.NET Core

tutorialspoint

SIMPLY EASY LEARNING

www.tutorialspoint.com

https://www.facebook.com/tutorialspointindia

https://twitter.com/tutorialspoint

About the Tutorial

.NET Core is the latest general purpose development platform maintained by Microsoft. It works across different platforms and has been redesigned in a way that makes .NET fast, flexible and modern.

.NET Core happens to be one of the major contributions by Microsoft. Developers can now build Android, iOS, Linux, Mac, and Windows applications with .NET, all in Open Source.

Audience

This tutorial is designed for software programmers who want to learn the basics of .NET Core.

Prerequisites

You should have a basic understanding of Computer Programming terminologies. A basic understanding of any of the programming languages is a plus.

Disclaimer & Copyright

Table of Contents

1. .NET Core – Overview

.NET Core is the latest general purpose development platform maintained by Microsoft. It works across different platforms and has been redesigned in a way that makes .NET fast, flexible and modern. This happens to be one of the major contributions by Microsoft. Developers can now build Android, iOS, Linux, Mac, and Windows applications with .NET, all in Open Source.

In this tutorial, we will cover .NET Core and a few new innovations including the .NET Framework updates, .NET Standard, and Universal Windows Platform updates, etc.

Characteristics of .NET Core

The following are the major characteristics of .NET Core:

Open source

- .NET Core is an open source implementation, using MIT and Apache 2 licenses.
- .NET Core is a .NET Foundation project and is available on GitHub.
- As an open source project, it promotes a more transparent development process and promotes an active and engaged community.

Cross-platform

- Application implemented in .NET Core can be run and its code can be reused regardless of your platform target.
- It currently supports three main operating systems (OS):
    - Windows
    - Linux
    - MacOS
- The supported Operating Systems (OS), CPUs and application scenarios will grow over time, provided by Microsoft, other companies, and individuals.

Flexible deployment

- There can be two types of deployments for .NET Core applications:
    - Framework-dependent deployment
    - Self-contained deployment

- With framework-dependent deployment, your app depends on a system-wide version of .NET Core on which your app and third-party dependencies are installed.
- With self-contained deployment, the .NET Core version used to build your application is also deployed along with your app and third-party dependencies and can run side-by-side with other versions.

Command-line tools

- All product scenarios can be exercised at the command-line.

Compatible

- .NET Core is compatible with .NET Framework, Xamarin and Mono, via the .NET Standard Library.

Modular

- .NET Core is released through NuGet in smaller assembly packages.
- .NET Framework is one large assembly that contains most of the core functionalities.
- .NET Core is made available as smaller feature-centric packages.
- This modular approach enables the developers to optimize their app by including just those NuGet packages which they need in their app.
- The benefits of a smaller app surface area include tighter security, reduced servicing, improved performance, and decreased costs in a pay-for-what-you-use model.

The .NET Core Platform

.NET Core Platform contains the following main parts:

- **.NET Runtime**: It provides a type system, assembly loading, a garbage collector, native interop and other basic services.
- **Fundamental Libraries**: A set of framework libraries, which provide primitive data types, app composition types and fundamental utilities.
- **SDK & Compiler**: A set of SDK tools and language compilers that enable the base developer experience, available in the .NET Core SDK.
- **'dotnet' app host**: it is used to launch .NET Core apps. It selects the runtime and hosts the runtime, provides an assembly loading policy and launches the app. The same host is also used to launch SDK tools in much the same way.

2. .NET Core – Prerequisites

In this chapter, we will discuss the various dependencies that you need to deploy and run. These include the .NET Core applications on Windows machines that are developed using Visual Studio.

Supported Windows Versions

.NET Core is supported on the following versions of Windows:

- Windows 7 SP1
- Windows 8.1
- Windows 10
- Windows Server 2008 R2 SP1 (Full Server or Server Core)
- Windows Server 2012 SP1 (Full Server or Server Core)
- Windows Server 2012 R2 SP1 (Full Server or Server Core)
- Windows Server 2016 (Full Server, Server Core or Nano Server)

Dependencies

- If you are running your .NET Core application on Windows versions earlier than Windows 10 and Windows Server 2016, then it will also require the Visual C++ Redistributable.
- This dependency is automatically installed for you if you use the .NET Core installer.
- You need to manually install the Visual C++ Redistributable for Visual Studio 2015 if you are installing .NET Core via the installer script or deploying a self-contained .NET Core application.
- For Windows 7 and Windows Server 2008 machines, you need to make sure that your Windows installation is up-to-date and also includes hotfix KB2533623 installed through Windows Update.

Prerequisites with Visual Studio

- To develop .NET Core applications using the .NET Core SDK, you can use any editor of your choice.
- However, if you want to develop .NET Core applications on Windows using Visual Studio, you can use the following two versions:
  - Visual Studio 2015
  - Visual Studio 2017 RC

- Projects created with Visual Studio 2015 will be project.json-based by default while projects created with Visual Studio 2017 RC will always be MSBuild-based.

3. .NET Core – Environment Setup

In this chapter, we will discuss the Environment Setup of .NET Core. It is a significant redesign of the .NET Framework. To use .NET Core in your application, there are two versions you can use:

- Visual Studio 2015
- Visual Studio 2017 RC

Visual Studio 2015

To use Visual Studio 2015, you must have installed the following;

- Microsoft Visual Studio 2015 Update 3
- Microsoft .NET Core 1.0.1 - VS 2015 Tooling Preview 2

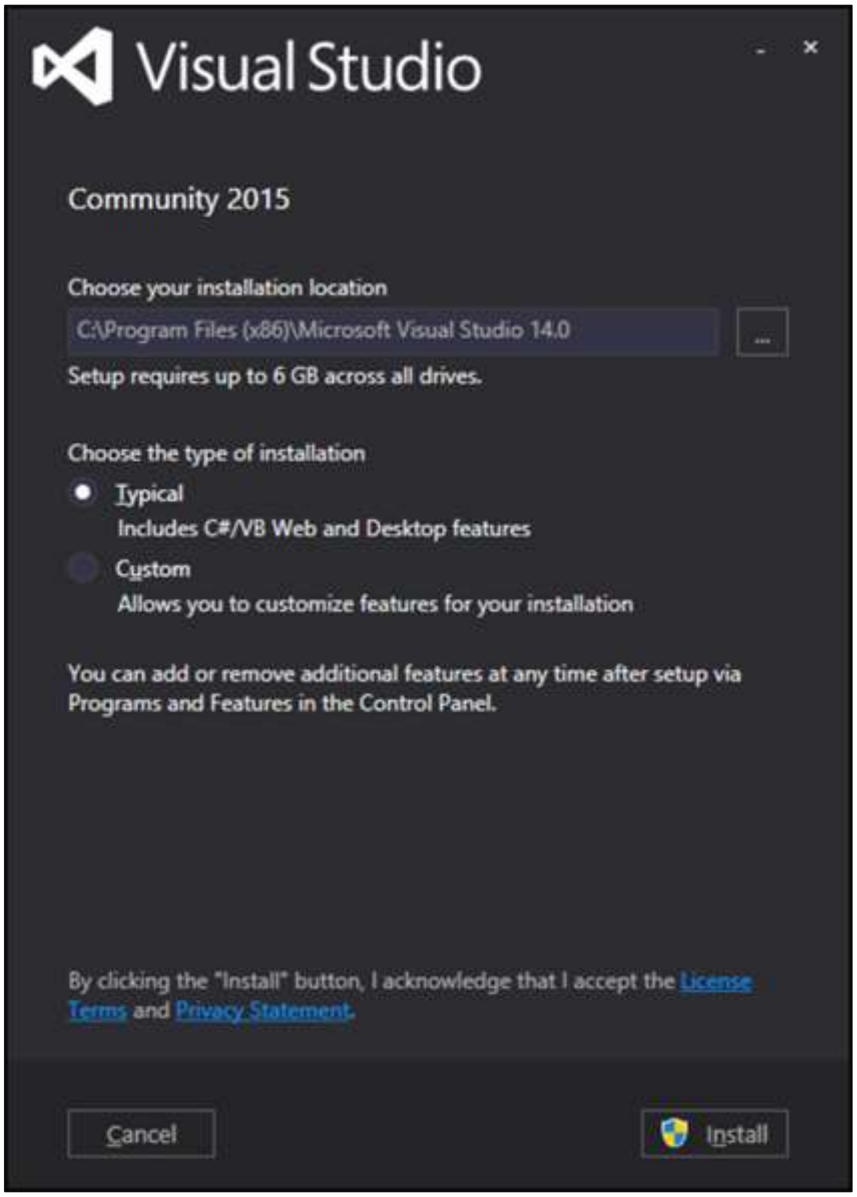
Microsoft provides a free version of visual studio which also contains the SQL Server and can be downloaded from https://www.visualstudio.com/en-us/downloads/download-visual-studio-vs.aspx and Microsoft .NET Core 1.0.1 - VS 2015 Tooling Preview 2 can be downloaded from https://go.microsoft.com/fwlink/?LinkId=817245.

You can also follow the installation guidelines on the following Url https://www.microsoft.com/net/core/#windowsvs2015.

Installation of Visual Studio 2015

Follow these steps to install Visual Studio 2015:

Step 1: Once the downloading completes, then run the installer. The following dialog box will be displayed.



Visual Studio

Community 2015

Choose your installation location

C:\Program Files (x86)\Microsoft Visual Studio 14.0

Setup requires up to 6 GB across all drives.

Choose the type of installation

● Typical

Includes C#/VB Web and Desktop features

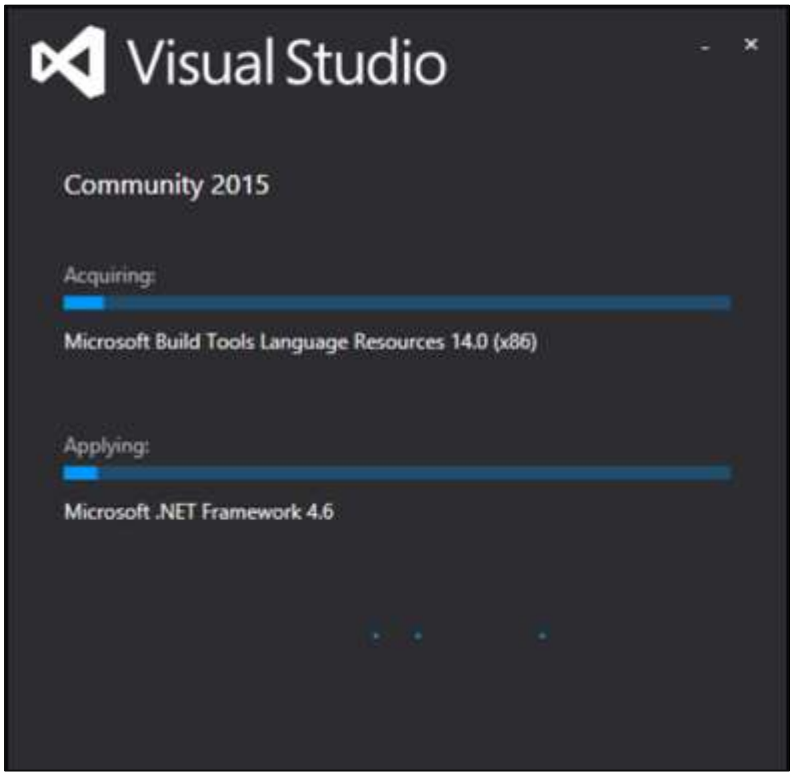
○ Custom

Allows you to customize features for your installation

You can add or remove additional features at any time after setup via Programs and Features in the Control Panel.

By clicking the "Install" button, I acknowledge that I accept the License Terms and Privacy Statement.

Step 2: Click **Install** to start the installation process.



Visual Studio

Community 2015

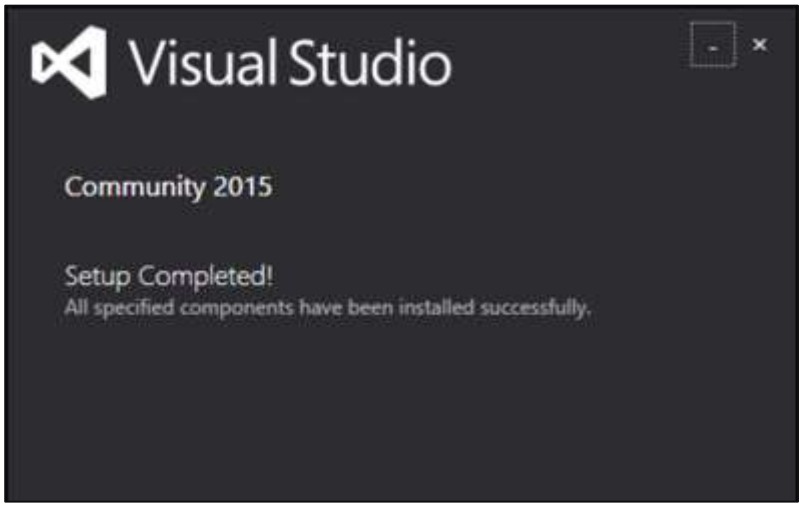
Acquiring:

Microsoft Build Tools Language Resources 14.0 (x86)

Applying:

Microsoft .NET Framework 4.6

Step 3: Once the installation completes, you will see the following dialog box.



Visual Studio

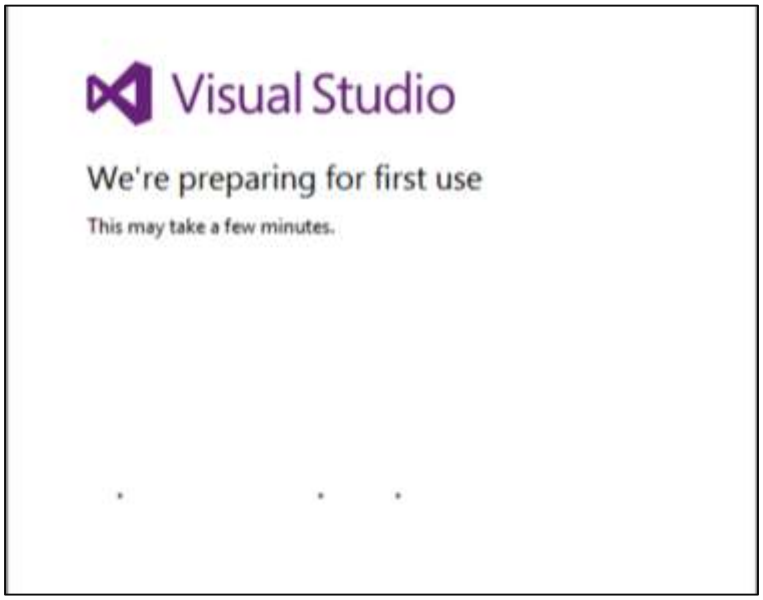
Community 2015

Setup Completed!

All specified components have been installed successfully.

Step 4: Close this dialog and restart your computer if required.

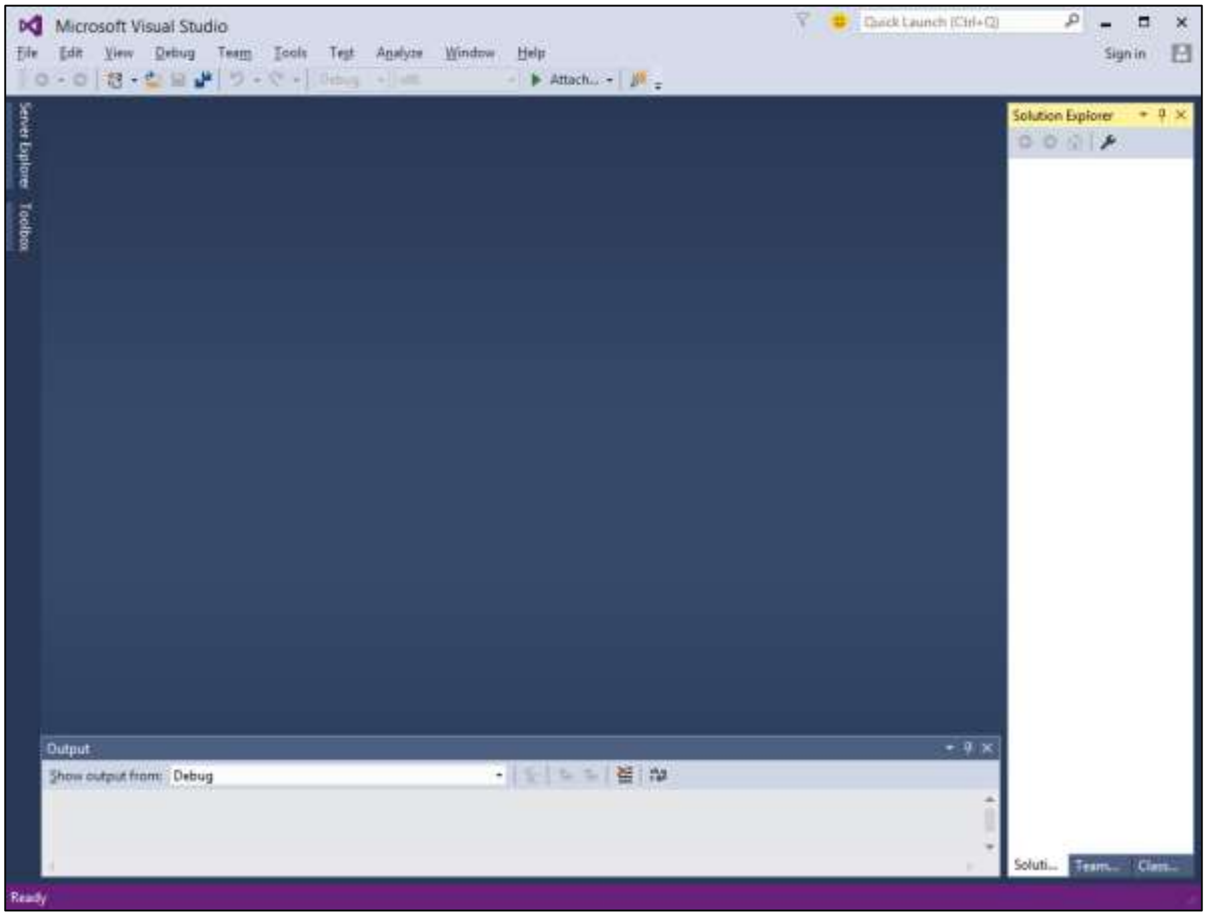
Step 5: Open Visual Studio from the Start Menu; you will receive the following dialog box. It may take a few minutes to load and finally be used for the first time.



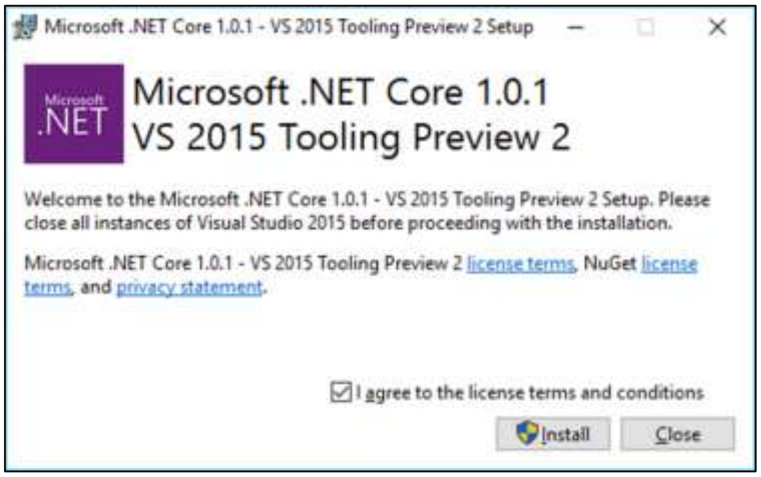
We're preparing for first use

This may take a few minutes.

Step 6: Once it is loaded, you will see the following screen.



Step 7: Once Visual Studio installation is finished, then close Visual Studio and launch Microsoft .NET Core - VS 2015 Tooling Preview 2.


Microsoft .NET Core 1.0.1 - VS 2015 Tooling Preview 2 Setup

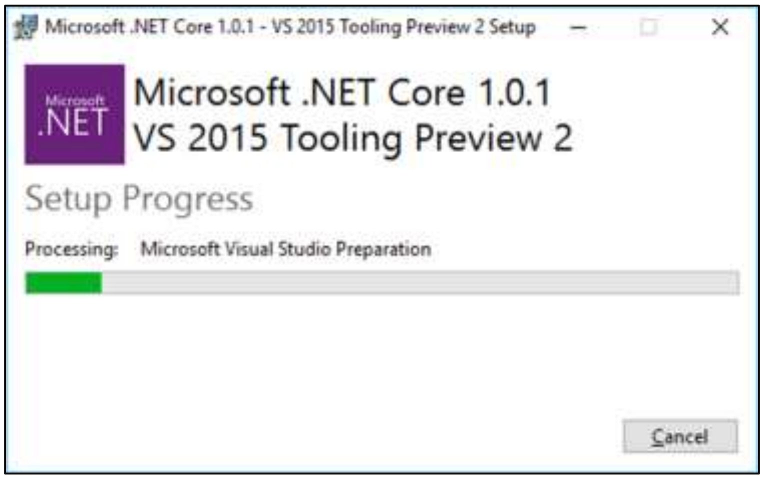
Microsoft .NET Core 1.0.1
VS 2015 Tooling Preview 2

Welcome to the Microsoft .NET Core 1.0.1 - VS 2015 Tooling Preview 2 Setup. Please close all instances of Visual Studio 2015 before proceeding with the installation.

Microsoft .NET Core 1.0.1 - VS 2015 Tooling Preview 2 license terms, NuGet license terms, and privacy statement.

☑ I agree to the license terms and conditions

Step 8: Check the checkbox and click Install.

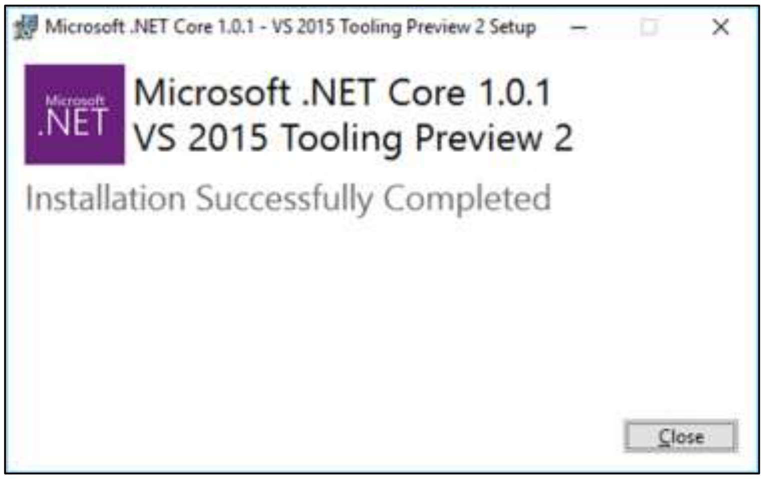

Microsoft .NET Core 1.0.1 - VS 2015 Tooling Preview 2 Setup

Microsoft .NET Core 1.0.1
VS 2015 Tooling Preview 2

Setup Progress

Processing: Microsoft Visual Studio Preparation

Step 9: Once the installation completes, you will see the following dialog box.


Microsoft .NET Core 1.0.1 - VS 2015 Tooling Preview 2 Setup

Microsoft .NET Core 1.0.1
VS 2015 Tooling Preview 2

Installation Successfully Completed

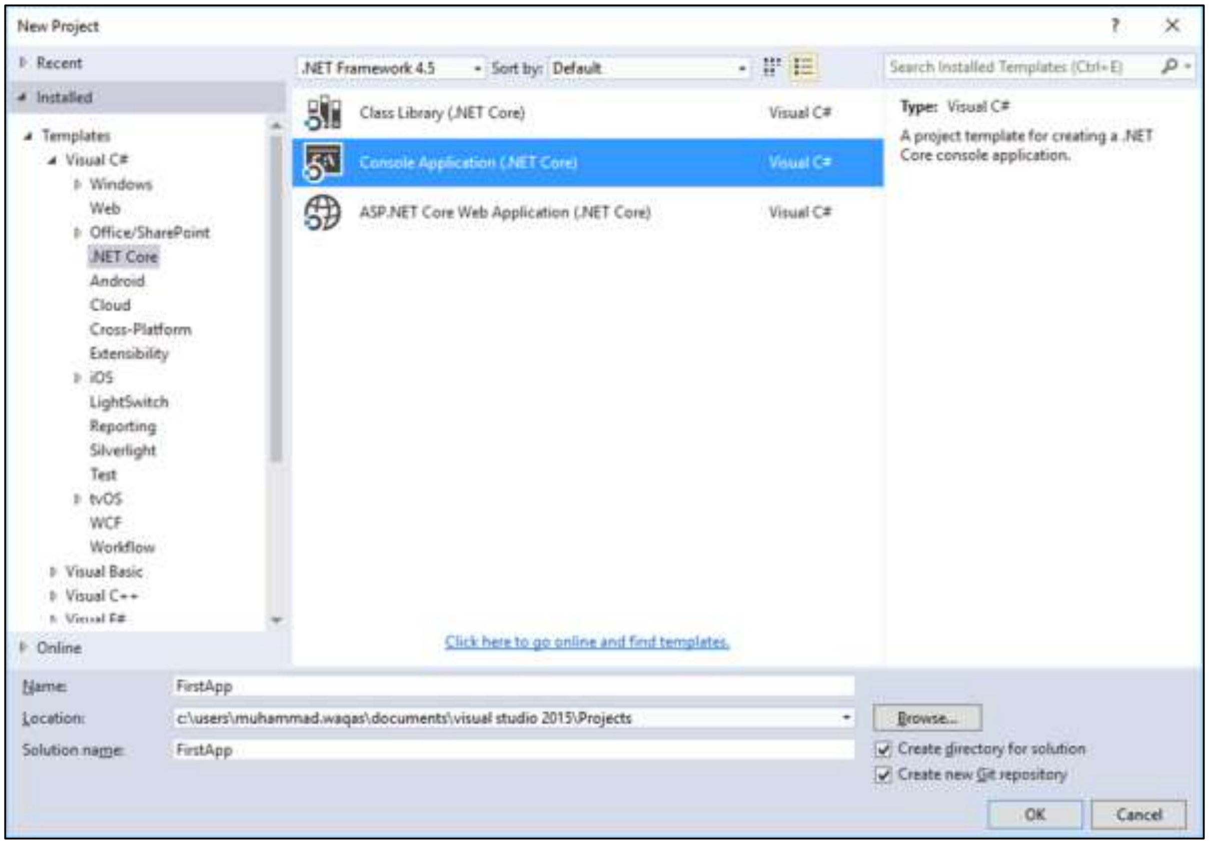
You are now ready to start your application using .NET Core.

Visual Studio 2017

In this tutorial, we will be using Visual Studio 2015, but if you want to use Visual Studio 2017, an experimental release of .NET Core tools for Visual Studio is included in Visual Studio 2017 RC and you can see the installation guidelines here https://www.microsoft.com/net/core/#windowsvs2017.

4. .NET Core – Getting Started

Visual Studio 2015 provides a full-featured development environment for developing .NET Core applications. In this chapter, we will be creating a new project inside Visual Studio. Once you have installed the Visual Studio 2015 tooling, you can start building a new .NET Core Application.



Type: Visual C#

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

Name: FirstApp

Location: c:\users\muhammad.waqas\documents\visual studio 2015\Projects

Solution name: FirstApp

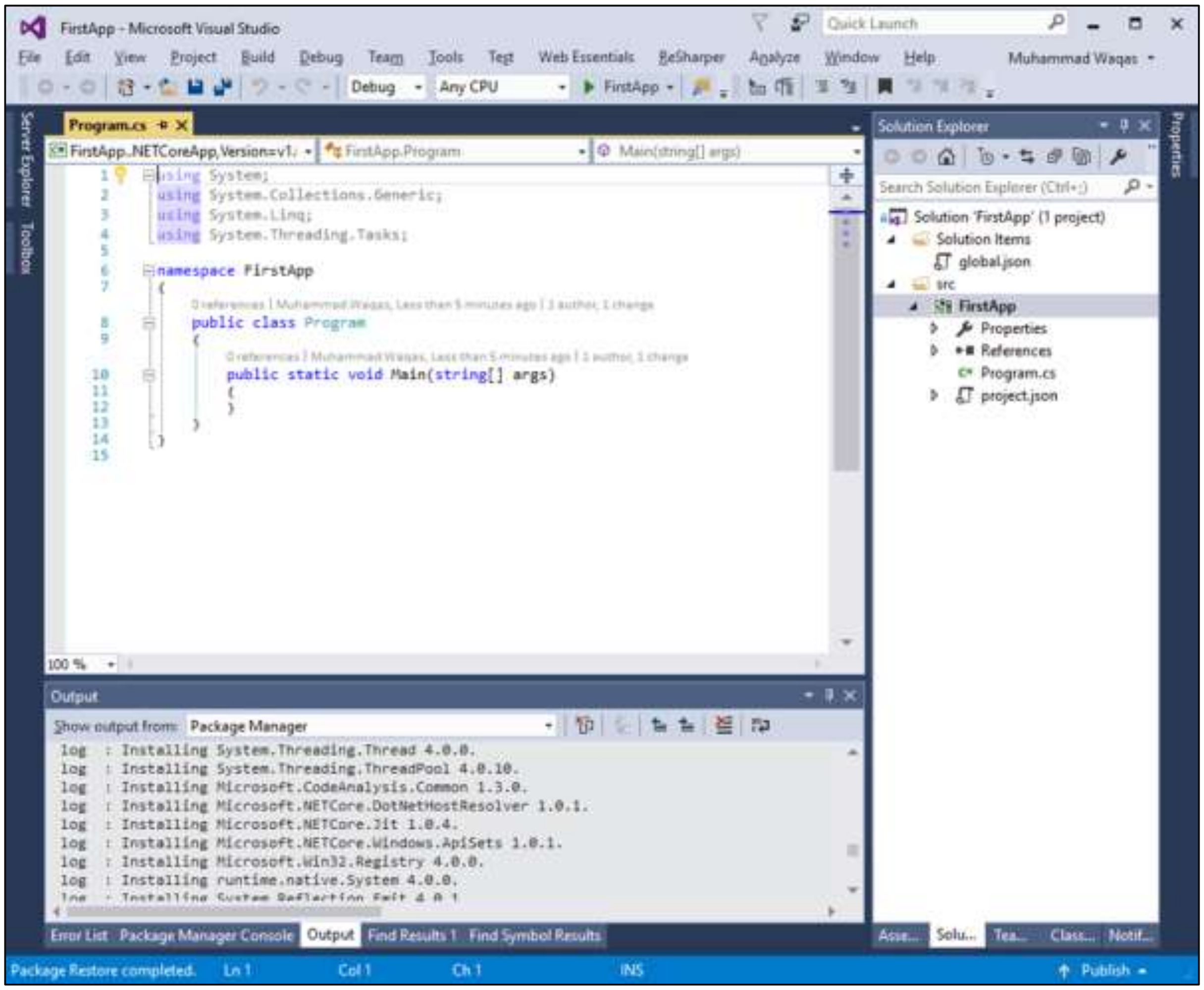
☑ Create directory for solution

☑ Create new Git repository

In the **New Project** dialog box, in the Templates list, expand the Visual C# node and select .NET Core and you should see the following three new project templates:

- Class Library (.NET Core)
- Console Application (.NET Core)
- ASP.NET Core Web Application (.NET Core)

In the middle pane on the New Project dialog box, select Console Application (.NET Core) and name it "FirstApp", then click OK.



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace FirstApp
{
    0 references | Muhammad Waqas, Less than 5 minutes ago | 1 author, 1 change
    public class Program
    {
        0 references | Muhammad Waqas, Less than 5 minutes ago | 1 author, 1 change
        public static void Main(string[] args)
        {
        }
    }
}
```

```
log  : Installing System.Threading.Thread 4.0.0.
log  : Installing System.Threading.ThreadPool 4.0.10.
log  : Installing Microsoft.CodeAnalysis.Common 1.3.0.
log  : Installing Microsoft.NETCore.DotNetHostResolver 1.0.1.
log  : Installing Microsoft.NETCore.Jit 1.0.4.
log  : Installing Microsoft.NETCore.Windows.ApiSets 1.0.1.
log  : Installing Microsoft.Win32.Registry 4.0.0.
log  : Installing runtime.native.System 4.0.0.
log  : Installing System.Reflection.Emit 4.0.1
```

Package Restore completed.

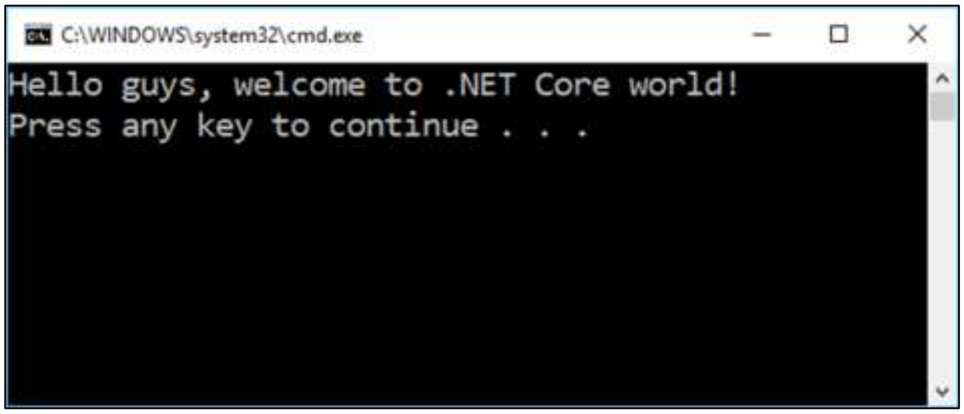
Visual Studio will open the newly created project, and you will see in the Solution Explorer window all of the files that are in this project.

To test that .NET core console application is working, let us add the following line.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;


namespace FirstApp
{
    public class Program
    {
        public static void Main(string[] args)
        {
            Console.WriteLine("Hello guys, welcome to .NET Core world!");
        }
    }
}
```

Now, run the application. You should see the following output.



```
Hello guys, welcome to .NET Core world!
Press any key to continue . . .
```

5. .NET Core – Numerics

.NET Core supports the standard numeric integral and floating-point primitives. It also supports the following types:

- System.Numerics.BigInteger which is an integral type with no upper or lower bound.
- System.Numerics.Complex is a type that represents complex numbers.
- A set of Single Instruction Multiple Data (SIMD)-enabled vector types in the System.Numerics namespace.

Integral types

.NET Core supports both signed and unsigned integers of different ranges from one byte to eight bytes in length. All integers are value types.

The following table represents the integral types and their size;

Type	Signed/ Unsigned	Size (bytes)	Minimum Value	Maximum Value
Byte	Unsigned	1	0	255
Int16	Signed	2	−32,768	32,767
Int32	Signed	4	−2,147,483,648	2,147,483,647
Int64	Signed	8	−9,223,372,036,854,775,808	9,223,372,036,854,775,807
SByte	Signed	1	-128	127
UInt16	Unsigned	2	0	65,535
UInt32	Unsigned	4	0	4,294,967,295
UInt64	Unsigned	8	0	18,446,744,073,709,551,615

Each integral type supports a standard set of arithmetic, comparison, equality, explicit conversion, and implicit conversion operators.

You can also work with the individual bits in an integer value by using the System.BitConverter class.

Floating-point types

.NET Core includes three primitive floating point types, which are shown in the following table.

Type	Size (bytes)	Minimum Value	Maximum Value

Double	8	−1.79769313486232e308	1.79769313486232e308
Single	4	−3.402823e38	3.402823e38
Decimal	16	−79,228,162,514,264,337,593,543,950,335	79,228,162,514,264,337,593,543,950,335

- Each floating-point type supports a standard set of arithmetic, comparison, equality, explicit conversion, and implicit conversion operators.
- You can also work with the individual bits in Double and Single values by using the BitConverter class.
- The Decimal structure has its own methods, Decimal.GetBits and Decimal.Decimal(Int32()), for working with a decimal value's individual bits, as well as its own set of methods for performing some additional mathematical operations.

BigInteger

- System.Numerics.BigInteger is an immutable type that represents an arbitrarily large integer whose value in theory has no upper or lower bounds.
- The methods of the BigInteger type is closely parallel to those of the other integral types.

Complex

- The System.Numerics.Complex type represents a complex number, i.e., a number with a real number part and an imaginary number part.
- It supports a standard set of arithmetic, comparison, equality, explicit conversion, and implicit conversion operators, as well as mathematical, algebraic, and trigonometric methods.

SIMD

- The Numerics namespace includes a set of SIMD-enabled vector types for .NET Core.
- SIMD allows some operations to be parallelized at the hardware level, which results in huge performance improvements in mathematical, scientific, and graphics apps that perform computations over vectors.
- The SIMD-enabled vector types in .NET Core include the following:
    - System.Numerics.Vector2, System.Numerics.Vector3, and System.Numerics.Vector4 types, which are 2, 3, and 4-dimensional vectors of type Single.
    - The Vector<T> structure that allows you to create a vector of any primitive numeric type. The primitive numeric types include all numeric types in the System namespace except for Decimal.

  - Two matrix types, System.Numerics.Matrix3x2, which represents a 3x2 matrix; and System.Numerics.Matrix4x4, which represents a 4x4 matrix.
  - The System.Numerics.Plane type, which represents a three-dimensional plane, and the System.Numerics.Quaternion type, which represents a vector that is used to encode three-dimensional physical rotations.

6. .NET Core – Garbage Collection

In this chapter, we will cover the concept of Garbage collection which is one of most important features of the .NET managed code platform. The garbage collector (GC) manages the allocation and release of memory. The garbage collector serves as an automatic memory manager.

- You do not need to know how to allocate and release memory or manage the lifetime of the objects that use that memory.
- An allocation is made any time you declare an object with a "new" keyword or a value type is boxed. Allocations are typically very fast.
- When there isn't enough memory to allocate an object, the GC must collect and dispose of garbage memory to make memory available for new allocations.
- This process is known as **garbage collection**.

Advantages of Garbage Collection

Garbage Collection provides the following benefits:

- You don't need to free memory manually while developing your application.
- It also allocates objects on the managed heap efficiently.
- When objects are no longer used then it will reclaim those objects by clearing their memory, and keeps the memory available for future allocations.
- Managed objects automatically get clean content to start with, so their constructors do not have to initialize every data field.
- It also provides memory safety by making sure that an object cannot use the content of another object.

Conditions for Garbage Collection

Garbage collection occurs when one of the following conditions is true.

- The system has low physical memory.
- The memory that is used by allocated objects on the managed heap surpasses an acceptable threshold. This threshold is continuously adjusted as the process runs.
- The **GC.Collect** method is called and in almost all cases, you do not have to call this method, because the garbage collector runs continuously. This method is primarily used for unique situations and testing.

Generations

The .NET Garbage Collector has 3 generations and each generation has its own heap that that is used for the storage of allocated objects. There is a basic principle that most objects are either short-lived or long-lived.

Generation First (0)

- In Generation 0, objects are first allocated.
- In this generation, objects often don't live past the first generation, since they are no longer in use (out of scope) by the time the next garbage collection occurs.
- Generation 0 is quick to collect because its associated heap is small.

Generation Second (1)

- In Generation 1, objects have a second chance space.
- Objects that are short-lived but survive the generation 0 collection (often based on coincidental timing) go to generation 1.
- Generation 1 collections are also quick because its associated heap is also small.
- The first two heaps remain small because objects are either collected or promoted to the next generation heap.

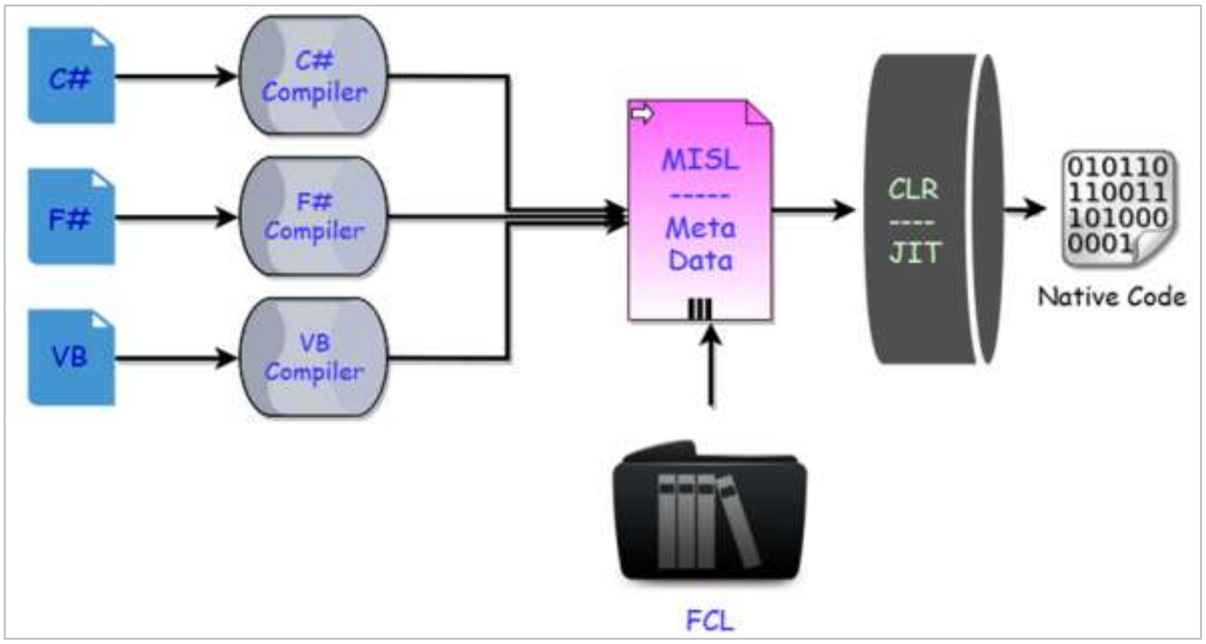
Generation Third (2)

- In Generation 2, all long objects are lived and its heap can grow to be very large.
- The objects in this generation can survive a long time and there is no next generation heap to further promote objects.
- The Garbage Collector has an additional heap for large objects known as Large Object Heap (LOH).
- It is reserved for objects that are 85,000 bytes or greater.
- Large objects are not allocated to the generational heaps but are allocated directly to the LOH.
- Generation 2 and LOH collections can take noticeable time for programs that have run for a long time or operate over large amounts of data.
- Large server programs are known to have heaps in the 10s of GBs.
- The GC employs a variety of techniques to reduce the amount of time that it blocks program execution.
- The primary approach is to do as much garbage collection work as possible on a background thread in a way that does not interfere with program execution.
- The GC also exposes a few ways for developers to influence its behavior, which can be quite useful to improve performance.

7. .NET Core – Code Execution

In this chapter, we will understand the execution process of .NET Core and compare it with the .NET Framework. The managed execution process includes the following steps.

- Choosing a compiler
- Compiling your code to MSIL
- Compiling MSIL to native code
- Running code



Choosing a Compiler

- It is a multi-language execution environment, the runtime supports a wide variety of data types and language features.
- To obtain the benefits provided by the common language runtime, you must use one or more language compilers that target the runtime.

Compiling your code to MSIL

- Compiling translates your source code into Microsoft Intermediate Language (MSIL) and generates the required metadata.

- Metadata describes the types in your code, including the definition of each type, the signatures of each type's members, the members that your code references, and other data that the runtime uses at execution time.
- The runtime locates and extracts the metadata from the file as well as from framework class libraries (FCL) as needed during execution.