Running Tests and Viewing Results

Running tests and viewing the results is easy to do using the Xcode test navigator, as you saw in Quick Start. There are several additional interactive ways to run tests. Xcode runs tests based on what test targets are included and enabled in a scheme. The test navigator allows you to directly control which test targets, classes, and methods are included, enabled, or disabled in a scheme without having to use the scheme editor.

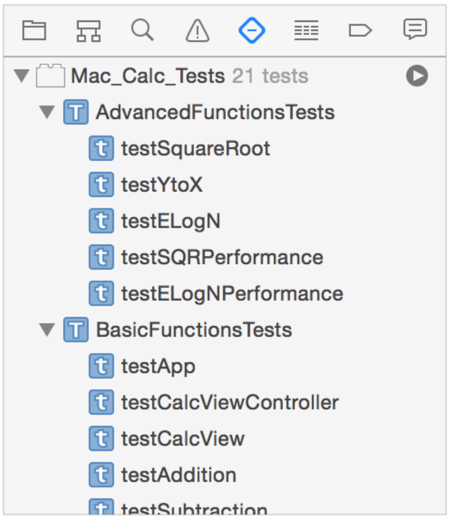
Commands for Running Tests

The test navigator provides you an easy way to run tests as part of your programming workflow. Tests can also be run either directly from the source editor or using the Product menu.

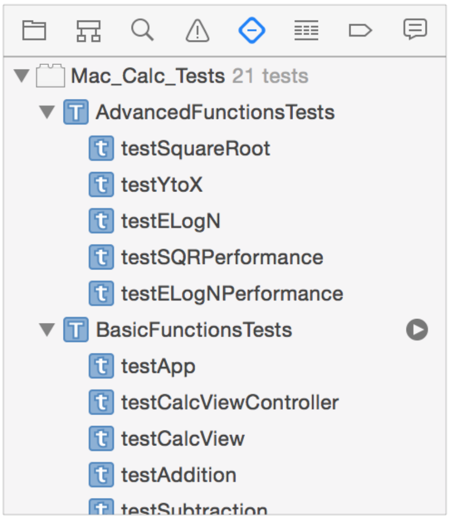
Using the Test Navigator

When you hold the pointer over a bundle, class, or method name in the test navigator, a Run button appears. You can run one test, all the tests in a class, or all the tests in a bundle depending on where you hold the pointer in the test navigator list.

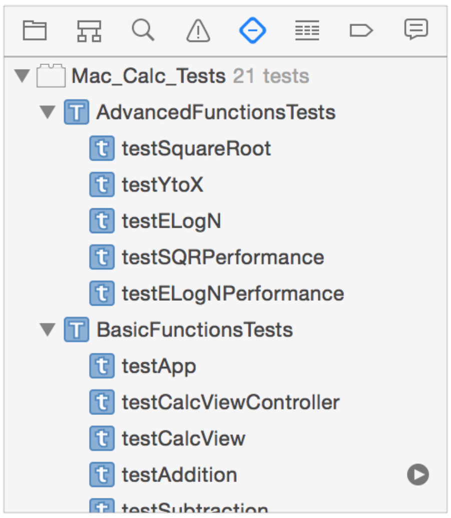
- To run all tests in a bundle, hold the pointer over the test bundle name and click the Run button that appears on the right.



- To run all tests in a class, hold the pointer over the class name and click the Run button that appears on the right.



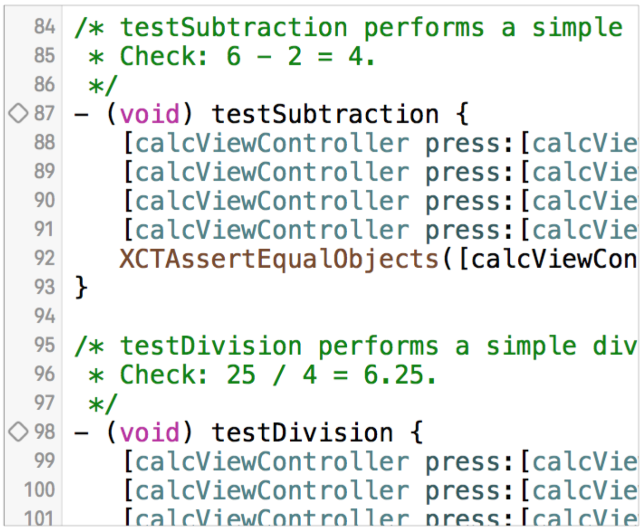
- To run a single test, hold the pointer over the test name and click the Run button that appears on the right.



Mac_Calc_Tests 21 tests
AdvancedFunctionsTests
testSquareRoot
testYtoX
testELogN
testSQRPerformance
testELogNPerformance
BasicFunctionsTests
testApp
testCalcViewController
testCalcView
testAddition
testSubtraction

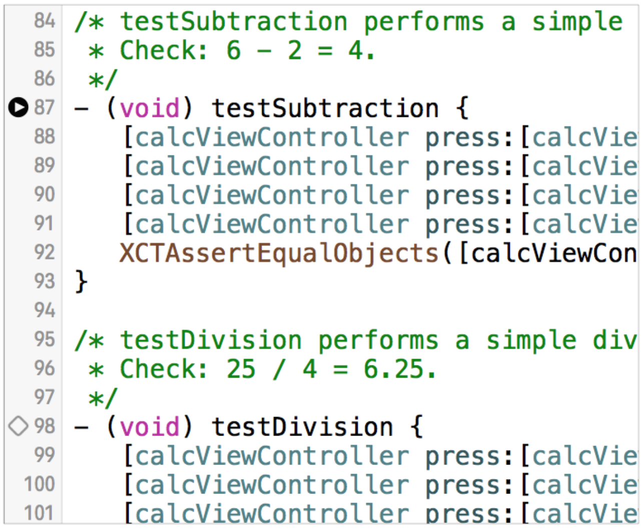
Using the Source Editor

When you have a test class open in the source editor, a clear indicator appears in the gutter next to each test method name. Holding the pointer over the indicator displays a Run button. Clicking the Run button runs the test method, and the indicator then displays the pass or fail status of a test. Holding the pointer over the indicator will again display the Run button to repeat the test. This mechanism always runs just the one test at a time.



```
/* testSubtraction performs a simple
 * Check: 6 - 2 = 4.
 */
- (void) testSubtraction {
    [calcViewController press:[calcVie
    [calcViewController press:[calcVie
    [calcViewController press:[calcVie
    [calcViewController press:[calcVie
    XCTAssertEqualObjects([calcViewCon
}

/* testDivision performs a simple div
 * Check: 25 / 4 = 6.25.
 */
- (void) testDivision {
    [calcViewController press:[calcVie
    [calcViewController press:[calcVie
    [calcViewController press:[calcVie
```



```
/* testSubtraction performs a simple
 * Check: 6 - 2 = 4.
 */
- (void) testSubtraction {
    [calcViewController press:[calcVie
    [calcViewController press:[calcVie
    [calcViewController press:[calcVie
    [calcViewController press:[calcVie
    XCTAssertEqualObjects([calcViewCon
}

/* testDivision performs a simple div
 * Check: 25 / 4 = 6.25.
 */
- (void) testDivision {
    [calcViewController press:[calcVie
    [calcViewController press:[calcVie
    [calcViewController press:[calcVie
```

Note: The same indicator appears next to the `@implementation` for the class as well, allowing you to run all of the tests in the class.

Using the Product Menu

The Product menu includes quickly accessible commands to run tests directly from the keyboard.

Product > **Test.** Runs the currently active scheme. The keyboard shortcut is Command-U.
Product > **Build for** > **Testing** and **Product** > **Perform Action** > **Test without Building.** These two commands can be used to build the test bundle products and run the tests independent of one another. These are convenience commands to shortcut the build and test processes. They're most useful when changing code to check for warnings and errors in the build process, and for speeding up testing when you know the build is up to date. The keyboard shortcuts are Shift-Command-U and Control-Command-U, respectively.
Product > **Perform Action** > **Test <testName>.** This dynamic menu item senses the current test method in which the editing insertion point is positioned when you're editing a test method and allows you to run that test with a keyboard shortcut. The command's name adapts to show the test it will run, for instance, Product > Perform Action > Test *testAddition.* The keyboard shortcut is Control-Option-Command-U.

Note: In addition to the source editor, this command also operates based on the selection in the project navigator and the test navigator. When either of those two navigators is active, the source editor does not have focus and the command takes the current selection in either of these navigators for input.

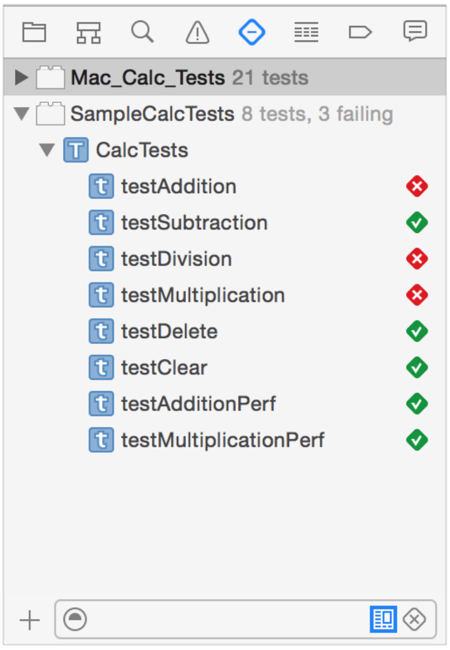
In the test navigator, the selection can be on a test bundle, class, or method. In the project navigator, the selection can be on the test class implementation file, for instance, `CalcTests.m`.

Product > **Perform Action** > **Test Again <testName>.** Reruns the last test method executed, most useful when debugging/editing code in which a test method exposed a problem. Like the Product > Perform Action > Test command, the name of the test that is run appears in the command, for example, Product > Perform Action > Test Again *testAddition.* The keyboard shortcut is Control-Option-Command-G.

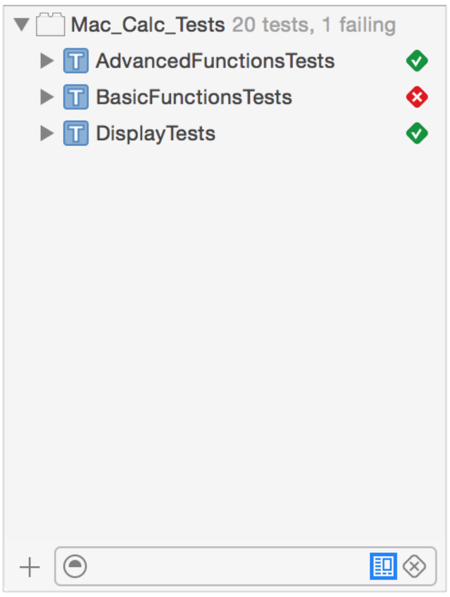
Display of Test Results

The XCTest framework displays the pass or fail results of a test method to Xcode in several ways. The following screenshots show where you can see the results.

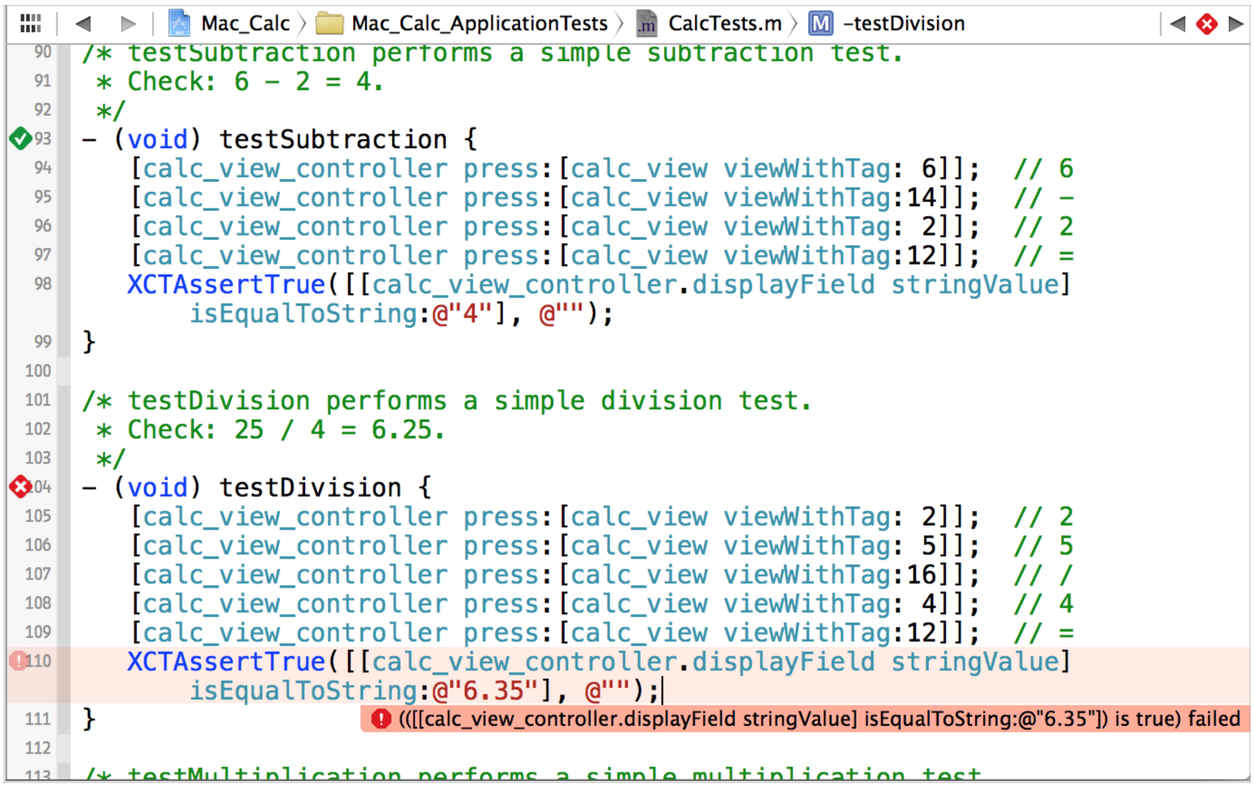
- In the test navigator, you can view pass/fail indicators after a test or group of tests is run.



If test methods are collapsed into their respective class, or test classes into the test bundles, the indicator reflects the aggregate status of the enclosed tests. In this example, at least one of the tests in the `BasicFunctionsTests` class has signaled a failure.



- In the source editor, you can view pass/fail indicators and debugging information.

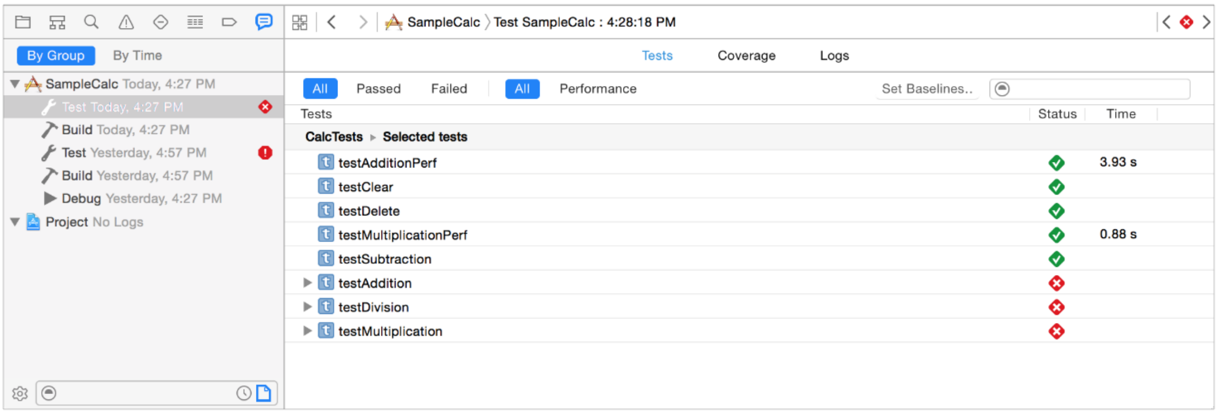

Mac_Calc 〉 Mac_Calc_ApplicationTests 〉 CalcTests.m 〉 -testDivision

```
/* testSubtraction performs a simple subtraction test.
 * Check: 6 - 2 = 4.
 */
- (void) testSubtraction {
    [calc_view_controller press:[calc_view viewWithTag: 6]];  // 6
    [calc_view_controller press:[calc_view viewWithTag:14]];  // -
    [calc_view_controller press:[calc_view viewWithTag: 2]];  // 2
    [calc_view_controller press:[calc_view viewWithTag:12]];  // =
    XCTAssertTrue([[calc_view_controller.displayField stringValue]
        isEqualToString:@"4"], @"");
}

/* testDivision performs a simple division test.
 * Check: 25 / 4 = 6.25.
 */
- (void) testDivision {
    [calc_view_controller press:[calc_view viewWithTag: 2]];  // 2
    [calc_view_controller press:[calc_view viewWithTag: 5]];  // 5
    [calc_view_controller press:[calc_view viewWithTag:16]];  // /
    [calc_view_controller press:[calc_view viewWithTag: 4]];  // 4
    [calc_view_controller press:[calc_view viewWithTag:12]];  // =
    XCTAssertTrue([[calc_view_controller.displayField stringValue]
        isEqualToString:@"6.35"], @"");
}            (([[calc_view_controller.displayField stringValue] isEqualToString:@"6.35"]) is true) failed

/* testMultiplication performs a simple multiplication test
```

- In the reports navigator, you can view results output by test runs. With the Tests panel active, select the test run you wish to inspect in the left hand panel. .

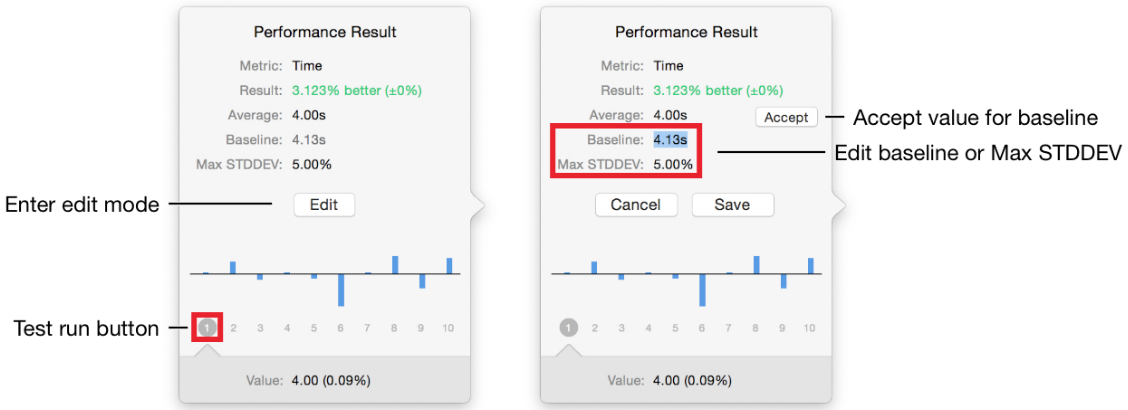


SampleCalc Today, 4:27 PM
Test Today, 4:27 PM
Build Today, 4:27 PM
Test Yesterday, 4:57 PM
Build Yesterday, 4:57 PM
Debug Yesterday, 4:27 PM
Project No Logs

CalcTests ▸ Selected tests

Tests	Status	Time
testAdditionPerf	✓	3.93 s
testClear	✓	
testDelete	✓	
testMultiplicationPerf	✓	0.88 s
testSubtraction	✓	
testAddition	✗	
testDivision	✗	
testMultiplication	✗	

For a performance test, click the value in the Time column to obtain a detailed report on the performance result. You can see the aggregate performance of the test as well as values for each of the ten runs by clicking the individual test run buttons. The Edit button enables you to set or modify the test baseline and the max standard deviation allowed in the indication of pass or fail.

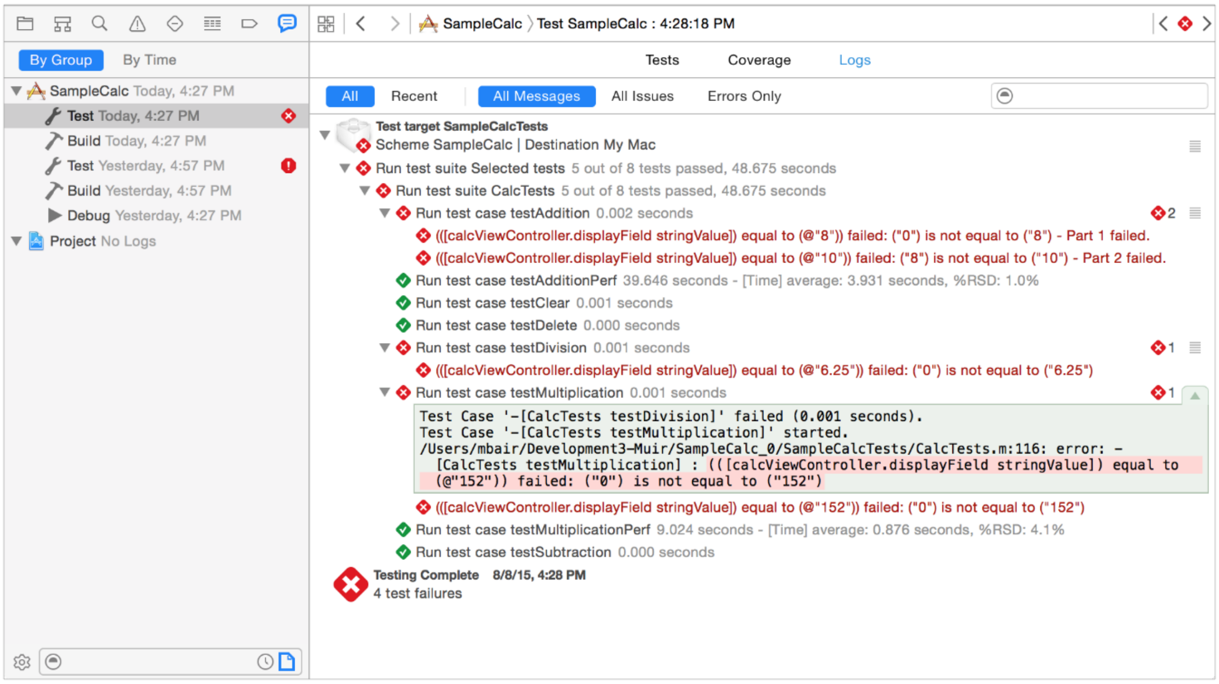


Performance Result

Metric: Time
Result: 3.123% better (±0%)
Average: 4.00s
Baseline: 4.13s
Max STDDEV: 5.00%

Value: 4.00 (0.09%)

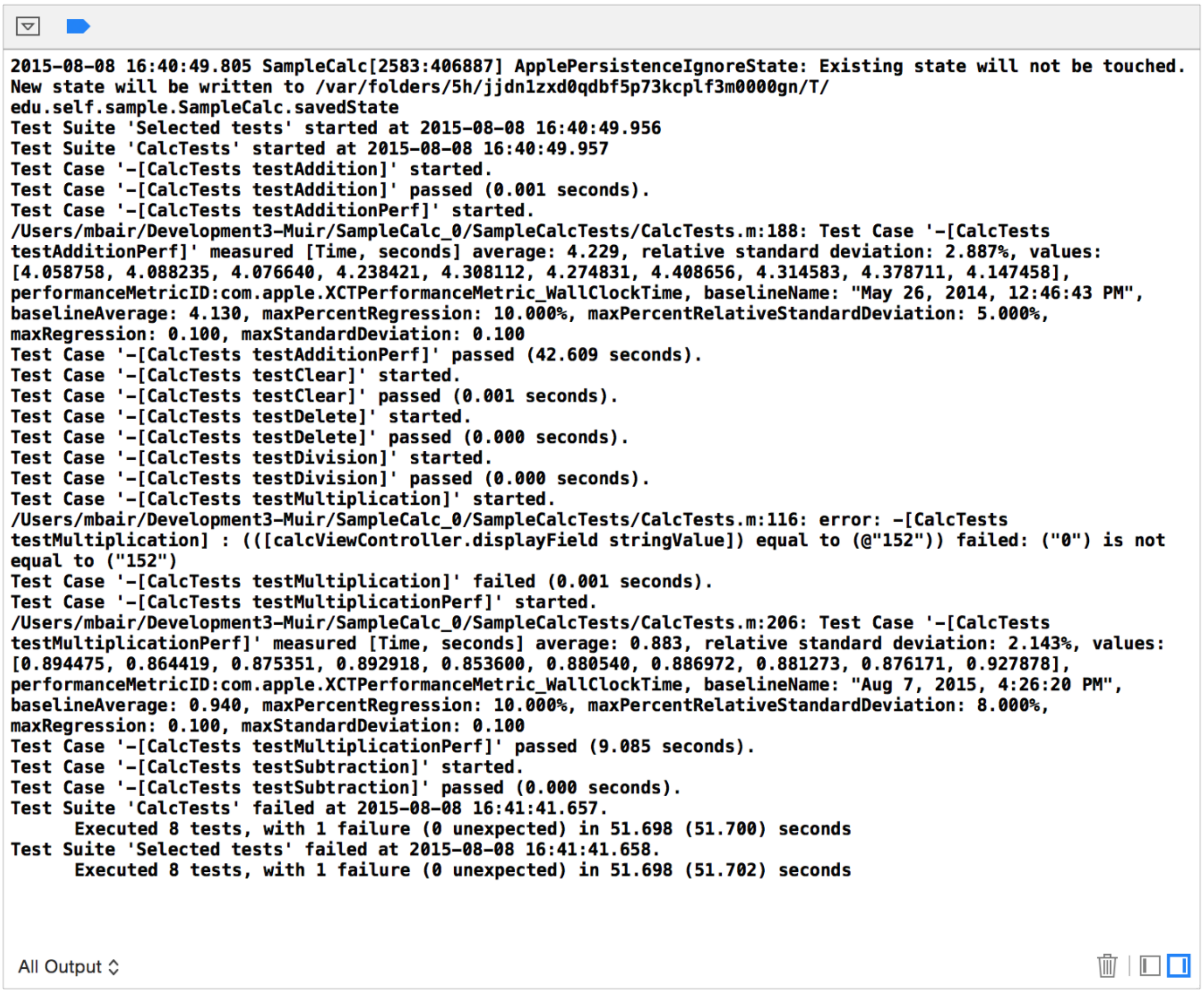
Using the Logs panel, you can view the associated failure description string and other summary output. By opening the disclosure triangles, you can drill down to all the details of the test run.



```
Test target SampleCalcTests
Scheme SampleCalc | Destination My Mac
Run test suite Selected tests 5 out of 8 tests passed, 48.675 seconds
  Run test suite CalcTests 5 out of 8 tests passed, 48.675 seconds
    Run test case testAddition 0.002 seconds                                    2
      (([calcViewController.displayField stringValue]) equal to (@"8")) failed: ("0") is not equal to ("8") - Part 1 failed.
      (([calcViewController.displayField stringValue]) equal to (@"10")) failed: ("8") is not equal to ("10") - Part 2 failed.
    Run test case testAdditionPerf 39.646 seconds - [Time] average: 3.931 seconds, %RSD: 1.0%
    Run test case testClear 0.001 seconds
    Run test case testDelete 0.000 seconds
    Run test case testDivision 0.001 seconds                                    1
      (([calcViewController.displayField stringValue]) equal to (@"6.25")) failed: ("0") is not equal to ("6.25")
    Run test case testMultiplication 0.001 seconds                              1
      Test Case '-[CalcTests testDivision]' failed (0.001 seconds).
      Test Case '-[CalcTests testMultiplication]' started.
      /Users/mbair/Development3-Muir/SampleCalc_0/SampleCalcTests/CalcTests.m:116: error: -
        [CalcTests testMultiplication] : (([calcViewController.displayField stringValue]) equal to
        (@"152")) failed: ("0") is not equal to ("152")
      (([calcViewController.displayField stringValue]) equal to (@"152")) failed: ("0") is not equal to ("152")
    Run test case testMultiplicationPerf 9.024 seconds - [Time] average: 0.876 seconds, %RSD: 4.1%
    Run test case testSubtraction 0.000 seconds
Testing Complete    8/8/15, 4:28 PM
4 test failures
```

Note: In addition to the disclosure triangles on the left of the item entries, the small icon to the right of a test failure item can be expanded to show more information, as you can see in the displayed `testMultiplication` failure above.

- The debug console displays comprehensive information about the test run in a textual format. It's the same information as shown by the log navigator, but if you have been actively engaged in debugging, any output from the debugging session also appears there.



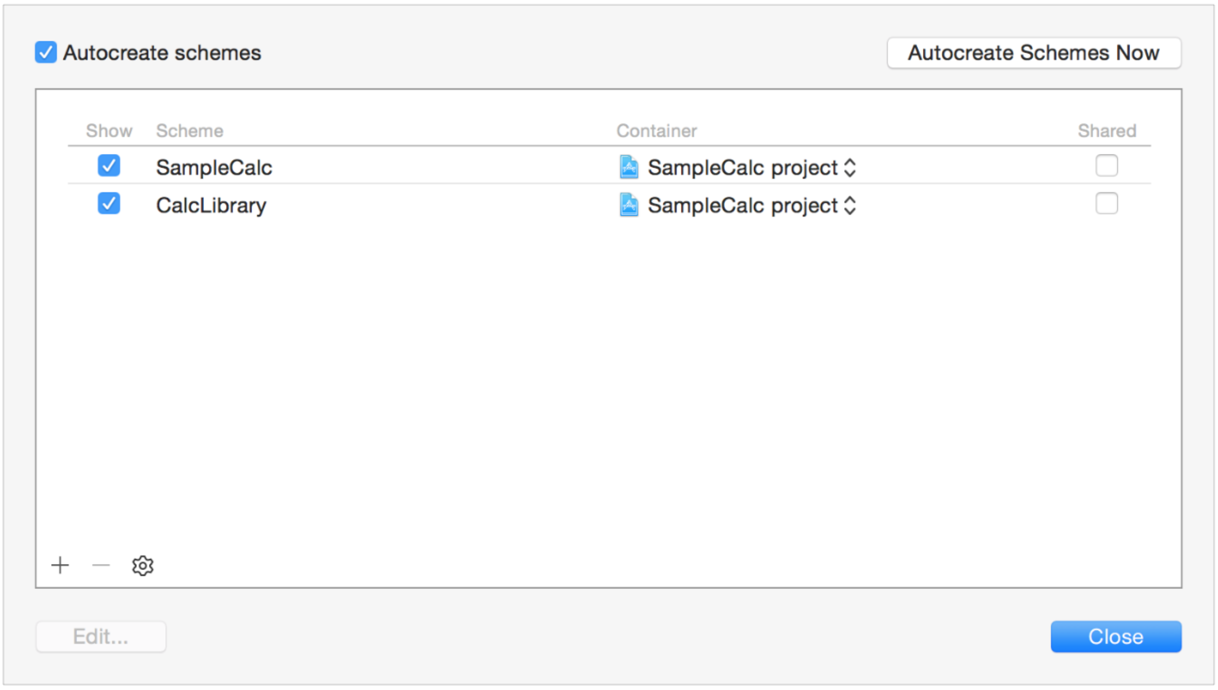
```
2015-08-08 16:40:49.805 SampleCalc[2583:406887] ApplePersistenceIgnoreState: Existing state will not be touched.
New state will be written to /var/folders/5h/jjdn1zxd0qdbf5p73kcplf3m0000gn/T/
edu.self.sample.SampleCalc.savedState
Test Suite 'Selected tests' started at 2015-08-08 16:40:49.956
Test Suite 'CalcTests' started at 2015-08-08 16:40:49.957
Test Case '-[CalcTests testAddition]' started.
Test Case '-[CalcTests testAddition]' passed (0.001 seconds).
Test Case '-[CalcTests testAdditionPerf]' started.
/Users/mbair/Development3-Muir/SampleCalc_0/SampleCalcTests/CalcTests.m:188: Test Case '-[CalcTests
testAdditionPerf]' measured [Time, seconds] average: 4.229, relative standard deviation: 2.887%, values:
[4.058758, 4.088235, 4.076640, 4.238421, 4.308112, 4.274831, 4.408656, 4.314583, 4.378711, 4.147458],
performanceMetricID:com.apple.XCTPerformanceMetric_WallClockTime, baselineName: "May 26, 2014, 12:46:43 PM",
baselineAverage: 4.130, maxPercentRegression: 10.000%, maxPercentRelativeStandardDeviation: 5.000%,
maxRegression: 0.100, maxStandardDeviation: 0.100
Test Case '-[CalcTests testAdditionPerf]' passed (42.609 seconds).
Test Case '-[CalcTests testClear]' started.
Test Case '-[CalcTests testClear]' passed (0.001 seconds).
Test Case '-[CalcTests testDelete]' started.
Test Case '-[CalcTests testDelete]' passed (0.000 seconds).
Test Case '-[CalcTests testDivision]' started.
Test Case '-[CalcTests testDivision]' passed (0.000 seconds).
Test Case '-[CalcTests testMultiplication]' started.
/Users/mbair/Development3-Muir/SampleCalc_0/SampleCalcTests/CalcTests.m:116: error: -[CalcTests
testMultiplication] : (([calcViewController.displayField stringValue]) equal to (@"152")) failed: ("0") is not
equal to ("152")
Test Case '-[CalcTests testMultiplication]' failed (0.001 seconds).
Test Case '-[CalcTests testMultiplicationPerf]' started.
/Users/mbair/Development3-Muir/SampleCalc_0/SampleCalcTests/CalcTests.m:206: Test Case '-[CalcTests
testMultiplicationPerf]' measured [Time, seconds] average: 0.883, relative standard deviation: 2.143%, values:
[0.894475, 0.864419, 0.875351, 0.892918, 0.853600, 0.880540, 0.886972, 0.881273, 0.876171, 0.927878],
performanceMetricID:com.apple.XCTPerformanceMetric_WallClockTime, baselineName: "Aug 7, 2015, 4:26:20 PM",
baselineAverage: 0.940, maxPercentRegression: 10.000%, maxPercentRelativeStandardDeviation: 8.000%,
maxRegression: 0.100, maxStandardDeviation: 0.100
Test Case '-[CalcTests testMultiplicationPerf]' passed (9.085 seconds).
Test Case '-[CalcTests testSubtraction]' started.
Test Case '-[CalcTests testSubtraction]' passed (0.000 seconds).
Test Suite 'CalcTests' failed at 2015-08-08 16:41:41.657.
      Executed 8 tests, with 1 failure (0 unexpected) in 51.698 (51.700) seconds
Test Suite 'Selected tests' failed at 2015-08-08 16:41:41.658.
      Executed 8 tests, with 1 failure (0 unexpected) in 51.698 (51.702) seconds
```

Working with Schemes and Test Targets

Xcode schemes control what the Build, Run, Test, and Debug menu commands do. Xcode manages the scheme configuration for you when you create test targets and perform other test system manipulations with the test navigator—for example, when you enable or disable a test method, test class, or test bundle. Using Xcode Server and continuous integration requires a scheme to be set to Shared using the checkbox in the Manage Schemes sheet, and checked into a source repository along with your project and source code.

To see the scheme configuration for your tests:

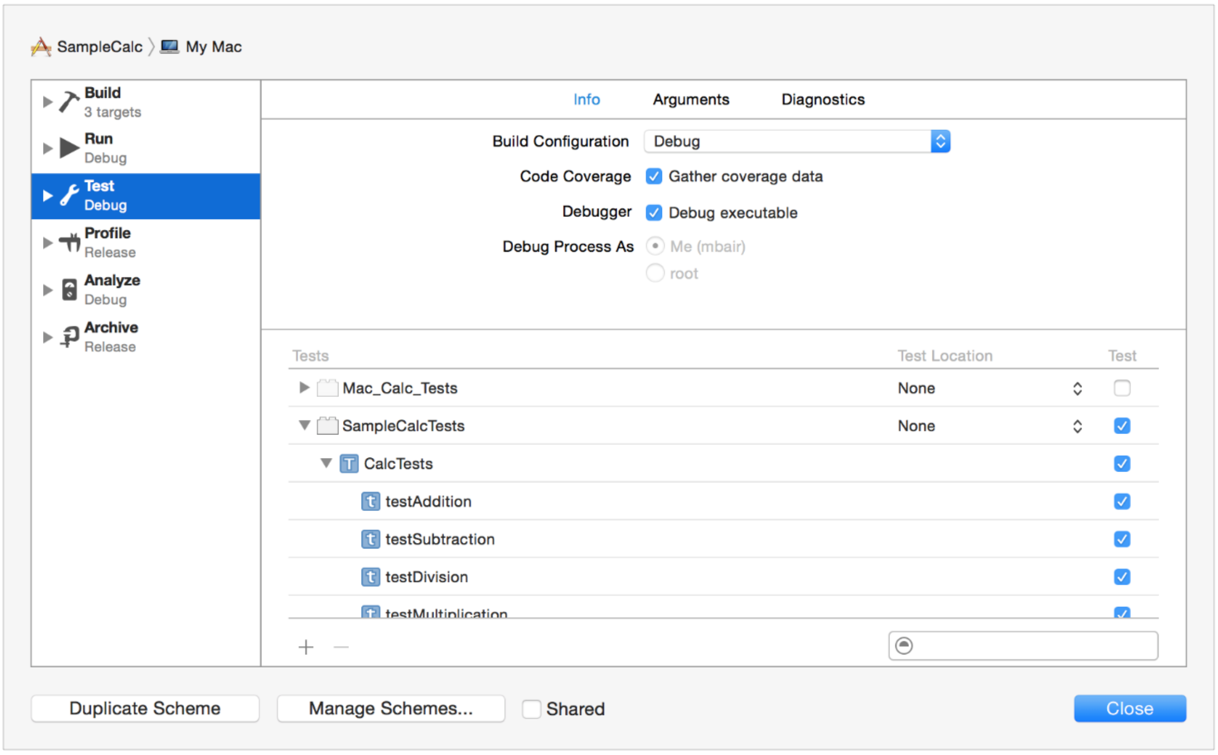
1. Choose the Scheme menu > Manage Schemes in the toolbar to present the scheme management sheet.



Show	Scheme	Container	Shared
✓	SampleCalc	SampleCalc project	
✓	CalcLibrary	SampleCalc project	

In this project there are two schemes, one to build the app and the other to build the library/framework. The checkboxes labeled Shared on the right configure a scheme as shared and available for use by bots with Xcode Server.

2. In the management sheet, double-click a scheme to display the scheme editor. A scheme's Test action identifies the tests Xcode performs when you execute the Test command.



Build Configuration: Debug
Code Coverage: ✓ Gather coverage data
Debugger: ✓ Debug executable
Debug Process As: ● Me (mbair) ○ root

Tests	Test Location	Test
Mac_Calc_Tests	None	
SampleCalcTests	None	✓
CalcTests		✓
testAddition		✓
testSubtraction		✓
testDivision		✓
testMultiplication		✓

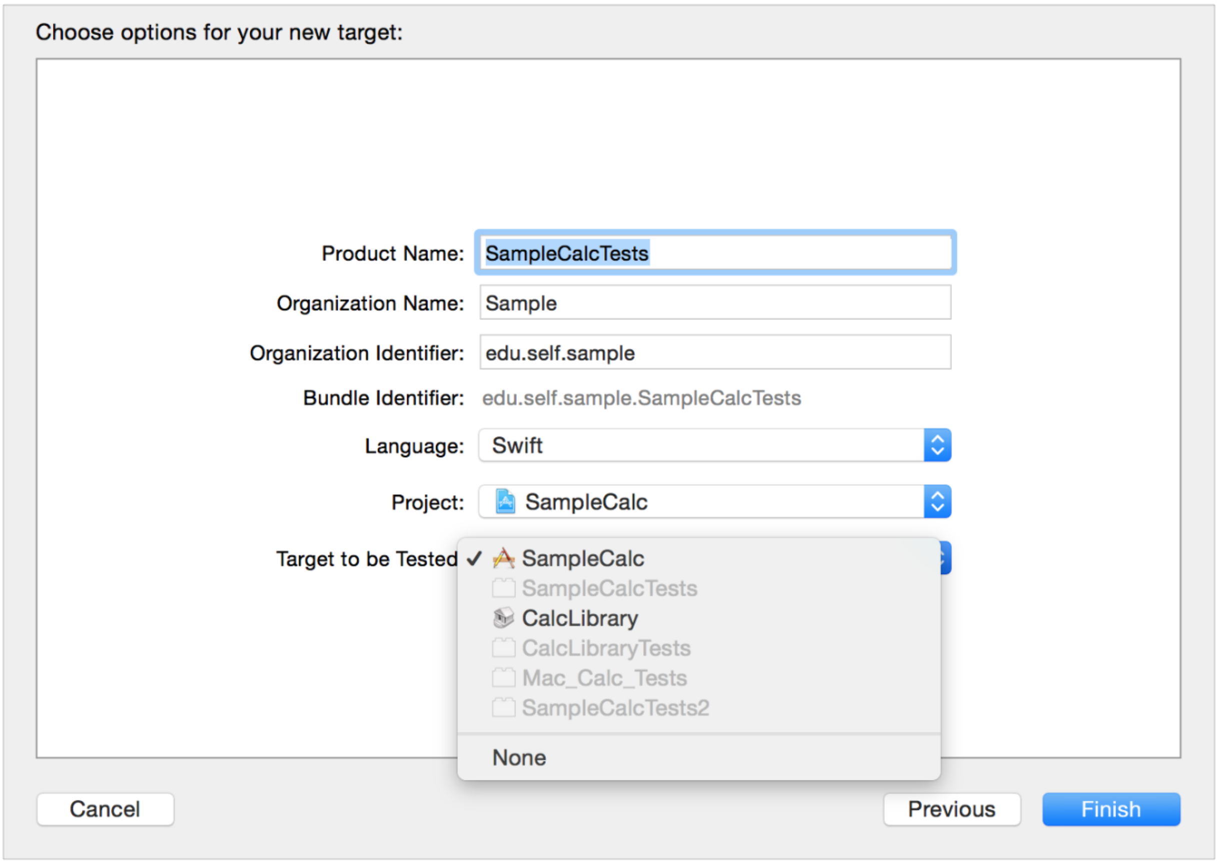
Note: The test navigator and configuration/setup assistants associated with test targets, test classes, and test methods normally maintain all the scheme settings for you with respect to test operations.

Extensive information about using, configuring, and editing schemes is available in the video presentation WWDC 2012: Working with Schemes and Projects in Xcode (408).

Build Settings—Testing Apps, Testing Libraries

App tests run in the context of your app, allowing you to create tests which combine behavior that comes from the different classes, libraries/frameworks, and functional aspects of your app. Library tests exercise the classes and methods within a library or framework, independent of your app, to validate that they behave as the library's specification requires.

Different build settings are required for these two types of test bundles. Configuring the build settings is performed for you when you create test targets by choosing the target parameter in the new target assistant. The target assistant is shown with the Target pop-up menu open. The app, `SampleCalc`, and the library/framework, `CalcLibrary`, are the available choices.



Choose options for your new target:

Product Name: SampleCalcTests
Organization Name: Sample
Organization Identifier: edu.self.sample
Bundle Identifier: edu.self.sample.SampleCalcTests
Language: Swift
Project: SampleCalc
Target to be Tested ✓ SampleCalc

Choosing `SampleCalc` as the associated build product for this test target configures the build settings to be for an app test. The app process hosts the execution of your tests; tests are executed after receiving the `applicationDidFinishLaunching` notification. The default Product Name, "SampleCalc Tests," for the test target is derived from the `SampleCalc` target name in this case; you can change that to suit your preferences.

If you choose `CalcLibrary` as the associated build product instead, the target assistant configures the build settings for a library test. Xcode bootstraps the testing runtime context, the library or framework, and your test code is hosted by a process managed by Xcode. The default Product Name for this case is derived from the library target ("CalcLibrary Tests"). You can change it to suit your preferences just as with the app test case.

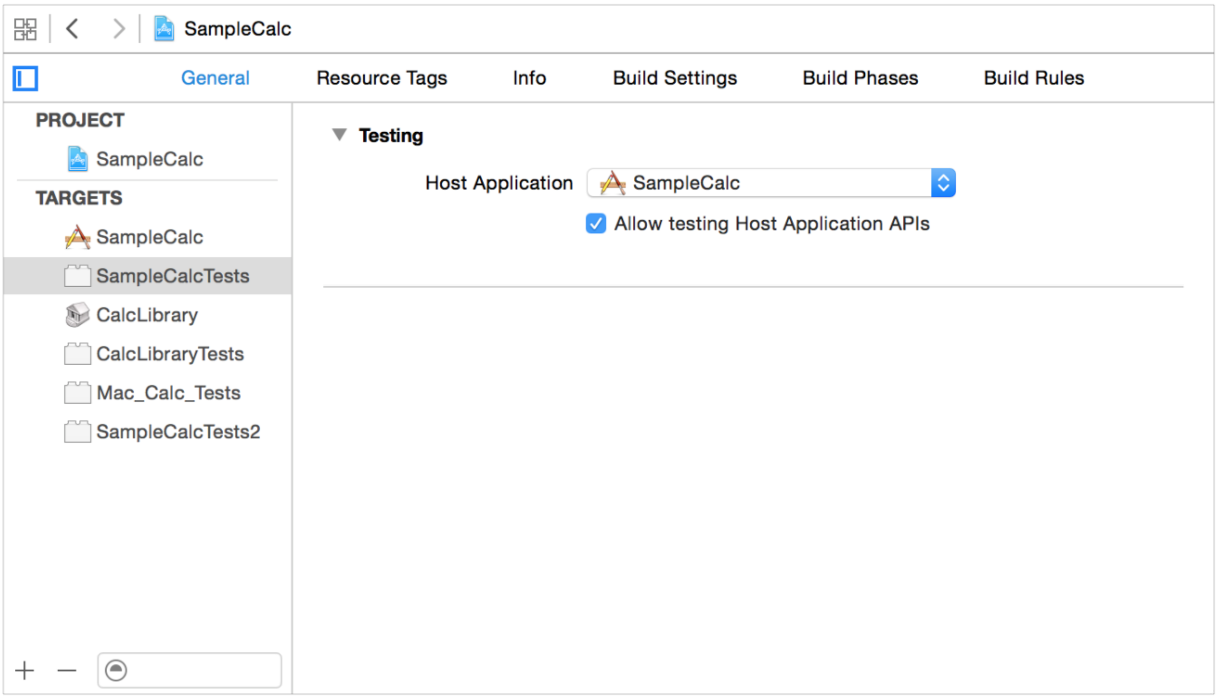
Build Setting Defaults

For most situations, choosing the correct test target to build product association is all you need to do to configure build settings for app and library tests. Xcode takes care of managing the build settings for you automatically. Because you might have a project that requires some complex build settings, it is useful to understand the standard build settings that Xcode sets up for app tests and library tests.

The `SampleCalc` project serves as an example to illustrate the correct default settings.

1. Enter the project editor by clicking the `SampleCalc` project in the project navigator, then select the `SampleCalcTests` app test target.

   In the General pane in the editor, a Target pop-up menu is displayed. The pop-up menu should show the `SampleCalc` app as target.



Testing

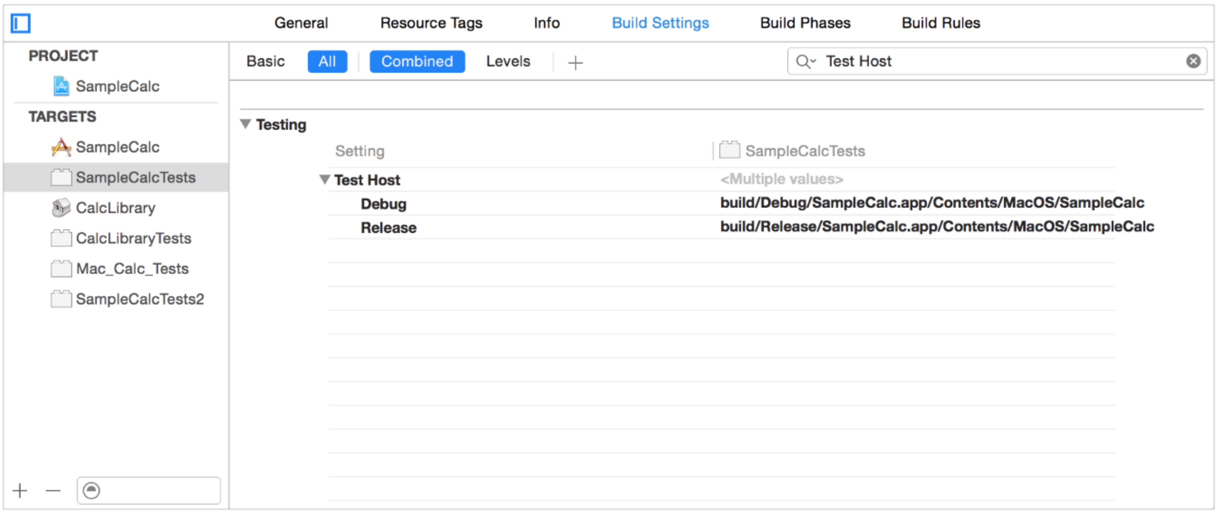
Host Application: SampleCalc

☑ Allow testing Host Application APIs

You can check that the build settings are correct for the `SampleCalcTests` target.

2. Click Build Settings, then type `Bundle Loader` in the search field. The app tests for `SampleCalc` are loaded by the the `SampleCalc` app. You'll see the path to the executable as customized parameters for both Debug and Release builds.

   The same paths appear if you search for `Test Host`, as shown.



Testing

Setting	SampleCalcTests
Test Host	<Multiple values>
Debug	build/Debug/SampleCalc.app/Contents/MacOS/SampleCalc
Release	build/Release/SampleCalc.app/Contents/MacOS/SampleCalc

The calculator library target of this project is named `CalcLibrary` and has an associated test target named `CalcLibraryTests`.

3. Select the `CalcLibraryTests` target and the General pane.

   The target is set to `None`. Similarly checking for `Bundle Loader` and `Test Host` in the Build Settings panel have no associated parameters. This indicates that Xcode has configured them with its defaults, which is the correct configuration.