QUNIT FUNDAMENTALS

This document guides:

* What is QUnit?
* How to use it for testing?

# What is QUnit

References:

<http://qunitjs.com/>

<https://github.com/jquery/qunit>

QUnit: JavaScript unit testing framework. Current version is 1.15.0.

# Setting up QUnit

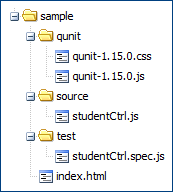
You need two things to use QUnit: qunit.js and qunit.css. You can use them at remote side OR download them to use at your local side.

To download them, use can access to:

<http://code.jquery.com/qunit/qunit-1.15.0.js>

<http://code.jquery.com/qunit/qunit-1.15.0.css>

Sample, I organizate my folders and files as:



qunit folder contains all libraries from qunit. source folder contains actually javascript source. test folder contains testing script files.

# A simple example

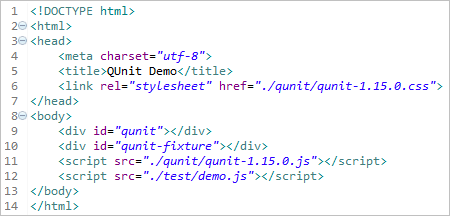
## Step 1, create file contains cases

We need file to contain one/group of case(s). I create new test file at sample/test/demo.js. I use ok function from QUnit to create test case as:



## Step 2, setup index file

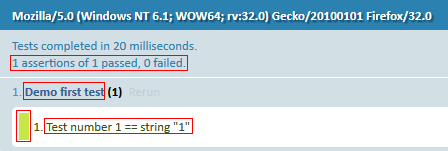
This file is used to include one/some files(s) that contain(s) cases. Modifying index.html file to test demo.js as:



Line 6 and line 11 use resource from QUnit. Line 9 and line 10 use to display the testing result. They are required. Line 12 uses to refer to test files.

## Step 3, use browser to view results

Using browser to open the file and viewing result as:



Description:

* 1 assertions of 1 passed, 0 failed means {passed assertions} assertions of {total of assertions} passed, {failed assertions } failed.
* Demo first test is test case name.
* Test number 1 == string "1" is assertion name.
* Green color says that one assertion is passed.

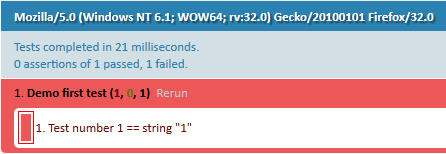
You can click on Rerun link at a specific case to only re-run this case.

## Try to view failed assertion

Trying to change the condition of test file to return false result as:



And refresh browser. You can see as:

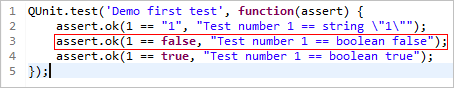


Description:

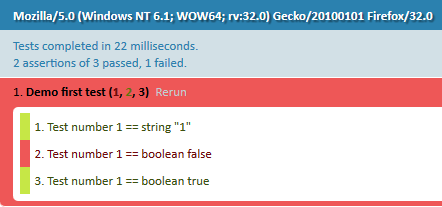
* Demo first test (1, 0, 1) means Demo first test ({failed assertions}, {passed assertions}, {total of assertions}).
* Red color says that one assertion is failed.

## Try to mix passed assertions and failed assertions

I try to create case that contains both passed and failed assertions as:



When refresh browser, you can see as:



# Assert – ok() function

Method to check boolean value.

Structure:

ok(state[, message])

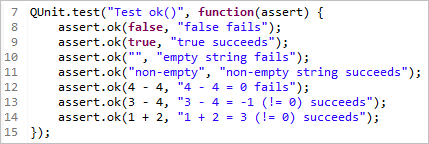
Description for structure:

* state is an expression. Required. Focusing on expression.
* message is a string. Optional.

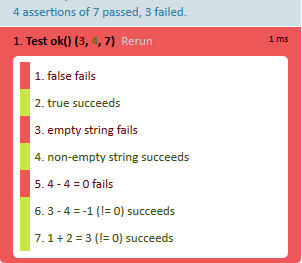
## Using with primitive types

Primitive types consists of: boolean, string, number

Sample as:



The result as:

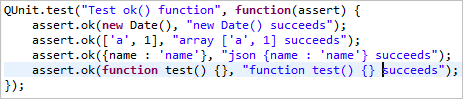


Remarks: false, empty string and zero number mean false.

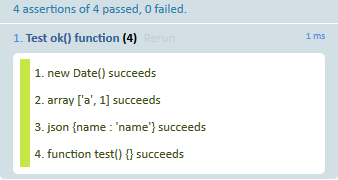
## Using with object

Object consists of: date, array, json, function …

Sample as:



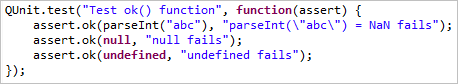
The result as:



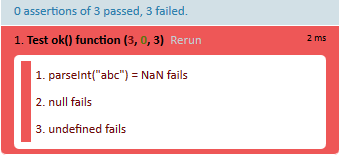
## Using with special types

Special types consist of: NaN, null and undefined.

Sample as:



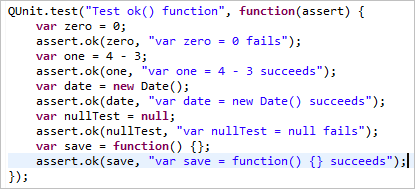
The result as:



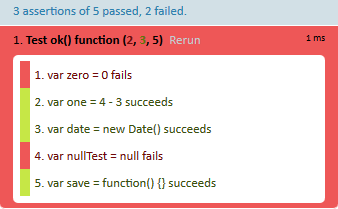
## Using with variables

It depends on the value of variable to return true/false.

Sample as:



The result as:



# Assert – equal() function

The equal assertion uses the simple comparison operator (==) to compare the actual and expected arguments. When they are equal, the assertion passes; otherwise, it fails. When it fails, both actual and expected values are displayed in the test result, in addition to a given message.

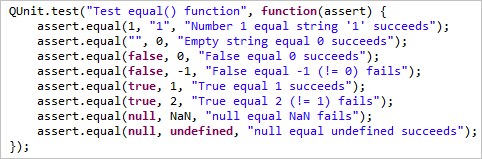
Structure:

equal(actual, expected[, message])

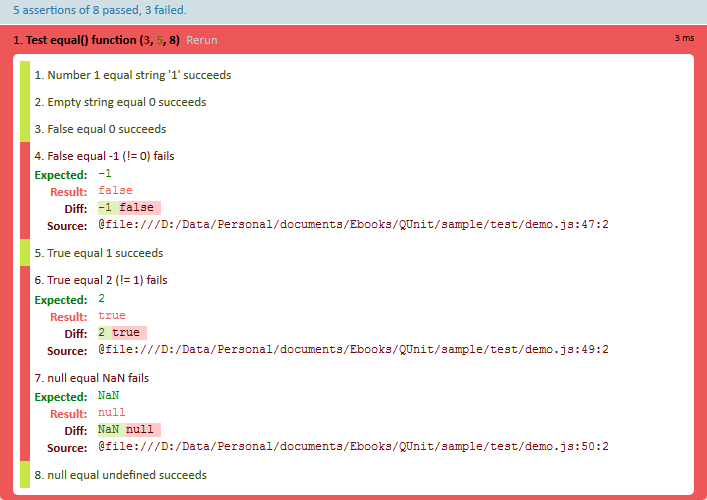
Description:

* actual is an expression. Required.
* expected is an expression. Required.
* message is a string. Optional.

Sample as:



The result as:



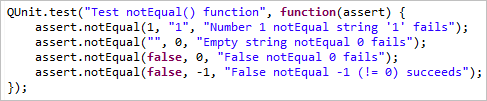
# Assert – notEqual() function

The opposite thing of equal() function.

Structure:

notEqual(actual, expected[, message])

Sample as:



# Assert – strictEqual() function

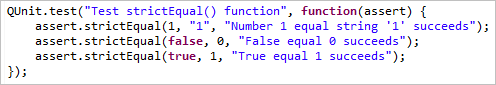
A strict type and value comparison. It uses strict equal (===) for comparing.

Structure:

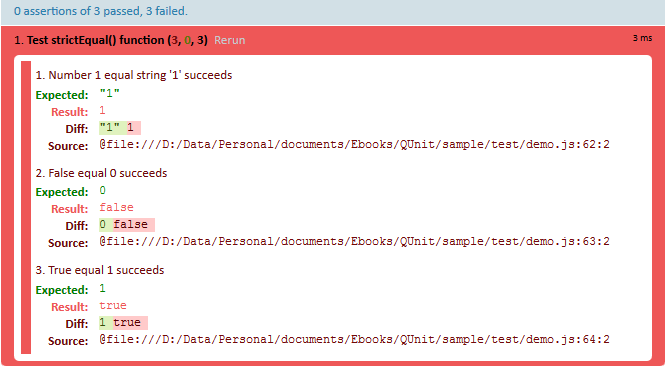
strictEqual(actual, expected[, message])

Referring to Assert – equal function to understand each parameters. They are same.

Sample as:



The result as:



# Assert – notStrictEqual() function

The opposite thing of strictEqual() function. It has same structure with strictEqual() function.

Structure:

notStrictEqual(actual, expected[, message])

Referring to Assert – equal function to understand each parameters.

# Assert – deepEqual() function

A deep recursive comparison, working on primitive types, arrays, objects, regular expressions, dates and functions. The deepEqual() assertion can be used just like equal() when comparing the value of objects, such that { key: value } is equal to { key: value }. For non-scalar values, identity will be disregarded by deepEqual. chuaxong

A deep recursive comparison, working on primitive types, arrays, objects, regular expressions, dates and functions.

expect()

Specify how many assertions are expected to run within a test.

notDeepEqual()

An inverted deep recursive comparison, working on primitive types, arrays, objects, regular expressions, dates and functions.

notPropEqual()

A strict comparison of an object’s own properties, checking for inequality.

propEqual()

A strict type and value comparison of an object’s own properties.

push()

Report the result of a custom assertion

strictEqual()

A strict type and value comparison.

throws()

Test if a callback throws an exception, and optionally compare the thrown error.