QUnit - Quick Guide

QUnit - Overview

Testing is the process of checking the functionality of the application whether it is working as per the requirements and to ensure that at the developer level, unit testing comes into picture. Unit testing is the testing of a single entity (class or method). Unit testing is very essential for every software organization to offer quality products to their clients.

Unit testing can be done in two ways as mentioned in the following table.

Manual testing	Automated testing
Executing the test cases manually without any tool support is known as manual testing.	Taking tool support and executing the test cases using automation tool is known as automation testing.
Time consuming and tedious. Since the test cases are executed by human resources, it is very slow and tedious.	Fast Automation. Runs test cases significantly faster than human resources.
Huge investment in human resources. As test cases need to be executed manually, more number of testers are required.	Less investment in human resources. Test cases are executed using automation tool hence, less number of testers are required.
Less reliable, as tests may not be performed with precision each time due to human errors.	More reliable. Automation tests perform precisely the same operation each time they are run.
Non-programmable. No programming can be done to write sophisticated tests, which fetch hidden information.	Programmable. Testers can program sophisticated tests to bring out hidden information.

What is QUnit?

QUnit is a unit testing framework for JavaScript programming language. It is important in the test driven development, and is used by jQuery, jQuery UI, and jQuery Mobile projects. QUnit is capable of testing any generic JavaScript codebase.

QUnit promotes the idea of "first testing then coding", which emphasizes on setting up the test data for a piece of code, which can be tested first and then implemented. This approach is like "test a little, code a little, test a little, code a little..." which increases the programmer's productivity and the stability of program code reducing the programmer's stress and the time spent on debugging.

Features of QUnit

QUnit is an open source framework used for writing and running tests. Following are its most prominent features −

- QUnit provides Assertions for testing expected results.
- QUnit provides Test fixtures for running tests.
- QUnit tests allow to write code faster, which increases the quality.
- QUnit is elegantly simple. It is less complex and takes less time.
- QUnit tests can be run automatically and they check their own results and provide immediate feedback. There's no need to manually comb through a report of test results.
- QUnit tests can be organized into test suites containing test cases and even other test suites.
- QUnit shows test progress in a bar that is green if the test is going fine, and it turns red when a test fails.

What is a Unit Test Case?

A Unit Test Case is a part of code which ensures that another part of the code (method) works as expected. To achieve the desired results quickly, test framework is required. QUnit is a perfect unit test framework for JavaScript programming language.

A formal written unit test case is characterized by a known input and by an expected output, which is worked out before the test is executed. The known input should test a precondition and the expected output should test a post-condition.

There must be at least two unit test cases for each requirement: one positive test and one negative test. If a requirement has sub-requirements, each sub-requirement must have at least two test cases as positive and negative.

QUnit - Environment Setup

There are two ways to use QUnit.

Local Installation − You can download QUnit library on your local machine and include it in your HTML code.

CDN Based Version − You can include QUnit library into your HTML code directly from Content Delivery Network (CDN).

Local Installation

Go to the https://code.jquery.com/qunit/ to download the latest version available.

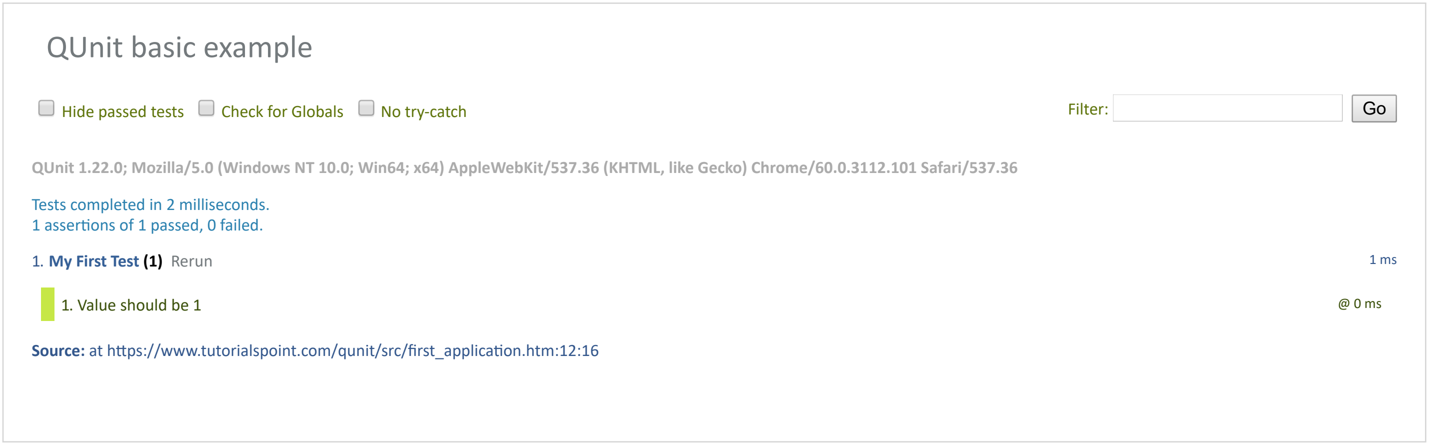
Place the downloaded **qunit-git.js** and **qunit-git.css** file in a directory of your website, e.g. /jquery.

Example

You can include **qunit-git.js** and **qunit-git.css** files in your HTML file as follows −

```
<html>
   <head>
      <meta charset = "utf-8">
      <title>QUnit basic example</title>
      <link rel = "stylesheet" href = "/jquery/qunit-git.css">
      <script src = "/jquery/qunit-git.js"></script>
   </head>

   <body>
      <div id = "qunit"></div>
      <div id = "qunit-fixture"></div>
      <script>
         QUnit.test( "My First Test", function( assert ) {
            var value = "1";
            assert.equal( value, "1", "Value should be 1" );
         });
      </script>
   </body>
</html>
```

This will produce following result −



QUnit basic example

```
QUnit 1.22.0; Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/60.0.3112.101 Safari/537.36

Tests completed in 2 milliseconds.
1 assertions of 1 passed, 0 failed.

1. My First Test (1)  Rerun                                                      1 ms

   1. Value should be 1                                                        @ 0 ms

Source: at https://www.tutorialspoint.com/qunit/src/first_application.htm:12:16
```

CDN Based Version

You can include QUnit library into your HTML code directly from Content Delivery Network (CDN).

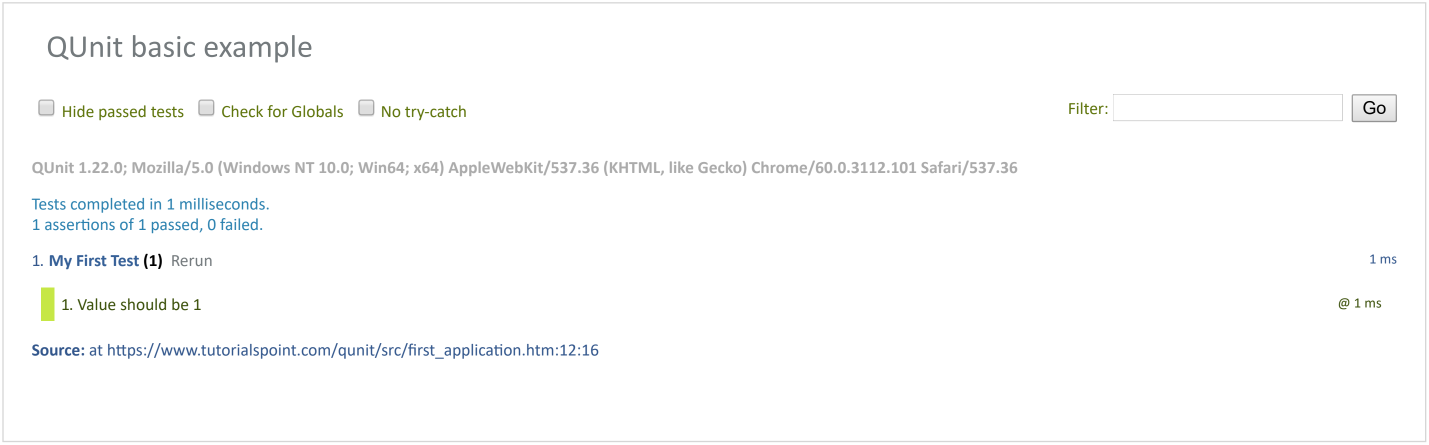
> We are using jQuery CDN version of the library throughout this tutorial.

Example

Let us rewrite the above example using QUnit library from jQuery CDN.

```
<html>
   <head>
      <meta charset = "utf-8">
      <title>QUnit basic example</title>
      <link rel = "stylesheet" href = "https://code.jquery.com/qunit/qunit-1.22.0.css">
      <script src = "https://code.jquery.com/qunit/qunit-1.22.0.js"></script>
   </head>

   <body>
      <div id = "qunit"></div>
      <div id = "qunit-fixture"></div>
      <script>
         QUnit.test( "My First Test", function( assert ) {
            var value = "1";
            assert.equal( value, "1", "Value should be 1" );
         });
      </script>
   </body>
</html>
```

This will produce following result –



QUnit basic example

```
QUnit 1.22.0; Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/60.0.3112.101 Safari/537.36

Tests completed in 1 milliseconds.
1 assertions of 1 passed, 0 failed.

1. My First Test (1) Rerun                                        1 ms

   1. Value should be 1                                           @ 1 ms

Source: at https://www.tutorialspoint.com/qunit/src/first_application.htm:12:16
```

QUnit - Basic Usage

Now we'll show you a step-by-step process to get a kickstart in QUnit using a basic example.

Import qunit.js

qunit.js of Qunit library represents the test runner and test framework.

```
<script src = "https://code.jquery.com/qunit/qunit-1.22.0.js"></script>
```

Import qunit.css

qunit.css of Qunit library styles the test suite page to display the test results.

```
<link rel = "stylesheet" href = "https://code.jquery.com/qunit/qunit-1.22.0.css">
```

Add Fixture

Add two div elements with **id = "qunit"** and **"qunit-fixture"**. These div elements are required and provide the fixture for tests.

```
<div id = "qunit"></div>
<div id = "qunit-fixture"></div>
```

Create a Function to Test

```
function square(x) {
   return x * x;
}
```

Create a Test Case

Make a call to the QUnit.test function, with two arguments.

Name – The name of the test to display the test results.

Function – Function testing code, having one or more assertions.

```
QUnit.test( "TestSquare", function( assert ) {
   var result = square(2);
   assert.equal( result, "4", "square(2) should be 4." );
});
```

Run the Test

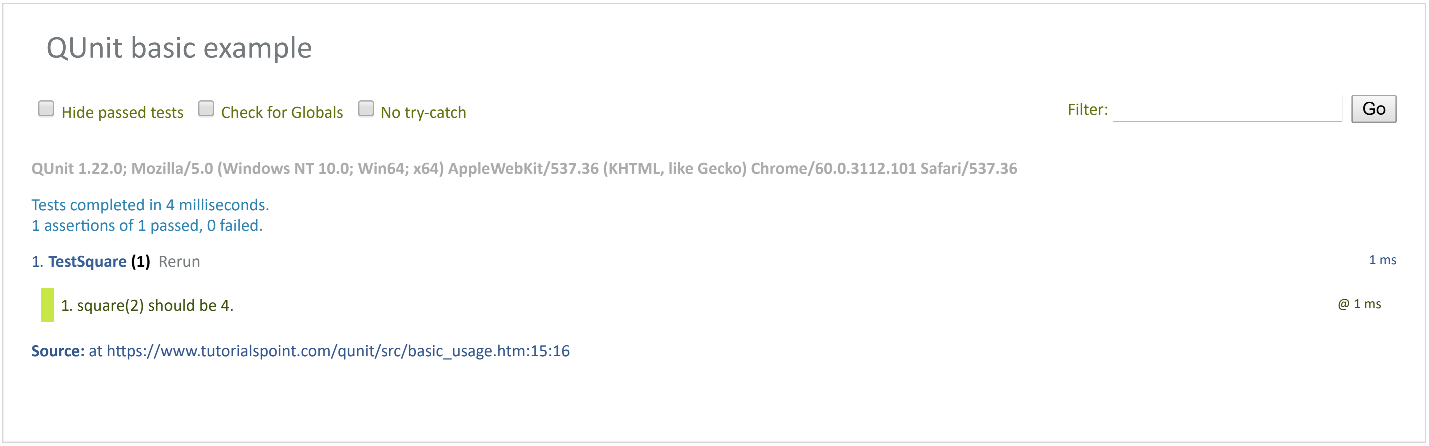
Now let us see the complete code in action.

```
<html>
   <head>
      <meta charset = "utf-8">
      <title>QUnit basic example</title>
      <link rel = "stylesheet" href = "https://code.jquery.com/qunit/qunit-1.22.0.css">
      <script src = "https://code.jquery.com/qunit/qunit-1.22.0.js"></script>
   </head>

   <body>
      <div id = "qunit"></div>
      <div id = "qunit-fixture"></div>

      <script>
         function square(x) {
            return x * x;
         }
         QUnit.test( "TestSquare", function( assert ) {
            var result = square(2);
            assert.equal( result, "4", "square(2) should be 4." );
         });
      </script>
   </body>
</html>
```

Load the page in the browser. The test runner calls **QUnit.test()** when the page gets loaded and adds the test to a queue. Execution of test case is deferred and controlled by the test runner.

Verify the Output

You should see the following result −



```
QUnit basic example

Hide passed tests   Check for Globals   No try-catch          Filter: [          ] Go

QUnit 1.22.0; Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/60.0.3112.101 Safari/537.36

Tests completed in 4 milliseconds.
1 assertions of 1 passed, 0 failed.

1. TestSquare (1)  Rerun                                                    1 ms

   1. square(2) should be 4.                                              @ 1 ms

Source: at https://www.tutorialspoint.com/qunit/src/basic_usage.htm:15:16
```

- **Header** − Test suite header displays the page title, a green bar when all tests are passed. Otherwise, a red bar when at least one test has failed, a bar with a three checkboxes to filter the test results, and a blue bar with the navigator.userAgent text to display the browser details.
- **Hide passed tests checkbox** − To hide the passed test cases and showcase only the failed test cases.
- **Check for globals checkbox** − To show the list of all properties on the window object, before and after each test, then check for differences. Modification to the properties will fail the test.
- **No try-catch checkbox** − To run test cases outside of a try-catch block so that in case of a test throwing an exception, the testrunner will die and show native exception.
- **Summary** − Shows the total time taken to run the test cases. Total test cases run and failed assertions.
- **Contents** − Shows the test results. Each test result has the name of the test followed by failed, passed, and total assertions. Each entry can be clicked to get further details.

QUnit - API

Important APIs of QUnit

Some of the important Category of QUnit are −

S.No	Category	Functionality
1	Assert	A set of assert methods.

2	Async Control	For asynchronous operations.
3	Callbacks	When integrating QUnit into other tools such as CI servers, these callbacks can be used as an API to read the test results.
4	Configuration and Utilities	These methods and properties are used as utility helpers and to configure QUnit. For example, to adjust the runtime behavior directly, extend the QUnit API via custom assertions, etc.
5	Test	For testing operations.

Category: Assert

It provides a set of assert methods.

S.No.	Methods & Description
1	**async()** Instruct QUnit to wait for an asynchronous operation.
2	**deepEqual()** A deep recursive comparison, working on primitive types, arrays, objects, regular expressions, dates, and functions.
3	**equal()** A non-strict comparison, roughly equivalent to JUnit's assertEquals.
4	**expect()** Specify how many assertions are expected to run within a test.
5	**notDeepEqual()** An inverted deep recursive comparison, working on primitive types, arrays, objects, regular expressions, dates, and functions.
6	**notEqual()** A non-strict comparison, checking for inequality.
7	**notOk()** A boolean check, inverse of ok() and CommonJS's assert.ok(), and equivalent to JUnit's assertFalse(). Passes if the first argument is false.
8	**notPropEqual()** A strict comparison of an object's own properties, checking for inequality.
9	**notStrictEqual()** A strict comparison, checking for inequality.
10	**ok()** A boolean check, equivalent to CommonJS's assert.ok() and JUnit's assertTrue(). Passes if the first argument is true.
11	**propEqual()** A strict type and value comparison of an object's own properties.
12	**push()** Report the result of a custom assertion.

13	**strictEqual()** A strict type and value comparison.
14	**throws()** Test if a callback throws an exception, and optionally compare the thrown error.

Category: Async Control

It provides a set of asynchronous operations.

S.No	Methods & Description
1	**async()** Instruct QUnit to wait for an asynchronous operation.
2	**QUnit.asyncTest()** DEPRECATED: Add an asynchronous test to run. The test must include a call to QUnit.start().
3	**QUnit.start()** PARTIALLY DEPRECATED: Start running the tests again after the testrunner was stopped. See QUnit.stop() and QUnit.config.autostart.
4	**QUnit.stop()** DEPRECATED: Increase the number of QUnit.start() calls the testrunner should wait for before continuing.
5	**QUnit.test()** Add a test to run.

Category: Callbacks

When integrating QUnit into other tools like CI servers, these callbacks can be used as an API to read the test results.

S.No	Methods & Description
1	**QUnit.begin()** Register a callback to fire whenever the test suite begins.
2	**QUnit.done()** Register a callback to fire whenever the test suite ends.
3	**QUnit.log()** Register a callback to fire whenever an assertion completes.
4	**QUnit.moduleDone()** Register a callback to fire whenever a module ends.
5	**QUnit.moduleStart()** Register a callback to fire whenever a module begins.

6	**QUnit.testDone()** Register a callback to fire whenever a test ends.
7	**QUnit.testStart()** Register a callback to fire whenever a test begins.

Category: Configuration and Utilities

These methods and properties are used as utility helpers and to configure QUnit. For example, to adjust the runtime behavior directly, extend the QUnit API via custom assertions, etc.

S.No	Methods & Description
1	**QUnit.assert** Namespace for QUnit assertions.
2	**QUnit.config** Configuration for QUnit.
3	**QUnit.dump.parse()** Advanced and extensible data dumping for JavaScript.
4	**QUnit.extend()** Copy the properties defined by the mixin object into the target object.
5	**QUnit.init()** DEPRECATED: Re-initialize the test runner.
6	**QUnit.push()** DEPRECATED: Report the result of a custom assertion.
7	**QUnit.reset()** DEPRECATED: Reset the test fixture in the DOM.
8	**QUnit.stack()** Returns a single line string representing the stacktrace (call stack).

Category: Test

It provides a set of testing operations.

S.No	Methods & Description
1	**QUnit.assert** Namespace for QUnit assertions.
2	**QUnit.asyncTest()** DEPRECATED: Add an asynchronous test to run. The test must include a call to QUnit.start().

3	**QUnit.module()** Group related tests under a single label.
4	**QUnit.only()** Adds a test to exclusively run, preventing all other tests from running.
5	**QUnit.skip()** Adds a test like object to be skipped.
6	**QUnit.test()** Adds a test to run.

QUnit - Using Assertions

All the assertions are in the Assert Category.

This category provides a set of assertion methods useful for writing tests. Only failed assertions are recorded.

S.No	Methods & Description
1	**async()** Instruct QUnit to wait for an asynchronous operation.
2	**deepEqual()** A deep recursive comparison, working on primitive types, arrays, objects, regular expressions, dates, and functions.
3	**equal()** A non-strict comparison, roughly equivalent to JUnit's assertEquals.
4	**expect()** Specify how many assertions are expected to run within a test.
5	**notDeepEqual()** An inverted deep recursive comparison, working on primitive types, arrays, objects, regular expressions, dates, and functions.
6	**notEqual()** A non-strict comparison, checking for inequality.
7	**notOk()** A boolean check, inverse of ok() and CommonJS's assert.ok(), and equivalent to JUnit's assertFalse(). Passes if the first argument is false.
8	**notPropEqual()** A strict comparison of an object's own properties, checking for inequality.
9	**notStrictEqual()** A strict comparison, checking for inequality.
10	**ok()**

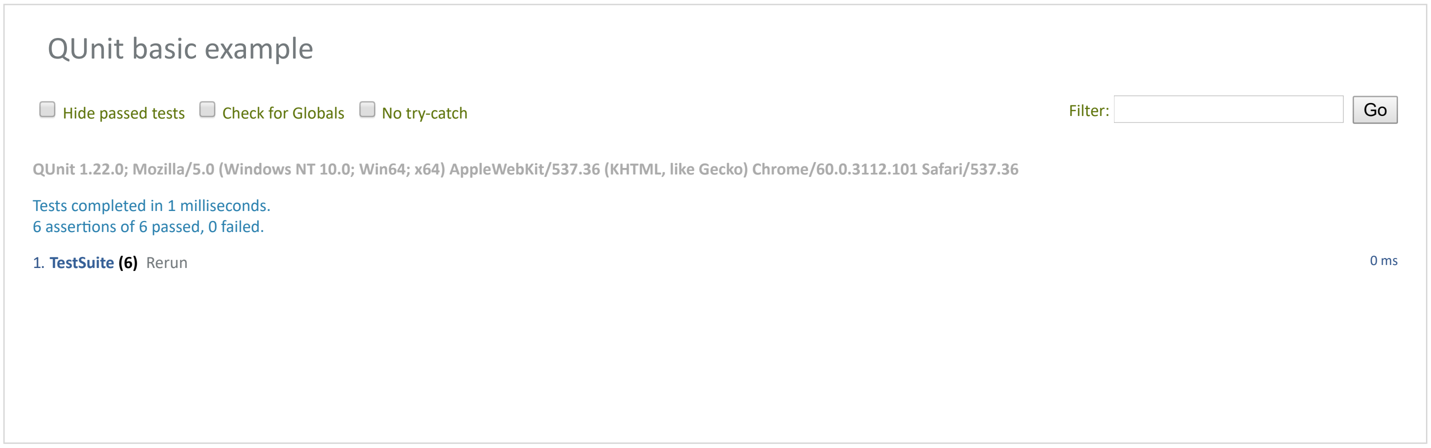
	A boolean check, equivalent to CommonJS's assert.ok() and JUnit's assertTrue(). Passes if the first argument is true.
11	**propEqual()** A strict type and value comparison of an object's own properties.
12	**push()** Report the result of a custom assertion.
13	**strictEqual()** A strict type and value comparison.
14	**throws()** Test if a callback throws an exception, and optionally compare the thrown error.

Let's try to cover most of the above mentioned methods in an example.

```
<html>
   <head>
      <meta charset = "utf-8">
      <title>QUnit basic example</title>
      <link rel = "stylesheet" href = "https://code.jquery.com/qunit/qunit-1.22.0.css">
      <script src = "https://code.jquery.com/qunit/qunit-1.22.0.js"></script>
   </head>

   <body>
      <div id = "qunit"></div>
      <div id = "qunit-fixture"></div>
      <script>
         QUnit.test( "TestSuite", function( assert ) {
            //test data
            var str1 = "abc";
            var str2 = "abc";
            var str3 = null;
            var val1 = 5;
            var val2 = 6;
            var expectedArray = ["one", "two", "three"];
            var resultArray =  ["one", "two", "three"];

            //Check that two objects are equal
            assert.equal(str1, str2, "Strings passed are equal.");

            //Check that two objects are not equal
            assert.notEqual(str1,str3, "Strings passed are not equal.");

            //Check that a condition is true
            assert.ok(val1 < val2, val1 + " is less than " + val2);

            //Check that a condition is false
            assert.notOk(val1 > val2, val2 + " is not less than " + val1);

            //Check whether two arrays are equal to each other.
            assert.deepEqual(expectedArray, resultArray ,"Arrays passed are equal.");

            //Check whether two arrays are equal to each other.
            assert.notDeepEqual(expectedArray, ["one", "two"],
               "Arrays passed are not equal.");
         });
      </script>
   </body>
</html>
```

Verify the Output

You should see the following result –


QUnit basic example

Hide passed tests Check for Globals No try-catch

Filter: Go

QUnit 1.22.0; Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/60.0.3112.101 Safari/537.36

Tests completed in 1 milliseconds.
6 assertions of 6 passed, 0 failed.

1. **TestSuite (6)** Rerun 0 ms

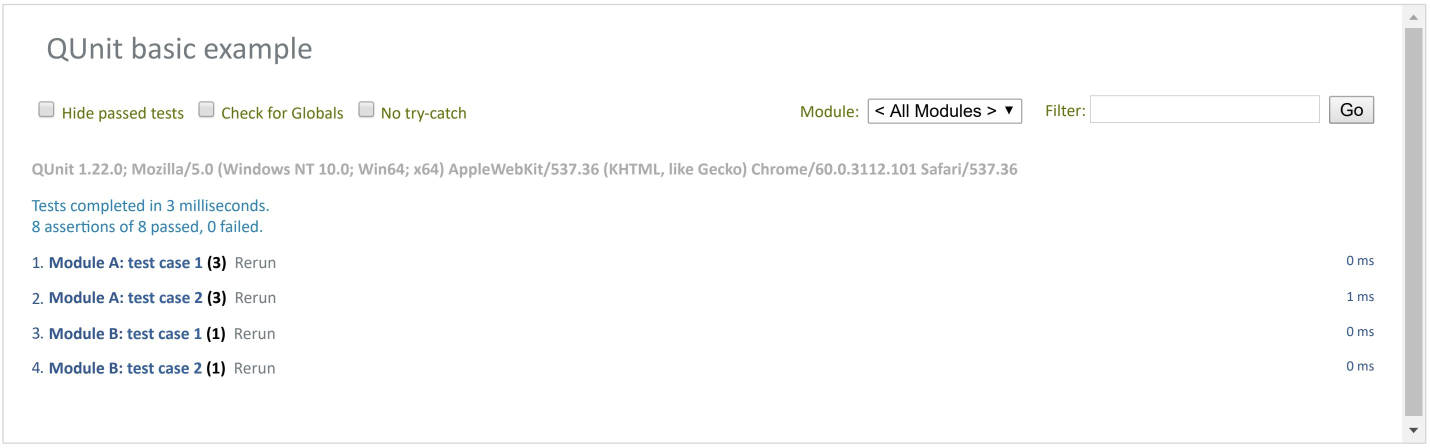
QUnit - Execution Procedure

This chapter explains the execution procedure of methods in QUnit, that states which method is called first and which one after that. Following is the execution procedure of the QUnit test API methods with an example.

```
<html>
   <head>
      <meta charset = "utf-8">
      <title>QUnit basic example</title>
      <link rel = "stylesheet" href = "https://code.jquery.com/qunit/qunit-1.22.0.css">
      <script src = "https://code.jquery.com/qunit/qunit-1.22.0.js"></script>
   </head>

   <body>
      <div id = "qunit"></div>
      <div id = "qunit-fixture"></div>
      <script>
         QUnit.module( "Module A", {
            beforeEach: function( assert ) {
               assert.ok( true, "before test case" );
            }, afterEach: function( assert ) {
               assert.ok( true, "after test case" );
            }
         });

         QUnit.test( "test case 1", function( assert ) {
            assert.ok( true, "Module A: in test case 1" );
         });

         QUnit.test( "test case 2", function( assert ) {
            assert.ok( true, "Module A: in test case 2" );
         });

         QUnit.module( "Module B" );
         QUnit.test( "test case 1", function( assert ) {
            assert.ok( true, "Module B: in test case 1" );
         });

         QUnit.test( "test case 2", function( assert ) {
            assert.ok( true, "Module B: in test case 2" );
         });
      </script>
   </body>
</html>
```

Verify the Output

You should see the following result –


QUnit basic example

Hide passed tests Check for Globals No try-catch Module: < All Modules > Filter: Go

QUnit 1.22.0; Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/60.0.3112.101 Safari/537.36

Tests completed in 3 milliseconds.
8 assertions of 8 passed, 0 failed.

1. **Module A: test case 1 (3)** Rerun 0 ms
2. **Module A: test case 2 (3)** Rerun 1 ms
3. **Module B: test case 1 (1)** Rerun 0 ms
4. **Module B: test case 2 (1)** Rerun 0 ms

This is how the QUnit execution procedure is.

The module is used to group test cases.

beforeEach() method executes for each test case however before executing the test case.

afterEach() method executes for each test case however after the execution of test case.

In between **beforeEach()** and **afterEach()** each test case executes.

Calling **QUnit.module()** again, simply reset any beforeEach/afterEach functions defined by another module previously.

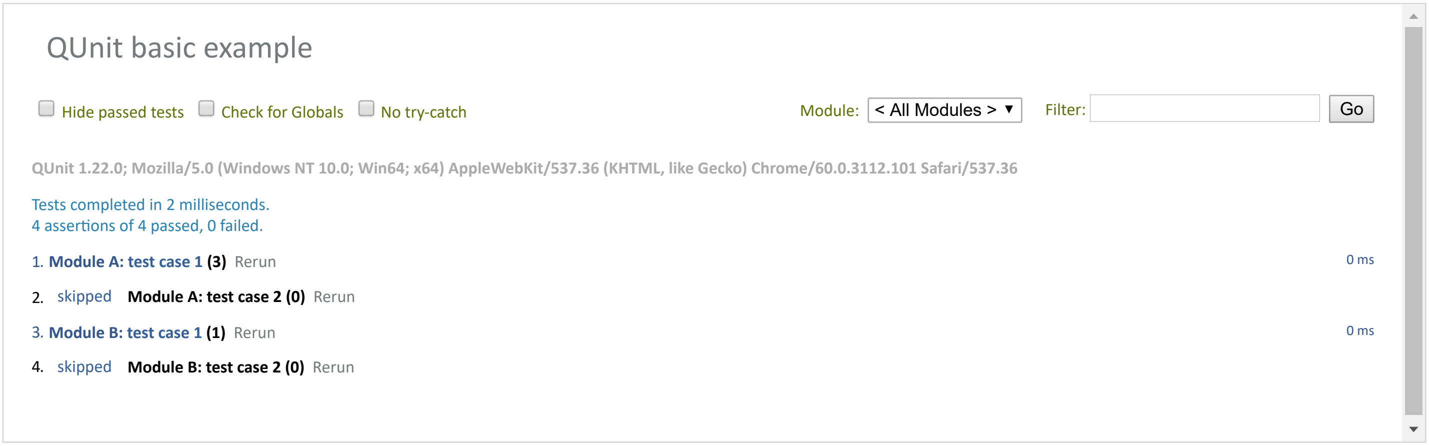
QUnit - Skip Test

Sometimes it happens that our code is not ready and the test case written to test that method/code fails if run. **QUnit.skip** helps in this regards. A test method written using Skip method will not be executed. Let's see the Skip method in action.

```
<html>
   <head>
      <meta charset = "utf-8">
      <title>QUnit basic example</title>
      <link rel = "stylesheet" href = "https://code.jquery.com/qunit/qunit-1.22.0.css">
      <script src = "https://code.jquery.com/qunit/qunit-1.22.0.js"></script>
   </head>

   <body>
      <div id = "qunit"></div>
      <div id = "qunit-fixture"></div>
      <script>
         QUnit.module( "Module A", {
            beforeEach: function( assert ) {
               assert.ok( true, "before test case" );
            }, afterEach: function( assert ) {
               assert.ok( true, "after test case" );
            }
         });

         QUnit.test( "test case 1", function( assert ) {
            assert.ok( true, "Module A: in test case 1" );
         });

         QUnit.skip( "test case 2", function( assert ) {
            assert.ok( true, "Module A: in test case 2" );
         });

         QUnit.module( "Module B" );
         QUnit.test( "test case 1", function( assert ) {
            assert.ok( true, "Module B: in test case 1" );
         });

         QUnit.skip( "test case 2", function( assert ) {
            assert.ok( true, "Module B: in test case 2" );
         });
      </script>
   </body>
</html>
```

Verify the Output

You should see the following result −


QUnit basic example

Hide passed tests   Check for Globals   No try-catch   Module: < All Modules >   Filter: Go

QUnit 1.22.0; Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/60.0.3112.101 Safari/537.36

Tests completed in 2 milliseconds.
4 assertions of 4 passed, 0 failed.

1. Module A: test case 1 (3) Rerun   0 ms
2. skipped   Module A: test case 2 (0) Rerun
3. Module B: test case 1 (1) Rerun   0 ms
4. skipped   Module B: test case 2 (0) Rerun

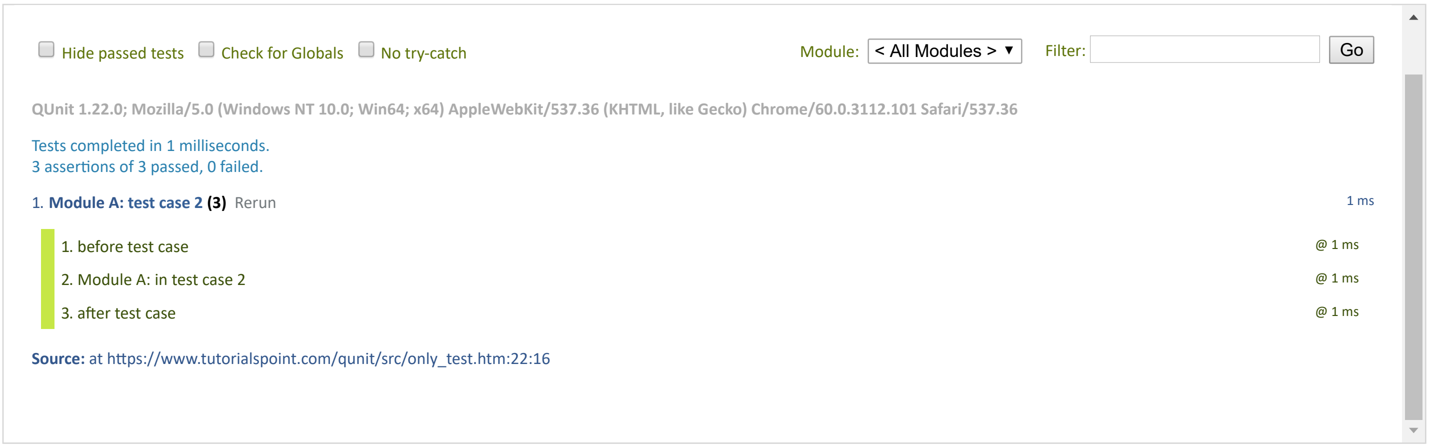
QUnit - Only Test

Sometimes it happens that our code is not ready and the test case written to test that method/code fails, if run. **QUnit.only** helps in this regards. A test method written using only method will be executed while other tests will not run. If more than one only methods are specified, then only the first one will execute. Let's see only method in action.

```
<html>
   <head>
      <meta charset = "utf-8">
      <title>QUnit basic example</title>
      <link rel = "stylesheet" href = "https://code.jquery.com/qunit/qunit-1.22.0.css">
      <script src = "https://code.jquery.com/qunit/qunit-1.22.0.js"></script>
   </head>

   <body>
      <div id = "qunit"></div>
      <div id = "qunit-fixture"></div>
      <script>
         QUnit.module( "Module A", {
            beforeEach: function( assert ) {
               assert.ok( true, "before test case" );
            }, afterEach: function( assert ) {
               assert.ok( true, "after test case" );
            }
         });

         QUnit.test( "test case 1", function( assert ) {
            assert.ok( true, "Module A: in test case 1" );
         });

         QUnit.only( "test case 2", function( assert ) {
            assert.ok( true, "Module A: in test case 2" );
         });

         QUnit.test( "test case 3", function( assert ) {
            assert.ok( true, "Module A: in test case 3" );
         });

         QUnit.test( "test case 4", function( assert ) {
            assert.ok( true, "Module A: in test case 4" );
         });
      </script>
   </body>
</html>
```

Verify the Output

You should see the following result −



QUnit 1.22.0; Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/60.0.3112.101 Safari/537.36

Tests completed in 1 milliseconds.
3 assertions of 3 passed, 0 failed.

1. Module A: test case 2 (3) Rerun 1 ms

1. before test case @ 1 ms
2. Module A: in test case 2 @ 1 ms
3. after test case @ 1 ms

Source: at https://www.tutorialspoint.com/qunit/src/only_test.htm:22:16

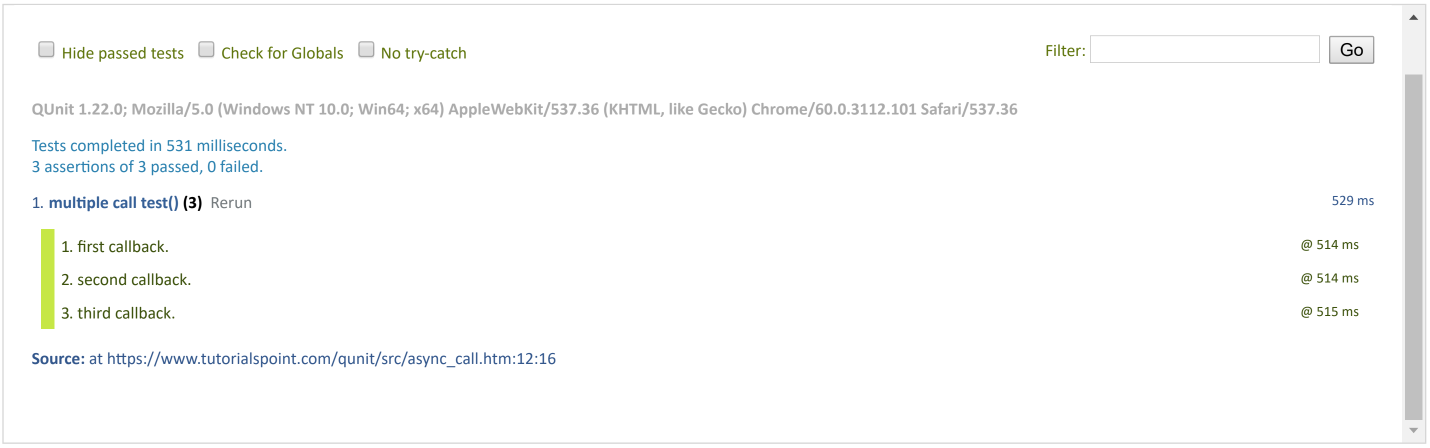
QUnit - Async Call

For every asynchronous operation in **QUnit.test()** callback, use **assert.async()**, which returns a "done" function that should be called when the operation has completed. assert.async() accepts call counts as a parameter. The callback returned from assert.async() will throw an Error, if it is invoked more than the accepted call count, if provided. Each **done()** call adds up to the call count. After every call gets completed, the test is done.

```
<html>
   <head>
      <meta charset = "utf-8">
      <title>QUnit basic example</title>
      <link rel = "stylesheet" href = "https://code.jquery.com/qunit/qunit-1.22.0.css">
      <script src = "https://code.jquery.com/qunit/qunit-1.22.0.js"></script>
   </head>

   <body>
      <div id = "qunit"></div>
      <div id = "qunit-fixture"></div>
      <script>
         QUnit.test( "multiple call test()", function( assert ) {
            var done = assert.async( 3 );

            setTimeout(function() {
               assert.ok( true, "first callback." );
               done();
            }, 500 );

            setTimeout(function() {
               assert.ok( true, "second callback." );
               done();
            }, 500 );

            setTimeout(function() {
               assert.ok( true, "third callback." );
               done();
            }, 500 );
         });
      </script>
   </body>
</html>
```

Verify the Output

You should see the following result –



```
Hide passed tests   Check for Globals   No try-catch                         Filter:            Go

QUnit 1.22.0; Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/60.0.3112.101 Safari/537.36

Tests completed in 531 milliseconds.
3 assertions of 3 passed, 0 failed.

1. multiple call test() (3)  Rerun                                                              529 ms

   1. first callback.                                                                        @ 514 ms
   2. second callback.                                                                       @ 514 ms
   3. third callback.                                                                        @ 515 ms

Source: at https://www.tutorialspoint.com/qunit/src/async_call.htm:12:16
```

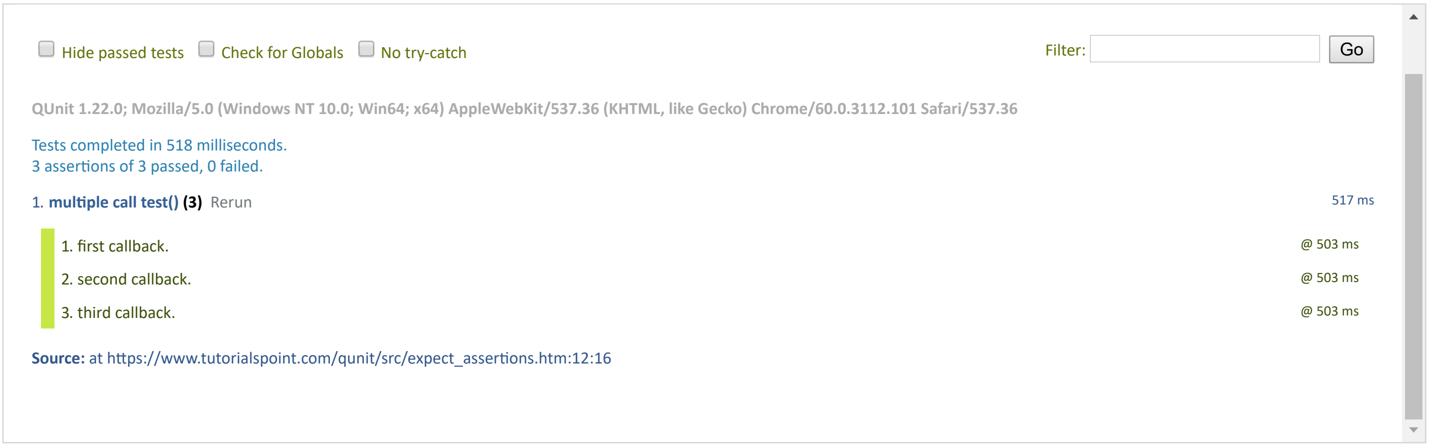
QUnit - Expect Assertions

We can use **assert.expect()** function to check the number of assertions made in the test. In the following example, we're expecting three assertions to be made in the test.

```
<html>
   <head>
      <meta charset = "utf-8">
      <title>QUnit basic example</title>
      <link rel = "stylesheet" href = "https://code.jquery.com/qunit/qunit-1.22.0.css">
      <script src = "https://code.jquery.com/qunit/qunit-1.22.0.js"></script>
   </head>

   <body>
      <div id = "qunit"></div>
      <div id = "qunit-fixture"></div>
      <script>
         QUnit.test( "multiple call test()", function( assert ) {
            assert.expect( 3 );
            var done = assert.async( 3 );

            setTimeout(function() {
               assert.ok( true, "first callback." );
               done();
            }, 500 );

            setTimeout(function() {
               assert.ok( true, "second callback." );
               done();
            }, 500 );

            setTimeout(function() {
               assert.ok( true, "third callback." );
               done();
            }, 500 );
         });
      </script>
   </body>
</html>
```

Verify the Output

You should see the following result –



```
Hide passed tests   Check for Globals   No try-catch                         Filter:            Go

QUnit 1.22.0; Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/60.0.3112.101 Safari/537.36

Tests completed in 518 milliseconds.
3 assertions of 3 passed, 0 failed.

1. multiple call test() (3)  Rerun                                                              517 ms

   1. first callback.                                                                        @ 503 ms
   2. second callback.                                                                       @ 503 ms
   3. third callback.                                                                        @ 503 ms

Source: at https://www.tutorialspoint.com/qunit/src/expect_assertions.htm:12:16
```

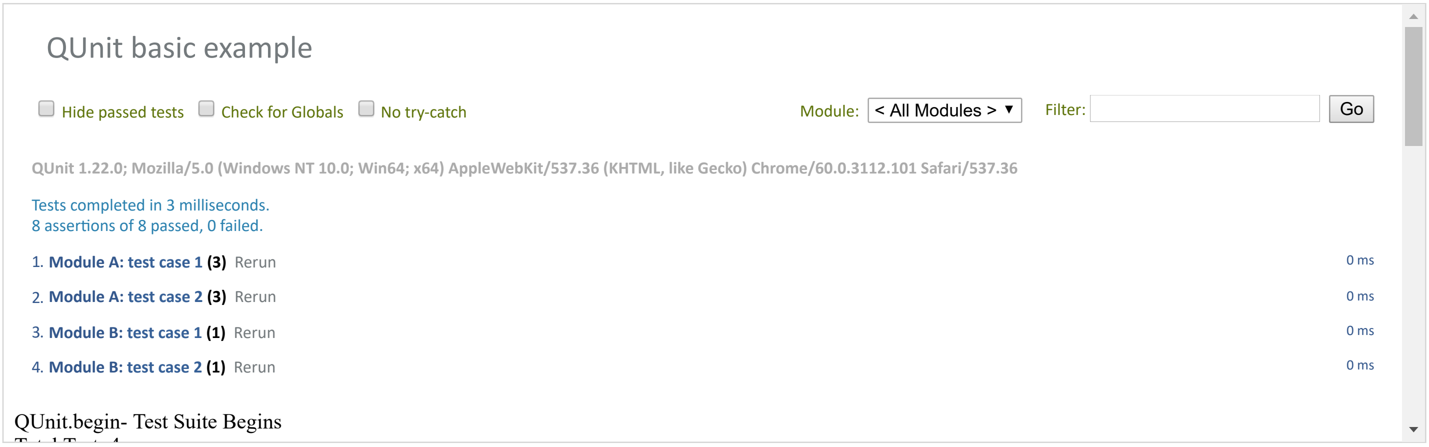
QUnit - Callbacks

When integrating QUnit into other tools like CI servers, these callbacks can be used as an API to read test results. Following is the execution procedure of the QUnit callback API method with an example.

```
<html>
   <head>
      <meta charset = "utf-8">
      <title>QUnit basic example</title>
      <link rel = "stylesheet" href = "https://code.jquery.com/qunit/qunit-1.22.0.css">
      <script src = "https://code.jquery.com/qunit/qunit-1.22.0.js"></script>
   </head>

   <body>
      <div id = "qunit"></div>
      <div id = "qunit-fixture"></div>
      <script>
         //Register a callback to fire whenever a testsuite starts.
         QUnit.begin(function( details ) {
            var data = document.getElementById("console").innerHTML;
            document.getElementById("console").innerHTML = "<br/>" +
               "QUnit.begin- Test Suite Begins " + "<br/>" +
               "Total Test: " + details.totalTests;
         });

         //Register a callback to fire whenever a test suite ends.
         QUnit.done(function( details ) {
            var data = document.getElementById("console").innerHTML;
            document.getElementById("console").innerHTML = data + "<br/><br/>" +
               "QUnit.done - Test Suite Finised" +  "<br/>" + "Total: " +
               details.total + " Failed: " + details.failed + " Passed:
               " + details.passed;
         });

         //Register a callback to fire whenever a module starts.
            QUnit.moduleStart(function( details ) {
               var data = document.getElementById("console").innerHTML;
               document.getElementById("console").innerHTML = data + "<br/><br/>" +
                  "QUnit.moduleStart - Module Begins " +  "<br/>" + details.name;
         });

         //Register a callback to fire whenever a module ends.
         QUnit.moduleDone(function( details ) {
            var data = document.getElementById("console").innerHTML;
            document.getElementById("console").innerHTML = data + "<br/><br/>" +
               "QUnit.moduleDone - Module Finished " +  "<br/>" + details.name +
               " Failed/total: " + details.failed +"/" + details.total ;
         });

         //Register a callback to fire whenever a test starts.
         QUnit.testStart(function( details ) {
            var data = document.getElementById("console").innerHTML;
            document.getElementById("console").innerHTML = data + "<br/><br/>" +
               "QUnit.testStart - Test Begins " +  "<br/>" + details.module +"
               " + details.name;
         });

         //Register a callback to fire whenever a test ends.
         QUnit.testDone(function( details ) {
            var data = document.getElementById("console").innerHTML;
            document.getElementById("console").innerHTML = data + "<br/><br/>" +
               "QUnit.testDone - Test Finished " +  "<br/>" + details.module +" "
               + details.name + "Failed/total: " + details.failed +" " + details.total+
               " "+ details.duration;
         });

         QUnit.module( "Module A", {
            beforeEach: function( assert ) {
               assert.ok( true, "before test case" );
            }, afterEach: function( assert ) {
               assert.ok( true, "after test case" );
            }
         });

         QUnit.test( "test case 1", function( assert ) {
            assert.ok( true, "Module A: in test case 1" );
         });

         QUnit.test( "test case 2", function( assert ) {
            assert.ok( true, "Module A: in test case 2" );
         });

         QUnit.module( "Module B" );
         QUnit.test( "test case 1", function( assert ) {
            assert.ok( true, "Module B: in test case 1" );
         });

         QUnit.test( "test case 2", function( assert ) {
            assert.ok( true, "Module B: in test case 2" );
         });
      </script>

      <div id = "console" ></div>
   </body>
</html>
```

Verify the Output

You should see the following result −



```
QUnit 1.22.0; Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/60.0.3112.101 Safari/537.36

Tests completed in 3 milliseconds.
8 assertions of 8 passed, 0 failed.

1. Module A: test case 1 (3) Rerun     0 ms
2. Module A: test case 2 (3) Rerun     0 ms
3. Module B: test case 1 (1) Rerun     0 ms
4. Module B: test case 2 (1) Rerun     0 ms

QUnit.begin- Test Suite Begins
```

QUnit - Nested Modules

Modules with grouped test functions are used to define nested modules. QUnit run tests on the parent module before going deep on the nested ones, even if they're declared first. The **beforeEach** and **afterEach** callbacks on a nested module call will stack in LIFO (Last In, First Out) Mode to the parent hooks. You can specify the code to run before and after each test using the argument and hooks.

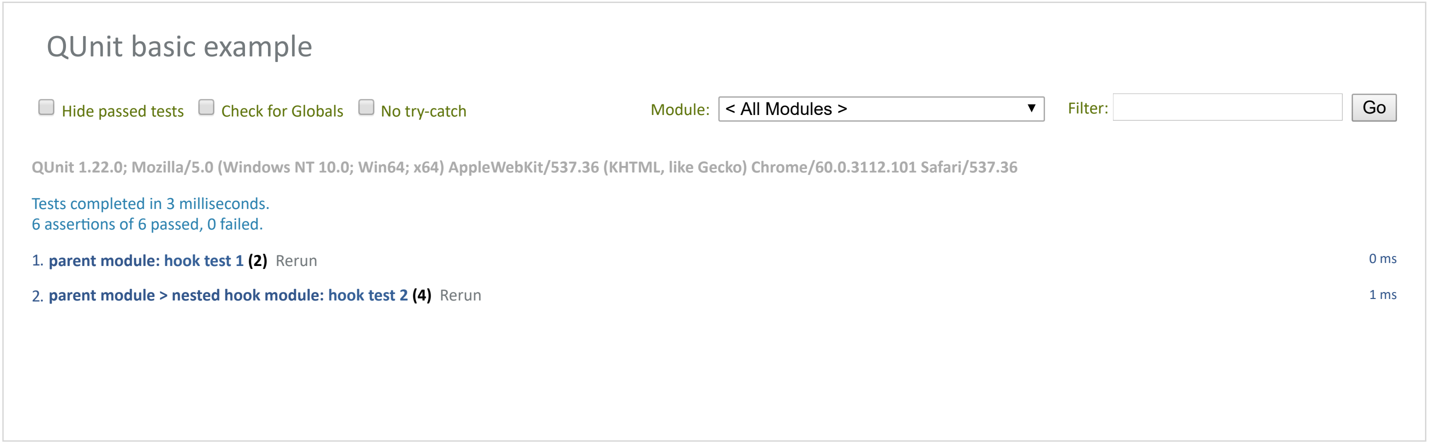
Hooks can also be used to create properties that will be shared on the context of each test. Any additional properties on the hooks object will be added to that context. The hooks argument is optional if you call QUnit.module with a callback argument.

The module's callback is invoked having the context as the test environment, with the environment's properties copied to the module's tests, hooks, and nested modules.

```
<html>
   <head>
      <meta charset = "utf-8">
      <title>QUnit basic example</title>
      <link rel = "stylesheet" href = "https://code.jquery.com/qunit/qunit-1.22.0.css">
      <script src = "https://code.jquery.com/qunit/qunit-1.22.0.js"></script>
   </head>

   <body>
      <div id = "qunit"></div>
      <div id = "qunit-fixture"></div>
      <script>
         QUnit.module( "parent module", function( hooks ) {
            hooks.beforeEach( function( assert ) {
               assert.ok( true, "beforeEach called" );
            });

            hooks.afterEach( function( assert ) {
               assert.ok( true, "afterEach called" );
            });

            QUnit.test( "hook test 1", function( assert ) {
               assert.expect( 2 );
            });

            QUnit.module( "nested hook module", function( hooks ) {
               // This will run after the parent module's beforeEach hook
               hooks.beforeEach( function( assert ) {
                  assert.ok( true, "nested beforeEach called" );
               });

               // This will run before the parent module's afterEach
               hooks.afterEach( function( assert ) {
                  assert.ok( true, "nested afterEach called" );
               });

               QUnit.test( "hook test 2", function( assert ) {
                  assert.expect( 4 );
               });
            });
         });
      </script>

      <div id = "console" ></div>
   </body>
</html>
```

Verify the Output

You should see the following result –



QUnit basic example

Hide passed tests   Check for Globals   No try-catch   Module: < All Modules >   Filter: Go

QUnit 1.22.0; Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/60.0.3112.101 Safari/537.36

Tests completed in 3 milliseconds.
6 assertions of 6 passed, 0 failed.

1. **parent module: hook test 1 (2)** Rerun   0 ms

2. **parent module > nested hook module: hook test 2 (4)** Rerun   1 ms