Class

XCTestCase

The primary class for defining test cases, test methods, and performance tests.

```
class XCTestCase
```

Mentioned in

- Defining Test Cases and Test Methods
- Set Up and Tear Down State in Your Tests
- Adding Attachments to Tests, Activities, and Issues
- Grouping Tests into Substeps with Activities

Overview

A test case is a group of related test methods, with optional setup and teardown before and after tests run. See Defining Test Cases and Test Methods for more information.

`XCTestCase` conforms to `XCTActivity`, so you can simplify complex tests by organizing them into activities, and attach output to tests for later analysis. For more information, see Activities and Attachments.

Create tests for asynchronous operations using expectations. For more information, see `Testing Asynchronous Operations with Expectations`. If your app uses Swift Concurrency, annotate test methods with `async` or `async throws` instead to test asynchronous operations, and use standard Swift concurrency patterns in your tests.

Create tests to measure performance for specific blocks of code using the methods in the Measuring Performance section below. Build performance tests as part of a continuous

improvement cycle for performance in your app. For more information, see Improving your app's performance.

Topics

Customizing Test Setup and Teardown

Set Up and Tear Down State in Your Tests

Prepare initial state before tests run, and clean up resources after tests complete.

`class func setUp()`

Provides an opportunity to customize initial state before a test case begins.

`func addTeardownBlock(() async throws -> Void)`

Registers a block of teardown code to run after the current test method ends.

`func addTeardownBlock(() throws -> Void)`

Registers a block of teardown code to run after the current test method ends.

`class func tearDown()`

Provides an opportunity to perform cleanup after a test case ends.

Managing Test Case Execution

`class var runsForEachTargetApplicationUIConfiguration: Bool`

A Boolean value that indicates whether your UI tests run once for each possible combination of orientation, localization, and other appearance settings your app supports.

`var continueAfterFailure: Bool`

A Boolean value that indicates whether a test method should continue running after a failure occurs.

`var executionTimeAllowance: TimeInterval`

The number of seconds, rounded up to the nearest minute, for a test to run before it fails with a timeout error.

Measuring Performance

`func measure(() -> Void)`

Measures the performance of a block of code.

`func measureMetrics([XCTPerformanceMetric], automaticallyStartMeasuring: Bool, for: () -> Void)`

Measures the performance of a block of code, optionally deferring the starting point for measurement.

`func measure(metrics: [any XCTMetric], block: () -> Void)`

Records the selected metrics for a block of code.

`func measure(metrics: [any XCTMetric], options: XCTMeasureOptions, block: () -> Void)`

Records the selected metrics, using the specified measurement options, for a block of code.

`func measure(options: XCTMeasureOptions, block: () -> Void)`

Records the performance, using the specified measurement options, for a block of code.

`func startMeasuring()`

Starts recording performance metrics within a block of code.

`func stopMeasuring()`

Ends recording performance metrics within a block of code.

`class var defaultPerformanceMetrics: [XCTPerformanceMetric]`

An array of default performance metrics the test records.

`class var defaultMetrics: [any XCTMetric]`

An array of default metrics the test uses to record performance.

`class var defaultMeasureOptions: XCTMeasureOptions`

The default measurement options the test uses to record performance.

`struct XCTPerformanceMetric`

Performance metrics that the test records.

Creating Asynchronous Test Expectations

To create asynchronous test expectations, use the convenience methods below, or create instances of the test expectation class and its subclasses manually.

`func expectation(description: String) -> XCTestExpectation`

Creates a new expectation with an associated description.

```
func expectation(for: NSPredicate, evaluatedWith: Any?, handler:
XCTNSPredicateExpectation.Handler?) -> XCTestExpectation
```

Creates an expectation that the test fulfills by evaluating the predicate with the specified object.

```
func expectation(forNotification: NSNotification.Name, object: Any?,
handler: XCTNSNotificationExpectation.Handler?) -> XCTestExpectation
```

Creates an expectation that the test fulfills when it receives a specific notification for a specified object.

```
func expectation(forNotification: NSNotification.Name, object: Any?,
notificationCenter: NotificationCenter, handler: XCTNSNotification
Expectation.Handler?) -> XCTestExpectation
```

Creates an expectation that the test fulfills when it receives a specific notification from a specific notification center for a specified object.

```
func keyValueObservingExpectation(for: Any, keyPath: String, expected
Value: Any?) -> XCTestExpectation
```

Creates an expectation that uses Key-Value Observing to observe a value until it matches an expected value.

```
func expectation<T, V>(that: KeyPath<T, V>, on: T, options: NSKeyValue
ObservingOptions, willEqual: V) -> XCTKeyPathExpectation<T, V>
```

Creates an expectation using key-value observing the test fulfills when the value of an observed property changes to an expected value.

```
func keyValueObservingExpectation(for: Any, keyPath: String, handler:
XCTKVOExpectation.Handler?) -> XCTestExpectation
```

Creates an expectation that uses Key-Value Observing to observe a value and respond to changes in that value by calling a provided handler.

~~`func expectation<T, V>(that: KeyPath<T, V>, on: T, options: NSKeyValue ObservingOptions, willSatisfy: XCTKeyPathExpectation<T, V>.Predicate?) -> XCTKeyPathExpectation<T, V>`~~

Creates an expectation using key-value observing the test fulfills when the value of an observed property changes and satisfies the conditions of a predicate's evaluation.

Deprecated

```
func expectation<T, V>(that: KeyPath<T, V>, on: T, options: NSKeyValue
ObservingOptions, willSatisfy: XCTKeyPathExpectation<T, V>.Asynchronous
Filter?) -> XCTKeyPathExpectation<T, V>
```

Creates an expectation using key-value observing to monitor changes to a given key path on a given object.

```
func expectation<T, V>(that: KeyPath<T, V>, on: T, options: NSKeyValue
ObservingOptions, willSatisfy: XCTKeyPathExpectation<T, V>.Synchronous
Filter?) -> XCTKeyPathExpectation<T, V>
```

Creates an expectation using key-value observing to monitor changes to a given key path on a given object.

Waiting for Expectations

To wait for the test to fulfill asynchronous test expectations, create a waiter object directly or by using the convenience methods.

```
func fulfillment(of: [XCTestExpectation], timeout: TimeInterval,
enforceOrder: Bool) async
```

Waits on a group of expectations for up to the specified timeout, optionally enforcing their order of fulfillment.

```
func wait(for: [XCTestExpectation])
```

Waits on a group of expectations.

```
func wait(for: [XCTestExpectation], enforceOrder: Bool)
```

Waits on a group of expectations optionally enforcing their order of fulfillment.

```
func wait(for: [XCTestExpectation], timeout: TimeInterval)
```

Waits for the test to fulfill a set of expectations within a specified time.

```
func wait(for: [XCTestExpectation], timeout: TimeInterval, enforceOrder
: Bool)
```

Waits for the test to satisfy an array of expectations and specifies whether they must occur in the array's order.

```
func waitForExpectations(timeout: TimeInterval, handler: (((any Error
)?) -> Void)?)
```

Waits until the test fulfills all expectations or until it times out.

```
typealias XCWaitCompletionHandler
```

A block the test runner calls when the test fulfills a waiter's expectations, or when it times out.

```
struct XCTestError
```

A type of error that can occur while the test waits to fulfill expectations.

`enum Code`

Error codes for errors that can occur while the test is waiting to fulfill expectations.

`let XCTestErrorDomain: String`

The error domain for errors that can occur while the test is waiting to fulfill expectations.

Monitoring UI Interruptions

Handling UI Interruptions

Improve your UI test's stability by handling interface changes that block the UI elements under test.

`func addUIInterruptionMonitor(withDescription: String, handler: (XCUIElement) -> Bool) -> any NSObjectProtocol`

Adds a handler to the current context.

`func removeUIInterruptionMonitor(any NSObjectProtocol)`

Removes a handler using the token from when you added the handler.

Creating Tests Programmatically

The test runner automatically detects methods you define in your test case subclasses. Use the symbols below if you need more customization for test case creation, such as to define test cases dynamically at run time.

`init(invocation: NSInvocation?)`

Initializes a test case with an invocation.

`init(selector: Selector)`

Initializes a test case with a selector.

`class var testInvocations: [NSInvocation]`

An array of invocations that represents each test method in the test case.

`var invocation: NSInvocation?`

The invocation for running the test.

`func invokeTest()`

Invokes the test.

`func record(XCTIssue)`

Records an issue during test execution.

~~`func recordFailure(withDescription: String, inFile: String, atLine: Int, expected: Bool)`~~

Records a failure during text execution.

Deprecated

`class var defaultTestSuite: XCTestSuite`

A test suite that contains test cases for all of the tests in the class.

Relationships

Inherits From

`XCTest`

Conforms To

`CVarArg`
`Copyable`
`CustomDebugStringConvertible`
`CustomStringConvertible`
`Equatable`
`Hashable`
`NSObjectProtocol`
`XCTActivity`
`XCTWaiterDelegate`

See Also

Test cases and test methods

Defining Test Cases and Test Methods

Add test cases and test methods to a test target to confirm that your code performs as expected.

```
class XCTest
```

An abstract base class for creating, managing, and executing tests.