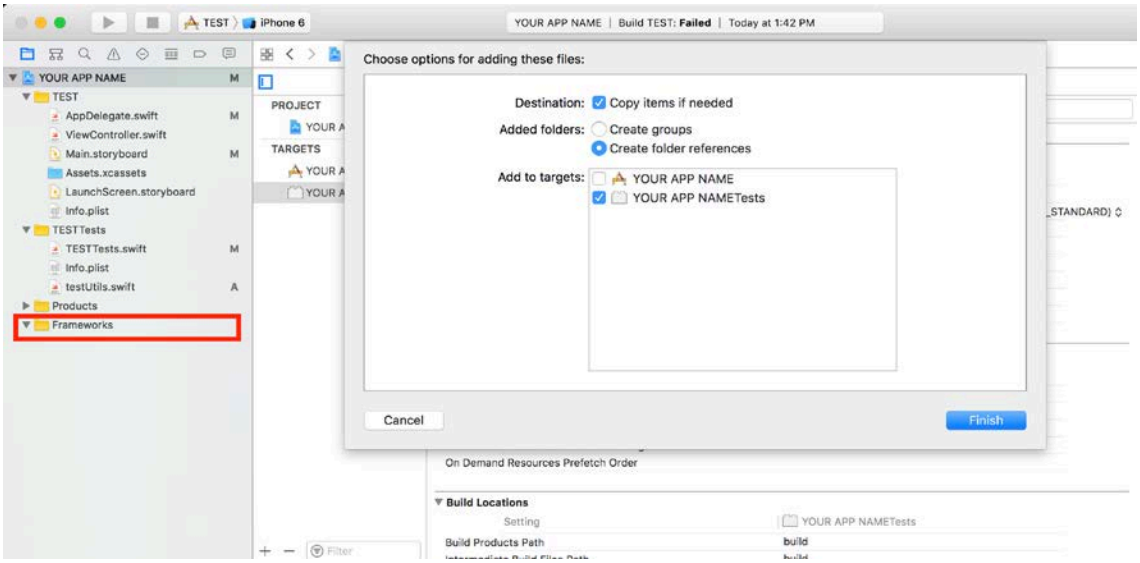
Installing Attest Framework to iOS App for Unit Testing

A step-by-step guide to set up Deque's Attest Framework in your app

For Automated Testing in Swift:

1. Add the framework to your Test Suite:

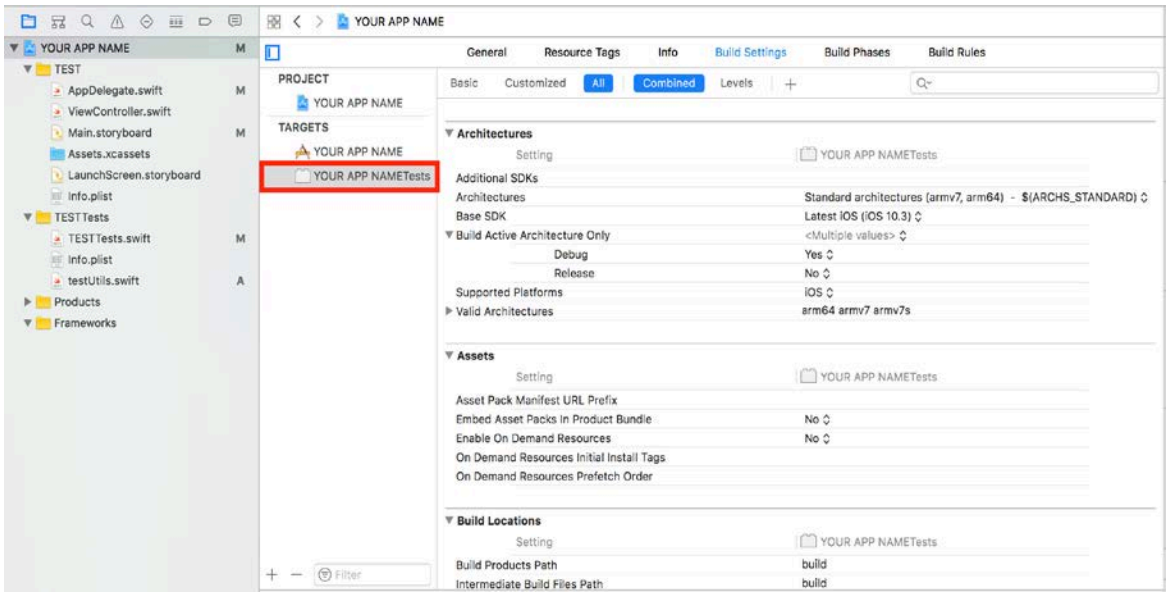
   With your app open in XCode, drag the “Attest.framework” file into the “Frameworks” folder of the app (located in the file selector). A pop up box will appear. Confirm that “Copy Items if needed,” “Create folder references,” and the Unit Test Target for your app are all selected (do not select the app itself!). It should look identical to the image below. Select “Finish” to import the framework.



Step 1: Add the Attest Framework to your Test Suite

2. Navigate to the Unit Test Target settings:

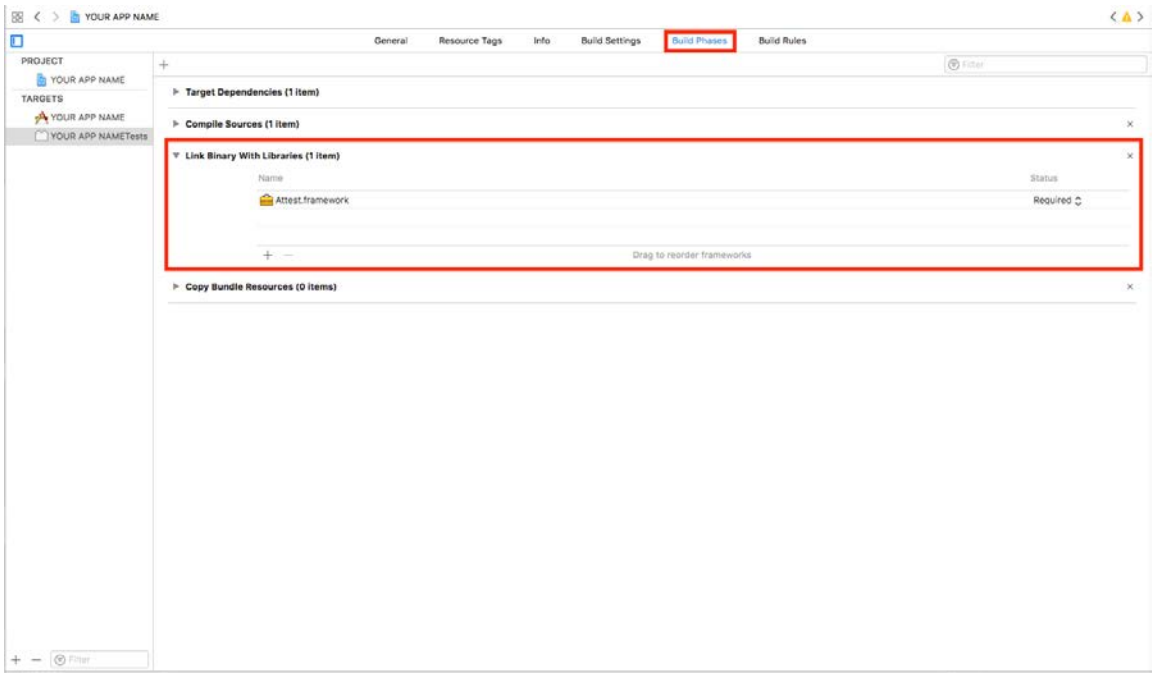
   Next, select the xcodeproj file of your app from the file selector. A screen with the tabs “Info” and “Build Settings” will appear, with your app name listed under “PROJECTS” on the left-hand side. Select the unit test target under “TARGETS,” as shown in the image below.



Step 2: Select unit tests under TARGETS

3. Confirm that the Attest.framework was properly added:

   Select the “Build Phases” tab at the top of the screen. Expand the “Link Binary With Libraries” section if it is not already expanded. Check that “Attest.framework” is already in the “Link Binary with Libraries” section. If it is not, drag and drop it into this section from the file selector.



Step 3: Confirm the Attest Framework is already properly linked into your Unit Tests

4. Confirm that your app builds and can run the test suite as normal.

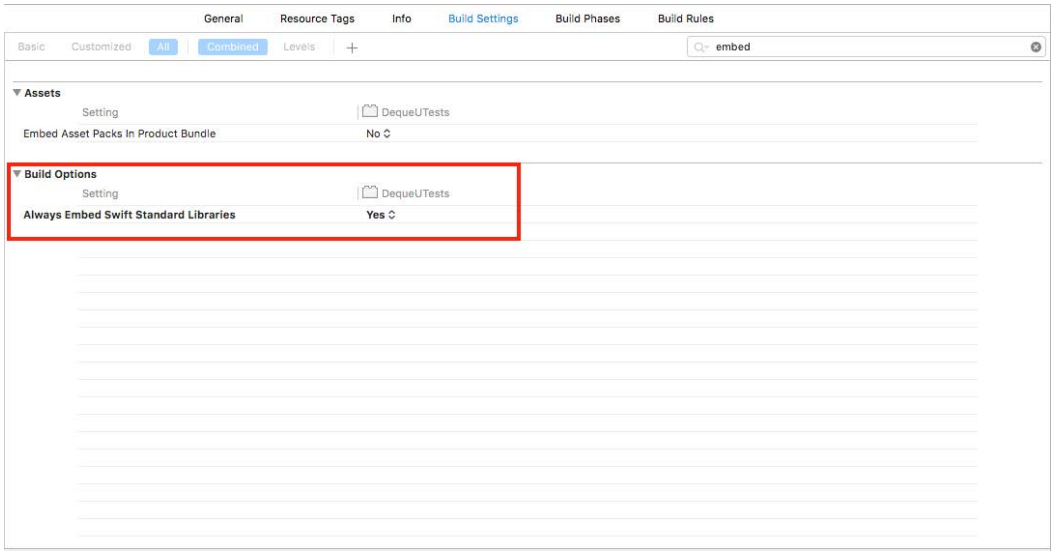
For Automated Testing in Objective-C:

1. Please follow steps 1-3 in “For Automated Testing in Swift.” Note that the app will not build – this is normal. To allow this framework to work properly with Objective-C, update the Build Settings to “Always Embed Swift Standard Libraries”:

    a. Navigate to the Unit Test Target settings (per Step 2 in “For Automated Testing in Swift).

    b. Click on the “Build Settings” tab.

    c. In the search box, type “embed.” Under “Build Options,” change “Always Embed Swift Standard Libraries” to “Yes.”



Step 1c: Update “Always Embed Swift Standard Libraries” to “Yes.”

    d. Confirm that your app builds and can run the test suite as normal.

Running the Attest Framework in Automated Tests (Example in Swift):

To run the Attest Framework during automated testing, a few lines of code will need to be added to your unit tests. Below are some examples of how to use the Attest Framework in your automated tests. We used XCTest to demonstrate.

The following code sample is a simple example of using the Attest Framework.

```
import XCTest
import Attest

@testable import TestApp

class TestAppTests: XCTestCase {

    func testAccessibility() {
        let viewController = UIViewController.initialFrom(storyBoardName: "Main")

        viewController.forceLoad()

        Attest.that(viewController: viewController).isAccessible()
    }
```

First, we call "initialFrom(storyBoardName:) to store Main.storyboard's initial View Controller in as a variable named "viewController." This function is part of an extension we added to UIViewController, available for your use. More information regarding this function and the UIViewController extensions can be found in the Attest Framework documentation.

Next, we call viewController.forceLoad() to ensure that all views have correctly loaded. There are some interesting things that iOS does not do in a unit testing environment regarding view loading. We have added extensions to UIViewController and UIView to help mitigate many of these issues. To ensure that the Attest Framework is used to its full potential, we highly recommend calling this "forceLoad()" function (or a variation of it) to your unit tests when using our framework. See the Attest Framework documentation regarding the UIView and UIViewController extensions for more information.

Finally, we call Attest.that(viewController:).isAccessible() to ensure that all views in viewController are accessible. Attest.that(viewController:) expects a UIViewController to test. There are other versions of Attest.that() that will be shown in the other examples, and can be found in the Attest Framework documentation. isAccessible() has an optional parameter (resultConsumer) that will be explained in other examples. If no parameter is defined in isAccessible(), it will assume that all views in the view controller should be completely accessible. Any view that is found to be inaccessible will fail the test case, and a description of the violation will be in the log.

The following examples will explain the optional parameter of isAccessible().

isAccessible()'s parameter, resultConsumer, is important to use if you expect accessibility violations (or "best practice" violations, or if you would just like to check the number of passing views). We will give a few different examples of how to make sure that the violations found are the ones that you expected.

Note: In these examples, we have removed the forceLoad() function for clarity.

```
func testAccessibility_2() {
    Attest.that(storyBoardName: "Main").isAccessible({(result:Attest.Result) in

        XCTAssertEqual(1, result.violations.count)

        XCTAssertEqual(RuleID.ColorContrast, result.violations.first?.ruleId)

        XCTAssertEqual(2, result.violations.first?.nodes.count)
    })
```

Notice in this example, we use Attest.that(storyBoardName:) instead of Attest.that(viewController:). This function retrieves the initial UIViewController from the storyboard listed and returns an initialized Assert, which is the class needed to call isAccessible(). This version of the Attest.that() function can retrieve a specific UIViewController in a Bundle as well; however, these are set to nil by default if no parameter is listed. See the Attest Framework documentation for more information on correct usage of the Attest.that() functions.

The parameter resultConsumer of the isAccessible() function is a function expecting an Attest.Result as the parameter. In the above example, we expect the initial UIViewController in storyboard "Main" to violate one Rule, as seen in the first call to XCTAssertEqual. The expected Rule being violated is the Color Contrast Rule, as shown in the second call to XCTAssertEqual. There will be two views in the initial UIViewController that are expected to violate this Rule, as seen in the third call to XCTAssertEqual. If there are additional violations, the test case will fail, and a description of the violation can be found in the log.

If you expect more than one Rule to be in violation, you can include a switch statement in your resultConsumer function, as seen in the following example:

```
func testAccessibility_3() {
    Attest.that(storyBoardName: "Main").isAccessible({(result:Attest.Result) in

        for violation in result.violations {
            switch violation.ruleId {
                case RuleID.DontIntersect:
                    XCTAssertEqual(4, violation.nodes.count, violation.description)

                case RuleID.DynamicType:
                    XCTAssertEqual(2, violation.nodes.count, violation.description)

                default:
                    XCTAssertEqual(0, violation.nodes.count, violation.description)
            }
        }
    })
```

In this example, the initial UIViewController in Main.storyboard is expected to have 4 violations of the DontIntersect Rule and 2 violations of the DynamicType Rule. The “default” Rule is listed to have 0 violations, since we do not expect any other Rules to be in violation. If this is not the case, and other Rules are violated, the test case will fail and a description of the violation(s) can be found in the log. See the Attest Framework documentation for more information on the different Rules and proper usage of the Attest.that() functions.

Running the Attest Framework in Automated Tests (Example in Objective-C):

To run the Attest Framework during automated testing, a few lines of code will need to be added to your unit tests. Below are some examples of how to use the Attest Framework in your automated tests. We used XCTest to demonstrate.

The following code sample is a simple Objective-C example of using the Attest Framework.

```
#import <XCTest/XCTest.h>
#import "ViewController.h"
#import <Attest/Attest-Swift.h>

@interface ViewController_test : XCTestCase {}

@end

@implementation ViewController_test

- (void)testAccessibility {

    ViewController* viewController = [UIViewController initialFromStoryBoardName:@"Main"];

    [viewController forceLoad];

    Result* result = [[Attest thatWithViewController:viewController] isAccessible];

    XCTAssertEqual(result.violations.count, 0);
```

First, we call initialFromStoryBoardName: to store Main.storyboard's initial View Controller in a variable named "viewController." This function is part of an extension we added to UIViewController, available for your use. More information regarding this function and the UIViewController extensions can be found in the Attest Framework documentation.

Next, we call [viewController forceLoad] to ensure that all views have correctly loaded. There are some interesting things that iOS does not do in a unit testing environment regarding view loading. We have added extensions to UIViewController and UIView to help mitigate many of these issues. To ensure that the Attest Framework is used to its full potential, we highly recommend calling this "forceLoad" function (or a variation of it) to your unit tests when using our framework. See the Attest Framework documentation regarding the UIView and UIViewController extensions for more information.

Finally, we call [[Attest thatWithViewController:] isAccessible] to ensure that all views in viewController are accessible. [Attest thatWithViewController:] expects a UIViewController to

test. There are other versions of [Attest that] that will be shown in the other examples, and can be found in the Attest Framework documentation. isAccessible returns a Result*. This Result can be printed or parsed through with Asserts to ensure that each view in the UIViewController is accessible. In this example, we expect that there will be no violations, so we used an XCTAssert to assure that the UIViewController has no violating views. We also decided to see a detailed list of the Result. Printing the Result's description shows the view hierarchy and lists the views in violation of WCAG2.0 guidelines, the views that do not pass WCAG2.0 best practices, the views that pass WCAG2.0 guidelines, and the views that are not applicable to certain Rules.

The following examples will explain the many things that can be done with the returned Result from isAccessible.

Note: In these examples, we have removed the forceLoad function for clarity.

```
- (void)testAccessibility2 {
    Result* result = [[Attest thatWithStoryBoardName:@"Main"
                                        viewControllerID:NULL bundle:NULL] isAccessible];

    NSLog(@"%@", result.description);

    XCTAssertEqual(result.violations.count, 1);
    XCTAssertEqual(result.violations.firstObject.nodes.count, 3);
    XCTAssertEqual(result.violations.firstObject.ruleId, RuleIDDynamicType);
```

Notice in this example, we use [Attest thatWithStoryBoardName] instead of [Attest thatWithViewController]. When the other parameters are NULL, it retrieves the initial UIViewController from the storyboard listed and returns an initialized Assert, which is the class needed to call isAccessible. This version of the [Attest that] function can retrieve a specific UIViewController in a Bundle as well; however, these are set to NULL in this example. See the Attest Framework documentation for more information on correct usage of the [Attest that] functions.

With the Result* returned, we can use XCTAsserts if we expect there to be violations. In this example, we expect the initial view controller of Main.storyboard to violate one Rule, as shown in the first call to XCTAssertEqual. We expect three views in this view controller to violate this Rule, as shown in the second call to XCTAssertEqual. The Rule being violated should be the Dynamic Type Rule, so we check that with the third call to XCTAssertEqual. If there are additional violations, the test case will fail, and a description can be found in the log.

If you expect more than one Rule to be in violation, you can use a switch statement to parse through each Rule, as seen in the following example:

```
- (void)testAccessibility3 {
    Result* result = [[Attest thatWithView:view] isAccessible];

    for (Entry *violation in result.violations) {
        switch(violation.ruleId) {

            case RuleIDDontIntersect:
                XCTAssertEqual(violation.nodes.count, 4, @"%@", violation.description);
                break;

            case RuleIDDynamicType:
                XCTAssertEqual(violation.nodes.count, 2, @"%@", violation.description);
                break;

            default:
                XCTAssertEqual(violation.nodes.count, 0, @"%@", violation.description);
                break;
        }
```

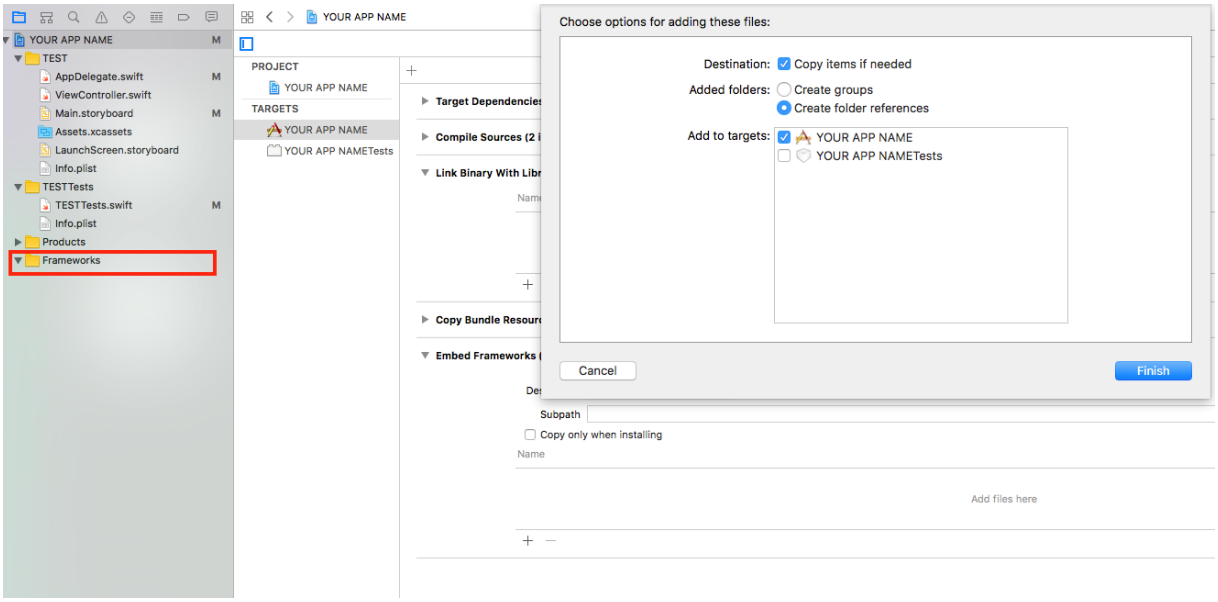
In this example, the view named “view” is expected to have 4 violations of the DontIntersect Rule and 2 violations of the DynamicType Rule. The “default” Rule is listed to have 0 violations, since we do not expect any other Rules to be in violation. If this is not the case and other Rules are violated, the test case will fail and a description of the violation(s) can be found in the log. See the Attest Framework documentation for more information on the different Rules and proper usage of the [Attest that] functions.

For Manual Testing in Swift:

IMPORTANT: In order for highlighting to work from the Attest UI Client, VoiceOver **MUST** be turned on. Highlighting will not work on Simulators.

1. Add the framework to your app:

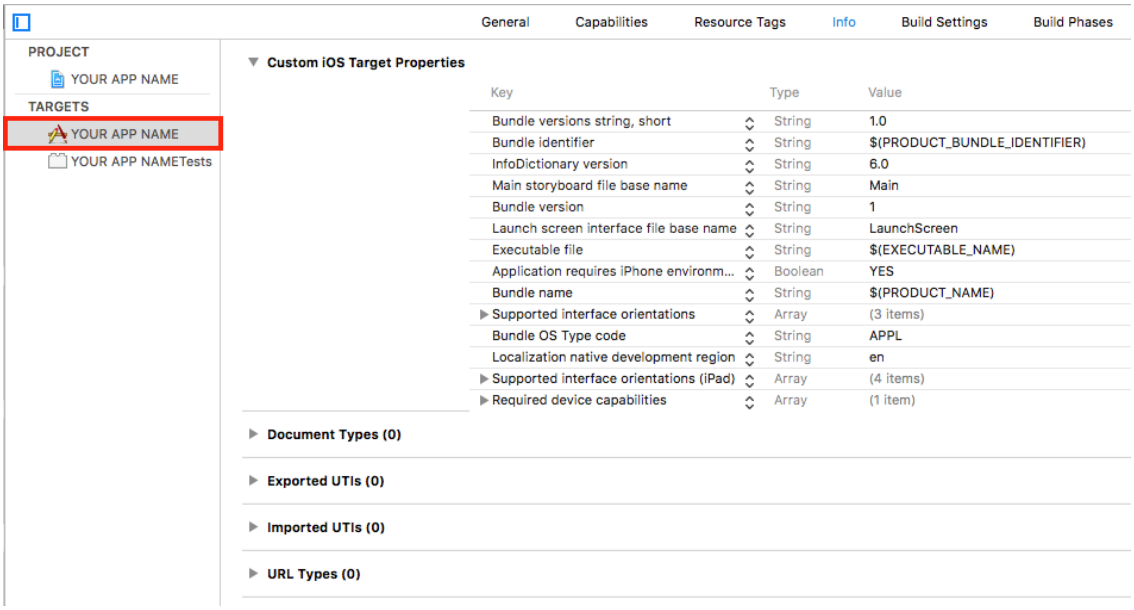
   With your app open in XCode, drag the "Attest.Framework" file into the "Frameworks" folder of your app (located in the file selector). A pop up box will appear. Confirm that "Copy Items if needed," "Create folder references," and the target for your app are all selected. It should look identical to the image below. Select "Finish" to import the framework.



Step 1: Add the Attest Framework to your app

2. Navigate to your app settings:

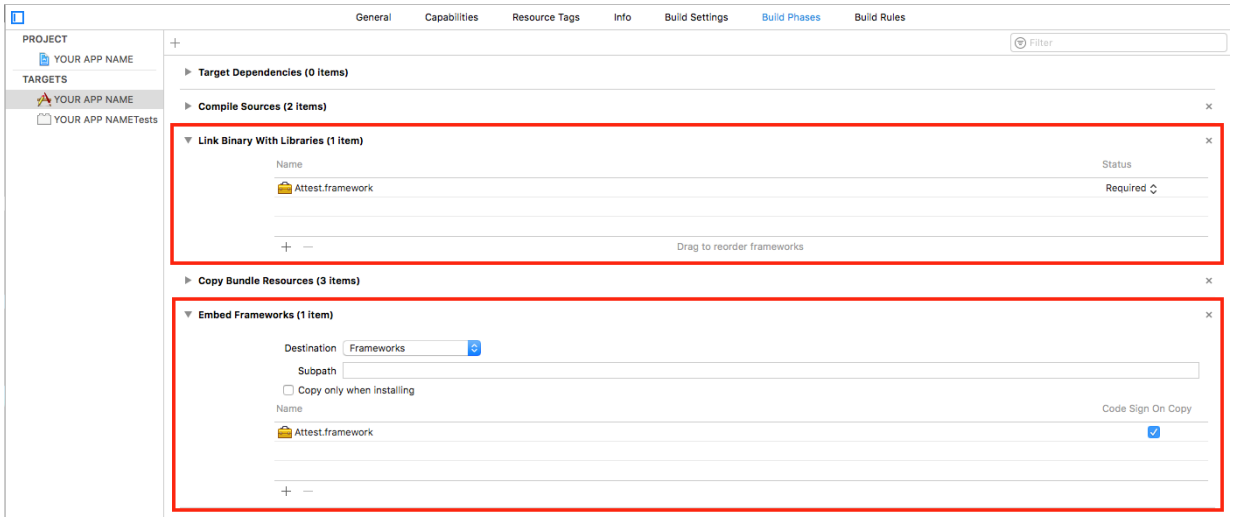
   Next, select the xcodeproj file of your app from the file selector. A screen with the tabs "Info" and "Build Settings" will appear, with your app name listed under "PROJECTS" on the left-hand side. Select your app name under "TARGETS," as shown in the image below.



Step 2: Select your app under "TARGETS"

3. Add the framework as an Embedded Framework:

   Select the "Build Phases" tab at the top of the screen.
   a. Confirm that "Attest.framework" is already in the "Link Binary with Libraries" section. If it is not, drag and drop it into this section from the file selector.
   b. Expand the "Embed Frameworks" section and drag the Attest Framework into it. It should look similar to the image below.



Step 3: Add the Attest Framework as an Embedded Framework

4. Confirm that your app builds and can run as normal.

5. Next, open the “AppDelegate.swift” file in your app.
    a. In this file, import the Attest Framework:

```
import Attest
```

    b. In the function applicationWillResignActive(_ application: UIApplication), add the following line of code:

```
Attest.stopServer()
```

    c. In the function applicationDidBecomeActive(_ application: UIApplication), add the following line of code:

```
Attest.startServer(48484)
```

    Note: the number “48484” may change, depending on your setup. Attest.startServer(_ port: UInt) asks for a port number as a parameter. In most cases, this number will be 48484.

    The AppDelegate file should look similar to this when you are done:

```
import UIKit
import Attest

@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate {

  func applicationWillResignActive(_ application: UIApplication) {

    Attest.stopServer()
  }

  func applicationDidBecomeActive(_ application: UIApplication) {

    Attest.startServer(48484)
  }
```

6. Confirm that your app builds and can run as normal.

For Manual Testing in Objective-C:

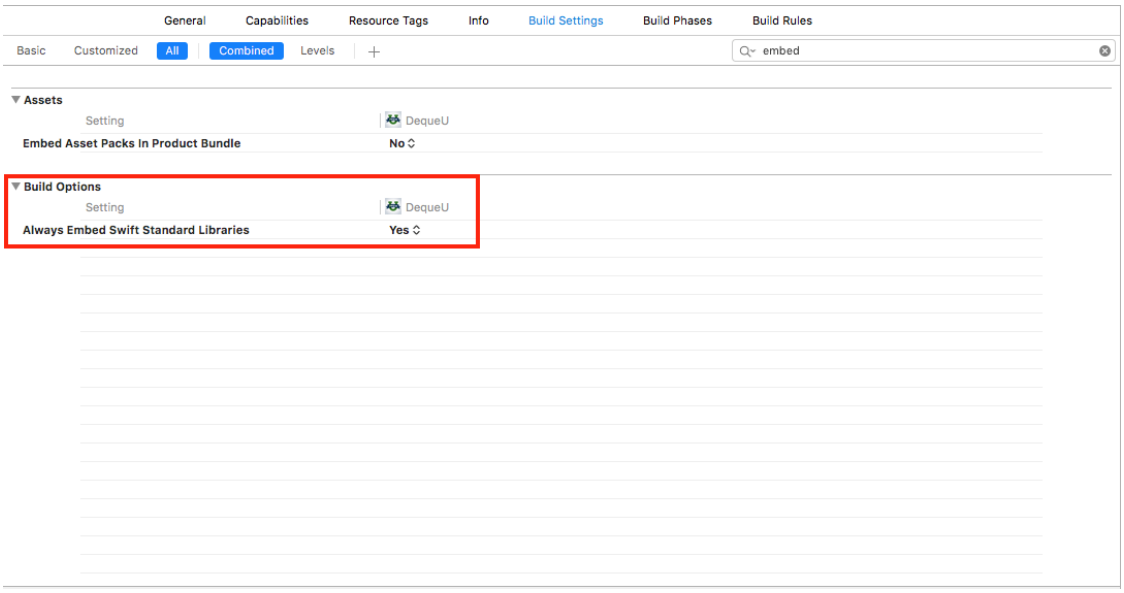
IMPORTANT: In order for highlighting to work from the Attest UI Client, VoiceOver **MUST** be turned on. Highlighting will not work on Simulators.

1. Please follow steps 1-3 in "For Manual Testing in Swift." Note that the app will not build – this is normal. To allow the framework to work properly with Objective-C, update the Build Settings to "Always Embed Swift Standard Libraries":

    a. Navigate to the App Target settings (per Step 2 in "For Manual Testing in Swift").

    b. Click on the "Build Settings" tab.

    c. In the search box, type "embed." Under "Build Options," change "Always Embed Swift Standard Libraries" to "Yes."



Step 1c: Update "Always Embed Swift Standard Libraries" to "Yes."

    d. Confirm that your app builds and can run the test suite as normal.

2. Next, open the "AppDelegate.m" file in your app.

    a. In this file, import the Attest Framework:

```
@import Attest;
```

    b. In the function –(void)applicationWillResignActive:(UIApplication *)application, add the

```
[Attest stopServer];
```

following line of code:

c. In the function –(void)applicationDidBecomeActive:(UIApplication *)application, add the following line of code:

```
[Attest startServer:48484];
```

Note: the number "48484" may change, depending on your setup. [Attest startServer(_ port: UInt)] asks for a port number as a parameter. In most cases, this number will be 48484.

The AppDelegate file should look similar to this when you are done:

```
@import Attest;

@implementation DQAppDelegate

- (void)applicationWillResignActive:(UIApplication *)application {
    [Attest stopServer];
}

- (void)applicationDidBecomeActive:(UIApplication *)application {
    [Attest startServer:48484];
}
```

3. Confirm that your app builds and can run as normal.