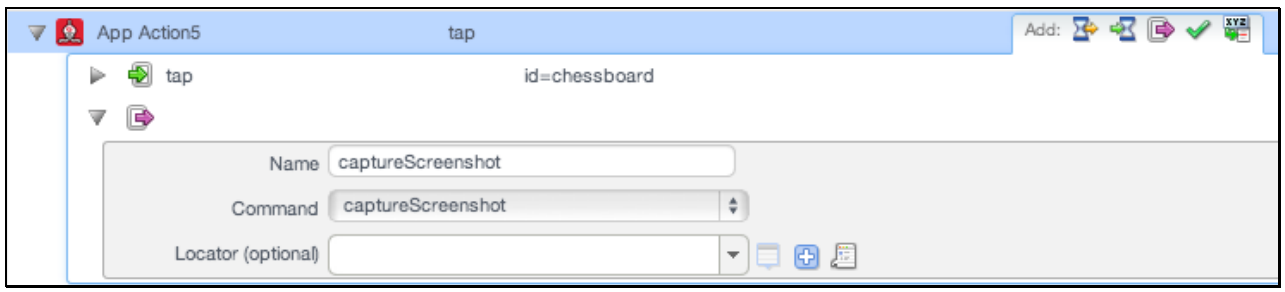
## Adding Outputs (Droidfish)

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.

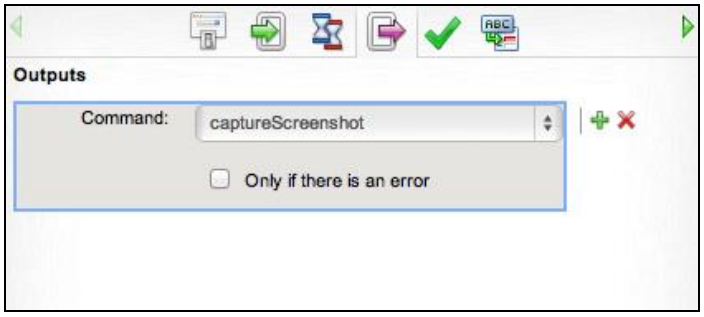
Additionally, we will add an output of the View Hierarchy in order to learn more about interesting things to validate. If you're a developer, you may already know many such things, however, if you're a tester who is not as familiar with an app's code base, this output is very useful.

1. Expand App Action5 in the Clip Editor, List view.


2. While hovering over this action, click the Outputs icon (fourth icon from the left) on the Add toolbar.



An Output form is added to the action with the default output, *captureScreenshot* shown. Leave the default *captureScreenshot* selected.



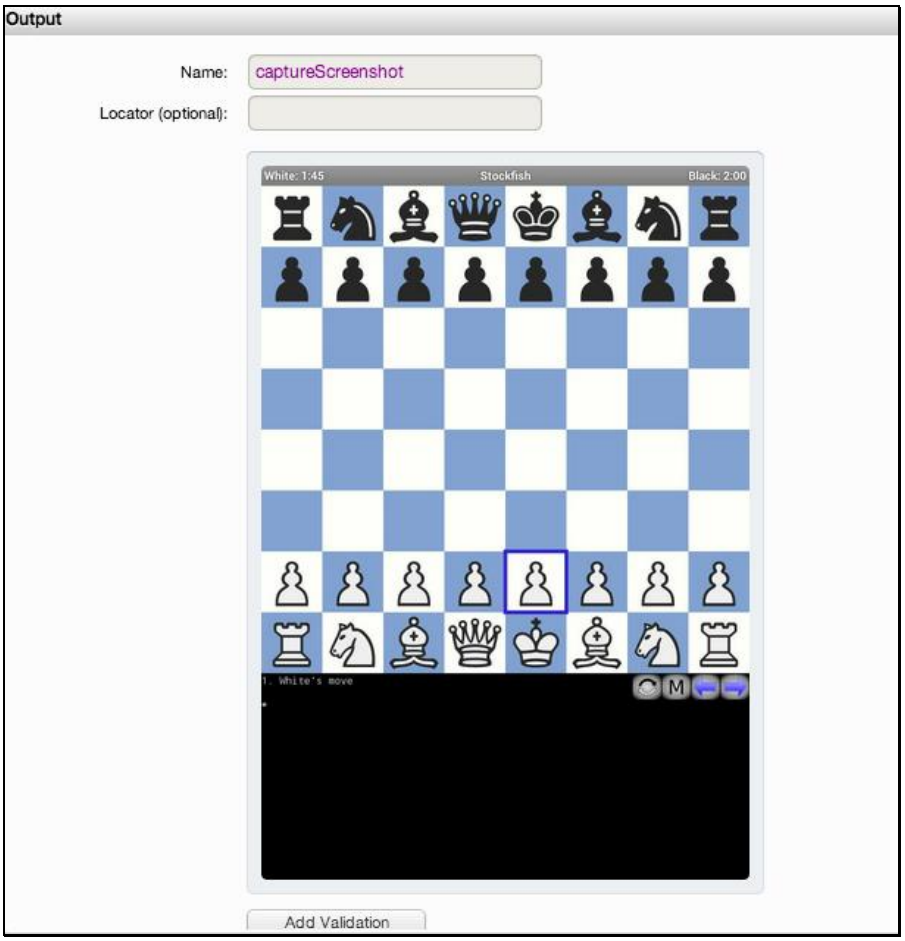

- Leave *Only if there is an error* unchecked to get a screenshot in every eventuality.

3. Click the Output icon on the Add toolbar a second time. A second form is added to the panel.
4. In the second output, click the Command drop down and select *outputViewHierarchy*. Leave the Locator field blank to get the entire view.


5. Click Save on the Clip Editor toolbar.
6. Return to the Composition Editor tab once again and click Play a second time.

## Inspecting Outputs (Droidfish)

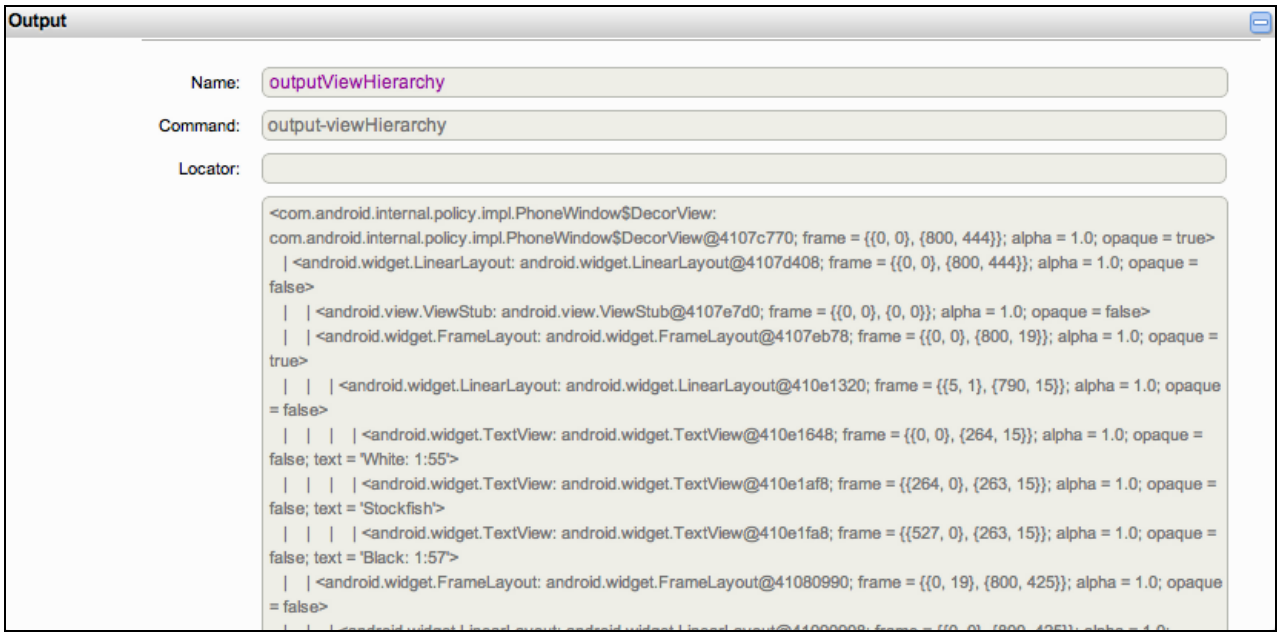
1. In the Result Details dashboard, select the App Action5 in the navigation tree or in the cover flow, locate the Outputs panel, and click its Maximize icon.




2. Scroll down in the Output panel to view the result for *outputViewHierarchy*. Look for interesting things to validate.

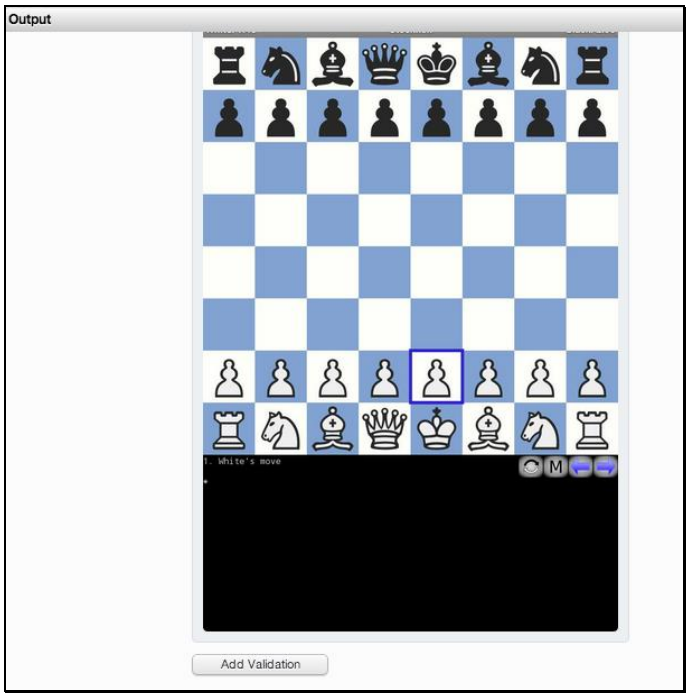
3. Optionally, copy the content of outputViewHierarchy for DroidFish and paste it into a separate text file.

This output can provide many text elements for validations that are of great use to a tester unfamiliar with an app's code base.



### Add an Image Validation (Droidfish)

1. Click the Validation button beneath the captured screenshot in the Output panel.




2. When you do so, focus returns to the Clip Editor and a *verifyScreenshot* is added to the given action (in this case, App Action5).
3. Specify a tolerance of 90 (as a percentage of image variation). If this fails when you play the test composition, try setting it to 80, and so forth.

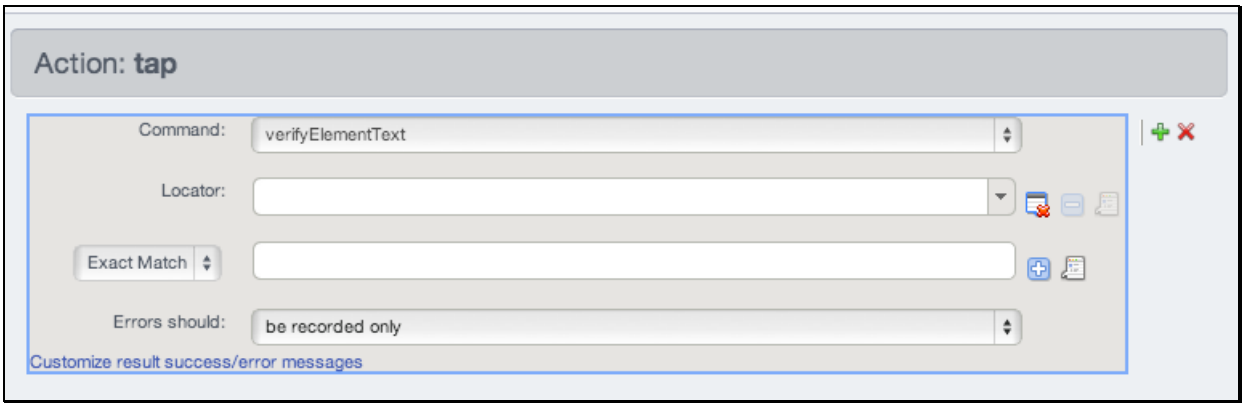
   In this case, accept the default Failure action for this validation.
4. Save the test clip.
5. In the Composition Editor, click Play once again.

### Add a Text Validation (Droidfish)

Since the test clip is already open, let's also add a text validation to the same action (App Action5). This time we will use the lower panel to do so. Ensure that your mobile device is connected and running TouchTest Agent before proceeding with the following steps.

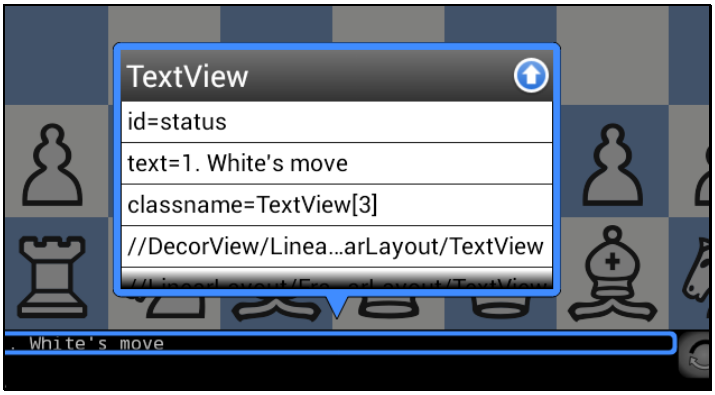
1. Double click App Action5 to open it in the lower panel.
2. Locate Validations in the tree on the left.
3. Click the green Plus (+) icon on the previously added form (the *verifyScreenshot* we added above).
4. Change the Command drop-down on the new validation form to *verifyElementText*.
5. Click the Record button (since Droidfish text mainly appears in the console area, we need to get the console locator).
6. Invoke Touch Locator mode by clicking the first icon to the right of the Locator field (shown below).

**TIP:** For more about the Touch Locator feature, see Touch Locator for Mobile Apps.



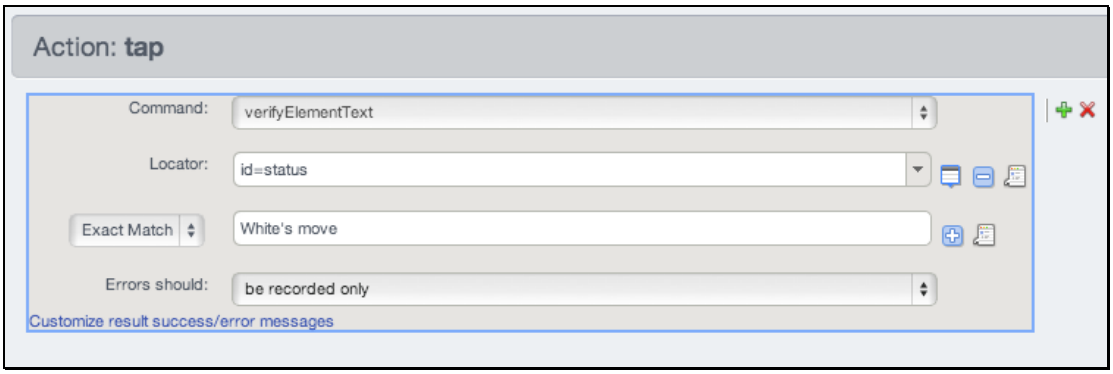

7. On the Android device, long press the text "1. White's move" until the blue border is constrained to that field.

8. When you do so, the Touch Locator box appears with the available locators.



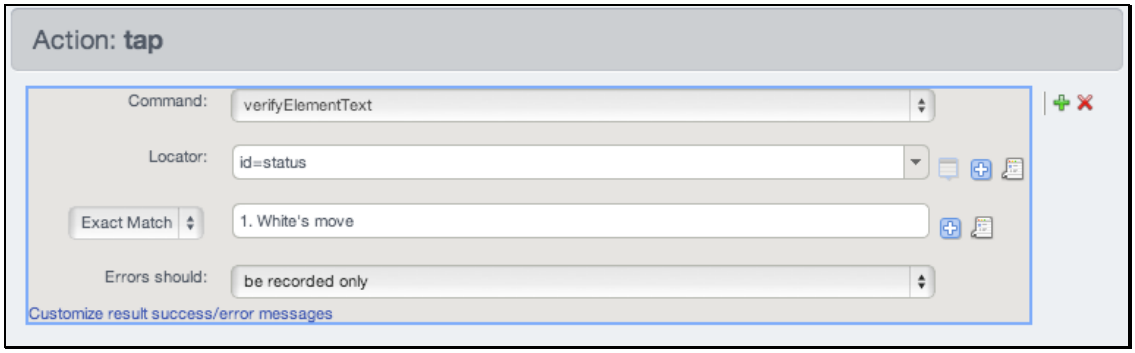

9. Click the Up Arrow icon to accept these locators and then click Record in the Clip Editor to stop the session. Before proceeding, delete any out-of-sequence app actions that you inadvertently recorded while in Touch Locator mode (e.g. since App Action5 was selected here, delete anything between it and App Action6).

10. In the Clip Editor, inspect the *verifyElementText* form in App Action5.




The locator field is now populated with the locator of the field whose text we want to verify. Optionally, click the Locator drop-down and note that all of the locators that appeared on the device for the given field have been populated but only the first is used. Should the validation fail, you can try one of the other locators. Also, note that we didn't change the locator for the app action itself but rather added a locator to *verifyElementText* solely.

11. In the Match field, leave Exact Match set and enter `1. White's move`.



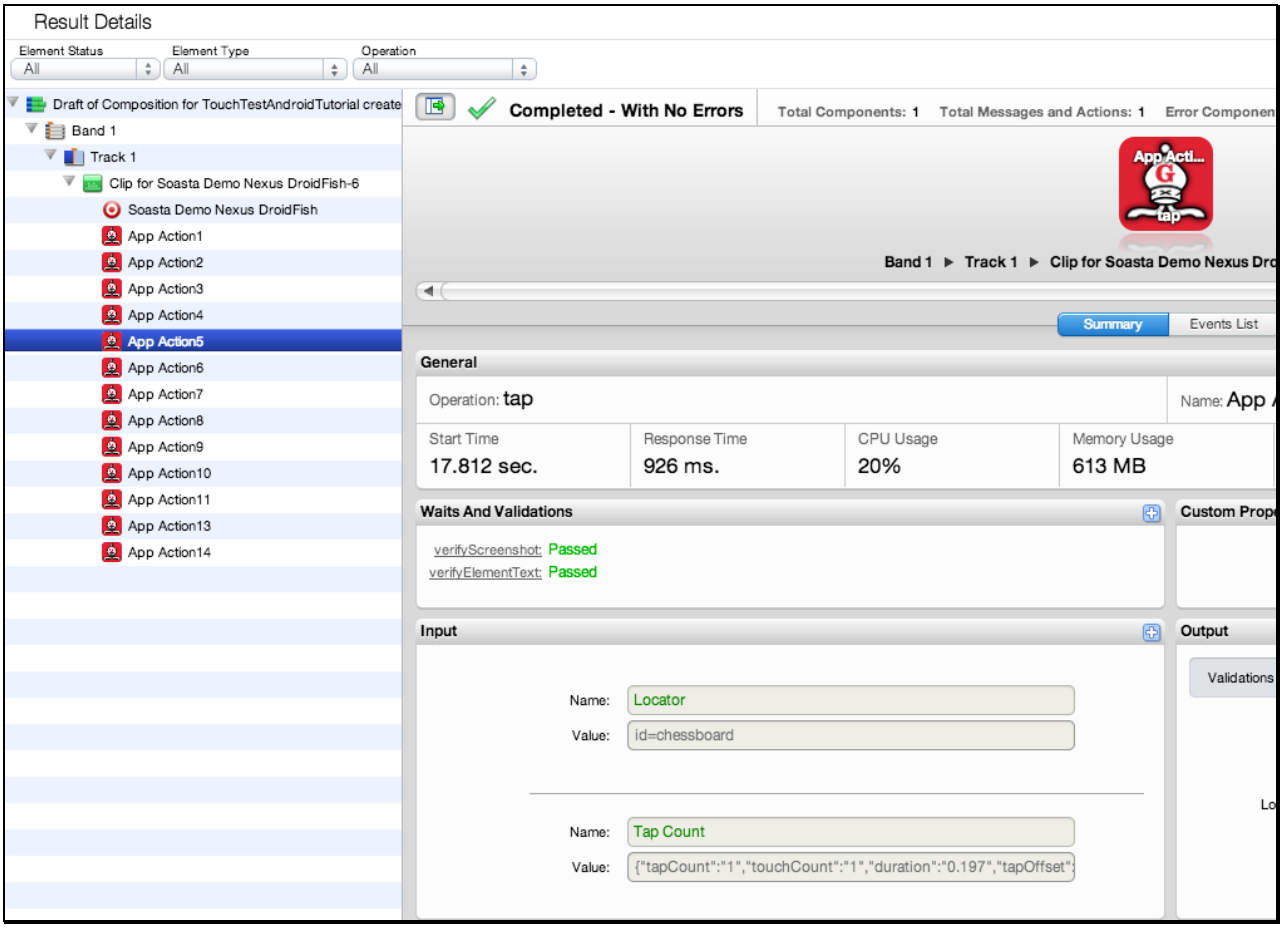

**TIP:** You can also verify on partial strings using regex or glob. Refer to Validations for Browser or App Actions for more about matching.

12. Save the test clip.

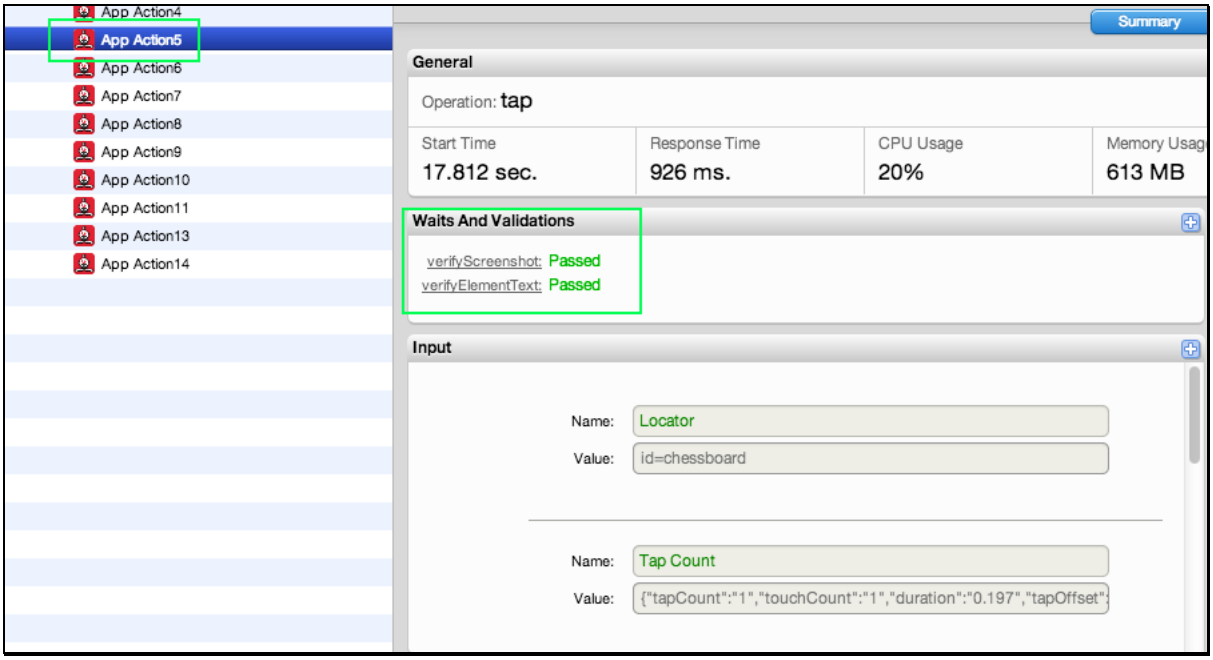
13. In the Composition Editor, click Play once again.

## Analyzing Validations in Results (Droidfish)

Now that we've added validations on an image and text, we will learn how to examine those validations on their merits. 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.

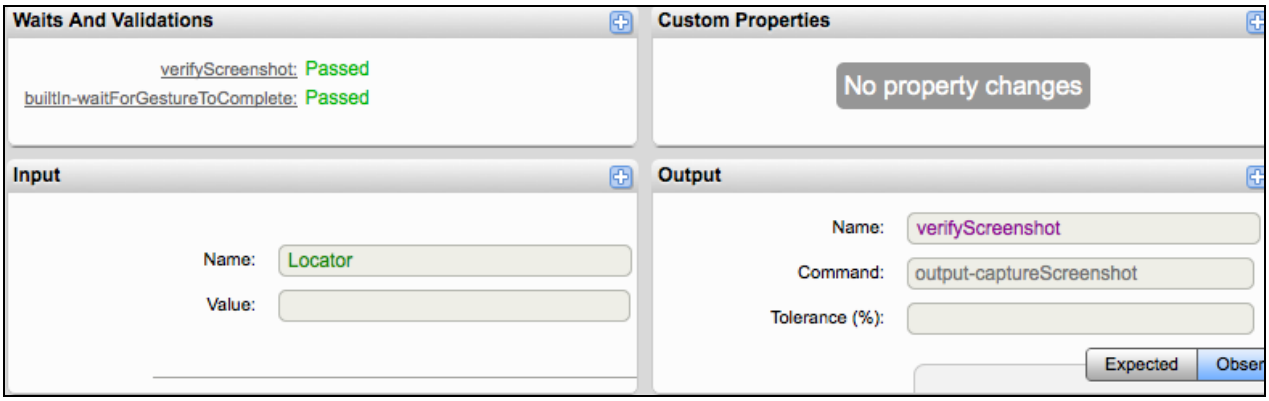



1. In Result Details, select the clip element that had the validation. For example, App Action5 (as shown above and below).




2. Inspect the information on the Summary tab for the selection. In the result above, the validation on App Action5 passed in the result shown.

3. Click *verifyScreenshot* in the Waits and Validations section to bring it into focus in the Output section.

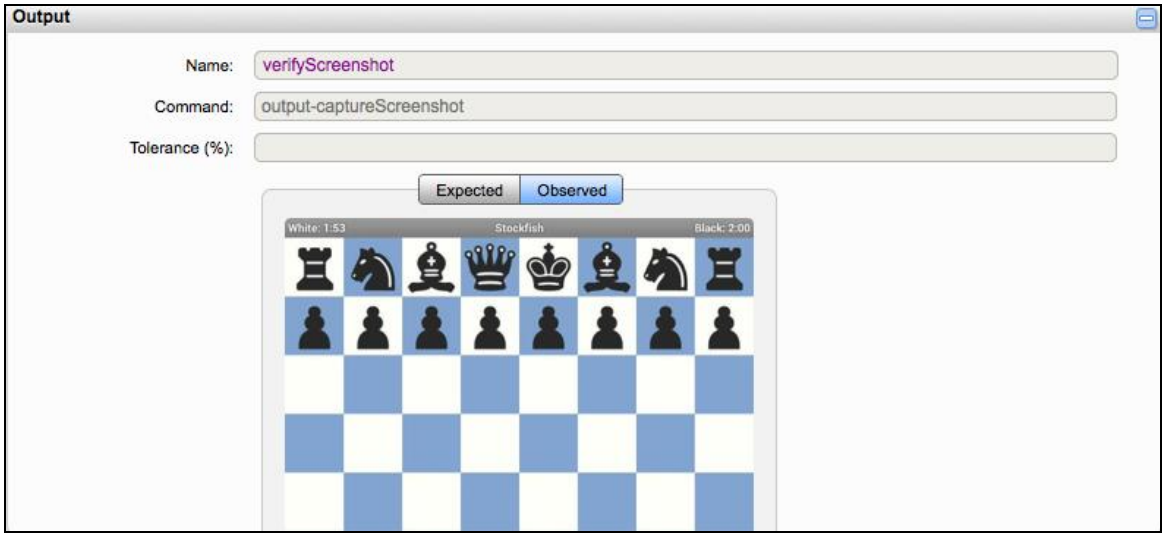



**Note:** Since we didn't check *Only if there is an error* in the Output form a shot of the success is included in this result for the given app action.

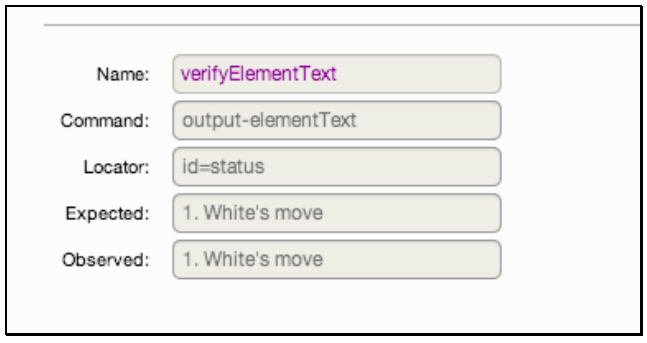
4. Click the Events list tab for the given selection to view action-related events, including validations. Click the Details arrow to inspect any event's details.

5. Click the Output section's Maximize icon to view the results for the *verifyScreenshot* and *verifyElementText* on this action in full.

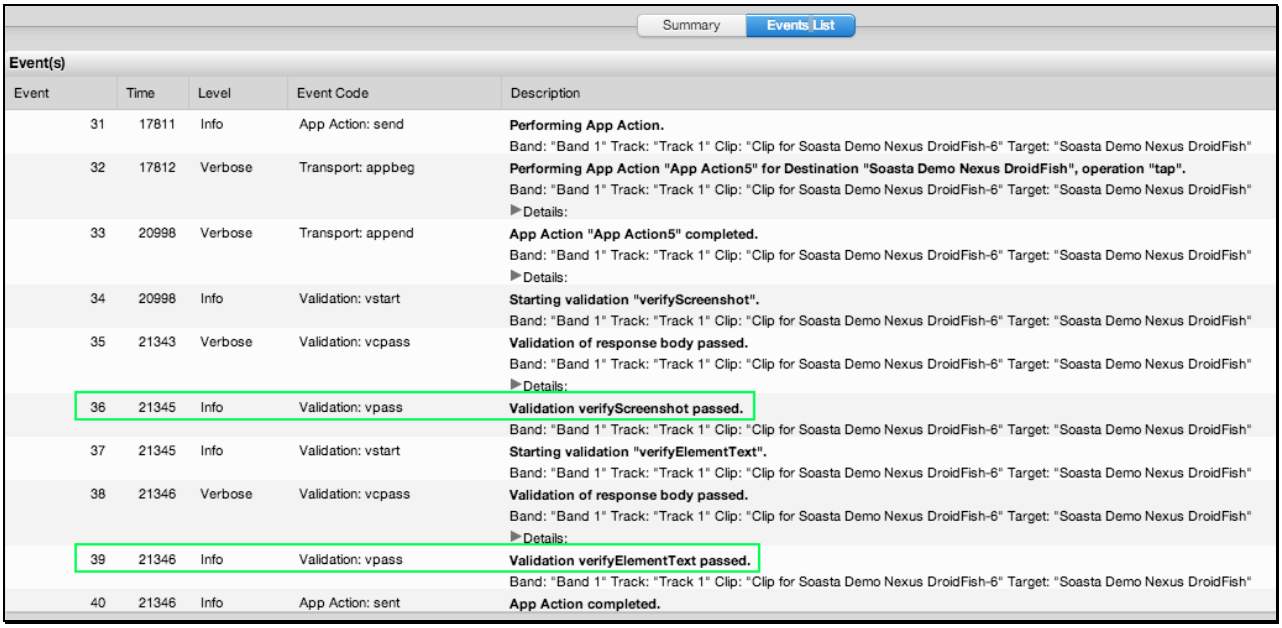
Click between the Expected and Observed tabs to see the comparison images.




- In this case the *verifyScreenshot* passed. If yours didn't pass, try lowering the tolerance (from 90 to 80 and beyond).
- In this case, the *verifyElementText* also passed.




6. Click the Events List tab in the middle panel of Result Details to view the complete text stream of events for the given selection. Note the validation relevant headings.


Summary
Events List
Event(s)
Event
Time
Level
Event Code
Description
31 17811 Info App Action: send
Performing App Action.
Band: "Band 1" Track: "Track 1" Clip: "Clip for Soasta Demo Nexus DroidFish-6" Target: "Soasta Demo Nexus DroidFish"
32 17812 Verbose Transport: appbeg
Performing App Action "App Action5" for Destination "Soasta Demo Nexus DroidFish", operation "tap".
Band: "Band 1" Track: "Track 1" Clip: "Clip for Soasta Demo Nexus DroidFish-6" Target: "Soasta Demo Nexus DroidFish"
Details:
33 20998 Verbose Transport: append
App Action "App Action5" completed.
Band: "Band 1" Track: "Track 1" Clip: "Clip for Soasta Demo Nexus DroidFish-6" Target: "Soasta Demo Nexus DroidFish"
Details:
34 20998 Info Validation: vstart
Starting validation "verifyScreenshot".
Band: "Band 1" Track: "Track 1" Clip: "Clip for Soasta Demo Nexus DroidFish-6" Target: "Soasta Demo Nexus DroidFish"
35 21343 Verbose Validation: vcpass
Validation of response body passed.
Band: "Band 1" Track: "Track 1" Clip: "Clip for Soasta Demo Nexus DroidFish-6" Target: "Soasta Demo Nexus DroidFish"
Details:
36 21345 Info Validation: vpass
Validation verifyScreenshot passed.
Band: "Band 1" Track: "Track 1" Clip: "Clip for Soasta Demo Nexus DroidFish-6" Target: "Soasta Demo Nexus DroidFish"
37 21345 Info Validation: vstart
Starting validation "verifyElementText".
Band: "Band 1" Track: "Track 1" Clip: "Clip for Soasta Demo Nexus DroidFish-6" Target: "Soasta Demo Nexus DroidFish"
38 21346 Verbose Validation: vcpass
Validation of response body passed.
Band: "Band 1" Track: "Track 1" Clip: "Clip for Soasta Demo Nexus DroidFish-6" Target: "Soasta Demo Nexus DroidFish"
Details:
39 21346 Info Validation: vpass
Validation verifyElementText passed.
Band: "Band 1" Track: "Track 1" Clip: "Clip for Soasta Demo Nexus DroidFish-6" Target: "Soasta Demo Nexus DroidFish"
40 21346 Info App Action: sent
App Action completed.

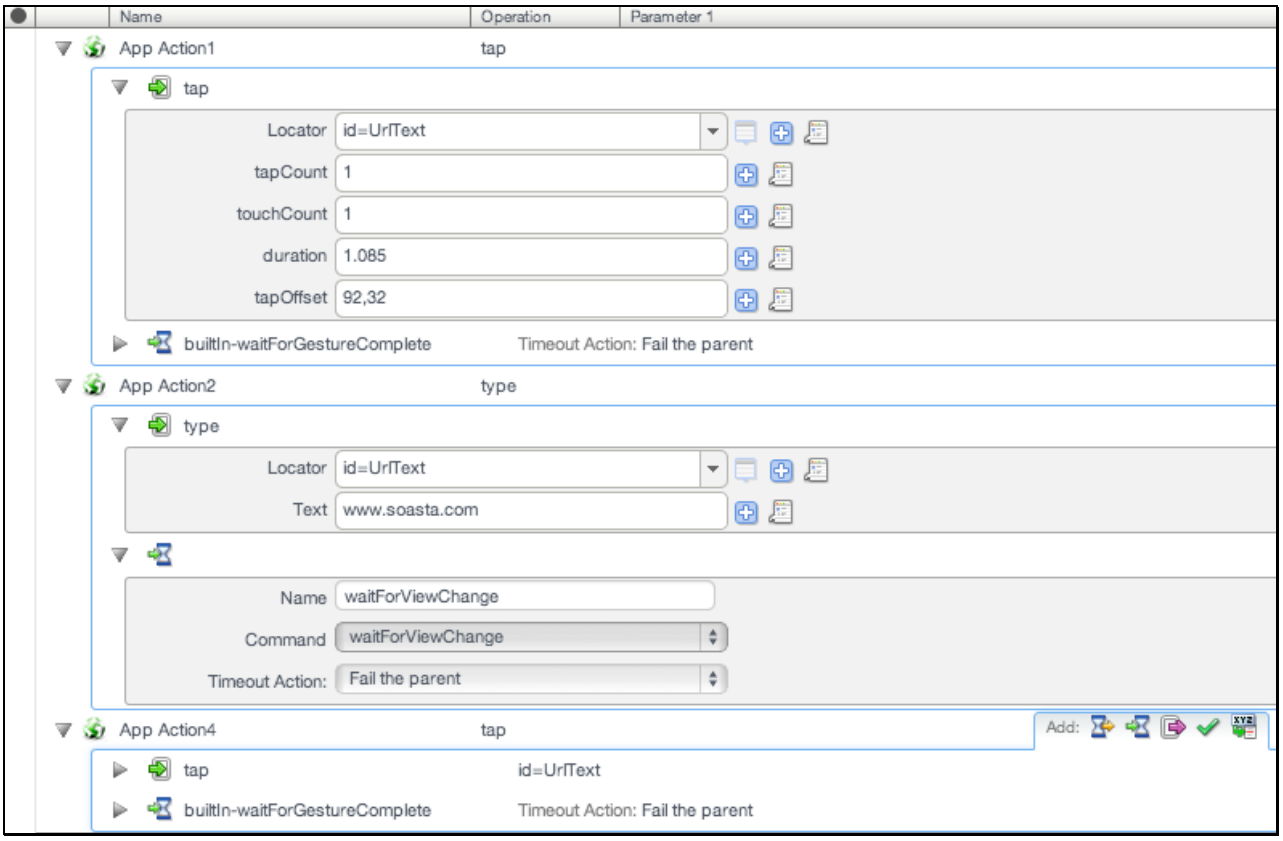
## Advanced Clip Editing (Zirco Browser)

Now that we've played this simple Zirco Browser test composition successfully, and learned how TouchTest will check the success or failure of a given composition, let's return to the test clip to inspect the clip elements and do some additional parameterization.

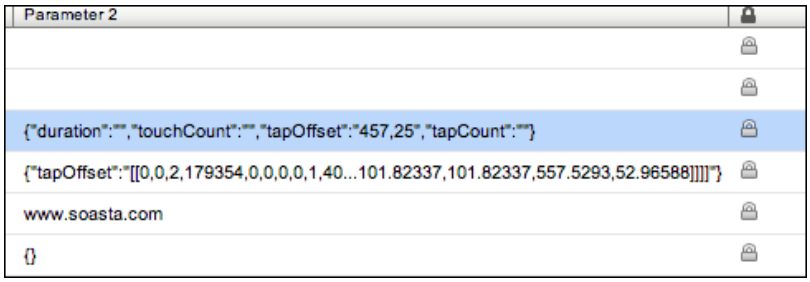
- Click the Clip Editor tab if it's still open, or right-click the test clip in the Composition Editor and choose Open in New Tab.
- Once the Clip Editor tab is in view, click the drop down Icon/List button on the toolbar and then select List.


The List view is useful while clip editing, because it shows all the parameters and their corresponding inputs in one tabular view.

Collapse and expand the App Action to access or hide its details; hover the mouse over a row to display the Add toolbar, which is used to add Pre- and Post-Action Waits, Outputs, Validations, and Property Sets. The expanded action consists of the action inputs, built-in waits associated with the action, and any accessors manually added by the user via the Add toolbar (or, via the lower panel Action Editor).




For a collapsed row, additional information is also displayed to the right of the Parameter 1 column.

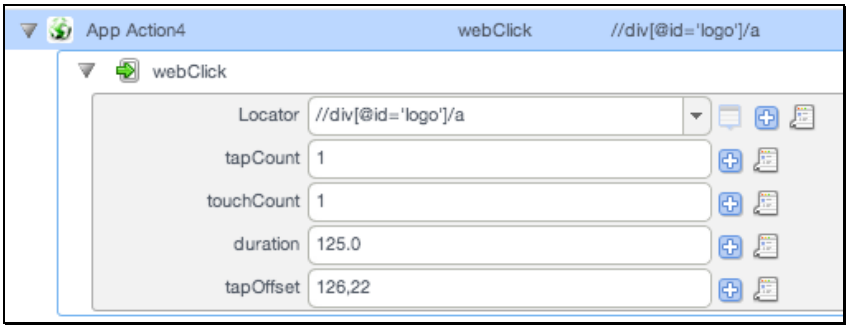


### Inspecting App Action Details (Zirco Browser)

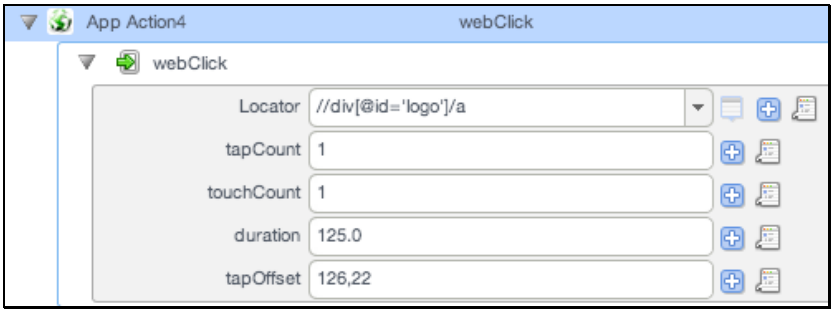
Examine elements and properties for any App Action in the workspace by expanding. When you do so, the Info Window appears.

In the test clip below the recorded App Action4 is expanded to show its details. The type of app action, *webClick,* represents the first action in the clip that renders a web page. This is the archetypal "hybrid" app action. In some cases, its parameter is an Xpath (just as it would be for a browser action in a desktop functional test). In other cases, a classname or even a TouchTestID will appear here.

- Locate App Action7 and click its Gear icon to pop out the Info Window.




5. Expand the *webClick* under App Action4. Its locator is an Xpath representing the SOASTA Logo.

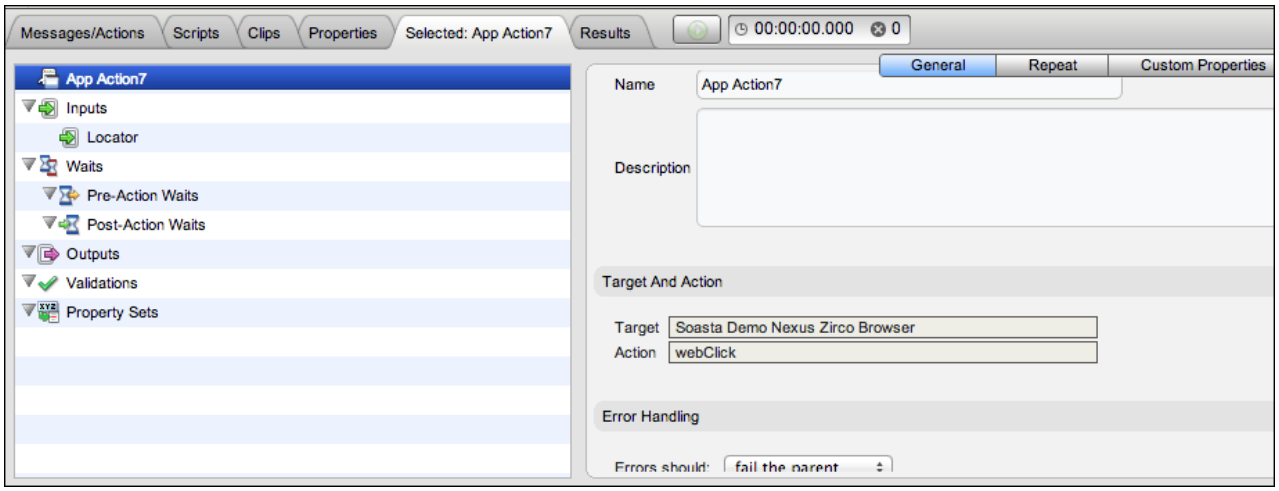


## App Action Properties (Zirco Browser)

Additional parameters, such as Custom Properties, can be set by double-clicking an app action to open it in the Clip Editor lower panel. Action level properties are shown in the tree on the left.

2. 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 that were seen in the expanded list view steps above can also be set in the lower panel by clicking that node in the tree and then performing the desired action on the right.

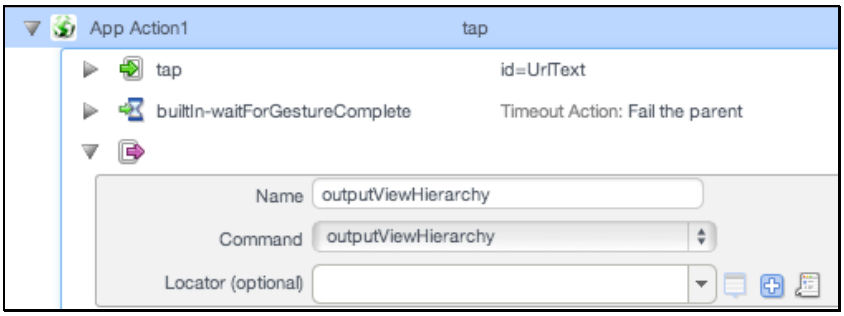
## Adding Outputs (Zirco Browser)

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.

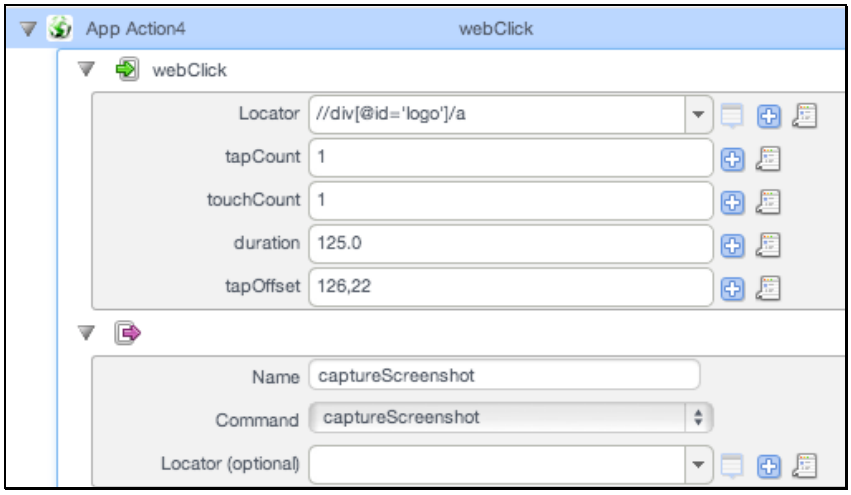
Additionally, we will add an output of the View Hierarchy in order to learn more about interesting things to validate. If you're a developer, you may already know many such

things, however, if you're a tester who is not as familiar with an app's code base, this output is very useful.

1. Click the Add toolbar, Outputs icon for App Action1 to add the new output.
2. Select *outputViewHierarchy* in the Command drop down.

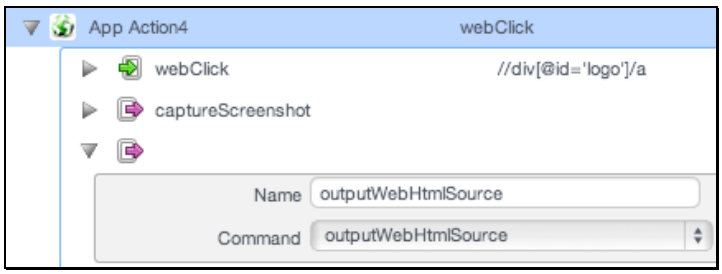



3. Leave the Locator field blank. This will return the entire hierarchy.
4. Optionally, also add an *outputXMLHierarchy* on App Action1.
5. Click the Add toolbar, Outputs icon for App Action4 to add an output.
6. Leave the default, *captureScreenshot*, set on App Action4.




- Leave *Only if there is an error* unchecked to get a screenshot in every eventuality.

7. Optionally, also add an *outputWebHtmlSource* on App Action4. This will provide a resource for interesting text to validate.



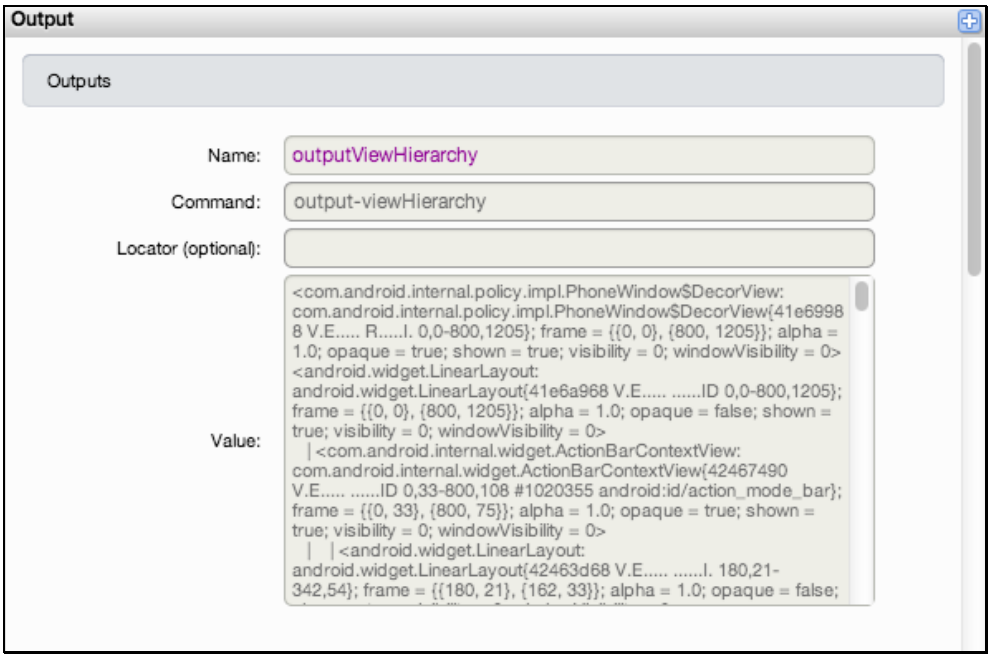

8. Click Save on the Clip Editor toolbar.

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

## Inspecting Outputs (Zirco Browser)

Now that we've added outputs to our Zirco Browser clip, let's check the Result Details dashboard for them.

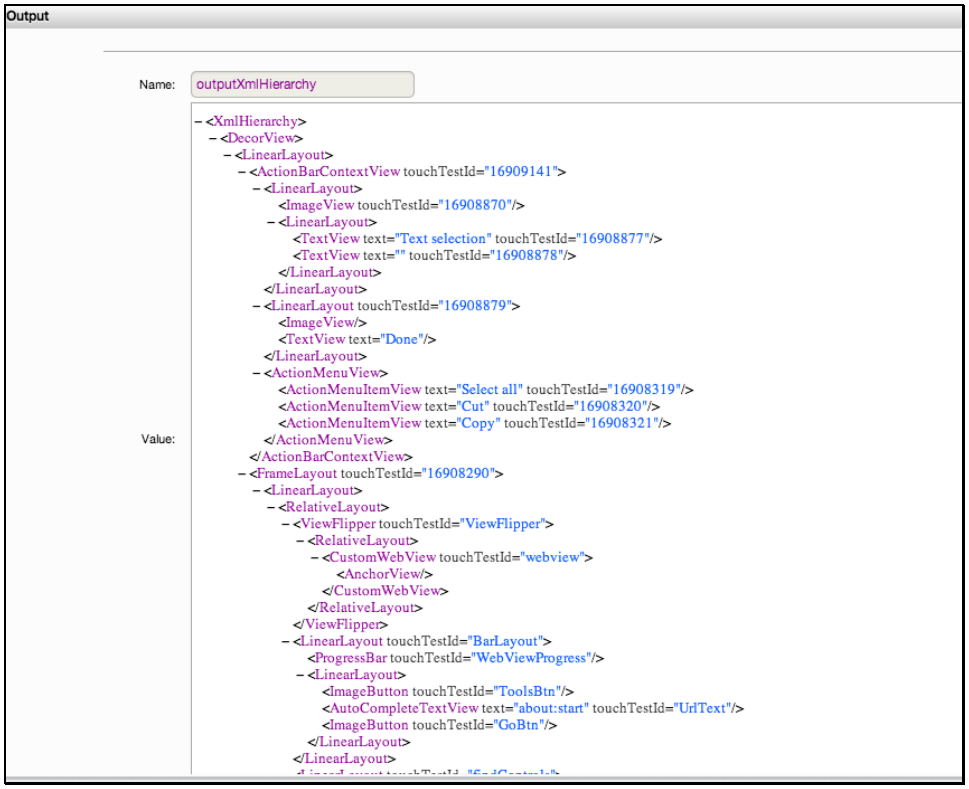
4. In the Result Details dashboard, select the App Action1 in the navigation tree or in the cover flow, locate the Outputs panel, and click its Maximize icon.




5. In the Output panel, locate the *outputViewHierarchy*. Look for interesting things to validate in the text that you find.

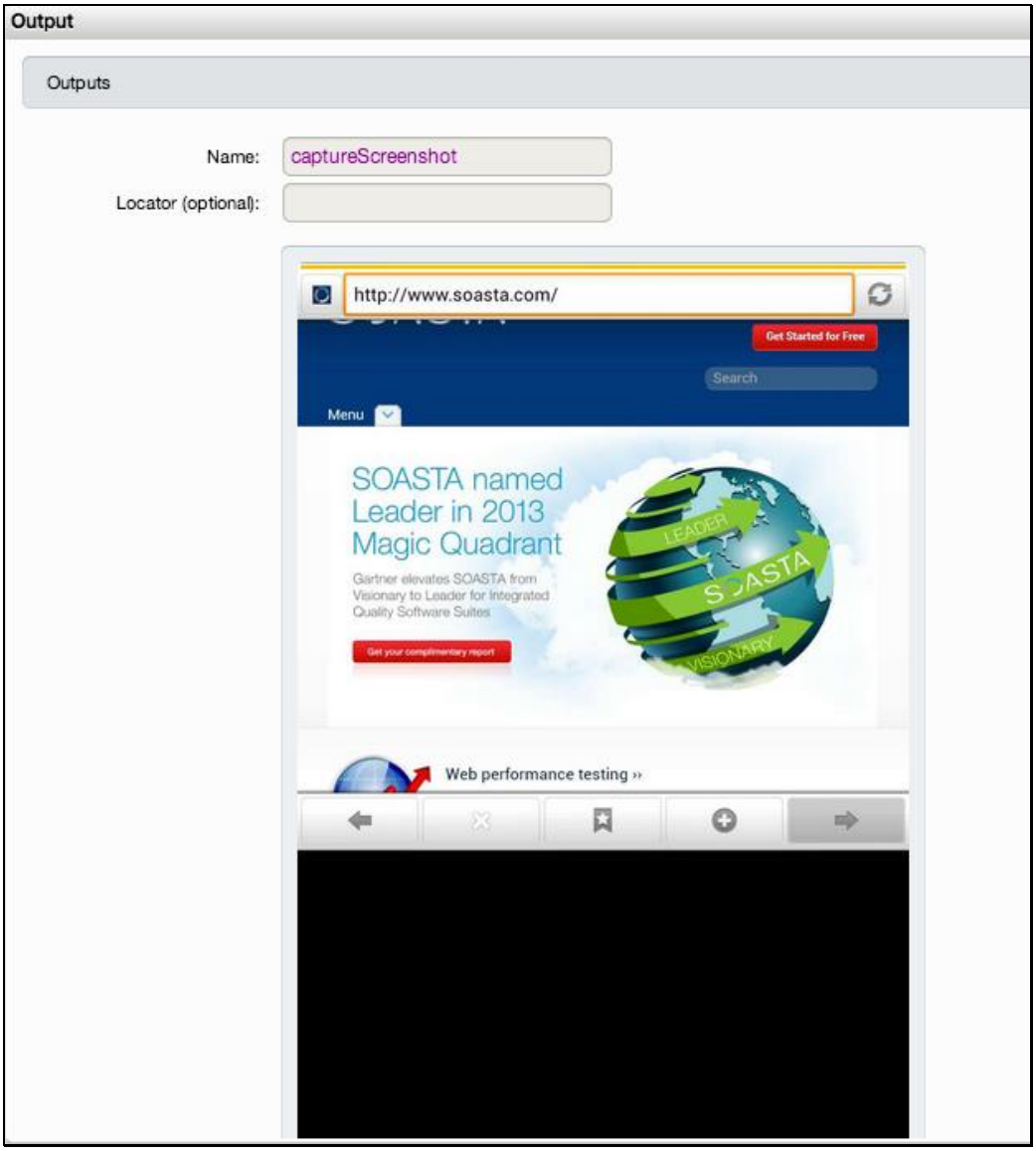
   Optionally, copy the text from this output into a separate text file to use as a source for interesting things to validate in the Zirco Browser app (as opposed to its web content). The View Hierarchy can provide many text elements for validation and property set creation that are of great use to a tester unfamiliar with an app's code base.

6. If you also added *outputXMLHierarchy*, locate it as well.

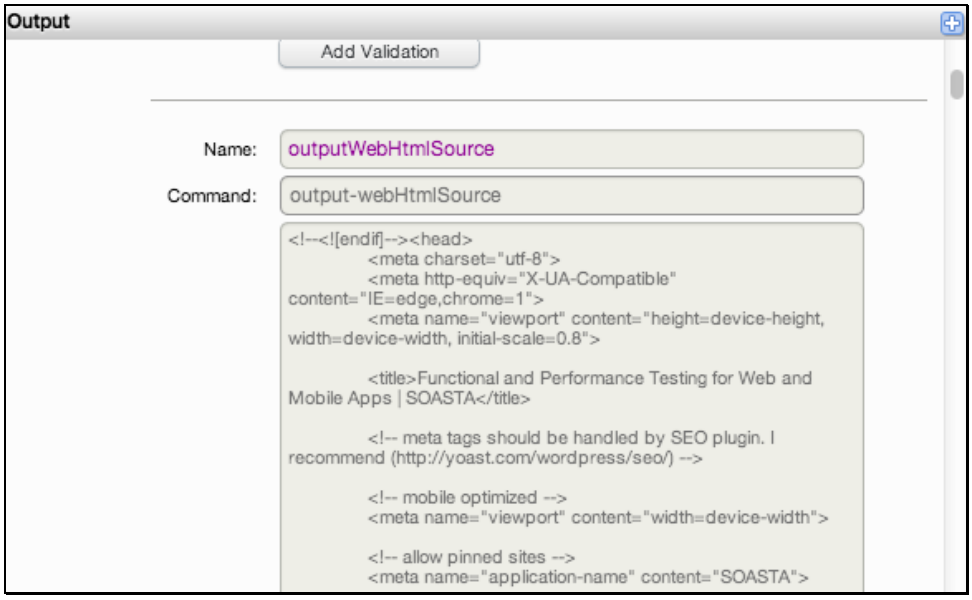



10. Finally, select App Action4 in the Result Details dashboard, and once again locate the Outputs section.

Expand it to view the screenshot set on this action.

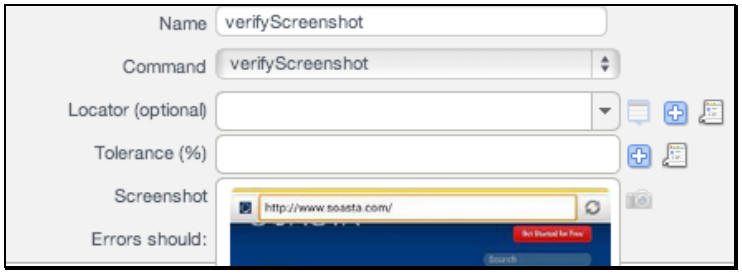


7. If you added *outputWebHtmlSource*, locate it in the Outputs section as well.




## Add an Image Validation (Zirco Browser)

1. Click the Add Validation button beneath the captured screenshot in the Output panel.
2. When you do so, focus returns to the Clip Editor and a *verifyScreenshot* is added to the action (in this case, App Action4).
3. Leave Locator blank.



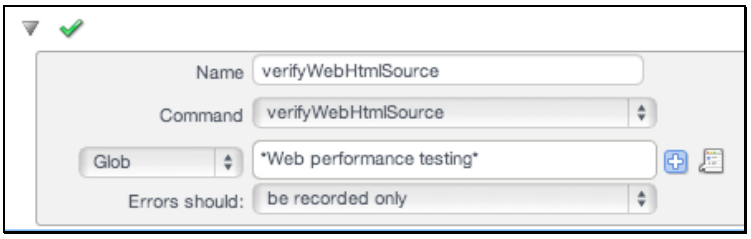

4. Specify a tolerance of 90 (as a percentage of image variation). If this fails when you play the test composition, try setting it to 80, and so forth.
5. In this case, accept the default Failure action for this validation.
6. Save the test clip.

7. In the Composition Editor, click Play once again.

### Adding an HTML Validation (Zirco Browser)

Since the test clip is already open, let's also add a validation on text from an HTML page to the same action (App Action4). This time we will use the lower panel to do so. Ensure that your mobile device is connected and running TouchTest Agent before proceeding with the following steps.

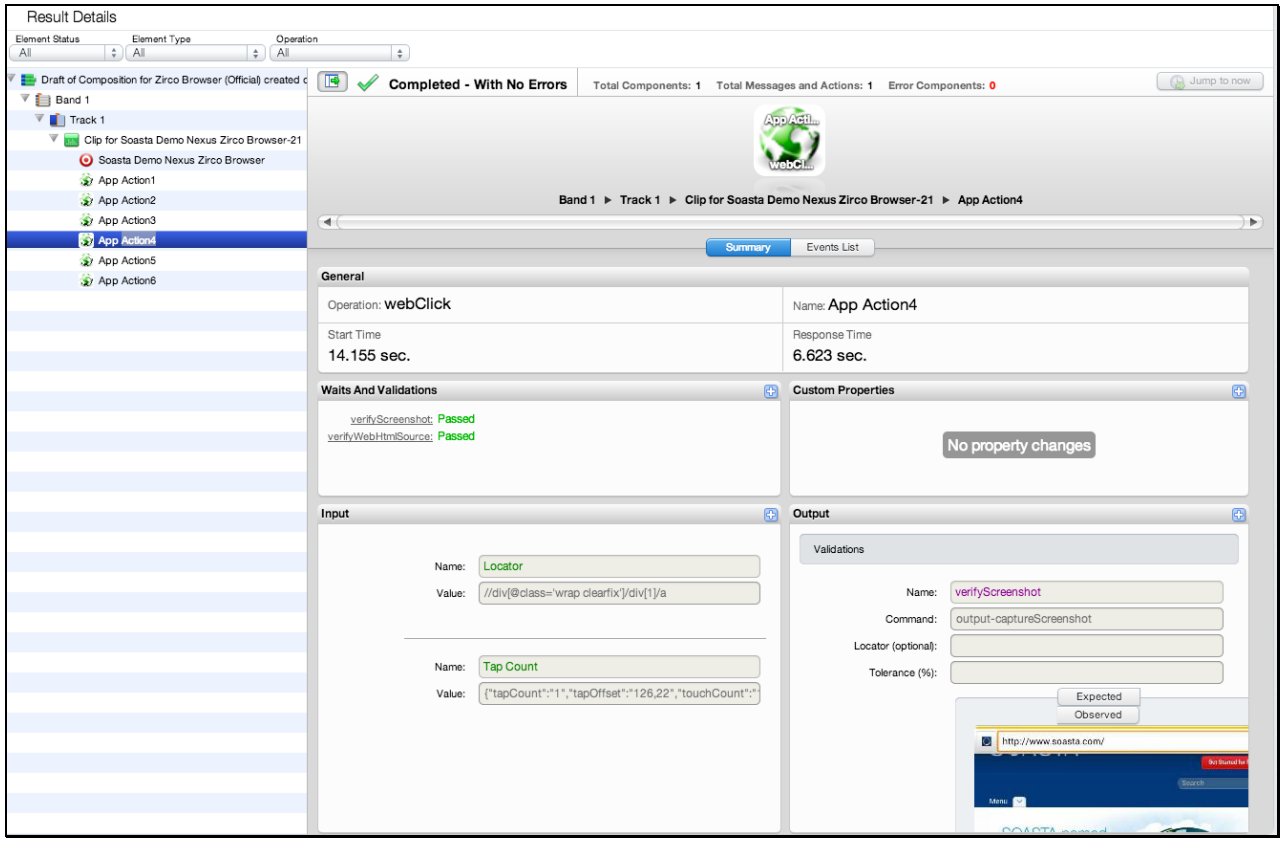
1. Double click App Action4 to open it in the lower panel.
2. Locate Validations in the tree on the left.
3. Click the green Plus (+) icon on the previously added form (the *verifyScreenshot* we added above).
4. Change the Command drop-down on the new validation form to *verifyWebHtmlSource*.
5. Change the Match field to Glob and then enter *`Web performance testing`*.



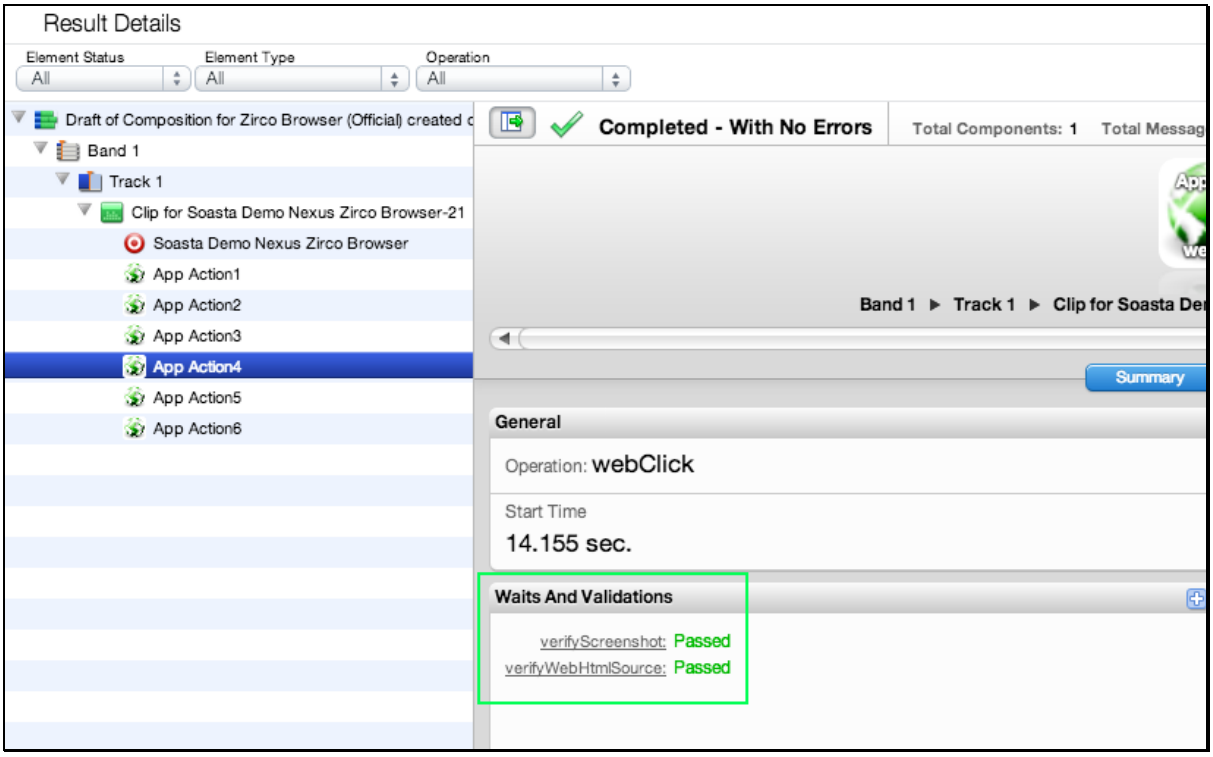

1. In this case, accept the default Failure action for this validation *Be recorded only*.
2. Save the test clip.
3. In the Composition Editor, click Play once again.

## Analyzing Validations in Results (Zirco Browser)

Now that we've added validations on an image, and on text in an HTML page, we will learn how to examine those validations on their merits. 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.

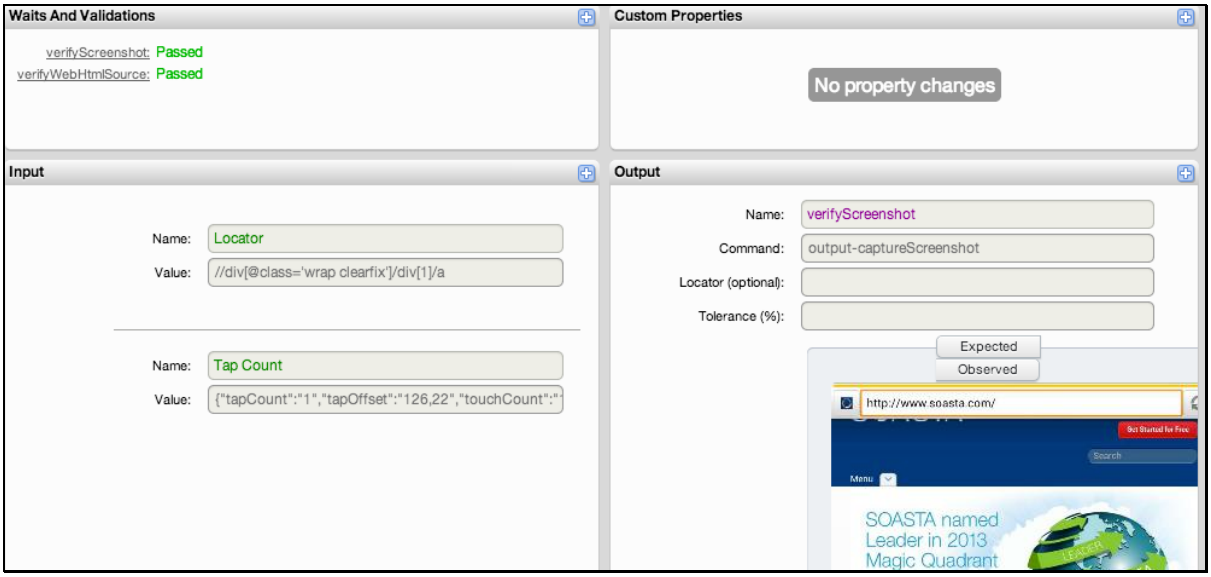


7. In Result Details, select the clip element that had the validation. For example, App Action4 (as shown below).



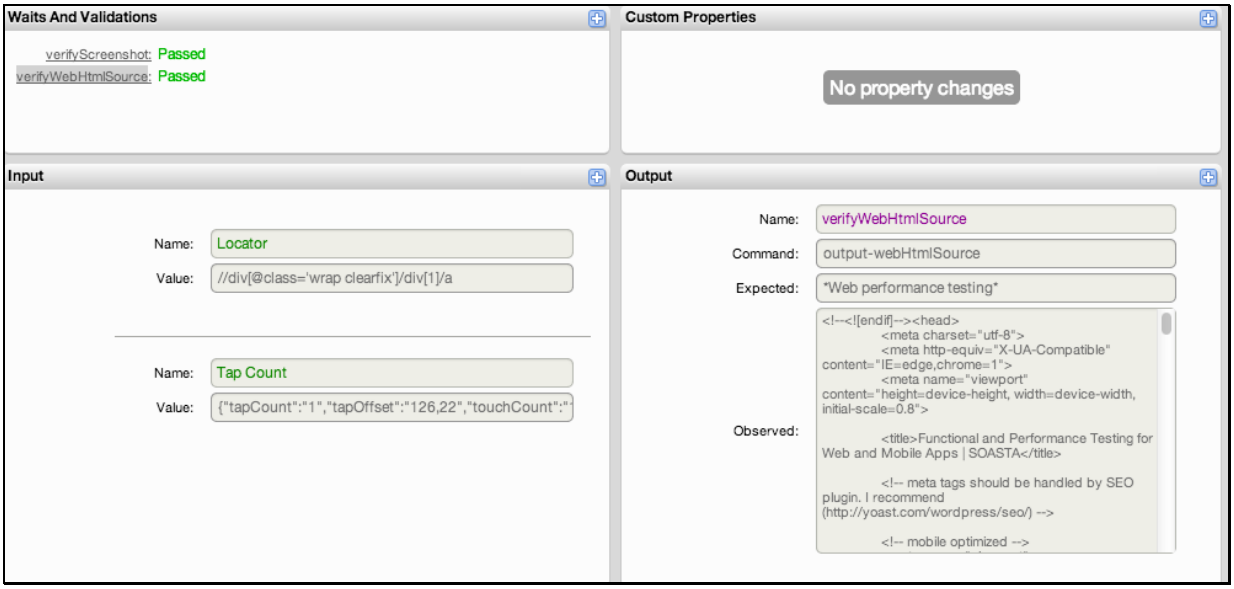

8. Inspect the information on the Summary tab for the selection. In the result above, both validations on App Action4 passed in the result shown.

9. Click *verifyScreenshot* in the Waits and Validations section to bring it into focus in the Output section.




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

1. Click *verifyWebHtmlSource* in the Waits and Validations section to bring it into focus in the Output section.




2. 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.

3. Click the Output section's Maximize icon to view the *verifyScreenshot* in full. Click between the Expected and Observed tabs to see the comparison images.

4. Click the Events List tab in the middle panel of Result Details to view the complete text stream of events for the given selection. Note the validation relevant headings.

Event(s)

| Event | Time | Level | Event Code | Description |
|---|---|---|---|---|
| 23 | 14155 | Info | App Action: send | **Performing App Action.** Band: "Band 1" Track: "Track 1" Clip: "Clip for Soasta Demo Nexus Zirco Browser-21" Target: "Soasta Demo Nexus Zirco Browser" |
| 24 | 14155 | Verbose | Transport: appbeg | **Performing App Action "App Action4" for Destination "Soasta Demo Nexus Zirco Browser", operation "webClick".** Band: "Band 1" Track: "Track 1" Clip: "Clip for Soasta Demo Nexus Zirco Browser-21" Target: "Soasta Demo Nexus Zirco Browser" ▶Details: |
| 25 | 23510 | Verbose | Transport: append | **App Action "App Action4" completed.** Band: "Band 1" Track: "Track 1" Clip: "Clip for Soasta Demo Nexus Zirco Browser-21" Target: "Soasta Demo Nexus Zirco Browser" ▶Details: |
| 26 | 23510 | Info | Validation: vstart | **Starting validation "verifyScreenshot".** Band: "Band 1" Track: "Track 1" Clip: "Clip for Soasta Demo Nexus Zirco Browser-21" Target: "Soasta Demo Nexus Zirco Browser" |
| 27 | 23841 | Verbose | Validation: vcpass | **Validation of response body passed.** Band: "Band 1" Track: "Track 1" Clip: "Clip for Soasta Demo Nexus Zirco Browser-21" Target: "Soasta Demo Nexus Zirco Browser" ▶Details: |
| 28 | 23842 | Info | Validation: vpass | **Validation verifyScreenshot passed.** Band: "Band 1" Track: "Track 1" Clip: "Clip for Soasta Demo Nexus Zirco Browser-21" Target: "Soasta Demo Nexus Zirco Browser" |
| 29 | 23842 | Info | Validation: vstart | **Starting validation "verifyWebHtmlSource".** Band: "Band 1" Track: "Track 1" Clip: "Clip for Soasta Demo Nexus Zirco Browser-21" Target: "Soasta Demo Nexus Zirco Browser" |
| 30 | 23845 | Verbose | Validation: vcpass | **Validation of response body passed.** Band: "Band 1" Track: "Track 1" Clip: "Clip for Soasta Demo Nexus Zirco Browser-21" Target: "Soasta Demo Nexus Zirco Browser" ▶Details: |
| 31 | 23846 | Info | Validation: vpass | **Validation verifyWebHtmlSource passed.** Band: "Band 1" Track: "Track 1" Clip: "Clip for Soasta Demo Nexus Zirco Browser-21" Target: "Soasta Demo Nexus Zirco Browser" |
| 32 | 23846 | Info | App Action: sent | **App Action completed.** Band: "Band 1" Track: "Track 1" Clip: "Clip for Soasta Demo Nexus Zirco Browser-21" Target: "Soasta Demo Nexus Zirco Browser" |

# Appendix I: Using TouchTestIDs in Your Project Source Code

TouchTestIDs (TTIDs) are provided as a means to make your tests more user-friendly and readable. For developers, SOASTA TouchTest also provides the ability to define explicit mobile app locators, known as TouchTestIDs (TTIDs) as an integral part of touch-testing. Once implemented, TouchTest gives preference to recording the TouchTestID as the default locator.

Minimally, using TouchTestIDs requires the following source code modifications for both iOS and Android projects:

1) An import statement in the view where TTIDs will be used (e.g. using the OS-specific syntax required).

2) In any given view, call `setTouchTestId` and assign a string parameter to each view that needs one.

Once TTIDs are added TouchTest gives preference to recording the TouchTestID as the default locator. The use of `touchTestId` in Locators is described below.

## *Adding TouchTestIDs to an Android (Native) App*

For Android developers, implementing TTIDs is a straightforward two-step process for each view where a TTID is desired. The use of `touchTestId` in Locators is described below.

1. Identify the source file where the view is initialized.
2. For each view that will include TouchTestIDs, initialize the view using:

   ```
   Import com.soasta.android.touchtest.TouchTestDriver;
   ```

3. Next, call the `setTouchTestId` method on each view that will be located.

   For example,

   ```
   TouchTestDriver.getInstance().setTouchTestId(aView, "aTouchTestId");
   ```

   The string parameter provided here (e.g. `aTouchTestId`) will be used to locate the element.

**Note:** There is no built in support for Conditional Compilation in Java. Users of Ant, maven, and other tools should still use the TouchTestId conditionally, in such a manner that guarantees it is not part of code that gets submitted to any store.

For example, you could write an Ant build.xml that runs a preprocessor before compile. A good Ant trick along similar lines can be found here.

## *Adding TouchTestIDs to an Android (Hybrid) App*

For HTML-based hybrid apps, as well as for web sites recorded with TouchTest Web for Android, TouchTest and TouchTest Web utilize the HTML `id` attribute as the TouchTest ID.

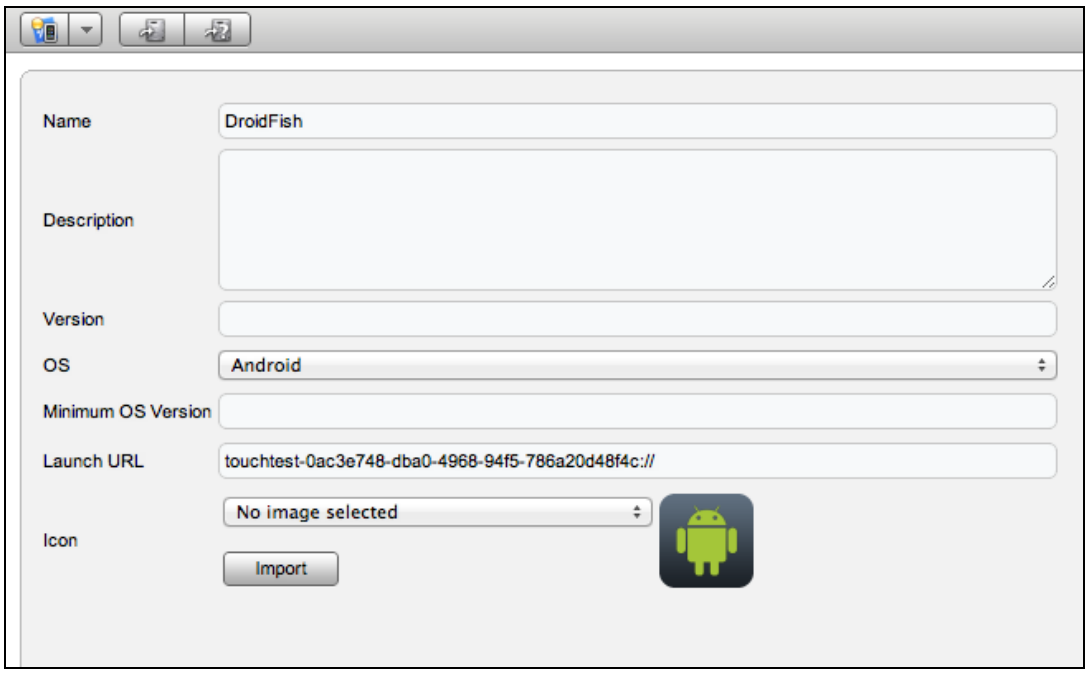
In the following HTML code, the `id="hello"` will be recorded as the TouchTestID:

```
<input type="submit" value="Hello!" onClick="buttonClicked()’’ id=‘‘hello">
```

# Appendix II: Adding a Mobile App Manually (Mobile Administrator)

In cases where manual project integration has been used, it will be necessary to manually add a mobile app to TouchTest®. These instructions apply equally to native and hybrid apps. As noted elsewhere, each TouchTest Lite user is also a Mobile Administrator for that Lite instance.

1. Select Central > Mobile Apps and then click New. The Mobile App form appears.
2. Enter the app name as it will appear in the drop-down for user selection. Generally, this will also be the Project name.
3. Optionally, enter a description and an app version number. Version number will generally match Project details.




4. Select Android as the OS. For this release, TouchTest supports Android 2.3.3 version or later.
5. Provide the launch URL. Users must add this application ID to their mobile app AndroidManifest.xml file. Without doing that, testing will not happen.

6. Optionally, import an app image for your mobile app to visually represent the correlation of TouchTest™ Agent with your app.

   Supported image types include JPEG, PNG, and GIF. Images can be pre-edited to the requisite 57 pixels wide by 57 pixels tall. Images that are not cropped will be shrunk to fit within the requisite dimensions.

7. Click Save to create this mobile app object in TouchTest

# Appendix III: Using Eclipse (Eclipse Developer Only)

This appendix preserves Eclipse development instructions that have been superceded by Gradle and presumes that the Android developer has a basic familiarity with the Eclipse development environment and has the following configuration:

- Eclipse (version 3.6 or later) with Android SDK version 10 or later installed along with the Eclipse plugin.
- Java 6 is installed.

### Installing the ADT for Mac OS X

- To download Eclipse for Mac OS X, click here. Most users download Eclipse Classic.
- For SDK install and instructions, click here.
  - Note that an SDK must be downloaded *and* enabled via the SDK Manager. For SDK Manager help, click here.

### Installing the ADT for Windows

- Download the ADT Bundle, which includes the Eclipse Android IDE, here. Once the page is loaded, click "Download for Other Platforms" at the bottom of the page and select the appropriate Windows version of the ADT Bundle to download (e.g. you'll need to know if your Windows system is 32-bit of 64-bit).
  - Also note, that if you don't already have a Java Development Kit installed on your Windows instance, you'll need to install it *before* you start the ADT Bundle install. Click here to get the JDK and then select either the Windows x86 or x64 version.
- With the JDK already installed, Windows users can proceed using the installation instructions here.

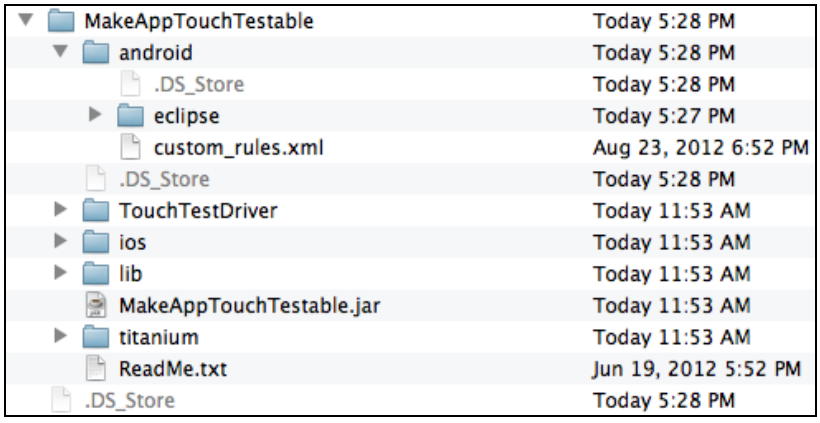
    - For help setting up a virtual device, click here.
- The Minimum Android Version supported for use with TouchTest™ is 2.3.3 (Gingerbread).
- Install the ADT Plugin for Eclipse here (or do so via the Eclipse user interface).
- The git executable file, for either Mac OS X or Windows, is used to retrieve the Android example projects used in this tutorial. GitHub can also be used. Windows git users should note that MakeAppTouchTestable will run from the Windows command prompt while Windows git versions (such as this one) offer both a Git Bash or a Windows Command Prompt
- The native app example, DroidFish, uses the Android NDK and so instructions are provided for installing it, although this is not a requirement of TouchTest itself, and users following only the Zirco Browser example can ignore NDK entirely.
- Unless you have a different Android native mobile app you'd like to use with this tutorial, install the Android NDK toolset into your Mac OS X or Windows Eclipse environment before proceeding. Additional configuration instructions are provided below at the appropriate time.

## *Downloading MakeAppTouchTestable Software (Eclipse Developer Only)*

As noted above, your app can be made "touch-testable" via the Gradle Plugin or by using the MATT utility.

1. For Eclipse or any other build system, download and unarchive the MakeAppTouchTestable Utility from the TouchTest Resources page.


**Note:** This archive contains the necessary drivers for both the manual and automatic methods (using Eclipse) described below.




### About the Eclipse Examples

Two Eclipse examples are presented in this tutorial in parallel, the first for native apps and the other for hybrid apps.

- To learn how to test a **native Android mobile app**, follow the DroidFish examples in this tutorial.

- For native Android apps, we'll set up the DroidFish sample app to use with a TouchTest test clip. We will use git to retrieve the necessary DroidFish source code. If you will not be using hybrid apps, skip all the sections pertaining to Zirco Browser.

To learn how to test a **hybrid Android mobile app**, follow the Zirco Browser example.

- For hybrid Android apps, we'll set up the Zirco Browser sample app to use with a TouchTest Hybrid test clip. If you will not be using native apps, skip all the sections pertaining to DroidFish.

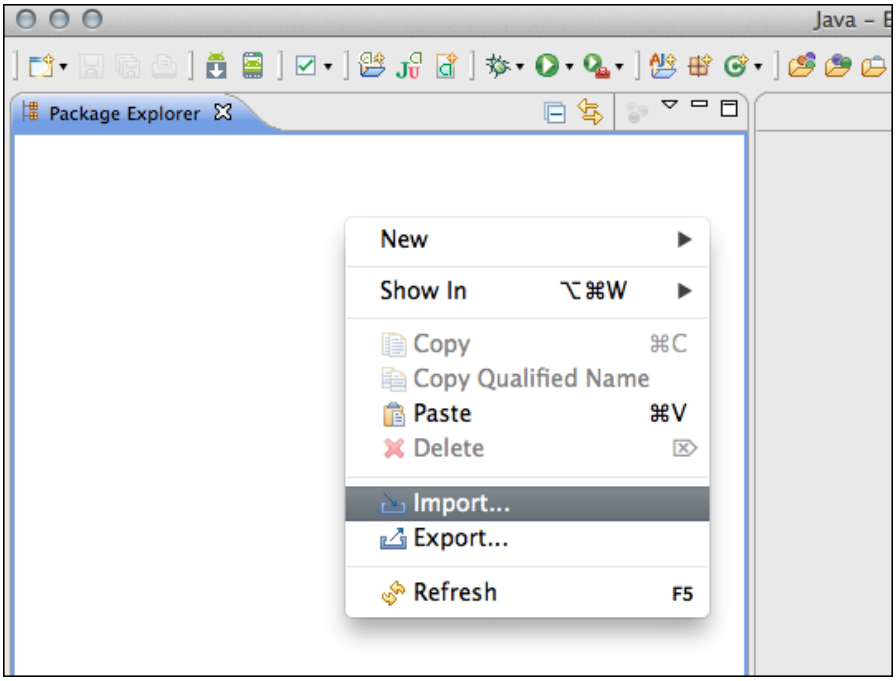
If you'd like to test both types, simply follow all of the instructions in this tutorial.

**Note:** Sections intended for developers only are marked so, although in some cases testers may also wish to follow them. Other sections are for general use (but may require assistance from someone else on your team, for example, when a developer or other tester is in charge of getting the app to test onto your mobile device so that you can focus on creating tests).

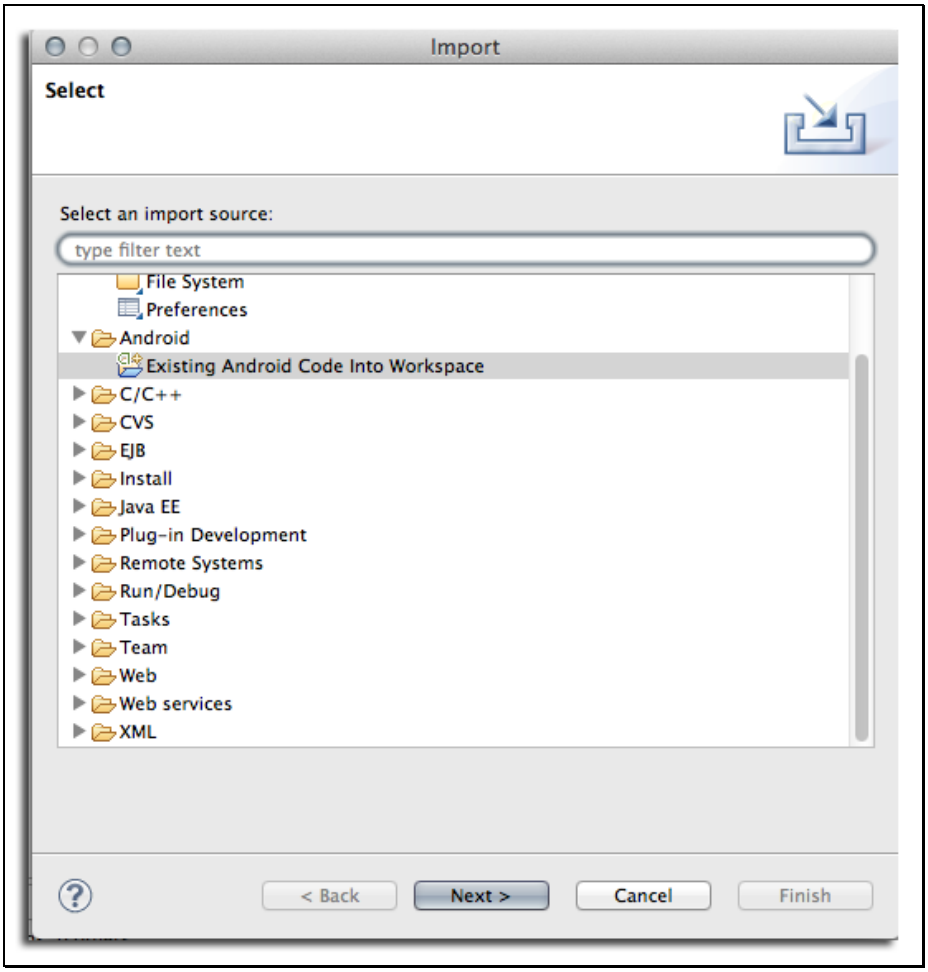
### Importing the DroidFish Project in Eclipse

Use the following instructions for both Mac OS X and Windows.

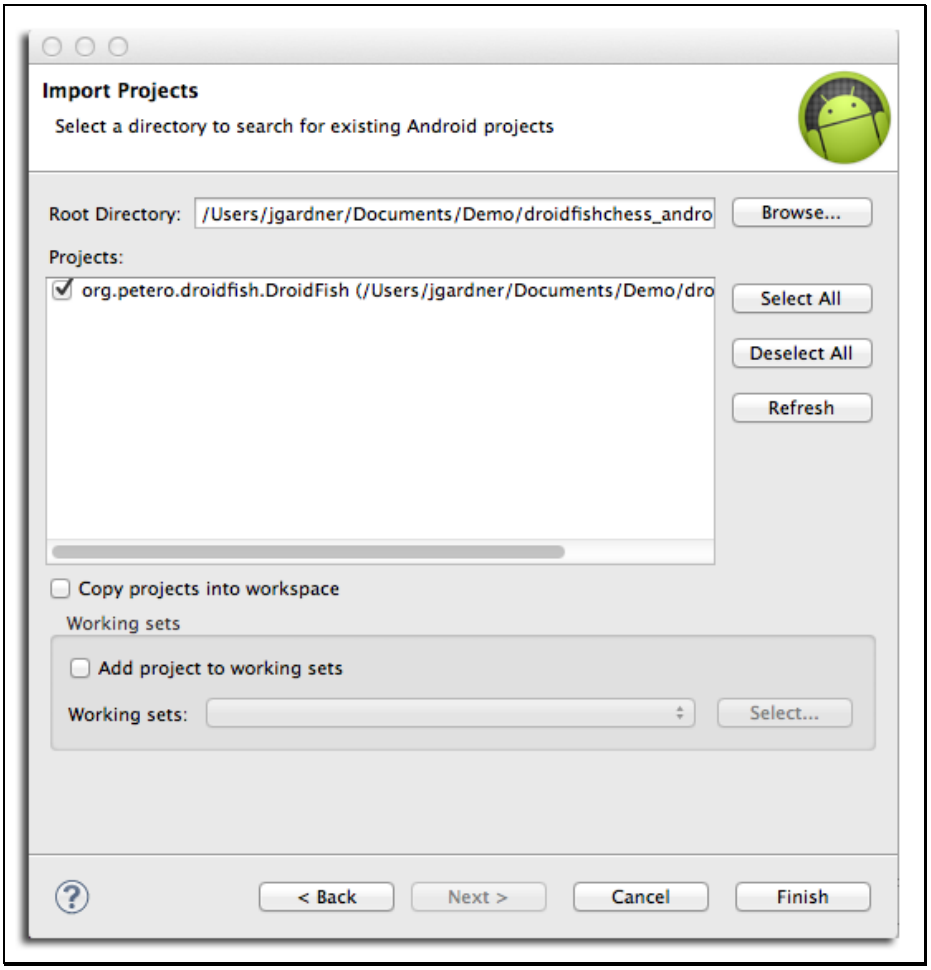
1. Launch Eclipse and right-click in the Package Explorer to choose Import.



2. In the Import box, choose General > Import Existing Android Code into Workspace.

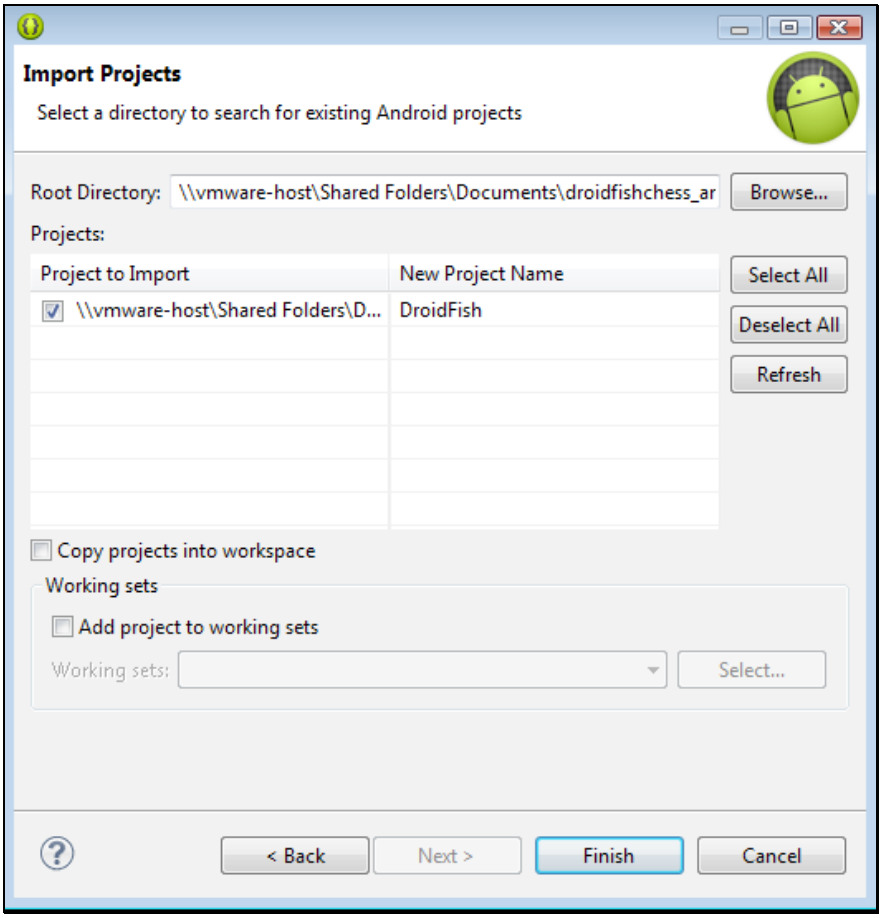


3. Select the root directory for the DroidFish (or other project).




The Mac OS X example is shown above, while the Windows example is shown below.

The environment is slightly different in appearance but includes the same details.

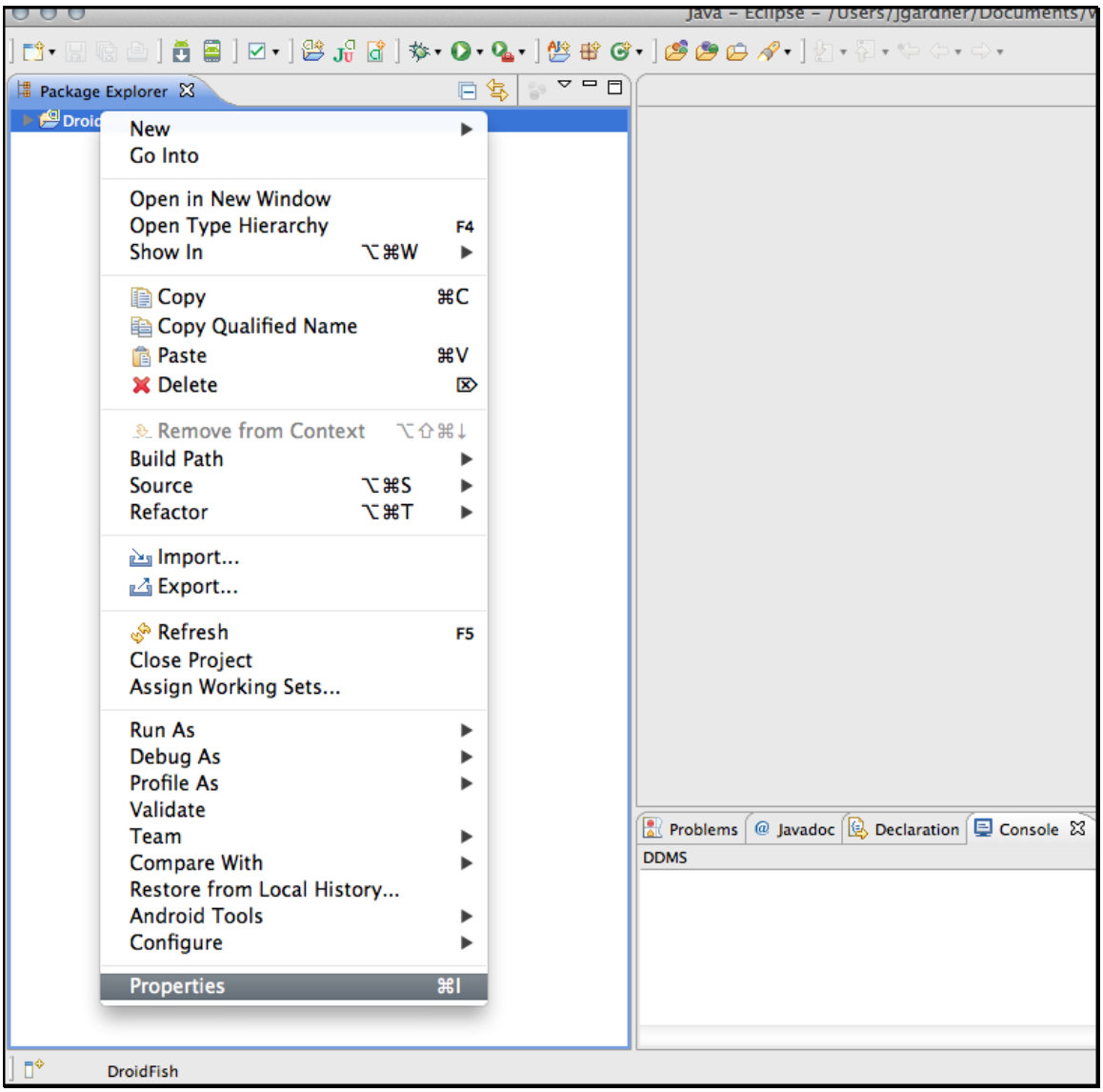



4. Click Finish to complete the import. The DroidFish project is added to the Package Explorer list in Eclipse.

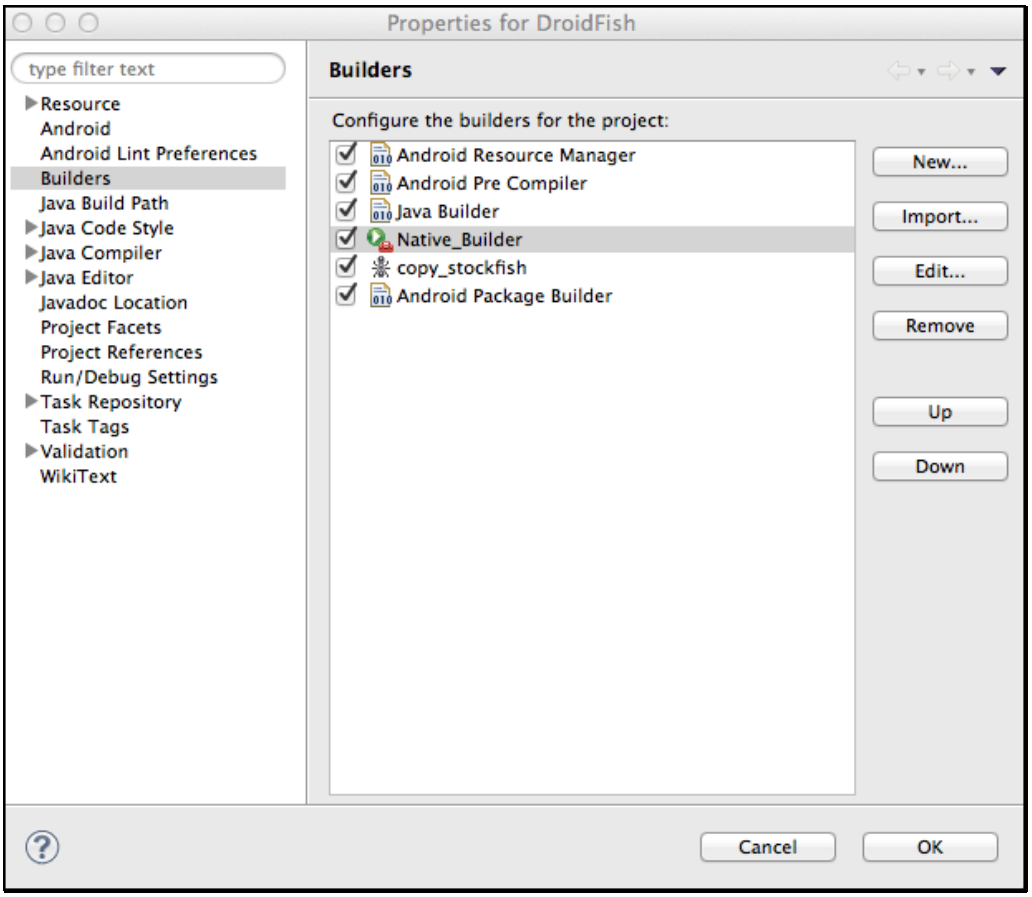
## Configuring the NDK Builder (Droidfish Only)

As noted above, Droidfish requires the NDK toolset in order to facilitate its use of native-code components. As a result, after our import we have a project that will not yet compile. Use the following steps to employ the NDK toolset.

1. Select the Droidfish project in the Package Explorer and then right-click to choose Properties.

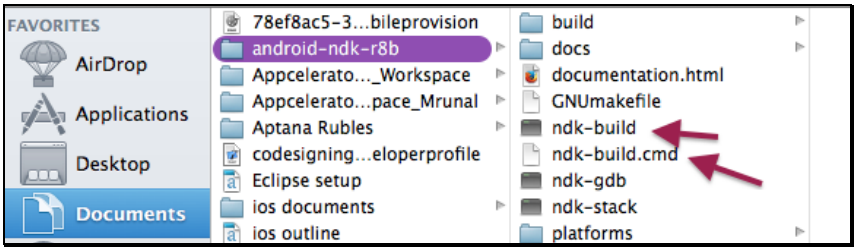


2. Select Builders in the Properties list on the left and then select Native_Builder.
3. With Native_Builder selected, click the Edit button.



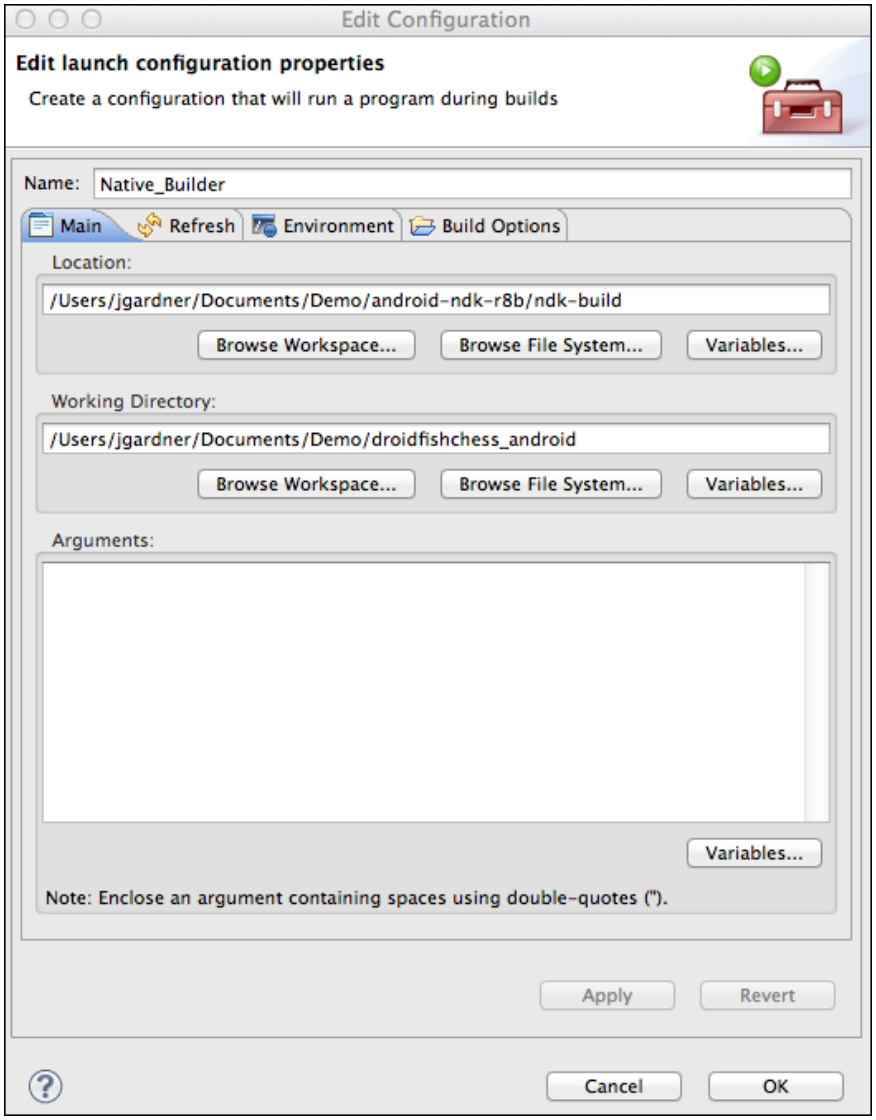

4. In the Edit Configuration box, Location field enter the path to the OS-specific NDK toolset's ndk-build executable.

Windows users will specify the location of the ndk-build.cmd file, which is found in the same location as the Mac version (e.g. in the download folder as shown below).




In the Mac OS X example below, this is `"/Users/<user>/Documents/Demo/android-ndk-r8b/ndk-build`." For Windows users, the path will be the same with the .cmd

extension added.




5. In the Working Directory, specify the location of Droidfish (same as noted above).
6. Click OK to accept the new builder configuration.

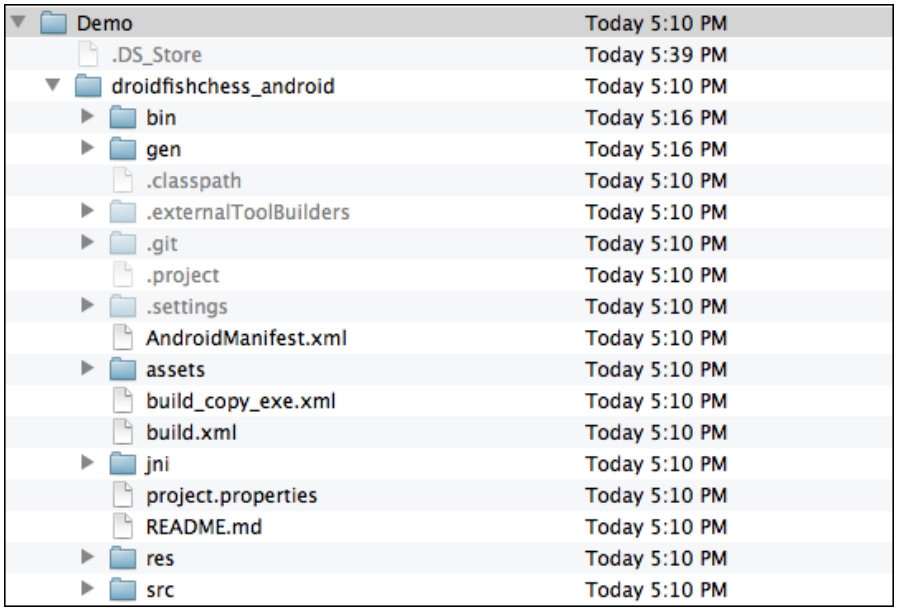
## *Setting Up the Droidfish Project in Eclipse (Native App Developer)*

Next, we will retrieve the native app, Droidfish, for use as an example app. After it is downloaded, we will then add TouchTest™ capabilities to it via the MakeAppTouchTestable utility.

You can use GitHub to download the DroidFish app's source code here, or use the git command from within either Terminal or the Windows Command Prompt using the following syntax (and presuming that git is in the path):

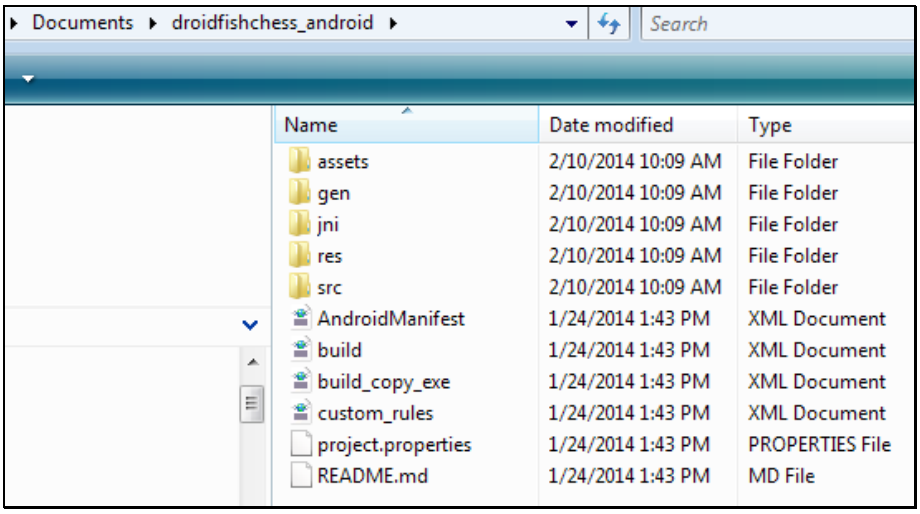
```
git clone https://github.com/elitecoder/droidfishchess_android
```

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




In our Mac OS X example, shown above, the DroidFish project is located in ~/Documents/Demo/droidfishchess_android.

In our Windows example, shown below, the DroidFish project is located in Users₩<user>₩Documents₩droidfishchess_android.

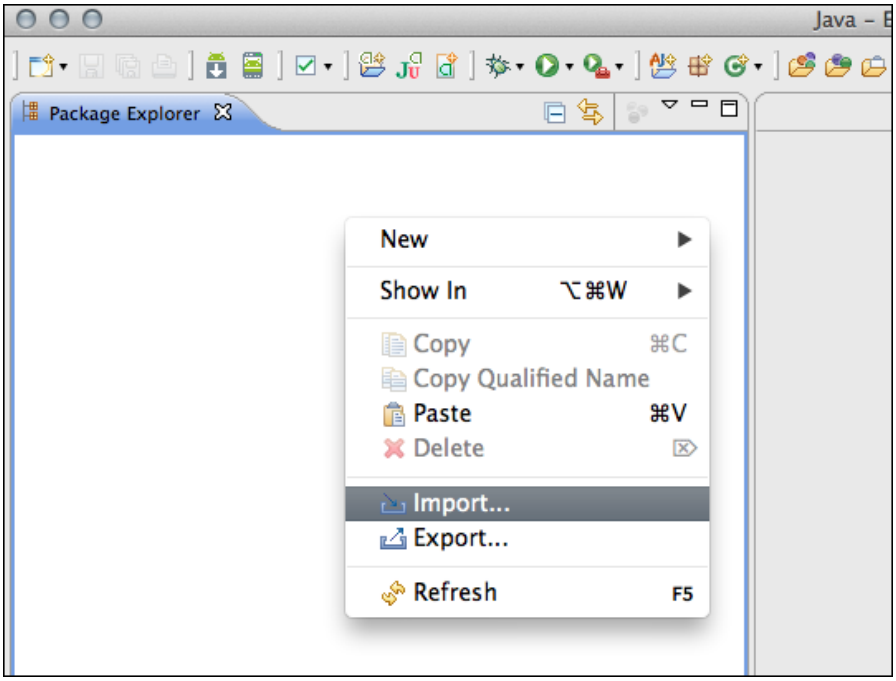



Note your own path for use in the next section as well as in the MakeAppTouchTestable section below.

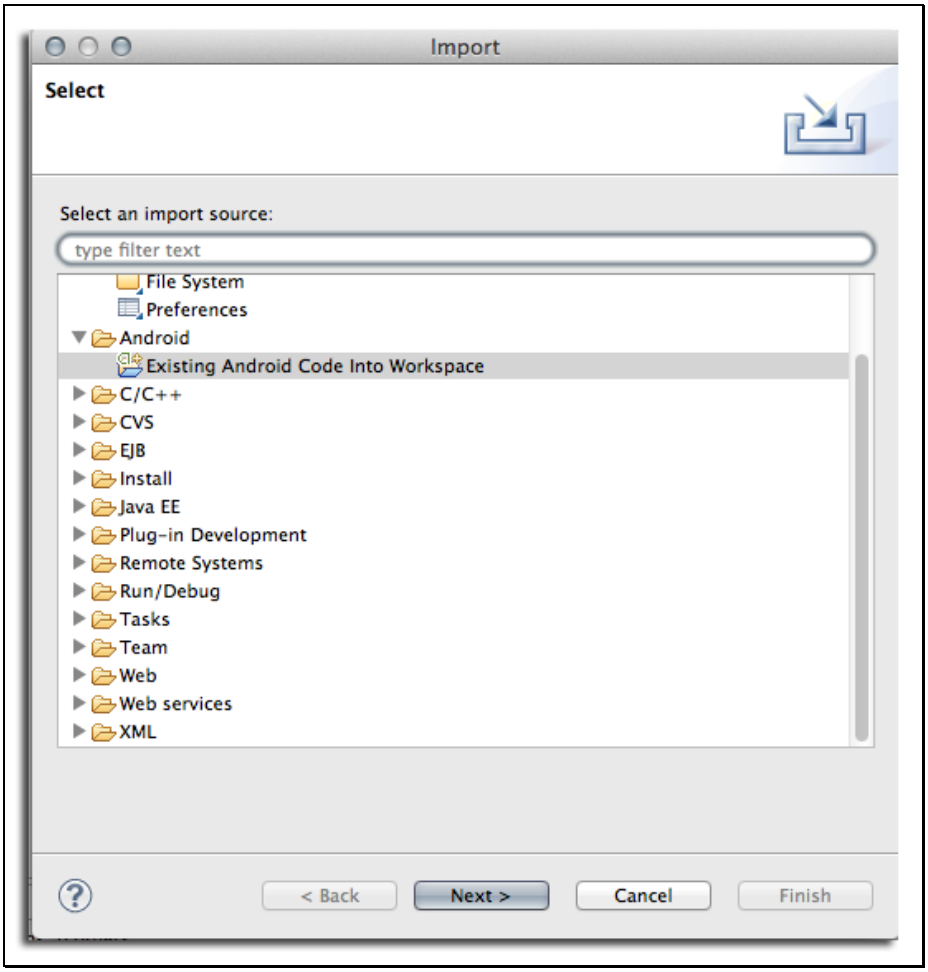
## Importing the DroidFish Project in Eclipse

Use the following instructions for both Mac OS X and Windows.

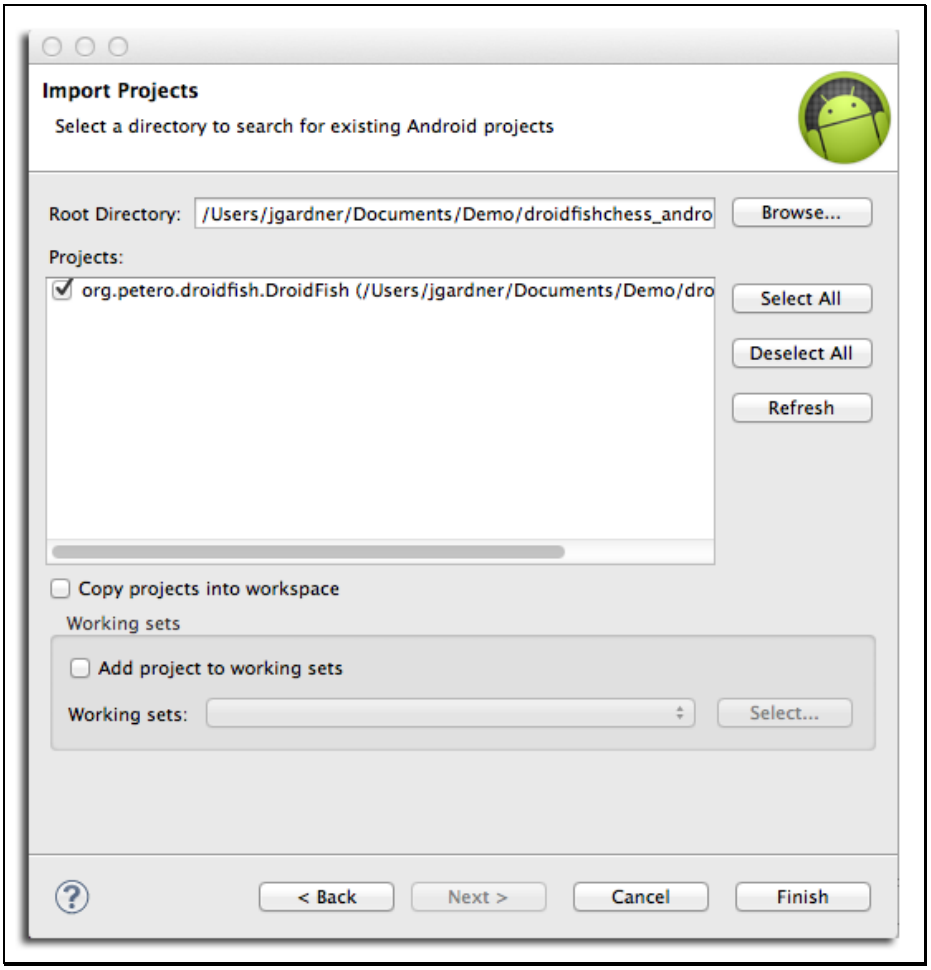
5. Launch Eclipse and right-click in the Package Explorer to choose Import.



6. In the Import box, choose General > Import Existing Android Code into Workspace.

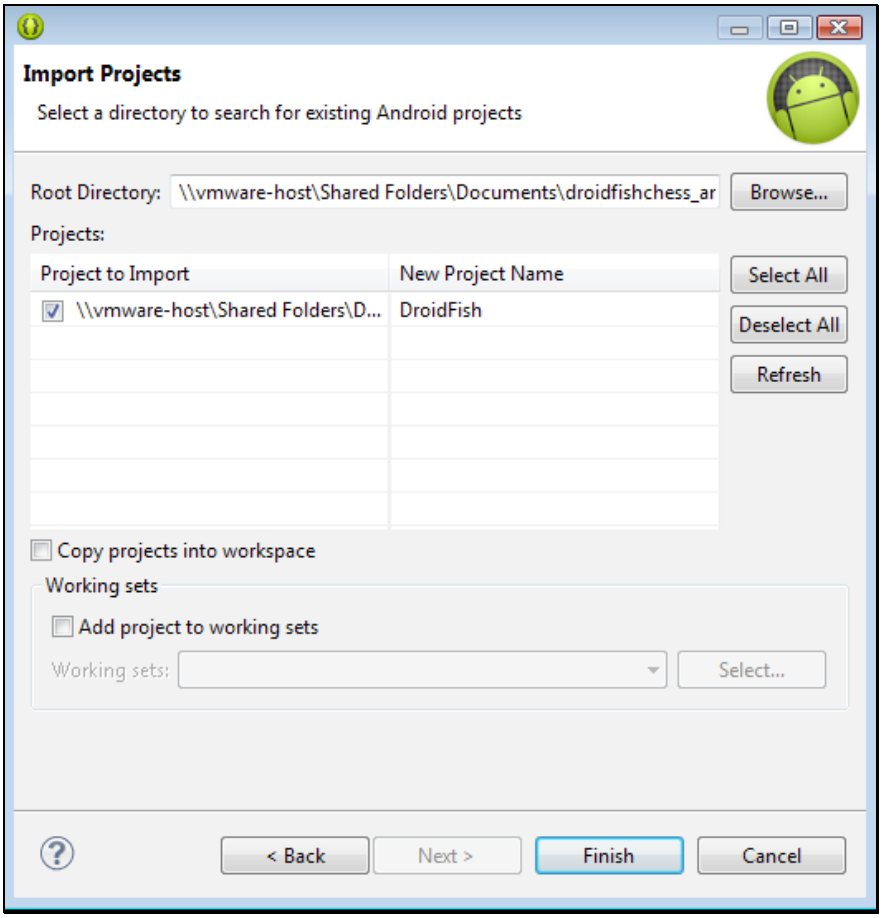


7. Select the root directory for the DroidFish (or other project).




The Mac OS X example is shown above, while the Windows example is shown below.

The environment is slightly different in appearance but includes the same details.

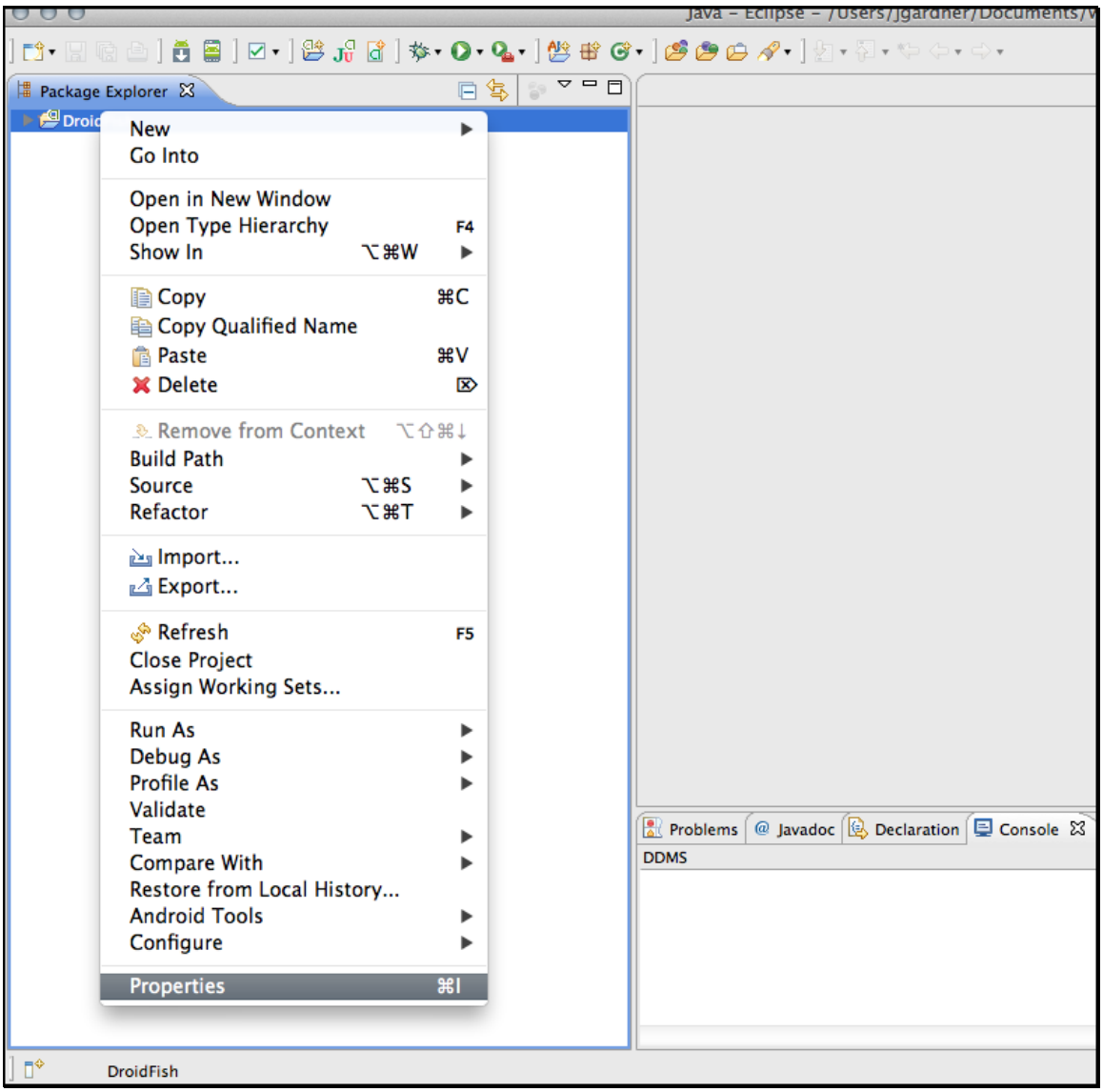



8. Click Finish to complete the import. The DroidFish project is added to the Package Explorer list in Eclipse.

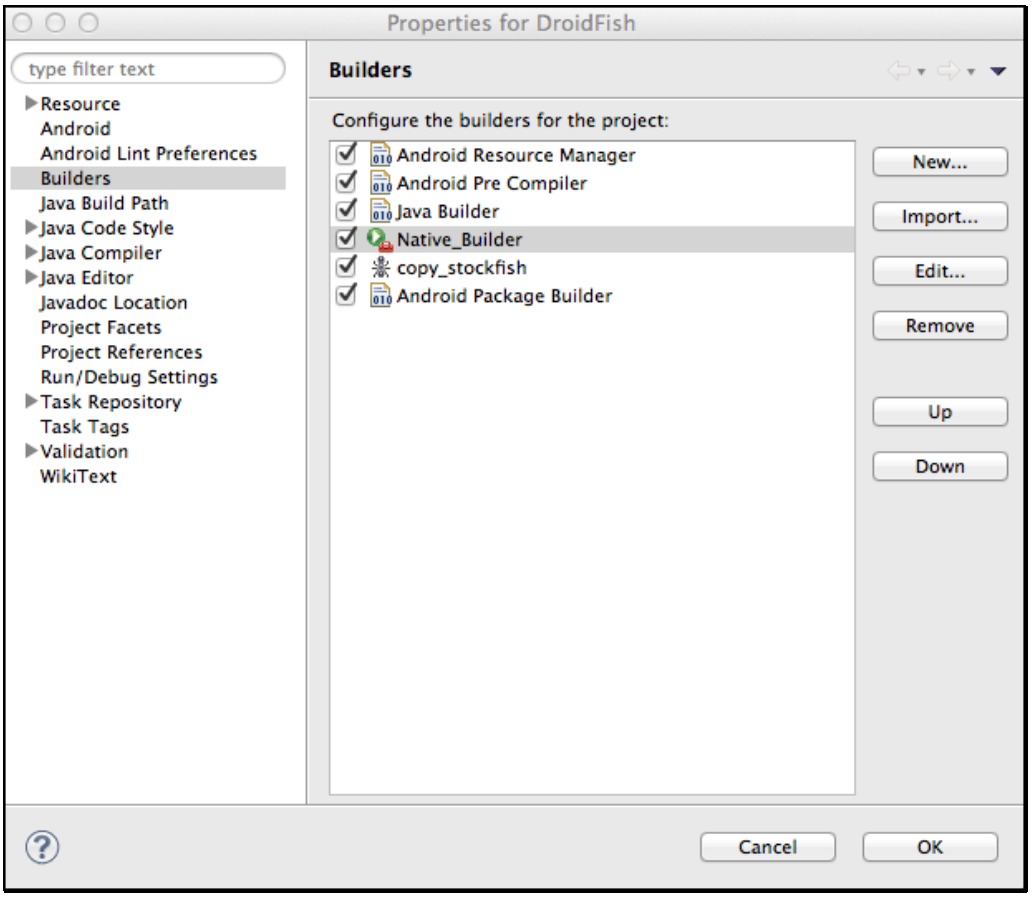
## Configuring the NDK Builder (Droidfish Only)

As noted above, Droidfish requires the NDK toolset in order to facilitate its use of native-code components. As a result, after our import we have a project that will not yet compile. Use the following steps to employ the NDK toolset.

7. Select the Droidfish project in the Package Explorer and then right-click to choose Properties.

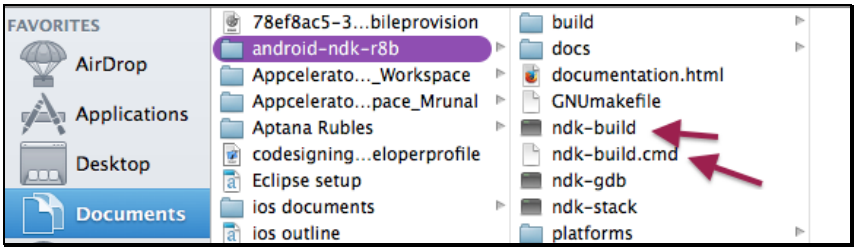


8. Select Builders in the Properties list on the left and then select Native_Builder.
9. With Native_Builder selected, click the Edit button.



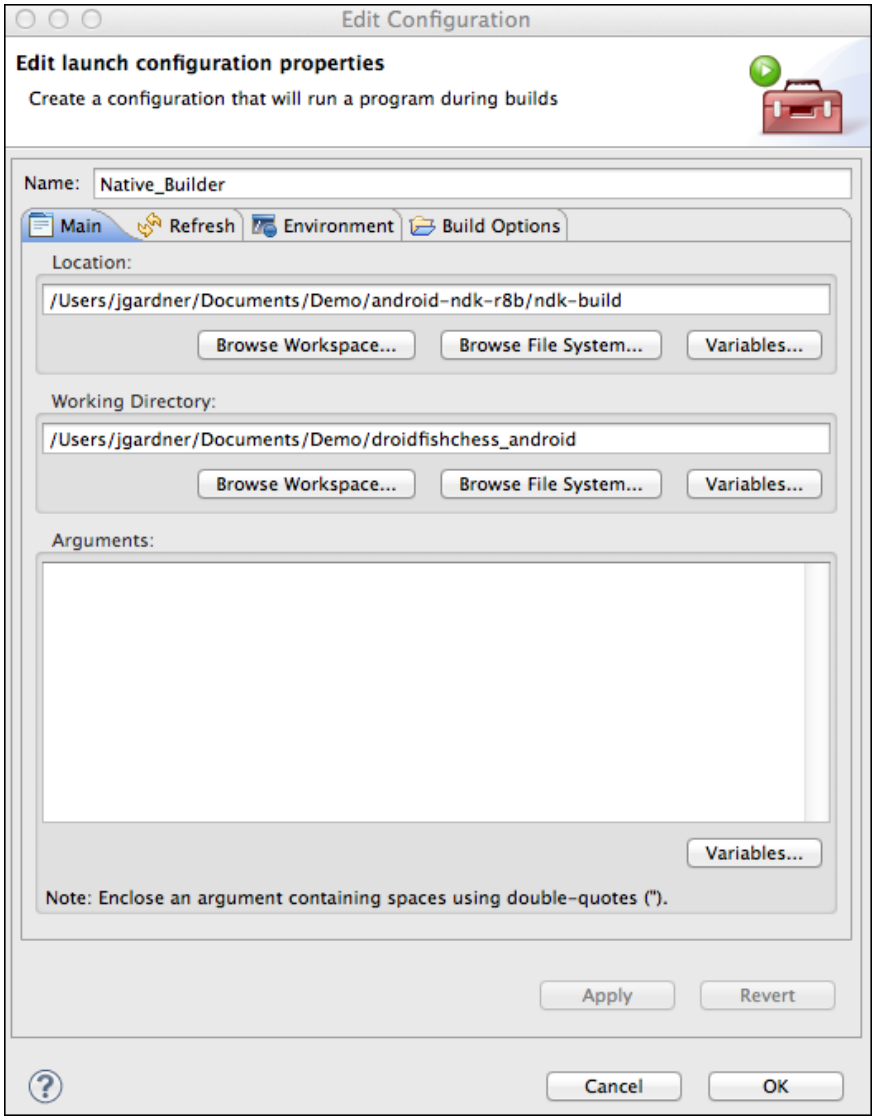

10. In the Edit Configuration box, Location field enter the path to the OS-specific NDK toolset's ndk-build executable.

Windows users will specify the location of the ndk-build.cmd file, which is found in the same location as the Mac version (e.g. in the download folder as shown below).




In the Mac OS X example below, this is `"/Users/<user>/Documents/Demo/android-ndk-r8b/ndk-build`." For Windows users, the path will be the same with the .cmd

extension added.




11. In the Working Directory, specify the location of Droidfish (same as noted above).
12. Click OK to accept the new builder configuration.

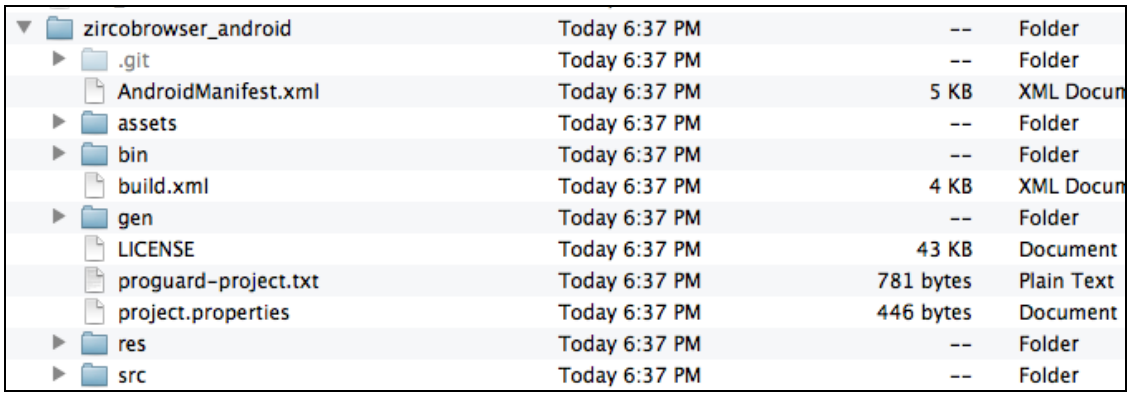
## Setting Up the Zirco Browser Project (Hybrid App Developer)

The Zirco Browser is presented here as an example of TouchTest Hybrid testing. In the following steps, we will retrieve the hybrid app example app, Zirco Browser, and then use the MakeAppTouchTestable utility to add TouchTest™ Hybrid capabilities to it. After which, the app will be deployed to an Android device.

1. On the command line, change to the folder where you'd like the source code to live. Use git (or GitHub) to download the Zirco Browser app's source:

   ```
   git clone https://github.com/elitecoder/zircobrowser_android
   ```

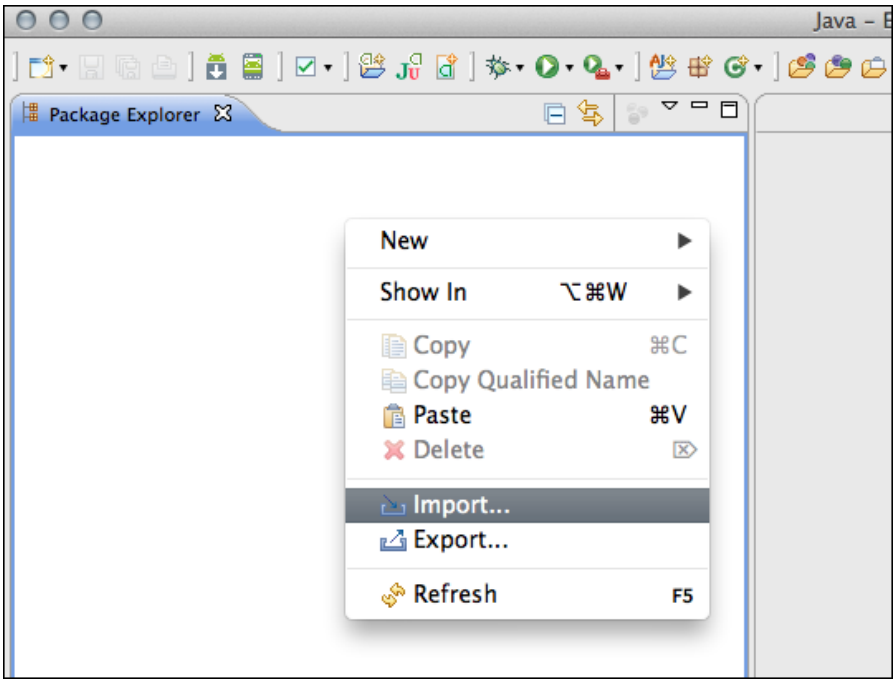
2. Once downloaded, inspect the project's components prior to running the MATT utility on it (if you are going to use MATT's apk parameter to instrument the APK file you can still do this inspection although the project itself will not be modified using that method). The unarchived project folders are shown below.



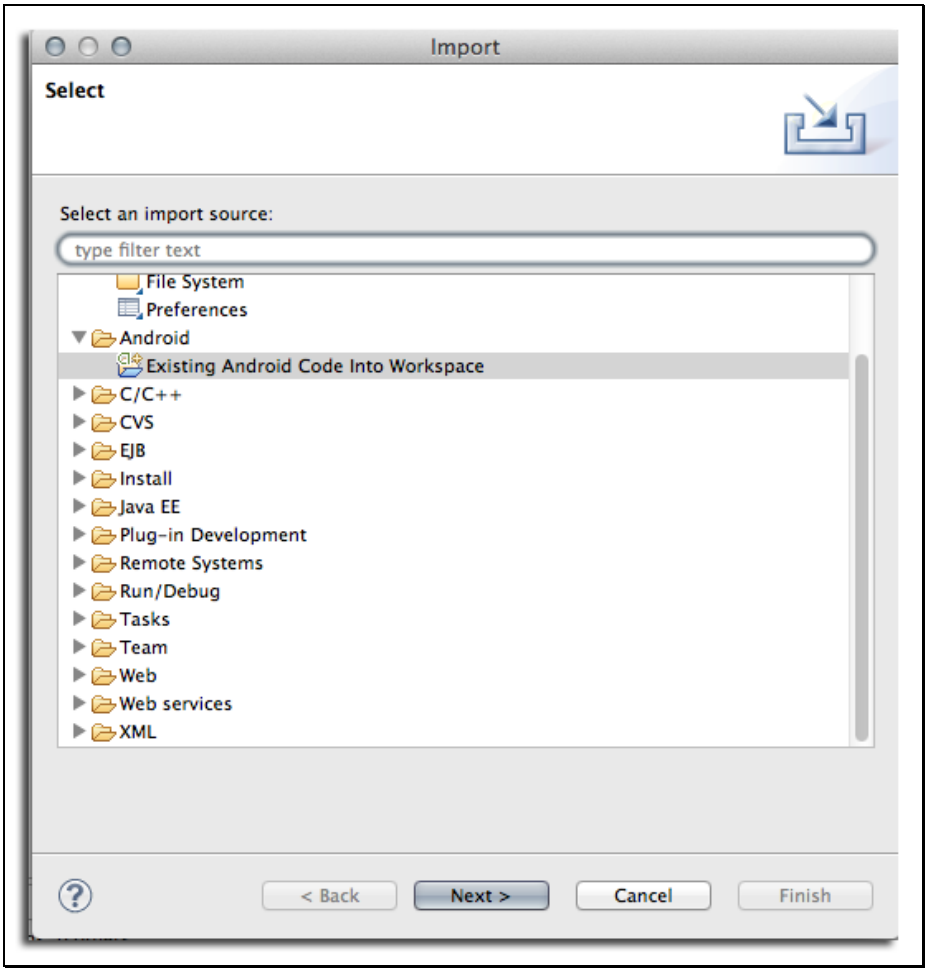

In our Mac OS X example, shown above, the Zirco Browser project is located in ~/Documents/Demo/zircobrowser_android. If your project is in Windows, use the syntax appropriate for the context. In either case, note your path for use in the next section as well as in the MakeAppTouchTestable section below.

## Importing the Zirco Browser Project

1. Launch Eclipse and right-click in the Package Explorer to choose Import.

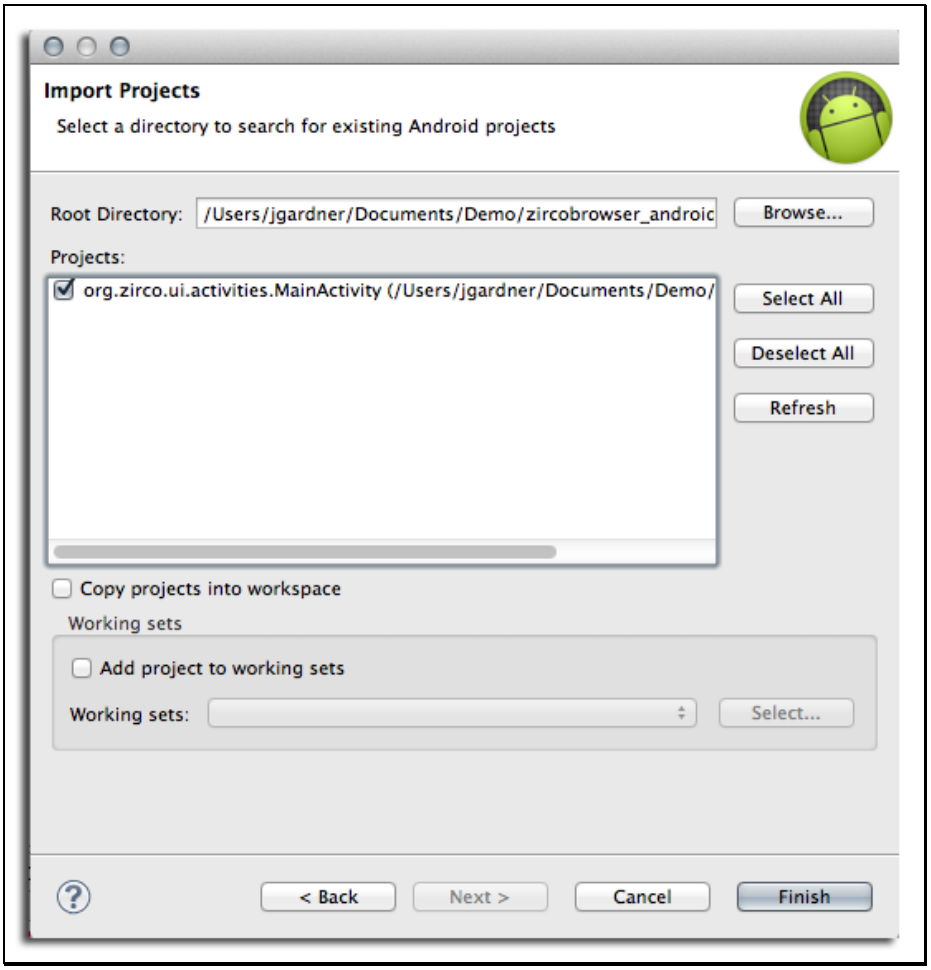


3. In the Import box, choose General > Import Existing Android Code into Workspace.

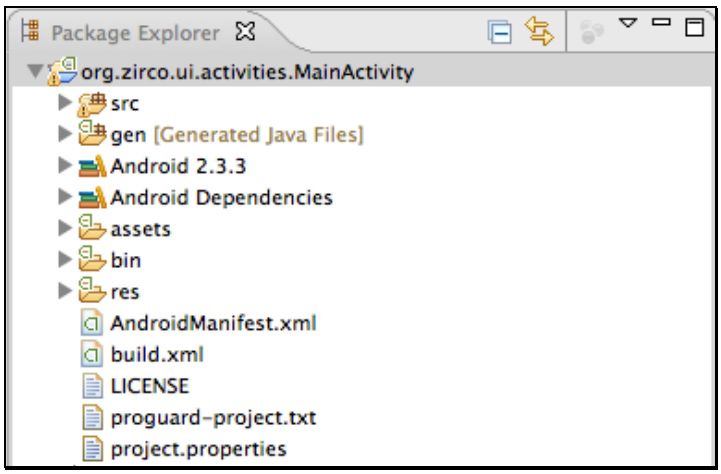



4. Click Next.

5. Select the root directory for the Zirco Browser (or other project).



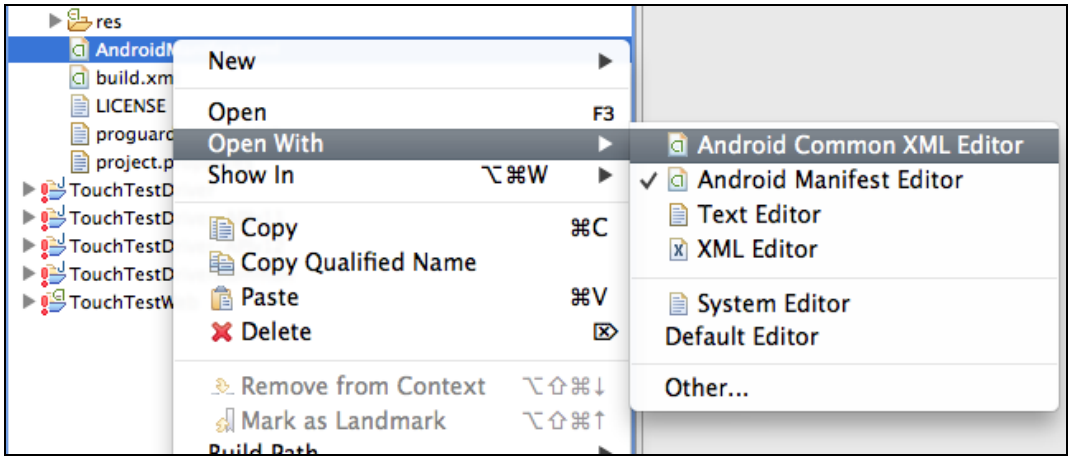

6. Click Finish to complete the import. The zirco-browser project is added to the Package Explorer list in Eclipse. The project begins to compile.



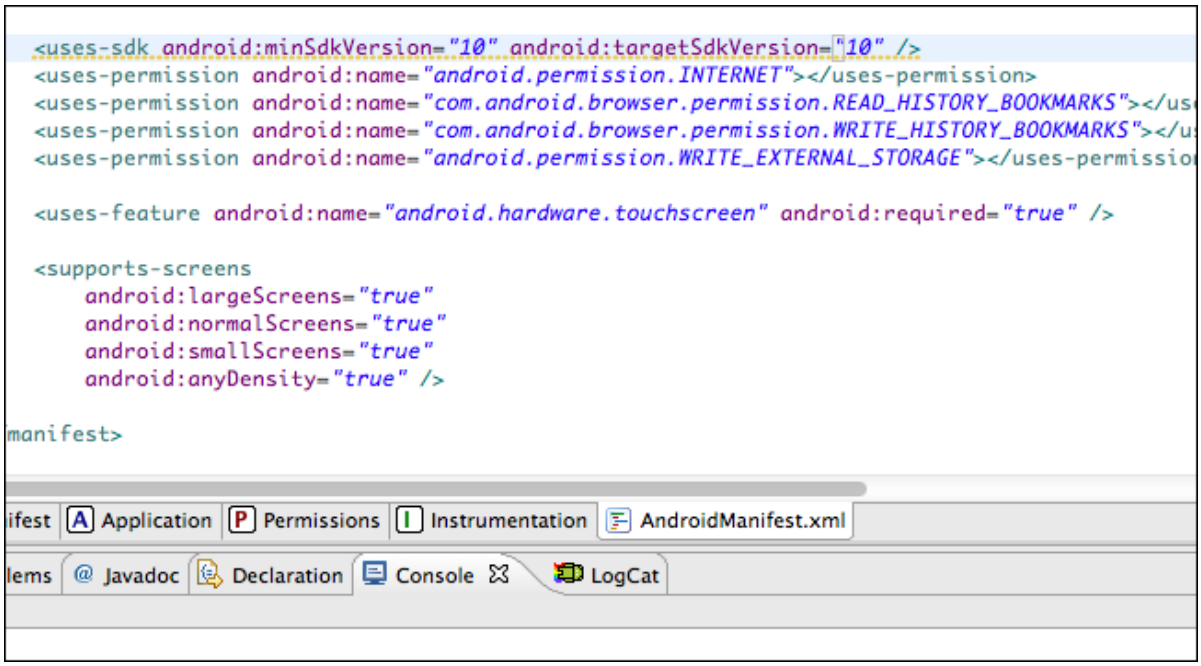

You can use "Run As..." to install the application to your Android device.

2. Locate and open the AndroidManifest.xml file with the Android Common XML Editor and find the following line:

```
<uses-sdk android:minSdkVersion="7" android:targetSdkVersion="8" />
```




Change both values to 10 as shown below. If it is already "10" leave it as is.




3. Close the project and exit Eclipse.

## *Using the MakeAppTouchTestable Utility (Developer Only)*

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

**Note:** The TouchTest user specified to run the MakeAppTouchTestable utility must be a user with Mobile Device Administrator rights. TouchTest Lite users have admin rights for the given device on their own instance.

### 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 the DroidFish APK TouchTestable (Native Developer Only)

This section presumes that the APK file has already been compiled. Do so at this time (without applying the MATT command) to proceed using the following steps. The Java memory parameter is used prior to MATT.

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 DroidFish APK using your own modified version of the MakeAppTouchTestable command below

   For Mac OS X:

```
java -Xmx1g -jar MakeAppTouchTestable.jar -apk <Android APK>  -androidsdk
<Android SDK Path> -url <TouchTest URL> -username <TouchTest user name> -password
<TouchTest password>
```

   For Windows:

```
C:\Users\<user>\MakeAppTouchTestable>java -Xmx1g -jar MakeAppTouchTestable.jar
-apk <Android APK>  -androidsdk <Android SDK Path> -url <TouchTest URL> -username
<TouchTest user name> -password <TouchTest password>
```

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

Where:

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

  `~/Documents/Demo/droidfishchess_android`

- `<Android SDK Path>` is the path to the Android SDK used to compile the APK file.
- `<TouchTest URL>` is the TouchTest Lite or TouchTest instance in use.
- Using the Java parameter `-Xmx1g` prior to jar allocates the necessary memory to complete the operation. Otherwise, the MATT command may throw an exception.

Here is a complete Mac OS X example:

```
java -Xmx1g -jar MakeAppTouchTestable.jar -apk
~/Documents/Demo/droidfishchess_android/bin/Droidfish.apk -androidsdk
~/Development/android-sdk-macosx -url http://10.0.1.9/concerto -username
SOASTA_DOC -password secret
```

Here is a complete Windows example:

```
C:\Users\<user>\MakeAppTouchTestable>java -Xmx1g -jar MakeAppTouchTestable.jar -
apk C:\Documents\Demo\droidfishchess_android\bin\Droidfish.apk  -androidsdk
C:\Development\android-sdk-macosx -url http://10.0.1.9/concerto -username
SOASTA_DOC -password secret -Xmx1g
```

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

  This argument is used in the TouchTest 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 ‘‘Droidfish://key1=value1&key2=value2&key3=value3’’
```

MakeAppTouchTestable will configure your project, and create a new Mobile App object in the TouchTest 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 "Droidfish" representing your Application "Droidfish" has
been created in TouchTest 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://
```

## Making the DroidFish Project TouchTestable (Native Developer Only)

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 DroidFish project using your own modified version of the MakeAppTouchTestable command below

   For Mac OS X:

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

   For Windows:

   ```
   C:\Users\<user>\MakeAppTouchTestable>sh
   MakeAppTouchTestable/bin/MakeAppTouchTestable
   -project <Android project directory>  -url <TouchTest URL> -username <TouchTest
   user name> -password <TouchTest password>
   ```

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

Where:

- `<Android project file>` is the path to the root folder of your project. As we noted above, our example path under Mac OS X was:

  ```
  ~/Documents/Demo/droidfishchess_android
  ```

- `<TouchTest URL>` is the TouchTest Lite or TouchTest instance in use. In the example below, we show a TouchTest Lite instance on a LAN with an Apple router, 10.0.1.9, but your TouchTest server may have a domain or IP address prior to /concerto.

Here is a complete Mac OS X example:

```
sh MakeAppTouchTestable/bin/MakeAppTouchTestable -project
~/Documents/Demo/droidfishchess_android
-url http://10.0.1.9/concerto -username SOASTA_DOC -password secret
```

Here is a complete Windows example:

C:₩Users₩<user>₩MakeAppTouchTestable>sh MakeAppTouchTestable/bin/MakeAppTouchTestable -project `C:\Documents\Demo\droidfishchess_android -url http://10.0.1.9/concerto -username SOASTA_DOC -password secret`

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

  This argument is used in the TouchTest repository to represent your mobile app and in the compiled app. For Eclipse projects, this setting originates in the AndroidManifest.xml. The launch URL in the compiled app and in the TouchTest, Mobile App, launch URL field must match for testing to occur.

For example,

```
-launchURL ‘‘Droidfish://key1=value1&key2=value2&key3=value3’’
```

MakeAppTouchTestable will configure your project, and create a new Mobile App object in the TouchTest 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 "Droidfish" representing your Application "Droidfish" has
been created in TouchTest 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://
```

**IMPORTANT:** In the next section, we inspect the project changes and then re-run the sample app using the steps in Install to the Device. Minimally, you must re-run the app after using the MakeAppTouchTestable utility.

## Making the Zirco Browser APK TouchTestable (Hybrid Developer Only)

This section presumes that the APK file has already been compiled. 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 Zirco Browser APK using your own modified version of the MakeAppTouchTestable command below:

   For Mac OS X:

```
sh MakeAppTouchTestable/bin/MakeAppTouchTestable -apk <Android APK>  -
androidsdk <Android SDK Path> -url <TouchTest URL> -username <TouchTest
user name> -password <TouchTest password>
```

   For Windows:

```
C:\Users\<user>\MakeAppTouchTestable>sh
MakeAppTouchTestable/bin/MakeAppTouchTestable
-apk <Android APK>  -androidsdk <Android SDK Path> -url <TouchTest URL> -username
<TouchTest user name> -password <TouchTest password>
```

**TIP:** Copy the above command into a text or other scratch file to begin making your own command.

where:

- `<Android APK>` is the path to the APK. For example:

  `~/Documents/Demo/zircobrowser_android/bin/zirco-browser.apk`

- `<Android SDK Path>` is the path to the Android SDK used to compile the APK file.
- `<TouchTest URL>` is the TouchTest Lite or TouchTest server that you use. So, the domain or IP address prior to the "/concerto/touchtest" string is what we mean by TouchTest URL. Here is a complete Mac OS X example:

```
sh MakeAppTouchTestable/bin/MakeAppTouchTestable -apk
~/Documents/Demo/zircobrowser_android/bin/zirco-browser.apk -url
http://10.0.1.9/concerto -username SOASTA_DOC -password secret
```

Here is a complete Windows example:

```
C:\Users\<user>\MakeAppTouchTestable>sh
MakeAppTouchTestable/bin/MakeAppTouchTestable
-apk C:\Documents\Demo\zircobrowser_android\bin\zirco-browser.apk -androidsdk
C:\Development\android-sdk-macosx -url http://10.0.1.9/concerto -username
SOASTA_DOC -password secret
```

- Advanced users can also manually specify an additional `launchurl flag`, using the URL syntax shown below.

  For example, you can impose the following convention, including optional arguments:

  ```
  -launchURL ‘‘ZircoBrowser://key1=value1&key2=value2&key3=value3’’
  ```

  The launch URL is used by TouchTest to open your mobile app via the corresponding device agent. The launch URL in the compiled app and in TouchTest's Central > Mobile App, Launch URL field must match for testing to succeed.

3. Once ready, run the modified command in Terminal from the MakeAppTouchTestable folder. When you do so, MakeAppTouchTestable will configure your APK, and create a new Mobile App object in the TouchTest 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 representing your Application "Zirco Browser" has been
   created in TouchTest 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://
   ```

MakeAppTouchTestable will configure your project, and create a new Mobile App object in the TouchTest server repository. The Mobile App object created will have the auto-created URL Scheme in its Launch URL field. You will see a message similar to the following:

```
Mobile App Object "Zirco Browser" representing your Application "Zirco
Browser" has been created in TouchTest Repository.
```

## Making the Zirco Browser Project TouchTestable (Hybrid Developer Only)

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 Zirco Browser project using your own modified version of the MakeAppTouchTestable command below:

   For Mac OS X:

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

   For Windows:

   C:₩Users₩<user>₩MakeAppTouchTestable>sh MakeAppTouchTestable/bin/MakeAppTouchTestable -project `<Android project directory>  -url <TouchTest URL> -username <TouchTest user name> -password <TouchTest password>`

**TIP:** Copy the above command into a text or other scratch file to begin making your own command.

where:

- `<Android project file>` is the path to the root folder of your project. As we noted above, our example path was:

  ```
  ~/Documents/Demo/zircobrowser_android
  ```

- `<TouchTest URL>` is the TouchTest Lite or TouchTest server that you use. So, the domain or IP address prior to the "/concerto/touchtest" string is what we mean by TouchTest URL.

In the example below, we show an IP address that was assigned for a given TouchTest Lite instance but it could be any other TouchTest URL where you have rights.

Here is a complete Mac OS X example:

```
sh MakeAppTouchTestable/bin/MakeAppTouchTestable -project
~/Documents/Demo/zircobrowser_android -url http://10.0.1.9/concerto -
username SOASTA_DOC -password secret
```

Here is a complete Windows example:

```
C:\Users\<user>\MakeAppTouchTestable>java -jar MakeAppTouchTestable.jar -
project C:\Documents\Demo\zircobrowser_android -url
http://10.0.1.9/concerto -username SOASTA_DOC -password secret
```

- Advanced users can also manually specify an additional `launchurl` flag, using the URL syntax shown below.

  For example, you can impose the following convention, including optional arguments:

  ```
  -launchURL ‘‘ZircoBrowser://key1=value1&key2=value2&key3=value3’’
  ```

  The launch URL is used by TouchTest to open your mobile app via the corresponding device agent. For Eclipse projects, this setting originates in the AndroidManifest.xml. The launch URL in the compiled app and in TouchTest's Central > Mobile App, Launch URL field must match for testing to occur.

3. Once ready, run the modified command in Terminal from the MakeAppTouchTestable folder. When you do so, MakeAppTouchTestable will configure your project, and create a new Mobile App object in the TouchTest 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 representing your Application "Zirco Browser" has been
   created in TouchTest 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://
```

MakeAppTouchTestable will configure your project, and create a new Mobile App object in the TouchTest server repository. The Mobile App object created will have the auto-created URL Scheme in its Launch URL field. You will see a message similar to the following:

```
Mobile App Object "Zirco Browser" representing your Application "Zirco
Browser" has been created in TouchTest Repository.
```

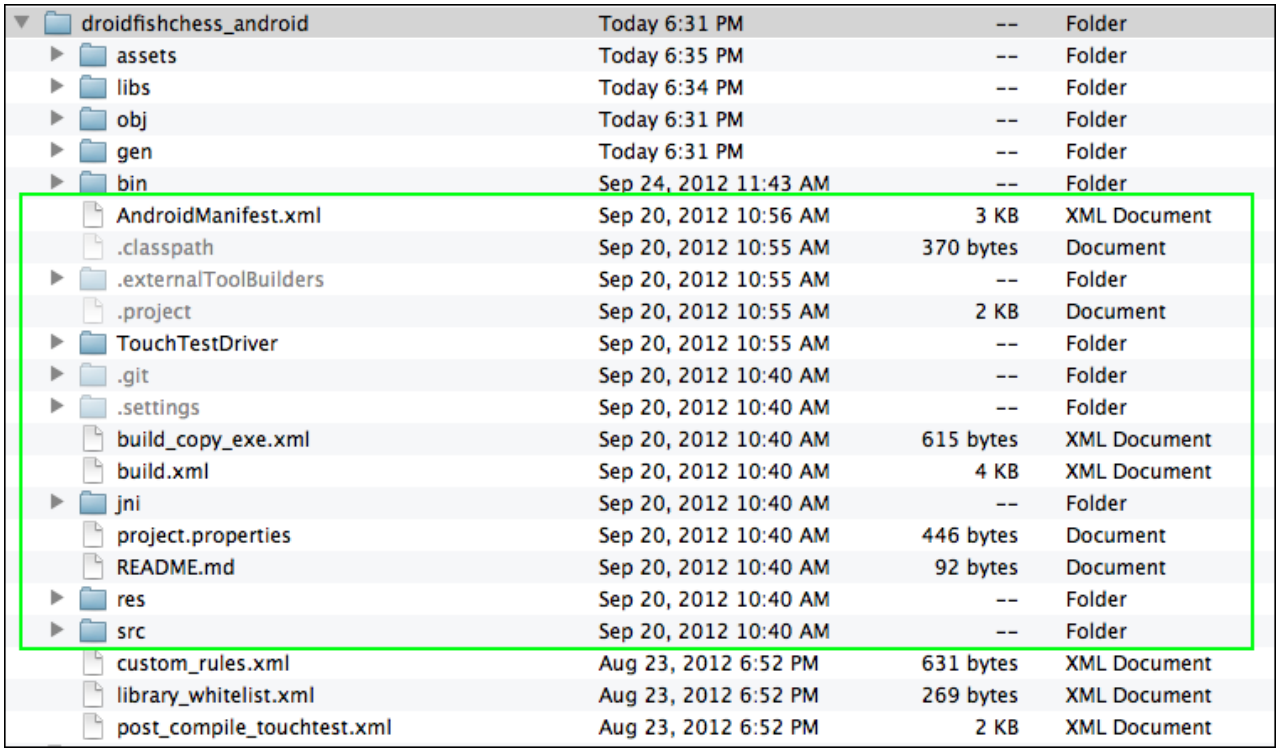
**IMPORTANT:** In the next section, we will inspect the project changes and then re-run the sample app using the steps in Install to the Device. Minimally, you must re-compile the APK after using the MakeAppTouchTestable utility.

## Inspecting a TouchTestable Project (Native or Hybrid using project method)

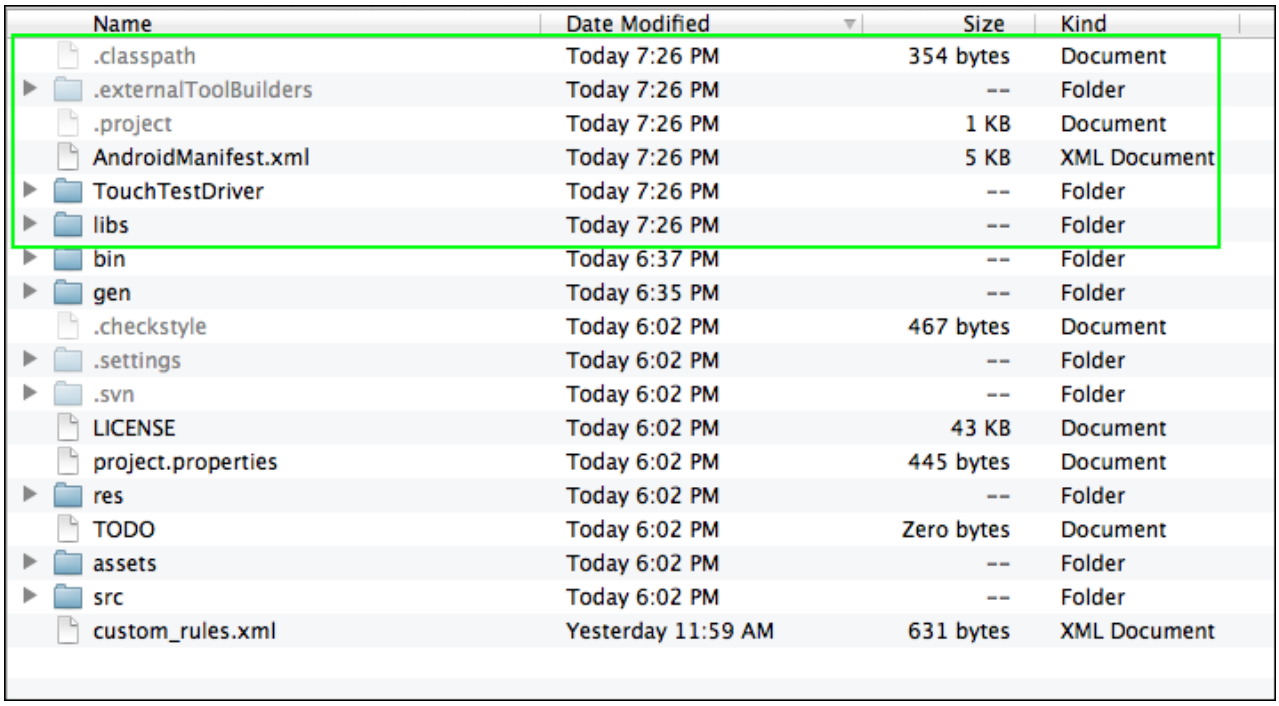
If we are using the MATT project parameter, we can visually inspect changes to the project in order to satisfy our curiosity, although this is not strictly necessary. If you used the MATT apk parameter skip ahead to the next section.

1. For either project, inspect the original source folder via Finder.

- The contents of the folder where the droidfishchess_android project was retrieved using git clone are shown below.



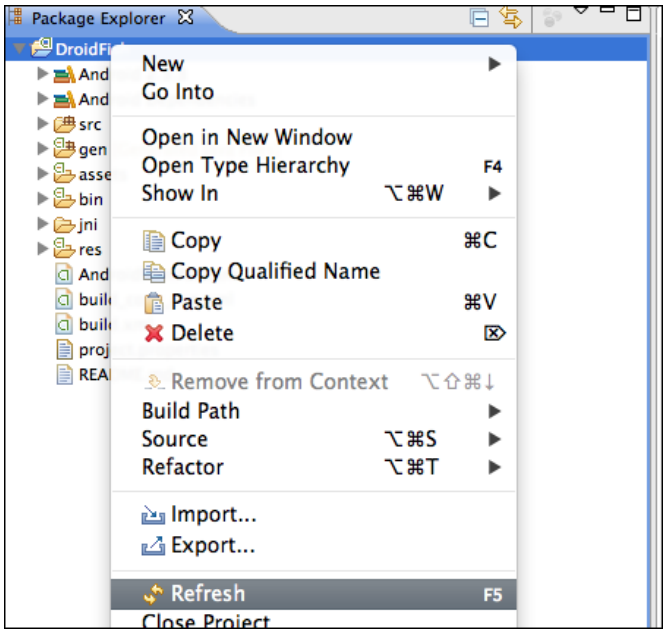
- The contents of the folder where the zircobrowser_android project was retrieved using git are shown below.



| Name | Date Modified | Size | Kind |
|---|---|---|---|
| .classpath | Today 7:26 PM | 354 bytes | Document |
| .externalToolBuilders | Today 7:26 PM | -- | Folder |
| .project | Today 7:26 PM | 1 KB | Document |
| AndroidManifest.xml | Today 7:26 PM | 5 KB | XML Document |
| TouchTestDriver | Today 7:26 PM | -- | Folder |
| libs | Today 7:26 PM | -- | Folder |
| bin | Today 6:37 PM | -- | Folder |
| gen | Today 6:35 PM | -- | Folder |
| .checkstyle | Today 6:02 PM | 467 bytes | Document |
| .settings | Today 6:02 PM | -- | Folder |
| .svn | Today 6:02 PM | -- | Folder |
| LICENSE | Today 6:02 PM | 43 KB | Document |
| project.properties | Today 6:02 PM | 445 bytes | Document |
| res | Today 6:02 PM | -- | Folder |
| TODO | Today 6:02 PM | Zero bytes | Document |
| assets | Today 6:02 PM | -- | Folder |
| src | Today 6:02 PM | -- | Folder |
| custom_rules.xml | Yesterday 11:59 AM | 631 bytes | XML Document |

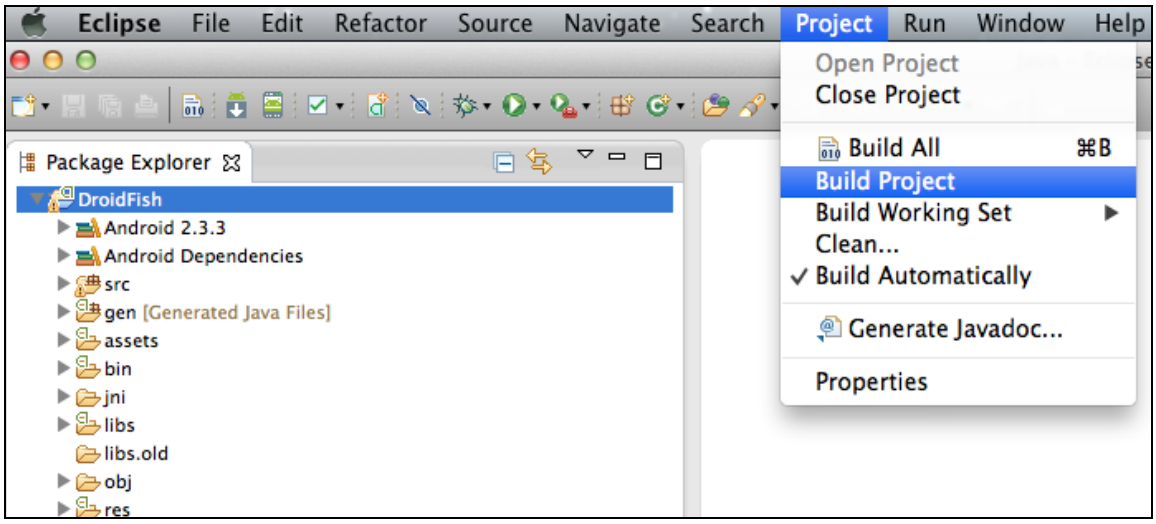
Note that in either project, the TouchTest Driver folder has been added.

2. In Eclipse, select the top-level project folder and choose Refresh.

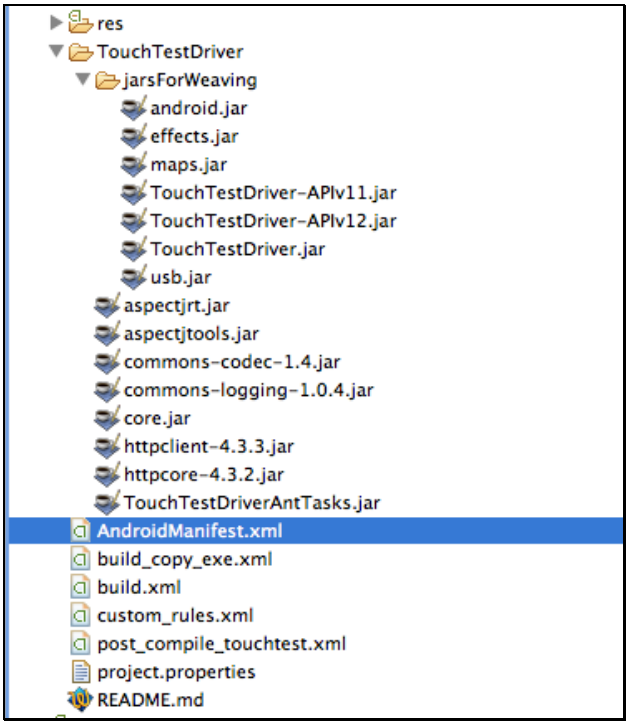


Note that the project includes the TouchTestDriver folder.

3. In Eclipse, select the Project menu, Build Project command.




4. Expand the TouchTestDriver folder and its subfolder (JarsForWeaving).




5. Open the AndroidManifest.xml file in XML mode (by clicking the AndroidManifest.xml tab in the workspace).

Note that the manifest now includes a new intent-filter section that includes the

launchURL data value in `android:scheme`.

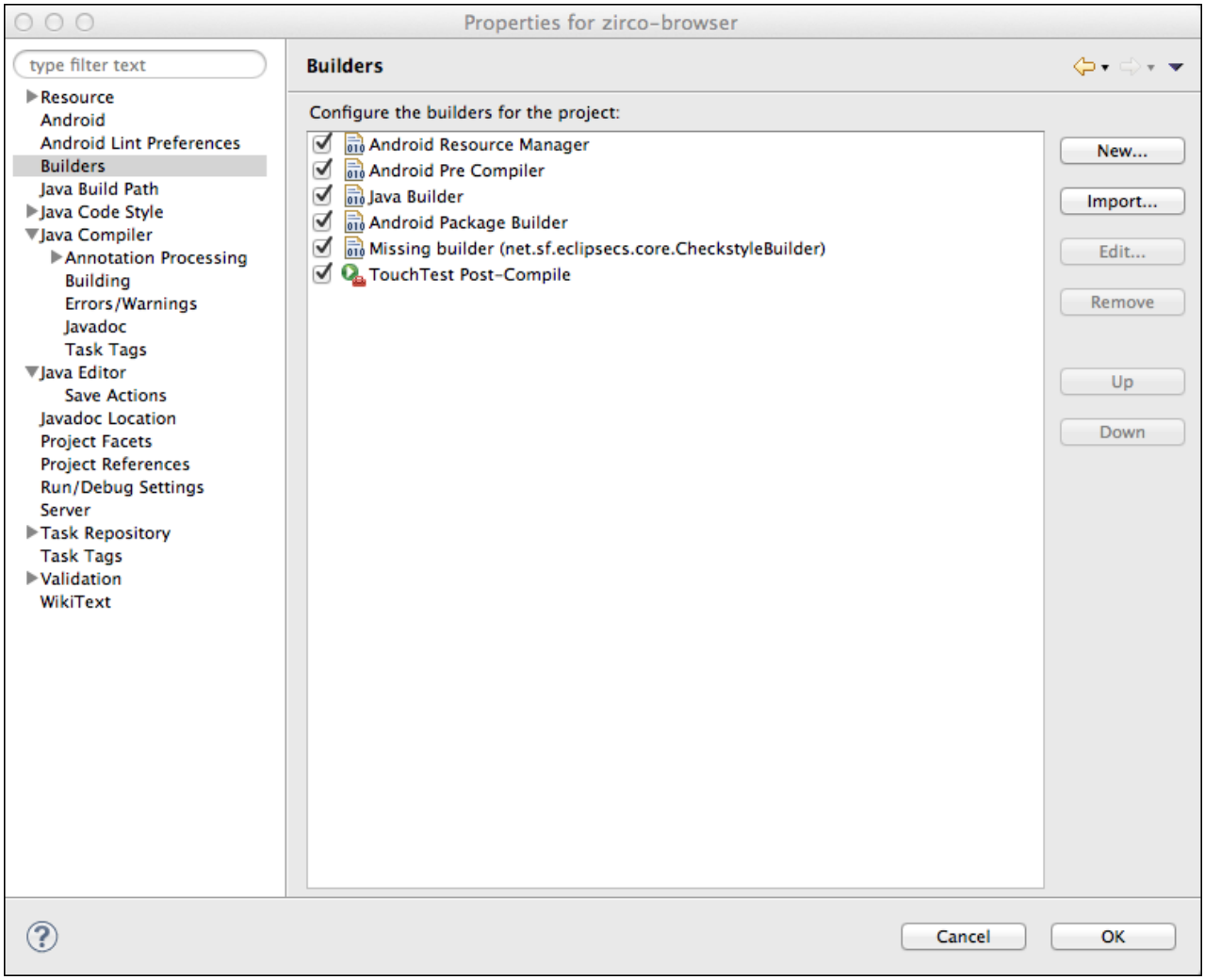
```
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>

                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
            <intent-filter>
                <action android:name="android.intent.action.VIEW"/>

                <category android:name="android.intent.category.DEFAULT"/>

                <data android:mimeType="application/x-chess-pgn"/>
            </intent-filter>
        <intent-filter>
                <data android:scheme="touchtest-0ac3e748-dba0-4968-94f5-786a20d48f4c"/>
                <action android:name="android.intent.action.VIEW"/>
                <category android:name="android.intent.category.BROWSABLE"/>
                <category android:name="android.intent.category.DEFAULT"/>
            </intent-filter>
        </activity>
```

6. Scroll down to the end of the AndroidManifest.xml.

Note the new service statement referencing TouchTest in `android:exported`.

```
<activity android:configChanges="orientation" android:label="@string/load_game_title" android:name=".activities.EditPGN
</activity>
<activity android:configChanges="orientation" android:label="@string/save_game_title" android:name=".activities.EditPGN
</activity>
<activity android:configChanges="orientation" android:label="@string/load_scid_game_title" android:name=".activities.Lo
</activity>
<activity android:label="@string/cpu_warning_title" android:name=".activities.CPUWarning">
</activity>
<service android:enabled="true" android:exported="false" android:name="com.soasta.android.touchtest.TouchTestService"/>
plication>

st>
```

7. In Eclipse's Project Explorer, right-click the ZircoBrowser folder, and then in the Properties box, select Builders. Ensure that the TouchTest Post-Compile box is checked.



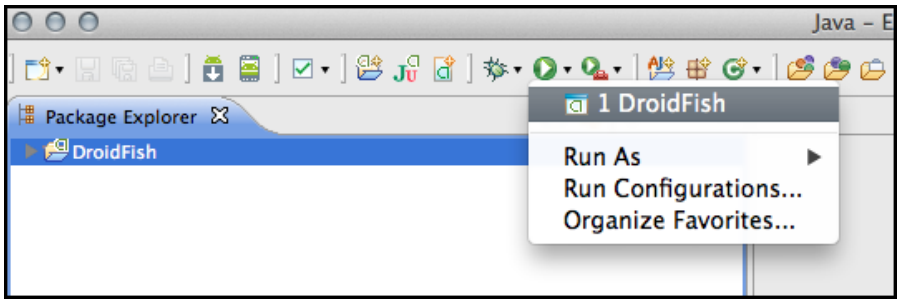

8. Now that the project is verified TouchTestable, send it to the device or simulator a second time using Run.

## Install using Eclipse

You can install an Android app to a device using Eclipse or adb (Gradle does this for the user for free but this is not the case in Eclipse or with other platforms). After completing install using whatever method, your are done with this Appendix, and you

should resume the main tutorial beginning with the section, *Inspecting the Mobile App in TouchTest® (Native or Hybrid.*

1. Connect the Android Device to the desktop client running Eclipse via USB. You can also use a simulator. A physical device must have the following set:
    - The stock browser on the device should support launch of native apps
    - The Developer Options, USB Debugging box should be enabled
    - The Security, Unknown sources box should be enabled
2. Click the Run Droidfish button on the toolbar to build the project and push it to the Android Device.




- If the Android device is connected, and no Android Virtual Device (AVD) is running, the app is installed to the device.
- If more than one device or emulator combination is available, then a selection box appears for you to choose
- If no device is connected, the Android Virtual Device (AVD) will run and the Droidfish app will be installed to it instead. The AVD must be using SDK 2.3.3 or later.

When all of the conditions and steps above are completed, the app is pushed onto the Android Device. The Eclipse Console will indicate success and the app will launch on the device.

```
[2012-09-19 20:05:21 - DroidFish] ------------------------------
[2012-09-19 20:05:21 - DroidFish] Android Launch!
[2012-09-19 20:05:21 - DroidFish] adb is running normally.
[2012-09-19 20:05:21 - DroidFish] Performing org.petero.droidfish.DroidFish activity
launch
[2012-09-19 20:05:21 - DroidFish] Automatic Target Mode: using device '015d15b4da23f411'
[2012-09-19 20:05:21 - DroidFish] Uploading DroidFish.apk onto device '015d15b4da23f411'
[2012-09-19 20:05:23 - DroidFish] Installing DroidFish.apk...
[2012-09-19 20:05:26 - DroidFish] Success!
[2012-09-19 20:05:26 - DroidFish] Starting activity org.petero.droidfish.DroidFish on
device 015d15b4da23f411
[2012-09-19 20:05:26 - DroidFish] ActivityManager: Starting: Intent {
act=android.intent.action.MAIN cat=[android.intent.category.LAUNCHER]
cmp=org.petero.droidfish/.DroidFish }
```

## Install from the Command Line using adb

1. Connect the Android Device to the desktop client or start the Simulator(s) before using adb. A physical device must have the following set:
    - The Developer Options, USB Debugging box should be enabled
    - The Security, Unknown sources box should be enabled
2. From the command line, execute the adb command using your own paths:

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

When all of the conditions and steps above are completed, the app is pushed onto the Android Device. The command line will indicate success and the app will launch on the device:

```
873 KB/s (2083295 bytes in 2.330s)
       pkg: /data/local/tmp/DroidFish-debug_TouchTest.apk
Success
```

SOASTA, Inc.

444 Castro St.

Mountain View, CA 94041

866.344.8766

http://www.soasta.com