How to stop Visual Studio from creating launchsettings.json

Share on:

In this short post I show how to stop an annoying behaviour of Visual Studio where it will keep creating a *launchsettings.json* file for integration test projects.

**tl;dr;** To stop Visual Studio from automatically creating launchsettings.json for projects that use the Web SDK, add <NoDefaultLaunchSettingsFile>true</NoDefaultLaunchSettingsFile> to your csproj.

[Background: Integration testing with *Microsoft.AspNetCore.Mvc.Testing*](https://andrewlock.net/how-to-stop-visual-studio-from-creating-launchsettings-json/#background-integration-testing-with-microsoft-aspnetcore-mvc-testing)

One of the great things about ASP.NET Core compared to the previous version of ASP.NET is that it's much easier to run integration tests. Being able to easily spin-up an instance of your app in a Docker container makes end-to-end testing far simpler.

In addition, ASP.NET Core allows for complete in-memory testing of your application, by using the TestServer available in the *Microsoft.AspNetCore.TestHost* package instead of Kestrel. This makes it easy to test your application inside a unit test, keeping everything in memory.

Note that TestServer doesn't support all the same features as Kestrel, so in some cases you may find differences in behaviour. I haven't personally run into any of those cases, but it's worth being aware of.

In addition to the *Microsoft.AspNetCore.TestHost* package that provides TestServer, Microsoft also provide a package called *Microsoft.AspNetCore.Mvc.Testing*. This package is designed specifically for testing MVC/Razor Pages applications, and [handles some of the tricky extra tasks](https://docs.microsoft.com/en-us/aspnet/core/test/integration-tests?#:%7E:text=handles%20the%20following%20tasks) you need to do for things to work correctly:

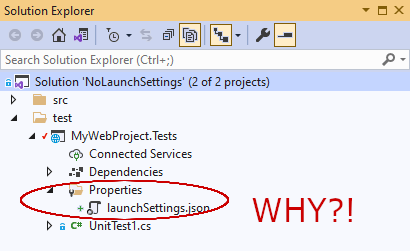
* Setting the "content root" for the test to your real application's project folder so that Razor views/pages can be found
* Copies *.deps* files to the appropriate folders
* Provides the WebApplicationFactory helper for automatically bootstrapping your application

I'm not going to go into the specifics of integration testing in this post. If you want to learn more, there are lots of resources available:

* I've written about integration testing several times on my blog, including in [this post](https://andrewlock.net/converting-integration-tests-to-net-core-3/) where I describe the transition from ASP.NET Core 2.x to 3.x
* I discuss integration testing with TestHost and WebApplicationFactory in [Chapter 23 of my book, ASP.NET Core in Action, second edition.](https://www.manning.com/books/asp-net-core-in-action-second-edition?utm_source=aspnetcore-in-action&utm_medium=affiliate&utm_campaign=book_lock2_asp_5_19_20&a_aid=aspnetcore-in-action&a_bid=44c089ee)
* The Microsoft documentation describes [how to use the package to test a Razor Pages app](https://docs.microsoft.com/aspnet/core/test/integration-tests)
* [Steve Gordon](http://stevejgordon.co.uk/) has a Pluralsight course: [Integration Testing ASP.NET Core Applications: Best Practices.](https://www.pluralsight.com/courses/integration-testing-asp-dot-net-core-applications-best-practices)

One of the only pre-requisites for using the *Microsoft.AspNetCore.Mvc.Testing* package is that your *test* project must reference the *Web* SDK in its project file (using <Project Sdk="Microsoft.NET.Sdk.Web">).

Unfortunately, this has one undesirable outcome: Visual Studio thinks that your test project is an ASP.NET Core application, so it "helpfully" creates a *launchsettings.json* file every time you open the project. This can be annoying, but [I recently found out that](https://github.com/DataDog/dd-trace-dotnet/pull/1414#discussion_r616748146) we can tell Visual Studio to stop it!



To see this in action, I'll create a dummy test project, show how Visual Studio behaves, and then show how to tell Visual Studio to stop it!

[Creating the project](https://andrewlock.net/how-to-stop-visual-studio-from-creating-launchsettings-json/#creating-the-project)

All we need for this example is a project that uses the Microsoft.NET.Sdk.Web SDK, but for now I'll create a semi-realistic solution. The following PowerShell snippet creates an "empty" ASP.NET Core web project in the *src* folder, a test project in the *test* folder, adds them to a *.sln* file, adds a reference to the web project into the test project, and adds the *Microsoft.AspNetCore.Mvc.Testing* project to the test project:

mkdir src

mkdir test

dotnet new sln

dotnet new web -o ./src/MyWebProject

dotnet new xunit -o ./test/MyWebProject.Tests

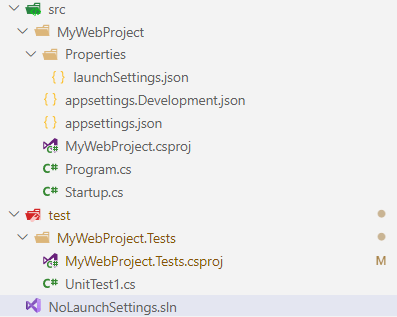
dotnet sln add ./src/MyWebProject

dotnet sln add ./test/MyWebProject.Tests

dotnet add ./test/MyWebProject.Tests reference ./src/MyWebProject

dotnet add ./test/MyWebProject.Tests package Microsoft.AspNetCore.Mvc.Testing

If we open the solution in Visual Studio Code, we can see that the web project has a *launchsettings.json* file in the *Properties* folder. This is standard and *useful* as it means we can easily control how we launch the application when developing locally by adding environment variables etc.



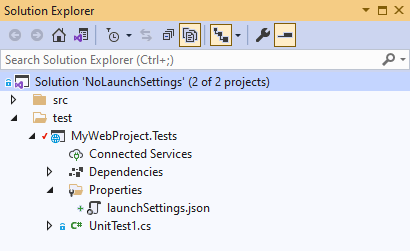
As I mentioned earlier, [one of the pre-requisites](https://docs.microsoft.com/en-us/aspnet/core/test/integration-tests?#test-app-prerequisites) for using the *Microsoft.AspNetCore.Mvc.Testing* package is that your *test* project must reference the Microsoft.NET.Sdk.Web SDK in its project file. Open the *MyWebProject.Tests.csproj* file in VS Code and change the first line from

<Project Sdk="Microsoft.NET.Sdk">

to

<Project Sdk="Microsoft.NET.Sdk.Web">

We've finished the setup for our test solution now. Now lets see what Visual Studio does when we open our solution:



Visual Studio has seen the Microsoft.NET.Sdk.Web SDK reference, and thinks "oh, *MyWebProject.Testing* must be a web project, I'll be useful and create a *launchsettings.json* file". Unfortunately, this *isn't* useful, as it's a test project. We're not going to be running the test project directly, rather we're going to be running it indirectly using dotnet test.

It doesn't really *matter* if the file exists, but it irks me. Deleting the file just means that Visual Studio will recreate it later. Luckily, we can tell Visual Studio to stop it, and that we know better.

[Stopping Visual Studio from generating a launchsettings.json file](https://andrewlock.net/how-to-stop-visual-studio-from-creating-launchsettings-json/#stopping-visual-studio-from-generating-a-launchsettings-json-file)

The simple solution is to add the NoDefaultLaunchSettingsFile property to your test application and set it to true, as shown in the following:

<Project Sdk="Microsoft.NET.Sdk.Web">

<PropertyGroup>

<TargetFramework>net5.0</TargetFramework>

<IsPackable>false</IsPackable>

*<!-- Add this line -->*

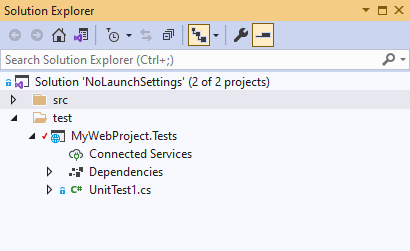
<NoDefaultLaunchSettingsFile>true</NoDefaultLaunchSettingsFile>

</PropertyGroup>

*<!-- PackageReference nodes not shown for brevity -->*

</Project>

Now when we open the solution in Visual Studio, VS will leave our test project alone, keeping it nice and clean:



You can remove the existing *launchsettings.json* file using:

rm -R ./test/MyWebProject.Tests/Properties

and rest easy with the knowledge that Visual Studio won't come back and recreate it again later!

[Summary](https://andrewlock.net/how-to-stop-visual-studio-from-creating-launchsettings-json/#summary)

By default, Visual Studio will generate a *launchsettings.json* file for any project that references the Microsoft.NET.Sdk.Web SDK. This is normally fine, but projects using the *Microsoft.AspNetCore.Mvc.Testing* package must also reference this SDK. This causes Visual Studio to generate the file for test projects that don't need it. To disable this behaviour, add <NoDefaultLaunchSettingsFile>true</NoDefaultLaunchSettingsFile> to your project file.