
QUnit

QUnit is a JavaScript unit testing framework.

Minimal Setup

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>QUnit Example</title>
  <link rel="stylesheet" href="//code.jquery.com/qunit/qunit-1.18.0.
css">
</head>
<body>
  <div id="qunit"></div>
  <div id="qunit-fixture"></div>
  <script src="//code.jquery.com/qunit/qunit-1.18.0.js"></script>
  <script src="tests.js"></script>
```

```
</body>
</html>
```

Assertions

- `ok(truthy [, message]` - A boolean check, passes if the first argument is truthy.

```
QUnit.test( "ok test", function( assert ) {
  assert.ok( true, "true succeeds" );
  assert.ok( "non-empty", "non-empty string succeeds" );

  assert.ok( false, "false fails" );
  assert.ok( 0, "0 fails" );
  assert.ok( NaN, "NaN fails" );
  assert.ok( "", "empty string fails" );
  assert.ok( null, "null fails" );
  assert.ok( undefined, "undefined fails" );
});
```

- `equal(value, expected[, message])` - Verify the value provided is equal the expected parameter using a non-strict comparison (==).

```
QUnit.test( "equal test", function( assert ) {
  assert.equal( 0, 0, "Zero, Zero; equal succeeds" );
  assert.equal( "", 0, "Empty, Zero; equal succeeds" );
  assert.equal( "", "", "Empty, Empty; equal succeeds" );
  assert.equal( 0, false, "Zero, false; equal succeeds" );

  assert.equal( "three", 3, "Three, 3; equal fails" );
  assert.equal( null, false, "null, false; equal fails" );
});
```

Compared to `ok()`, `equal()` makes it much easier to debug tests that failed, because it's obvious which value caused the test to fail.

When you need a strict comparison `(===)`, use `strictEqual()` instead.

- `deepEqual(value, expected[, message])` - A recursive, strict comparison that works on all the JavaScript types. The assertion passes if value and expected are identical in terms of properties, values, and they have the same prototype.

```
QUnit.test( "deepEqual test", function( assert ) {
  var obj = { foo: "bar" };

  assert.deepEqual( obj, { foo: "bar" }, "Two objects can be the sam
```

```
e in value" );
});
```

- `notDeepEqual(actual, expected[, message])` - Same as deepEqual() but tests for inequality.

```
QUnit.test( "notDeepEqual test", function( assert ) {
  var obj = { foo: "bar" };

  assert.notDeepEqual( obj, { foo: "bla" }, "Different object, same
 key, different value, not equal" );
});
```

- `notEqual(actual, expected[, message])` - A non-strict comparison, checking for inequality.

```
QUnit.test( "a test", function( assert ) {
  assert.notEqual( 1, "2", "String '2' and number 1 don't have the s
ame value" );
});
```

- `propEqual(value, expected[, message])` - A strict comparison of the properties and values of an object. The assertion passes if all the properties and the values are identical.

```
QUnit.test( "propEqual test", function( assert ) {
  function Foo( x, y, z ) {
    this.x = x;
    this.y = y;
    this.z = z;
  }
  Foo.prototype.doA = function () {};
  Foo.prototype.doB = function () {};
  Foo.prototype.bar = 'prototype';

  var foo = new Foo( 1, "2", [] );
  var bar = {
    x : 1,
    y : "2",
    z : []
  };
  assert.propEqual( foo, bar, "Strictly the same properties without
 comparing objects constructors." );
});
```

- `strictEqual(actual, expected[, message])` - Verify the value provided is equal to the expected parameter using a strict comparison (===);

```
QUnit.test( "strictEqual test", function( assert ) {
  assert.strictEqual( 1, 1, "1 and 1 have the same value and type"
);
});
```

- `notPropEqual(actual, expected[, message])` - A strict comparison of an object's own properties, checking for inequality.

```
QUnit.test( "notPropEqual test", function( assert ) {
  function Foo( x, y, z ) {
    this.x = x;
    this.y = y;
    this.z = z;
  }
  Foo.prototype.doA = function () {};
  Foo.prototype.doB = function () {};
  Foo.prototype.bar = 'prototype';

    var foo = new Foo( 1, "2", [] );
    var bar = new Foo( "1", 2, {} );
  assert.notPropEqual( foo, bar, "Properties values are strictly com
pared." );
});
```

- `notStrictEqual(actual, expected[, message])` - Same as strictEqual() but tests for inequality

```
QUnit.test( "a test", function( assert ) {
  assert.notStrictEqual( 1, "1", "String '1' and number 1 have the s
ame value but not the same type" );
});
```

- `throws(function [, expected ] [, message ])` - Test if a callback throws an exception, and optionally compare the thrown error;

```
QUnit.test( "throws", function( assert ) {

  function CustomError( message ) {
    this.message = message;
  }

  CustomError.prototype.toString = function() {
    return this.message;
  };

  assert.throws(
    function() {
      throw "error"
```

```
    },
    "throws with just a message, not using the 'expected' argument"
  );

  assert.throws(
    function() {
      throw new CustomError("some error description");
    },
    /description/,
    "raised error message contains 'description'"
  );

  assert.throws(
    function() {
      throw new CustomError();
    },
    CustomError,
    "raised error is an instance of CustomError"
  );

  assert.throws(
    function() {
      throw new CustomError("some error description");
    },
    new CustomError("some error description"),
    "raised error instance matches the CustomError instance"
  );

  assert.throws(
    function() {
      throw new CustomError("some error description");
    },
    function( err ) {
      return err.toString() === "some error description";
    },
    "raised error instance satisfies the callback function"
  );
});
```