



# xUnit Testing

---

.NET CORE

*xUnit.net is a free, open source, community-focused unit testing tool for the .NET Framework. xUnit.net is part of the .NET Foundation.*

[HTTPS://XUNIT.NET/](https://xunit.net/)

# Arrange, Act, Assert

<https://docs.microsoft.com/en-us/visualstudio/test/unit-test-basics?view=vs-2019>

---

The Arrange, Act, Assert pattern is a common way of writing unit tests for a method under test.

- The Arrange section of a unit test method initializes objects and sets the value of the data that is passed to the method under test.
- The Act section invokes the method under test with the arranged parameters.
- The Assert section verifies that the action of the method under test behaves as expected.

```
[TestMethod]
public void Withdraw_ValidAmount_ChangesBalance()
{
    // arrange
    double currentBalance = 10.0;
    double withdrawal = 1.0;
    double expected = 9.0;
    var account = new CheckingAccount("JohnDoe", currentBalance);

    // act
    account.Withdraw(withdrawal);

    // assert
    Assert.AreEqual(expected, account.Balance);
}
```

# xUnit Testing Step By Step in Visual Studio

<https://docs.microsoft.com/en-us/dotnet/core/testing/unit-testing-with-dotnet-test#create-a-test>  
<https://docs.microsoft.com/en-us/dotnet/core/testing/unit-testing-with-dotnet-test>

---

1. Open a Solution in Visual Studio.
2. Right-Click the Solution.
3. Add >> new project...
4. Type “xunit” in the template search box.
5. Select xUnit Test Project(.NET Core).
6. Name the project whatever you want (VS inserts ‘\_’ for spaces).
7. Right-Click ‘Dependencies’ in the test project.
8. Click ‘Add Reference’
9. In the left pane of the Add References window, click ‘Projects’
10. In the center pane, click to check the Projects containing methods you want to test.
11. Click ‘OK’
12. Add your tests to the Test project.

# xUnit Testing in VS Code Step-by-Step

<https://docs.microsoft.com/en-us/dotnet/core/testing/unit-testing-with-dotnet-test>

---

1. `dotnet new sln -o [sln name]` - Create a .sln and a directory of the same name holding that .sln to which you will add the various projects
3. `cd [newDirectoryName]` into the new directory.
4. `dotnet new console -o [projectName]` - to create the project to be tested. You can skip this step if you already have an app created.
5. Add whatever code you want to the project (i.e. things to test).
6. `dotnet sln add ./[directoryOfProject]/[NameOfProject].csproj` - to add the project to the .sln
7. `dotnet new xunit -o [testingProjectName].Tests` - to create the testing project
8. `dotnet sln add ./[directoryOfTestingProject]/[NameOfTestingProject].csproj` - to add the testing project to the solution.
9. `dotnet add`  
    `./[directoryOfTestingProject]/[NameOfTestingProject].csproj reference ./[directoryOfProject]/[NameOfProject].csproj` - to add the project as a dependency to the testing project
10. Make sure your project classes are public.
11. Add `using [projects' namespace];` to the testing project.
12. Create tests in the testing project
13. From the testing project directory, run `dotnet test` to run the tests.

# InMemory DB Step-by-Step

<https://docs.microsoft.com/en-us/ef/core/miscellaneous/testing/in-memory>

---

**EF Core** database providers do not have to be relational databases. *InMemory* is designed to be a general purpose database for testing, and is not designed to mimic a relational database.

There are a few steps to setting up a *InMemory* DB.

1. Set up the constructor in your DB Context class to accept a DB configuration parameter called *DbContextOptions*.

```
public class BloggingContext : DbContext
{
    public BloggingContext()
    { }

    public BloggingContext(DbContextOptions<BloggingContext> options)
        : base(options)
    { }
}
```

# InMemory DB

## Step-by-Step

<https://docs.microsoft.com/en-us/ef/core/miscellaneous/testing/in-memory>

---

2. Alter your ***DbContext.OnConfiguring()*** to check for an already configured DB and not use your production DB if there's already a DB configured.

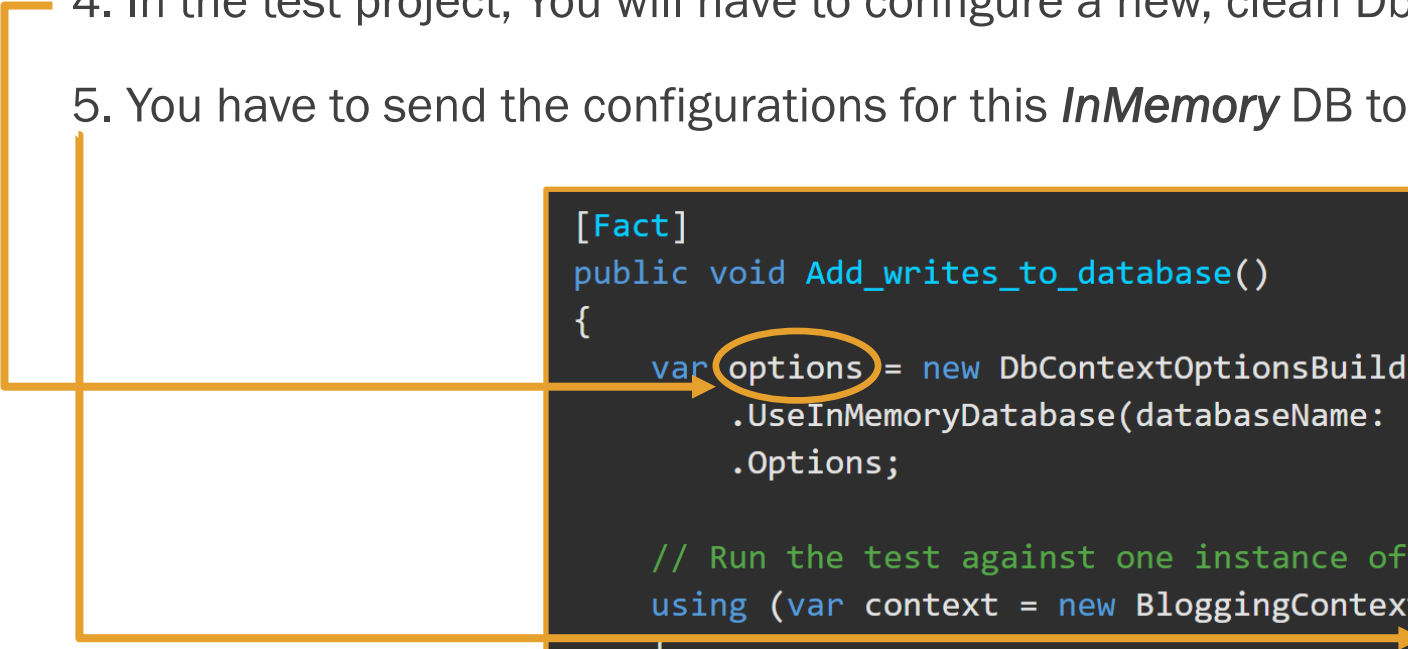
```
protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
{
    if (!optionsBuilder.IsConfigured)
    {
        optionsBuilder.UseSqlServer(@"Server=(localdb)\mssqllocaldb;Database=EF
    }
}
```



# InMemory DB Step-by-Step

<https://docs.microsoft.com/en-us/ef/core/miscellaneous/testing/in-memory>

3. Right-click your test project to download the NuGet Package *Microsoft.EntityFrameworkCore.InMemory*.
4. In the test project, You will have to configure a new, clean Db context for every test.
5. You have to send the configurations for this *InMemory* DB to the DbContext Constructor on instantiation.



```
[Fact]
public void Add_writes_to_database()
{
    var options = new DbContextOptionsBuilder<BloggngContext>()
        .UseInMemoryDatabase(databaseName: "Add_writes_to_database")
        .Options;

    // Run the test against one instance of the context
    using (var context = new BloggngContext(options))
    {
        var service = new BlogService(context);
        service.Add("https://example.com");
        context.SaveChanges();
    }
}
```



# InMemory DB Step-by-Step

<https://docs.microsoft.com/en-us/ef/core/miscellaneous/testing/in-memory>

Here is a  
sample test for  
comparison.

Arrange

```
[Fact]
public void Find_searches_url()
{
    var options = new DbContextOptionsBuilder<BlogggingContext>()
        .UseInMemoryDatabase(databaseName: "Find_searches_url")
        .Options;
```

Act

```
    // Insert seed data into the database using one instance of the context
    using (var context = new BlogggingContext(options))
    {
        context.Blogs.Add(new Blog { Url = "https://example.com/cats" });
        context.Blogs.Add(new Blog { Url = "https://example.com/catfish" });
        context.Blogs.Add(new Blog { Url = "https://example.com/dogs" });
        context.SaveChanges();
    }
```

Assert

```
    // Use a clean instance of the context to run the test
    using (var context = new BlogggingContext(options))
    {
        var service = new BlogService(context);
        var result = service.Find("cat");
        Assert.Equal(2, result.Count());
    }
}
```

# Great resource

---

<https://www.thereformedprogrammer.net/using-in-memory-databases-for-unit-testing-ef-core-applications/>