Getting Started with Instrument Control Using C#

Introduction

As a user of test and measurement instrumentation, you rely on your hardware to achieve the best possible measurement quality. After unpacking and installing your equipment, you walk through the steps provided in a quick start guide to help provide no less than the cursory training which provides you the hands-on knowledge needed to establish your test scenario setup and execution. At some point you will find that your basic setup evolves into an established system which performs a variety of routine tasks. Instead of relying on rote memory of procedures, you will likely look towards options to automate the instrument control, test execution and evaluation, and on to data archival and analysis – all of this to be handled by software.

This document will offer a cursory introduction to getting started with using C# through Microsoft's Visual Studio software development package on a Windows PC to perform test automation with practically any piece of instrumentation in your desired system or lab inventory. For simplicity, we will use a Bird BNA1000 Vector Network Analyzer. We will first offer some insight into the benefits of C# and why you might prefer this software option over others. Then we will provide general details on acquiring and license-based usage of the Visual Studio software. We will then cover installation of the recommended software options. Finally, we share some of the basic building blocks for connecting to and controlling your instruments using an examples-based approach. Finally, we share some notes on instrument drivers.

Reasons for Using C#

C# is a popular and powerful option from Microsoft as part of their Visual Studio suite of software development products for a wide variety of uses inclusive of test, measurement, and automation applications. There are several reasons why C# is often chosen for test automation, but standing out among them is that its users can create a Windows Forms application to design graphical user interfaces (GUIs) that can be tailored to each specific application's needs. The graphical nature of a C# Forms application allows users to visually design the flow of their programs. Visual Studio provides a range of data visualization tools, including charts, graphs, and customizable user interfaces, making it easier to interpret and display test results. This is especially important when presenting data to non-technical stakeholders.

C# is an object-oriented programming (OOP) language which promotes an organized, modular approach to programming. Engineers can create reusable classes that can be easily integrated into larger systems. This modularity simplifies test system maintenance and enhances code reusability. Furthermore, Visual Studio offers a vast library of pre-built libraries to help with tasks such as signal processing, control, and analysis. These libraries can save time and effort by providing ready-made solutions to common testing and measurement challenges.

Looking to become a Visual Studio expert? Microsoft (MS) offers certification programs and extensive training options to help engineers and technicians become proficient in Visual Studio, ensuring that they can effectively implement test automation solutions.

Preparing Your System for C# Usage

For those who are just getting started and not looking to use the software as a full-time commitment, consider using the Visual Studio (VS) 2022 Community Edition. This free-to-use version provides you with all the capabilities found in the Visual Studio Pro editions, including the LINX toolkit for use with Raspberry Pi, BeagleBoard, and Arduino along with access to the web development software (named G) for creating web-based applications when you need them. Bear in mind that all of this is offered to you in the VS Community Edition so long as you adhere to at least the following:

- Only for personal, non-commercial use. You can build your own knowledge base, but it is expected that you are not developing code *for profit*.
- You cannot use the software "for teaching or research at a degree-granting educational institution". Microsoft may have discount offers specifically for colleges and universities, so it's best to speak with a sales specialist about this.

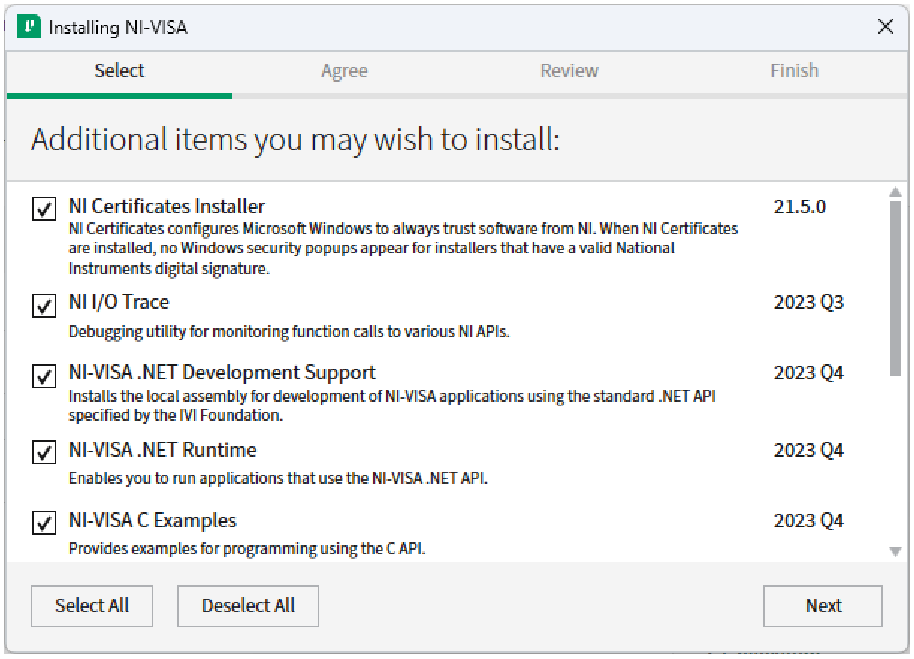
For more detail, see the Microsoft Visual Studio Community 2022 Software Usage Terms.

First Install NI-VISA

Before you download and install Visual Studio, it is highly recommended that you first download and install the National Instruments (NI) version of the **Virtual Instrument Software Architecture**, or **NI-VISA**. This will provide you with a great foundation for extensive instrument communication as well as some debugging tools. You can download NI-VISA from this landing page.

Note that if you do not already have a registered user account then you will need to create one.

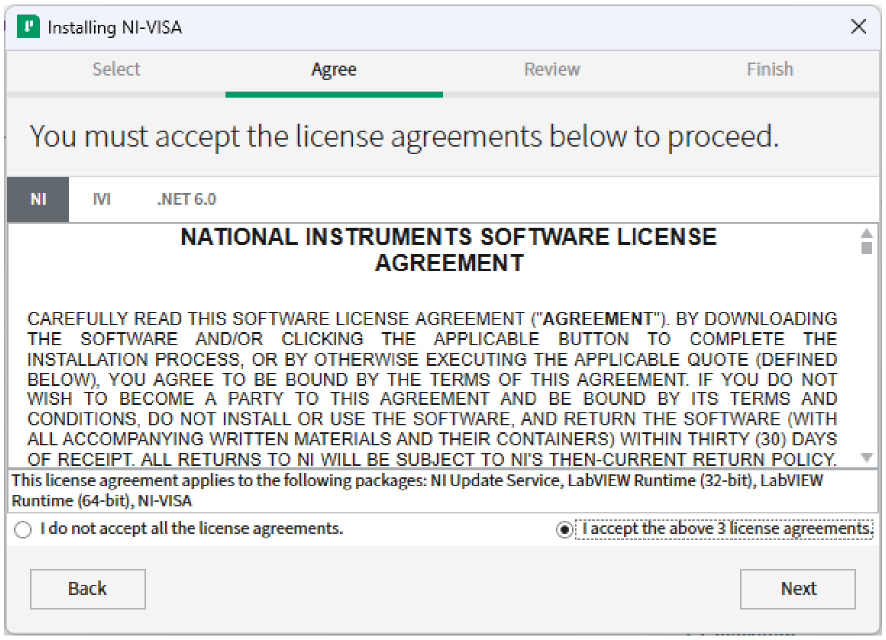
When the installer launches, it will offer you a few additional items. Note that it is necessary to ensure the ***NI-VISA .NET Development Support*** option is selected – it should be by default but it is important to verify. Select all other items you are interested in – if in doubt, just keep all the defaults – and accept these by clicking **Next**.



Additional items you may wish to install:

✓	NI Certificates Installer NI Certificates configures Microsoft Windows to always trust software from NI. When NI Certificates are installed, no Windows security popups appear for installers that have a valid National Instruments digital signature.	21.5.0
✓	NI I/O Trace Debugging utility for monitoring function calls to various NI APIs.	2023 Q3
✓	NI-VISA .NET Development Support Installs the local assembly for development of NI-VISA applications using the standard .NET API specified by the IVI Foundation.	2023 Q4
✓	NI-VISA .NET Runtime Enables you to run applications that use the NI-VISA .NET API.	2023 Q4
✓	NI-VISA C Examples Provides examples for programming using the C API.	2023 Q4

You will need to accept any/all license agreements in order to proceed. Read carefully. If there's anything you do not agree with, then discontinue installation, delete the downloaded installer (likely housed in your *Downloads* folder), and discard this document. Otherwise, click **Next**.


Installing NI-VISA

You must accept the license agreements below to proceed.

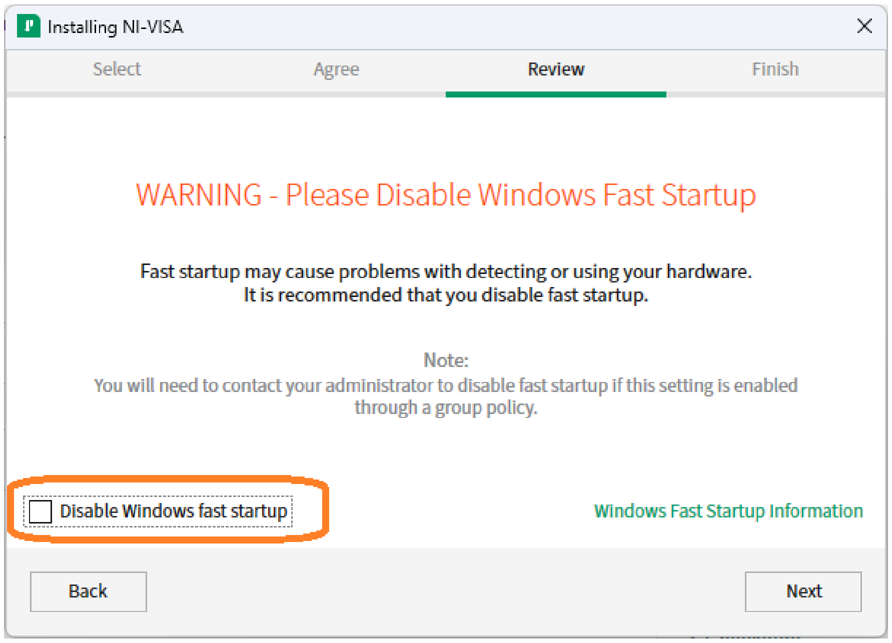
NATIONAL INSTRUMENTS SOFTWARE LICENSE AGREEMENT

CAREFULLY READ THIS SOFTWARE LICENSE AGREEMENT ("**AGREEMENT**"). BY DOWNLOADING THE SOFTWARE AND/OR CLICKING THE APPLICABLE BUTTON TO COMPLETE THE INSTALLATION PROCESS, OR BY OTHERWISE EXECUTING THE APPLICABLE QUOTE (DEFINED BELOW), YOU AGREE TO BE BOUND BY THE TERMS OF THIS AGREEMENT. IF YOU DO NOT WISH TO BECOME A PARTY TO THIS AGREEMENT AND BE BOUND BY ITS TERMS AND CONDITIONS, DO NOT INSTALL OR USE THE SOFTWARE, AND RETURN THE SOFTWARE (WITH ALL ACCOMPANYING WRITTEN MATERIALS AND THEIR CONTAINERS) WITHIN THIRTY (30) DAYS OF RECEIPT. ALL RETURNS TO NI WILL BE SUBJECT TO NI'S THEN-CURRENT RETURN POLICY.

This license agreement applies to the following packages: NI Update Service, LabVIEW Runtime (32-bit), LabVIEW Runtime (64-bit), NI-VISA

I do not accept all the license agreements. [✓ I accept the above 3 license agreements.]

You are then given the option to disable Windows Fast Startup. While it is recommended by NI, if you are installing on a computer controlled by an IT department, you may not have the option to do this. If in doubt, uncheck the option and click **Next** to continue.


Installing NI-VISA

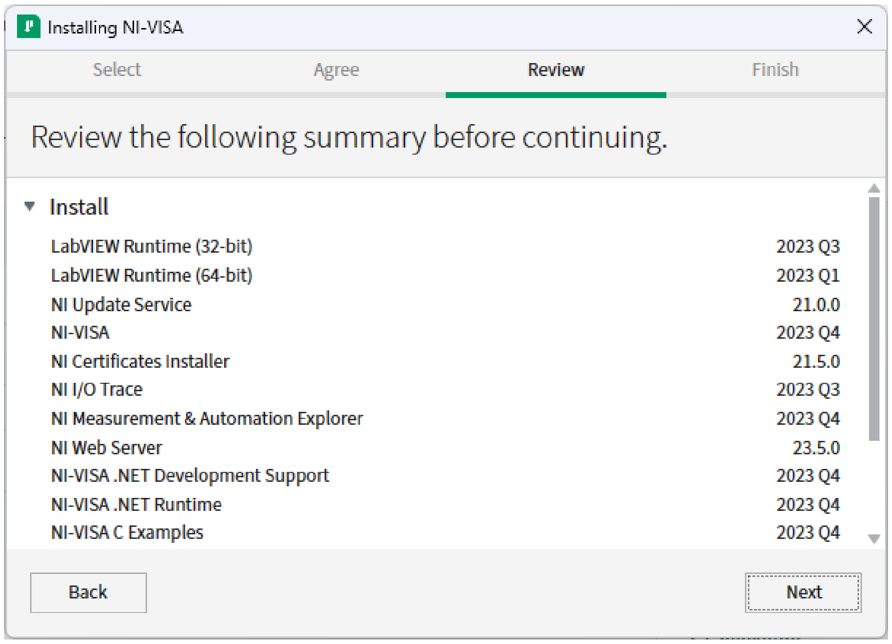
WARNING - Please Disable Windows Fast Startup

Fast startup may cause problems with detecting or using your hardware.
It is recommended that you disable fast startup.

Note:
You will need to contact your administrator to disable fast startup if this setting is enabled through a group policy.

☐ Disable Windows fast startup

You will then be given the opportunity to review all the software options you have elected to install. Click **Next** to continue.

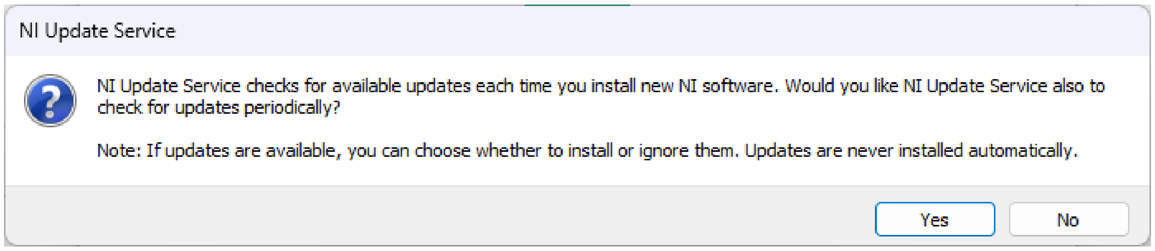

Installing NI-VISA

Review the following summary before continuing.

Install

LabVIEW Runtime (32-bit)	2023 Q3
LabVIEW Runtime (64-bit)	2023 Q1
NI Update Service	21.0.0
NI-VISA	2023 Q4
NI Certificates Installer	21.5.0
NI I/O Trace	2023 Q3
NI Measurement & Automation Explorer	2023 Q4
NI Web Server	23.5.0
NI-VISA .NET Development Support	2023 Q4
NI-VISA .NET Runtime	2023 Q4
NI-VISA C Examples	2023 Q4

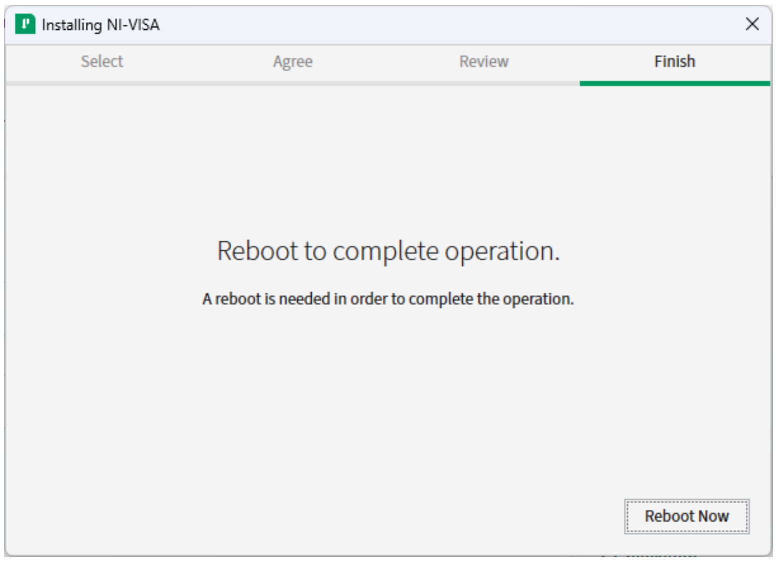
Near the end of the installation, you may be prompted to enable the NI Update Service which will run in the background after you power on your computer and periodically check for updates. You can select whether to apply updates automatically or be prompted to choose whether or not to install. If you are comfortable with the automatic update checking, click **Yes**, otherwise click **No**.


NI Update Service

NI Update Service checks for available updates each time you install new NI software. Would you like NI Update Service also to check for updates periodically?

Note: If updates are available, you can choose whether to install or ignore them. Updates are never installed automatically.

Finally, after successful installation of the VISA software you will be prompted to reboot your system. Click **Reboot Now**.


Installing NI-VISA

Reboot to complete operation.

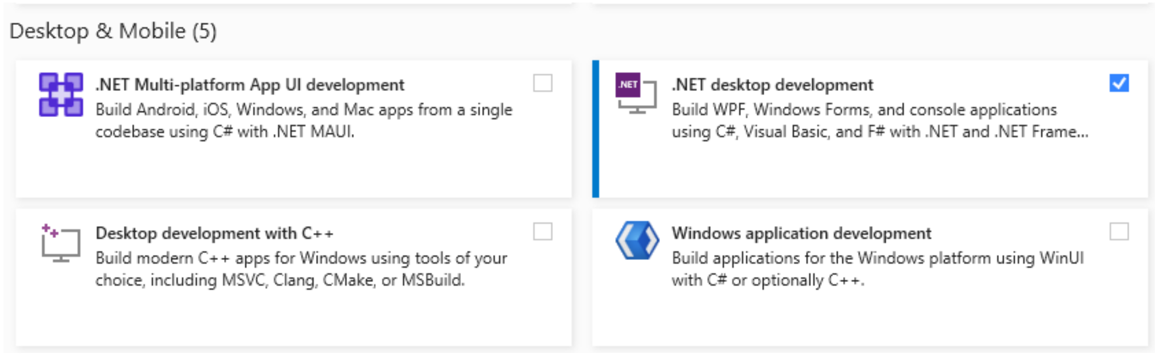
A reboot is needed in order to complete the operation.

Then install Visual Studio Community Edition

You can download the VS Community software from the Microsoft website on this landing page.

The download is a standard application setup installer executable – about 4 MB for the 2022 Community version. Double-clicking on the executable will start the install. *Fair warning: the install will require some patience on your part.*

When the installer launches, it will offer you a number of options to choose from. For the purposes of this tutorial, you will only need the **.NET desktop development** tools.



Desktop & Mobile (5)

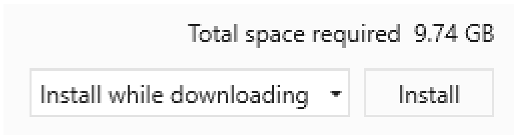
.NET Multi-platform App UI development [ ]
Build Android, iOS, Windows, and Mac apps from a single codebase using C# with .NET MAUI.

.NET desktop development [✓]
Build WPF, Windows Forms, and console applications using C#, Visual Basic, and F# with .NET and .NET Frame...

Desktop development with C++ [ ]
Build modern C++ apps for Windows using tools of your choice, including MSVC, Clang, CMake, or MSBuild.

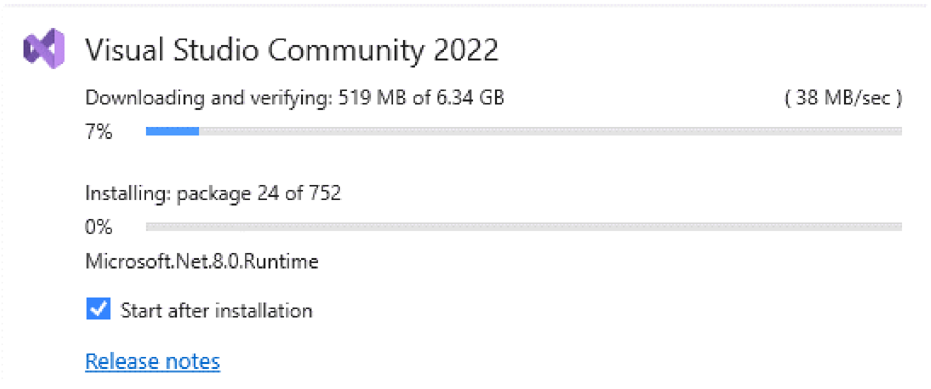
Windows application development [ ]
Build applications for the Windows platform using WinUI with C# or optionally C++.

In the bottom right of the installer you will be shown the amount of space required for the software tools and the options for installation that you can set depending on your PC usage needs. Go ahead and accept these by clicking **Install**.



Total space required 9.74 GB

You will need to accept any/all license agreements in order to proceed. Read carefully. If there’s anything you do not agree with, then discontinue installation, delete the downloaded installer (likely housed in your Downloads folder), and discard this document. Otherwise, click **Next**.



Visual Studio Community 2022

Downloading and verifying: 519 MB of 6.34 GB (38 MB/sec)
7%

Installing: package 24 of 752
0%
Microsoft.Net.8.0.Runtime

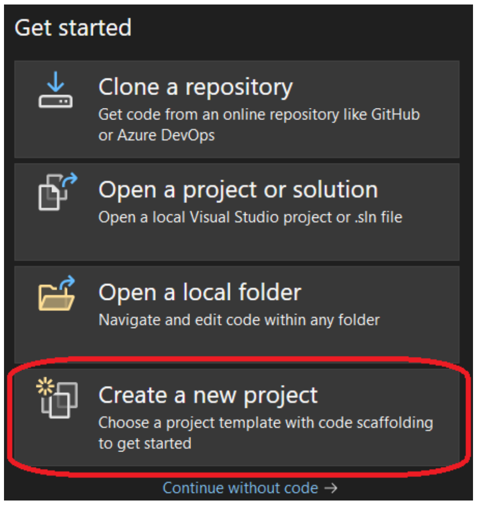
[✓] Start after installation

Once the installation is complete, you may be asked to reboot your PC. Go ahead and do so.

Starting with Your First Windows Console App

The first thing you will need to do to create your code is to start Visual Studio which you can do from the Windows Start menu, locating and selecting the Visual Studio 2022 application from the list of options.

After you are presented with the starting dialog, choose **Create a new project** from the series of buttons provided.



Get started

Clone a repository
Get code from an online repository like GitHub or Azure DevOps

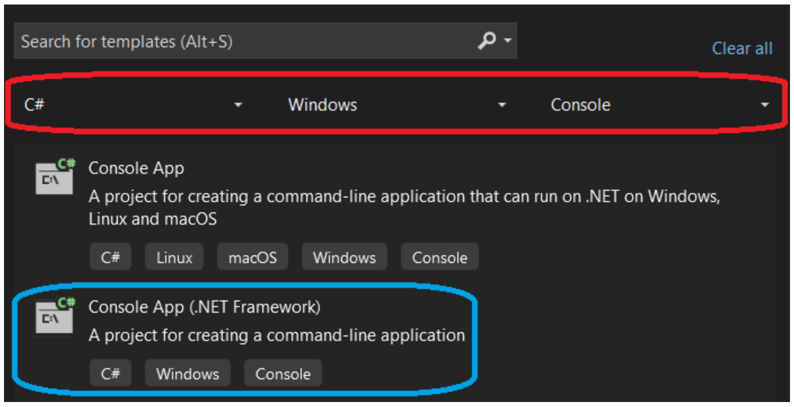
Open a project or solution
Open a local Visual Studio project or .sln file

Open a local folder
Navigate and edit code within any folder

Create a new project
Choose a project template with code scaffolding to get started

Continue without code →

The dialog will update to provide you the option to select what type of project you wish to create. For the purposes of this document, you will want to locate and select the **Console App (.NET Framework)**, and there are drop-down filters at the top of the selection column to help you locate this choice. Highlight then click **Next**.



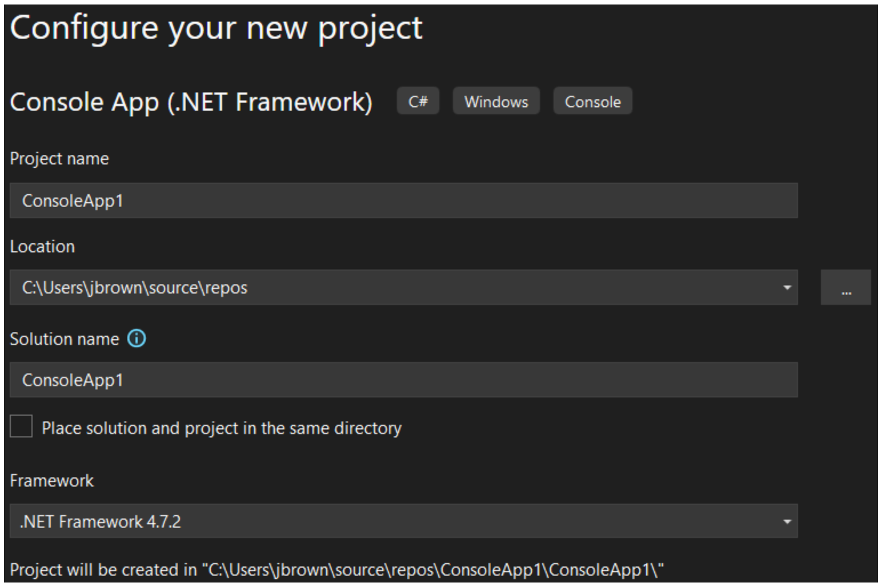
Search for templates (Alt+S) Clear all

C# Windows Console

Console App
A project for creating a command-line application that can run on .NET on Windows, Linux and macOS
C# Linux macOS Windows Console

Console App (.NET Framework)
A project for creating a command-line application
C# Windows Console

You will then be presented with a few final options; all the defaults can be accepted for this tutorial though you may wish to change the Project Name to something more to your liking. Once satisfied, click **Next**.



Configure your new project

Console App (.NET Framework)

Project name: ConsoleApp1

Location: C:\Users\jbrown\source\repos

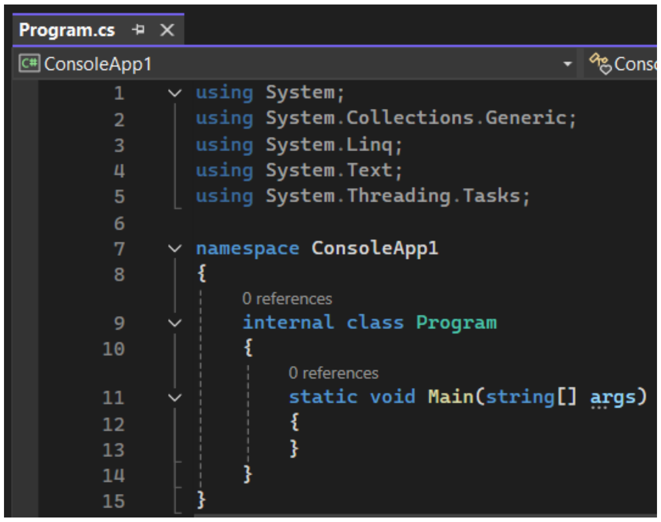
Solution name: ConsoleApp1

Place solution and project in the same directory

Framework: .NET Framework 4.7.2

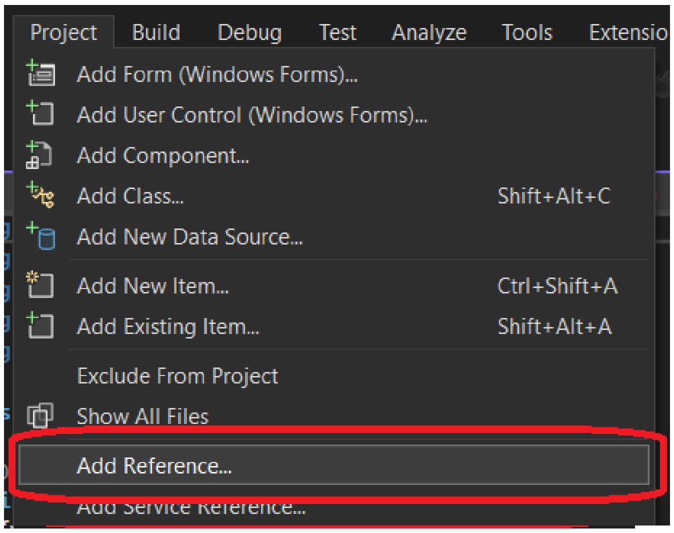
Project will be created in "C:\Users\jbrown\source\repos\ConsoleApp1\ConsoleApp1\"

Your starting view will be the source file named **Program.cs** which has some .NET starter code to help you frame up your own code additions. The code you will add for this tutorial will have scope within the Main function.



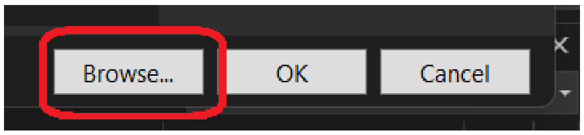
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ConsoleApp1
{
    0 references
    internal class Program
    {
        0 references
        static void Main(string[] args)
        {
        }
    }
}
```

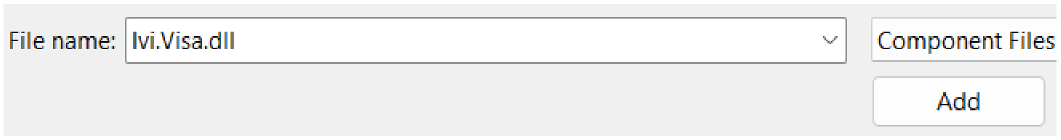
You will need to add references to the NI-VISA .NET library tools to gain access to the methods used to communicate with your test instrumentation. Use the menu bar to navigate to **Project->Add Reference**.



Click on the **Browse** button in the bottom right corner of the dialog then add the ***NationalInstruments.Visa.dll*** and ***Ivi.Visa.dll*** by navigating to the two following directories, selecting the DLL of interest, and clicking on the **Add** button.

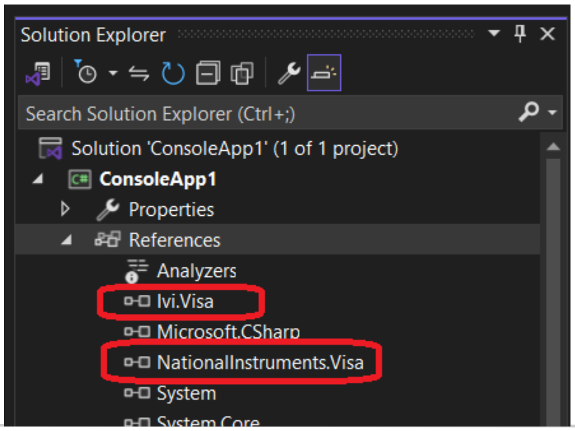
- *C:\Program Files\IVI Foundation\VISA\Microsoft.NET\Framework64\v4.0.30319\NI VISA.NET 23.8\NationalInstruments.Visa.dll*
- *C:\Program Files\IVI Foundation\VISA\Microsoft.NET\Framework64\v2.0.50727\VISA.NET Shared Components 7.2.0\Ivi.Visa.dll*



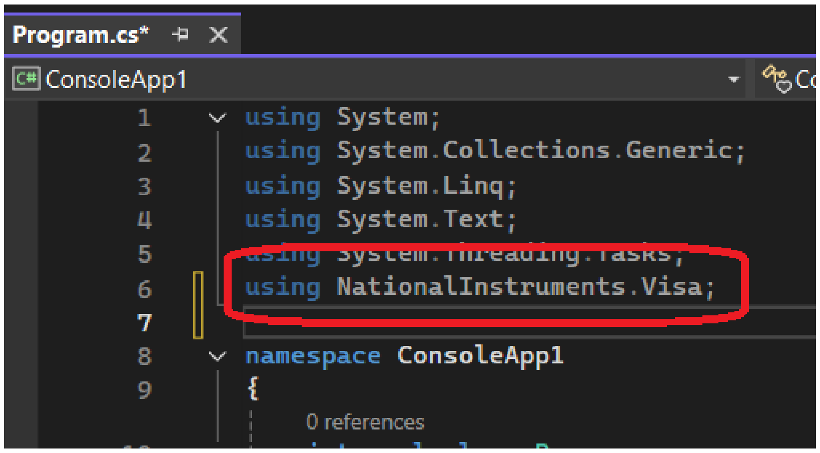

File name: Ivi.Visa.dll

NOTE: The above .Net framework DLL versions are current to the date of this document's creation and may vary for your system and installation options.

Once added, you can verify that the references via the Solution Explorer (*View->Solution Explorer*) and expanding the *ConsoleApp1->References* to show all that are included.



Finally, you will account for the reference in your Program.cs source code by adding "`using NationalInstruments.Visa;`" along with the other code references at the top of the source file.



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

namespace ConsoleApp1
{
```

Create Code with Open, Write, Read, and Close Instrument Operations

There are a number of ways that you can consider making your command transactions fast and efficient, but it is important that you start with the absolute basics first: open, close, write, and read.

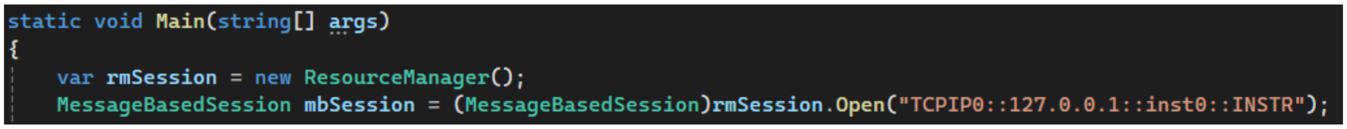
To get to these actions, you will start by declaring a VISA Resource Manager is a part of VISA that manages resources, including:

- Opening, closing, and finding resources
- Setting and retrieving resource attributes
- Generating events on resources
- Controlling access to devices

The Resource Manager (RM) can be used to scan a system for devices connected through different interface buses, and then returns its findings to the user.

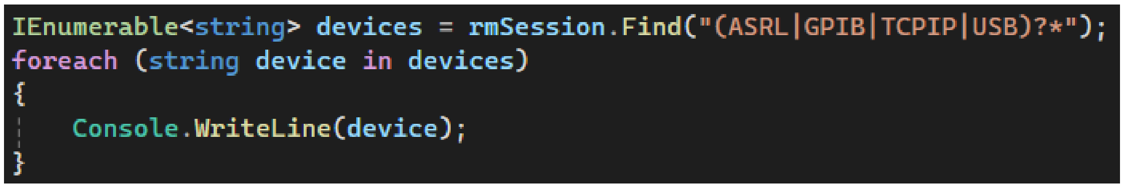
The RM is then used to open a message-based session which will act as a communication path to a VISA Resource – the physical device or instrument - that allows for the control and reception of data from instruments. Message-based communication is a standard protocol that can be used for a variety of hardware interfaces, including RS-232, GPIB, Ethernet, VXI, and USB.

In your code, start by adding a resource manager object just after the Main() declaration and use it to instantiate the message-based system object:

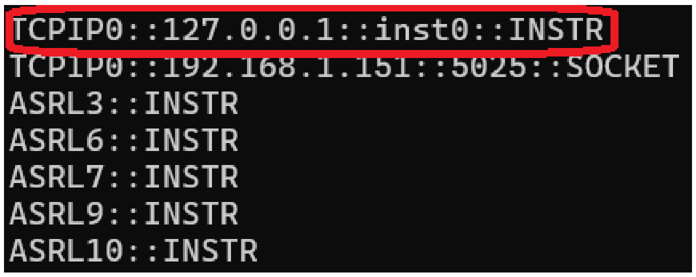


```
static void Main(string[] args)
{
    var rmSession = new ResourceManager();
    MessageBasedSession mbSession = (MessageBasedSession)rmSession.Open("TCPIP0::127.0.0.1::inst0::INSTR");
```

Note that the instantiation of the message-based session involves the call to the RM's Open() method which requires the instrument resource string to identify both your target device and its communications protocol. The string used in the above image is the default for the Bird Vector Network Analyzers. When you need some idea of what the connected instrument options are, you can use the RM's Find() method to generate a list which you can pick from:



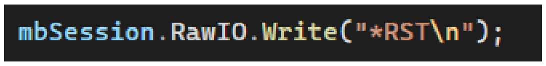
```
IEnumerable<string> devices = rmSession.Find("(ASRL|GPIB|TCPIP|USB)?*");
foreach (string device in devices)
{
    Console.WriteLine(device);
}
```



```
TCPIP0::127.0.0.1::inst0::INSTR
TCPIP0::192.168.1.151::5025::SOCKET
ASRL3::INSTR
ASRL6::INSTR
ASRL7::INSTR
ASRL9::INSTR
ASRL10::INSTR
```

For a more visual tool, consider using NI VISA Interactive Control. This will allow for instrument identification and expose you to the communications, but it will not create code for you.

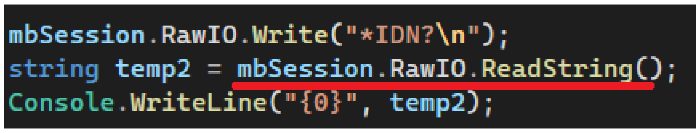
The two most common VISA commands you will work with are write and read, used for sending commands to your device and getting responses back from them, respectively. Typical instrument commands, SCPI in particular, will have a specific hierarchy and format to follow: any commands with a question mark at the end and sent to the device will yield a response, those without are simply used for settings and control. In either case, you will need to send the commands to the device which can be done using the *Write()* method. As an example, you can send the universal ***RST** command (which tells the device to return to a state where all default settings are applied) like the following:



```
mbSession.RawIO.Write("*RST\n");
```

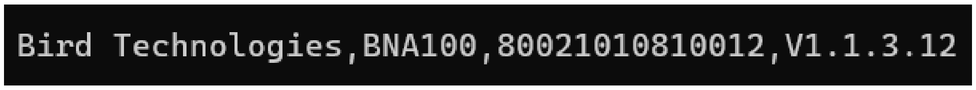
Note that the command string ends with the "\n" (line feed) escape sequence which VISA identifies as a termination character sequence. Other options are "\r" (carriage return) and variation of the two put together. While "\n" is adequate for most devices, consult your device's documentation to understand what works best for your instrument.

When a command ends with "?", it is used to tell your instrument that it expects a response, and to get the response you must follow your *Write()* with a *Read()* method. As an example, you can send the universal ***IDN?** command and get its reply – then printed to the console window – as follows:



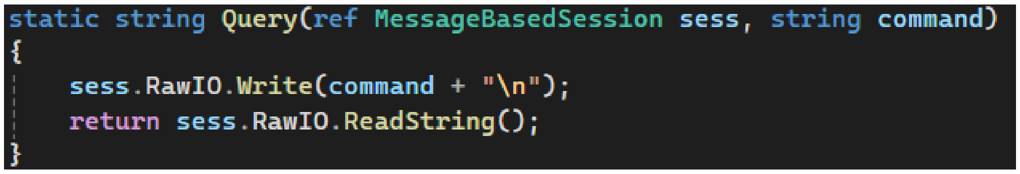
```
mbSession.RawIO.Write("*IDN?\n");
string temp2 = mbSession.RawIO.ReadString();
Console.WriteLine("{0}", temp2);
```

This yields the following comma-separated standard output showing the device manufacturer, model, serial number, and firmware version:



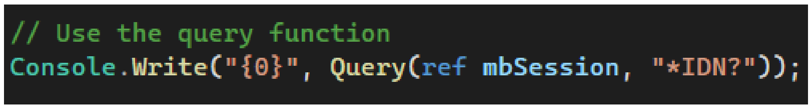
```
Bird Technologies,BNA100,80021010810012,V1.1.3.12
```

Commands ending with the “?” are often referred to as *queries*, and you may find some VISA implementations with a convenient pre-built query method. While this C# .NET VISA implementation does not, you can easily create a function of your own that wraps the write and read then provides a return value:



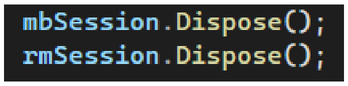
```
static string Query(ref MessageBasedSession sess, string command)
{
    sess.RawIO.Write(command + "\n");
    return sess.RawIO.ReadString();
}
```

This can then be called just like any other function and help reduce the verbosity of your code:



```
// Use the query function
Console.Write("{0}", Query(ref mbSession, "*IDN?"));
```

After you have strung your series of writes, reads, and queries together that now fully automates your otherwise manual test sequence, it is appropriate that your code concludes properly. The NI-VISA .NET library tools promote this with the *Dispose()* method, for all device instances as well as the resource manager, helping to release memory resources reserved for each.



```
mbSession.Dispose();
rmSession.Dispose();
```

A Word on Instrument Drivers

The code snippets provided above are used to give you an idea on how you might start to establish a base of code to reuse and share with your test development team. Bear in mind that many instrument manufacturers often take this type of development support as a necessity and build a library of methods and attributes specific to a particular model of instrument then make available for public use. This is what we refer to as an instrument “driver” and they are often made available in a variety of programming languages. A C# implementation is often part of an Interchangeable Virtual Instruments (IVI) driver made available by the device manufacturer. You can search the official IVI website to see if a driver is available.

While IVI drivers provide benefits such as state saving, simulation, and context-sensitive help, this often comes with an added degree of processing overhead. If efficiency is one of your primary test concerns, consider using the starter guidance provided within this document to help create your own streamlined driver code base.

Conclusion

Visual Studio is a popular and powerful programming language that enables engineers and programmers in creating control code and user interfaces. The software can be used for learning and sharing without fee, but also offers price plans for those who intend to use it to serve their own paying customers and those teaching at degree-granting institutions.

You now know where to get the VS 2022 Community software and how to install it. Moreover, you now have the knowledge on how to use basic VISA commands to connect with, write to, read from, and properly close out (dispose) of a connected instrument communication session. You should also be able to build your own custom, reusable code.

Instrument drivers provide a great starting point for instrument control, cutting down on the amount of code you might need to create yourself. When you cannot find a driver you are looking for – either through the IVI Foundation or directly from your instrument's manufacturer – you can always create your own.

For the full code example used to create this document, please visit the Bird GitHub repository. You can navigate the directories to find additional examples specifically designed for use with Bird's BNA100 and BNA1000 series of vector network analyzers. For additional NI-VISA .NET example code, you will find this available on your PC in the following directory after the installation steps above are completed: *<Public Documents>\National Instruments\NI-VISA\Examples\DotNET<x.x>*