SOASTA

# TouchTest™ OpenGL Tutorial

## Table of Contents

SOASTA

## About This Tutorial

Unlike other testing solutions that only test results displayed or actions taken at the user-interface layer, CloudTest® Mobile can validate tests using app internal values, and conditionally wait for internal application state changes. For mobile apps that do not show internal app states, such as those based on the OpenGL game engine, Cocos2d, additional techniques can be employed in the Xcode project to expose them. These techniques are described in the following sections.

This tutorial uses iLabyrinth, an OpenGL open-source application, as an example of exposing values where those values are not already exposed. The techniques used here can also be used to expose any application internal state to which the developer has rights. They are equally applicable to analogous iOS apps.

For example, in a Calendar application, one might choose not only to validate at the user-interface level but using internal values that are exposed using this tutorial.

The SOASTA TouchTest OpenGL Tutorial is intended as a "deep" sequel to the SOASTA TouchTest Tutorial, which provides an introduction to basic CloudTest Mobile concepts and best practices.

- Additional material pertinent to developers may be found in the SOASTA TouchTest Developer Guide, as well as in the SOASTA TouchTest Advanced Tutorial, although neither are necessary to begin this tutorial.

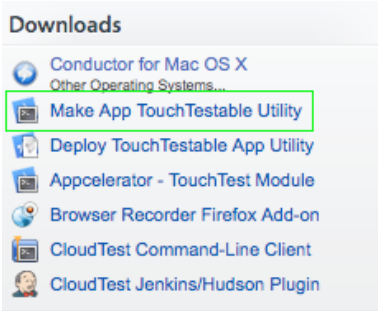
# Prerequisites

This tutorial requires the latest version of the MakeAppTouchTestable utility, available from the CloudTest Welcome page, as well as an Xcode project to extend using the techniques enclosed.

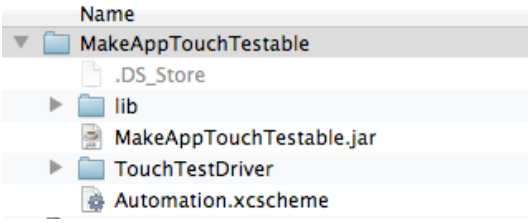
## MakeApp TouchTestable Utility

The following prerequisites should be performed by a developer or Mobile Device Administrator:

1. Download and unarchive the CloudTest *Make App TouchTestable Utility* from the Central > Welcome page, Downloads section.




This archive contains the following:




**Note:** It is not necessary to open any of the contents after unarchiving.

- MakeAppTouchTestable.jar utility is the script that will make the necessary project modifications and create a mobile app in CloudTest®
- The TouchTestDriver folder. The contents of this folder will be automatically copied to your project

## iLabyrinth Source Project

While this tutorial uses iLabyrinth, the another OpenGl (or other apps with undisclosed app internal states) can be substituted.

1. Download and unarchive the Xcode Project for the iLabyrinth application. This can be done using either of the following methods:
   - From Terminal, use:

     ```
     git clone https://github.com/ud7/iLabyrinth.git
     ```
   - In the browser, download the archive from:

     https://github.com/ud7/iLabyrinth

## Using the MakeAppTouchTestable Utility

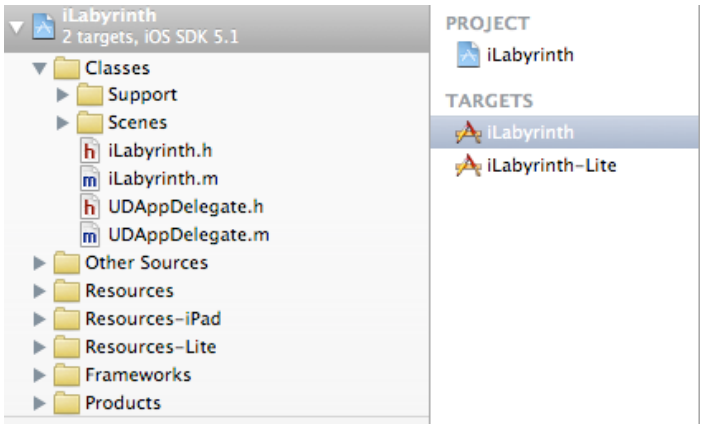
TouchTest™ includes the MakeAppTouchTestable, which will automatically add the necessary components to an existing Xcode project to deploy TouchTest™, and additionally, the utility will also create the Mobile App entry in CloudTest® .

**TIP:** If you'd like to perform these Xcode project customization steps manually, refer to the "Appendix: Manually Adding TouchTest™ to the Xcode Project" included at the end of this tutorial.

**SOASTA highly recommends that you create a new Xcode target for use with TouchTest.** This allows you to easily build two versions of your app: a testable version that is linked with TouchTest Driver, and a production-ready version that does not include any TouchTest™ functionality.

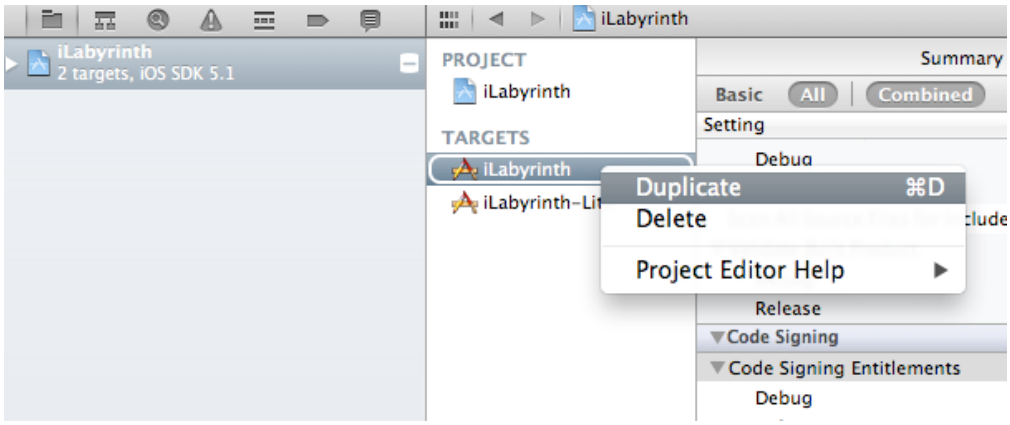
1. Open the iLabyrinth Xcode project file.

The project's components prior to running the utility are shown below.




Note that out-of-the-box there is one target, *iLabyrinth*, defined in this project. We will now duplicate that target for use with TouchTest.

2. Right-click the *iLabyrinth* target and choose “Duplicate”.




3. Xcode will create a new target called *iLabyrinth copy*.
4. Open a Terminal window and navigate to the location where the ZIP was unarchived.

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

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

   where:

   - `<Xcode project file>` is the path to the “.xcodeproj” file representing your project
   - `<target name>` is the name of the Xcode target you would like to modify

   Here is a complete example:

```
sh MakeAppTouchTestable/bin/MakeAppTouchtestable -project ~/Documents/
Demo/iLabyrinth/iOS/iLabyrinth.xcodeproj” -target “iLabyrinth copy” -url
http://ctmobile.soasta.com/concerto -username SOASTA_DOC -password secret
```

**TIP:** In your own projects if the MakeAppTouchTestable utility is not able to find a main file or finds multiple main files for some reason, it will be necessary to add the parameter `–mainfile` as shown below in order to provide the correct path to the project’s main file:

```
sh MakeAppTouchTestable/bin/MakeAppTouchtestable -project ~/
Documents/Demo/iLabyrinth/iLabyrinth.xcodeproj -target
"iLabyrinth copy" -url http://ctmobile.soasta.com/concerto -
username bob@acme.com -password secret –mainfile <Main file
project directory>/main.m
```

For example, the <Main file project directory might be:

```
/Users/<username>/Desktop/Project/Classes/main.m
```

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

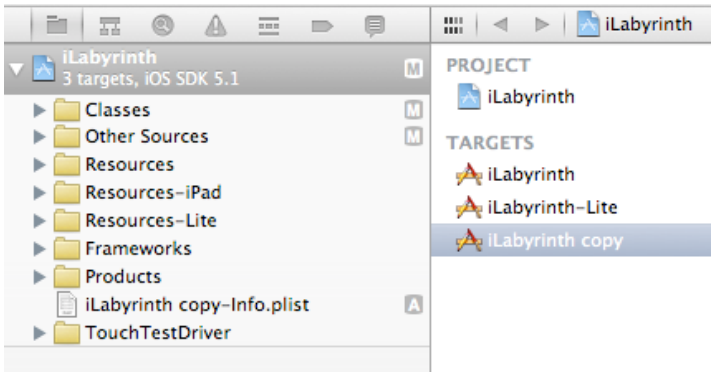
**TIP:** In this basic example, we do not use MATT's `launchurl` flag to create a launch URL. In which case, MakeAppTouchTestable will auto-generate the URL for us. If the flag is used, be sure to avoid spaces and underscores as they will cause an error.

You will see a message similar to the following:

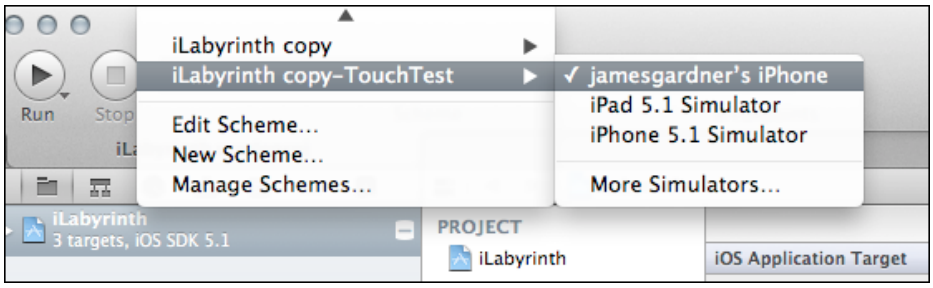
```
Mobile App Object representing your Application "iLabyrinth" has been created
in CloudTest Repository.
```

## Inspecting the TouchTestable Xcode Project

Now that the specified Xcode project has been modified, let's take a look at it. In the screenshot below, note that a new project folder now exists for the TouchTest Driver.




In addition, click the Scheme drop-down in the Xcode toolbar and note the entre with the suffixes "-TouchTest". Select "iLabyrinth-copy-TouchTest" and then the tethered device or simulator.

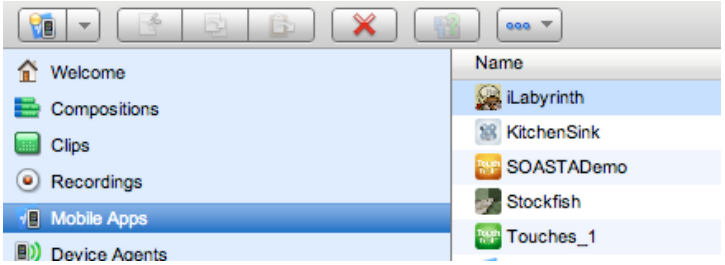


## Inspecting the Mobile App in CloudTest®

In the steps above at the end of each run of the MakeAppTouchTestable.jar we were notified that the “Mobile App Object” had been created in the CloudTest® Repository.

**TIP:** This mobile app will appear 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.

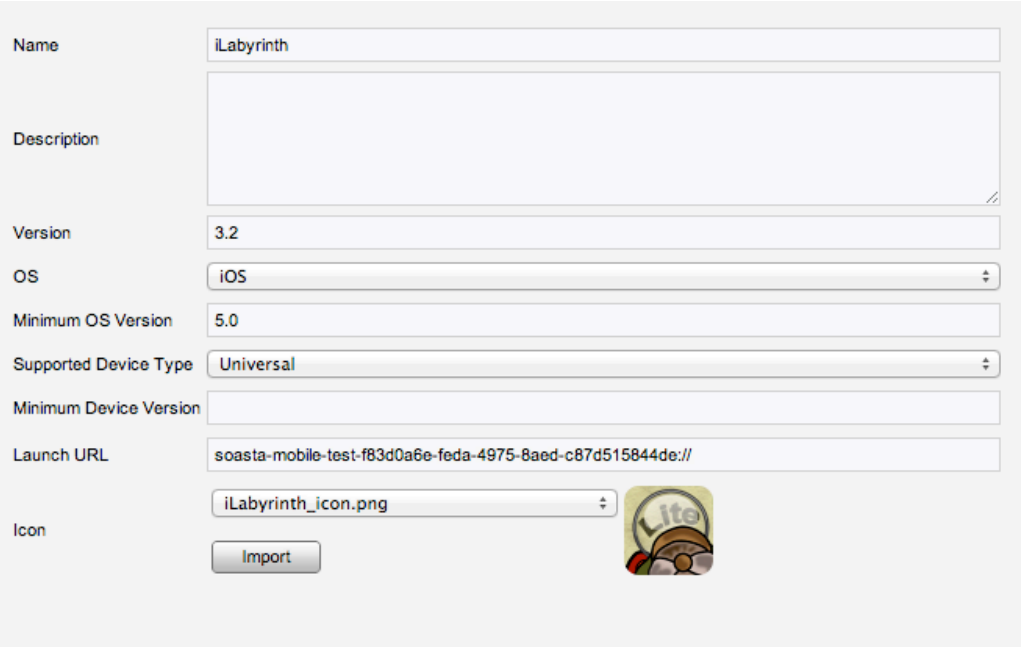
1. Optionally, verify that the Mobile App has been added by logging into CloudTest® and looking for its entry in the Central > Mobile Apps list. For example, in the screenshot below the *iLabyrinth* mobile app appears as expected.




2. Double-click the iLabyrinth mobile app to inspect its details.

   The Mobile App detail form appears.

   - All of the fields shown were populated from the Xcode project, with the exception of Supported Device Type and Minimum OS Version.
   - The default Supported Device Type is Universal (e.g. both iPhone and iPad). Change it to be iPhone- or iPad-specific, if desired.



Note: The Launch URL here must match the URL Types section, URL Schemes entry for iLabyrinth (not including the "`://`" syntax shown in the Mobile App detail above).

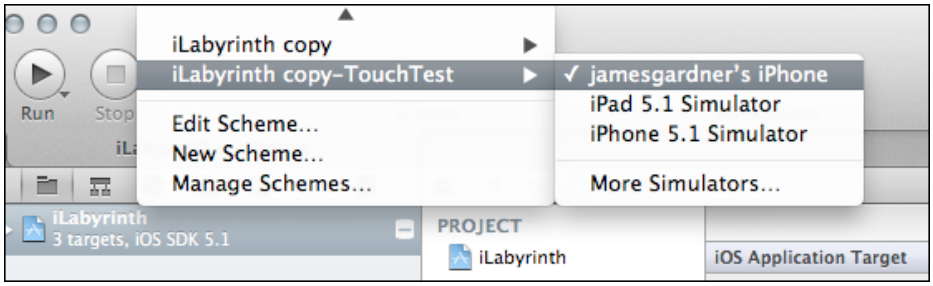
- The default Minimum OS Version supported in TouchTest™ beta is iOS 5.0.

## Launching the TouchTestable App from Xcode

Using the new schemes that were added to your Xcode project by the MakeAppTouchTestable utility, you can now easily:

- Deploy the TouchTestable app to an iOS device or simulator.
- Prepare the iOS device for TouchTest™ recording or playback.
- Use the Xcode debugger to debug your app while a test is running.

1. To deploy and run the TouchTestable app, select the "-TouchTest" scheme from the Xcode Run drop-down (i.e. "iLabyrinth copy-TouchTest"), and then select either a physically-connected iOS device or a built-in Simulator for the given device (i.e. iPad 5.1 Simulator).
2. Click the Run button after making your selection.




Once the app finishes building, it is deployed on the device, and the TouchTest Agent page will appear in Safari. At this point, the device is ready to start recording or playback.

## Registering Your Device to Use TouchTest™

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

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

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

   *http://ctmobile.soasta.com/concerto/touchtest*

The following screen appears.


**Note:** If you're using a Simulator, use the "Tap here…" link if it appears. This link will appear below the Login button in all configurations that require it.

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

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

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


The Unique Device Identifier (UDID) will be used to register the mobile device for use with TouchTest™.

3. Click the Register Device button to continue.
    a. First, the Install Profile screen appears. Click the Install button to proceed.

b. The Unsigned Profile alert appears to indicate that mobile device settings will be changed. Click Install Now to proceed.


c. If a passcode is in effect on the mobile device, an additional prompt will appear to authorize the profile installation.

4. When prompted, give the TouchTest Agent a name. For example *Tester iPad*. Note that this name will be used throughout the product to refer to this device. Once entered the device name can only be changed by an Administrator.
5. Once this name is entered, click Submit for administrator approval.


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

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

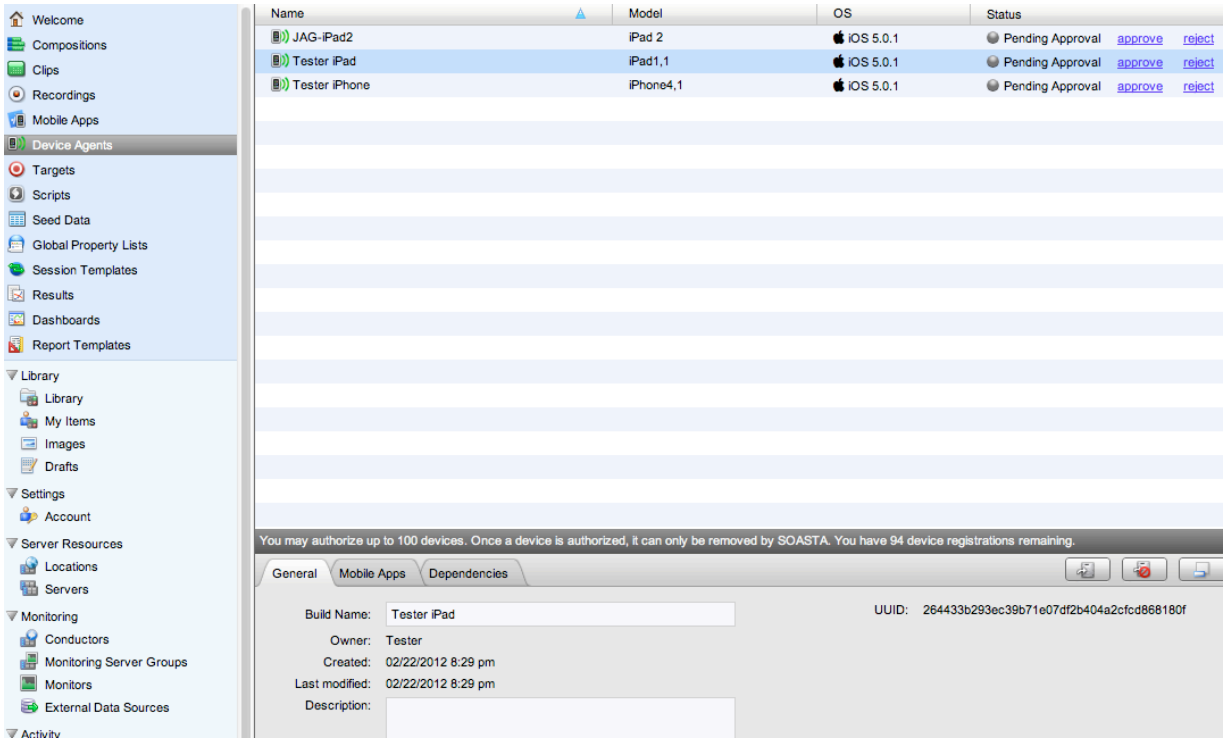
If your device is approved by the Mobile Device Administrator, the Connected page will appear the first time TouchTest™ is launched in Safari on the approved device. On subsequent launches click Login to Connect and Logout to Disconnect.


## Approving a Mobile Device (Administrator Only)

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

1. Login as the user with mobile device administrative rights.
2. Click Central > Device Agents


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




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

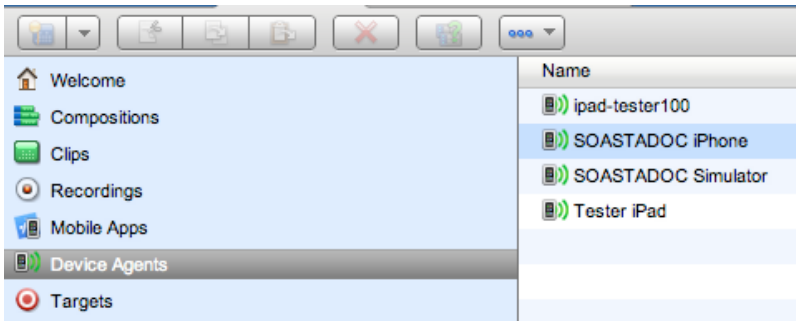
3. Click Approve to complete adding a device and Reject to deny its access.

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

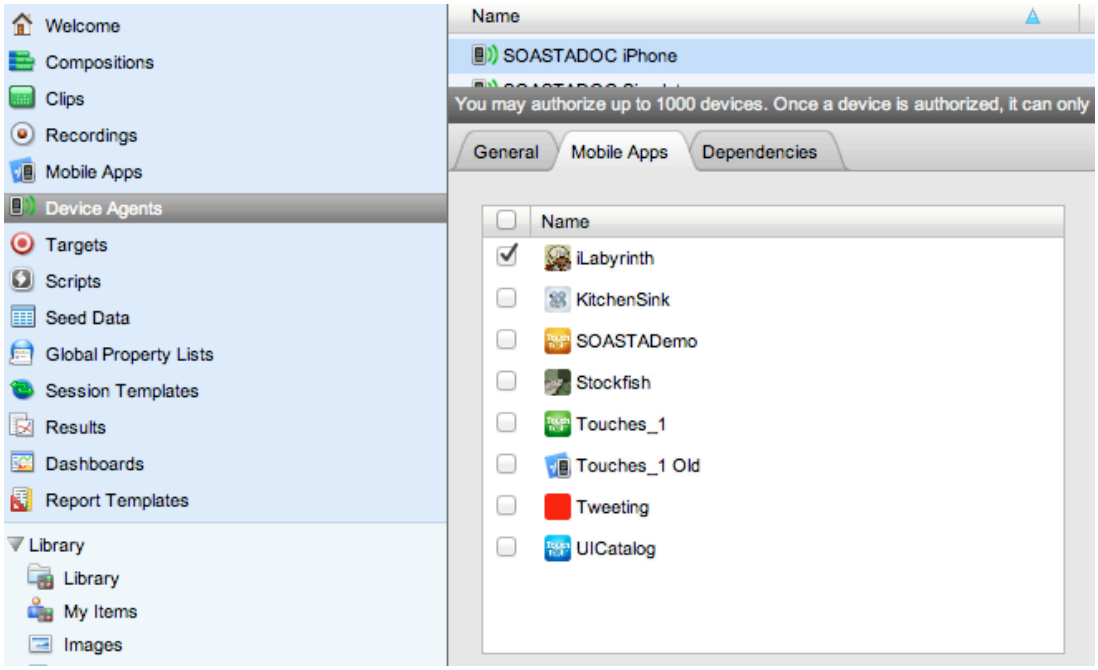
## Associating Mobile Apps with a Device

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

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




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




3. Locate and check the Moblile App(s) that you want to authorize this device to access. For example, *iLabyrinth*.
4. Click Save on the lower panel toolbar.

## Record a TouchTest™ Clip using iLabyrinth

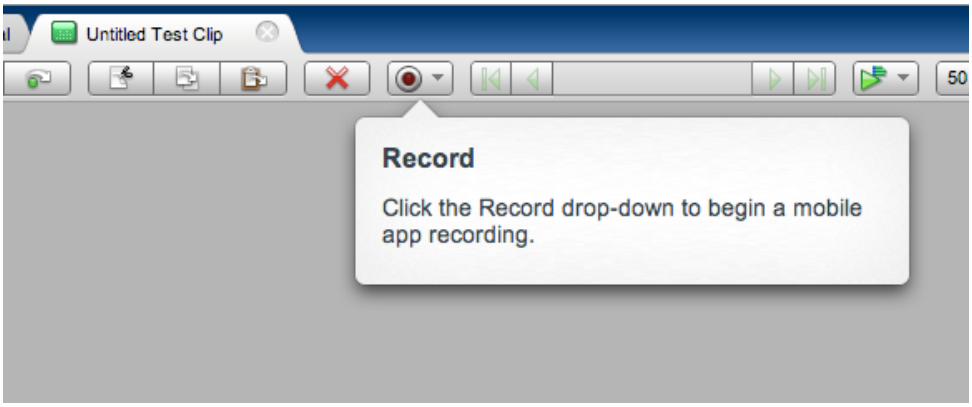
First, we will record a simple iLabyrinth clip and add an output to display the ViewHierarchy from iLabyrinth. Because we have yet to make modifications that will provide solutions to app internal states, this clip will not capture all of the necessary app internal state information that our final test composition will need in order to succeed.

1. Start the TouchTest Agent in Safari before proceeding.

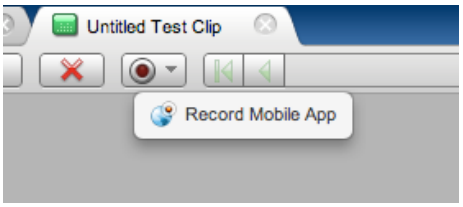

Once successfully logged on, its Status will be *Connected*.

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

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

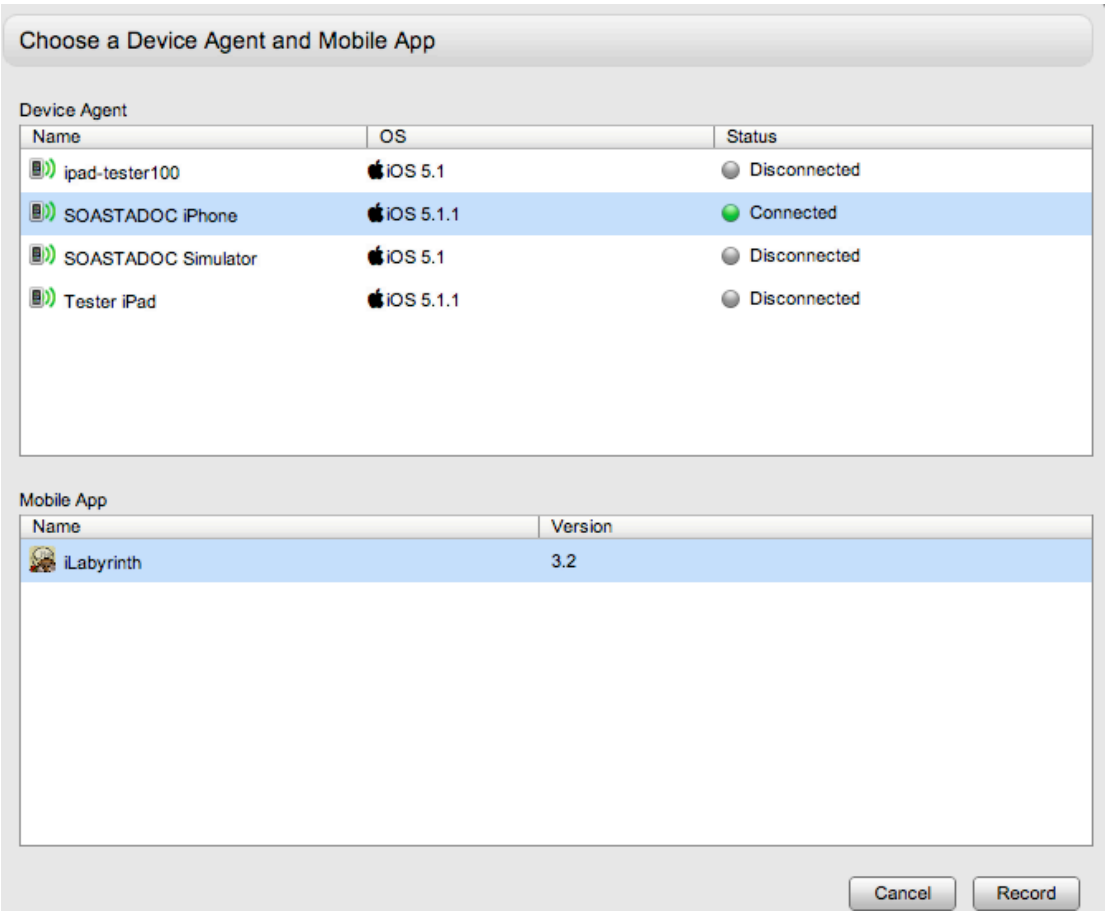


3. Once ready, click the Record drop-down and then select Record Mobile App.




The Choose a Device Agent and Mobile App wizard appears.

**Note:** If the steps to associate mobile apps with the device have been completed, they will appear here in the Mobile App list whenever that device agent is selected.



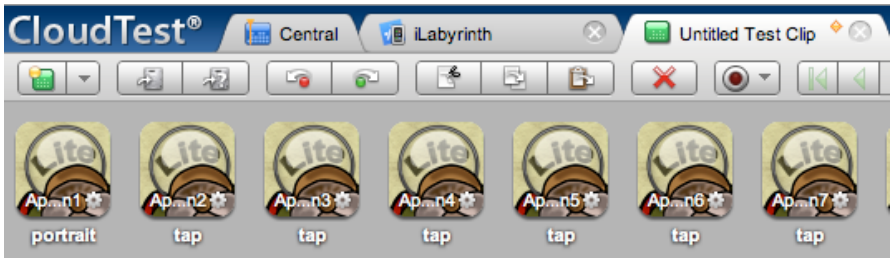

4. Select the TouchTest Agent that you created above and also select the mobile app.
5. Click the Record button in the wizard once your selection is made. TouchTest Agent will launch the selected app on the selected device.

The iLabyrinth app launches to its initial screen.

6. Perform the a series of game actions on your mobile device.
   - Select Play.

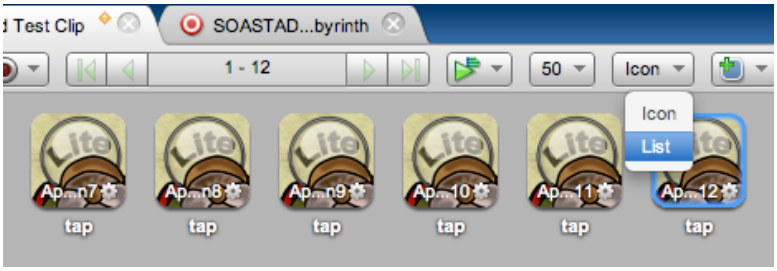
- Tap the path your agent will take in the labyrinth.

7. Click the Record button again to end recording.

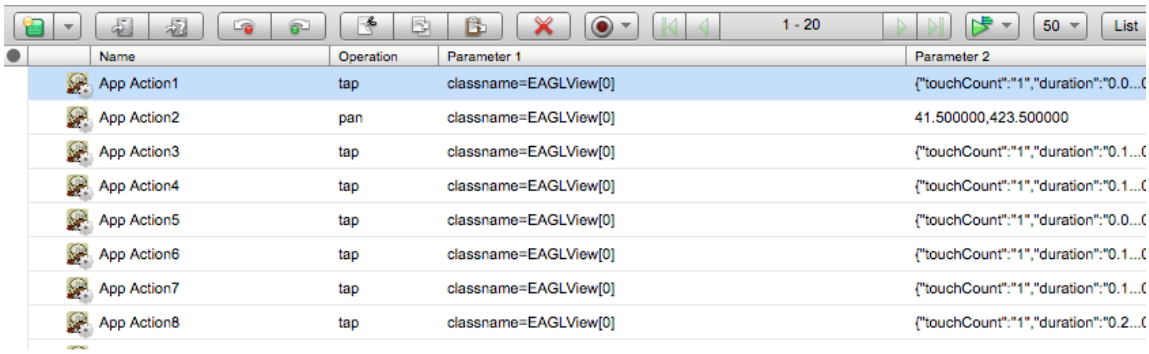



For each app action performed, the Clip Editor adds an app action to the clip.

8. Click Save on the Clip Editor toolbar and name the clip.
9. Next, switch to List view by clicking the Icon drop-down. This will provide a better view on clip element details.



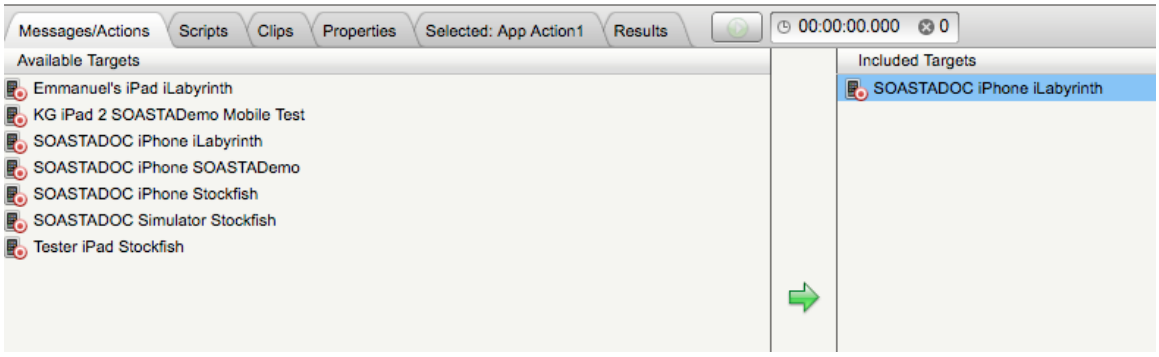

After switching to List view, CloudTest will remember this preference and subsequent clips will be recorded and/or opened in that mode.



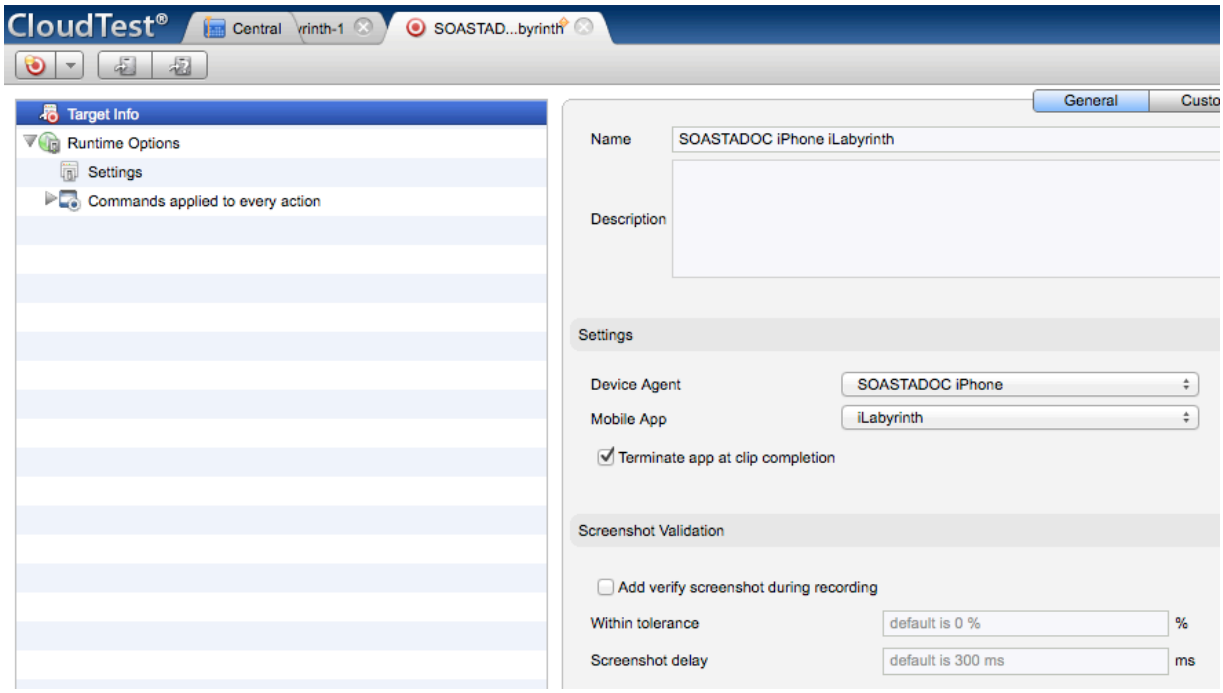

## Adding a Target-Level Output

Before adding this simple test clip to a composition, let's add an output to capture the ViewHierarchy for the iLabyrinth app.

1. Click the Messages/Actions tab in the Clip Editor (at the bottom of the editor).

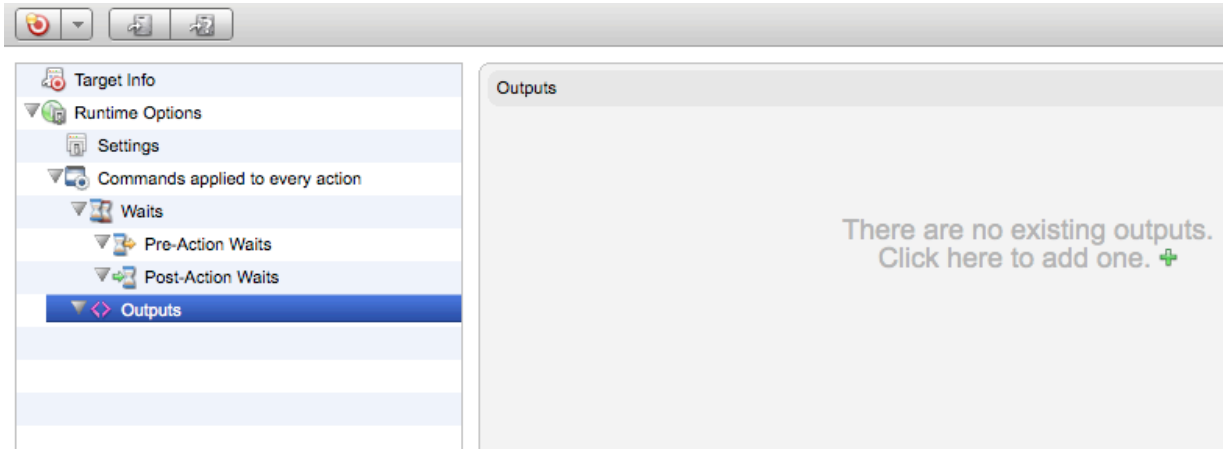



2. In the Included Targets list, double-click the mobile target. The target opens in a new Target Editor tab.

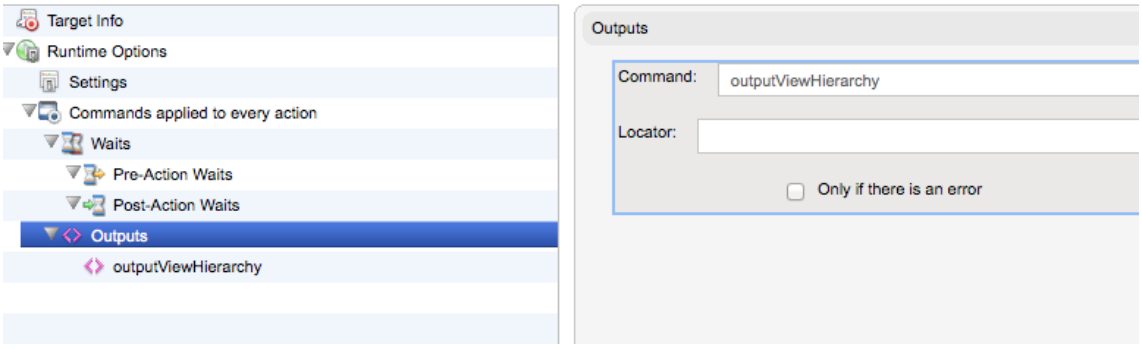



3. Click the arrow to expand Commands applied to every action.

4. Select the Outputs node.



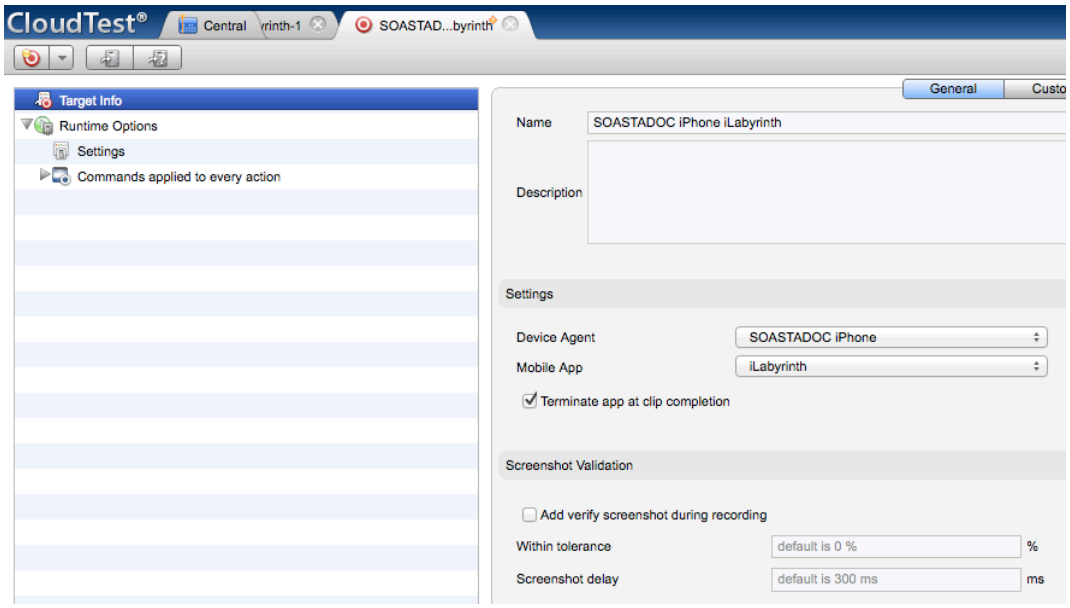

5. Click the green Plus icon to add an output to the workspace.
6. Change the Outputs form, Command drop-down to `outputViewHierarchy`. Leave the Locator field blank.




Leave the Target Editor open for the next section.

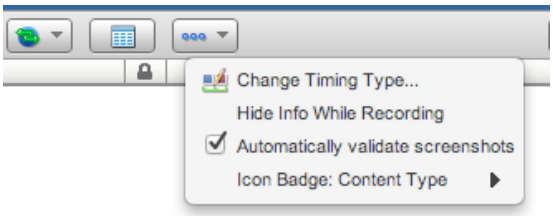
## Turn on Screenshot Validation at the Target Level

1. With the target still open in the Target Editor, click the Target Info node.




2. In the Screenshot Validation section, check the Add verify screenshot during recording box.

**TIP:** Screenshot validation can be also be toggled while recording is active via the Clip Editor, Etc. (…) menu.




3. Save the target.

## Examine the outputViewHierarchy

Play the simple composition to determine which part of the app internal state is actually exposed.

1. In the Clip Editor tab, click the Use in Composition drop-down, then select the Play in Test Composition command.
2. In the Save Test Clip box, name the composition and click OK.
3. The test begins to play and the Result Details dashboard displays.

When the clip appears in the Navigation Tree, click it and expand the Outputs form.

**Output**

Outputs

Name: outputViewHierarchy

Command: output-viewHierarchy

Value: <EAGLView: 0x22cc50; frame = (0 0; 320 480); layer = <CAEAGLLayer: 0x22cda0>>

The ViewHierarchy shown above demonstrates that the OpenGL app internal state is unexposed. What we see here is only one object, that of the main frame. Additionally, since only the main frame is shown in the ViewHierarchy, a meaningful test cannot be built.

## Exposing the App Internal State using TouchTest

In this section, we will learn how the iOS Developer can use TouchTest techniques to expose information from objects that don't display their internal data.

Because we ran the MakeAppTouchTestable utility above (or alternately, the Xcode project modifications described in the Appendix at the end of this tutorial), our project already has the following TouchTest modifications:

- An #ifdef in main.m
- A TouchTestDriver startSession URL in AppDelegate (UDAppDelegate.m in iLabyrinth)

Additionally, we will manually extend the iLabyrinth source to expose some values and characters from the game that will be used in waits and validations in the test clip.

In order to to expose the iLabyrinth app internal states the following project additions will be made:

- Create a new class, TouchTestHelper.h
- Add a helper file, TouchTestHelper.m that has the following:
    - An import statement `#import TouchTestHelper.h`
    - An implementation statement `@implementation TouchTestHelper`
    - An implementation statement `@implementation UIApplication`

    This method will define a category in the iOS framework class, UIApplication to add a CloudTest Mobile method to the existing class. Other TouchTestHelper details are presented in the following two sections.

### Creating the TouchTestHelper.h

Create a new file in your project named TouchTestHelper.h. You can use the code snippets for each step below.

1. In your text editor, add the following three lines to import the necessary header files:

```
#import <Foundation/Foundation.h>
#import "CCScene.h"
#import "UDCharacter.h"
```

2. Add an @interface line that calls NSObject with the additional lines shown below:

```
@interface TouchTestHelper : NSObject

+ (TouchTestHelper *)sharedInstance;
+ (id)accessObject:(NSString*)name inObject:(id)object;

- (NSString *)getAppInternalValue:(NSString *)value withArgs:
(NSDictionary *)args;
```

3. Add the `@end` statement.

Your TouchTestHelper.h file should look like this:

```
#import <Foundation/Foundation.h>
#import "CCScene.h"
#import "UDCharacter.h"

@interface TouchTestHelper : NSObject

+ (TouchTestHelper *)sharedInstance;
+ (id)accessObject:(NSString*)name inObject:(id)object;

- (NSString *)getAppInternalValue:(NSString *)value withArgs:(NSDictionary *)args;

@end
```

## Creating the TouchTestHelper.m File

1. Create a new file named TouchTestHelper.m. You can use the code snippets for each step below.
2. Add three import statements to utilize the header file, game scene and layer:

```
#import "TouchTestHelper.h"
#import "UDGameScene.h"
#import "UDGameLayer.h"
```

1. Add the implementation line to specify TouchTestHelper:

```
@implementation TouchTestHelper
id originalDelegate;

static TouchTestHelper *_sharedInstance = nil;
```

2. Add a singleton accessor section:

```
/**
 * Singleton accessor
 */
+ (TouchTestHelper *)sharedInstance {
        @synchronized( [TouchTestHelper class] ) {
                if ( !_sharedInstance ) _sharedInstance = [[self alloc]
init];
                return _sharedInstance;
        }
        // to avoid compiler warning
        return nil;
}
```

3. Add an Introspection utility method:

```
/**
 * Introspection util method
 */
+(id)accessObject:(NSString*)name inObject:(id)object
{
  return [object valueForKey:name];
```

```
}
```

4. Using NSString, add the getAppInternalValue method that will return the app internal states using the accompanying definitions. The returned values will be used in CloudTest via the accessors outputAppInternalState, waitForAppInternalState, and verifyAppInternalState.

```
/**
 *
 * This method returns the app internal states defined as :
 *
 *   - scene                : return the current scene class name, empty
string if no game scene found.
 *   - character.position   : return the main character position, N/A if
it is not a the current scene do not involve the character
 *   - character.status     : main character status, if it is walking the
status is "busy" else it is idle even if the current scene do not involve
the character
 *
 */
-(NSString *)getAppInternalValue:(NSString *)value withArgs:(NSDictionary
*)args
{
  NSString *result = @"n-a";

  // Get the current scene
  CCScene *_currentScene = [[CCDirector sharedDirector] runningScene];

  if([value isEqual:@"scene"])
  {
    // Get the current scene
    if(_currentScene)
    {
      result = [[_currentScene class] description];
    }
  }
  else if([value isEqual:@"character.position"])
  {
    // Get the current character position
    if([_currentScene isKindOfClass:[UDGameScene class]])
    {
      UDGameLayer *gameLayer = (UDGameLayer*) [TouchTestHelper
accessObject:@"_gameLayer" inObject:_currentScene];
      UDCharacter *character = (UDCharacter*) [TouchTestHelper
accessObject:@"_char" inObject:gameLayer];
      result = NSStringFromCGPoint([character gridPosition]);
    }
  }
  else if([value isEqual:@"character.status"])
  {
      BOOL isWalking = NO;
      // If it is a game scene it happen the charDidFinishWalking is fired
before the character finished moving
      // In that case we will double check also if the character really
ended
      if([_currentScene isKindOfClass:[UDGameScene class]])
      {
        UDGameLayer *gameLayer = (UDGameLayer*) [TouchTestHelper
```

```
accessObject:@"_gameLayer" inObject:_currentScene];
        UDCharacter *character = (UDCharacter*) [TouchTestHelper
accessObject:@"_char" inObject:gameLayer];
        NSLog(@"Flagged as non walking and buzy = %@", [character
isBussy] ? @"YES" : @"NO");
        isWalking = [character isBussy];
      }

      // Get the current character status, walking boolean is assigned
using UDCharacterDelegate (charWillBeginWalking/charDidFinishWalking)
      result = isWalking ? @"busy" : @"idle";
  }

  NSLog(@"%@='%@'", value, result);

  return result;
}

 @end
```

5. Finally, add a category on UIApplication to get some values and arguments that will be used in CloudTest's waitForAppInternalState wait.

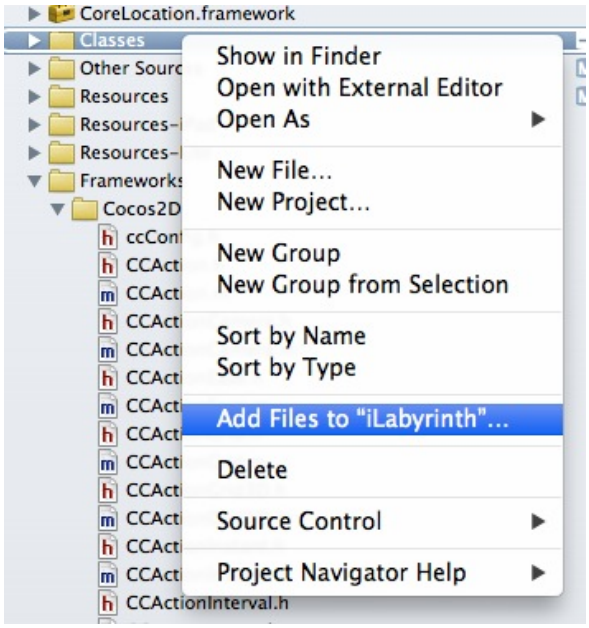
   This category will get the current scene (runingScene) from the Cocos2D framework, the character position on the game screen (gridPosition), as well as the character status (`busy`/`idle`).

```
/**
 * Category on UIApplication required to initialize the
appInternalValue accessor in TouchTest
 */
@implementation UIApplication (CloudTestMobile)

/**
 * This is a general purpose function to return internal app state.
 */
+(NSString *)getAppInternalValue:(NSString *)value withArgs:
(NSDictionary *)args
{
  return [[TouchTestHelper sharedInstance]
getAppInternalValue:value withArgs:args];
}


@end
```

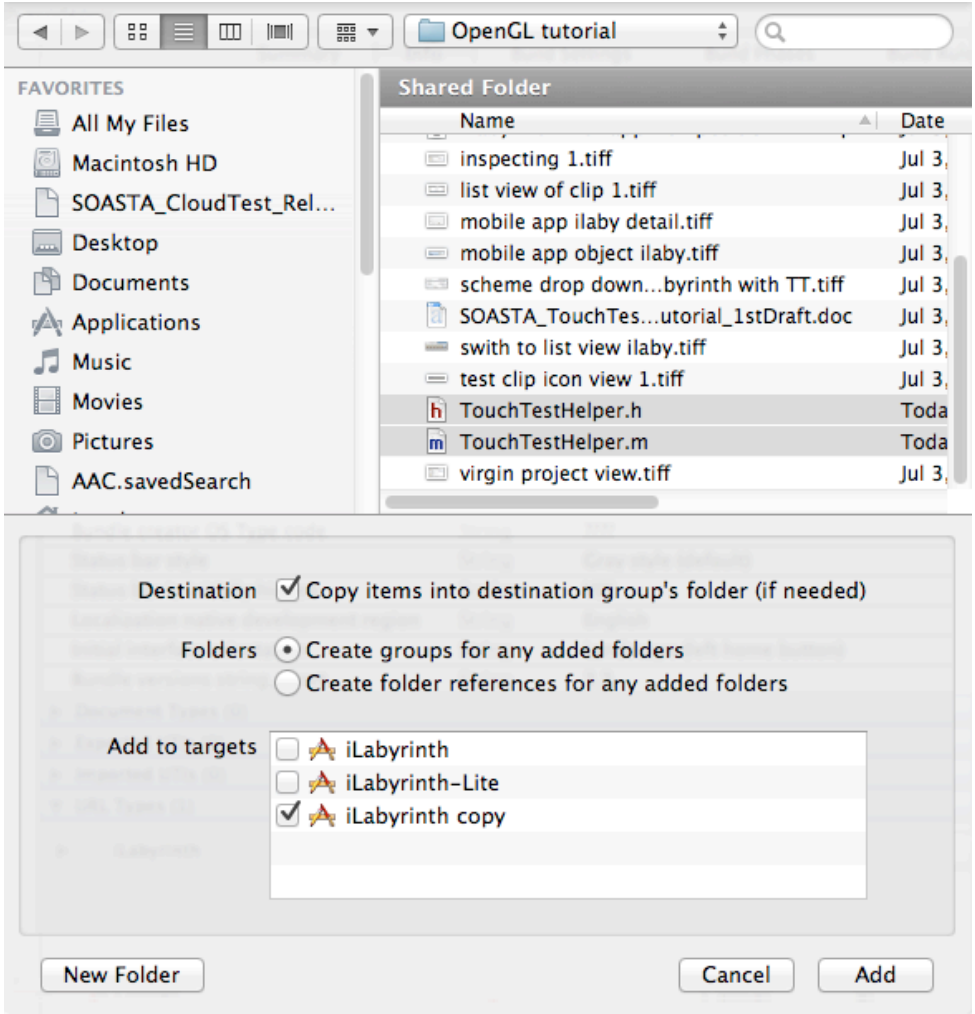
### Add the TouchTestHelper Files to the Automation Target

Both of the TouchTestHelper files should be added to the project using the following steps.

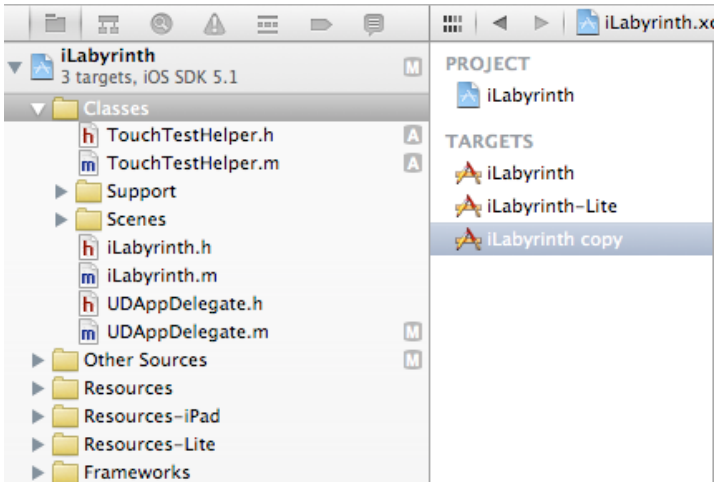
1. With the project open, select the iLabyrinth > Classes folder and right-click to choose Add Files to Project.




2. Navigate to the location of the TouchTestHelper files and select them.
3. Check the box marked "Copy items into the destination group's folder" (if necessary).




4. Specify the newly-created Automation Target (iLabyrinth copy) by checking its box in the Add to targets section in the lower panel.
5. Click Add to complete adding the files.



## Managing Animations and Map Levels in iLabyrinth

Since the iLabyrinth app, like many OpenGL apps, includes character and scene animations we will offer some options to turn them off in the sections below. Additionally, a technique to provide TouchTest easier access to the game levels selected on the PickMap is presented.

### Revising iLabyrinth.m to Access Map Levels

To gain easier access to each iLabyrinth game level, insert the following code as described below.

1. Open the iLabyrinth.m file.
2. Locate the following section:

```
- (BOOL)canPlayMap:(NSUInteger)map {
#if TARGET_IPHONE_SIMULATOR
    // If we are on simulator, enable all maps for easyer testing
    return YES;
#endif

     if( map == 1 || [_compleatedMaps containsObject:[NSNumber
numberWithInt:map]] || [_compleatedMaps containsObject:[NSNumber
numberWithInt:map-1]] ){
             return YES;
     }

return NO
}
```

3. Insert the following #ifdef statement after the #endif line shown above.

```
#ifdef TOUCHTESTDRIVER
  // If we are on TOUCHTESTDRIVER, enable all maps for easier testing
  return YES;
#endif
```

### Revising UDGameLayer.m to Turn off Water Animation

To make it easier to take and verify screenshots of the mobile app during testing, we will turn off iLabyrinth's water animations since they are not crucial to the test. This will be done by unincluding all but one of the water sprites used in the app.

**Note:** Although CloudTest's tolerance setting could do the same thing, it won't do as efficiently in this case because the water size is too near the character size, which might miss real image validation failure.

When relying upon image comparison validation, it can be useful to disable game animations that might otherwise be in display. By using this optional technique, you can workaround having to set image tolerance for each verifyScreenshot.

The following steps will turn off iLabyrinth's water animations using #ifndef with the TOUCHTESTDRIVER macro.

1. Open the UDGameLayer.m file (in the Scenes, UDGameScene, GameLayer folder).
2. Go to the end of the first sprite line (the line whose PNG value is `@"S21_2.png`) or Line 226.

```
NSMutableArray *animFrames = [NSMutableArray array];
[animFrames addObject:[[CCSpriteFrameCache sharedSpriteFrameCache] spriteFrameByName:@"S21_2.png"]];
[animFrames addObject:[[CCSpriteFrameCache sharedSpriteFrameCache] spriteFrameByName:@"S21_3.png"]];
[animFrames addObject:[[CCSpriteFrameCache sharedSpriteFrameCache] spriteFrameByName:@"S21_4.png"]];
[animFrames addObject:[[CCSpriteFrameCache sharedSpriteFrameCache] spriteFrameByName:@"S21_5.png"]];
[animFrames addObject:[[CCSpriteFrameCache sharedSpriteFrameCache] spriteFrameByName:@"S21_6.png"]];
[animFrames addObject:[[CCSpriteFrameCache sharedSpriteFrameCache] spriteFrameByName:@"S21_7.png"]];
[animFrames addObject:[[CCSpriteFrameCache sharedSpriteFrameCache] spriteFrameByName:@"S21_8.png"]];

[upperSprite runAction:[CCRepeatForever actionWithAction:
                        [CCAnimate actionWithAnimation:
                         [CCAnimation animationWithFrames:animFrames delay:0.5f]]]];
```

1. Add a new line before the next sprite line:

   **#ifndef TOUCHTESTDRIVER**

2. Add a new line after the last sprite line:

   **#endif**

Your modifications should look like this:

```
NSMutableArray *animFrames = [NSMutableArray array];
[animFrames addObject:[[CCSpriteFrameCache sharedSpriteFrameCache] spriteFrameByName:@"S21_2.png"]];
#ifndef TOUCHTESTDRIVER
[animFrames addObject:[[CCSpriteFrameCache sharedSpriteFrameCache] spriteFrameByName:@"S21_3.png"]];
[animFrames addObject:[[CCSpriteFrameCache sharedSpriteFrameCache] spriteFrameByName:@"S21_4.png"]];
[animFrames addObject:[[CCSpriteFrameCache sharedSpriteFrameCache] spriteFrameByName:@"S21_5.png"]];
[animFrames addObject:[[CCSpriteFrameCache sharedSpriteFrameCache] spriteFrameByName:@"S21_6.png"]];
[animFrames addObject:[[CCSpriteFrameCache sharedSpriteFrameCache] spriteFrameByName:@"S21_7.png"]];
[animFrames addObject:[[CCSpriteFrameCache sharedSpriteFrameCache] spriteFrameByName:@"S21_8.png"]];
#endif
[upperSprite runAction:[CCRepeatForever actionWithAction:
                        [CCAnimate actionWithAnimation:
                         [CCAnimation animationWithFrames:animFrames delay:0.5f]]]];
```

## Adding App Internal States to a Test

Next, let's return to the Clip Editor and revise the test clip created above to use the newly exposed app internal states. We will use the following guidelines:

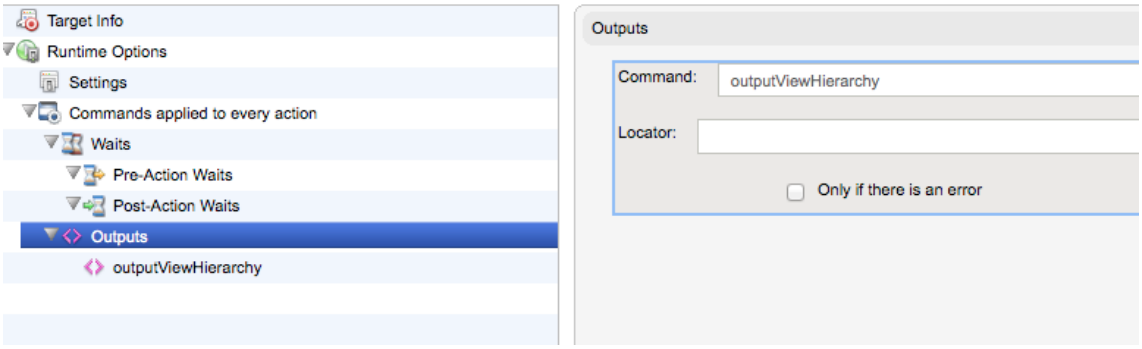
- At the target level, add a post-wait action for character.status `idle` so that no actions are performed during a character's animation.
- Each time the action changes the scene, add a post-wait action for the scene value.
- Optionally, use character position to perform some validations.

## Adding Outputs for AppInternalState at the Target Level

Next, we'll add two outputs that utilize outputAppInternalValue.

1. Click the arrow to expand Commands applied to every action.


4. Change the Outputs form, Command drop-down to `outputViewHierarchy`. Leave the Locator field blank.



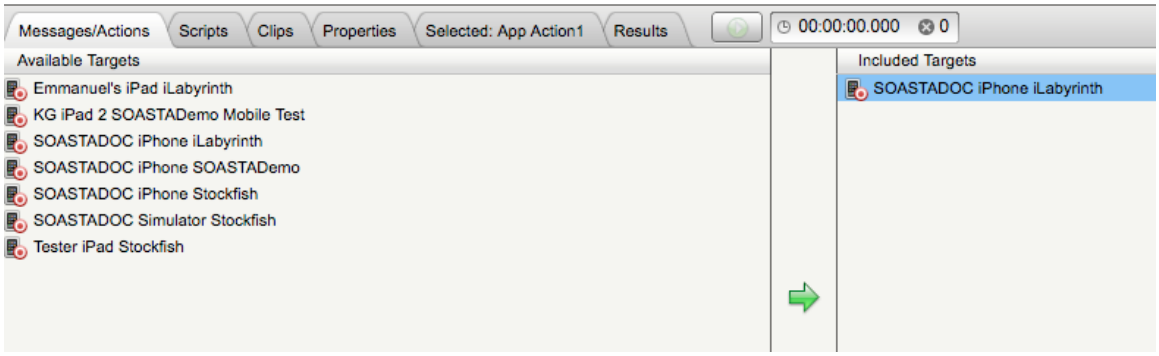

5. Save the target.

## Adding a Wait for Idle at the Target Level

Before adding this simple test clip to a composition, let's add an target-level post-action wait that will wait for the character status to be idle.

**Note:** The developer will want to ensure that accessor is going to return ‘idle’ in cases where the character isn’t in display. For example, on the game scene or before pressing play game, there’s no character, the accessor is ‘idle’ for cases when the character is not present.

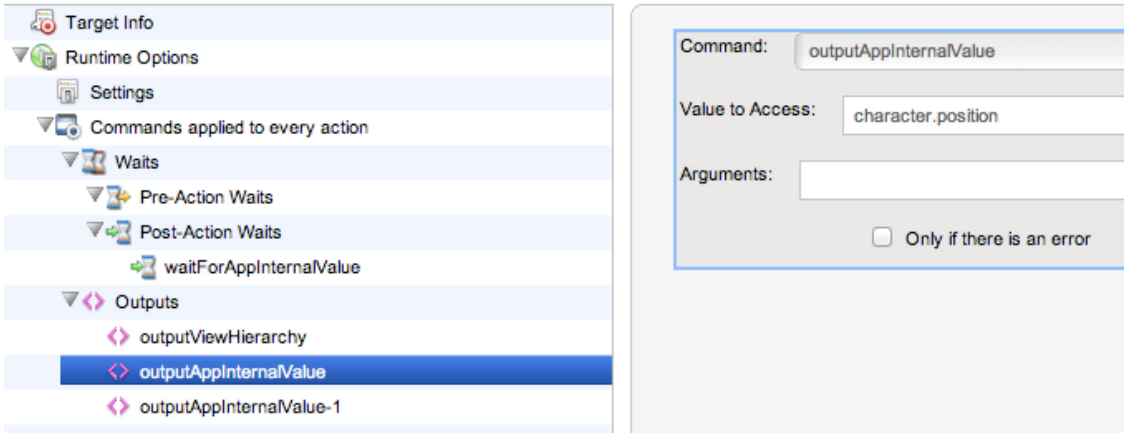
2. Click the Messages/Actions tab in the Clip Editor (at the bottom of the editor).



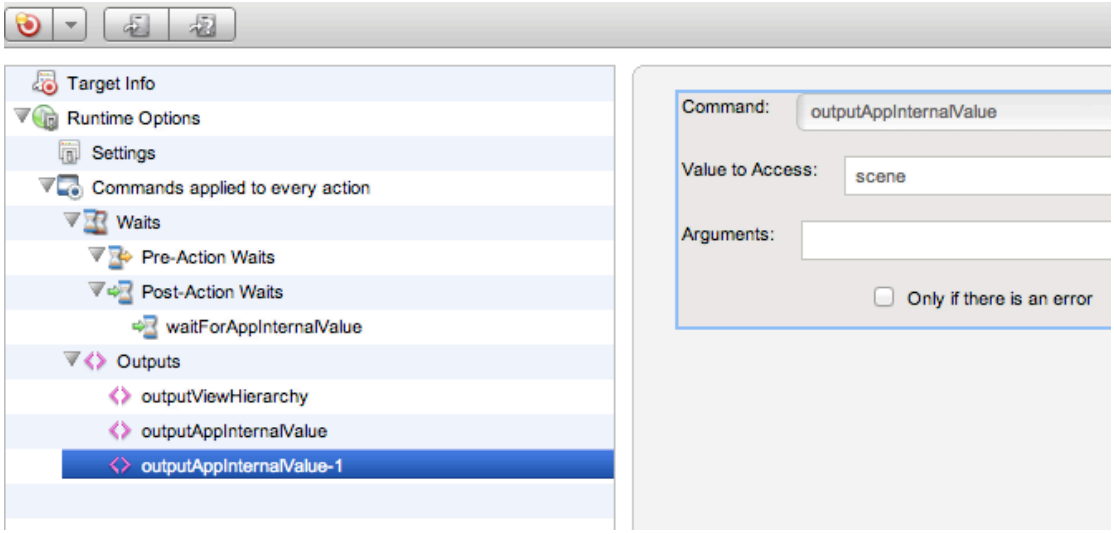

3. In the Included Targets list, double-click the mobile target. The target opens in the Target Editor tab as before.
4. Click the arrow to expand Commands applied to every action.


5. Select the Waits, Post-Action Waits node.


6. Click the green Plus icon to add an output to the workspace.
7. Change the Outputs form, Command drop-down to `waitForAppInternalValue`.




8. In the Value to Access field, use the `character.position` value we defined in the app source above.
9. Click the green Plus icon a second time to add another output.
10. Change the Outputs form, Command drop-down to `waitForAppInternalValue`.
11. In the Value to Access field, use the `scene` value we defined in the app source above.



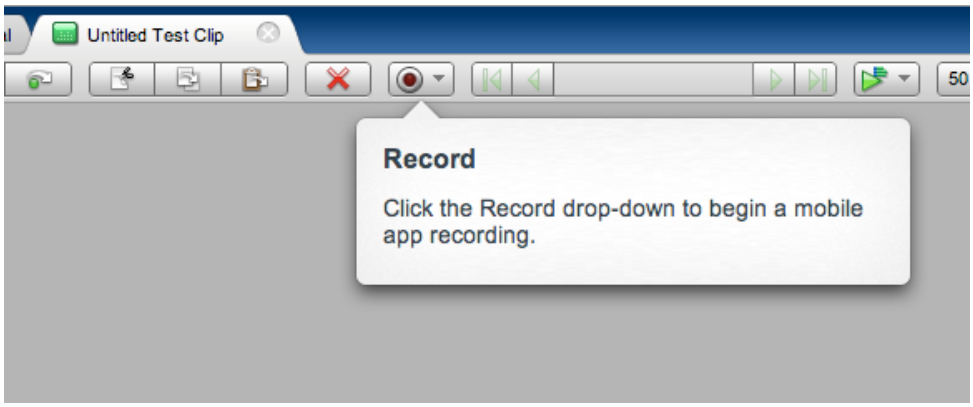

12. Save the target.

## Record a Second Clip with Screenshot Validation

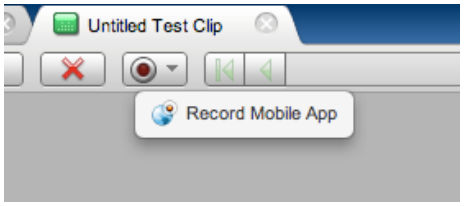
Now that the underlying mobile target has screenshot validation enabled, and the project app internal values are exposed, create a second Untitled Test Clip using iLabyrinth.

1. With Central > Clips selected, click New on the Central toolbar.

A new Untitled Test Clip opens in a Clip Editor tab.




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



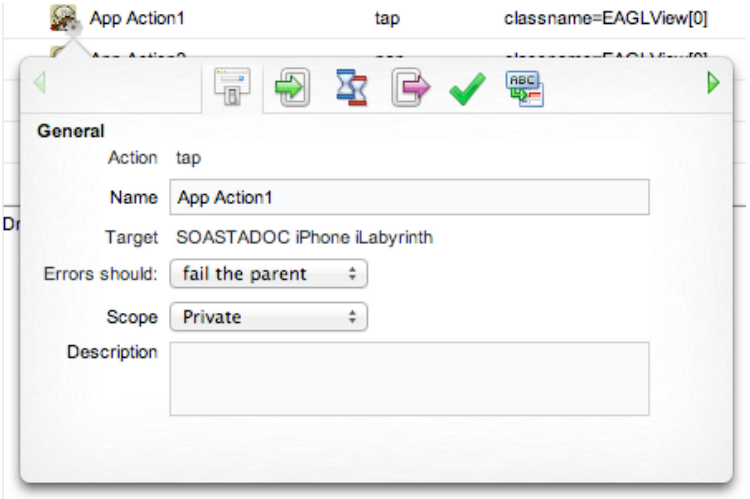

The Choose a Device Agent and Mobile App wizard appears. Select the same Device Agent and Mobile App to utilize the same underlying mobile target as before (i.e. the one that has Screenshot Validation enabled).

3. Perform one complete level of the iLabyrinth game at any level of the game.
4. Click the Record button again to end recording.

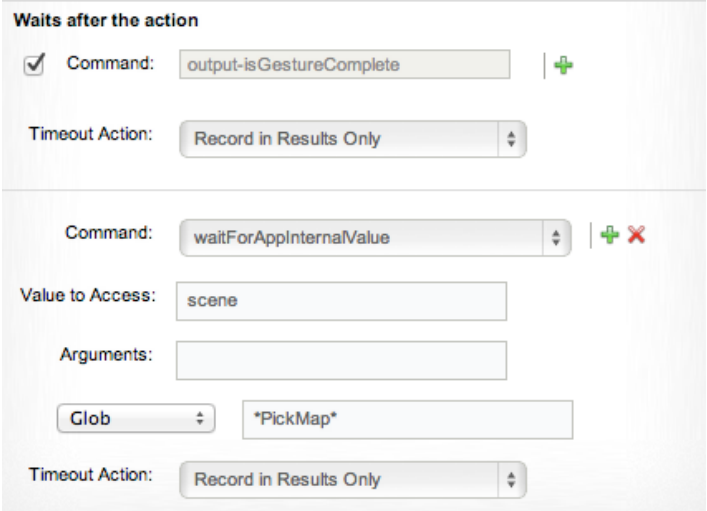
## Adding a Post-Wait for a Scene Change

For every scene change at the action level, we will add a post-action wait using the steps below.

1. Click App Action1's Gear icon to show the Info Window.
2. In the General tab, change App Action1 to New Game.



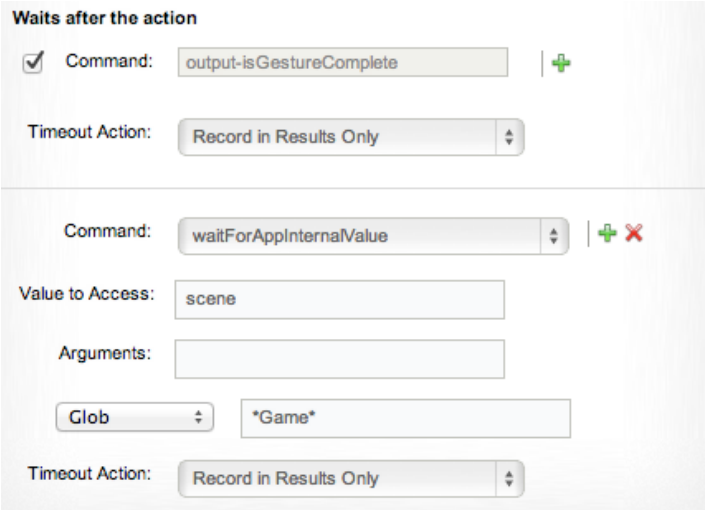
3. Click the Waits tab.
4. In the Waits after the action section, click the green Plus icon.
5. Change the Command to `waitForAppInternalValue`.




6. The Value to Access is `scene`.
7. Change the Pattern selector to `Glob` and enter `*PickMap*` to (this will get a match from the UDPickMapScene).

**TIP:** This post-action wait needs to be added for every action that changes the scene. In iLabyrinth, this means adding the wait for every switch between UDPickMapScene, UDGameScene, UDMenuScene, and so forth.

For example, add a second post-action wait for the action that changes the scene to UDGame Scene. In this post-action wait, we will use a Glob, `*Game*`, that will match UDGameScene.

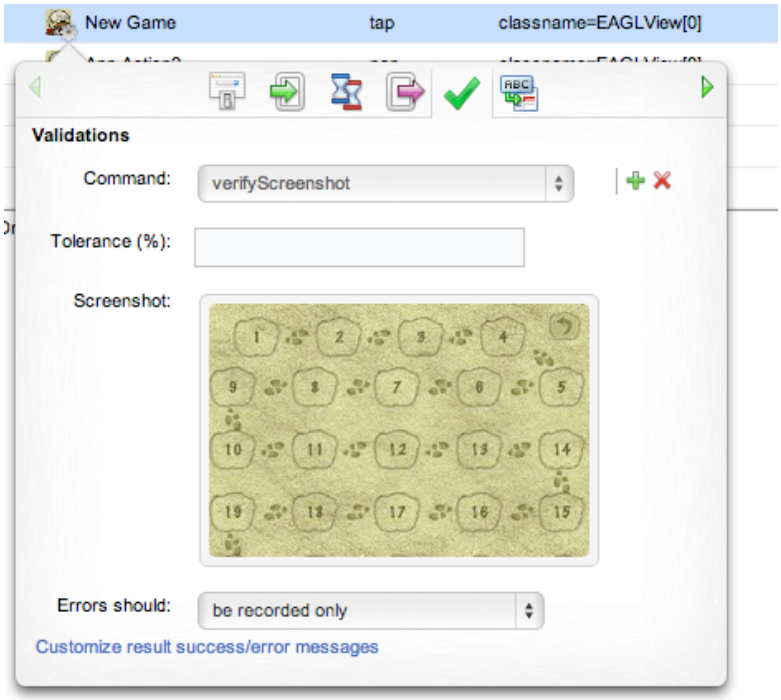

Waits after the action
Command:
output-isGestureComplete
Timeout Action:
Record in Results Only
Command:
waitForAppInternalValue
Value to Access:
scene
Arguments:
Glob
*Game*
Timeout Action:
Record in Results Only

## Examining Image Validations using verifyScreenshot

Validations are the technique used to verify that a test event occurs as expected.

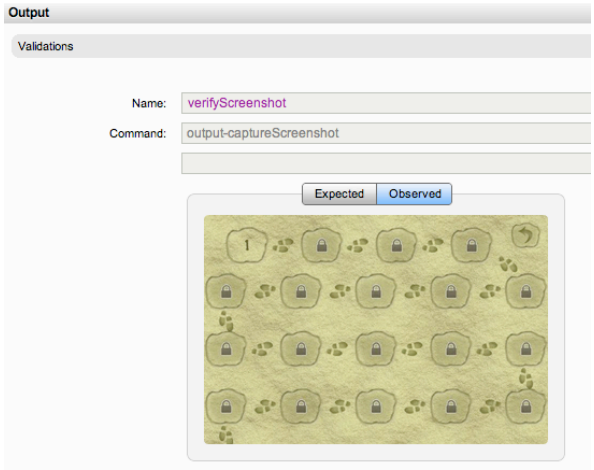
By turning on Screenshot Validation at the target level before creating a mobile recording, we automatically added the `verifyScreenshot` validation to each of the recorded app actions.

1. With the the Info Window for App Action1 open, click the Validation tab.
2. Note that the validation has been added and the screenshot to compare has also been captured during mobile app recording.

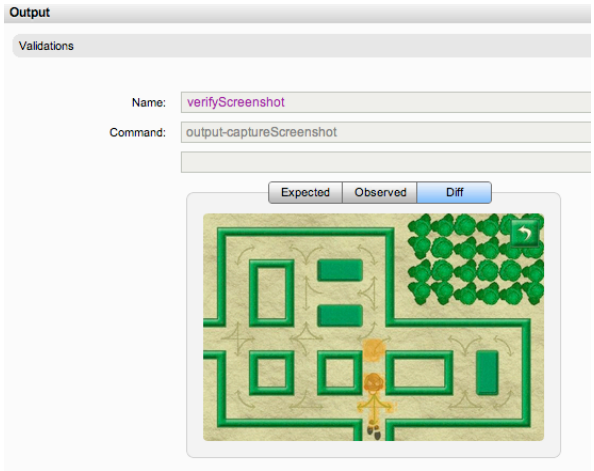



**TIP:** This validation can be added for as many actions as desired.

In the case of verifyScreenshot, CloudTest Mobile will verify by image comparison in the Outputs form, Validations section, Expected (captured while recording) and the Observed (captured during playback) tabs.



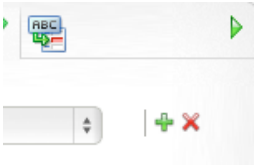

Where any change is detected in the comparison, a Diffs tab will also appear in the Result Details, Outputs form.




## Improving Test Clip Readability

Use the following clip editing techniques to ensure a readable, easy-to-follow test result.

1. Optionally, make the following additional test clip changes by scrolling through the Info Window (using the right arrow at the top of the window).

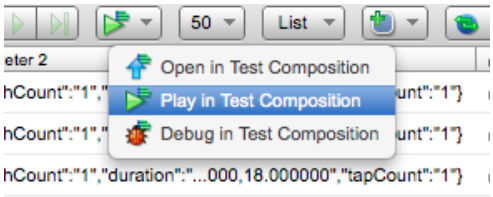


- Rename App Action2 to Start Game.
- Select the remaining app actions,  right-click, and then choose Add to Group.
- Rename the new Group1 to Play game

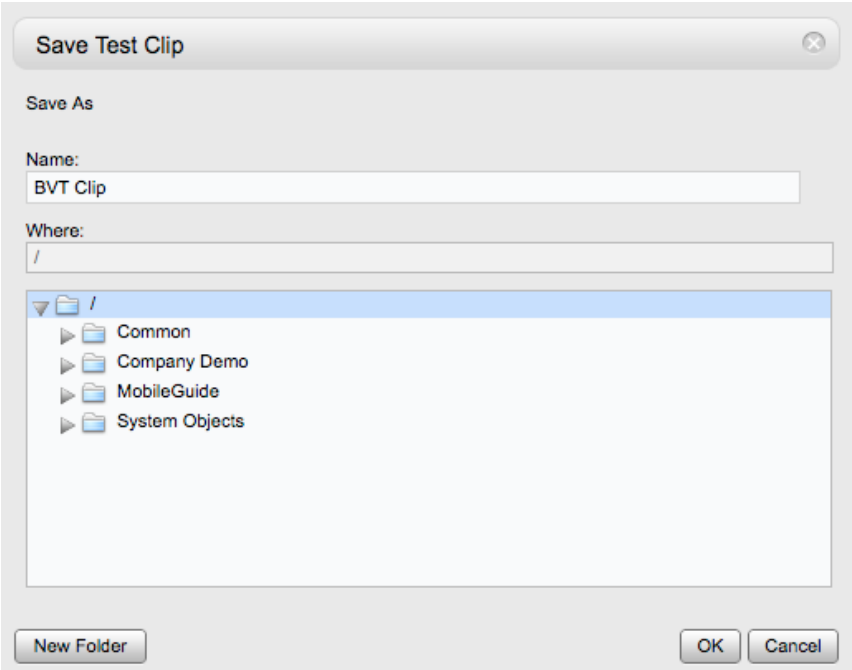
| Name | Operation | Parameter 1 |
|---|---|---|
| New Game | tap | classname=EAGLView[0] |
| Start Game | tap | classname=EAGLView[0] |
| Play game | | |
| App Action3 | tap | classname=EAGLView[0] |
| App Action4 | tap | classname=EAGLView[0] |
| App Action5 | tap | classname=EAGLView[0] |
| App Action6 | tap | classname=EAGLView[0] |
| App Action7 | tap | classname=EAGLView[0] |
| App Action8 | tap | classname=EAGLView[0] |
| App Action9 | tap | classname=EAGLView[0] |

## Viewing App Internal States in a Result

1. From the Clip Editor toolbar, click the Use in Test Composition drop-down and select Play in Test Composition from the menu.




2. The Save dialog box appears. Name the clip and click OK.




If your TouchTest Agent status is “Connected” the composition plays, otherwise respond to the prompt by starting it and then clicking Continue.

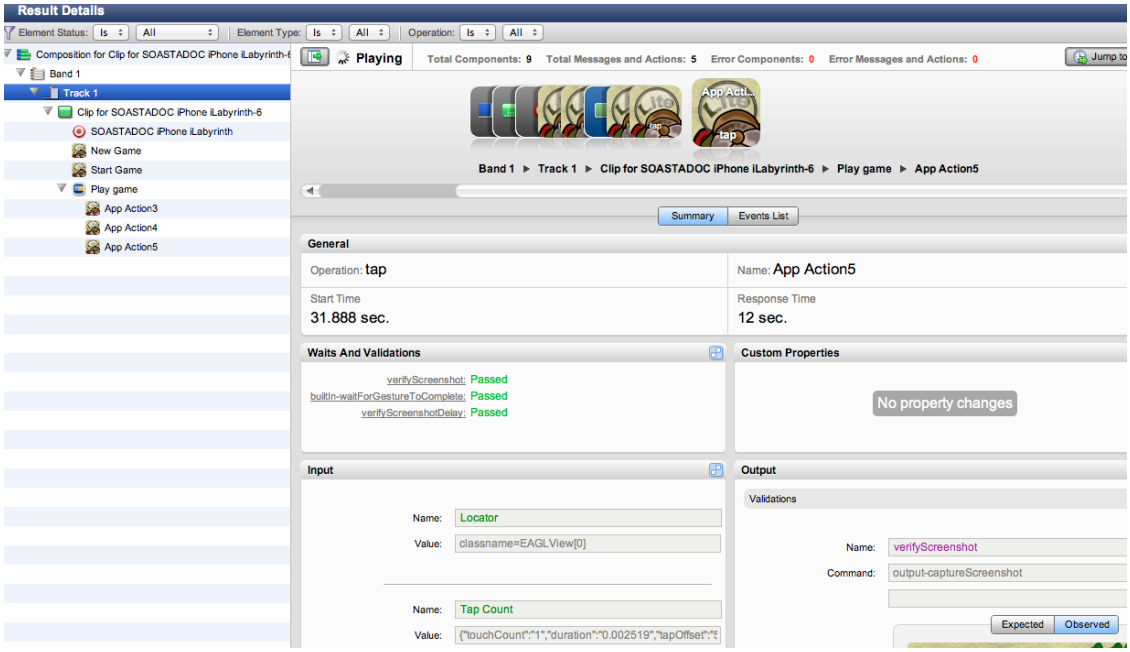
When you do so, a new Composition Editor tab opens with the example clip placed into Track 1. Once loaded, the test composition begins to play and the  mobile app actions are repeated on the mobile device precisely as specified.

## Inspecting Result Details

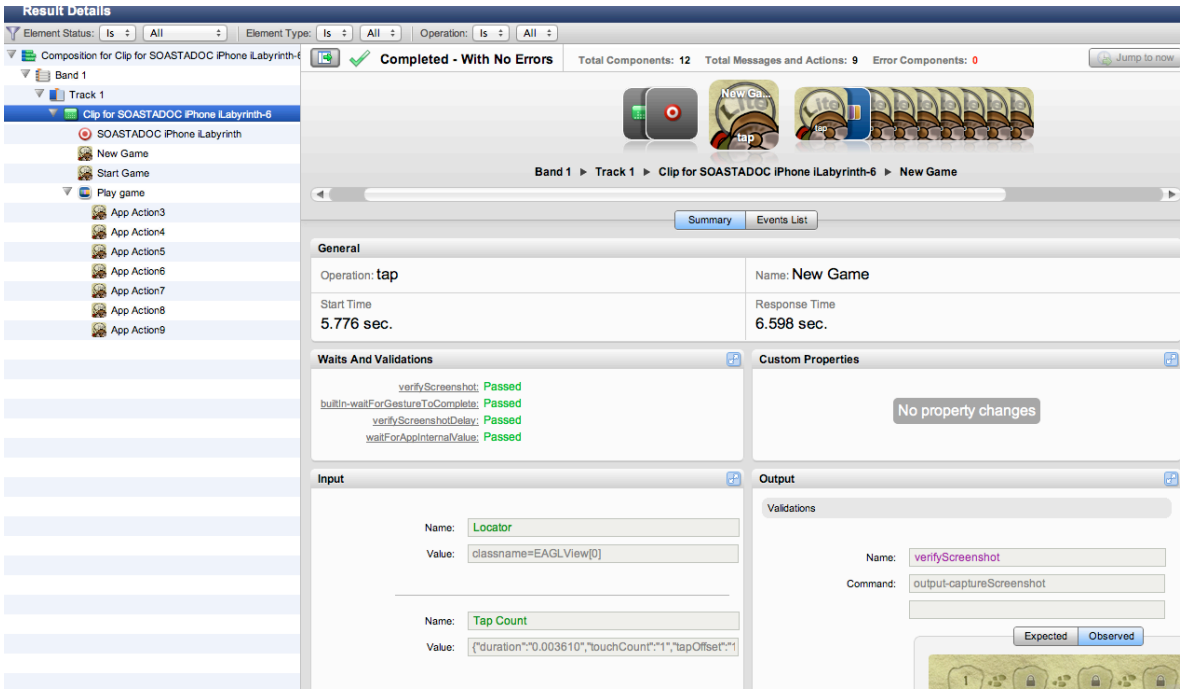
While the test runs, the Composition Editor’s Status Indicator changes to “Playing”

The Play tab displays and the Result Details dashboard is shown. The Result Details dashboard helps to discover the cause of errors in your test, if any.

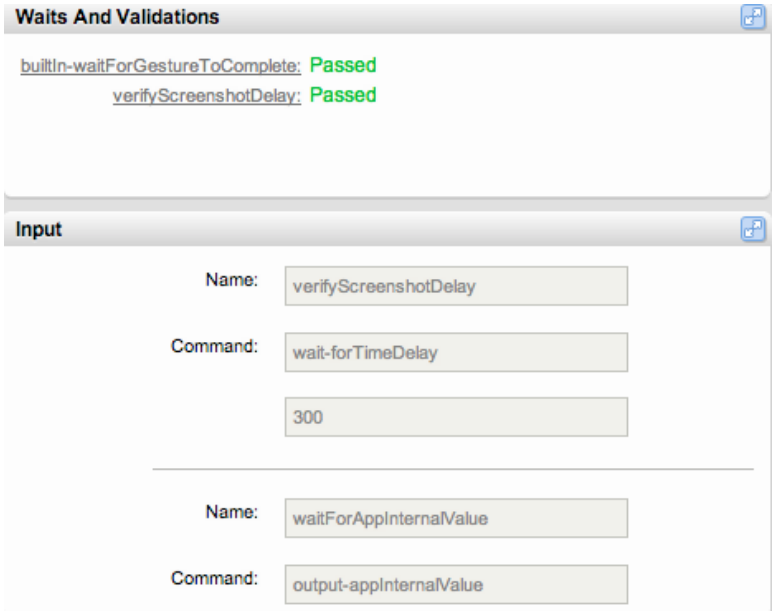
While play continues results are posted in the Composition Editor, Play tab, Result Details widget.



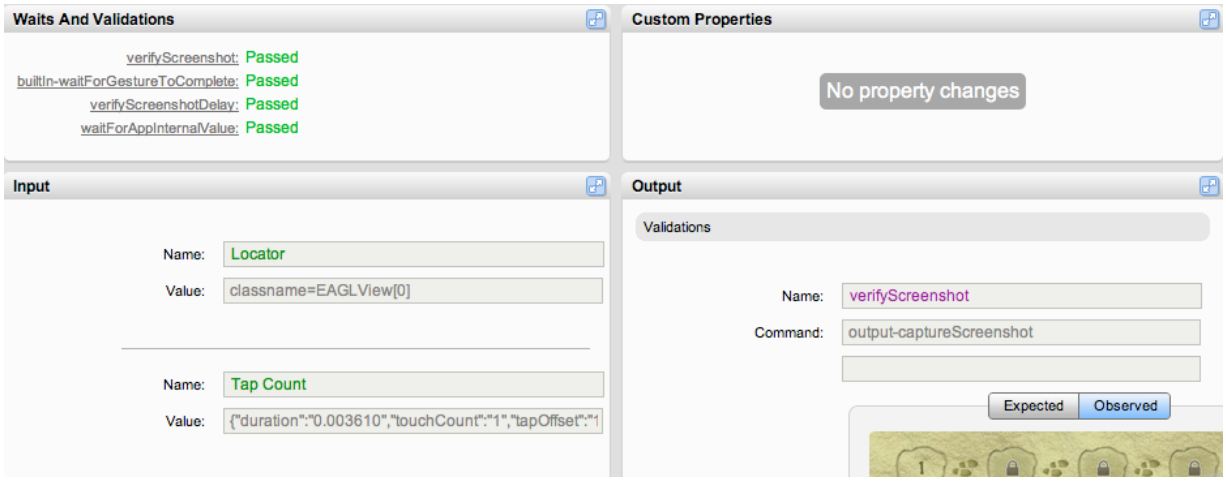

1. Once results are complete, click the clip in the Navigation Tree.



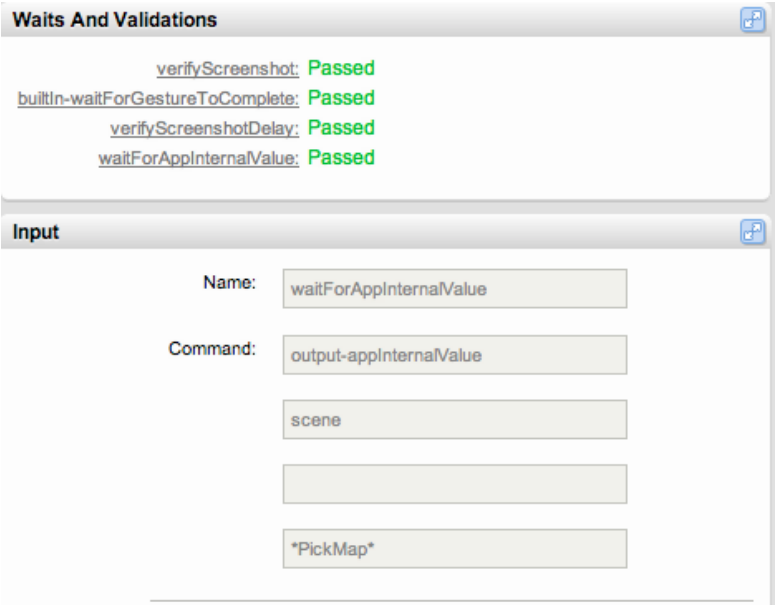
2. In the Waits and Validations section, click the `verifyScrenshotDelay`. This delay was automatically added as part of enabling screenshot validation.



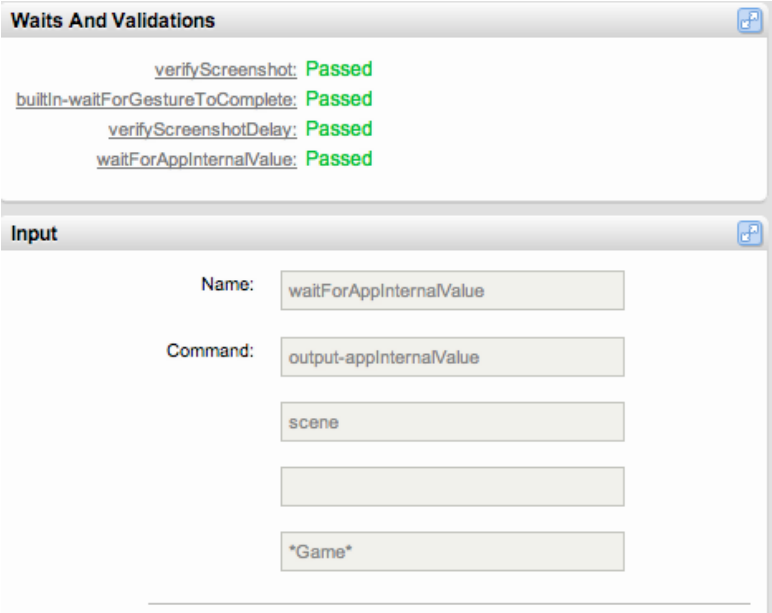

3. Click the first app action, `New Game`. In the result shown below, all of the accessors succeeded.



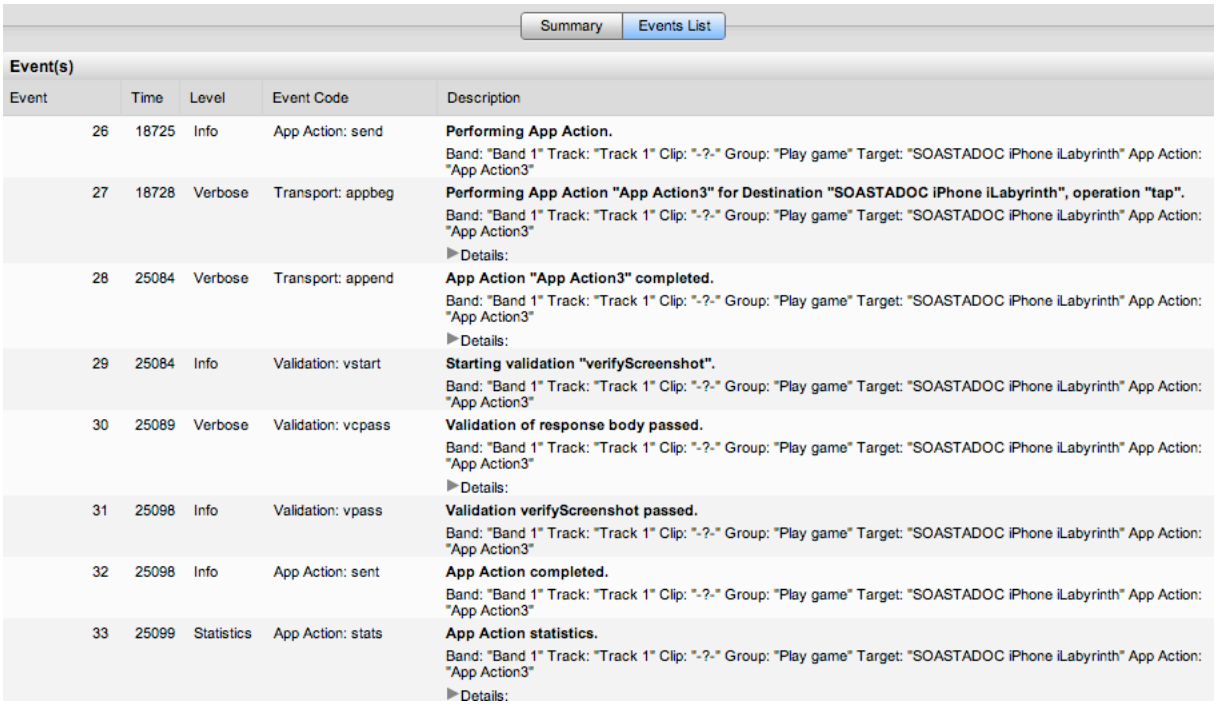
4. Click the waitForAppInternalValue item in the list. The Input node below snaps-to and the wait details are displayed.



5. Next, click the Start Game app action, and not that the waitForAppInternalvalue for the Game scene also passed.




6. For any selection, click the Events List tab in the workspace to examine additional details about that action.



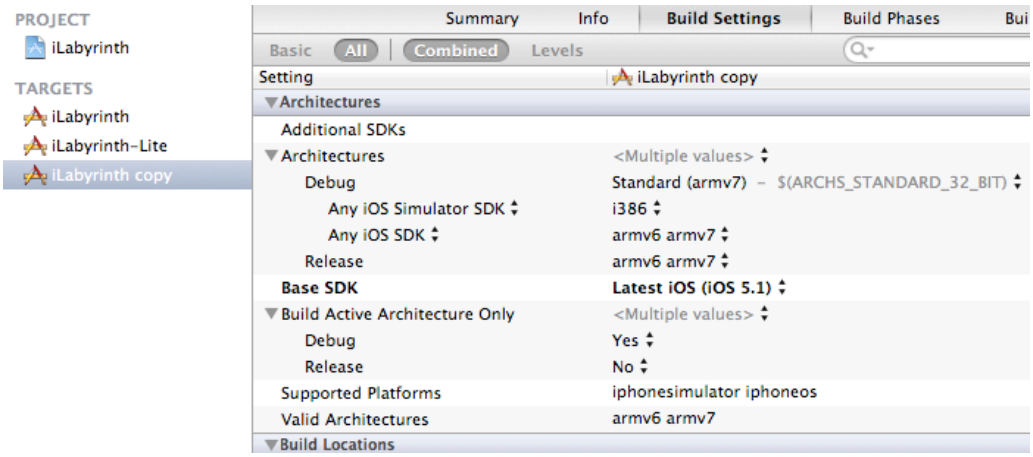

| Event | Time | Level | Event Code | Description |
|---|---|---|---|---|
| 26 | 18725 | Info | App Action: send | **Performing App Action.** Band: "Band 1" Track: "Track 1" Clip: "-?-" Group: "Play game" Target: "SOASTADOC iPhone iLabyrinth" App Action: "App Action3" |
| 27 | 18728 | Verbose | Transport: appbeg | **Performing App Action "App Action3" for Destination "SOASTADOC iPhone iLabyrinth", operation "tap".** Band: "Band 1" Track: "Track 1" Clip: "-?-" Group: "Play game" Target: "SOASTADOC iPhone iLabyrinth" App Action: "App Action3" ▶Details: |
| 28 | 25084 | Verbose | Transport: append | **App Action "App Action3" completed.** Band: "Band 1" Track: "Track 1" Clip: "-?-" Group: "Play game" Target: "SOASTADOC iPhone iLabyrinth" App Action: "App Action3" ▶Details: |
| 29 | 25084 | Info | Validation: vstart | **Starting validation "verifyScreenshot".** Band: "Band 1" Track: "Track 1" Clip: "-?-" Group: "Play game" Target: "SOASTADOC iPhone iLabyrinth" App Action: "App Action3" |
| 30 | 25089 | Verbose | Validation: vcpass | **Validation of response body passed.** Band: "Band 1" Track: "Track 1" Clip: "-?-" Group: "Play game" Target: "SOASTADOC iPhone iLabyrinth" App Action: "App Action3" ▶Details: |
| 31 | 25098 | Info | Validation: vpass | **Validation verifyScreenshot passed.** Band: "Band 1" Track: "Track 1" Clip: "-?-" Group: "Play game" Target: "SOASTADOC iPhone iLabyrinth" App Action: "App Action3" |
| 32 | 25098 | Info | App Action: sent | **App Action completed.** Band: "Band 1" Track: "Track 1" Clip: "-?-" Group: "Play game" Target: "SOASTADOC iPhone iLabyrinth" App Action: "App Action3" |
| 33 | 25099 | Statistics | App Action: stats | **App Action statistics.** Band: "Band 1" Track: "Track 1" Clip: "-?-" Group: "Play game" Target: "SOASTADOC iPhone iLabyrinth" App Action: "App Action3" ▶Details: |

# Appendix: Manually Adding TouchTest™ to the Xcode Project

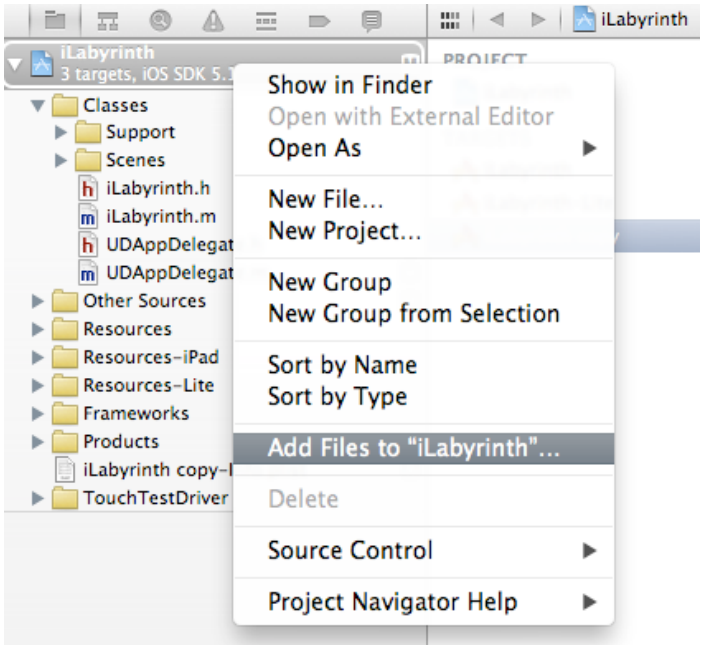
Instructions to manually configure your iLabyrinth Xcode project for TouchTest™ are provided below. If you've already done so using the MakeAppTouchTestable utility it is not necessary to do so again.

To get started, let's first create a duplicate of the project. By using a duplicate, we'll always have the original project as a benchmark.

1. Open the iLabyrinth.xcodeproj file in Xcode.
2. Select the iOS version you wish to build by doing the following:
   a) Select the duplicate "iLabyrinth copy" Target. If this copy doesn't exist create it now.
   b) In the Build Settings, Architectures section, select the Base SDK field.
   c) Select the iOS SDK version to use. For example, Latest iOS (iOS 5.1).

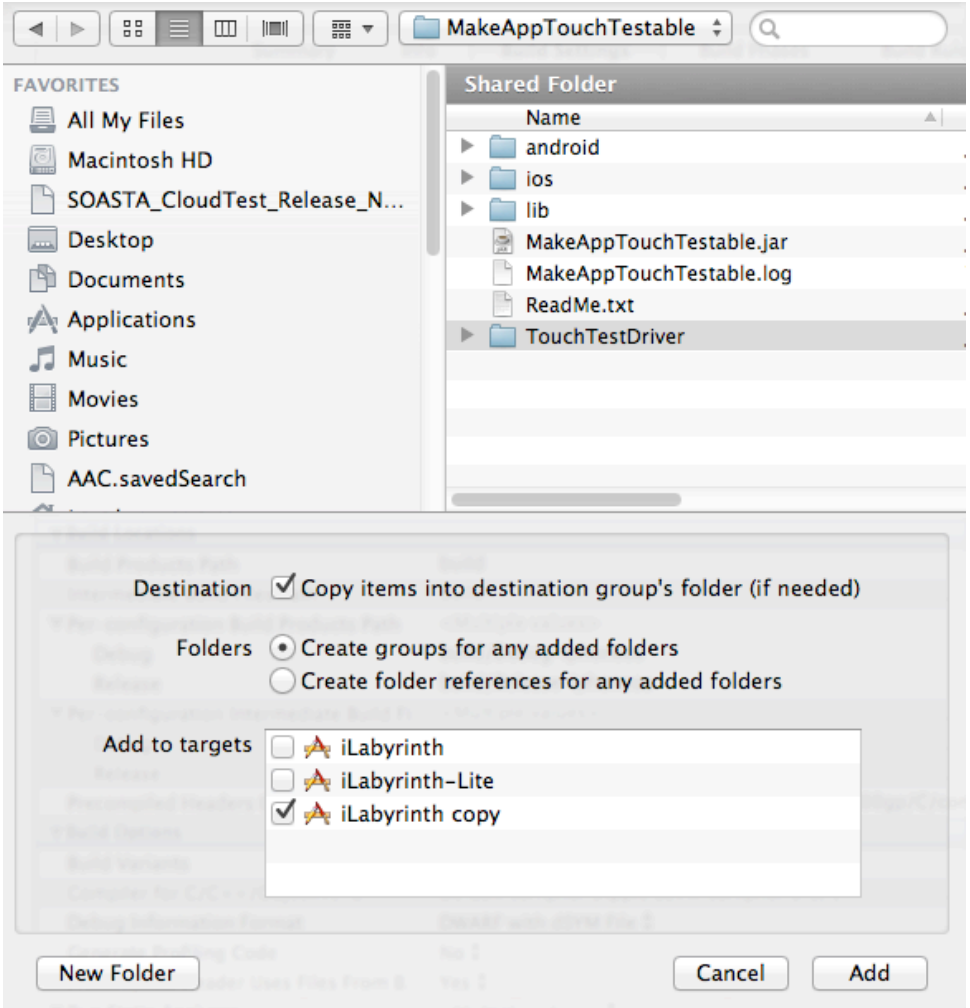


3. Select the Project in the (leftmost) list and then right-click to choose Add Files to “iLabyrinth…”.




4. Navigate to the location where you unarchived MakeAppTouchTestable and select the TouchTestDriver folder.
5. Check the box marked "Copy items into the destination group's folder” (if necessary).

6. Specify the newly-created target from above by checking its box in the Add to targets section in the lower panel.




The example above shows that TouchTest Driver has been selected for addition into the project and the duplicate target, *iLabyrinth copy*, is specified in the Add to targets field.

7. With your project still selected in the project tree on the left, select the target (for example, iLabyrinth copy target) and then select the Build Settings tab.
8. In the Build Settings tab, scroll down to locate the Linking section, and then in the Other Linker Flags field perform the following:

**Note:** It is not necessary to expand the Other Linker Flags field unless you're maintaining different flags for debug and release. If you are maintaining different flags, enter them as necessary. Otherwise, you can click into the field and paste or enter your flag.

- If the user project has the `-all_load` flag already present, add the `-ObjC` flag.

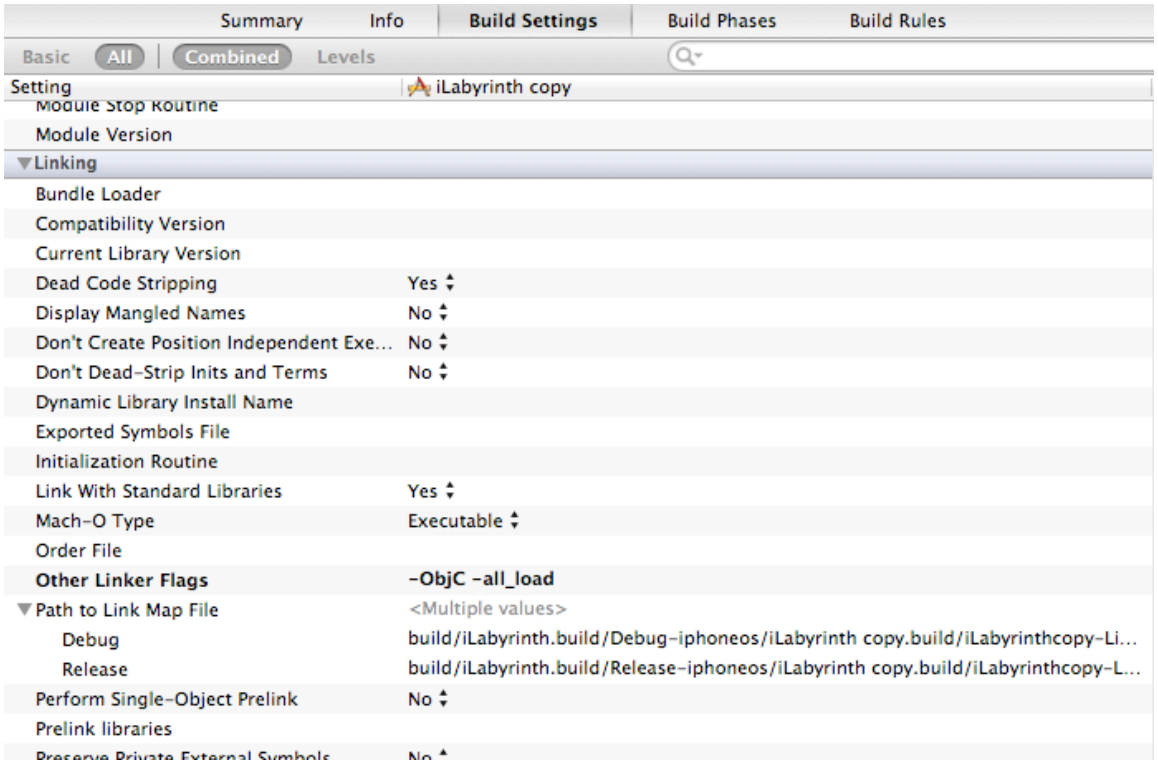
—Or—

If the user project already has the `-ObjC` flag, add `-all_load` flag.

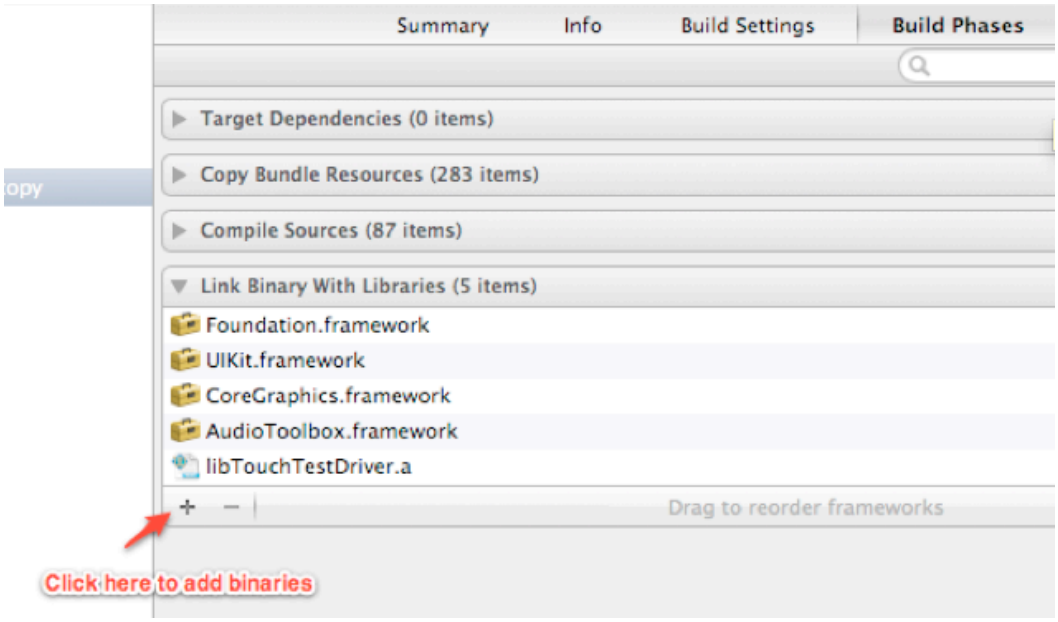
- If both `-all_load` and `-ObjC` flags are present, no change is necessary.
- If none of the above options are fulfilled, then add the `TouchTestDriver/libTouchTestDriver.a` flag.

Example:

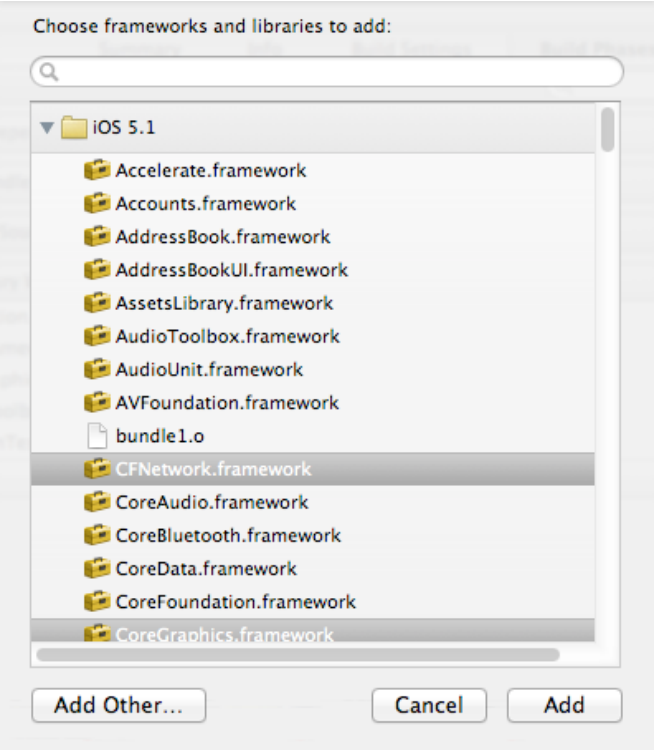
```
-force_load TouchTestDriver/libTouchTestDriver.a
```



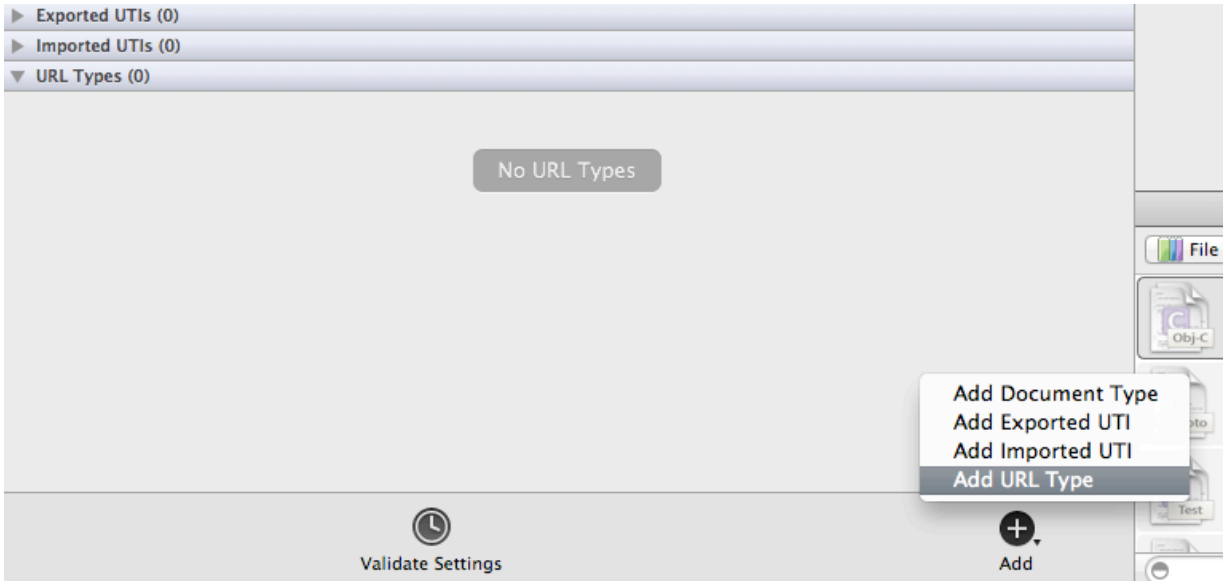
9. Select the Build Phases tab, locate and expand the Link Binary with Libraries section and then click the Plus (+) button.



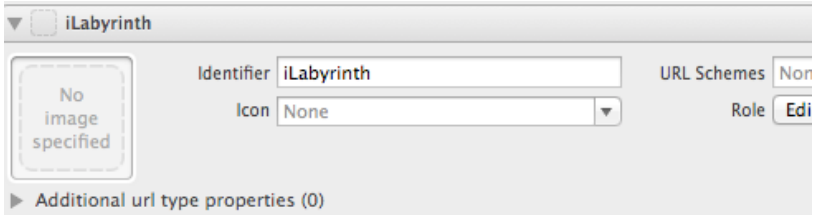

10. In the Choose frameworks and libraries to add: dialog box, add the CoreGraphics.framefork and the CFNetwork.framework.



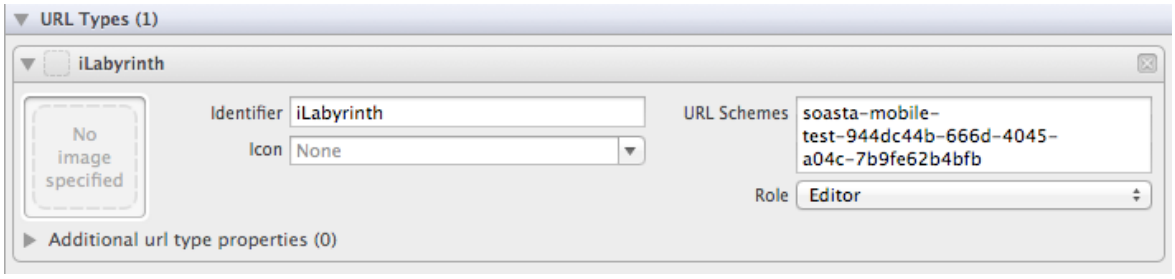
11. Next, select the Info tab, select the URL Types section, and then add a URL Type to the target.



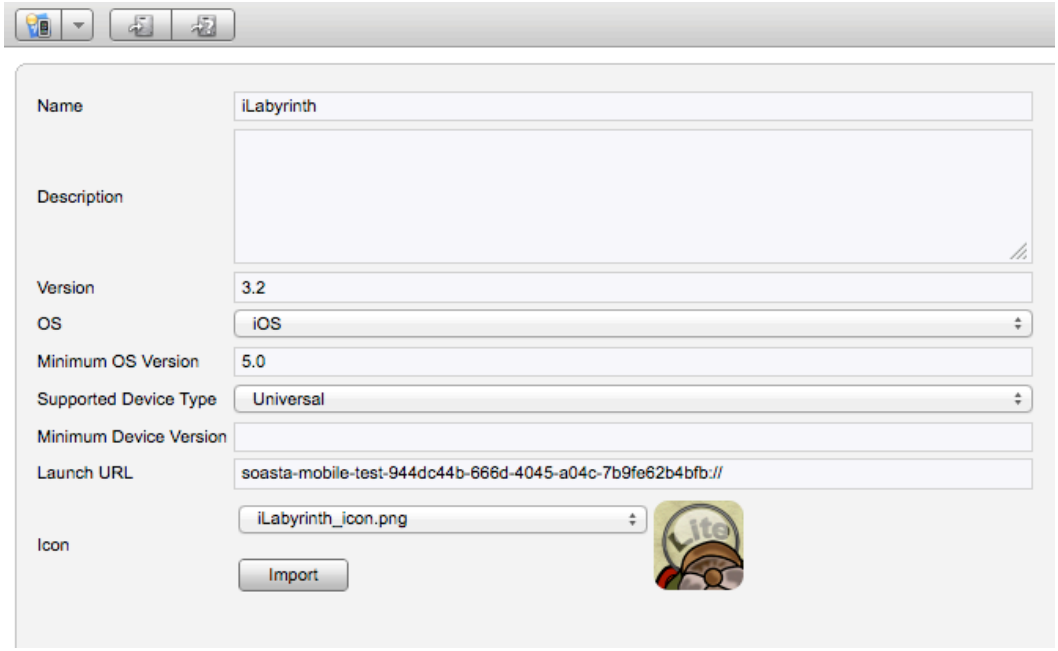

12. Enter the target name as the Identifier. For example, *iLabyrinth*.




13. Enter the URL Scheme.



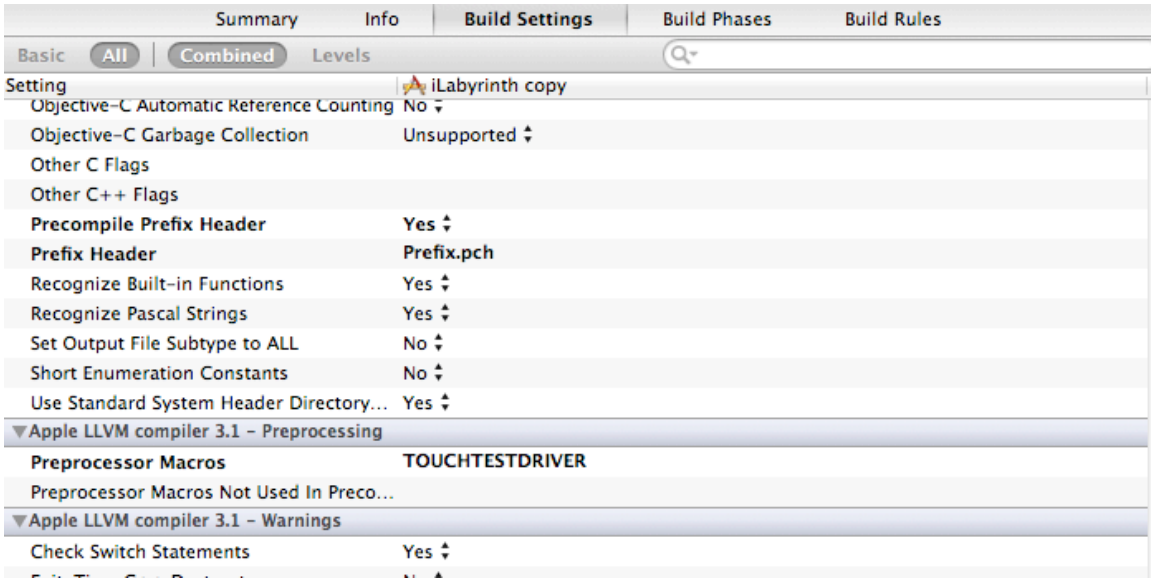
**Note:** In the Xcode URL Schemes field the "soasta-mobile-test-AppID" format is used. The URL Scheme here MUST be the same as the prefix portion of the Launch URL, as specified in the CloudTest® user's Mobile App Object (covered below). For example, `soasta-mobile-test-944dc44b-666d-4045-a04c-7b9fe62b4bfb.`




**TIP:** The automatic method generates this ID in the duplicate target. For the manual method, you can use a command like *uuidgen* in Terminal if a uuid is desired. However, only requirement is that the Xcode URL Scheme matches the one in CloudTest's Mobile App form (with the protocol syntax `://` added).

14. Next, select the Build Settings tab a second time, and locate the Apple LLVM compiler 3.1 section (shown below).

15. Click the Preprocessor Macros heading and the Plus + icon to add TOUCHTESTDRIVER.




Finally, we will make some source code modifications to the project's main.m and AppDelegate files.

16. In main.m, include the required header file by adding the following code:

```
#ifdef TOUCHTESTDRIVER
#import "TouchTestDriver.h"
#endif
```

17. Next, also in main.m, include the following code in the main() function. This code will initialize TouchTestDriver.

```
#ifdef TOUCHTESTDRIVER
  [TouchTestDriver initDriver];
#endif
```

Your changes in the prior two steps revising main.m (or main.mm) should look like this:

```
//
//  main.m
//  iLabyrinth
//
//  Created by Rolandas Razma on 5/12/10.
//  Copyright 2010 UD7. All rights reserved.
//

#ifdef TOUCHTESTDRIVER
#import "TouchTestDriver.h"
#endif
#import <UIKit/UIKit.h>


int main(int argc, char *argv[]) {
  #ifdef TOUCHTESTDRIVER
  [TouchTestDriver initDriver];
  #endif

    @autoreleasepool {
        return UIApplicationMain(argc, argv, nil, @"UDAppDelegate");
    }
}
```

18. In the AppDelegate file, include the required header by adding the following code:

```
#ifdef TOUCHTESTDRIVER
#import "TouchTestDriver.h"
#endif
```

Your changes to AppDelegate should look like this:

```
//
//  UDAppDelegate.m
//  iLabyrinth
//
//  Created by Rolandas Razma on 5/12/10.
//  Copyright 2010 UD7. All rights reserved.
//

#ifdef TOUCHTESTDRIVER
#import "TouchTestDriver.h"
#endif
#import "UDAppDelegate.h"
#import "iLabyrinth.h"
#import "UDMenuScene.h"
#import "UDGameScene.h"
#import "SimpleAudioEngine.h"
#import "UDGameEndScene.h"
#import "CCDirectorIOS.h"
```

19. Then, include the following code to either the `handleOpenURL` or the `openURL` method, whichever exists in AppDelegate (they are mutually exclusive).

```
#ifdef TOUCHTESTDRIVER
  [TouchTestDriver startSession:url];
#endif
```

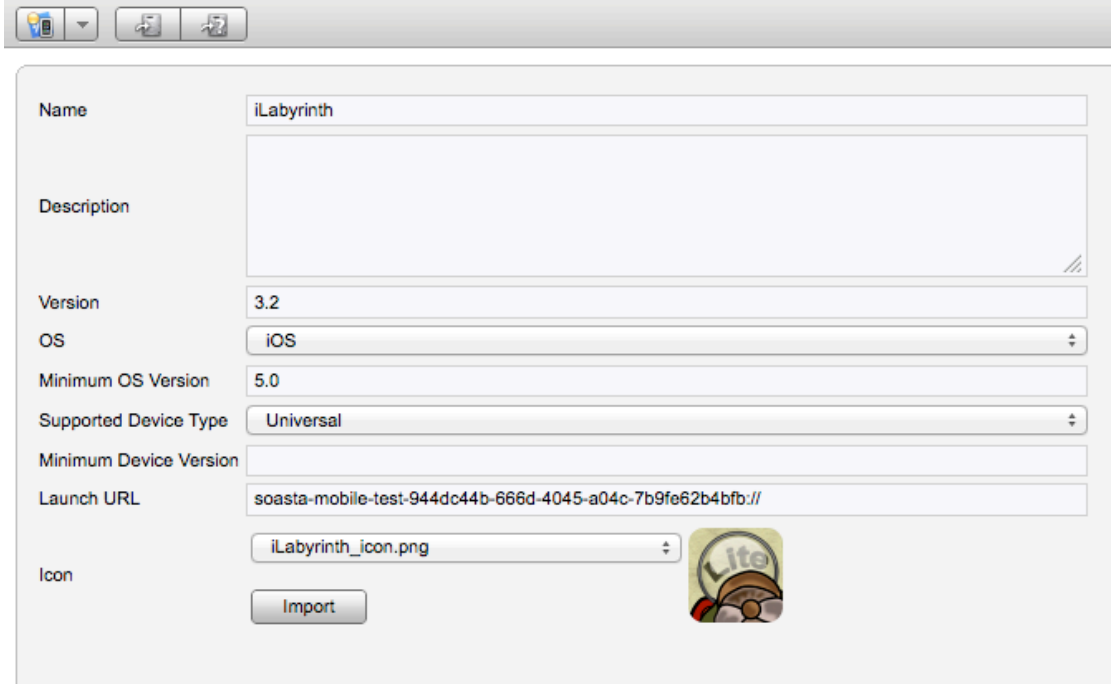
**Note:** If neither of these methods exist, please add the following method (the text below can be copied and pasted):

```
- (BOOL)application:(UIApplication *)application openURL:(NSURL *)url sourceApplication:
(NSString *)sourceApplication annotation:(id)annotation

{

  if (!url)

  {

    return NO;

  }


  #ifdef TOUCHTESTDRIVER

    [TouchTestDriver startSession:url];

  #endif


  return YES;

}
```

## Adding a Mobile App to CloudTest® Manually

If you didn't use the MakeAppTouchTestable utility, it will be necessary to manually add a mobile app to CloudTest®.

1. To do so, 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 Xcode project name. For the iLabyrinth example, you will want to enter *iLabyrinth copy* if you're also going to also use *iLabyrinth*.




3. Optionally, enter a description and an app version number. Version number will generally match Xcode project details.
4. Only iOS is supported currently. For this release, TouchTest™ supports iOS 5.0+ versions only.
5. Set the Supported Device Type to Universal (if your tests will include various iOS devices), or choose iPhone or iPad (whichever applies).
6. In the Launch URL field, provide the unique URL Scheme you defined in XCode, plus any additional arguments relevant to your mobile app.

   For example,

   ```
   soasta-mobile-test-944dc44b-666d-4045-a04c-7b9fe62b4bfb://
   key1=value1&key2=value2&key3=value3
   ```

where `soasta-mobile-test-appID://` is the name of your mobile app including the `://` and additional arguments are in the form `key1=value1&key2=value2`.

**Note:** Without a correctly formed Launch URL testing will not happen.

7. 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.

8. Click Save to create this mobile app object in CloudTest® .

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