Chapter 1: Getting started with C# Language

Version	Release Date
1.0	2002-01-01
1.2	2003-04-01
2.0	2005-09-01
3.0	2007-08-01
4.0	2010-04-01
5.0	2013-06-01
6.0	2015-07-01
7.0	2017-03-07

Section 1.1: Creating a new console application (Visual Studio)

1. Open Visual Studio
2. In the toolbar, go to **File** → **New Project**
3. Select the **Console Application** project type
4. Open the file `Program.cs` in the Solution Explorer
5. Add the following code to `Main()`:

```
public class Program
{
    public static void Main()
    {
        // Prints a message to the console.
        System.Console.WriteLine("Hello, World!");

        /* Wait for the user to press a key. This is a common
           way to prevent the console window from terminating
           and disappearing before the programmer can see the contents
           of the window, when the application is run via Start from within VS. */
        System.Console.ReadKey();
    }
}
```

6. In the toolbar, click **Debug** -> **Start Debugging** or hit **F5** or **ctrl + F5** (running without debugger) to run the program.

Live Demo on ideone

Explanation

- `class Program` is a class declaration. The class `Program` contains the data and method definitions that your program uses. Classes generally contain multiple methods. Methods define the behavior of the class. However, the `Program` class has only one method: `Main`.
- `static void Main()` defines the `Main` method, which is the entry point for all C# programs. The `Main` method states what the class does when executed. Only one `Main` method is allowed per class.
- `System.Console.WriteLine("Hello, world!");` method prints a given data (in this example, `Hello, world!`) as an output in the console window.

- `System.Console.ReadKey()`, ensures that the program won't close immediately after displaying the message. It does this by waiting for the user to press a key on the keyboard. Any key press from the user will terminate the program. The program terminates when it has finished the last line of code in the `main()` method.

Using the command line

To compile via command line use either `MSBuild` or `csc.exe` *(the C# compiler)*, both part of the Microsoft Build Tools package.

To compile this example, run the following command in the same directory where `HelloWorld.cs` is located:

```
%WINDIR%\\Microsoft.NET\\Framework64\\v4.0.30319\\csc.exe HelloWorld.cs
```

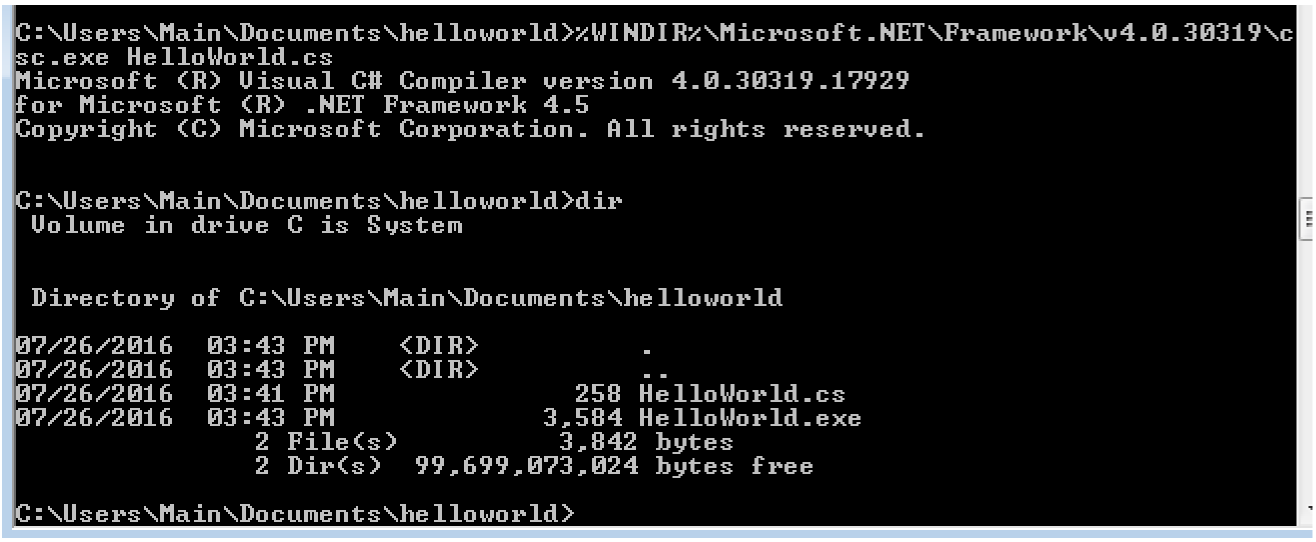
It can also be possible that you have two main methods inside one application. In this case, you have to tell the compiler which main method to execute by typing the following command in the **console**.(suppose Class `ClassA` also has a main method in the same `HelloWorld.cs` file in HelloWorld namespace)

```
%WINDIR%\\Microsoft.NET\\Framework64\\v4.0.30319\\csc.exe HelloWorld.cs /main:HelloWorld.ClassA
```

where HelloWorld is namespace

Note*: This is the path where* ***.NET framework v4.0*** *is located in general. Change the path according to your .NET version. In addition, the directory might be* ***framework*** *instead of* ***framework64*** *if you're using the 32-bit .NET Framework. From the Windows Command Prompt, you can list all the csc.exe Framework paths by running the following commands (the first for 32-bit Frameworks):*

```
dir %WINDIR%\\Microsoft.NET\\Framework\\csc.exe /s/b
dir %WINDIR%\\Microsoft.NET\\Framework64\\csc.exe /s/b
```



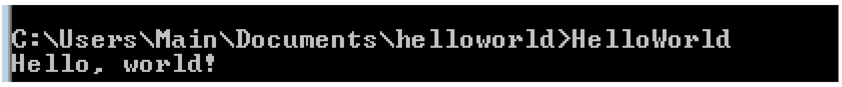
```
C:\Users\Main\Documents\helloworld>%WINDIR%\Microsoft.NET\Framework\v4.0.30319\c
sc.exe HelloWorld.cs
Microsoft (R) Visual C# Compiler version 4.0.30319.17929
for Microsoft (R) .NET Framework 4.5
Copyright (C) Microsoft Corporation. All rights reserved.


C:\Users\Main\Documents\helloworld>dir
 Volume in drive C is System


 Directory of C:\Users\Main\Documents\helloworld

07/26/2016  03:43 PM    <DIR>          .
07/26/2016  03:43 PM    <DIR>          ..
07/26/2016  03:41 PM               258 HelloWorld.cs
07/26/2016  03:43 PM             3,584 HelloWorld.exe
               2 File(s)          3,842 bytes
               2 Dir(s)  99,699,073,024 bytes free

C:\Users\Main\Documents\helloworld>
```

There should now be an executable file named `HelloWorld.exe` in the same directory. To execute the program from the command prompt, simply type the executable's name and hit `Enter` as follows:

```
HelloWorld.exe
```

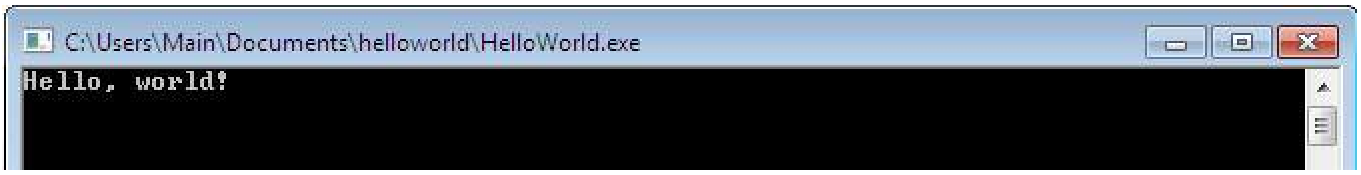
This will produce:

> Hello, world!



```
C:\Users\Main\Documents\helloworld>HelloWorld
Hello, world!
```

You may also double click the executable and launch a new console window with the message "**Hello, world!**"



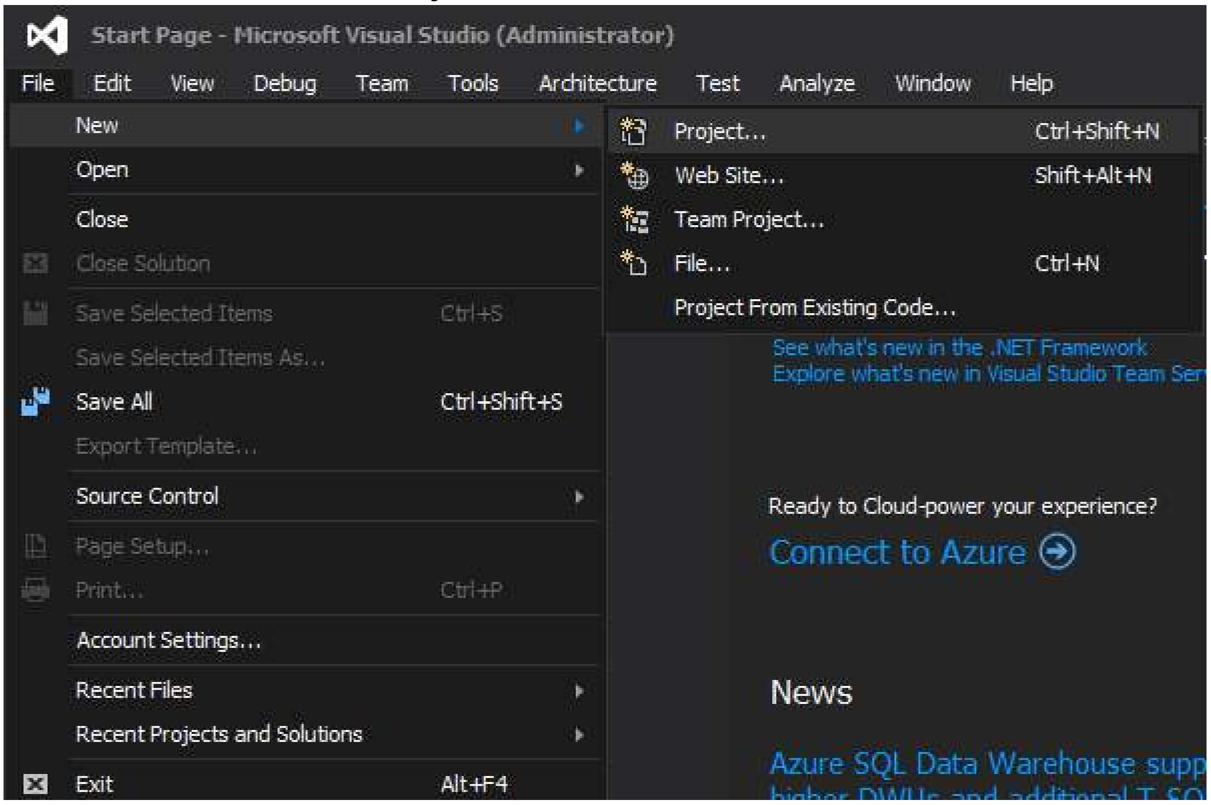
```
C:\Users\Main\Documents\helloworld\HelloWorld.exe
Hello, world!
```

Section 1.2: Creating a new project in Visual Studio (console application) and Running it in Debug mode

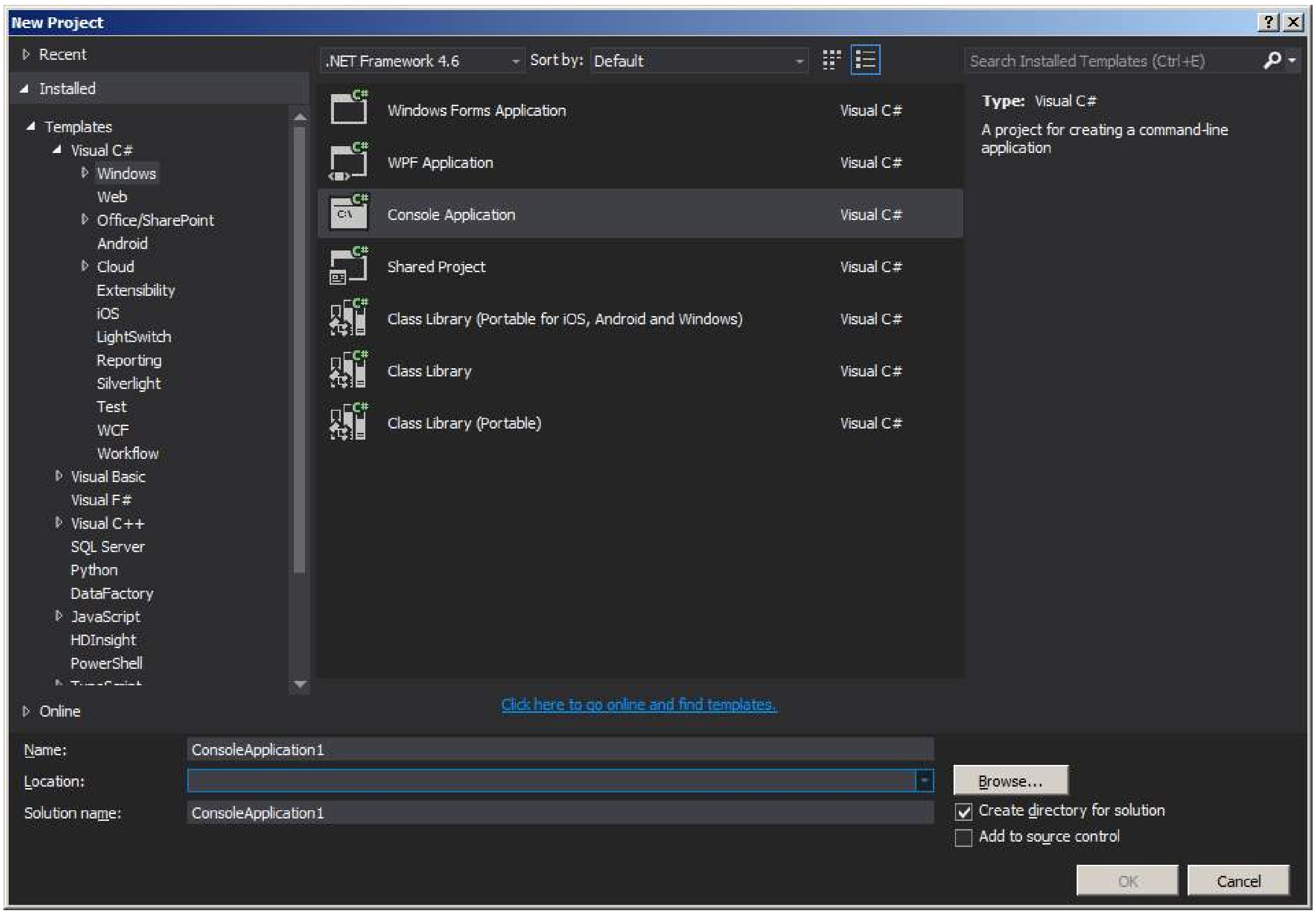
1. **Download and install Visual Studio**. Visual Studio can be downloaded from VisualStudio.com. The Community edition is suggested, first because it is free, and second because it involves all the general features and can be extended further.

2. **Open Visual Studio.**

3. **Welcome.** Go to **File** → **New** → **Project**.


Start Page - Microsoft Visual Studio (Administrator)
File
Edit
View
Debug
Team
Tools
Architecture
Test
Analyze
Window
Help
New
Open
Close
Close Solution
Save Selected Items Ctrl+S
Save Selected Items As...
Save All Ctrl+Shift+S
Export Template...
Source Control
Page Setup...
Print... Ctrl+P
Account Settings...
Recent Files
Recent Projects and Solutions
Exit Alt+F4
Project... Ctrl+Shift+N
Web Site... Shift+Alt+N
Team Project...
File... Ctrl+N
Project From Existing Code...
See what's new in the .NET Framework
Explore what's new in Visual Studio Team Ser
Ready to Cloud-power your experience?
Connect to Azure
News
Azure SQL Data Warehouse supp

4. Click **Templates** → **Visual C#** → **Console Application**



New Project

- Recent
- Installed
  - Templates
    - Visual C#
      - Windows
      - Web
      - Office/SharePoint
      - Android
      - Cloud
      - Extensibility
      - iOS
      - LightSwitch
      - Reporting
      - Silverlight
      - Test
      - WCF
      - Workflow
    - Visual Basic
    - Visual F#
    - Visual C++
    - SQL Server
    - Python
    - DataFactory
    - JavaScript
    - HDInsight
    - PowerShell
- Online

.NET Framework 4.6 Sort by: Default

Search Installed Templates (Ctrl+E)

Windows Forms Application	Visual C#
WPF Application	Visual C#
Console Application	Visual C#
Shared Project	Visual C#
Class Library (Portable for iOS, Android and Windows)	Visual C#
Class Library	Visual C#
Class Library (Portable)	Visual C#

Type: Visual C#

A project for creating a command-line application

Click here to go online and find templates.

Name: ConsoleApplication1

Location:

Solution name: ConsoleApplication1

Browse...

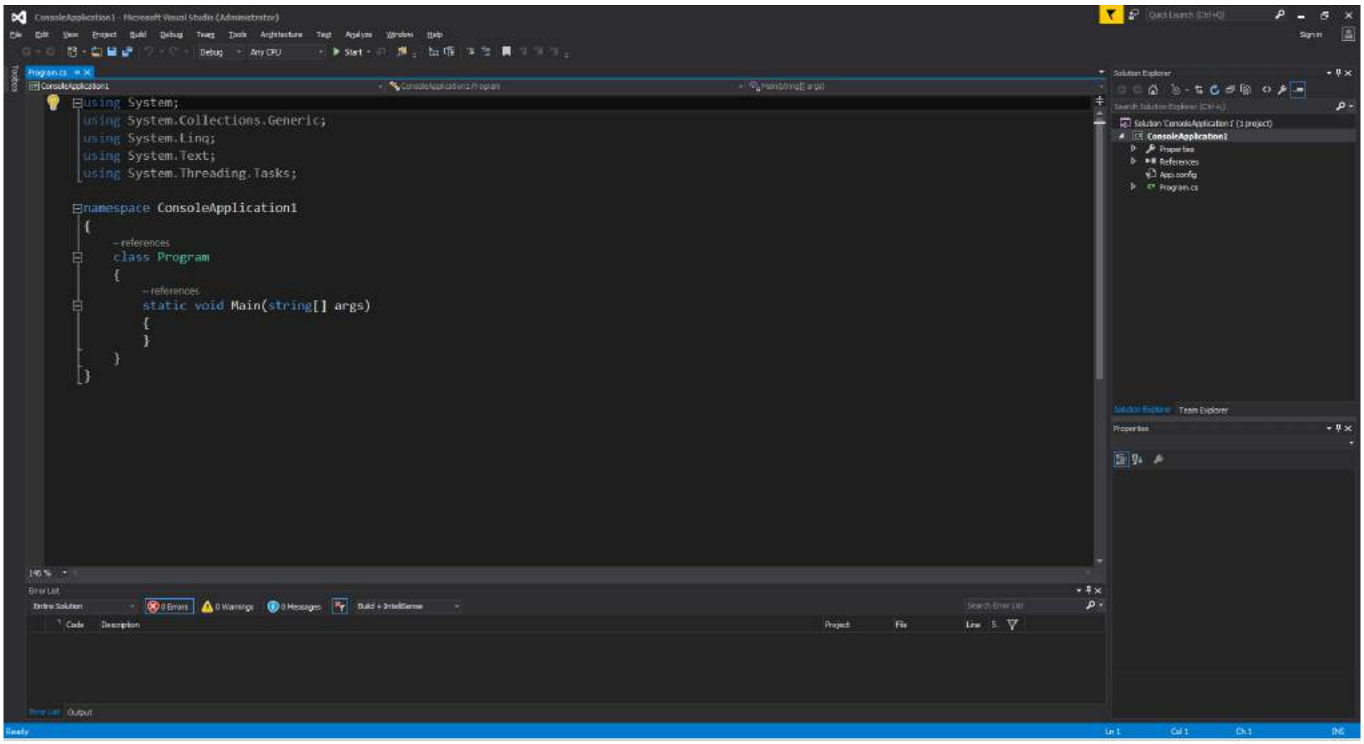
[✓] Create directory for solution

[ ] Add to source control

OK Cancel

5. **After selecting Console Application,** Enter a name for your project, and a location to save and press OK. Don't worry about the Solution name.

6. **Project created**. The newly created project will look similar to:



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

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
        }
    }
}
```

(Always use descriptive names for projects so that they can easily be distinguished from other projects. It is recommended not to use spaces in project or class name.)

7. **Write code.** You can now update your `Program.cs` to present "Hello world!" to the user.

```
using System;

namespace ConsoleApplication1
{
    public class Program
    {
        public static void Main(string[] args)
        {
        }
    }
}
```

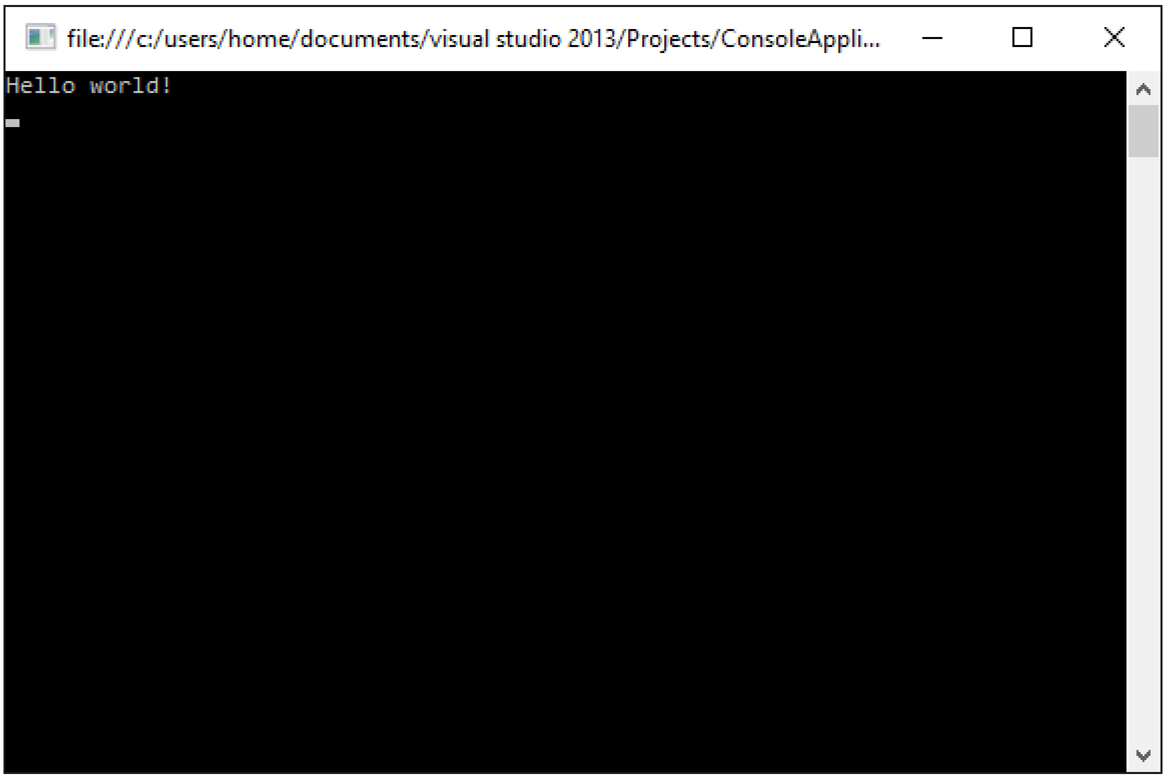
Add the following two lines to the `public static void Main(string[] args)` object in `Program.cs`: (make sure it's inside the braces)

```
Console.WriteLine("Hello world!");
Console.Read();
```

Why `Console.Read()`? The first line prints out the text "Hello world!" to the console, and the second line waits for a single character to be entered; in effect, this causes the program to pause execution so that you're able to see the output while debugging. Without `Console.Read();`, when you start debugging the application it will just print "Hello world!" to the console and then immediately close. Your code window should now look like the following:

```
using System;

namespace ConsoleApplication1
{
    public class Program
    {
        public static void Main(string[] args)
        {
            Console.WriteLine("Hello world!");
            Console.Read();
        }
    }
}
```

8. **Debug your program.** Press the Start Button on the toolbar near the top of the window ▶ Start ▾ or press F5 on your keyboard to run your application. If the button is not present, you can run the program from the top menu: **Debug** → **Start Debugging**. The program will compile and then open a console window. It should look similar to the following screenshot:


file:///c:/users/home/documents/visual studio 2013/Projects/ConsoleAppli...

```
Hello world!
```

9. **Stop the program.** To close the program, just press any key on your keyboard. The `Console.Read()` we added was for this same purpose. Another way to close the program is by going to the menu where the `Start` button was, and clicking on the `Stop` button.

Section 1.3: Creating a new program using .NET Core

First install the **.NET Core SDK** by going through the installation instructions for the platform of your choice:

- Windows
- OSX
- Linux
- Docker

After the installation has completed, open a command prompt, or terminal window.

1. Create a new directory with `mkdir hello_world` and change into the newly created directory with `cd hello_world`.

2. Create a new console application with `dotnet new console`.
   This will produce two files:

   - **hello_world.csproj**

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

     <PropertyGroup>
       <OutputType>Exe</OutputType>
       <TargetFramework>netcoreapp1.1</TargetFramework>
     </PropertyGroup>

   </Project>
   ```

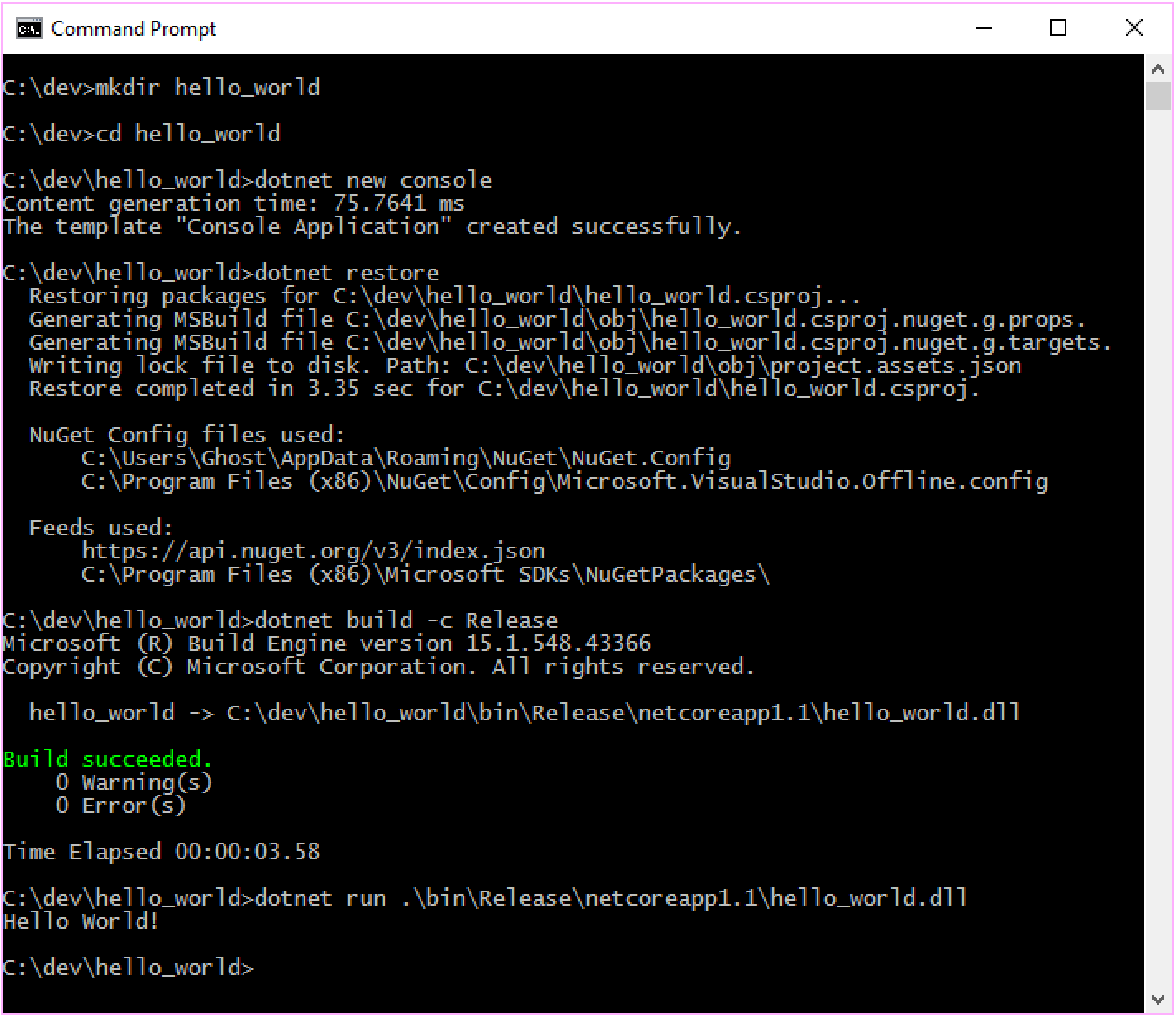
   - **Program.cs**

   ```
   using System;
   ```

```
namespace hello_world
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
        }
    }
}
```

3. Restore the needed packages with `dotnet restore`.
4. *Optional* Build the application with `dotnet build` for Debug or `dotnet build -c Release` for Release. `dotnet run` will also run the compiler and throw build errors, if any are found.
5. Run the application with `dotnet run` for Debug or `dotnet run .\bin\Release\netcoreapp1.1\hello_world.dll` for Release.

Command Prompt output



```
C:\dev>mkdir hello_world

C:\dev>cd hello_world

C:\dev\hello_world>dotnet new console
Content generation time: 75.7641 ms
The template "Console Application" created successfully.

C:\dev\hello_world>dotnet restore
  Restoring packages for C:\dev\hello_world\hello_world.csproj...
  Generating MSBuild file C:\dev\hello_world\obj\hello_world.csproj.nuget.g.props.
  Generating MSBuild file C:\dev\hello_world\obj\hello_world.csproj.nuget.g.targets.
  Writing lock file to disk. Path: C:\dev\hello_world\obj\project.assets.json
  Restore completed in 3.35 sec for C:\dev\hello_world\hello_world.csproj.

  NuGet Config files used:
      C:\Users\Ghost\AppData\Roaming\NuGet\NuGet.Config
      C:\Program Files (x86)\NuGet\Config\Microsoft.VisualStudio.Offline.config

  Feeds used:
      https://api.nuget.org/v3/index.json
      C:\Program Files (x86)\Microsoft SDKs\NuGetPackages\

C:\dev\hello_world>dotnet build -c Release
Microsoft (R) Build Engine version 15.1.548.43366
Copyright (C) Microsoft Corporation. All rights reserved.

  hello_world -> C:\dev\hello_world\bin\Release\netcoreapp1.1\hello_world.dll

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

Time Elapsed 00:00:03.58

C:\dev\hello_world>dotnet run .\bin\Release\netcoreapp1.1\hello_world.dll
Hello World!

C:\dev\hello_world>
```

Section 1.4: Creating a new program using Mono

First install Mono by going through the install instructions for the platform of your choice as described in their installation section.

Mono is available for Mac OS X, Windows and Linux.

After installation is done, create a text file, name it `HelloWorld.cs` and copy the following content into it:

```
public class Program
{
    public static void Main()
    {
        System.Console.WriteLine("Hello, world!");
        System.Console.WriteLine("Press any key to exit..");
        System.Console.Read();
    }
}
```

If you are using Windows, run the Mono Command Prompt which is included in the Mono installation and ensures that the necessary environment variables are set. If on Mac or Linux, open a new terminal.

To compile the newly created file, run the following command in the directory containing `HelloWorld.cs`:

```
mcs -out:HelloWorld.exe HelloWorld.cs
```

The resulting `HelloWorld.exe` can then be executed with:

```
mono HelloWorld.exe
```

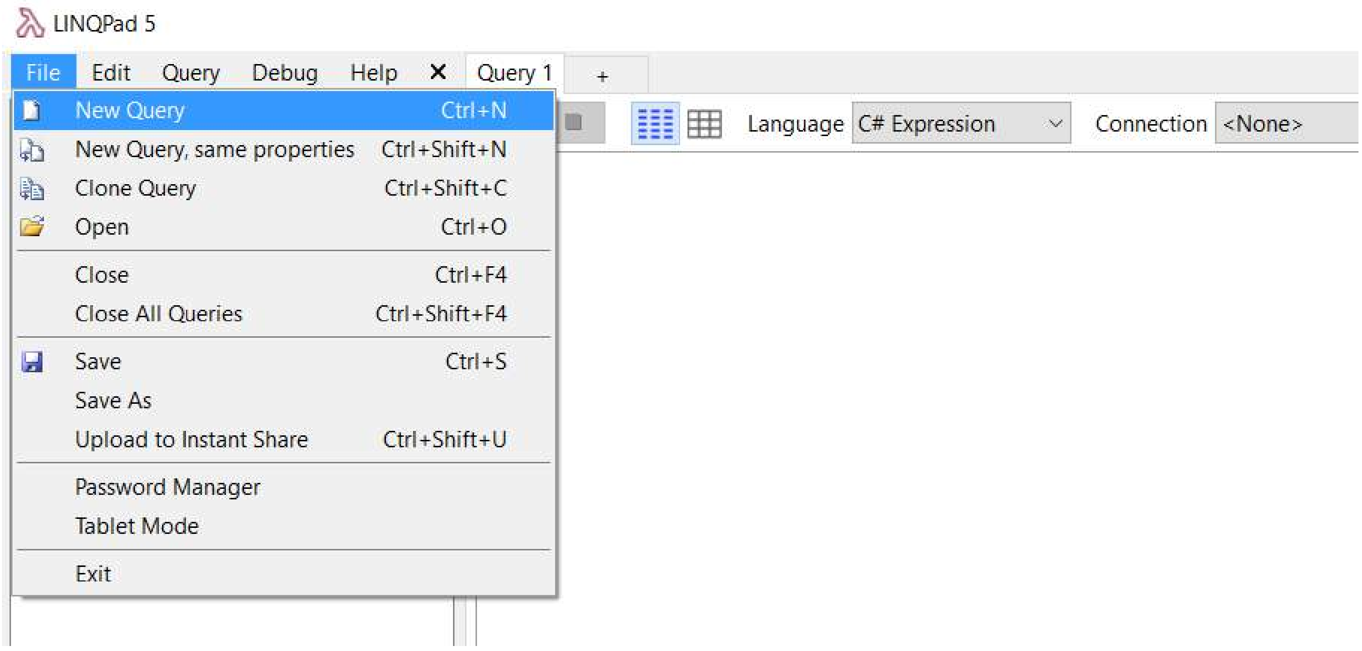
which will produce the output:

```
Hello, world!
Press any key to exit..
```

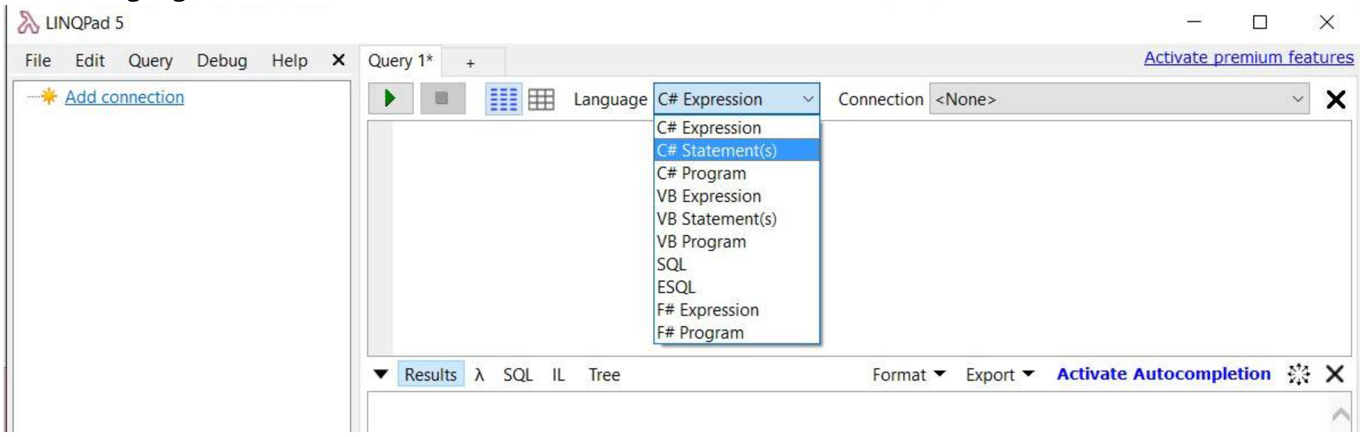
Section 1.5: Creating a new query using LinqPad

LinqPad is a great tool that allows you to learn and test features of .Net languages (C#, F# and VB.Net.)

1. Install LinqPad
2. Create a new Query (`Ctrl` + `N`)

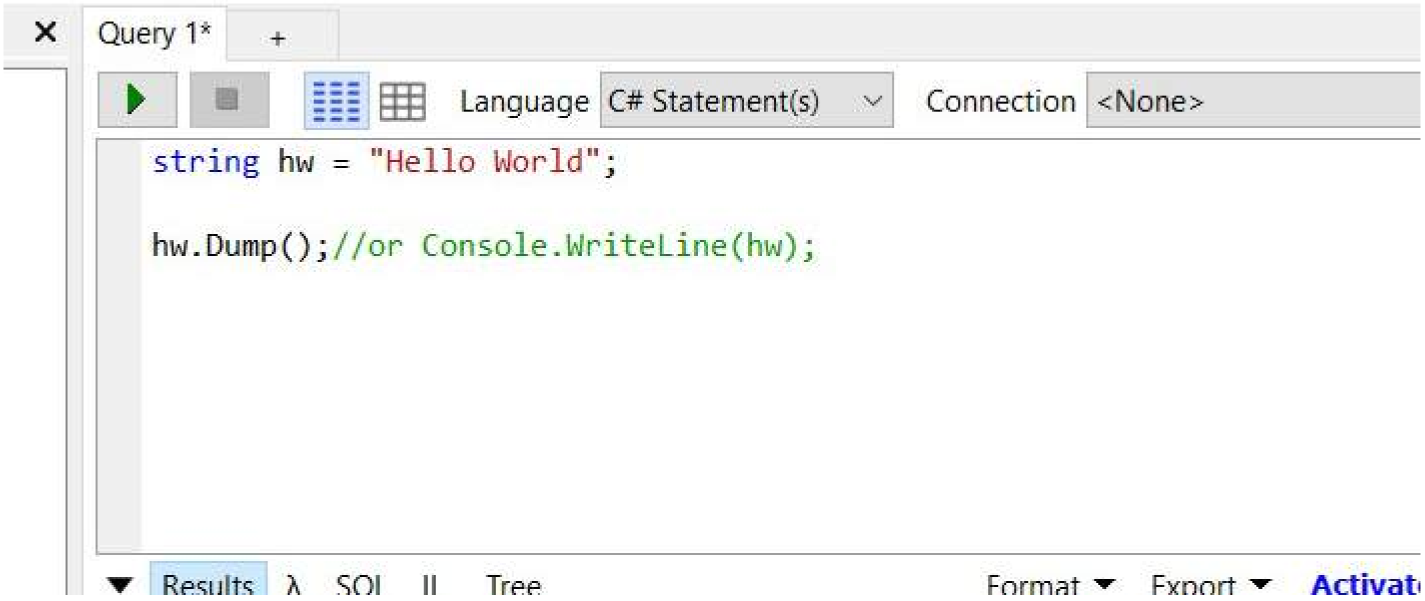


3. Under language, select "C# statements"



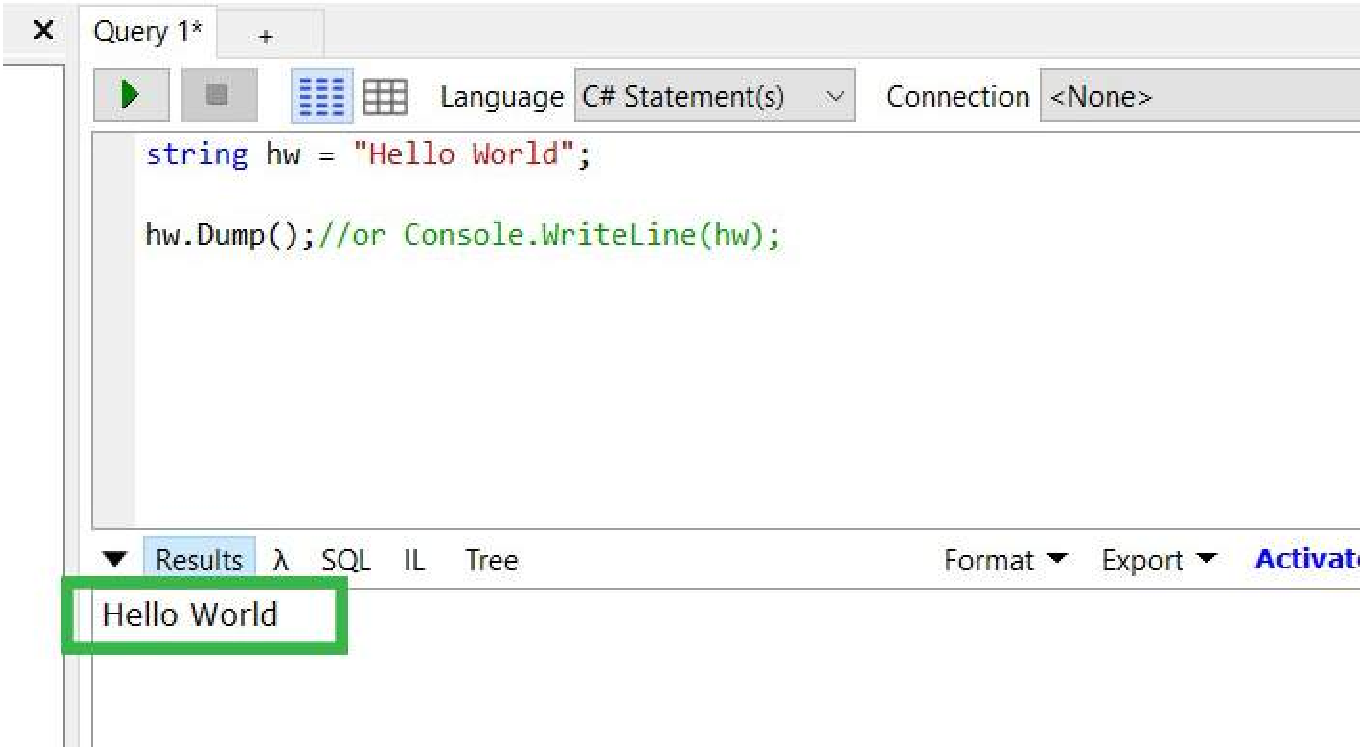
4. Type the following code and hit run (F5)

```
string hw = "Hello World";

hw.Dump(); //or Console.WriteLine(hw);
```



```
string hw = "Hello World";

hw.Dump();//or Console.WriteLine(hw);
```

5. You should see "Hello World" printed out in the results screen.

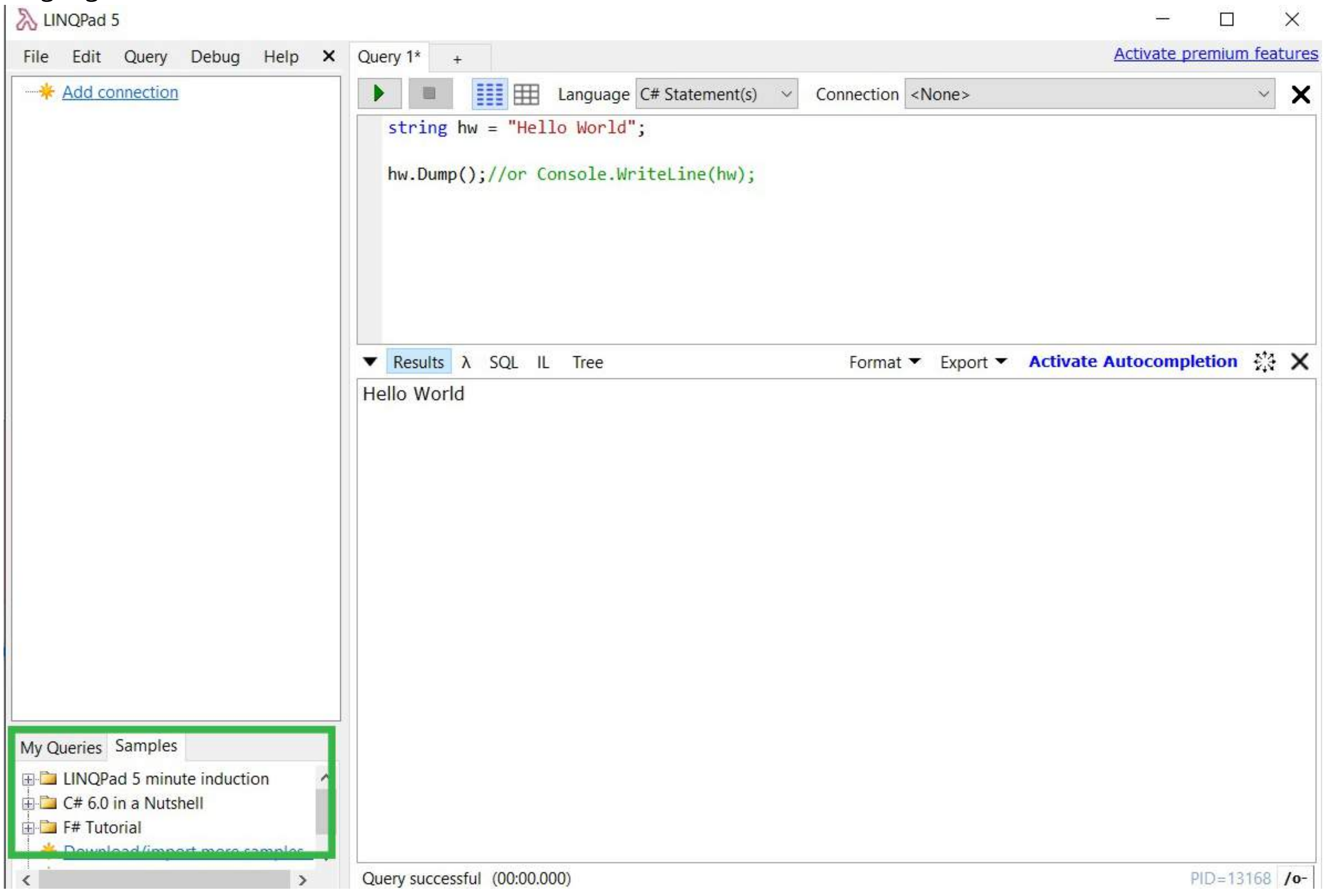


Language: C# Statement(s)
Connection: <None>

```
string hw = "Hello World";

hw.Dump();//or Console.WriteLine(hw);
```

```
Hello World
```

6. Now that you have created your first .Net program, go and check out the samples included in LinqPad via the "Samples" browser. There are many great examples that will show you many different features of the .Net languages.



Language: C# Statement(s)
Connection: <None>

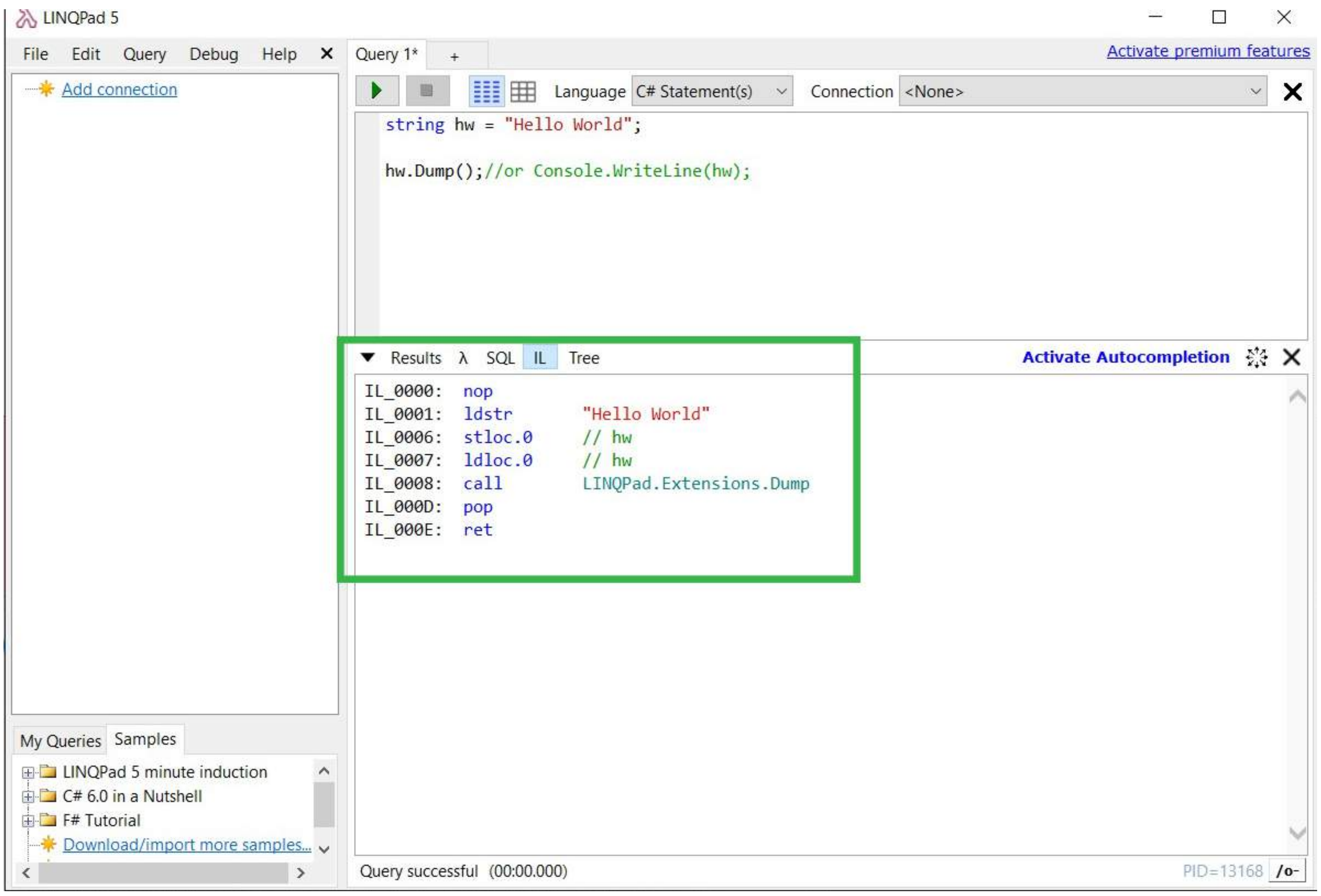
```
string hw = "Hello World";

hw.Dump();//or Console.WriteLine(hw);
```

```
Hello World
```

Query successful (00:00.000)
PID=13168 /o-

Notes:

1. If you click on "IL", you can inspect the IL code that your .net code generates. This is a great learning tool.



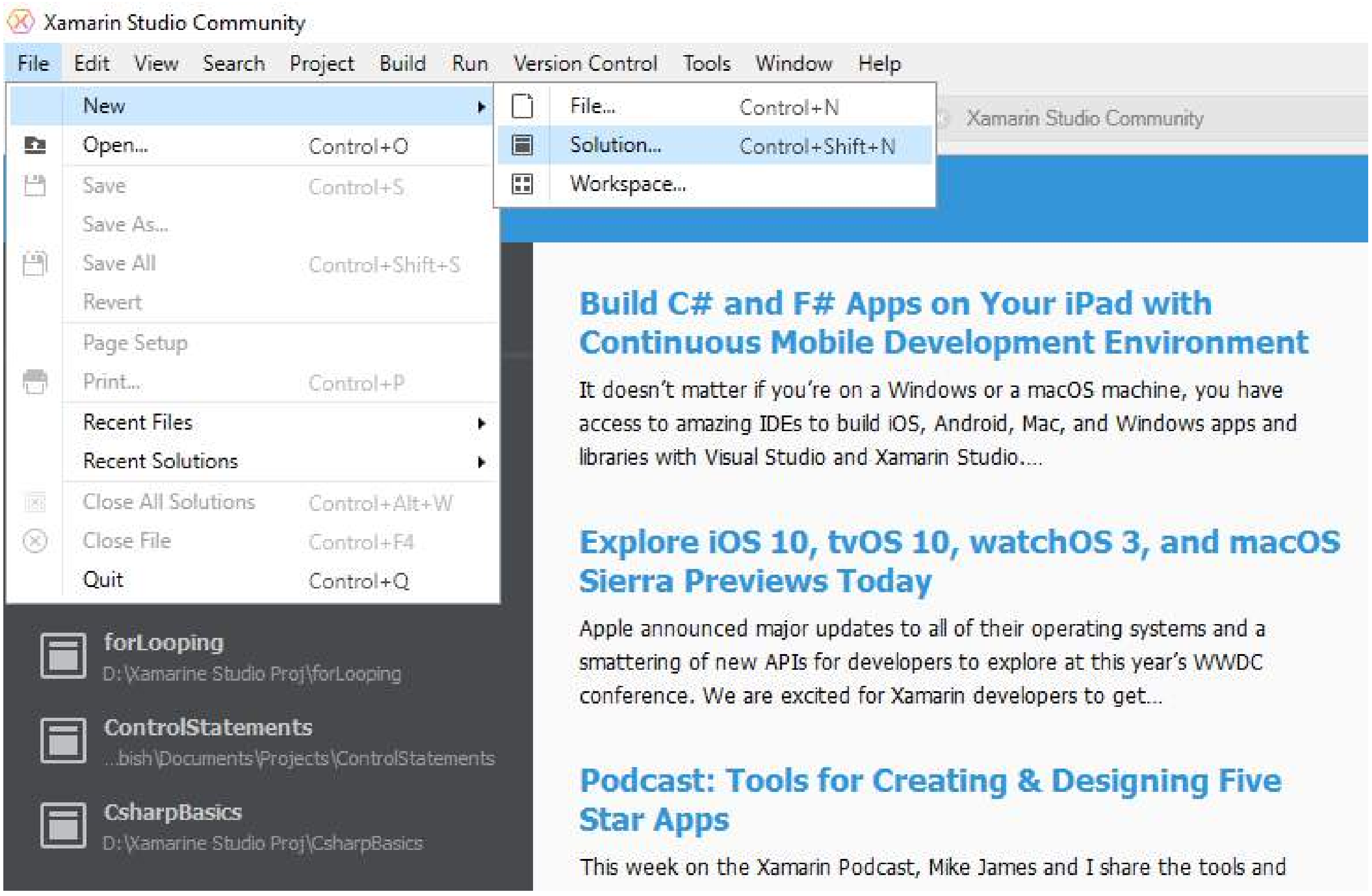
```
string hw = "Hello World";

hw.Dump();//or Console.WriteLine(hw);
```

```
IL_0000:  nop
IL_0001:  ldstr       "Hello World"
IL_0006:  stloc.0     // hw
IL_0007:  ldloc.0     // hw
IL_0008:  call        LINQPad.Extensions.Dump
IL_000D:  pop
IL_000E:  ret
```

2. When using `LINQ to SQL` or `Linq to Entities` you can inspect the SQL that's being generated which is another great way to learn about LINQ.

Section 1.6: Creating a new project using Xamarin Studio

1. Download and install Xamarin Studio Community.
2. Open Xamarin Studio.
3. Click **File** → **New** → **Solution**.



Build C# and F# Apps on Your iPad with Continuous Mobile Development Environment

It doesn't matter if you're on a Windows or a macOS machine, you have access to amazing IDEs to build iOS, Android, Mac, and Windows apps and libraries with Visual Studio and Xamarin Studio....

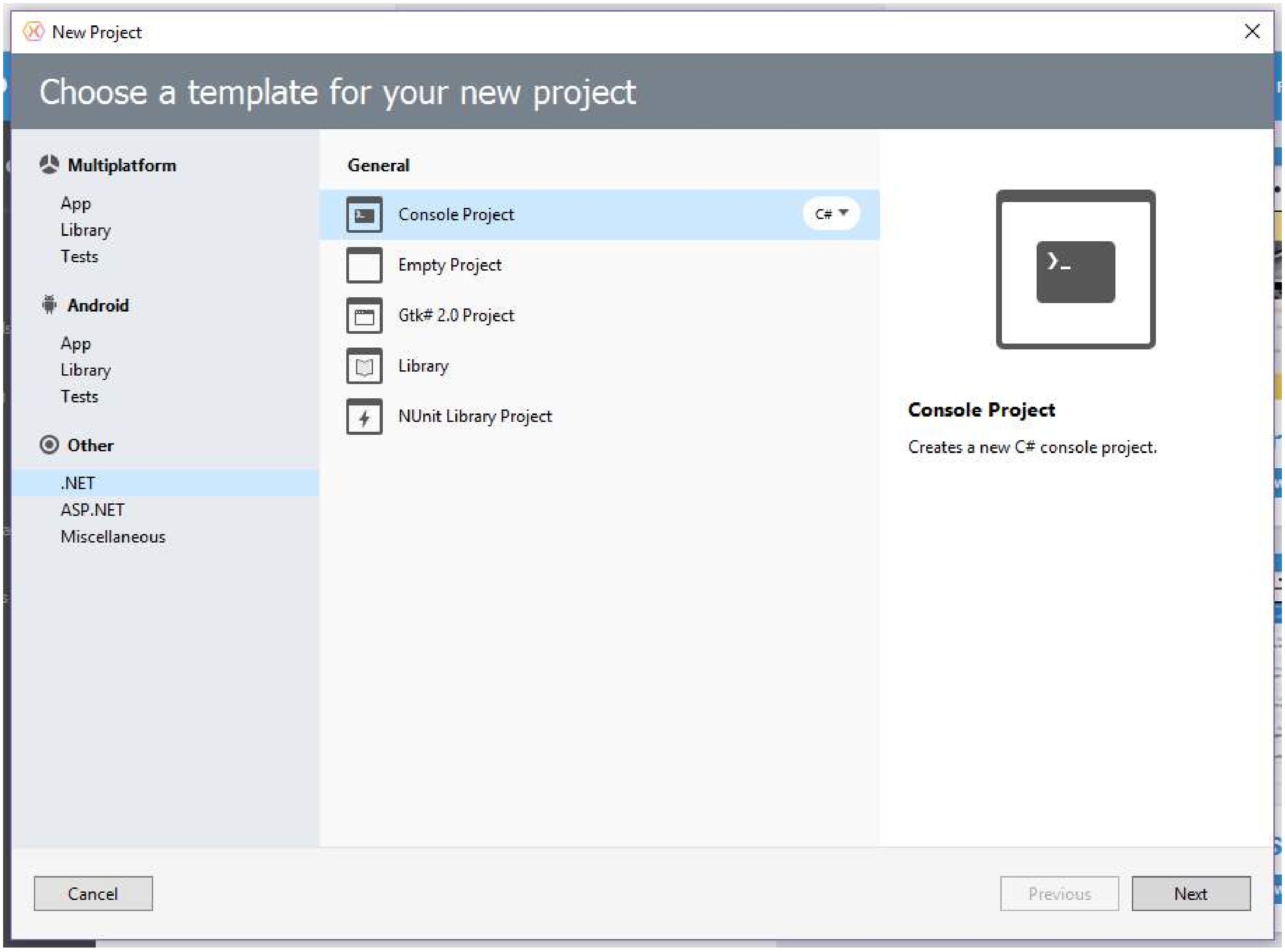
Explore iOS 10, tvOS 10, watchOS 3, and macOS Sierra Previews Today

Apple announced major updates to all of their operating systems and a smattering of new APIs for developers to explore at this year's WWDC conference. We are excited for Xamarin developers to get...

Podcast: Tools for Creating & Designing Five Star Apps

This week on the Xamarin Podcast, Mike James and I share the tools and

4. Click **.NET** → **Console Project** and choose **C#**.
5. Click `Next` to proceed.


New Project

Choose a template for your new project

Multiplatform
- App
- Library
- Tests

Android
- App
- Library
- Tests

Other
- .NET
- ASP.NET
- Miscellaneous

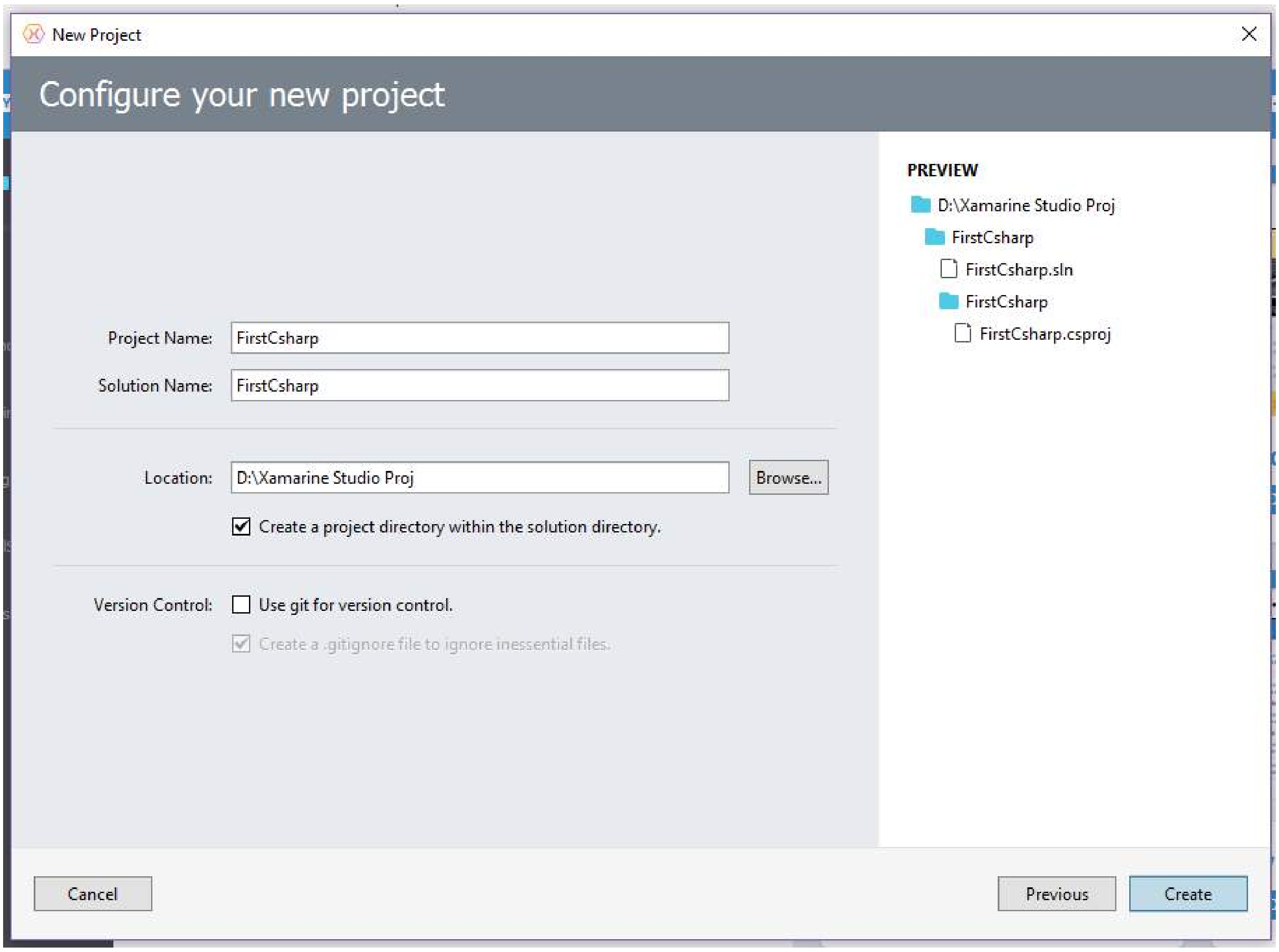
General
- Console Project C#
- Empty Project
- Gtk# 2.0 Project
- Library
- NUnit Library Project

Console Project

Creates a new C# console project.

Cancel | Previous | Next

6. Enter the **Project Name** and `Browse...` for a **Location** to Save and then click `Create`.


New Project

Configure your new project

Project Name: FirstCsharp

Solution Name: FirstCsharp

Location: D:\Xamarine Studio Proj

[✓] Create a project directory within the solution directory.

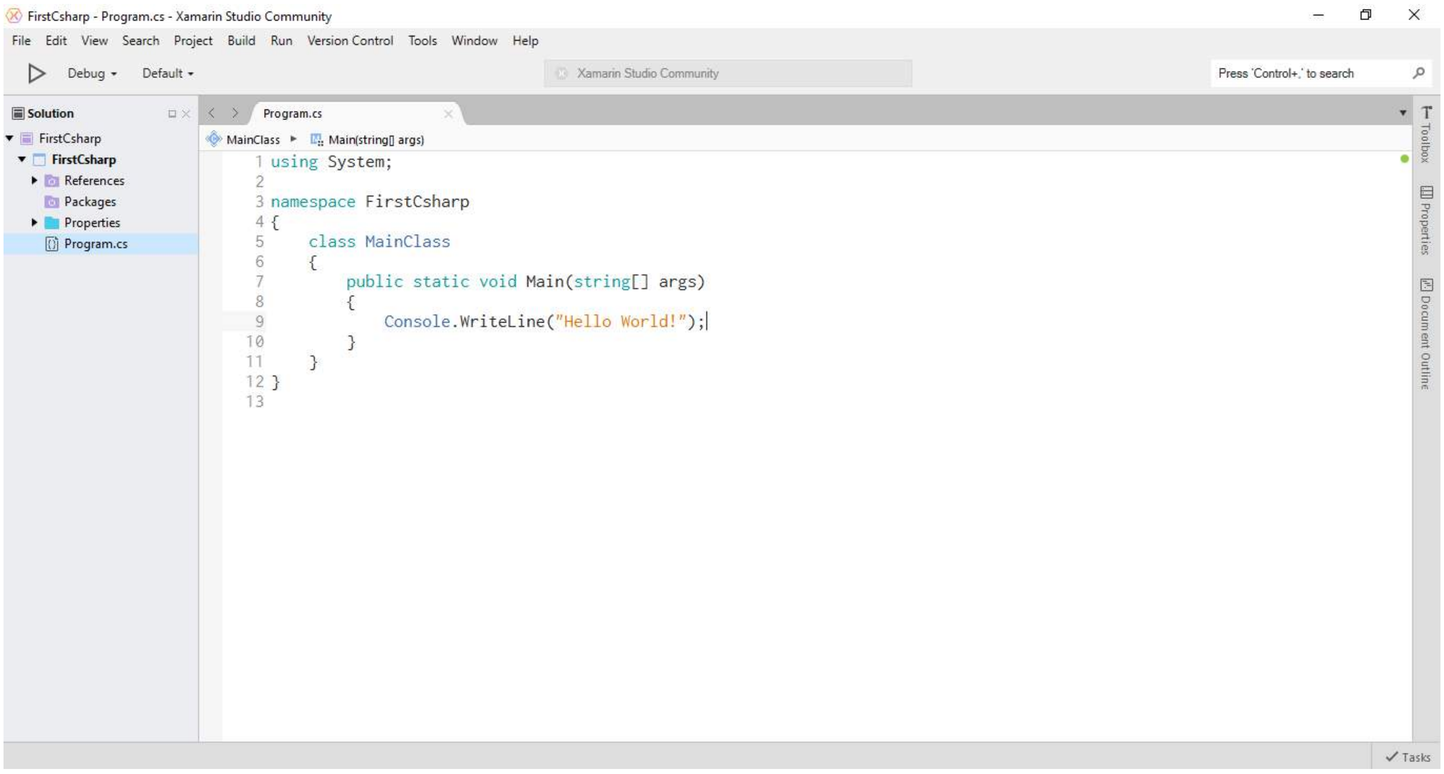
Version Control: [ ] Use git for version control.

[✓] Create a .gitignore file to ignore inessential files.

PREVIEW

- D:\Xamarine Studio Proj
  - FirstCsharp
    - FirstCsharp.sln
    - FirstCsharp
      - FirstCsharp.csproj

7. The newly created project will look similar to:

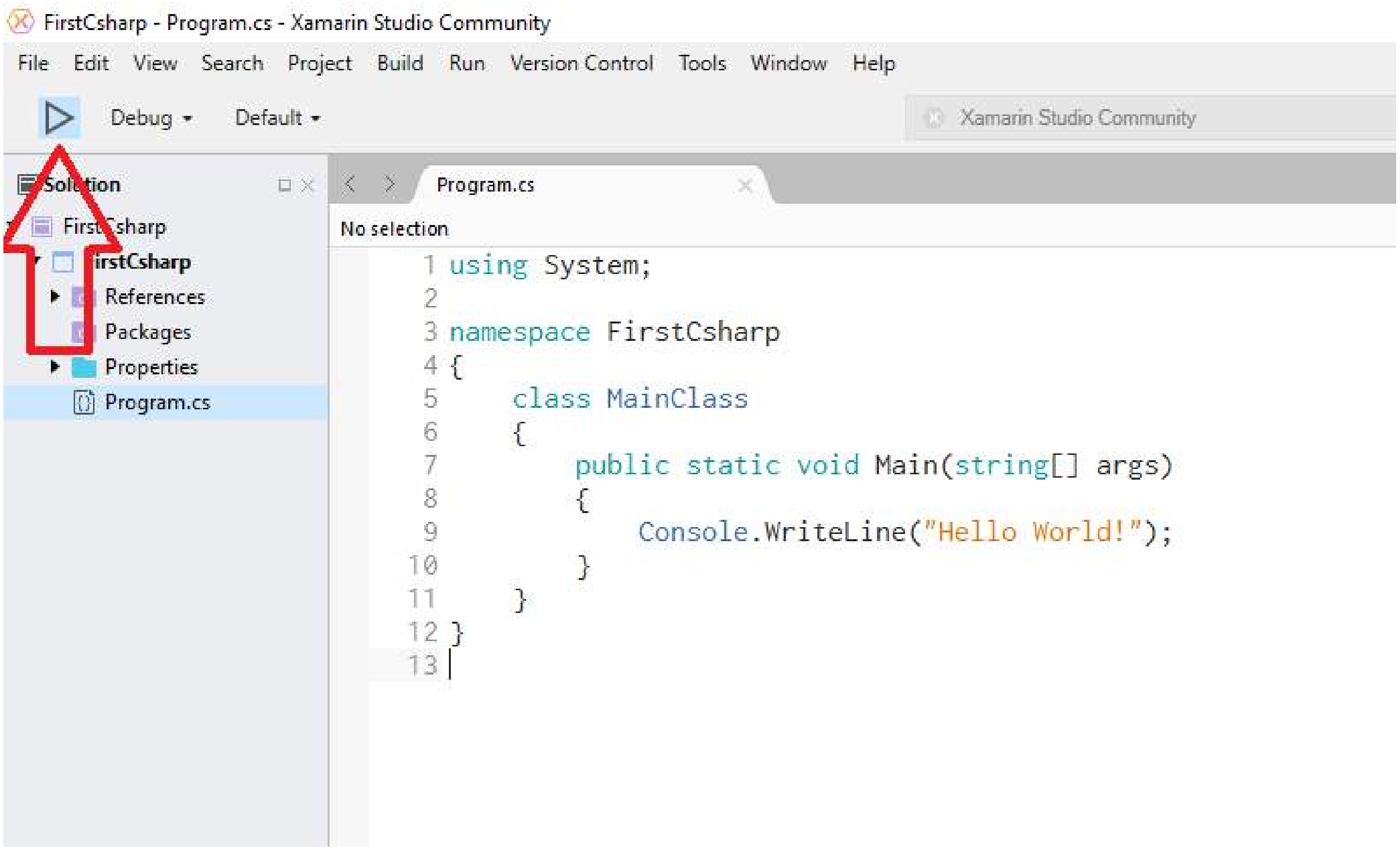

FirstCsharp - Program.cs - Xamarin Studio Community

```
using System;

namespace FirstCsharp
{
    class MainClass
    {
        public static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
        }
    }
}
```

8. This is the code in the Text Editor:

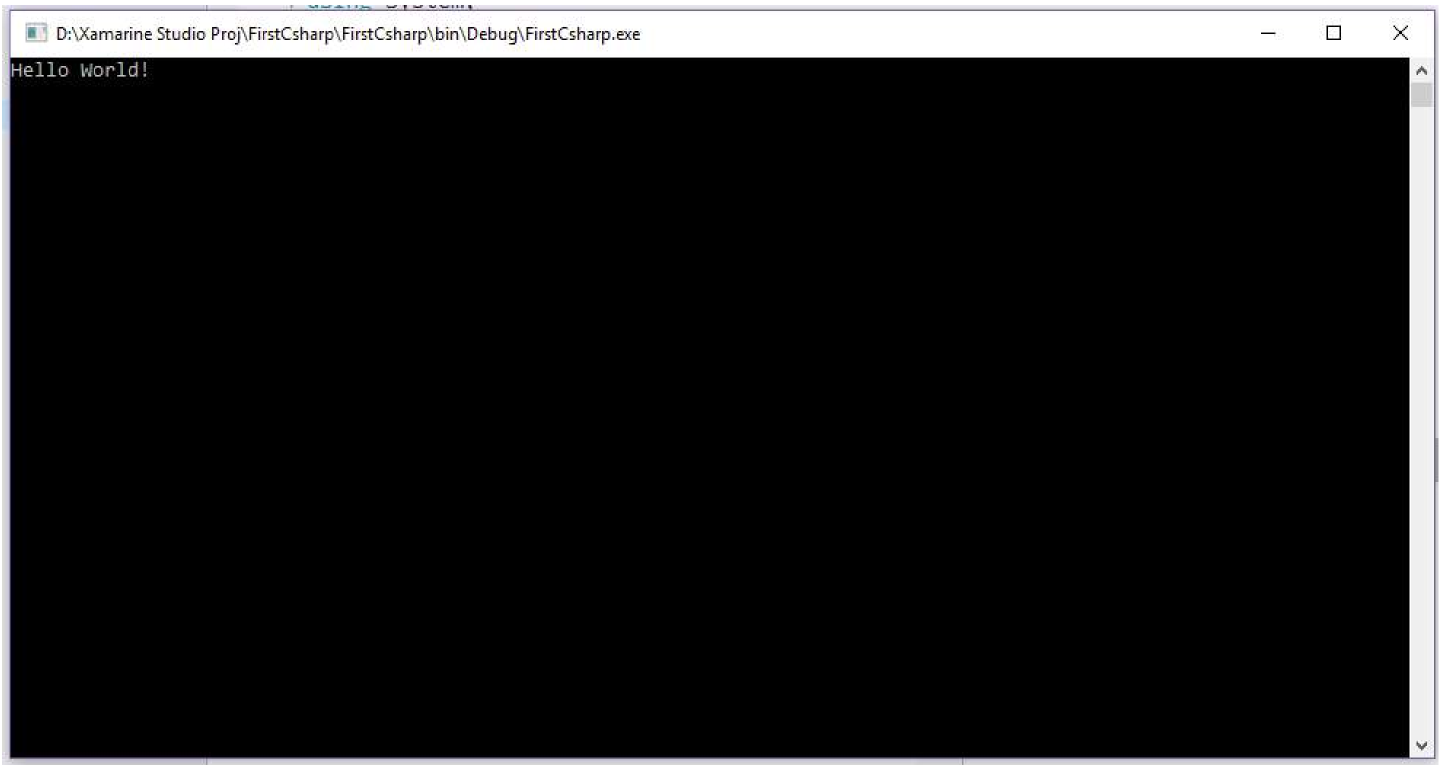
```
using System;

namespace FirstCsharp
{
    public class MainClass
    {
        public static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
            Console.ReadLine();
        }
    }
}
```

9. To run the code, press F5 or click the **Play Button** as shown below:



```
using System;

namespace FirstCsharp
{
    class MainClass
    {
        public static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
        }
    }
}
```

10. Following is the Output:


D:\Xamarine Studio Proj\FirstCsharp\FirstCsharp\bin\Debug\FirstCsharp.exe

```
Hello World!
```