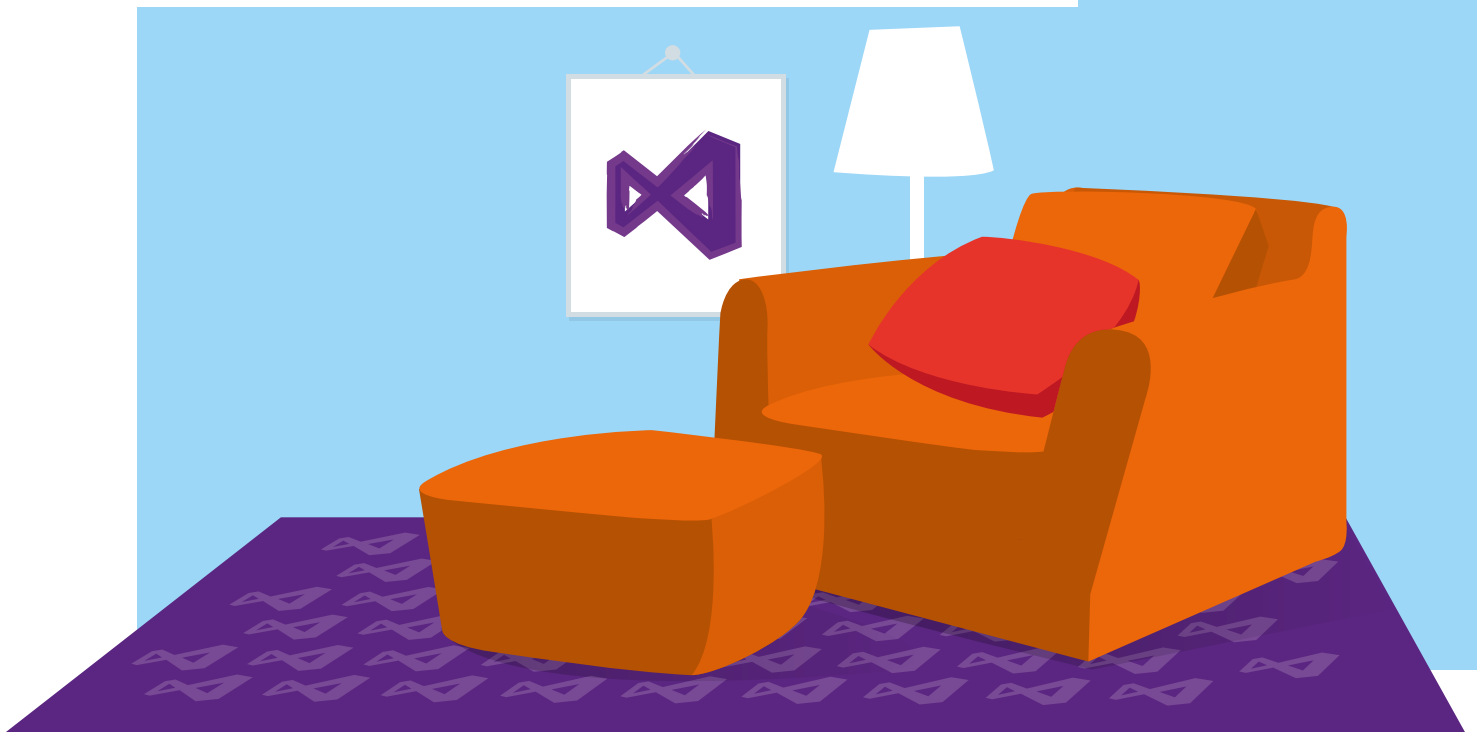



INTEGRATED TESTING ENVIRONMENT

for Developers and Coding Testers



Telerik

TEST STUDIO



Test Studio's plugin for Visual Studio enables developers and testers comfortable writing code to work in the environment where they're most productive.

Write Code Where Needed

Every test automation project will require some level of coding to be successful. Test Studio's record and playback creates powerful, maintainable tests, but you'll still need to write code at some point to cover common, critical aspects such as configuration, backing APIs, or test oracles.

Setup and Teardown/ Configuration

Complex tests require clear, flexible configuration actions that keep the overall test suite maintainable over the long run. Pushing setup, teardown, and configuration to code versus the system's interface dramatically speeds up test execution by leveraging the system's own internal functionality through internal APIs, web service endpoints, or database stored procedures.

Let's have a look at some common scenarios where a team might drop to code to handle specific situations.

Setup for CRUD Tests

It's critical to keep test cases, manual or automatic, focused and granular. If we're working with tests focusing on Create, Retrieve, Update, or Delete (CRUD) actions, then automation at the UI level should be focused on that specific scenario, not getting the system ready to test.

Let's use a test which focuses on checking whether or not we can successfully create a forum post as a new user. The test is around creating the forum post, but we have prerequisites for a new user and a forum to create the post in. We could use UI automation to log on as an administrator and create both the new user and the forum, but that could easily take 30-45 seconds to work through. Moreover, we have to acknowledge that much of UI automation is brittle by its very nature.

Instead of using the UI to create the user and forum for the post, we should prefer to use existing API functionality within the system to manage that effort for us. (We need

to ensure those APIs are properly tested elsewhere in the system, of course!) Using these APIs ensures our tests run faster, and we're also keeping the overall test suite much more maintainable.

Configuration Actions

Part of keeping your test suite lean and focused on high-value tests is ensuring you're not testing components which don't make sense to test. Using coded steps to disable and re-enable these components during automated testing runs is a great way to keep your tests smoother and targeted to functionality your teams are writing.

A perfect example of this is the infamous CAPTCHA anti-spambot guards. We at Telerik often get questions on how to test CAPTCHA, and our constant response is "Don't." CAPTCHA is a tool provided by a third party, and generally speaking you shouldn't spend your time testing someone else's software.

Another example of a third party tool that doesn't make sense to automate is TinyMCE, a popular open source rich text editor. TinyMCE has its own test suite validating its basic functionality. Why would you want to write tests to check this within your own system?

In both these cases it makes perfect sense to create configuration switches in your system to turn off this functionality and bypass it for your test automation. In CAPTCHA's case you would work with your system developers to create configuration options to remove CAPTCHA from workflows in your system. You can use coded steps to alter these configuration options using your own methods, or leveraging the various APIs under the System.Configuration.ConfigurationManager class.

Backing APIs

As powerful as Test Studio is, every project will end up needing some form of coded infrastructure at some point. Backing APIs make it much easier to handle the setup and configuration tasks mentioned above. You could work with Test Studio Standalone and use an empty coded step in an empty test as a holding place for custom utility classes and namespaces; however, that approach is only suitable for prototypes and tiny projects.

Working with Test Studio inside Visual Studio makes it much easier to split out backing APIs/infrastructure to separate projects where they're much more easily managed. This ensures teams are able to keep areas of responsibility properly separated out from the tests themselves.

Extend tests

You'll also find yourself wanting to drop to code to extend your tests for various reasons. Testing particular conditions on controls such as Trees or Grids is a fine example of this situation.

Let's use this grid as an example for two situations: verifying the count in the grid, and verifying the count of rows meeting specific criteria.

Region	Company	LastName	FirstName	Id	
Europe	Top Notch Music Academy	Beethoven	Ludwig	9	Edit
New Earth	Blue Sun	Cobb	Jayne	12	Edit
Eastern	Relativity Inc	Einstein	Albert	8	Edit
Midwest	Telerik	Holmes	Jim	1	Edit
Scotland	Bravely Bravely, LLC	Knight	Robin	4	Edit
Scotland	Round Table Hotels	Leodegrance	Guinevere	5	Edit
Midwest	Tip Top Software	McGillicuddy	Katy	3	Edit
Western	Merwin Consulting Ltd	Merwin	Sarah	6	Edit
New Earth	Serenity, Inc.	Reynolds	Malcom	7	Edit

First, we'll count up the total rows in the grid's body. There are several ways to do this; here's one example.

```
IList<HtmlTableRow> rows = Find.AllByXPath<HtmlTableRow>("//tbody/tr");  
Assert.AreEqual(9, rows.Count);
```

In this example we're using Xpath to grab all table **row** elements under the table body element. (`//tbody/tr`) That's stored in the **rows** collection, and we then simply use the Assert class to validate we've got the correct count.

Another test might be to validate a table generated by a report or other query. Using the table above, we could say we're validating two rows with users from the New Earth region will be displayed when we run the query against the baseline dataset.

The following example shows a different way of interacting with the Grid on the page. In this case, we've stored the grid in the Element Repository and are able to extract it as a strongly typed object. We're then looking for elements containing the text we're looking for. This will effectively pull us off two table cell elements — this entire test scenario is predicated on us making these decisions based on our knowledge of the baseline dataset, and the absolute trust that we don't have to worry about pulling off some other element(s) in the table. →

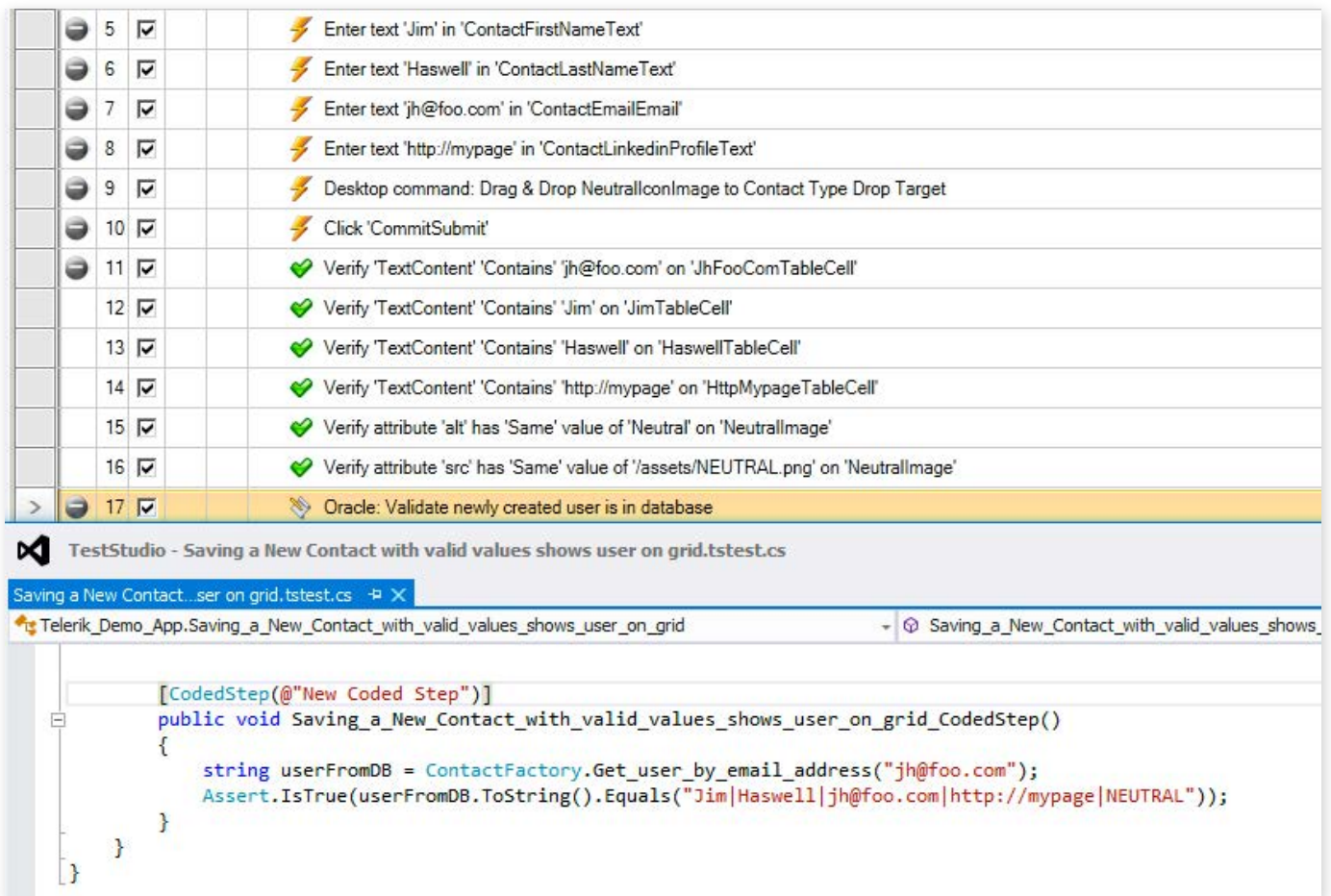
```
RadGrid grid = Pages.HomePage.Content _ Grid;  
IList<Element> newEarthContacts =  
    grid.Find.AllByContent("New Earth");  
Assert.AreEqual(2, newEarthContacts.Count);
```

Oracles

Test oracles are another critical area where you'll find yourself needing to write code. Oracles (sometimes also referred to as "heuristics") are the final check in your tests. These are the true validations of whether your system's actually functioning as you expect it to—and it's generally not just a simple validation on the UI.

An oft-used example is that of creating a new item in your system's UI. Your test may log you on, navigate through a few steps and submit something, then finally give you a visual confirmation your item has been created. If your test finishes off with a validation of the UI, then the test is only partially complete because you can't be sure the item was truly created in your database or underlying persistence system. An oracle should be used to connect to that persistence layer and validate the data item to ensure there's not been some issue with unhandled exceptions, caching, or other problem which might have the UI updating while not saving things to the database.

Below is a practical example of this concept.



5	✓	⚡ Enter text 'Jim' in 'ContactFirstNameText'
6	✓	⚡ Enter text 'Haswell' in 'ContactLastNameText'
7	✓	⚡ Enter text 'jh@foo.com' in 'ContactEmailEmail'
8	✓	⚡ Enter text 'http://mypage' in 'ContactLinkedInProfileText'
9	✓	⚡ Desktop command: Drag & Drop NeutralIconImage to Contact Type Drop Target
10	✓	⚡ Click 'CommitSubmit'
11	✓	✓ Verify 'TextContent' 'Contains' 'jh@foo.com' on 'JhFooComTableCell'
12	✓	✓ Verify 'TextContent' 'Contains' 'Jim' on 'JimTableCell'
13	✓	✓ Verify 'TextContent' 'Contains' 'Haswell' on 'HaswellTableCell'
14	✓	✓ Verify 'TextContent' 'Contains' 'http://mypage' on 'HttpMypageTableCell'
15	✓	✓ Verify attribute 'alt' has 'Same' value of 'Neutral' on 'NeutralImage'
16	✓	✓ Verify attribute 'src' has 'Same' value of '/assets/NEUTRAL.png' on 'NeutralImage'
17	✓	🔗 Oracle: Validate newly created user is in database

TestStudio - Saving a New Contact with valid values shows user on grid.tstest.cs

Saving a New Contact...ser on grid.tstest.cs

Telerik_Demo_App.Saving_a_New_Contact_with_valid_values_shows_user_on_grid

```
[CodedStep(@"New Coded Step")]
public void Saving_a_New_Contact_with_valid_values_shows_user_on_grid_CodedStep()
{
    string userFromDB = ContactFactory.Get_user_by_email_address("jh@foo.com");
    Assert.IsTrue(userFromDB.ToString().Equals("Jim|Haswell|jh@foo.com|http://mypage|NEUTRAL"));
}
}
```

Here we're doing a two step validation: first in steps 11-16 we're confirming the UI accurately reflects the edits we've made. In step 17 we have a coded step which confirms the user was created in the database—the code for that is shown in the window below and also reflects our use of a backing API to handle easy communication to the database via the ContactFactory helper class.

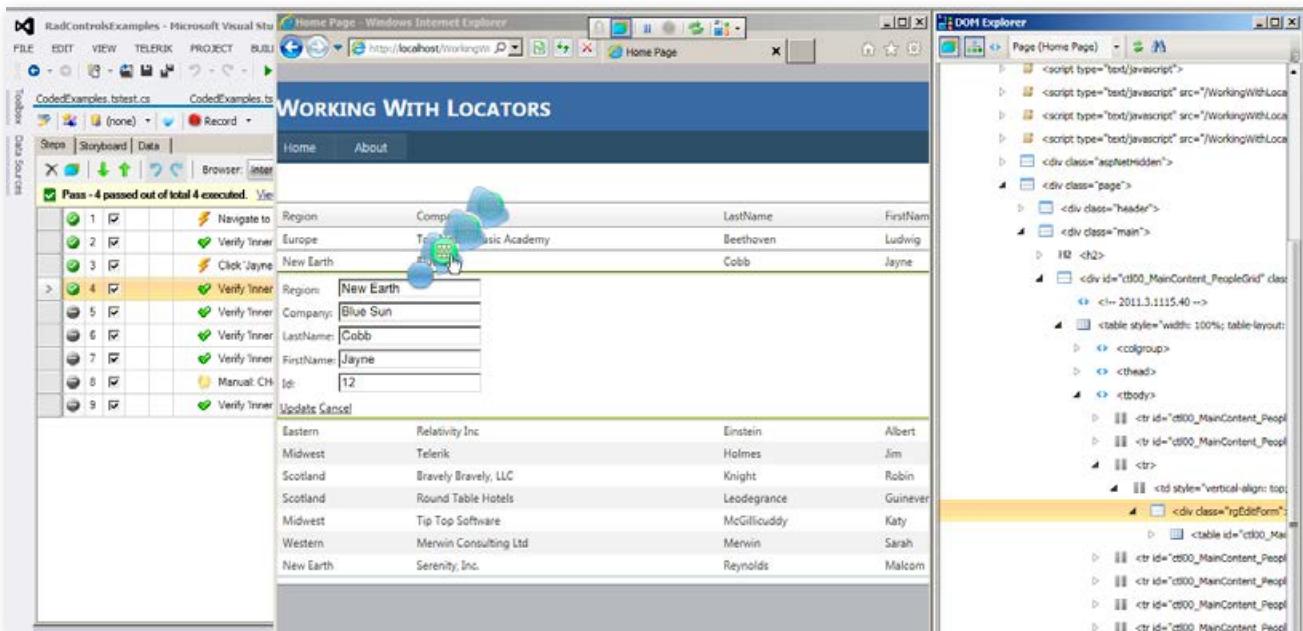
Functional Testing Features

Developers working on functional testing don't have to be focused solely on coding tasks. They should also be able to work with any of the tasks around creating, editing, and maintaining automated UI testing. Test Studio's plugin gives developers (and coding testers!) the ability to work seamlessly with functional tests directly within Visual Studio.

Working inside Visual Studio leaves team members with complete access to the critical features that make Test Studio Standalone so useful:

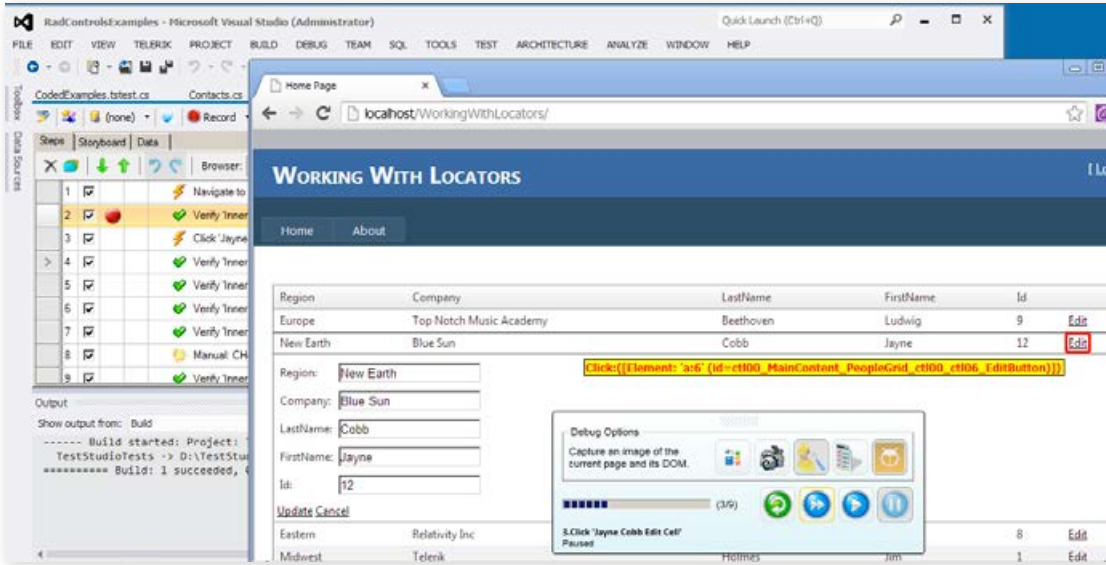
Recorder

Record and update tests. Use [the DOM explorer](#) to craft custom find logic to ensure your tests make the best use of locators specific to your particular application and UI.



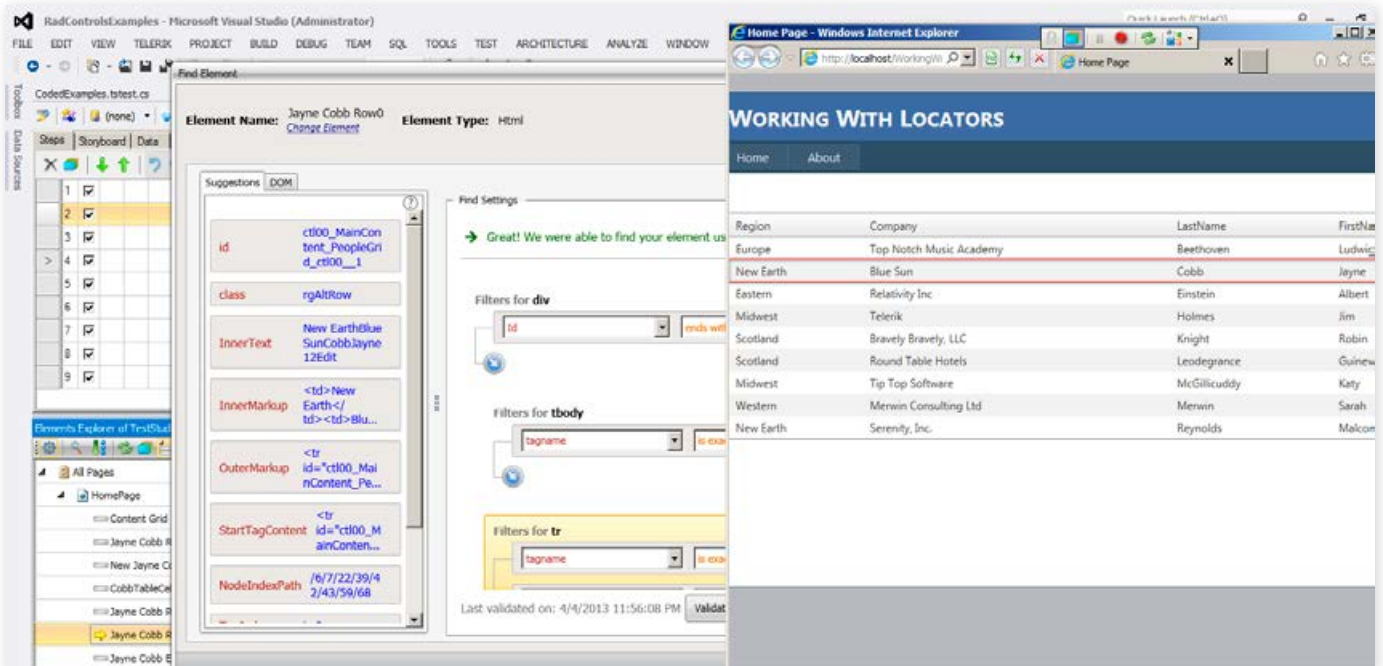
Debugger

Debug both code via Visual Studio and [recorded steps via Test Studio's Visual Debugger](#) straight from within your Visual Studio environment.



Element repository

The [element repository](#) is one of the most powerful features of Test Studio. Within Visual Studio you've got full access to this centralized storage for all your element find logic/locators. Elements are easily updated with the same Edit Find Logic dialogs you get within the Standalone version.



Cross browser compatibility

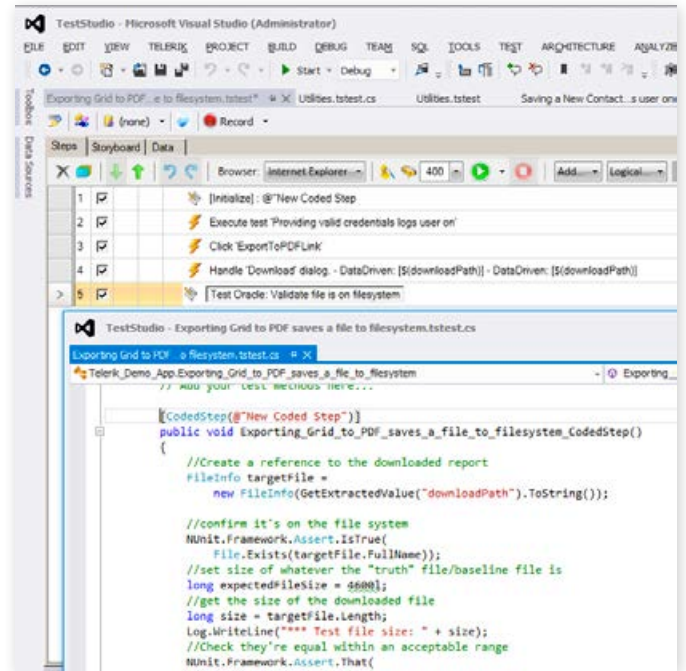
As a developer or coding tester you're not limited to one browser. The Visual Studio plugin for Test Studio enables you to work with all our supported browsers: Firefox, Safari, Chrome, and Internet Explorer.

Familiar Environment

Perhaps most important to developers is the ability to work right in an environment you're familiar and comfortable with. You've undoubtedly customized Visual Studio to fit your particular needs: display settings, window behavior and layout, and any number of other plugins and productivity tools such as [Telerik's JustCode](#).

Working with Test Studio inside of Visual Studio leaves you the ability to pull in other tools easily through NuGet. Do you prefer using NUnit or MbUnit for their powerful Assert and fluent interfaces? Go right ahead! (Note that you can also use outside assemblies easily straight from within Test Studio Standalone, too. You won't be handicapping other team members using that interface!)

Here's another example of using a test oracle for validation, this time via NUnit. The example below shows evaluating a downloaded file against a baseline or "truth" file. NUnit's "Within" clause from its fluent API makes range comparisons simple.



Source control integration for your test suites is a snap from within Visual Studio too. Use the exact same approach you do with your current toolset. Test Studio, either the Standalone or Visual Studio plugin versions, play nicely with all source control tooling, so you won't have to alter your habits, teams' workflows, or worse yet, change your source control system.

[Read more about Test Studio's plugin for Visual Studio](#)

[Download 30-day trial with priority support](#)



Telerik
TEST STUDIO