* [Step 4 - Authentication](https://testcafe.devexpress.com/Documentation/Getting_Started/Test_Code_Guide/" \l "Step_4_-_Authentication)
* [Step 5 - Test Cases](https://testcafe.devexpress.com/Documentation/Getting_Started/Test_Code_Guide/#Step_5_-_Test_Cases)
* [Step 6 - Mixins](https://testcafe.devexpress.com/Documentation/Getting_Started/Test_Code_Guide/#Step_6_-_Mixins)

TEST CODE GUIDE

This guide will teach you how to compose tests by merely writing JavaScript code. You will learn the basics of TestCafe API, advancing step by step, from the simple to more elaborate testing techniques.

All tests in this guide target a sample page available at [http://testcafe.devexpress.com/Example](https://testcafe.devexpress.com/Example). You can try running each of them against this page by yourself.

**General Code Structure**

TestCafe organizes tests into fixtures. Test fixtures are JavaScript files with *\*.test.js* extensions that contain one or more tests for a single target URL.

The following snippet illustrates basic elements of a TestCafe fixture.

* "@fixture Fixture Name";
* "@page ./target/page/URL";
* "@test"["Test 1 Name"] = {
* "Step 1 Name" : function() {
* // Step 1 code
* },
* "Step 2 Name" : function() {
* // Step 2 code
* },
* //…
* };
* "@test"["Test 2 Name"] = {
* //…
* };

The fixture has a name marked by the @fixture keyword. The target webpage is specified using the @page keyword.

Individual tests are represented by records in a hashtable with their names as keys. Tests are comprised of steps – functions that contain code of the respective test step.

**Step 1 - Performing Actions on a Page**

Every test should be capable of interacting with page content. Here is a 'Hello, world' test that types a developer name into a text editor and then clicks the Submit button.

* "@fixture Getting Started - Step 1";
* "@page http://testcafe.devexpress.com/Example";
* "@test"["Hello world"] = {
* "1.Type the name": function() {
* act. type("#Developer\_Name", "John Smith");
* },
* "2.Click the Submit button": function() {
* act.click("#submit-button");
* }
* };

All [user actions](https://testcafe.devexpress.com/Documentation/API_Reference/Test_Fixture_API/User_Actions/) are methods exposed by the act object. These include click, hover, type, file upload and many more.

Target webpage elements can be identified by objects of [different types](https://testcafe.devexpress.com/Documentation/API_Reference/Test_Fixture_API/Miscellaneous/#User_action_target).

Note that this test is divided into two steps. That's because one step can contain only one user action (with no limitations on custom code).

View Test Run



**Step 2 - Assertions**

A functional test should also check the result of actions performed. For this purpose, you have to add an assertion.

Clicking the Submit button opens a “Thank you” page whose header addresses a user by the entered name. The sample test will check if the name is correct.

* @fixture Getting Started - Step 2";
* "@page http://testcafe.devexpress.com/Example";
* "@test"["Hello world"] = {
* "1.Type the name": function() {
* act.type("#Developer\_Name", "John Smith");
* },
* "2.Click the Submit button": function() {
* act.click("#submit-button");
* },
* "3.Assert the name": function() {
* eq($(".article-header").text(), "Thank You, John Smith!");
* }
* };

The eq assertion method checks objects for equality. There are a number of other assertions in TestCafe API. For a complete list, see [Assertions](https://testcafe.devexpress.com/Documentation/API_Reference/Test_Fixture_API/Assertions/).

View Test Run



**Step 3 - Dialog Handling**

Note the Populate button on the sample page. This button fills the Developer Name text editor with a predefined name. Before it does so, a confirmation dialog is invoked. You must answer OK to proceed. The following test demonstrates how you can do this.

* "@fixture Getting Started - Step 3";
* "@page http://testcafe.devexpress.com/Example";
* "@test"["Hello world"] = {
* "1.Populate the name": function() {
* handleConfirm(true);
* act.click(":containsExcludeChildren(Populate)");
* },
* "2.Click the Submit button": function() {
* act.click("#submit-button");
* },
* "3.Assert the name": function() {
* eq($(".article-header").text(), "Thank You, Peter Parker!");
* }
* };

TestCafe API provides a number of functions that can handle various native dialogs such as alerts, confirmations and prompts. These functions must be called before an action that invokes the dialog.

You can find the full list of dialog handling functions at [Native Dialog Handling](https://testcafe.devexpress.com/Documentation/API_Reference/Test_Fixture_API/Native_Dialog_Handling/).

View Test Run



**Step 4 - Authentication**

At this step, you will see how TestCafe handles webpage authentication.

The subsequent test clicks the Authentication button that performs Http Basic authentication, opens a Personal Cabinet and checks whether the user name is correct.

* "@fixture Fixture Name";
* "@fixture Getting Started - Step 4";
* "@page http://testcafe.devexpress.com/Example";
* "@auth test:1234";
* "@test"["Hello world"] = {
* "1.Populate the name": function() {
* handleConfirm(true);
* act.click(":containsExcludeChildren(Populate)");
* },
* "2.Click Authentication": function() {
* act.click(":containsExcludeChildren(Authentication)");
* },
* "3.Assert the name": function() {
* eq($(".user-name").text(), "Peter Parker");
* }
* };

To pass the authentication, the fixture uses the @auth keyword to provide login and password, separated by semicolon.

See [Http Authentication](https://testcafe.devexpress.com/Documentation/Api_Reference/Test_Fixture_API/Common_Concepts/#Http_Authentication) to learn more about authentication.

View Test Run



**Step 5 - Test Cases**

The next step is to check if the occupation in the Personal Cabinet is displayed correctly.

Assume you have three persons: a developer, a QA engineer and a product manager. The first thing that may come to mind is to write three separate tests as shown below.

* "@fixture Getting Started - Step 5";
* "@page http://testcafe.devexpress.com/Example";
* "@auth test:1234";
* "@test"["Peter"] = {
* '1.Select in input "Your name:"': function() {
* act.select("#Developer\_Name");
* },
* "2.Type the name": function() {
* act.type("#Developer\_Name", "Peter Parker", {
* caretPos: 0
* });
* },
* "3.Click Authentication": function() {
* act.click(":containsExcludeChildren(Authentication)");
* },
* "4.Assert the user name and occupation": function() {
* eq($(".user-name").text(), "Peter Parker");
* eq($(".occupation").text(), "Developer");
* },
* '5.Click link "Back"': function() {
* act.click(":containsExcludeChildren(Back)");
* }
* };
* "@test"["John"] = {
* '1.Select in input "Your name:"': function() {
* act.select("#Developer\_Name");
* },
* "2.Type the name": function() {
* act.type("#Developer\_Name", "John Smith", {
* caretPos: 0
* });
* },
* "3.Click Authentication": function() {
* act.click(":containsExcludeChildren(Authentication)");
* },
* "4.Assert the name and occupation": function() {
* eq($(".user-name").text(), "John Smith");
* eq($(".occupation").text(), "Product Manager");
* },
* '5.Click link "Back"': function() {
* act.click(":containsExcludeChildren(Back)");
* }
* };
* "@test"["Bob"] = {
* '1.Select in input "Your name:"': function() {
* act.select("#Developer\_Name");
* },
* "2.Type the name": function() {
* act.type("#Developer\_Name", "Bob Marley", {
* caretPos: 0
* });
* },
* "3.Click Authentication": function() {
* act.click(":containsExcludeChildren(Authentication)");
* },
* "4.Assert the name and occupation": function() {
* eq($(".user-name").text(), "Bob Marley");
* eq($(".occupation").text(), "QA Engineer");
* },
* '5.Click link "Back"': function() {
* act.click(":containsExcludeChildren(Back)");
* }
* };

You can see that code is repeated almost identically three times. To avoid this, use *test cases*.

A test case is a set of parameters passed to a test through **this** object. When test cases are defined, you can run the test multiple times with different parameter sets. Such a test is called *parameterized*.

The following code demonstrates how you can rewrite three tests into one parameterized test.

* "@fixture Getting Started";
* "@page http://testcafe.devexpress.com/Example";
* "@auth test:1234";
* "@test"["Test Cases"] = {
* "@testCases": [
* {"@name": "Peter", userName: "Peter Parker", occupation: "Developer"},
* {"@name": "John", userName: "John Smith", occupation: "Product Manager"},
* {"@name": "Bob", userName: "Bob Marley", occupation: "QA Engineer"}
* ],
* '1.Select in input "Your name:"': function() {
* act.select("#Developer\_Name");
* },
* "2.Type the name:": function() {
* act.type("#Developer\_Name", this.userName, {
* caretPos: 0
* });
* },
* "3.Click Authentication": function() {
* act.click(":containsExcludeChildren(Authentication)");
* },
* "4.Assert the name and occupation": function() {
* eq($(".user-name").text(), this.userName);
* eq($(".occupation").text(), this.occupation);
* },
* '5.Click link "Back"': function() {
* act.click(":containsExcludeChildren(Back)");
* }
* };

Each test case must have a name specified by using the @namekeyword followed by any number of parameters.

Test cases can also be declared in a separate JSON file, which allows you to share them between tests. To learn more, see [Test Parameterization](https://testcafe.devexpress.com/Documentation/Api_Reference/Test_Fixture_API/Common_Concepts/#Test_Parameterization).

View Test Run



**Step 6 - Mixins**

Now assume that you need another test that covers the capability to request vacations which is available in the Personal Cabinet.

* "@fixture Getting Started - Step 6";
* "@page http://testcafe.devexpress.com/Example";
* "@auth test:1234";
* "@test"["Test Cases"] = {
* "@testCases": [
* {"@name": "Peter", userName: "Peter Parker", occupation: "Developer"},
* {"@name": "John", userName: "John Smith", occupation: "Product Manager"},
* {"@name": "Bob", userName: "Bob Marley", occupation: "QA Engineer"}
* ],
* '1.Select in input "Your name:"': function() {
* act.select("#Developer\_Name");
* },
* "2.Type the name:": function() {
* act.type("#Developer\_Name", this.userName, {
* caretPos: 0
* });
* },
* "3.Click Authentication": function() {
* act.click(":containsExcludeChildren(Authentication)");
* },
* "4.Assert the name and occupation": function() {
* eq($(".user-name").text(), this.userName);
* eq($(".occupation").text(), this.occupation);
* },
* '5.Click link "Back"': function() {
* act.click(":containsExcludeChildren(Back)");
* }
* };
* "@test"["Vacation"] = {
* '1.Select in input "Your name:"': function() {
* act.select("#Developer\_Name");
* },
* "2.Type the name": function() {
* act.type("#Developer\_Name", "Peter Parker", {
* caretPos: 0
* });
* },
* "3.Click Authentication": function() {
* act.click(":containsExcludeChildren(Authentication)");
* },
* "3.Type the vacation duration": function() {
* act.type(".days-taken-spinedit[name='daysTaken']", "5");
* },
* "5.Click Request": function() {
* act.click(":containsExcludeChildren(Request)");
* },
* "6.Check days taken": function() {
* eq($(".days-taken").text(), "9");
* }
* };

This new test requests a vacation.

These tests share the first three steps: selecting the content of the "Your name" box, entering the developer name and clicking the Authentication button.

TestCafé provides the capability to create **mixins** – pieces of code that can be reused in multiple tests.

* "@fixture Getting Started";
* "@page http://testcafe.devexpress.com/Example";
* "@auth test:1234";
* "@mixin" ["Authentication"] = {
* '1.Select in input "Your name:"': function() {
* act.select("#Developer\_Name");
* },
* "2.Type the name": function() {
* act.type("#Developer\_Name", this.userName);
* },
* "3.Click Authentication": function() {
* act.click(":containsExcludeChildren(Authentication)");
* }
* };
* "@test"["Test Cases"] = {
* "@testCases": [
* {"@name": "Peter", userName: "Peter Parker", occupation: "Developer"},
* {"@name": "John", userName: "John Smith", occupation: "Product Manager"},
* {"@name": "Bob", userName: "Bob Marley", occupation: "QA Engineer"}
* ],
* "1.Enter the cabinet": "@mixin Authentication",
* "2.Assert the user name and occupation": function() {
* eq($(".user-name").text(), this.userName);
* eq($(".occupation").text(), this.occupation);
* }
* };
* "@test"["Vacation"] = {
* "1.Save the name": function() {
* this.userName = "Peter Parker";
* },
* "2.Enter the cabinet": "@mixin Authentication",
* "3.Type the vacation duration": function() {
* act.type(".days-taken-spinedit[name='daysTaken']", "5");
* },
* "4.Click Request": function() {
* act.click(":containsExcludeChildren(Request)");
* },
* "5.Check days taken": function() {
* eq($(".days-taken").text(), "9");
* }
* };

Mixins are marked with the @mixinkeyword and referenced to by the same keyword when used in tests.

Note that the new test now contains one more step – saving the person name. That’s because the first test is parameterized, and hence the mixin shared by both tests must obtain the developer name from **this** object.

For more information, see [Mixins](https://testcafe.devexpress.com/Documentation/Api_Reference/Test_Fixture_API/Common_Concepts/" \l "Mixins).

View Test Run