Breaking the Ice with .NET Core 1.0

Jorge Serrano Pérez

Microsoft MVP

V1.0 – July 2016

Introduction

Disclaimer: First all, apologies for the possible grammar mistakes that I can take writing this ebook in English, but is not my mother tongue.
In other case, and the most important, I hope to transmit the knowledge correctly for all people reading this.

The objective of this ebook is help to the people who want to break the first bounds and stoppers when they want to start with a technology.

In this case, I want to help you to start your road with .NET Core 1.0.

I am not going to teach you all details of .NET Core 1.0 because they are out of scope of this ebook, and instead I want to cover some things about .NET Core practicing and enjoying from the first time without think too much in other things.
But all those things are very important too that you will have to learn in the future (VERY NECESSARY AND RECOMMENDED).

Here I want to write a practical ebook, with simple and easy exercises, using different alternatives to break the first barriers that will help you to continue learning and deeping about .NET Core.

This book is not a silver bullet, of course, is a modest help to start walking in .NET Core. I hope that helps!.

In other case, if you prefer start to see or investigate more about .NET Core, I invite you to visit the official site on Internet in the next link:
https://dotnet.github.io/

How use this ebook?

To clarify some parts of this ebook, I am going to identify with some icons, the OS and tools that you will need to use.

The icon to indicate a tip, is:
Tip

The icon to indicate that you have to install something is:
Install

The icons to indicate the OS we have to use in the sample code are:
Windows
Mac

The icons to indicate the tools I have used will be:
Notepad

VS Code
VS 2015
Command

All samples of code have been compiled and tested with the RTM 1.0 version of .NET Core.

What tools I will need?

All code samples are based in C# as programming language, using as Operating System a **Windows** system or a **Mac OS** system.

Both Operating Systems should have .NET Core installed.
As recommended OS, Windows 10 for Windows platforms, and Yosemite or El Capitan for Mac OS.

Note: Don't forget to install .NET Core before start, and Microsoft Visual Studio Code and Microsoft Visual Studio 2015 if you want to cover all samples of code of this ebook without problems.

With all these tools, you will be ready to start from this point.
Welcome and enjoy it!

Install .NET Core before to start

Note: I assume that you have installed Microsoft Visual Studio 2015 (if you have a Windows Operating System), and Microsoft Visual Studio Code (for Windows and Mac).

If you have installed .NET Core, you can jump this chapter.
If you didn't install it before you will have to install .NET Core now and restart your computer to apply the changes correctly.

If you want to read some details and more information about .NET, you can access to this link:
https://www.microsoft.com/net/

Windows
For Windows, you should install these packages (one or all of them as I did to learn all possibilities):

Install
.NET Core SDK for Windows
https://go.microsoft.com/fwlink/?LinkID=809122

Install
Or if you have Visual Studio 2015:

Microsoft Visual Studio Update 3
https://www.visualstudio.com/news/releasenotes/vs2015-update3-vs

.NET Core for Visual Studio
https://go.microsoft.com/fwlink/?LinkId=817245

Mac

For Mac, you should install the next packages (for Yosemite or El Capitan):

Install

.NET Core here:
https://www.microsoft.com/net/core#macos

Other additional Software packages

Visual Studio Code

Install

Visual Studio Code runs in both, Windows and Mac.

You can install this editor from this link:
https://code.visualstudio.com/

Visual Studio Express

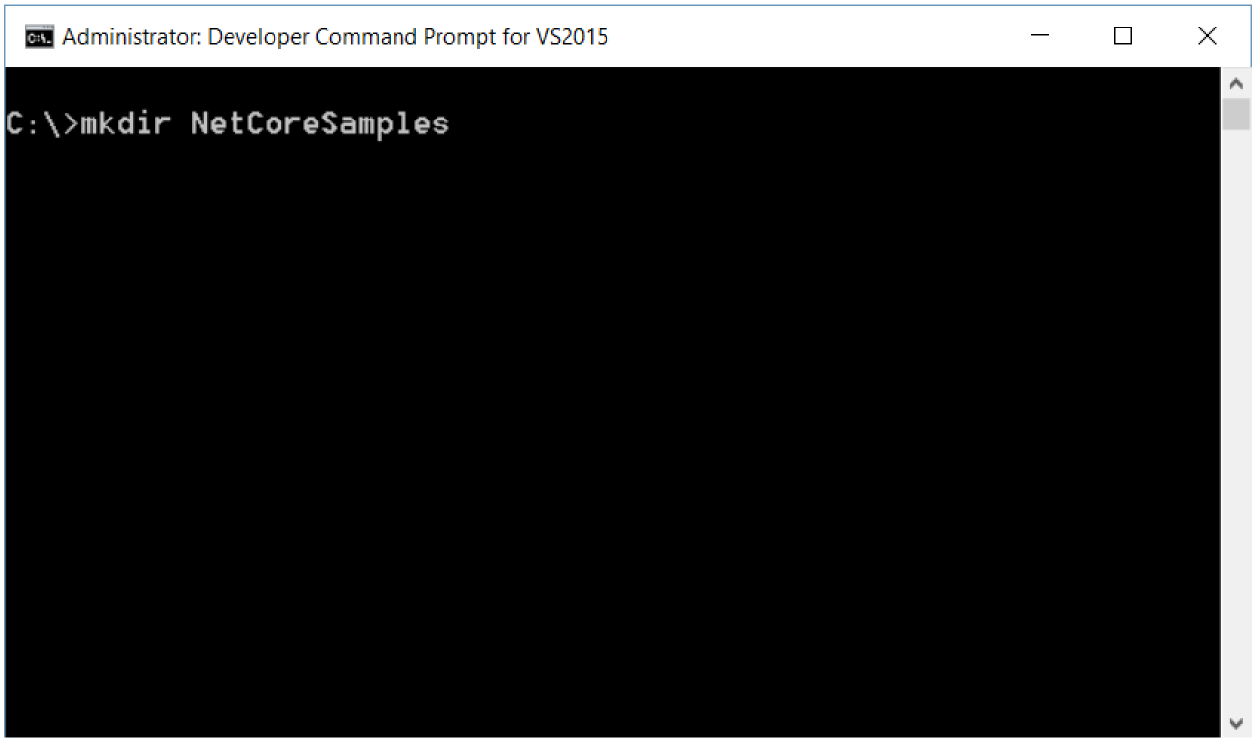
Install

Visual Studio Express will help you to work with an environment similar to Visual Studio 2015 to start with .NET Core (only for Windows).

You can install it from here:
https://www.visualstudio.com/es-es/products/visual-studio-express-vs.aspx

Organize your code

Before to start, you should create a folder to organize your code.
In my case, I have created in Windows a folder name “**NetCoreSamples**”.
You can create this folder as you want.



```
C:\>mkdir NetCoreSamples
```

Inside of this folder, I will create the next projects as subfolders to organize the code.

01.Hello World

Windows Notepad Command

Here, you are going to create the typical “Hello World” sample.
A very easy and simple sample to begin your road through .NET Core.

First all, you are going to create a new folder that I have named as “**01.Hello World**”.
After that, you should open a **Developer Command Prompt** for Visual Studio 2015 to set the prompt in the directory that you created before.

The first action that you will have to do will be to create a **project.json** file.
We will open a **Notepad** and we will write a file named **project.json**.
Inside of this file, we will write the next code:

```
{
  "version": "1.0.0-*",
  "buildOptions": {
    "emitEntryPoint": true
  },
  "dependencies": {
    "Microsoft.NETCore.App": {
      "type": "platform",
      "version": "1.0.0"
    }
  },

  "frameworks": {
    "netcoreapp1.0": {
      "imports": "dnxcore50"
    }
  }
}
```

The **project.json** file contains the list of all dependencies of our application.
Here, we can add some extra actions as the output action, the version of our application, etc.

Now, from the command prompt, you are going to write the next command:
dotnet restore

A **project.lock.json** will be created now.
We have to think about this file as a package folder for the project.

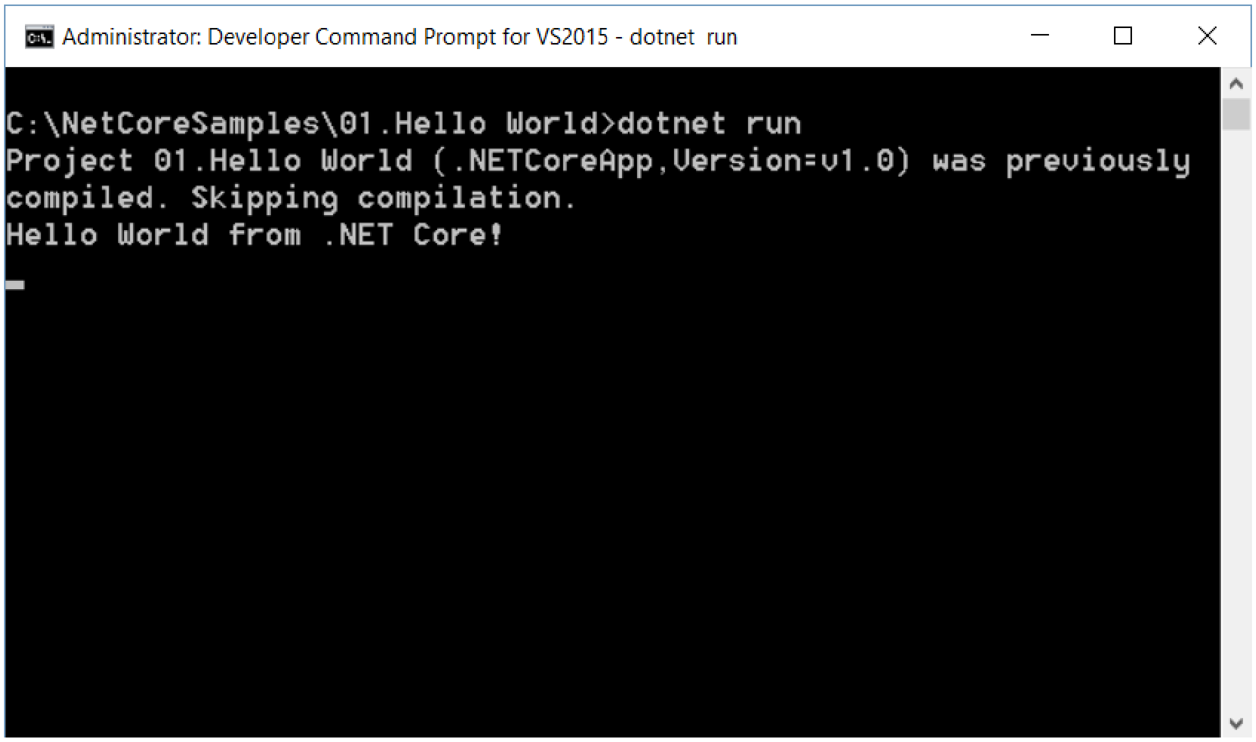
Now, you are going to write the code of your application.
Write the next code in the file **Program.cs**.

```
namespace NetCoreSamples
{
    using System;

    public class Program
    {
        public static void Main(string[] args)
        {
            Console.WriteLine("Hello World from .NET Core!");
            Console.ReadLine();
        }
    }
}
```

Now, execute the command in the command prompt to build our project:
dotnet build

And finally, execute our first sample in .NET Core from the commando prompt executing the command:
dotnet run

As you can see, our first .NET Core App is running!!!


Administrator: Developer Command Prompt for VS2015 - dotnet run

```
C:\NetCoreSamples\01.Hello World>dotnet run
Project 01.Hello World (.NETCoreApp,Version=v1.0) was previously
compiled. Skipping compilation.
Hello World from .NET Core!
```

02.Distance Converter

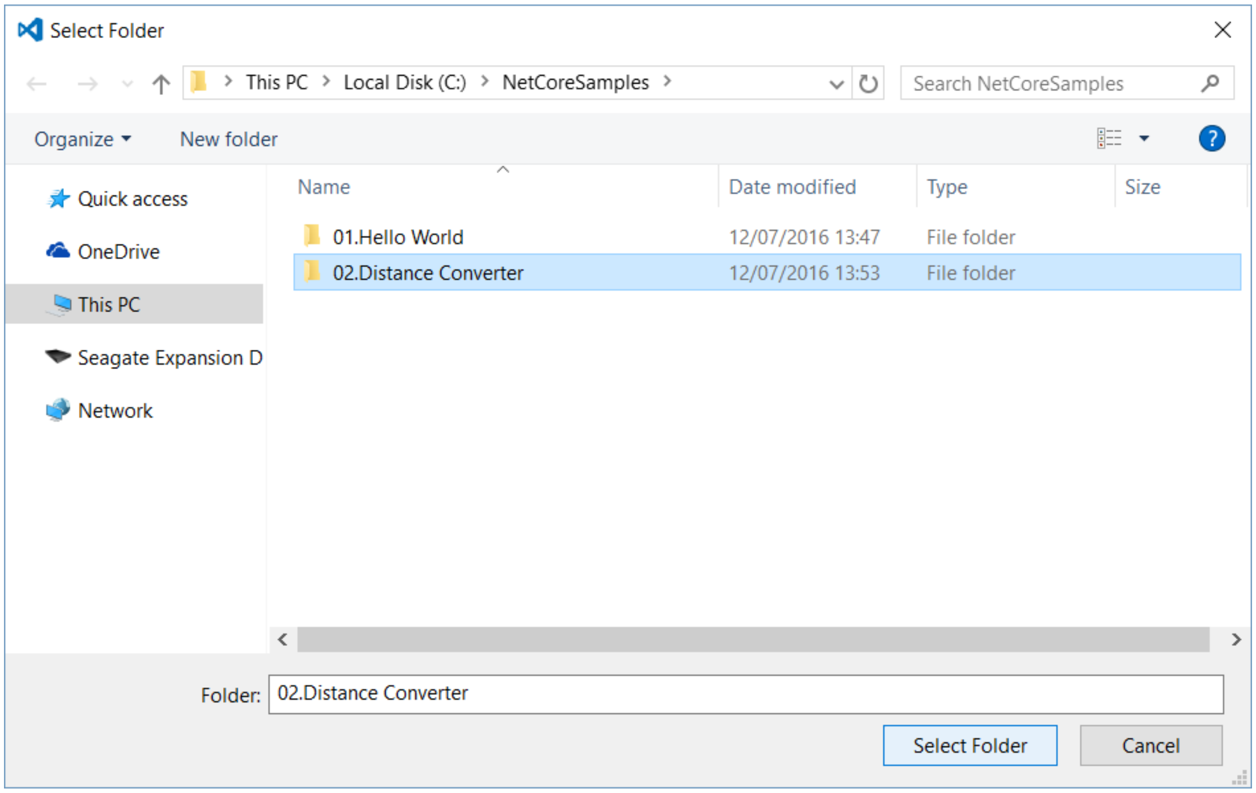
Windows VS Code

In this time, you are going to create another simple sample of code.
A distance converter application that will convert from Kilometers to Miles and viceversa.

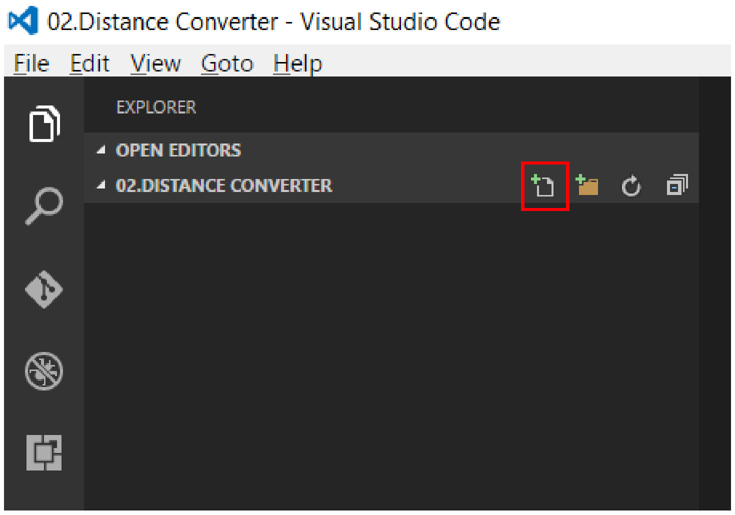
First all, you are going to create a new folder that I have named as "**02.Distance Converter**".
After that, you should open a **Developer Command Prompt** for Visual Studio 2015 to set the prompt in the directory that we created before.

From Visual Studio Code, you are going to click on the menu **File > Open Folder**.

Go to the path where is your project folder and select it doing click on the **Select Folder** button of this window.



Now, in the Visual Studio Code editor, you should do click on the **New File** icon.



Write **project.json** as new file, and add the next code (identical to the previous *Hello World* sample code if you want copy/paste the code):

```
{
  "version": "1.0.0-*",
  "buildOptions": {
    "emitEntryPoint": true
  },

  "dependencies": {
    "Microsoft.NETCore.App": {
      "type": "platform",
      "version": "1.0.0"
    }
  },

  "frameworks": {
    "netcoreapp1.0": {
      "imports": "dnxcore50"
    }
  }
}
```

Now, from the command prompt, write and execute the next command:
dotnet restore

A **project.lock.json** will be created now as we saw in the *Hello World* sample code.

Now, we are going to write the code of our application.
Add a new file to the project as **Converter.cs**.
Write the next code for this file.

```
namespace NetCoreSamples
{

    public class Converter
    {

        public static double KilometersToMiles(double kilometers)
        {
                    return kilometers * 0.621371192;
        }


        public static double MilesToKilometers(double miles)
        {
            return miles * 1.609344;
        }


    }
}
```

Repeat the same steps but in this case to create the file **Program.cs**, and write the next code:

```
namespace NetCoreSamples
{
    using System;


    public class Program
    {
        public static void Main(string[] args)
        {
                        var action = String.Empty;


            while (action != "X")
            {
                Console.Clear();
                Console.WriteLine("X = Exit | M = Convert to Miles | K = Convert to Kilometers");


                action = Console.ReadLine();


                // Kilometers to Miles
                if (action == "M")
                {
                    Console.Clear();
                    Console.WriteLine("Write the kilometers");
                    var kilometers = Convert.ToDouble(Console.ReadLine());
                    Console.WriteLine(Converter.KilometersToMiles(kilometers).ToString());
                    Console.ReadLine();
                }
                // Miles to Kilometers
                if (action == "K")
                {
                    Console.Clear();
                    Console.WriteLine("Write the miles");
                    var miles = Convert.ToDouble(Console.ReadLine());
                    Console.WriteLine(Converter.MilesToKilometers(miles).ToString());
                    Console.ReadLine();
                }
            }

            Console.Clear();
            Console.WriteLine("End of program");
            Console.ReadLine();
        }
    }
}
```

Now, execute the command in the command prompt:
dotnet build

And finally, run this sample from the command prompt executing the command:
dotnet run

Tip

If we write the command **dotnet run** directly, .NET Core will execute the **build** action first, and the **run** action finally.

03.Count Words

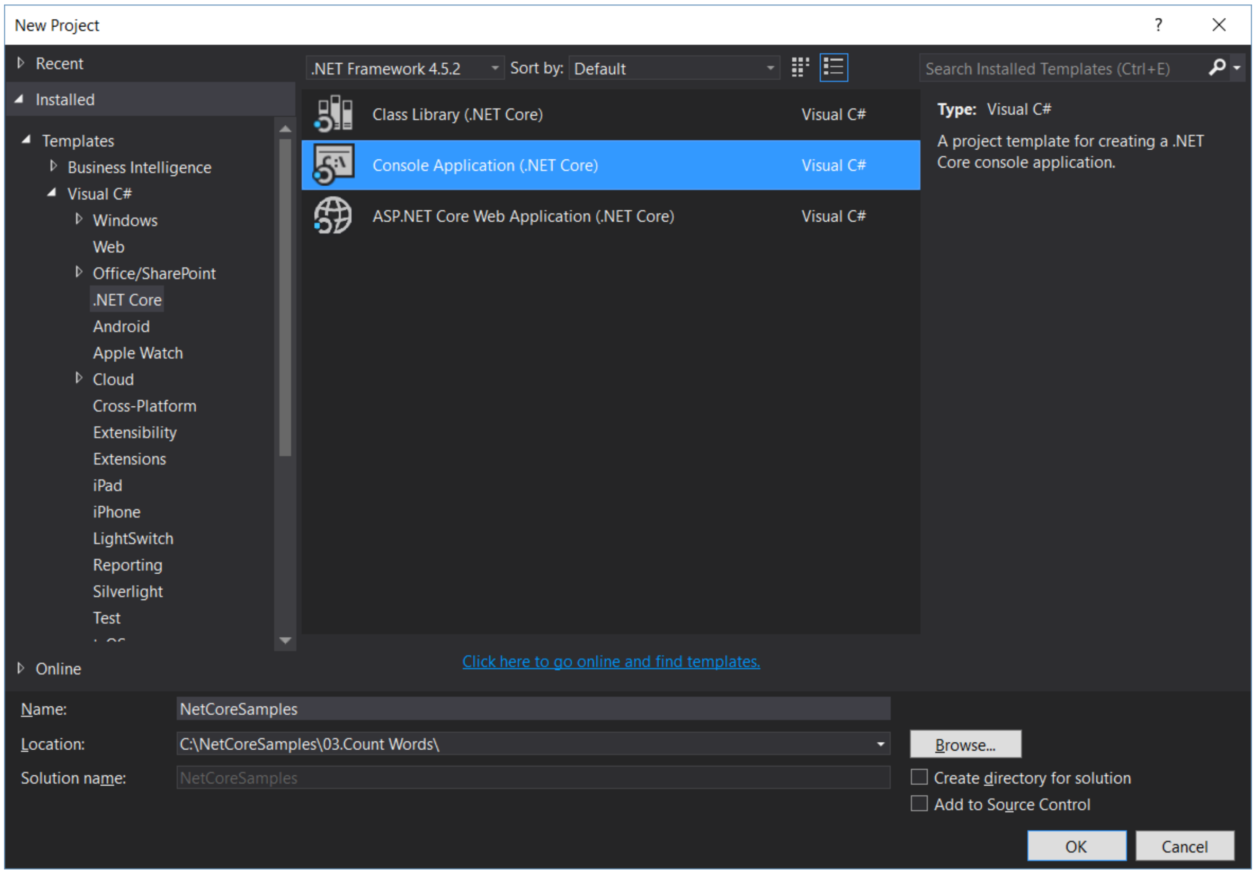
Windows VS 2015

In this sample, you are going to create a sample with Visual Studio 2015 to count the number of words in a phrase.

First all, you are going to create a new folder that I have named as "**03.Count Words**".
As we did in the other samples, you should open a **Developer Command Prompt** for Visual Studio 2015 to set the prompt in the directory that we created before.

Open Visual Studio 2015 now.

Click on the **New Project** option of Visual Studio 2015, selecting **.NET Core** templates, and of all templates, **Console Application (.NET Core)** as you can see in the next image.



Class Library (.NET Core)	Visual C#
Console Application (.NET Core)	Visual C#
ASP.NET Core Web Application (.NET Core)	Visual C#

Type: Visual C#

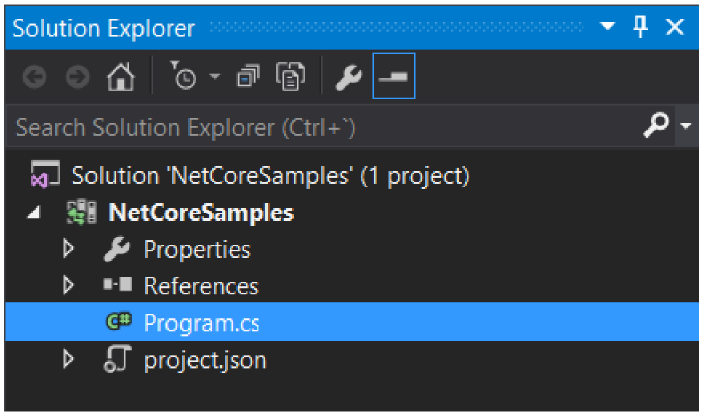
A project template for creating a .NET Core console application.

Name: NetCoreSamples

Location: C:\NetCoreSamples\03.Count Words\

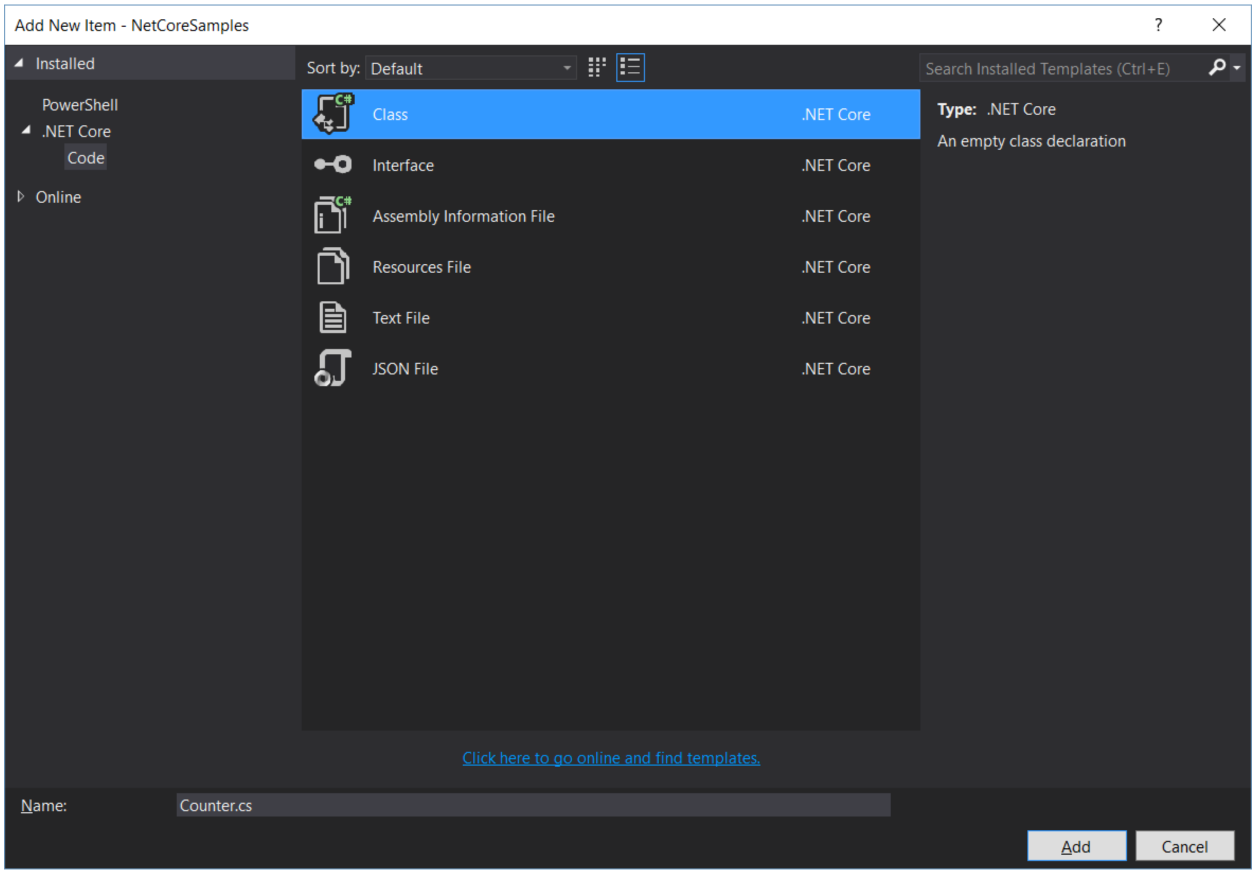
Solution name: NetCoreSamples

We will click on the **OK** button to create the solution and the project.



As you can see here, the template has created the files **project.json** and **Program.cs** for us.

Here, you are going to create a new file.
Click with the right mouse button in the **Solution Explorer** window and select the **Add > Class** option.
Name the new class as **Counter.cs**.


Add New Item - NetCoreSamples

Installed
PowerShell
.NET Core
Code
Online
Sort by: Default
Search Installed Templates (Ctrl+E)

Class	.NET Core
Interface	.NET Core
Assembly Information File	.NET Core
Resources File	.NET Core
Text File	.NET Core
JSON File	.NET Core

Type: .NET Core
An empty class declaration

Click here to go online and find templates.

Name: Counter.cs

Write the next code for this file.

```
namespace NetCoreSamples
{

  using System;

  public class Counter
  {

    public int WordsFromText(string text)
    {
      string[] source = text.Split(new char[] { '.', '?', '!', ' ', ';', ':', ',' }, StringSplitOptions.RemoveEmptyEntries);

      return source.Length;
    }
    }


}
```

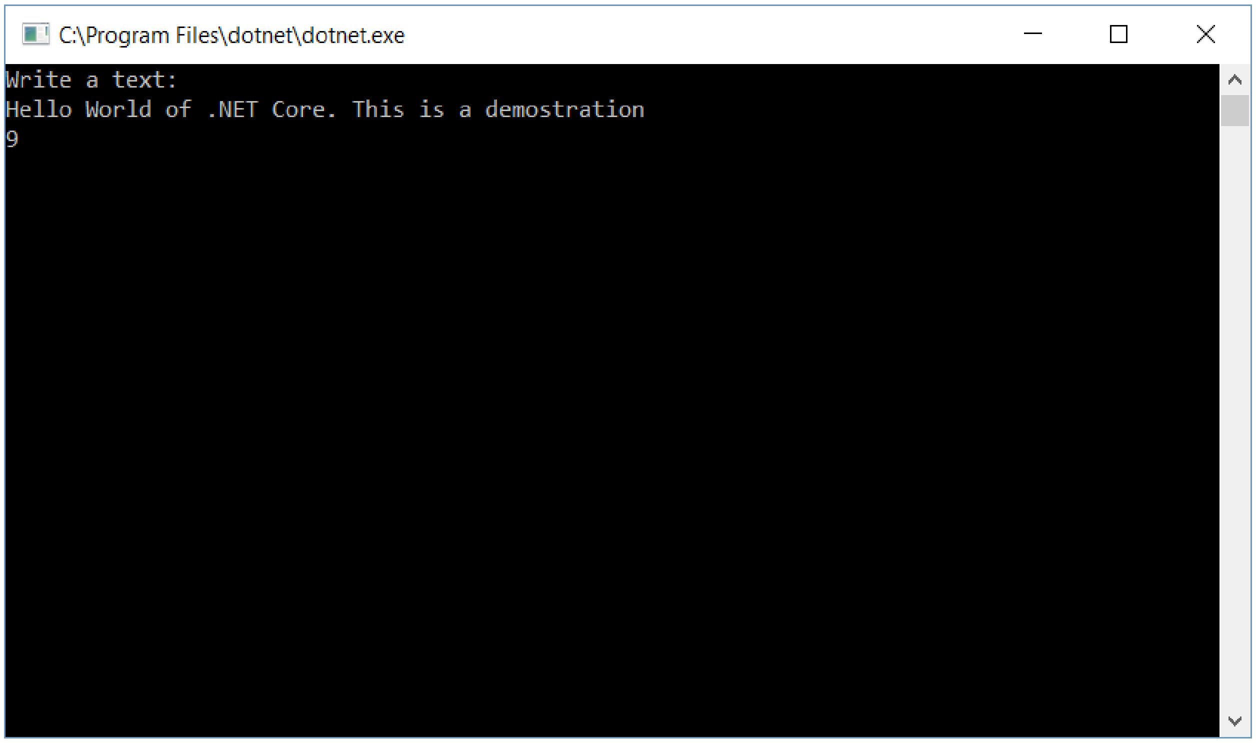
Now, modify the **Program.cs** code with the next code:

```
namespace NetCoreSamples
{

    using System;

    public class Program
    {

        public static void Main(string[] args)
        {
            Console.WriteLine("Write a text:");
            var text = Console.ReadLine();

            var counter = new Counter();
            Console.WriteLine(counter.WordsFromText(text).ToString());

            Console.ReadLine();
        }

    }

}
```

And finally, execute the application or press the **F5** key in Visual Studio 2015.

Our application will be executed from Visual Studio 2015.
If we set the path of this code sample with the command prompt and execute the command **dotnet run**, the application will be executed too as we saw in the previous samples.

Our demo in execution is what you can see in the next image:



```
Write a text:
Hello World of .NET Core. This is a demostration
9
```

04.Temperature Converter

Mac VS Code

You are going to create a new project with Visual Studio Code and C# in Mac OS X.
The project is going to convert the temperature in Farenheit to Celsius.

Note: I suppose that you have installed .NET Core and Visual Studio Code on your Mac previously.

Tip

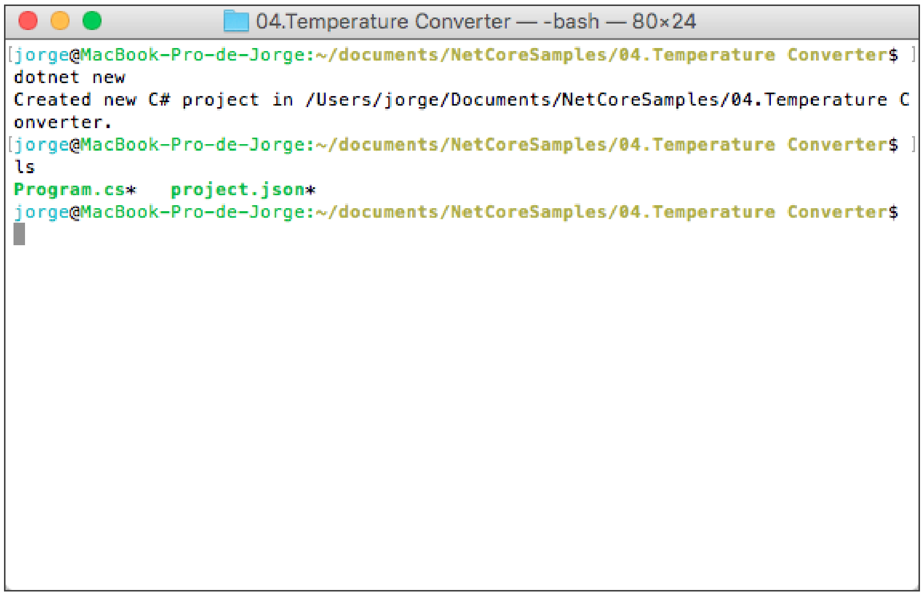
If you have installed Visual Studio Code on your Mac, you can install the **ms-vscode.csharp** extension.
https://marketplace.visualstudio.com/items?itemName=ms-vscode.csharp

First all, you should create the directories on the Mac as we created them in Windows.
Inside of the **NetCoreSamples** directory, you are going to create a new folder that I have named as "**04.Temperature Converter**".
Furthermore, you should open a **Terminal bash prompt** on our Mac to execute the commands of .NET Core.

After all these steps, you will have to execute the next command in the Terminal prompt:
dotnet new

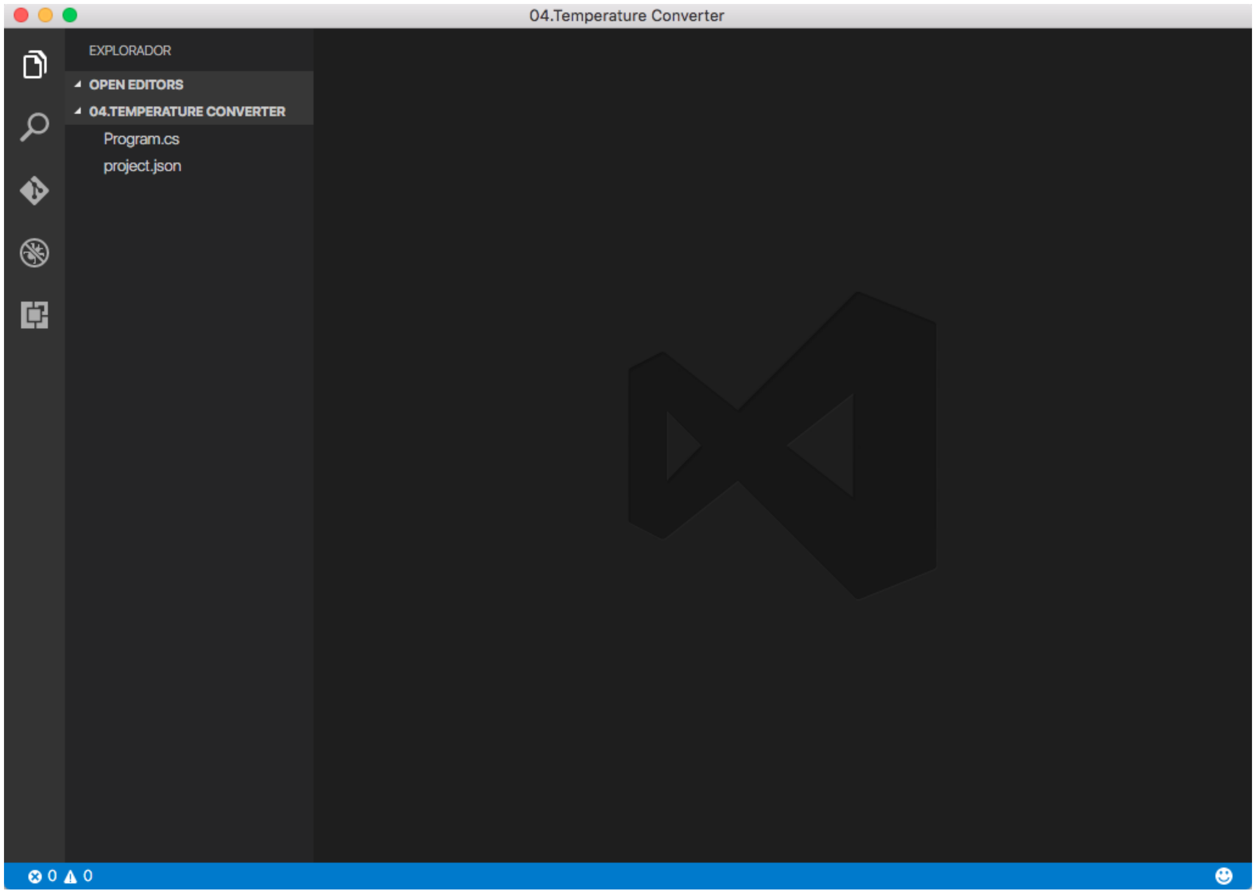
You will see a message with the text "*Created new C# project in …*".

This command will generate two files for us: **project.json** and **Program.cs**.
You can see all this in the next image:

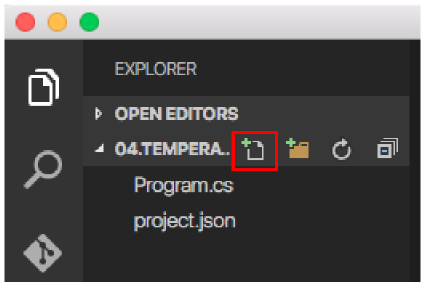

04.Temperature Converter — -bash — 80×24

```
[jorge@MacBook-Pro-de-Jorge:~/documents/NetCoreSamples/04.Temperature Converter$ ]
dotnet new
Created new C# project in /Users/jorge/Documents/NetCoreSamples/04.Temperature C
onverter.
[jorge@MacBook-Pro-de-Jorge:~/documents/NetCoreSamples/04.Temperature Converter$ ]
ls
Program.cs*   project.json*
jorge@MacBook-Pro-de-Jorge:~/documents/NetCoreSamples/04.Temperature Converter$
```

Now, in Visual Studio Code, you are going to open the project. To do this, do click in **File > Open**.
You are going to select the path that we created before to start this project, and click on the **Open** button.
You should see the project opened in Visual Studio Code as you can see in the next image.



Now, you are going to create a new file (a C# class).
To add a new class in the project you are going to do click on the **New File** option as you can see in the next image.



Here, write **Converter.cs** as file name.
And write something of code in this class of C# too.

```
namespace ConsoleApplication
{

    public class Converter
    {

        public double FarenheitToCelsius(double temperature)
        {
            return (5.0/9.0) * (temperature - 32);
        }

    }

}
```

Modify the **Program.cs** code with the next code now:

```
namespace ConsoleApplication
{

    using System;

    public class Program
    {
```

```
    public static void Main(string[] args)
    {
      Console.WriteLine("Write the Farenheit temperature:");
      var temperature = Console.ReadLine();

      var converter = new Converter();
      var result = converter.FarenheitToCelsius(Convert.ToDouble(temperature));

      Console.WriteLine(String.Format("The temperature in Celsius is {0}", result));
      Console.ReadLine();
    }

  }

}
```

Execute before run the application the next command in the Terminal prompt:
dotnet restore

And finally, execute the command to run our application:
dotnet run

Tip

You can restore the packages of your application from Visual Studio Code directly.
Press the keys **Command Shift P** in Visual Studio Code, and write **dotnet**.
You will see the command **dotnet:Restore Packages**.
If do click on this command, the **dotnet restore** command will appear and if you select after the project, this command will be executed for it.

05.App Settings

Mac VS Code

In this code sample you are going to read an app settings file and write some text simulating a simple log process.

The steps here are very similar to the same steps in the Temperature Converter sample.

First all, you should create the directory for the project.

Inside of the **NetCoreSamples** directory, you are going to create a new folder that I have named as "**05.App Settings**".

In the **Terminal bash prompt** of your Mac, execute the next command:
dotnet new

This command will generate two files for us: **project.json** and **Program.cs**.

You are going to open the project with Visual Studio Code.
To do this, click in **File > Open** in Visual Studio Code.
You should select the path that we created before, and click on the **Open** button.
You should see the project opened in Visual Studio Code.

The first action that you should take now, is create a new file named **appsettings.json**.
Remember that to add a new file on Visual Studio Code, you have to click on the **New File** option inside of the Explorer window.

Write the next code in the **appsettings.json** file:

```
{
  "Logging": {
    "LogFile": "logFile.log"
  }
}
```

LogFile represents the name of the file where the application will write information.

Now, open the file **project.json** and modify this file as:

```
{
  "version": "1.0.0-*",
  "buildOptions": {
    "emitEntryPoint": true,
    "copyToOutput": {
      "include": "appsettings.json"
    }
  },


  "dependencies": {
    "Microsoft.Extensions.Configuration": "1.0.0",
    "Microsoft.Extensions.Configuration.Json": "1.0.0",
    "Microsoft.NETCore.App": {
      "type": "platform",
      "version": "1.0.0"
    }
  },

  "frameworks": {
    "netcoreapp1.0": {
      "imports": "dnxcore50"
    }
  }
}
```

Here, you are including the **appsettings.json** file in the output directory.
You can see the dependencies included in the project too.
There are two references to access to the settings that we have created before.

Finally, open and edit the file **Program.cs**.
Modify this file as:

```
namespace NetCoreSamples
{

	using Microsoft.Extensions.Configuration;
	using System;
	using System.IO;

	public class Program
	{

		private static string logFile;

		public static void Main(string[] args)
		{
			// Set up configuration sources.
			var builder = new ConfigurationBuilder()
				.AddJsonFile("appsettings.json", optional: false, reloadOnChange: true);

			var configuration = builder.Build();
```

```
                    logFile = configuration["Logging:LogFile"];

                    Console.WriteLine($"LogFile: {logFile}");

                    var dateTime = DateTime.Now;

                    WriteToLog($"Test at {dateTime}");
                    Console.WriteLine("The text has been registered on the log");

                    Console.ReadLine();
            }



            private static void WriteToLog(string text)
            {
                    var log = File.AppendText(logFile);
                    log.WriteLine(text);
                    log.Dispose();
            }

        }

}
```

Tip

To access to the **LogFile** property in the **appsettings.json** file, you have use this code line:
logFile = configuration["Logging:LogFile"];
However, you can use too this other way to access to this property:
var configurationSection = configuration.GetSection("Logging");
logFile = configurationSection["LogFile"];
And to write the value of logFile you have used:
Console.WriteLine($"LogFile: {logFile}");
But you can use this other code too:
Console.WriteLine($"LogFile: {configuration["Logging:LogFile"]}");

06.Resources

Windows VS Code

In this exercise, you are going to create an application to read a resource file.
You will use the English and Spanish resource data.
By default, English.

First all, you are going to create a new folder that I have named as “**06.Resources**”.

Note: To avoid problems when you compile the application, rename this from 06.Resources to Resources.

The first thing you are going to do, is prepare the **appsettings.json** file.
Write the next code in the **appsettings.json** file:

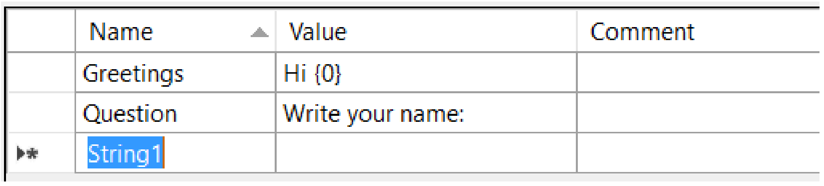
```
{
  "Language":"en-US"
}
```

The application will have to read this value and load the resources from this value.
The default language here will EN, and the application supports in this sample, EN and ES as languages.

You will want to read the settings in the application and copy this file in the output solution, so write the next code for the **project.json** file.

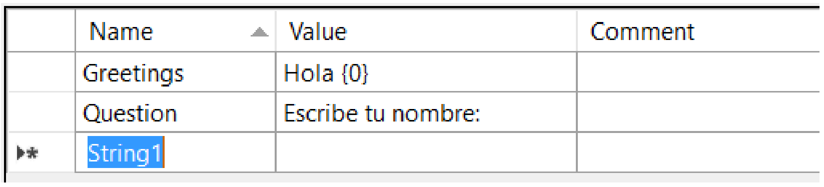
```
{
  "version": "1.0.0-*",
  "buildOptions": {
    "emitEntryPoint": true,
    "copyToOutput": {
      "include": "appsettings.json"
    }
  },
  "dependencies": {},
  "frameworks": {
    "netcoreapp1.0": {
      "dependencies": {
        "Microsoft.Extensions.Configuration": "1.0.0",
        "Microsoft.Extensions.Configuration.Json": "1.0.0",
        "Microsoft.NETCore.App": {
          "type": "platform",
          "version": "1.0.0"
        }
      },
      "imports": "dnxcore50"
    }
  }
}
```

You are going to generate the resource files (resx).
Here I have used Visual Studio 2015 to generate the **Resources.resx** and the **Resources.es-ES.resx** files.
The **Resources.resx** file will have the next content:



	Name	Value	Comment
	Greetings	Hi {0}	
	Question	Write your name:	
▸*	String1		

The **Resources.es-ES.resx** will have the next one:



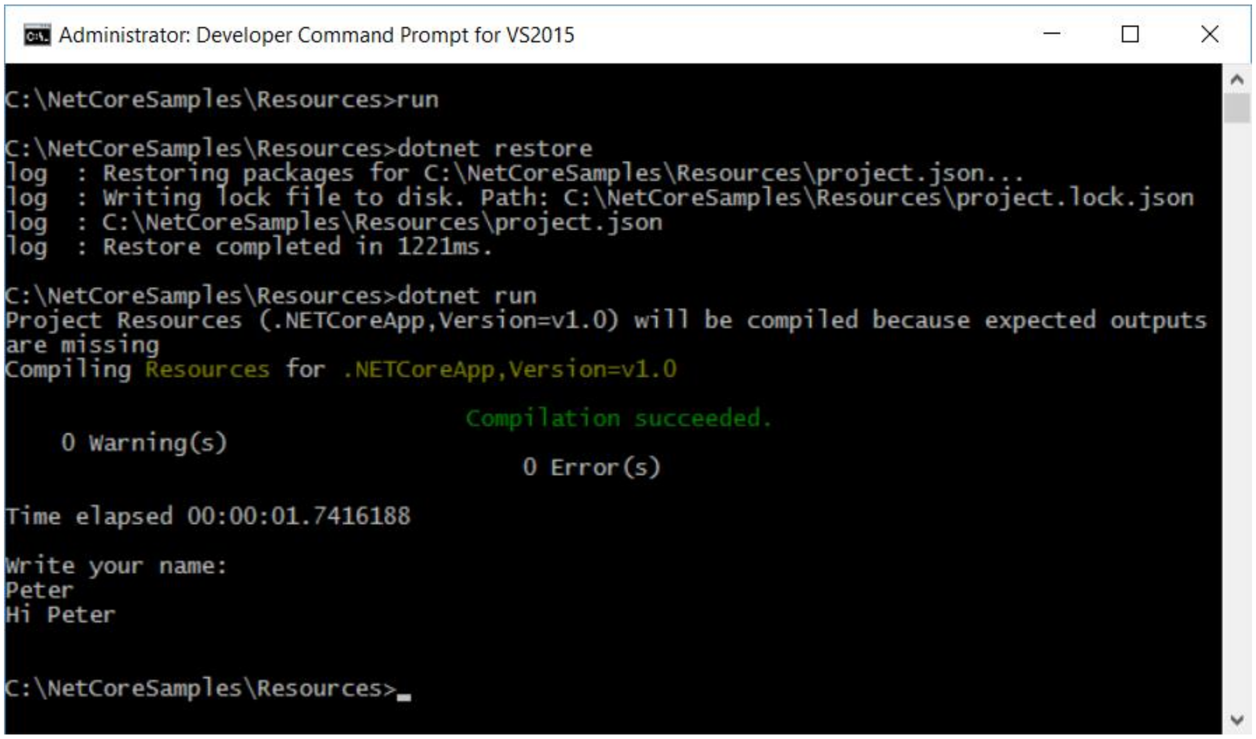
	Name	Value	Comment
	Greetings	Hola {0}	
	Question	Escribe tu nombre:	
▸*	String1		

Finally, create a last code file. The **Program.cs** file.
Add the next code in the file.

```
namespace Resources
{

  using Microsoft.Extensions.Configuration;
  using System;
  using System.Globalization;

  public class Program
  {

    public static void Main(string[] args)
    {
      // Set up configuration sources.
      var builder = new ConfigurationBuilder()
                  .AddJsonFile("appsettings.json", optional: false, reloadOnChange: true);
```

```
            var configuration = builder.Build();

            string language = configuration["Language"];

            var question = Resources.ResourceManager.GetString("Question", new CultureInfo(language));
            Console.WriteLine(question);
            var name = Console.ReadLine();

            var text = Resources.ResourceManager.GetString("Greetings", new CultureInfo(language));
            Console.WriteLine(text, name);
            Console.ReadLine();
        }

    }

}
```

Now, remember rename the directory to **Resources**, and execute the application from the **Developer Command Prompt** for Visual Studio 2015 running the command **run** or **dotnet run**.


Administrator: Developer Command Prompt for VS2015

```
C:\NetCoreSamples\Resources>run

C:\NetCoreSamples\Resources>dotnet restore
log  : Restoring packages for C:\NetCoreSamples\Resources\project.json...
log  : Writing lock file to disk. Path: C:\NetCoreSamples\Resources\project.lock.json
log  : C:\NetCoreSamples\Resources\project.json
log  : Restore completed in 1221ms.

C:\NetCoreSamples\Resources>dotnet run
Project Resources (.NETCoreApp,Version=v1.0) will be compiled because expected outputs
are missing
Compiling Resources for .NETCoreApp,Version=v1.0

                                 Compilation succeeded.
    0 Warning(s)
                                     0 Error(s)

Time elapsed 00:00:01.7416188

Write your name:
Peter
Hi Peter


C:\NetCoreSamples\Resources>_
```

Edit the **appsetting.json** file and change en-US by es-ES for example and execute the application again. You should see the messages in Spanish loading the correct resources strings.

07.Database

Windows VS Code

In this exercise, you are going to access to a Sqlite database to create, select, edit and delete data information using Dapper as Micro ORM.

The database will be very simple.
We are preparing a party and we want to maintain a list of our friends. ☺

Well, you are going to create a new folder that I have named as “**07.Database**”.
As you have seen until here, you should open a **Developer Command Prompt** for Visual Studio 2015 to set the prompt in the directory that we created before.

From Visual Studio Code, you are going to click on the menu **File > Open Folder**.
Go to the path where is your project folder and select it doing click on the **Select Folder** button of this window.

The first thing you are going to do, is prepare the **appsettings.json** file.
Write the next code in the **appsettings.json** file:

```
{
  "ConnectionStrings": {
    "DatabaseName": "agenda.db"
  }
}
```

In the settings, you are writing the name of your Sqlite database (**agenda.db**).

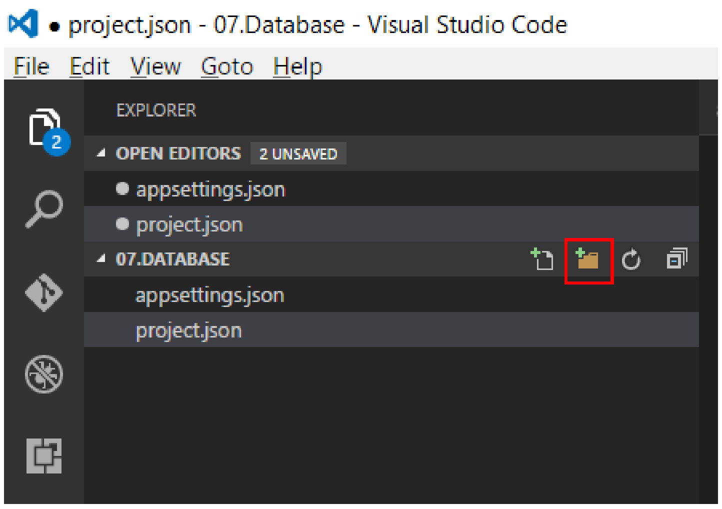
Now you are going to create the **project.json** file with the references that you will use in this project (Sqlite, Dapper and the libraries to access to the settings that you have seen in other samples).

The code of **project.json** is:

```
{
  "version": "1.0.0-*",
  "buildOptions": {
    "emitEntryPoint": true,
    "copyToOutput": {
```

```
    "include": "appsettings.json"
  }
 },

 "dependencies": {
  "Microsoft.Extensions.Configuration": "1.0.0",
  "Microsoft.Extensions.Configuration.Json": "1.0.0",
  "Dapper": "1.50.0",
  "Microsoft.Data.Sqlite": "1.0.0",
  "Microsoft.NETCore.App": {
    "type": "platform",
    "version": "1.0.0"
  }
 },

 "frameworks": {
  "netcoreapp1.0": {
    "imports": "dnxcore50"
  }
 }
}
```

Now, create two folders in the project.
A **Model** folder and a **Repository** folder.
You can do this doing click on the **New Folder** button:



Click in the **Model** folder you have created and add a new file.
Name this new file as **MyFriend.cs**.
Write the next code for this class:

```
namespace Database.Model
{

    using System;

    public class MyFriend
    {

        public int Id { get; set; }
        public string Name { get; set; }
        public string PhoneNumber { get; set; }
        public DateTime CreatedAt { get; set; }
        public DateTime ModifiedAt { get; set; }

    }

}
```

Now, click in the **Repository** folder and add a new file with the name **AgendaRepository.cs**.
Add the next code for this new class:

```
namespace Database.Repository
{

    using Dapper;
    using Database.Model;
    using Microsoft.Data.Sqlite;
    using System;
    using System.Collections.Generic;
    using System.Data;
    using System.Linq;

    public class AgendaRepository
    {

```

```
    private readonly string databaseName = String.Empty;
    private IDbConnection connection = null;

    public AgendaRepository(string database)
    {
      this.databaseName = database;

      this.connection = new SqliteConnection("" + new SqliteConnectionStringBuilder
      {
        DataSource = this.databaseName
      });

      this.connection.Open();

      // Bootstrap database
      var response = CreateDatabase();
    }

    private bool CreateDatabase()
    {
      try
      {
        var sql = "CREATE TABLE if not exists MyFriends (Id integer primary key, Name varchar(50) not null, PhoneNumber
varchar(16) null, CreatedAt datetime default current_timestamp, ModifiedAt datetime null)";
        var command = new SqliteCommand(sql, this.connection as SqliteConnection);
        command.ExecuteNonQuery();

        return true;
      }
      catch (Exception)
      {
        return false;
      }
    }

    public IEnumerable<MyFriend> GetAll()
    {
      try
```

```
        {
            string sql = "SELECT Id, Name, PhoneNumber, CreatedAt, ModifiedAt FROM MyFriends";
            return this.connection.Query<MyFriend>(sql).ToList();
        }
        catch (Exception ex)
        {
            return null;
        }
    }

    public MyFriend GetBy(int id)
    {
        try
        {
            string sql = "SELECT Id, Name, PhoneNumber, CreatedAt, ModifiedAt FROM MyFriends WHERE Id = @Id";
            return this.connection.Query<MyFriend>(sql, new { Id = id }).FirstOrDefault();
        }
        catch (Exception ex)
        {
            return null;
        }
    }

    public bool Create(MyFriend myFriend)
    {
        try
        {
            string sql = "INSERT INTO MyFriends (Name, PhoneNumber) VALUES (@Name, @PhoneNumber)";
            this.connection.Execute(sql, myFriend);

            return true;
        }
        catch (Exception ex)
        {
            return false;
        }
    }
```

```
        public bool Modify(MyFriend myFriend)
        {
            try
            {
                string sql = "UPDATE MyFriends SET PhoneNumber = @PhoneNumber, ModifiedAt = @ModifiedAt WHERE Id = @Id";
                this.connection.Execute(sql, myFriend);

                return true;
            }
            catch (Exception ex)
            {
                return false;
            }
        }

        public bool DeleteBy(int id)
        {
            try
            {
                string sql = "DELETE FROM MyFriends WHERE Id = @Id";
                this.connection.Execute(sql, new { Id = id });

                return true;
            }
            catch (Exception ex)
            {
                return false;
            }
        }

    }

}
```

Finally, add a new file in the root of the project, and name this file as **Program.cs**.
Write the next code for this class:

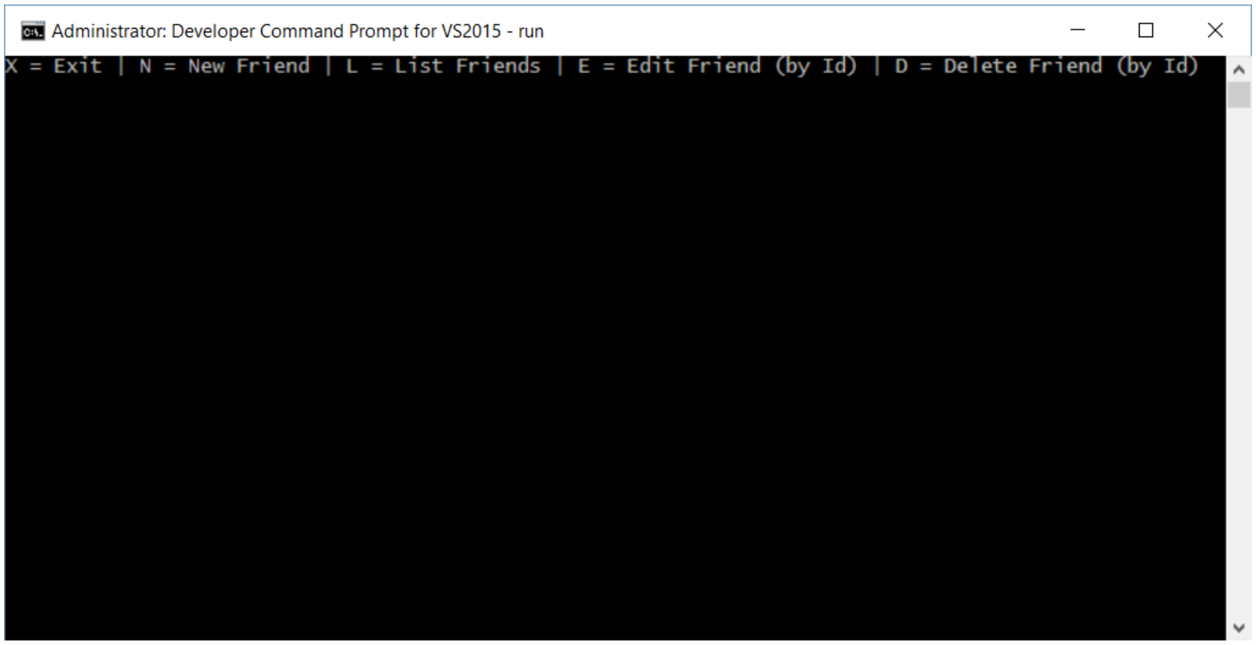
```
namespace Database
```

```
{

    using Database.Model;
    using Database.Repository;
    using Microsoft.Extensions.Configuration;
    using System;
    using System.Linq;

    public class Program
    {

        private const string DATABASE_KEY = "DatabaseName";
        private static AgendaRepository repository = null;

        public static void Main(string[] args)
        {
            // Enable to app to read json setting files
            var builder = new ConfigurationBuilder()
                .AddJsonFile("appsettings.json", optional: false, reloadOnChange: true);

            var configuration = builder.Build();

            // Get the connection string
            var databaseName = configuration.GetConnectionString(DATABASE_KEY);

            // Get the repository
            repository = new AgendaRepository(databaseName);

            // Execute the logic of the program
            Execute();

            Console.Clear();
            Console.WriteLine("Thanks for use the program... :-)");
            Console.ReadLine();
        }

        private static void Execute()
        {
```

```
        var action = String.Empty;

        while (action != "X")
        {
            Console.Clear();
            Console.WriteLine("X = Exit | N = New Friend | L = List Friends | E = Edit Friend (by Id) | D = Delete Friend (by Id)");
            action = Console.ReadLine();

            Console.Clear();

            if (action == "L")
            {
                ListFriends();
            }

            if (action == "N")
            {
                AddNewFriend();
            }

            if (action == "E")
            {
                EditFriend();
            }

            if (action == "D")
            {
                DeleteFriend();
            }

        }
    }

    private static void ListFriends()
    {
        var entities = repository.GetAll().ToList();

        if (entities.Count > 0)
        {
            Console.WriteLine("Id \t Name \t\t PhoneNumber \t CreatedAt \t\t ModifiedAt");

            foreach (var entity in entities)
            {
                Console.WriteLine($"{entity.Id}    \t    {entity.Name}    \t\t    {entity.PhoneNumber}    \t    {entity.CreatedAt}    \t
{entity.ModifiedAt}");
            }
        }
        else
        {
            Console.WriteLine("You don't have friends! :-P");
        }

        Console.WriteLine("");
        Console.WriteLine("Press any key to continue...");
        Console.ReadLine();
    }

    private static void AddNewFriend()
    {
        var myFriend = new MyFriend();

        var data = String.Empty;
        while (String.IsNullOrEmpty(data))
        {
            Console.Clear();
            Console.WriteLine("Write the Name of your friend:");
            data = Console.ReadLine();
            myFriend.Name = data;
        }

        Console.WriteLine($"Write the Phone Number of {myFriend.Name}:");
        data = Console.ReadLine();
        myFriend.PhoneNumber = data;

        var response = repository.Create(myFriend);
        if (response)
        {
            Console.WriteLine("Your friend has been added to the list");
        }
        else
        {
            Console.WriteLine("An error ocurred while your friend was added to the list");
        }

        Console.WriteLine("");
        Console.WriteLine("Press any key to continue...");
        Console.ReadLine();
    }

    private static void EditFriend()
    {
        var myFriend = new MyFriend();

        var data = String.Empty;
        while (String.IsNullOrEmpty(data))
        {
            Console.Clear();
            Console.WriteLine("Write the Id of your friend:");
            data = Console.ReadLine();
            myFriend.Id = Convert.ToInt32(data);
        }

        myFriend = repository.GetBy(myFriend.Id);

        Console.WriteLine($"Write the new Phone Number of {myFriend.Name}:");
        data = Console.ReadLine();
        myFriend.PhoneNumber = data;

        myFriend.ModifiedAt = DateTime.UtcNow;
        var response = repository.Modify(myFriend);
        if (response)
        {
            Console.WriteLine("Your friend has been modified in the list");
        }
    else
    {
        Console.WriteLine("An error ocurred while your friend was modified in the list");
    }

    Console.WriteLine("");
    Console.WriteLine("Press any key to continue...");
    Console.ReadLine();
}

private static void DeleteFriend()
{
    var id = 0;

    var data = String.Empty;
    while (String.IsNullOrEmpty(data))
    {
        Console.Clear();
        Console.WriteLine("Write the Id of your friend:");
        data = Console.ReadLine();
        id = Convert.ToInt32(data);
    }

    var response = repository.DeleteBy(id);

    if (response)
    {
        Console.WriteLine("Your friend has been deleted from the list");
    }
    else
    {
        Console.WriteLine("An error ocurred while your friend was deleting from the list");
    }

    Console.WriteLine("");
    Console.WriteLine("Press any key to continue...");
    Console.ReadLine();
}
```

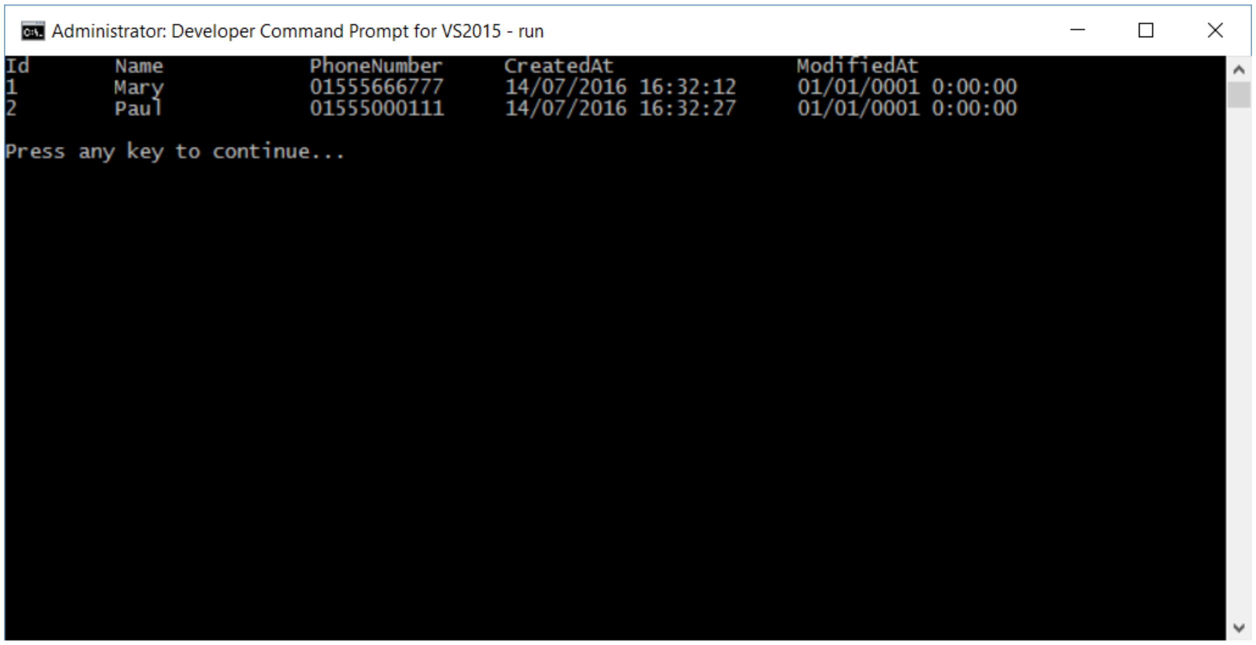
```
    }

}
```

Now, open the **Developer Command Prompt** for Visual Studio 2015 and execute the application.



Administrator: Developer Command Prompt for VS2015 - run

```
X = Exit | N = New Friend | L = List Friends | E = Edit Friend (by Id) | D = Delete Friend (by Id)
```



Administrator: Developer Command Prompt for VS2015 - run

```
Id        Name            PhoneNumber     CreatedAt              ModifiedAt
1         Mary            01555666777     14/07/2016 16:32:12    01/01/0001 0:00:00
2         Paul            01555000111     14/07/2016 16:32:27    01/01/0001 0:00:00

Press any key to continue...
```

Install

If you want to browse your Sqlite database, you can use the DB Browser for Sqlite that you will find in the next link:
http://sqlitebrowser.org/

Extras

If you are thinking now in deep more with .NET Core, I suggest you to see the .NET Core Api Reference.
Here, you will find the namespaces of the .NET Core Class Library.
Is a complete API reference to search about some functionality.
https://docs.microsoft.com/en-us/dotnet/core/api/index

The .NET Core tooling is going to move from project.json to MSBuild based projects.
In this ebook we have seen how to do all samples with project.json, but you can use MSBuild for .NET Core projects today.
For more information, read the information about it here:
https://docs.microsoft.com/en-us/dotnet/articles/core/tutorials/target-dotnetcore-with-msbuild

GitHub Repo with the samples:
https://github.com/J0rgeSerran0/NetCoreSamples

GitHub Information about the author of this ebook:
https://j0rgeserran0.github.io/

GitHub MicroPosts or Pills (in Spanish)
https://github.com/J0rgeSerran0/Blog/blob/master/index.md

J0rgeSerran0

HAPPY CODING!