CONTINUOUS INTEGRATION API

This topic provides information about the API available to you when building Node.js applications that run TestCafe tests programmatically.

[**TestCafe Constructor**](https://testcafe.devexpress.com/Documentation/API_Reference/Continuous_Integration_API/TestCafe_Constructor/)

The TestCafe constructor is used to create a TestCafe instance, to obtain access to the Continuous Integration API.

[**TestCafe Methods**](https://testcafe.devexpress.com/Documentation/API_Reference/Continuous_Integration_API/TestCafe_Methods/)

Utilize the methods listed in this section to determine the available fixtures and workers, and use this information to execute your tests.

[**TestCafe Events**](https://testcafe.devexpress.com/Documentation/API_Reference/Continuous_Integration_API/TestCafe_Events/)

The TestCafe events allow you to respond to task completion and worker collection changes.

[**Miscellaneous**](https://testcafe.devexpress.com/Documentation/API_Reference/Continuous_Integration_API/Miscellaneous/)

This section provides various information related to Continuous Integration API.

# TESTCAFE CONSTRUCTOR

The TestCafe constructor is used to create a TestCafe instance, to obtain access to the Continuous Integration API.

#### Constructor

Creates a new TestCafe instance.

* var TestCafe = require('testcafe');
* var testCafe = new TestCafe(options);

**options** - Object. Contains startup options for TestCafe. If it is not specified, TestCafe will be unable to initialize and will terminate the process.

* var options = {
* controlPanelPort: 1337,
* servicePort1: 1338,
* servicePort2: 5556,
* hostname: '127.0.0.1',
* testsDir: 'D:\\TestCafe-14.2\\tests',
* reportsPath: 'D:\\Testing\\Reports',
* browsers: browsers
* });

**controlPanelPort** - Number [0-65535]. The port number where you can access the TestCafe Control Panel UI.

**servicePort1**, **servicePort2** - Number [0-65535]. The port numbers used by TestCafe to perform testing.

**hostname** - String (optional). The hostname of your computer. Used when you need to enable remote testing. If not specified, localhost is used.

**testDir** - String. The path to the directory where you store your tests.

**reportsPath** - String. The path to the directory where test-run reports will be saved.

Each test run creates a subfolder in this directory with its unique ID as a name - {reportsPath}\{reportUid}. This is where the report is saved in the JSON format. If screenshots are taken during the test run, they are stored in the {reportsPath}\{reportUid}\screenshots subfolder. If **reportsPath** is not specified, reports are saved to a default location that depends on the platform.

Windows: <SystemDisk>:\Users\<username>\Documents\TestCafe\reports  
OSX: /Library/Application Support/TestCafe/<username>/reports  
Linux: TestCafe/reports

Note that you can also specify the destination to save the report for an individual test run. To do this, pass the **reportPath** option to the [runTests](https://testcafe.devexpress.com/Documentation/API_Reference/Continuous_Integration_API/TestCafe_Methods" \l "runTests)method. In this case, a copy of the report will be saved to the specified directory.

**browsers** - Lists all browsers that TestCafe can use for testing.

* var browsers = {
* 'Mozilla Firefox': {
* path: 'C:\\Program Files (x86)\\Mozilla Firefox\\firefox.exe',
* cmd: '-new-window -foreground'
* },
* 'Internet Explorer': {
* path: 'C:\\Program Files\\Internet Explorer\\iexplore.exe',
* }
* }

**Keys** in the **browsers** object ('Mozilla Firefox' and 'Internet Explorer' in the example above) are unique browser names that will be used in TestCafe UI.

**path** - String. The path to the executable file that starts the browser.

**cmd** - String (optional). Additional command line parameters.

TESTCAFE METHODS

Utilize the methods listed in this section to determine the available fixtures and workers, and use this information to execute your tests.

**listAvailableBrowsers**

Returns an array of strings identifying the available browsers.

* TestCafe.listAvailableBrowsers();

**return value**: Array of String.

* ["Google Chrome", "Mozilla Firefox", "Internet Explorer"]

The strings returned by this method are the same as those specified in the [TestCafe Settings](https://testcafe.devexpress.com/Documentation/Using_TestCafe/Control_Panel/" \l "TestCafe_Settings) or in the [TestCafe Constructor](https://testcafe.devexpress.com/Documentation/API_Reference/Continuous_Integration_API/TestCafe_Constructor). You can analyze the array returned to select the browsers that will run the tests. Then transmit the filtered (or unchanged) list to the [runTests](https://testcafe.devexpress.com/Documentation/API_Reference/Continuous_Integration_API/TestCafe_Methods" \l "runTests) method to use these browsers for test execution.

**listConnectedWorkers**

Returns an array of strings identifying the connected remote workers.

* TestCafe.listConnectedWorkers();

**return value**: Array of String.

* [ "Google Chrome 64 bit", "My iPhone", "Surface" ]

To learn how to connect remote workers when running TestCafe in a Continuous Integration system, refer to the [Continuous Integration](https://testcafe.devexpress.com/Documentation/Using_TestCafe/Continuous_Integration/#Connect_Workers_-_Remote_Computers_and_Mobile_Devices) tutorial.

The strings returned by this method are the same as those specified in the URLs used to connect workers. You can analyze the array returned to select the workers that will run the tests. Then transmit the filtered or unchanged list to the [runTests](https://testcafe.devexpress.com/Documentation/API_Reference/Continuous_Integration_API/TestCafe_Methods" \l "runTests) method to use these workers for test execution.

**listDirectory**

Obtains information about fixtures at a specified path.

* TestCafe.listDirectory(path, callback(error, directoryInfo))

**path**: An empty string specifies the root test directory set in the [TestCafe Settings](https://testcafe.devexpress.com/Documentation/Using_TestCafe/Control_Panel/" \l "TestCafe_Settings) or in the [TestCafe Constructor](https://testcafe.devexpress.com/Documentation/API_Reference/Continuous_Integration_API/TestCafe_Constructor). If you need to obtain data from a directory within the root folder, specify a string with the target directory name. To access folders at deeper nesting levels, pass an array of strings corresponding to the folder structure you wish to navigate.

**error**: **null** if the information is successfully obtained. Otherwise, a string with an error code.

**directoryInfo**: An object that contains fixture and test information in the specified directories.

When you run tests using the [runTests](https://testcafe.devexpress.com/Documentation/API_Reference/Continuous_Integration_API/TestCafe_Methods" \l "runTests) method, you need to specify which fixtures or tests are to be executed. This can be done in one of the following ways.

* Use default settings so that TestCafe will run all fixtures in the root test folder and its nested directories. In this instance, it is not necessary to use the **listDirectory** function.
* Specify individual fixtures or tests to be executed. In this case, you will need to use the **listDirectory** method to obtain fixture and test information.

Below is a basic example of how to call the **listDirectory** method to obtain information about the root tests directory.

* var TestCafe = require('testcafe');
* var testCafe = new TestCafe();
* testCafe.listDirectory("", function processDirectory(error, directoryInfo) {
* console.log(directoryInfo);
* });

Sample output listing.

* { dirs:
* [ { name: 'DevExpress', path: 'DevExpress' },
* { name: 'TestCafe', path: 'TestCafe' } ],
* buildErrs: null,
* fixtures:
* [ { uid: 'e48c0dbe-c316-4405-a255-6602cc4cb29e',
* name: 'TestCafe Example Page',
* fileName: 'testcafe\_example\_page.test.js',
* page: 'http://testcafe.devexpress.com/Example',
* tests: [Object] } ]
* }

Note that the **buildErrs** field above displays a **null** value. When TestCafe retrieves fixture information, it pre-builds each JavaScript file and checks for errors. If errors are found, they will be reflected in this field. The following code line illustrates how you can output the content of this field to the Console.

The following code shows how to log the **buildErrs** field only .

* testCafe.listDirectory("", function processDirectory(error, directoryInfo) {
* console.log(directoryInfo.buildErrs);
* });

The code below illustrates sample output for a build error.

* [{
* fixtureUid: '1fd5ffa3-ae83-4854-b945-4bd8203ad075',
* fixture: {
* fileName: 'testcafe\_example\_page.test.js',
* path: [],
* name: 'testcafe\_example\_page.test.js',
* page: 'http://testcafe.devexpress.dev/',
* sharedJs: '',
* tests: []
* },
* code: 'FIXTURE\_FILE\_ERR\_FIXTURE\_IS\_UNDEFINED',
* filename: 'D:\\TestCafe-13.1\\tests\\testcafe\_example\_page.test.js'
* }]

Similarly, you can log and learn about the structure of tests in a specific fixture. This example uses JSONPath to access a fixture by its name.

* var jsonPath = require('JSONPath');
* testCafe.listDirectory('', function processDirectory(error, directoryInfo) {
* console.log(jsonPath.eval(directoryInfo, "$.fixtures[?(@.name=='TestCafe Example Page')].tests"));
* });

Sample output listing:

* [
* [{
* uid: 'ee1ee8a9-1dac-476b-8536-6f36a35cf873',
* name: 'act.type samples'
* }, {
* uid: '4a15e24b-3200-4af0-98f4-a0c8724615c8',
* name: 'act.click, array parameter, ok() assertion'
* }, {
* uid: 'a812203d-c4e4-4370-90c8-24e90d12df8d',
* name: 'Video Example'
* }]
* ]

**runTests**

Runs tests. You can use this method to run individual tests (by using their UIDs), a group of tests (fixtures or folders), as well as all the tests from the specified fixture or folder, and all tests against all fixtures and subfolders in the test folder.

* TestCafe.runTests(options[, callback(errors, taskUid, workers)])

**options**: Object. Specifies which tests to run, and which registered browsers and connected workers to use.

* {
* sourceType: 'dir' | 'fixture' | 'test' | 'group',
* source: 'TestCafe/Example' | '3818c372-684b-4f26-a12b-a6b5d7eaca3a' | ['ASPxCallback', 'ASPxGridview/Tests', 'ASPxHtmlEditor/ToolbarItemsTests/DropdownMenu'],
* browsers: ['Internet Explorer', 'Chrome'],
* workers: ['iPad'],
* quarantineMode: false | true,
* reportFormat: 'junit' | 'nunit' | 'json',
* reportPath: 'TestCafe/reports/report1.xml',
* takeScreenshotOnFails: false | true,
* failOnJsErrors: false | true
* }

**sourceType**: String. Specifies whether to run all tests in a directory, in a fixture, a group of tests or a specific test.  
By default, if the **sourceType** parameter is empty, it is identified automatically. Specify this parameter when you wish to run tests from the specific fixture or folder that has the same name (specified in the **source** parameter). In this case, with the omitted source type, tests will be run according to the following priority: folder->fixture. To run tests against all fixtures in your tests folder, omit the **sourceType** and **source** parameters.

**source**: String. If the **sourceType** parameter value is 'dir', specify the directory path relative to the tests root folder. If **sourceType** is 'fixture' or 'test', specify the relative path in the format 'DirectoryName/FixtureName/TestName' or corresponding Uid, which can be obtained using the [listDirectory](https://testcafe.devexpress.com/Documentation/API_Reference/Continuous_Integration_API/TestCafe_Methods" \l "listDirectory)method. If the **sourceType** parameter is 'group', specify an array with square brackets. The array can contain folders, fixtures, tests and their combinations. A path to a fixture or to a test should be specified in the following format: DirectoryName/FixtureName/TestName.  
Without specifying the **sourceType**, set the **source** parameter value to the name of a specific fixture or folder you wish to run tests from. If in the test folder there is a folder or a fixture with the same name, tests will be run according to the following priority: folder->fixture. To run tests against all fixtures in your tests folder, omit the **sourceType** and **source** parameters.

**browsers**: Array of String. Specifies which registered browsers to use when testing. You can initialize this field using the [listAvailableBrowsers](https://testcafe.devexpress.com/Documentation/API_Reference/Continuous_Integration_API/TestCafe_Methods" \l "listAvailableBrowsers) method.

**workers**: Array of String. Specifies which connected workers to use when testing. You can initialize this field using the [listConnectedWorkers](https://testcafe.devexpress.com/Documentation/API_Reference/Continuous_Integration_API/TestCafe_Methods" \l "listConnectedWorkers) method.

**quarantineMode**: Boolean. **true** to enable the quarantine mode for tests that fail. The quarantine mode allows you to isolate unstable tests. When a test fails, it gets into the quarantine where it will be run five more times. If the test fails in all attempts, it is marked as failed. If any of the test runs are successful, the test will be marked as unstable and will be added to a specific list of unstable tests in the test task report.

**reportFormat**: String. Specifies the format in which a test report should be saved - JUnit ('junit'), NUnit ('nunit') or JSON ('json') (by default). See [Test Run Reports](https://testcafe.devexpress.com/Documentation/API_Reference/Continuous_Integration_API/Miscellaneous#Test_Run_Reports) for examples.

**reportPath**: String. Specifies the path where the report file should be saved, and the file name. The path can be absolute or relative from the folder that contains the node.js application.

Note that there is also a *global report storage*. Its location is specified by the **reportsPath** option of the [TestCafe constructor](https://testcafe.devexpress.com/Documentation/API_Reference/Continuous_Integration_API/TestCafe_Constructor).

Regardless of the **reportPath** and **reportFormat** options passed to **runTests**, reports are additionally saved to the global storage in the JSON format. If screenshots are taken during the test run, they are saved to the global storage only.

**takeScreenshotOnFails**: Boolean. **true** to take a screenshot of the tested page whenever an assertion fails or an error occurs. Screenshots are saved to the test report.

**failOnJsErrors**: Boolean. **true** to fail a test whenever a JavaScript error occurs on a tested web page.

**Important note**

You can also handle the [taskComplete](https://testcafe.devexpress.com/Documentation/API_Reference/Continuous_Integration_API/TestCafe_Events" \l "taskComplete) and [taskUpdated](https://testcafe.devexpress.com/Documentation/API_Reference/Continuous_Integration_API/TestCafe_Events" \l "taskUpdated) events to save test execution results in the specified format.

The following is a simple example that runs a specific test on all registered browsers and connected workers, and saves execution results in the JUnit format.

* var jsonPath = require('JSONPath');
* testCafe.listDirectory('', processDirectory);
* function processDirectory(error, directoryInfo) {
* var jsonPathQuery = "$.fixtures[?(@.name=='TestCafe Example Page')].tests[?(@.name=='act.type samples')].uid";
* var testUid = jsonPath.eval(directoryInfo, jsonPathQuery);
* runTest(testUid);
* }
* function runTest(testUid) {
* var runOptions = {
* sourceType: 'test',
* source: testUid,
* workers: testCafe.listConnectedWorkers(),
* browsers: testCafe.listAvailableBrowsers(),
* quarantineMode: true,
* reportFormat: 'junit',
* reportPath: '/Report.xml'
* };
* testCafe.runTests(runOptions);
* }

Using the optional callback parameter, you can check whether or not the task has actually started, which task identifiers were used, and which workers were able to run. If a task fails to run, you can also determine the error type, log an error message and close the process with an error code. Consider the following example, which tries to run a fixture that does not exist.

* var runOptions = {
* sourceType: 'fixture',
* source: 'invalid\_value',
* browsers: testCafe.listAvailableBrowsers()
* };
* testCafe.runTests(runOptions, function (errors, taskUid, workers) {
* if (errors != null) {
* console.log(errors);
* process.exit(1);
* }
* });

You will get the following output.

The "invalid\_value" test target is empty or not found.

The following is comprehensive information on callback function parameters.

**errors**: Array of String. Lists errors that occurred when trying to launch the task. **null** if the task has been started successfully.  
**taskUid**: String. Identifies the task being executed. The same identifiers are used in the [taskComplete](https://testcafe.devexpress.com/Documentation/API_Reference/Continuous_Integration_API/TestCafe_Events" \l "taskCompleted) and [taskUpdated](https://testcafe.devexpress.com/Documentation/API_Reference/Continuous_Integration_API/TestCafe_Events" \l "taskUpdated) events. **null** if the task has not started.  
**workers**: Array of String. Identifies workers and browsers in which the tests are being executed. **null** if the task has not started.

# TESTCAFE EVENTS

The TestCafe events allow you to respond to task completion and worker collection changes.

#### taskComplete

Occurs when a testing task is completed.

Test tasks are initiated by calling the [runTests](https://testcafe.devexpress.com/Documentation/API_Reference/Continuous_Integration_API/TestCafe_Methods" \l "runTests) method. A task can include an individual test, a fixture, or all fixtures within the root tests directory. Once all tests in a task are complete, the TestCafe raises the **taskComplete** event, which lists detailed information about passed and failed tests (if any). Handle this event to log results, and return a success or error code to the system, depending on the results of the test execution.

* testCafe.on('taskComplete', function (report));

The following is a simple example that handles the **taskComplete** event to check if any of the tests have failed. If so, it will log the report and return an error to the user.

* testCafe.on('taskComplete', complete);
* testCafe.runTests({
* browsers: testCafe.listAvailableBrowsers(),
* emulateCursor: false,
* quarantineMode: true
* });
* function complete(report) {
* if (report.failed == 0) {
* console.log("All tests passed successfully!");
* process.exit(0);
* } else {
* console.log(JSON.stringify(report, null, 4));
* process.exit(1);
* }
* }

In the code above, **report** is an object that contains detailed information about the results of test task execution. To learn about its structure, see [Test Run Reports](https://testcafe.devexpress.com/Documentation/API_Reference/Continuous_Integration_API/Miscellaneous#Test_Run_Reports).

#### taskUpdated

Occurs each time an individual test has been executed.

Test tasks are initiated by calling the [runTests](https://testcafe.devexpress.com/Documentation/API_Reference/Continuous_Integration_API/TestCafe_Methods" \l "runTests) method. A task can include an individual test, a fixture, or all fixtures within the root tests directory. The **taskUpdated** event fires when the task enters the Pending or Running state, and after each individual test has been executed. After all tests have been executed and the task changes its state to Completed, TestCafe raises the [taskComplete](https://testcafe.devexpress.com/Documentation/API_Reference/Continuous_Integration_API/TestCafe_Events" \l "taskComplete) event.

* testCafe.on('taskUpdated', function (report));

The following example handles the **taskUpdated** event to display condensed information about each task execution stage.

* testCafe.on('taskUpdated', updated);
* testCafe.runTests({
* browsers: testCafe.listAvailableBrowsers(),
* emulateCursor: false,
* quarantineMode: true
* });
* function updated(report) {
* console.log('Task: ' + report.name + ', Status:' + report.status);
* var processedTests = report.failed + report.passed;
* console.log('Processed ' + processedTests + ' of ' + report.testCount + ', failed: ' + report.failed);
* console.log('======================');
* }

In the code above, **report** is an object that contains detailed information about the results of test task execution. To learn about its structure, see [Test Run Reports](https://testcafe.devexpress.com/Documentation/API_Reference/Continuous_Integration_API/Miscellaneous#Test_Run_Reports).

#### workerAdded

Occurs immediately after a remote worker has been connected.

* testCafe.on('workerAdded', function (workerName));

**workerName** - String. The worker name that was specified when the worker connected to TestCafe.

Note that browsers registered in the TestCafe configuration file or passed to the constructor in code are also registered as workers, and thus raise this event as well. The following example shows how to differentiate between the predefined and remote workers in order to run tests in remote workers only.

* testCafe.on('workerAdded', function (worker) {
* if (testCafe.listAvailableBrowsers().indexOf(worker) > -1)
* return;
* var options = {
* workers: [worker],
* emulateCursor: false;
* }
* testCafe.runTests(options, function () {
* testCafe.on('taskComplete', function (report) {
* console.log(report);
* });
* });
* });

#### workerDisconnected

Occurs immediately after a remote worker has been disconnected.

* testCafe.on('workerDisconnected', function (workerName));

**workerName** - String. The worker name.

Handle this event to perform any cleanup actions required after handling the [workerAdded](https://testcafe.devexpress.com/Documentation/API_Reference/Continuous_Integration_API/TestCafe_Events" \l "workerAdded) event.

Note that this event does not fire when a browser registered in the TestCafe configuration file or passed to the constructor in code is closed (unlike the[workerAdded](https://testcafe.devexpress.com/Documentation/API_Reference/Continuous_Integration_API/TestCafe_Events#workerAdded) event that is raised when such a browser is launched).

# MISCELLANEOUS

This section provides various information related to Continuous Integration API.

#### Test Run Reports

Test run reports returned by [taskComplete](https://testcafe.devexpress.com/Documentation/API_Reference/Continuous_Integration_API/TestCafe_Events" \l "taskComplete) and [taskUpdated](https://testcafe.devexpress.com/Documentation/API_Reference/Continuous_Integration_API/TestCafe_Events" \l "taskUpdated) events contain detailed information about the results of test task execution.

By default, reports are presented in the JSON format. You can change this to the JUnit or NUnit format. To specify the report format, pass an appropriate value to the [runTests](https://testcafe.devexpress.com/Documentation/API_Reference/Continuous_Integration_API/TestCafe_Methods" \l "runTests) method in the **reportFormat** parameter.

#### JSON Reports

A JSON report consist of the following fields.

**uid** - the unique ID of the executed test.  
**name** - the name of the object that owns the executed test(s) (directory name, fixture name and single test name).  
**status** - information that indicates the state of the testing operation (pending, running, failed and succeeded).  
**startedAt** - the date and time when testing started.  
**completedAt** - the date and time when testing completed.  
**time** - specifies the time, in seconds, of tests execution.  
**workerNames** - identifies the browsers for test execution.  
**browserVersions** - specifies the versions of the browsers used for test execution.  
**testCount** - the total number of executed tests.  
**failed** - the number of failed tests.  
**passed** - the number of passed tests.  
**testErrReports** - information about the failed tests.  
**passedTests** - information about the passed tests.  
**unstableTests** - information about unstable tests.

The code below illustrates the result of the test execution that the **report** object can contain.

* {
* uid: '842a36e2-a12c-48d8-a946-f878896ad53c',
* name: '"Root" directory tests',
* status: 'succeeded',
* startedAt: ['07/29/2013', '12:36:12', 'PM'],
* completedAt: ['07/29/2013', '12:37:15', 'PM'],
* workerNames: ['Google Chrome', 'Internet Explorer', 'Mozilla Firefox'],
* browserVersions: { "Google Chrome": "41.0.2272", "Internet Explorer": "11.0.9600.17690", "Mozilla Firefox": "36.0.4"},
* testCount: 10,
* failed: 0,
* passed: 10,
* testErrReports: {},
* passedTests: {},
* unstableTests: {}
* }

As you can see, the example above suggests that there were no test execution errors. In this instance, the **testErrReports** field is empty. The code below lists the information you can obtain from the **testErrReports** field when tests fail.

* "testErrReports": {
* 'f60f7573-fa42-4f2e-b760-33883eff9a05': {
* name: 'Click a label that does not exist',
* fixtureName: 'My Page',
* fixturePath: '',
* time: 10,
* errs: [{
* msg: 'Error on step "Click label": "click" action target does not contain DOM-elements.',
* workerName: 'Internet Explorer',
* browserVersion: 'Internet Explorer': '11.0.9600.17690'
* }]
* }
* }

**passedTests** field provides the details about tests that passed successfully.

* "passedTests": {
* 'd72022c7-a694-4db5-8fef-96a2cc9be988': {
* fixtureName: 'Data Binding',
* fixturePath: '',
* name: 'Filter',
* time: 7
* }
* }

The following example illustrates the information contained in the **unstableTests** field if there are tests that have been marked as unstable in the quarantine.

* "unstableTests": {
* 'fd93159b-db41-48e3-9df3-11504fa765de': {
* name: 'Click a label',
* fixtureName: 'My Fixture',
* fixturePath: '',
* workerNames: [
* 'Mozilla Firefox'
* ]
* browserVersions:{
* "Mozilla Firefox": "36.0.4"
* }
* }
* }

#### JUnit Reports

When the report format is set to JUnit, the report is returned as a string value containing data in the standard JUnit format.

* <?xml version="1.0" encoding="UTF-8" ?>
* <testsuites>
* <testsuite name="&quot;My Fixture&quot; fixture tests" errors="1" failures="1" tests="3" time="23">
* <testcase name="'/My Fixture' - Should show header" time="2">
* <error>
* Worker: Google Chrome (version 41.0.2272)
* Message: Assertion failed at step &quot;1.Assert&quot;: eq($(&quot;.article-header&quot;).is(&quot;:visible&quot;), false);
* Expected: false
* Actual: true

* </error>
* </testcase>
* <testcase name="'/My Fixture' - Should prevent unauthorized access" time="15" />
* <testcase name="'/My Fixture' - Should validate entered data" time="6" />
* </testsuite>
* </testsuites>

#### NUnit Reports

When report format is set to NUnit, the report is returned as a string value containing data in the standard NUnit format.

* <?xml version="1.0" encoding="UTF-8" ?>
* <test-results name="&quot;My Fixture&quot; fixture tests" total="3" error="1" failures="1" not-run="0" time="23">
* <test-suite name="&quot;My Fixture&quot; fixture tests" success="False" executed="True" time="23" >
* <results>
* <test-case name="'/My Fixture' - Should show header" success="False" executed="True" time="2">
* <failure>
* <message>
* Worker: Google Chrome (version 41.0.2272)
* Message: Assertion failed at step &quot;1.Assert&quot;: eq($(&quot;.article-header&quot;).is(&quot;:visible&quot;), false);
* Expected: false
* Actual: true

* </message>
* </failure>
* </test-case>
* <test-case name="'/My Fixture' - Should prevent unauthorized access" success="True" executed="True" time="15" />
* <test-case name="'/My Fixture' - Should validate entered data" success="True" executed="True" time="6" />
* </results>
* </test-suite>
* </test-results>