



Getting Started with Oscilloscope Automation in C#

APPLICATION NOTE



Introduction

Most modern test and measurement instrumentation today can be configured and controlled via a remote programmable interface that is accessible over physical interfaces such as Ethernet, USB or GPIB. Even complex instruments like oscilloscopes can be fully controlled and directed to perform complex tests using only its programmable interface. In test and measurement, often there is a need to perform a series of tests, collect measurement data and repeat these actions multiple times on one or more devices under test. When performing repetitive testing and measurements, automation of instrumentation is key for consistency of test methodology, repeatability of measurement results, time savings and reduction of the risk for human error. For these reasons, often engineers choose to spend time to take advantage of the remote programmable interface capabilities of their instrument and write test code to automate their test and measurement applications. For many of these engineers, C# (pronounced C Sharp) is the programming language of choice.

C# is a versatile and powerful programming language that was developed by Microsoft as part of its .NET framework. It is widely used for building a variety of applications, ranging from desktop software to web applications and even mobile apps. Using easily integrated third-party libraries, C# is an excellent choice for automated test applications as well. Many engineers in test and measurement choose to write their automated test code in C# for many reasons, including:

- Excellent instrument communication support available through the IVI VISA.NET library.
- Hundreds of useful libraries built in to the .NET Framework make everyday code tasks easy and are well documented.
- Development performed using the powerful and easy to use Visual Studio Integrated Development Environment.
- Free to use Visual Studio Community Edition available.
- IntelliSense in the Visual Studio code editor makes writing code and working with new code libraries a breeze.

- .NET Winforms library makes writing programs with a GUI easy.
- Clean syntax, similar to C/C++ that is familiar for many people.
- Object oriented language encapsulates code into objects making it more modular and reusable.
- Runtime memory manager automatically allocates and deallocates memory, making manual memory management unnecessary, avoiding memory leaks.
- Additional libraries readily available to extend the .NET framework through the NuGet package manager that is integrated into Visual Studio.

Getting Started

Recommended System Requirements

The follow list contains the recommend system requirements for following along with this guide.

- Personal computer running Windows 10 or Windows 11
 - Core i5-2500 or newer processor
 - 8 GB of RAM or greater
 - > 15 GB of free disk space

Recommended Equipment

- [Tektronix Oscilloscope](#)
 - 2/4/5/6 Series MSO Mixed Signal Oscilloscope
 - 3 Series MDO Mixed Domain Oscilloscope
 - MSO/DPO5000 B Series Oscilloscope
 - DPO7000 C Series Oscilloscope
 - MSO/DPO70000 B C Series Performance Oscilloscope
 - MSO/DPO/DSA70000 D/DX Series Performance Oscilloscope
 - DPO70000SX Series Performance Oscilloscope

Install the Development Environment

Before you can start automating oscilloscopes using C#, you will need to get your development environment setup. In this guide we will be using Microsoft Visual Studio Community 2022 as our development environment, NI-VISA as our instrument communications library and the IVI VISA.NET library for interfacing with VISA in C#.

Install Visual Studio

1. Download Visual Studio:

Go to <http://visualstudio.com> and download and install Visual Studio 2022. For this guide we will use Visual Studio Community 2022, Microsoft's free to use version of Visual Studio, but Visual Studio Professional or Enterprise 2022 may be used as well. Earlier versions of Visual Studio can also be used; however, the steps for setting up your project in these versions may vary slightly from what is shown in this guide.

2. Install Visual Studio:

Double-click the installer for Visual Studio to run it. During setup, the Visual Studio Installer will ask you to choose the type of Workload(s) you plan to use with Visual Studio. Select ".NET desktop development" then click the Install button to begin the installation processes.

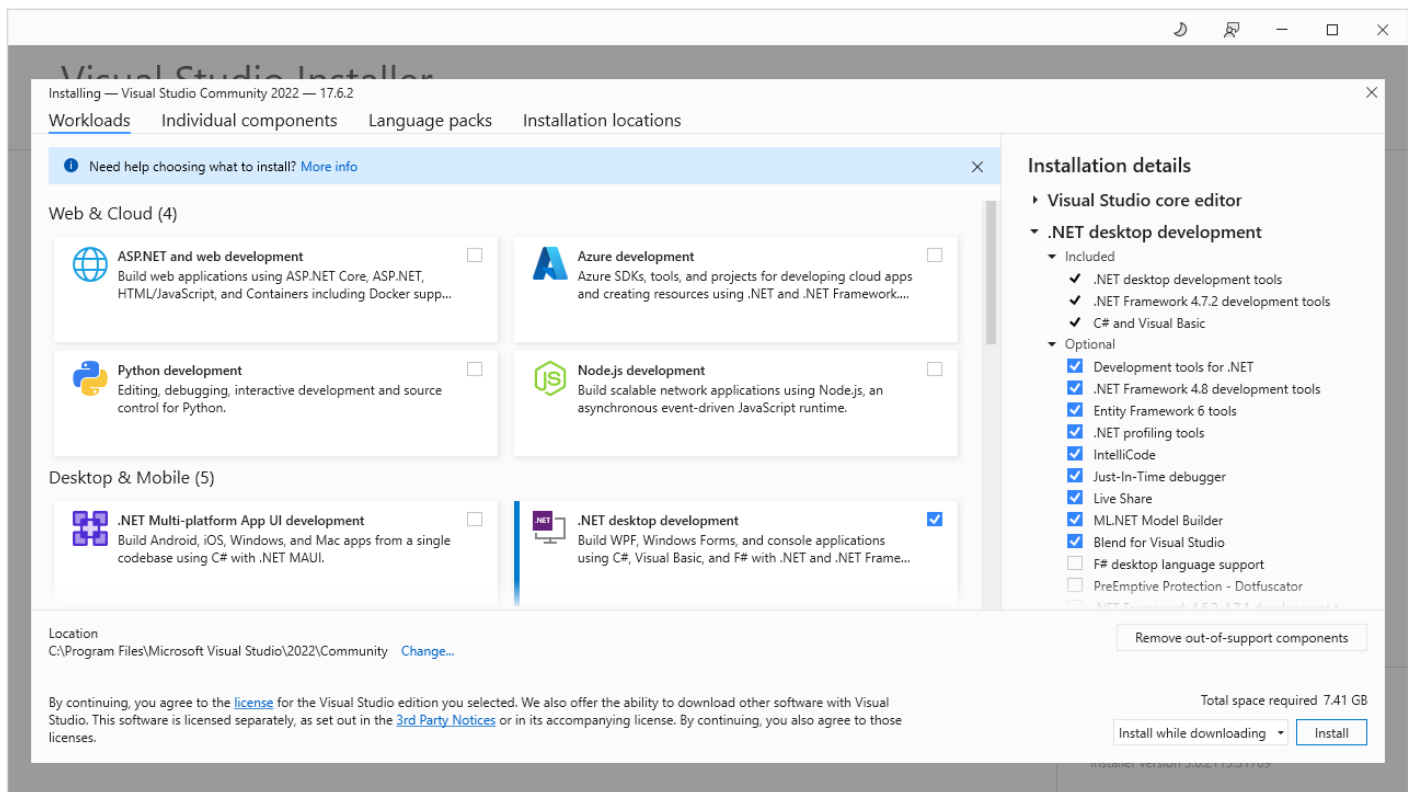


Figure 1: Select .NET desktop development in the Visual Studio installer

- When installation is complete, the installer will ask you to personalize Visual Studio. Since we will be developing in C#, it is generally recommended you choose Visual C# from the Development Settings drop-down.

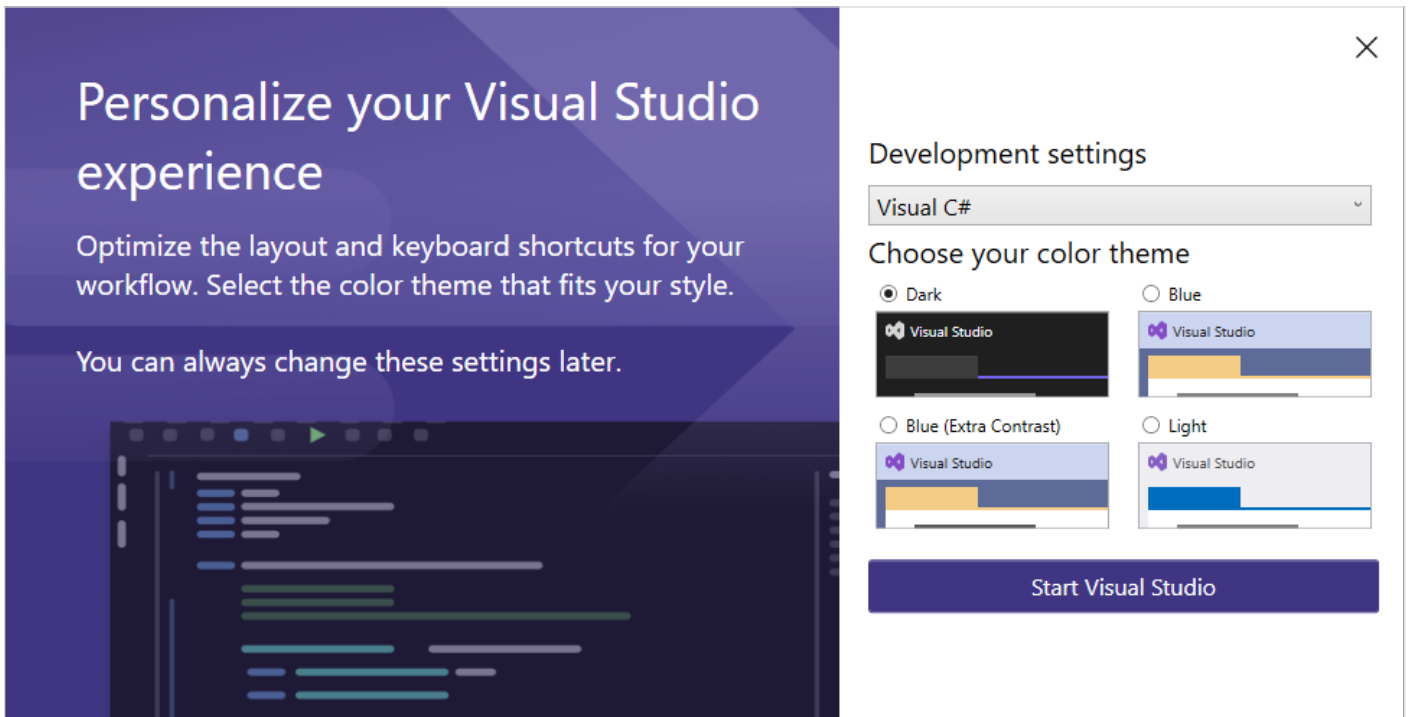


Figure 2: Personalize the theme used in Visual Studio

- Once you have made your selections, click Start Visual Studio.
- Visual Studio will take a few minutes to prepare itself for use. Once it is complete you will be presented with the Visual Studio 2022 Getting Started window. Close this window for now by clicking the close button in the upper right corner before proceeding to install NI-VISA.

Install VISA

Before we can begin writing programs to control instruments with C#, we need to install the VISA communications library on the system in which we installed Visual Studio. You should install NI-VISA now.

Note: If you have not yet installed Visual Studio, it is recommended that you do so before proceeding to install NI-VISA. The installer for NI-VISA will detect that Visual Studio is installed and will automatically make sure that the correct components are selected and installed for use in code development.

In this guide we will be using NI-VISA 2023 Q2. Other versions of NI-VISA as early as version 17 will work but the setup process may vary from what is shown in this guide and a separate installation of the IVI Compliance Package may be required to gain support for the IVI VISA.NET application programming interface. NI-VISA 2023 Q2 contains all needed packages and will be the only file you need to download and install.

Note: When downloading and installing NI-VISA, if there is an option between a Full version and a Run-time version, be sure to get the Full version. The Full version has additional tools and libraries that are needed for code development.

A complete guide on how to install VISA and use it for instrument control can be found in the E-book *Getting Started Controlling Instrument with VISA* which can be downloaded from tek.com.

Developing Instrument Control Applications with C#

With Visual Studio and NI-VISA installed, you are now ready to begin developing programs to control instruments using C#. For the next step in this guide, we will show you how to create a new C# project in Visual Studio, set it up to use the VISA communications library and then write some code to perform some simple oscilloscope communication.

Creating a New C# Console Project for Instrument Control (Hello World)

The first example presented in just about every programming introduction is the classic “Hello World” program. This guide will be no different and you will learn how to create the Instrument Control equivalent of the Hello World program by creating a program that connects to an instrument, queries its ID string and then prints it to the screen. We will then guide you to modify this program to perform some basic oscilloscope control where we will reset the instrument, turn on a measurement and then fetch the measurement value and print it to the screen.

1. Launch Visual Studio and it will bring you to the Visual Studio Getting Started screen. On the Getting Started screen click the option called “Create a new project.”

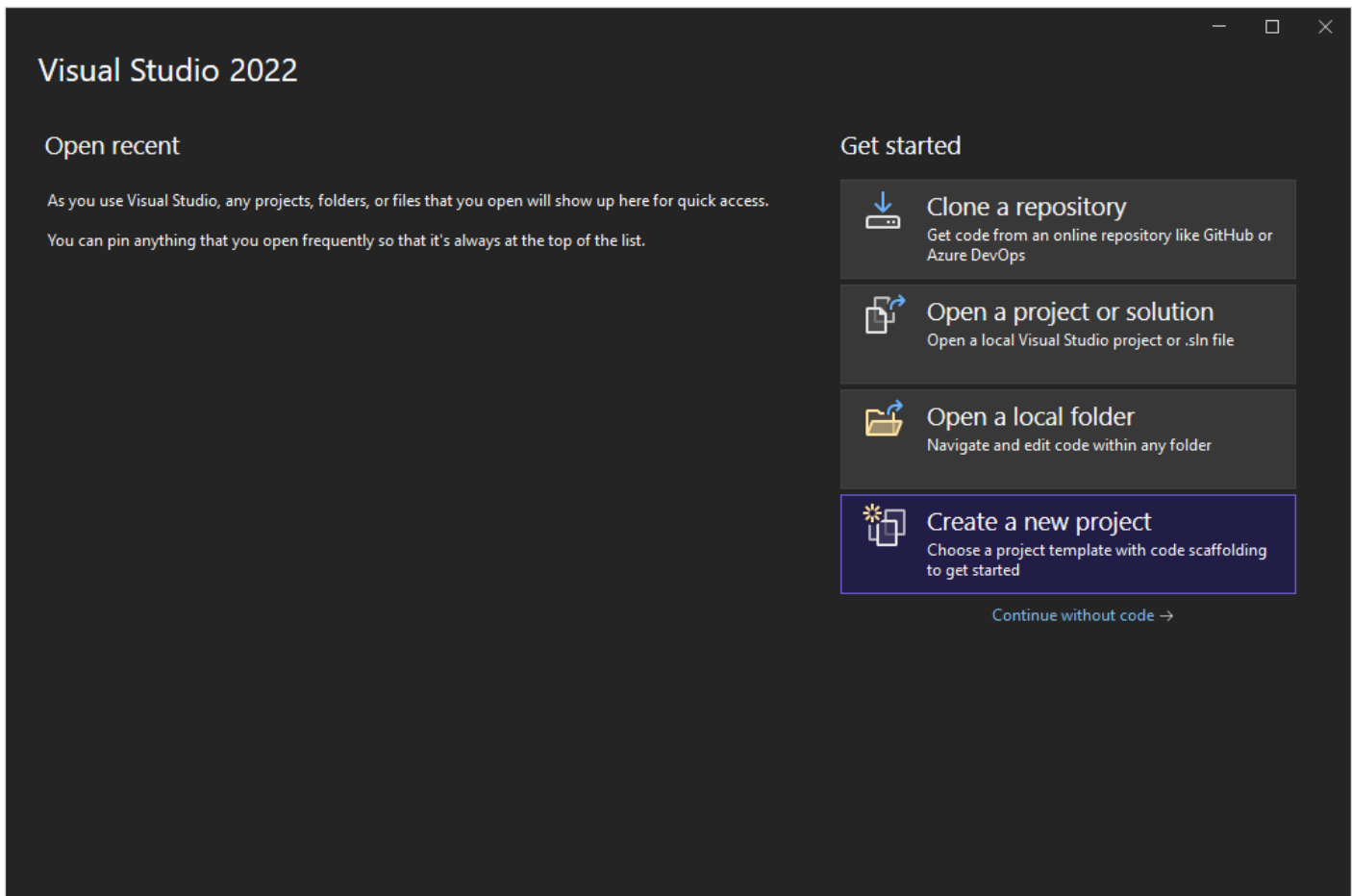


Figure 3: The Visual Studio Getting Started screen.

- From the Create a New Project Screen, scroll down the project template list and select the C# project called “Console App (.NET Framework)” then click Next. You can also enter the template name into the Search box at the top of the screen to make finding it quicker.

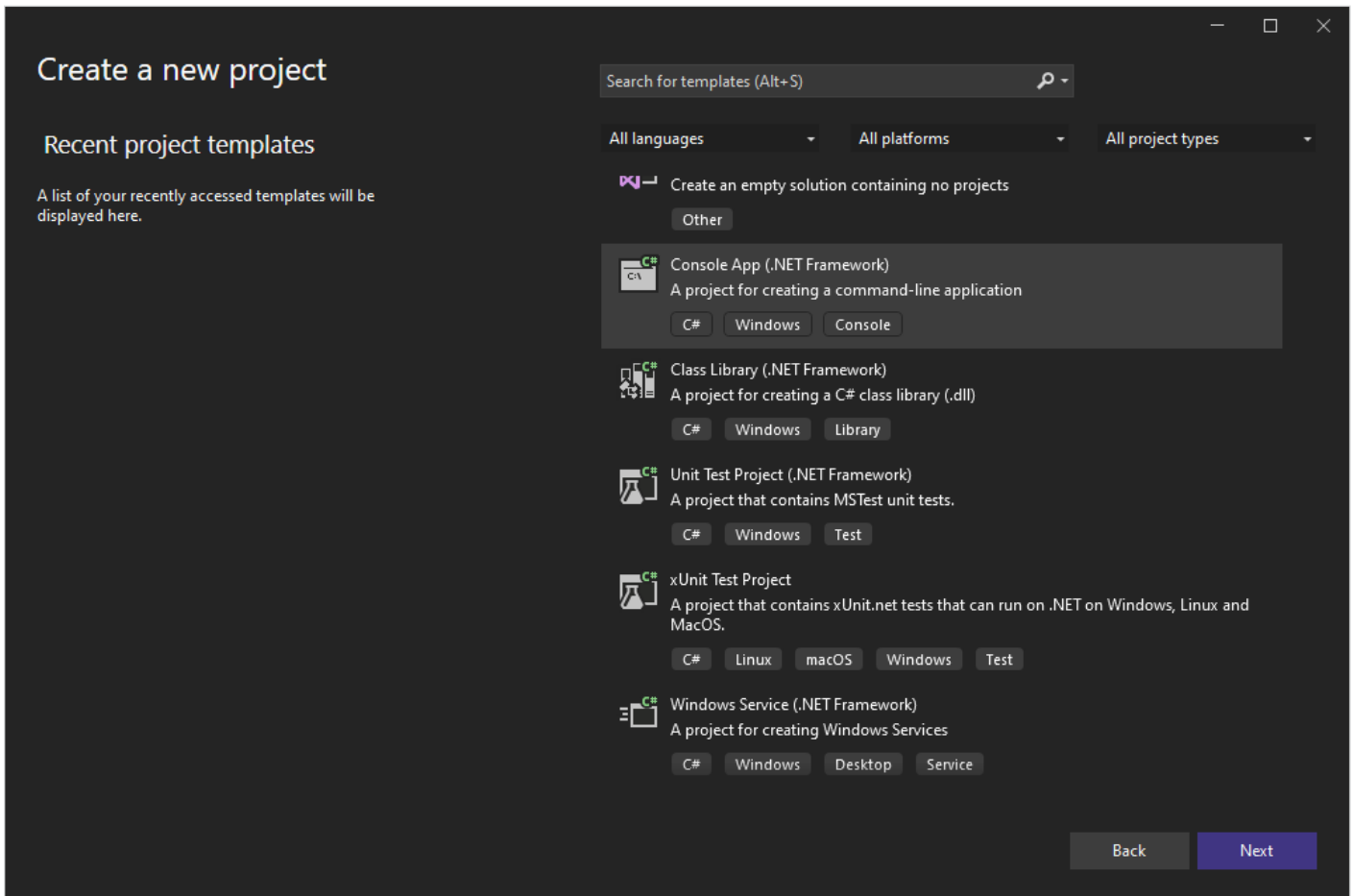


Figure 4: Select the type of project you want and click Next. Visual Studio will create a new project based on the selected template.

Note: The project list will contain a similar C# project that is just called “Console Project.” This is not the correct project and selecting it will create a console project that uses .NET Core instead of .NET framework. The IVI VISA .NET library is built on the .NET Framework, not .NET Core so it is important that you choose the .NET Framework based C# Console project.

3. Give the project a name and select a file location to store the project in.

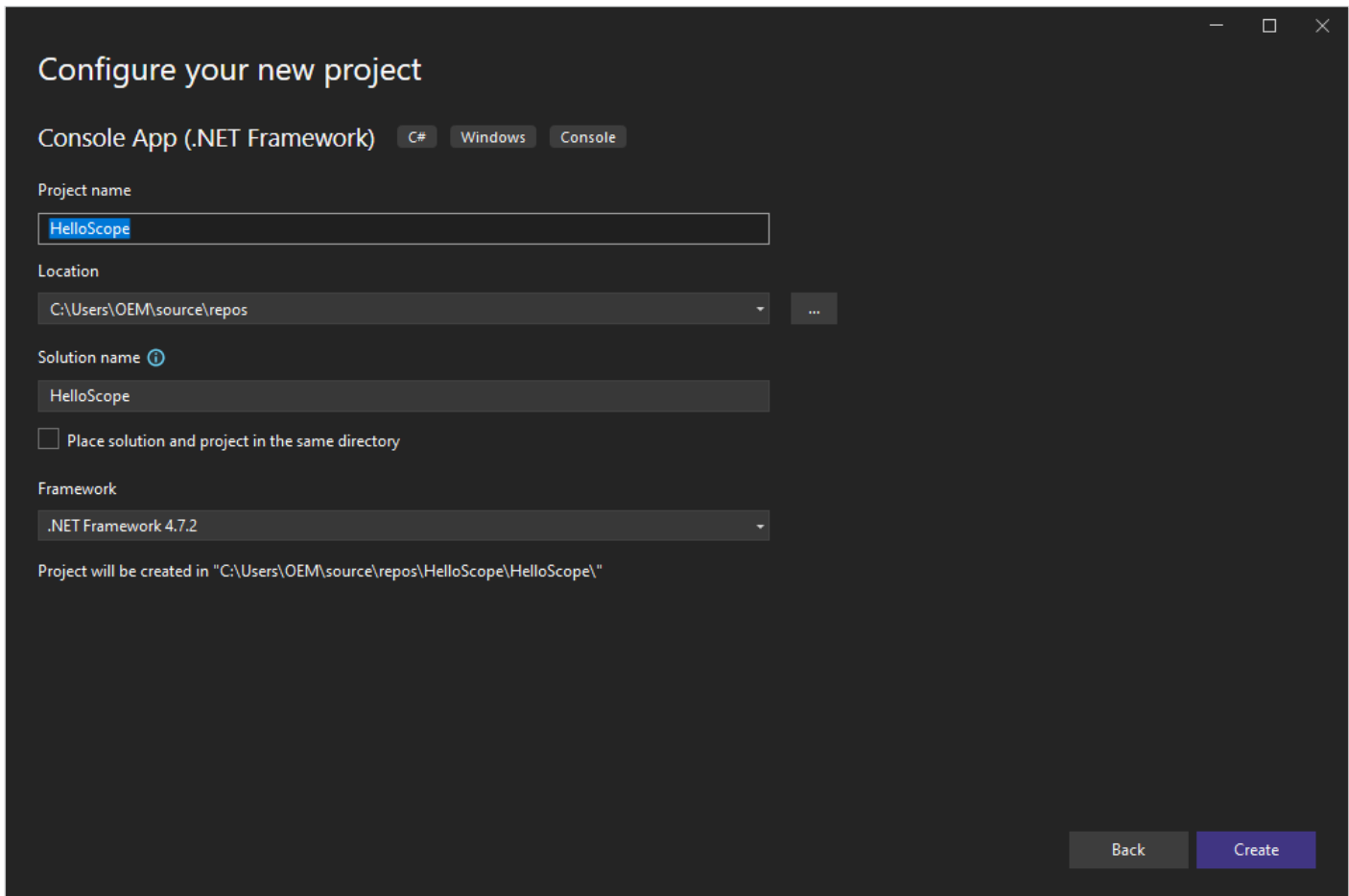


Figure 5: Enter the project's name into the wizard. You can also change the location of the project here.

4. In the Framework drop-down, make sure .NET Framework 4.7.2 is selected then click the Create button to create the project.

After Visual Studio creates the project, you will be presented with the full Visual Studio interface for editing the project. The main code file for the project, “Program.cs” will be open in code editor and the Solution Explorer pane, which provides access to the Properties, References and files in the project, can be accessed. Before we start adding code, we need to prepare our project by adding a reference to VISA to our code.

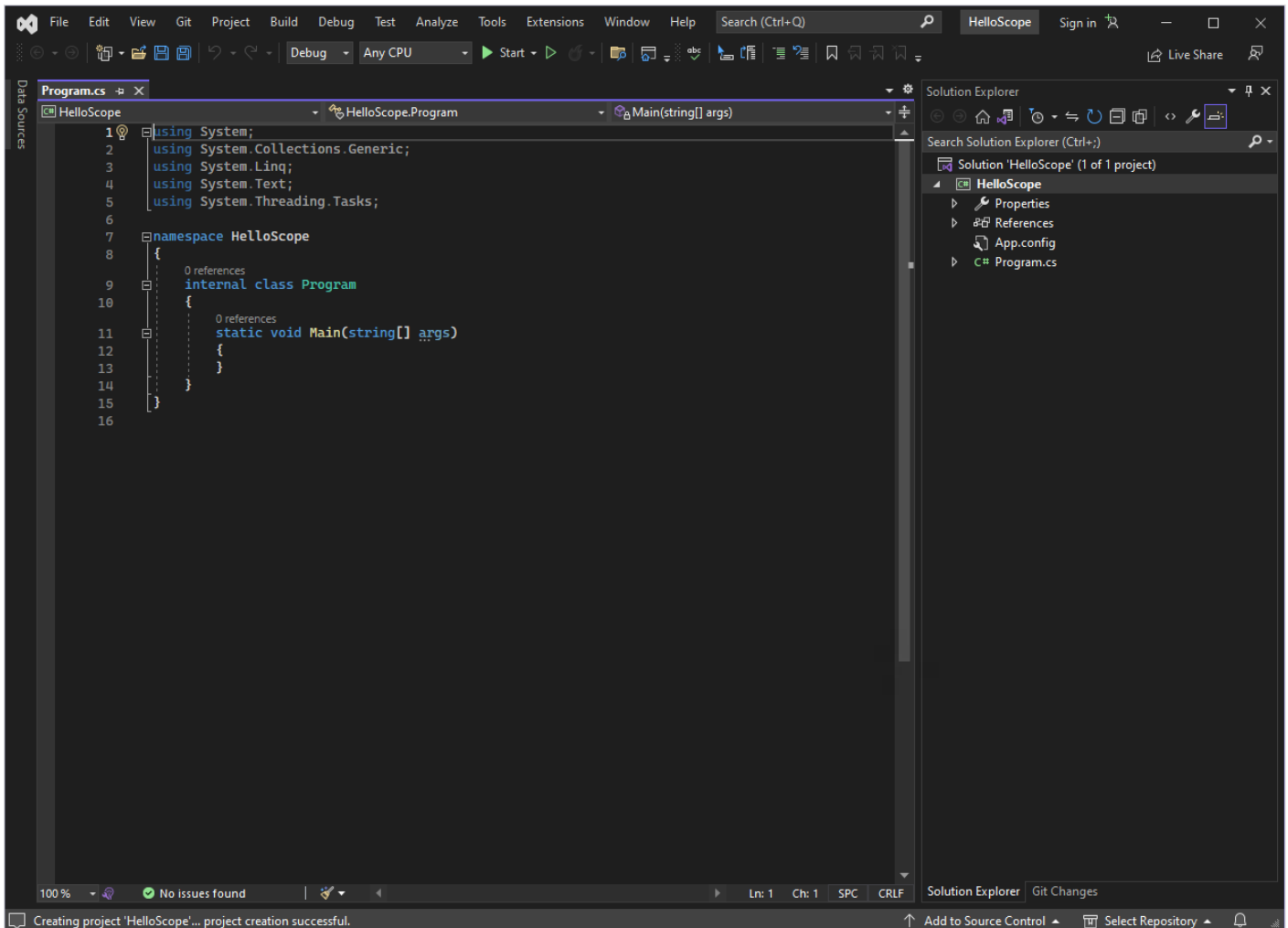


Figure 6: The Visual Studio Integrated Development Environment.

5. Our code will communicate with instruments by using the IVI VISA .NET library which was installed as part of the NI-VISA installer. Before we can use this library in our code, we first need to add a reference to it in our project. To add the reference, go into the Solution Explorer pane, right-click on References and select from the menu Add Reference...

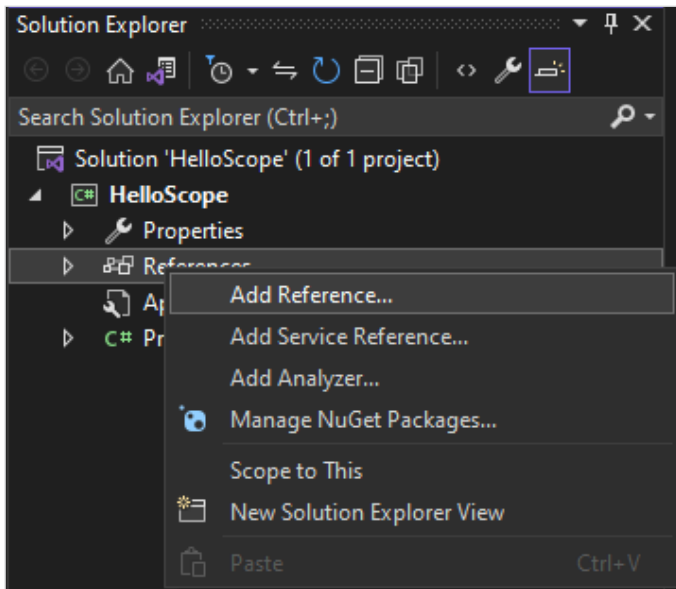


Figure 7: Add a reference to VISA by right-click References in the Solution Explorer and select Add Reference...

6. In the Reference Manager window, under Assemblies, click on “Extensions”. Scroll through the list and find the assembly named “Ivi.Visa Assembly” and click the checkbox next to it to select it. Click OK to add the reference to the project.

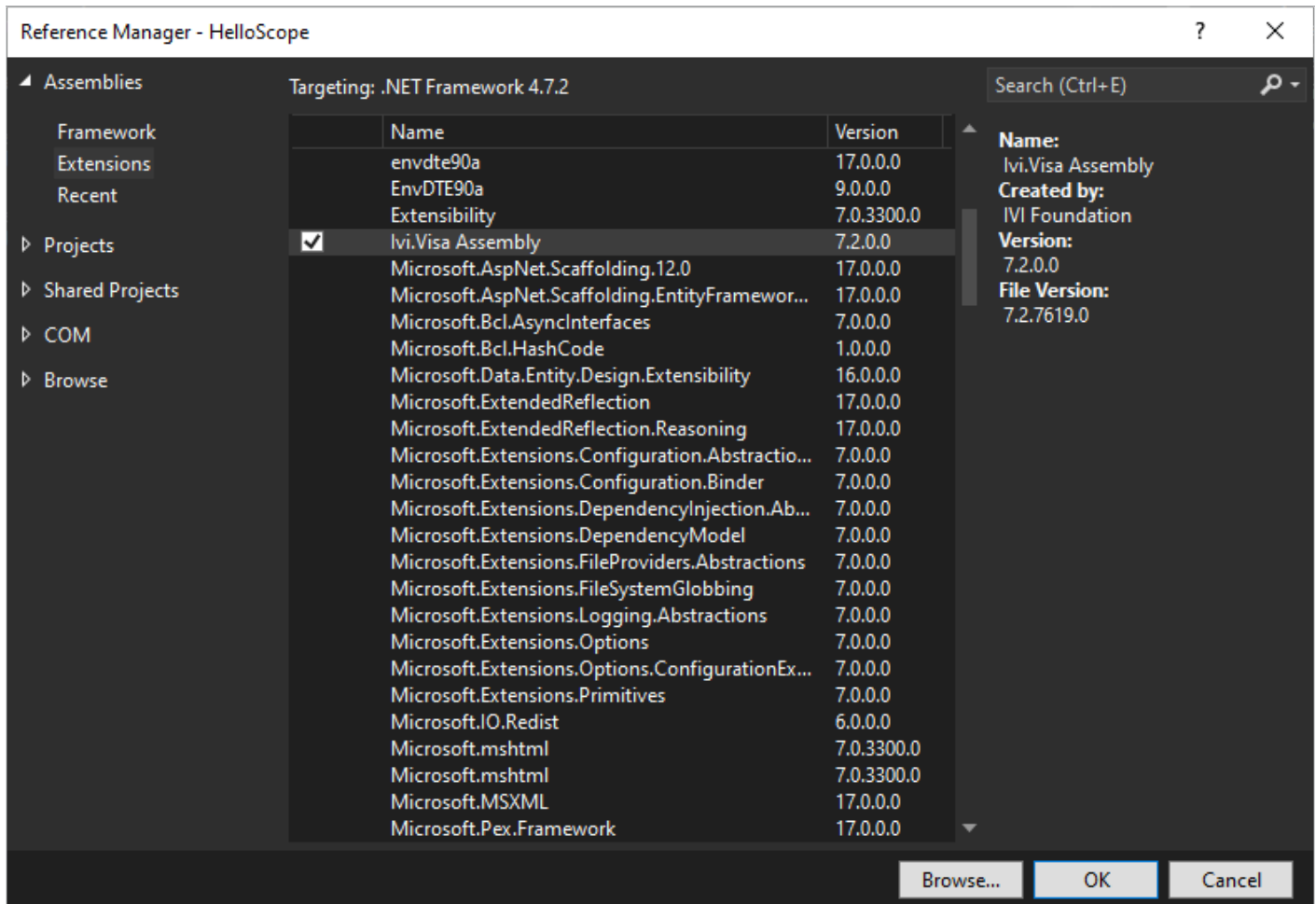


Figure 8: Add a reference to Ivi.Visa Assembly.

Question: Why did we add a reference to Ivi.Visa and not to NI-VISA?

Answer: The IVI VISA .NET library is a standardized .NET library for instrument control that is vendor agnostic. This means that any program written to use the IVI VISA .NET library can be used with any vendor's VISA implementation if that implementation supports the IVI standard VISA .NET interface.

With the reference to the IVIVISA .NET library added, we are now ready to begin writing code.

7. Go to the open Program.cs file in the code editor and at the top of the file you will see several “using” statements. After the last using statement add a new line and enter

8. `using Ivi.Visa;`

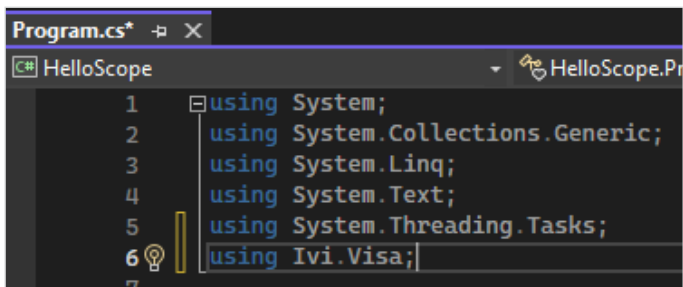


Figure 9: Using statements reduces the amount of typing needed when writing code and help direct the code editor.

This line allows us to access the objects contained in the Ivi.Visa namespace without having to type the entire namespace each time we declare or use one of these objects. This not only reduces the amount of typing, but it also helps the editor to make autocomplete suggestions as you type.

9. Further down in the file you will see where the static method `Main(string[] args)` is declared and followed by a pair of ellipsis. Between the ellipsis add the following code.

```
1. string visaRsrcAddr = "TCPIP::192.168.1.2::inst0::INSTR";
2. var scope = GlobalResourceManager.Open(visaRsrcAddr) as IMessageBasedSession;
3. using (scope)
4. {
5.     scope.FormattedIO.WriteLine("*IDN?");
6.     Console.WriteLine(scope.FormattedIO.ReadLine());
7.
8.     Console.WriteLine("Press the Enter key to continue.");
9.     Console.ReadLine();
10. }
```

The code we added will open a connection to the instrument using VISA, send the query command `*IDN?` to the instrument and then readback the response from the instrument and print it to the console. The program will then prompt us to press the Enter key to continue and then will wait until Enter is pressed.

The `using` statement around the `scope` object on line 3 in the code snippet above ensures that if any Exceptions are thrown by our code when it runs, that the connection will still be properly closed before the program quits.

10. In the line where `string visaRsrcAddr` is declared and assigned, edit the string to match the VISA Resource Address of your instrument.

11. Now that we have added some code to the file, we are ready to run our program. Click the Run button in the menu bar or press F5 to quickly compile and run our code. When the code runs you should see output in the console window that looks similar to the following.

A screenshot of a Windows console window. The title bar shows the file path: C:\Users\OEM\source\repos\HelloScope\HelloScope\bin\Debug\HelloScope.exe. The console output consists of two lines: 'TEKTRONIX,MSO68B,Q000048,CF:91.1CT FV:1.44.3.433' and 'Press the Enter key to continue.' The console background is black with white text.

Figure 10: The output from our basic HelloScope example.

Note: If the code failed and threw an exception, the most common reason is because VISA was unable to connect to the instrument. This is usually because the VISA Resource Address was entered incorrectly or because the instrument is no longer connected or turned on.

All right! Your program was able to connect to the instrument, send a command to query its ID and then read it back. This is great, but overall, it isn't a very useful application. Let add some more code to this example and actually do something with the oscilloscope.

12. Modify your code to look like the following.

```
1. using System;
2. using System.Collections.Generic;
3. using System.Linq;
4. using System.Text;
5. using System.Threading.Tasks;
6. using Ivi.Visa;
7.
8. namespace HelloScope
9. {
10.     internal class Program
11.     {
12.         static void Main(string[] args)
13.         {
14.             // Edit this to match the VISA Resource Address of your instrument
15.             string visaRsrcAddr = "TCPIP::192.168.1.2::inst0::INSTR";
16.
17.             // Open a connection to the instrument located at the visaRsrcAddr string
18.             // Cast the returned object as an IMessageBasedSession. This is the interface of the
19.             // IVI VISA library used to send and receive commands and data.
20.             var scope = GlobalResourceManager.Open(visaRsrcAddr) as IMessageBasedSession;
21.
22.             // using statement ensures that the connection will be closed even if an exception is thrown.
23.             using (scope)
24.             {
25.                 // Query instrument ID and print response to console
26.                 scope.FormattedIO.WriteLine("*IDN?");
27.                 Console.WriteLine(scope.FormattedIO.ReadLine());
28.
29.                 // Reset the instrument to default state and wait for it to complete
30.                 Console.WriteLine("Resetting instrument...");
31.                 scope.FormattedIO.WriteLine("*RST");
32.                 scope.FormattedIO.WriteLine("*OPC?");
33.                 scope.RawIO.ReadString();
34.                 Console.WriteLine("Done!");
35.
36.                 // Perform an Autoset and wait for it to complete
37.                 Console.WriteLine("Autoset instrument...");
38.                 scope.FormattedIO.WriteLine("AUTOSET EXECUTE");
39.                 scope.FormattedIO.WriteLine("*OPC?");
40.                 scope.RawIO.ReadString();
41.                 Console.WriteLine("Done!");
42.
43.                 // Add an amplitude measurement and stop acquiring
44.                 scope.FormattedIO.WriteLine("MEASU:ADDMEAS AMPLITUDE");
```

```

45.         scope.FormattedIO.WriteLine("ACQ:STATE STOP");
46.         scope.FormattedIO.WriteLine("*OPC?");
47.         scope.RawIO.ReadString();
48.
49.         // Initiate a single acquisition and wait for it to complete
50.         Console.Write("Performing Single Sequence...");
51.         scope.FormattedIO.WriteLine("ACQ:STOPAFTER SEQUENCE");
52.         scope.FormattedIO.WriteLine("ACQ:STATE RUN");
53.         scope.FormattedIO.WriteLine("*OPC?");
54.         scope.RawIO.ReadString();
55.         Console.WriteLine("Done!\r\n");
56.
57.         // Fetch the measurement result and print to console
58.         scope.FormattedIO.WriteLine("MEASU:MEAS1:RESULTS:CURRENTACQ:MEAN?");
59.         float ampl = float.Parse(scope.FormattedIO.ReadLine());
60.         Console.WriteLine($"Signal Amplitude: {ampl} Volts\r\n");
61.
62.         Console.WriteLine("Press the Enter key to continue.");
63.         Console.ReadLine();
64.     }
65. }
66. }
67. }

```

Now your code will do the following:

1. Connect to the oscilloscope
2. Query its ID and print it to the console
3. Reset the oscilloscope to its default state
4. Autoset the oscilloscope
5. Add an amplitude measurement
6. Acquire a single sequence
7. Fetch the measured amplitude value and print it to the console

Note: The example code listed above is designed for use with [Tektronix 2/4/5/6 Series MSO Mixed Signal Oscilloscopes](#). To make this code work with 3 Series MDO, MSO/DPO5000 B, DPO7000 C, MSO/DSA/DPO70000 B C D DX, DPO70000SX Series Oscilloscopes, make the following changes.

Replace the line

```
scope.FormattedIO.WriteLine("MEASU:ADDMEAS AMPLITUDE");
```

with

```
scope.FormattedIO.WriteLine("MEASU:IMM:TYPE AMPLITUDE");
```

and replace the line

```
scope.FormattedIO.WriteLine("MEASU:MEAS1:RESULTS:CURRENTACQ:MEAN?");
```

with

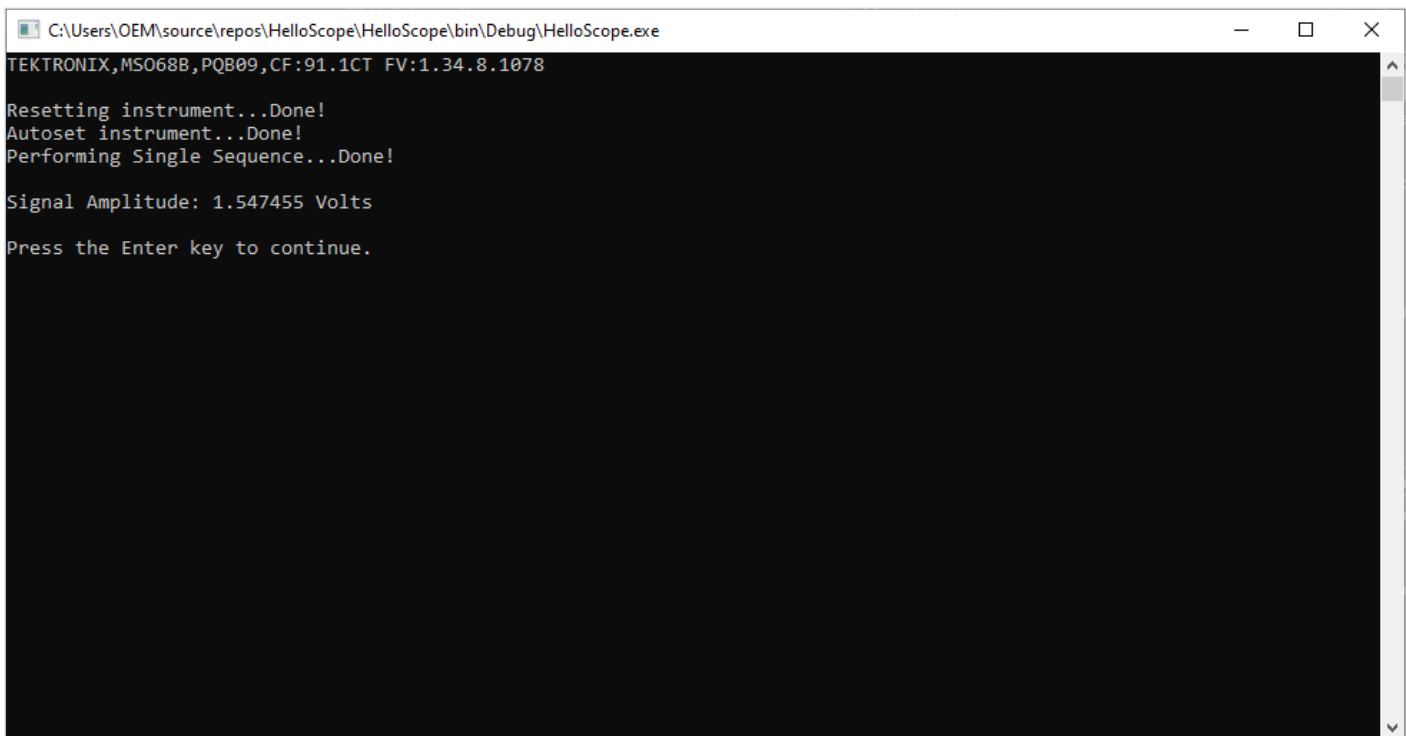
```
scope.FormattedIO.WriteLine("MEASU:IMM:VAL?");
```

Notice that the code includes the lines

```
scope.FormattedIO.WriteLine(" *OPC? ");  
scope.RawIO.ReadString();
```

after several of the operations. This is the Operation Complete query command and it is used to keep the code synchronized with the oscilloscope operations. Certain long running oscilloscope operations like performing a reset, autoset or acquiring a single sequence will cause the oscilloscope to lower the Operation Complete Flag in the oscilloscope status and raise it when the operation has completed. The *OPC? command is a blocking command that will not return a response until the OPC flag is set high. By querying *OPC? we can block our code from continuing until the command returns a response.

Once you have finished editing your code, click the Run button to compile and run the code. If everything is successful, the output of your program should look like the following.



```
C:\Users\OEM\source\repos\HelloScope\HelloScope\bin\Debug\HelloScope.exe  
TEKTRONIX,MSO68B,PQB09,CF:91.1CT FV:1.34.8.1078  
Resetting instrument...Done!  
Autoset instrument...Done!  
Performing Single Sequence...Done!  
Signal Amplitude: 1.547455 Volts  
Press the Enter key to continue.
```

Figure 11: The output from our longer HelloScope example.

Congratulations! You have successfully written a program using C# that connects to and instrument, controls it and reads back data from it. You are now ready to start developing your own advanced instrument control applications.

Pulling Examples from GitHub

To aid in learning to write programs to control Tektronix instruments, Tektronix has made available many example programs on the Tektronix GitHub in the Programmatic Control Examples repository. This repository can be found at <https://github.com/tektronix/Programmatic-Control-Examples>. For the next example we will pull the code from the Tektronix GitHub at the URL above. Use the following step to get a copy of this repository on to your computer.

1. Go to the Tektronix Programmatic-Control-Examples repository at the URL above.
2. Clone the repository using Git or download it as a ZIP file and extract it to your PC. You can find the information needed to clone or download the repository by clicking on the green <> Code button on the web page of the repo.

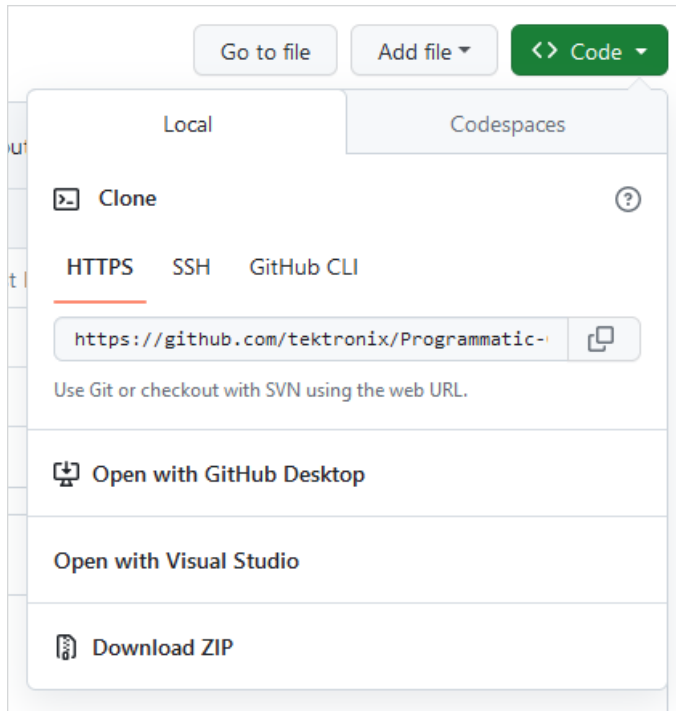


Figure 12: Cloning or downloading the GitHub repository can be accessed from the Green <> Code button on the repo's main page.

Curve Query C# Windows Forms Example

For this example, rather than starting from scratch, we will be pulling the code from the [Tektronix GitHub repository](#). If you have not completed the steps above in Pulling Examples from GitHub, please do so now.

This example demonstrates how to create an automated test and measurement application with a graphical user interface that will fetch a waveform from an oscilloscope and display it on the user interface. This example uses the C# Windows Forms (.NET Framework) project type in Visual Studio to create a program with a Windows Forms GUI, the IVI VISA .NET library for communications and the OxyPlot graphing library for displaying the waveform data on the user interface. OxyPlot is installed in the project using the built-in NuGet package manager in Visual Studio and the library will be downloaded automatically when you compile the project.

Note: This project is designed to work with Tektronix 2/4/5/6 Series MSO Mixed Signal Oscilloscopes, 3 Series MDO Mixed Domain Oscilloscopes and Tektronix MSO/DPO5000 B, DPO7000 C, MSO/DPO70000 B C, MSO/DPO/DSA70000 D DX and DPO70000SX Series Oscilloscopes. It may work with other Tektronix Oscilloscope Series as well (MDO/MSO/DPO3000/4000, 3 Series MDO, etc.), but has not been tested.

1. After you have cloned, or downloaded as a ZIP and extracted, the Tektronix Programmatic-Control-Examples repo to your computer, open the folder containing the files in Windows Explorer and use the search bar in Windows Explorer to find the folder named "CSharpCurveQueryWinforms".
2. Inside the CSharpCurveQueryWinforms folder, open the file "CurveQueryWinforms.sln" in Visual Studio.
3. After the project loads in Visual Studio, go to the Solution Explorer pane and double-click on the file named "CurveQueryMain.cs". This will load the Windows Forms graphical user interface for this example program inside the visual editor.
4. In the visual editor, on the main form, double-click on the button labeled "Get Waveform". This will open the code editor and go directly to the method that contains the code that will run when you click on the Get Waveform button. Inside this method you will find the code that connects to the instrument, fetches the waveform data, processes it, and then displays it on screen.
5. Click the Run button in Visual Studio to compile and run the code.
6. When the program has loaded, enter the VISA Resource Name of your instrument into the text box labeled VISA Resource Name and select a channel to fetch.
7. On the oscilloscope to which you will connect, make sure it has acquired a waveform on the channel you selected earlier then click the Get Waveform button in the Curve Query Example GUI.

The program will connect to the instrument, query its ID and then fetch the waveform data from the channel and display it on screen.

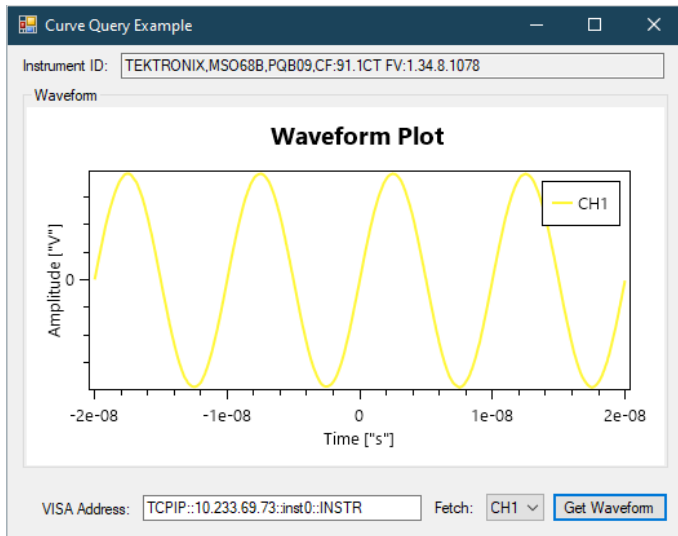


Figure 13: The Curve Query Example will fetch waveform data from the oscilloscope and display it on screen.

Taking the Next Steps

It is common for developers to copy and paste code from examples; this not only saves time but also helps them learn along the way. Browse the code examples on the Tektronix Github for finished solutions and inspiration!

C# is an excellent language for building automated test and measurement applications. Instrument communication support through the IVI VISA.NET library makes controlling and instrument through its remote programable interface a breeze. The Visual Studio integrated development environment is user-friendly and offers powerful functionality that makes it easier to write and debug code in C#. With its clean syntax and extensive library support, C# enables engineers to write code that is both efficient and maintainable.

Contact Information:

Australia 1 800 709 465
Austria* 00800 2255 4835
Balkans, Israel, South Africa and other ISE Countries +41 52 675 3777
Belgium* 00800 2255 4835
Brazil +55 (11) 3530-8901
Canada 1 800 833 9200
Central East Europe / Baltics +41 52 675 3777
Central Europe / Greece +41 52 675 3777
Denmark +45 80 88 1401
Finland +41 52 675 3777
France* 00800 2255 4835
Germany* 00800 2255 4835
Hong Kong 400 820 5835
India 000 800 650 1835
Indonesia 007 803 601 5249
Italy 00800 2255 4835
Japan 81 (3) 6714 3086
Luxembourg +41 52 675 3777
Malaysia 1 800 22 55835
Mexico, Central/South America and Caribbean 52 (55) 88 69 35 25
Middle East, Asia, and North Africa +41 52 675 3777
The Netherlands* 00800 2255 4835
New Zealand 0800 800 238
Norway 800 16098
People's Republic of China 400 820 5835
Philippines 1 800 1601 0077
Poland +41 52 675 3777
Portugal 80 08 12370
Republic of Korea +82 2 565 1455
Russia / CIS +7 (495) 6647564
Singapore 800 6011 473
South Africa +41 52 675 3777
Spain* 00800 2255 4835
Sweden* 00800 2255 4835
Switzerland* 00800 2255 4835
Taiwan 886 (2) 2656 6688
Thailand 1 800 011 931
United Kingdom / Ireland* 00800 2255 4835
USA 1 800 833 9200
Vietnam 12060128

* European toll-free number. If not accessible, call: +41 52 675 3777

Rev. 02.2022

Find more valuable resources at TEK.COM

Copyright © Tektronix. All rights reserved. Tektronix products are covered by U.S. and foreign patents, issued and pending. Information in this publication supersedes that in all previously published material. Specification and price change privileges reserved. TEKTRONIX and TEK are registered trademarks of Tektronix, Inc. All other trade names referenced are the service marks, trademarks or registered trademarks of their respective companies.

7/2423 SBG 61W-74018-0

