THORLABS

Motion Control

Kinesis with C#

Quick Start Guide

Table of Contents

Chapter 1 Scope

The purpose of this guide is to provide users with the ability to write simple programs in the C# language to control Kinesis hardware within Microsoft Visual Studio on a Windows PC. This guide aims to help programmers fast-forward their understanding of the Kinesis .NET API as well as assist Foundation-level programmers get to grips with programming Kinesis-compatible Devices.

By the end of this guide, users should have gained the ability to connect and control Kinesis hardware by calling methods stored in the Kinesis DLL component files included with Kinesis. Users will be able to communicate with devices through coding a simple Console application and secondly be able to create a simple WPF (Windows Presentation Foundation) User Interface which will load the Kinesis Device Panel views.

NOTE: The examples contained in this guide focus around the use of KDC101 K-Cube DC Servo Controller & compatible stages/actuators. The concepts outlined in this guide can be applied to other Kinesis compatible hardware. Application code examples for other Kinesis compatible devices are contained in the .NET API help file.

Chapter 2 Getting Started

2.1 Downloading Kinesis

The first step you will need to take is to download and install Kinesis, the Thorlabs Automated Motion Control software package which can be used to control a wide range of Thorlabs hardware. Users of Thorlabs Motion Control hardware may also be aware of APT software, an older version of Thorlabs Motion Control software which Kinesis supersedes and shares similarities with.

Figure 1 – Kinesis software versions are available to download for free from www.thorlabs.com.

When downloading Kinesis, be sure to choose the version depending on the **System Type** of your PC. You can check the System Type by searching **System Information** in the **Windows Start menu**.

NOTE: The examples within this guide make use of **Kinesis 64-bit Software for 64-bit Windows** component assemblies (DLLs), which can be found through Windows Explorer in **C:\Program Files\Thorlabs\Kinesis** after installation.

2.2 Kinesis Software

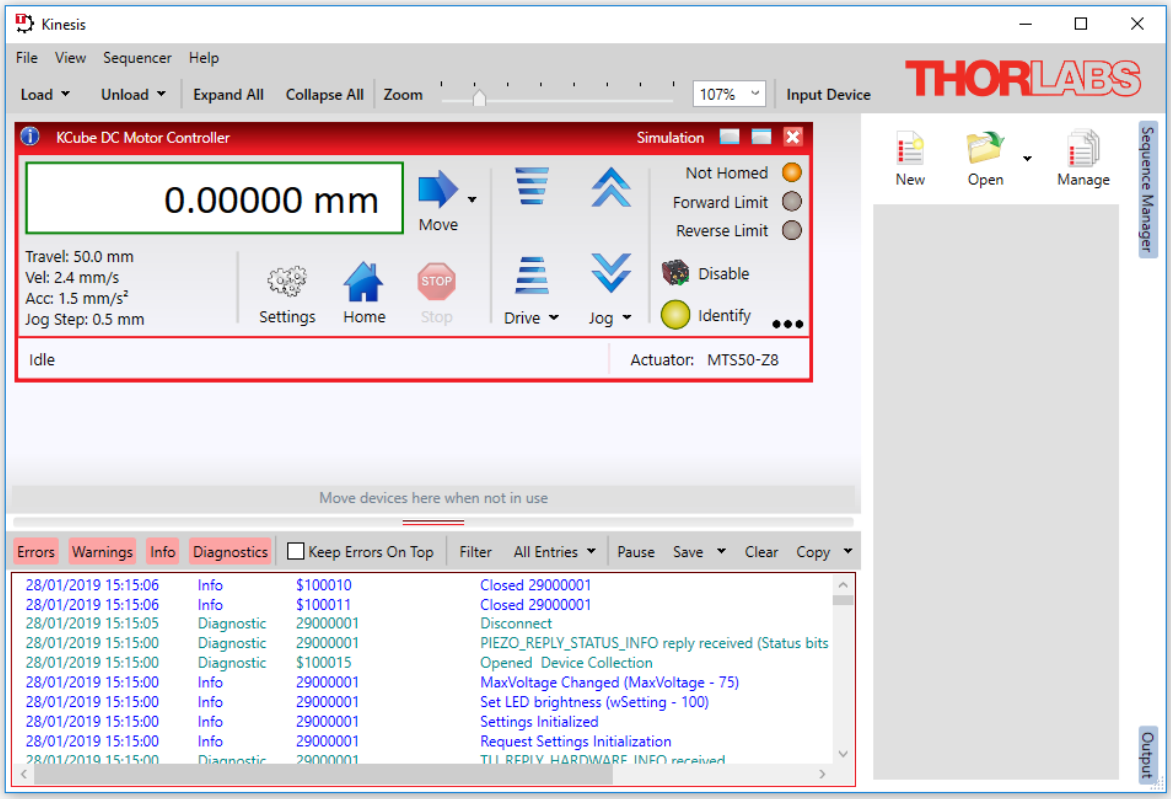


Figure 2 – The Kinesis.exe GUI.

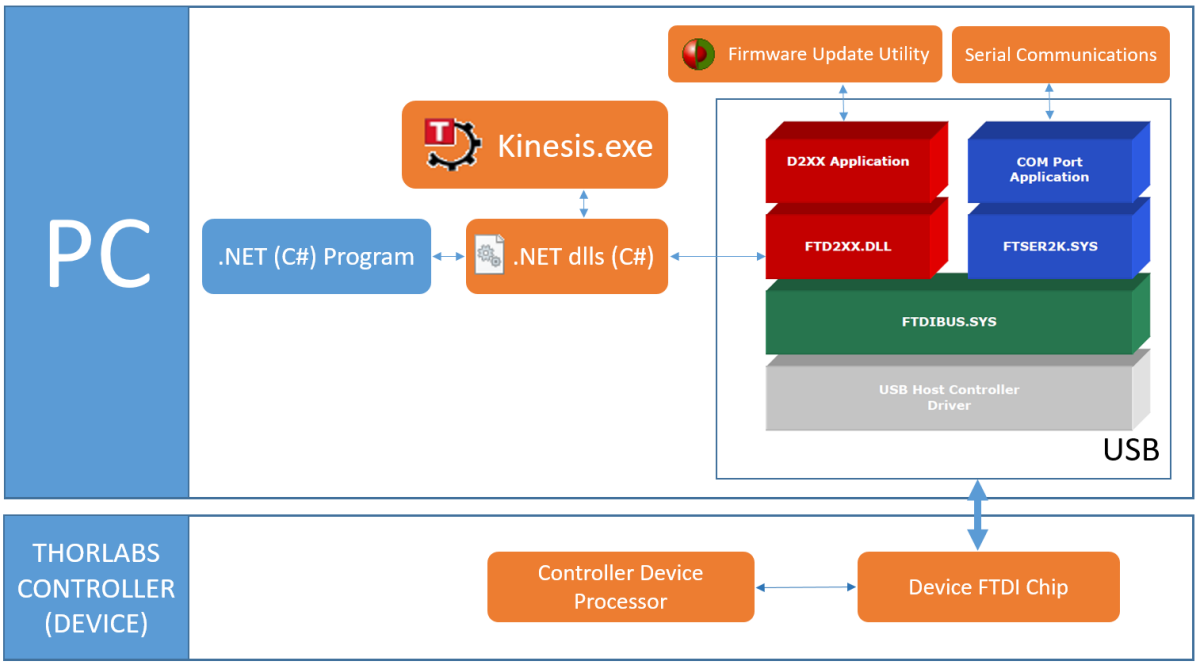


Figure 3 – Simplified Kinesis architecture.

After you've installed Kinesis, you will find within your C: drive a range of files worth getting to know. This includes:

- **Thorlabs.MotionControl.Kinesis.exe** – This is Kinesis. Simply click this executable to run the software.

  NOTE: Kinesis.exe contains an intuitive **Sequencer** - worth exploring for rapidly managing Kinesis move/command sequences without having to write any code.

- **Kinesis Simulator** – You can run this utility alongside Kinesis to simulate connected hardware.

- **Firmware Update utility** – Users can keep their device firmware up-to-date with the latest versions of Kinesis releases. The latest version of Kinesis can be downloaded from our website **here**.

- **.DLL (Assembly) files** - A large list of .NET "Dynamically Linked Library" (.dll) files. These are the building blocks of functionality used in the Kinesis Application and the important component used in this guide.

- **Help files** – Including the **.NET API Help file** which contains the full list of functions and properties, as well as code examples. An **API** is what's known as an Application Programming Interface, which in Lehman's terms is the code exposed to customers to write their own programs with Kinesis hardware, along with help and rules needed to write your program. **The .NET Framework** is a Microsoft framework which provides several development frameworks that you can use to build common application types.

2.3 DLL Files - "Which do I use and how do I make use of them?"

A **DLL** is a dynamically linked library, and a type of assembly. A 'DLL' (or .dll) is a file which contains chunks of code (functions and properties stored in classes) - the same code which has also been compiled to create Kinesis.exe. This functionality can be accessed in your own program by storing these assembly files within your Project folder. In Visual Studio you will also need to reference the .NET type assemblies in your program as described in the examples contained within this guide.

2.4 Device Specific DLLs

Not all assembly files will be needed for every Kinesis .NET program you write. Choosing which files to make use of depends on the 'Device' you're using as well as the type of 'Visual Studio Project' you wish to write.

A **Device** is the USB compatible Thorlabs Electronics Controller (e.g. KDC101 - K-Cube DC Servo Motor Controller), not the connected Stage/Actuator being driven.

After installing Kinesis, you'll have access to the .NET API Help document (file path C:\Program Files\Thorlabs\Kinesis\Thorlabs.MotionControl.DotNet_API). In this document, you will find a full list of devices supported by Kinesis along with the assemblies each device interacts with. For each device there are three device specific assemblies such as the following example list of .dlls for KDC101:

Name	Purpose
…KCubeDCServo.UI.dll	Provides access for users wanting to embed the Kinesis User Interface into their code.
…KCubeDCServo.CLI.dll	Contains the core functionality code of the device.
…KCubeDCServo.dll	Lower level code accessed by .CLI dll

Table 1 – KDC101 Device Specific .dll description

2.5 Native C DLLs and .NET Assemblies – what is the difference and should I care?

In the .NET API help guide you will see some assemblies are .NET dlls and some are Native C dlls which can be confusing. The important thing you will need to know when working through this guide is what to do with each type. This is outlined in the following table:

Name	DLL Type	Purpose	Things to do
…KCubeDCServo.UI.dll	.NET dll	Provides access for users wanting to embed the Kinesis User Interface into their code.	• Store in Project Folder • Add Reference
…KCubeDCServo.CLI.dll	.NET dll	Contains the core functionality code of the device.	• Store in Project Folder • Add Reference
…KCubeDCServo.dll	Native C dll	Lower level code accessed by the .CLI dll	Store in Project Folder only.

Table 2 - KDC101 Device Specific .dll types and instructions.

2.6 Generic DLLs

Generic assemblies follow an important pillar of object-oriented programming known as ‘Inheritance’, which means a device can inherit behaviours from code stored in generic classes. This means our Software Engineers can avoid duplicating large chunks of code where common behaviour exists between types of devices which would otherwise make software such as Kinesis.exe take up more memory on your PC.
Each specific .NET device assembly is built on and supported by a collection of generic assemblies – assemblies containing code used by different types of Kinesis compatible devices.

e.g. Thorlabs.DeviceManager,**DeviceManagerCLI** is a generic assembly containing Thorlabs.DeviceManager,DeviceManagerCLI.**DeviceManagerCLI** class which contains the **BuildDeviceList()** function which will interact with all Kinesis compatible device types. This is further described in figure 4 below.

The full collection of assemblies (DLLs) used by a device are also known as the device ‘Dependencies’.

NOTE: Assemblies are written in their ‘Namespace’ format. Namespaces are a hierarchical way of organising groups of classes and are useful for managing access to code based on the type of devices being used.

e.g. Thorlabs.MotionControl.DeviceManagerCLI.DeviceManagerCLI.BuildDeviceList()

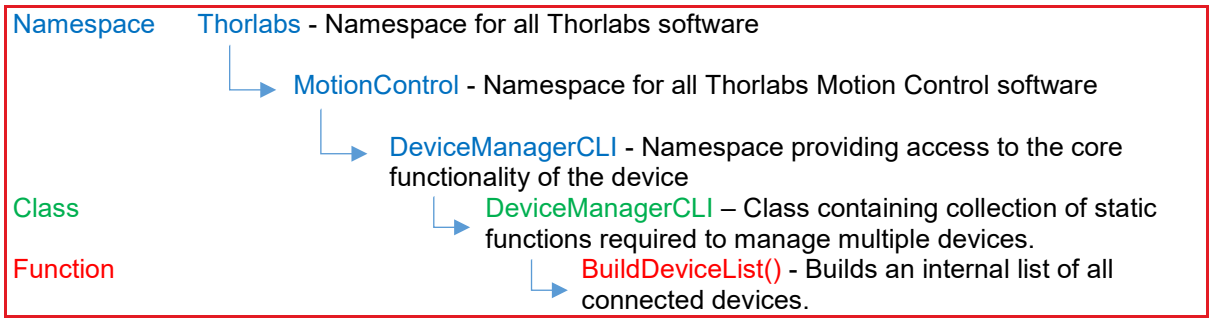


Figure 4 – Example Kinesis .DLL class hierarchy of BuildDeviceList().

In summary, you will generally do THREE things with the .dlls, each of which are covered in this guide.

1) Copy .dlls into your Project folder.
2) Reference the .NET type .dlls in your Visual Studio Project via Solution Explorer/
3) Allow calling of functions and assignment of properties by implementing 'Using' directives in your script.

2.7 Where can I find a list of functions and properties?

Functions and properties are defined and listed in the .NET API Help document ((C:\Program Files\Thorlabs\Kinesis\Thorlabs.MotionControl.DotNet_API). Within the .NET API document, you can click through the Namespaces to classes until you reach the list of functions and properties contained within the class.

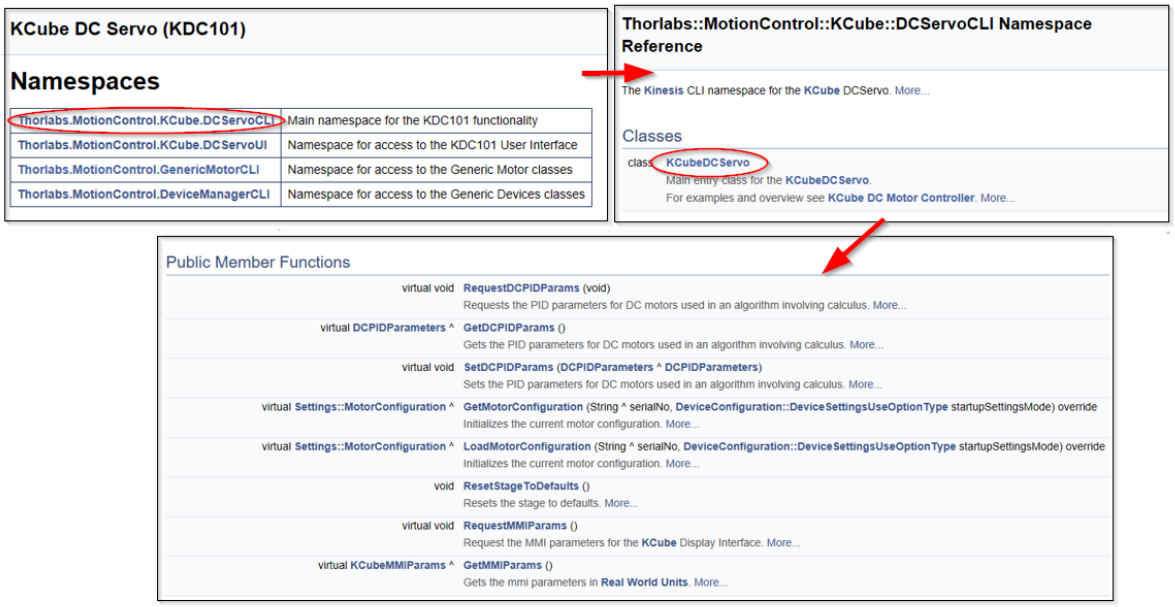


Figure 5- Access Public Types, Functions, Properties and Events from within the CLI .NET namespace and child classes.

Alternatively you can use 'Index' to search for specific methods and filter for the appropriate namespace location based on the device you're using.

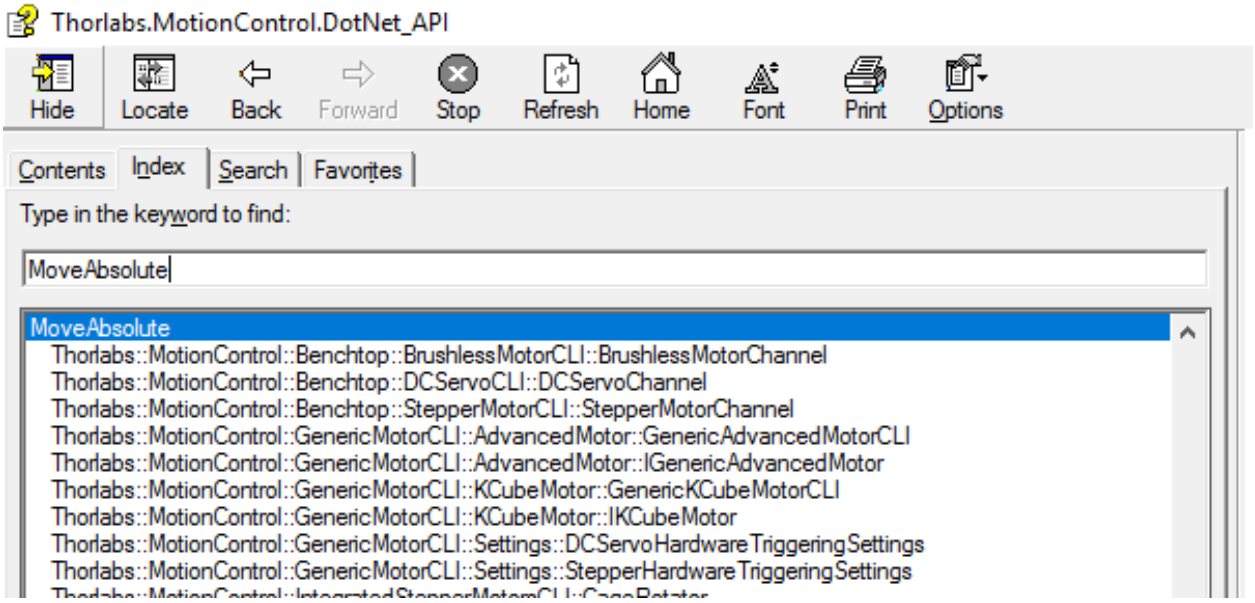


Figure 6 – The .NET API help guide 'Index' is useful for searching for filtering functions and properties.

Chapter 3 Visual Studio

Visual Studio is an integrated development environment from Microsoft. It exists in different package names which differ in functionality and cost. These are: Community, Professional, Enterprise and Code. Visual Studio Community is free to download from the Microsoft Website and is the IDE used within this guide.

NOTE: An IDE is an Integrated Development Environment. This is a software application used to create software applications. This will typically contain a source code editor, build automation tools and a debugger.

3.1 Key Features

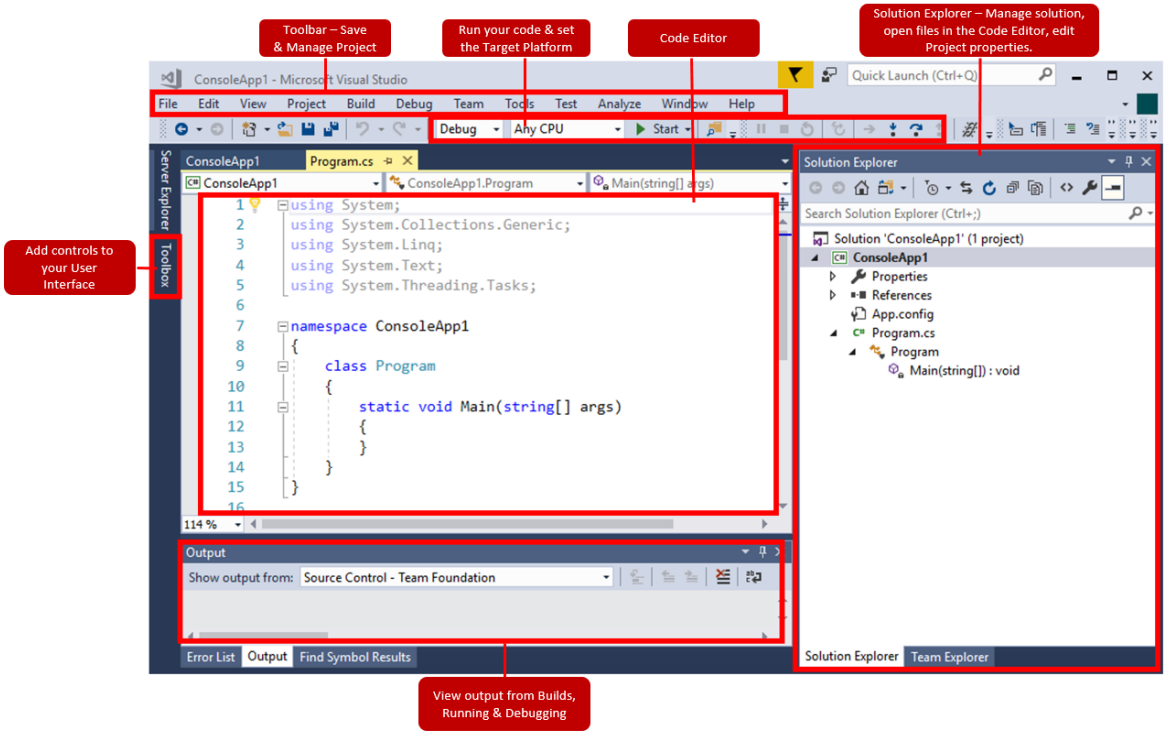


Figure 7 – Visual Studio Key Features.

3.2 Windows Workloads

Once you've installed VS Community, you can install a range of Windows Workloads. Visual Studio can take up a lot of memory so rather than giving you a huge single software package containing every feature which you may not need, workloads allow users to pick and choose the functionality they require. Workloads enable you to install packets of features required for various programming languages, frameworks or platforms you wish to work with. These can be installed during initial installation of Visual Studio. Alternatively from the Visual Studio Toolbar access **Tools** > **Get Tools and Features**.

To build the applications in this guide, you will need to install the **.NET desktop development** workload.

Access **Tools** > **Get Tools and Features**.in the Visual Studio toolbar

Tick the **.NET desktop development** workload.

Click **Modify** to install the workload.

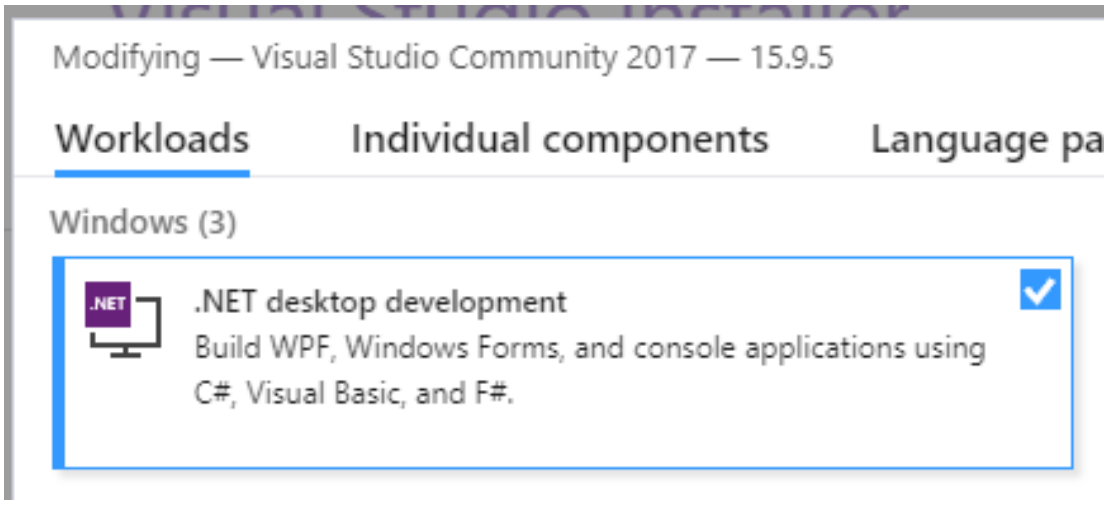


Figure 8 – The .NET desktop development Workload is required to run these examples.

3.3 Inserting Line Numbers

You can add 'Line numbers' to the Code Editor to keep track of Code Edits, and find sources of error when Debugging your program.

Select **Tools** > **Options**.

Under **Text Editor** > **All Languages** > **General** select and tick **Line Numbers**.

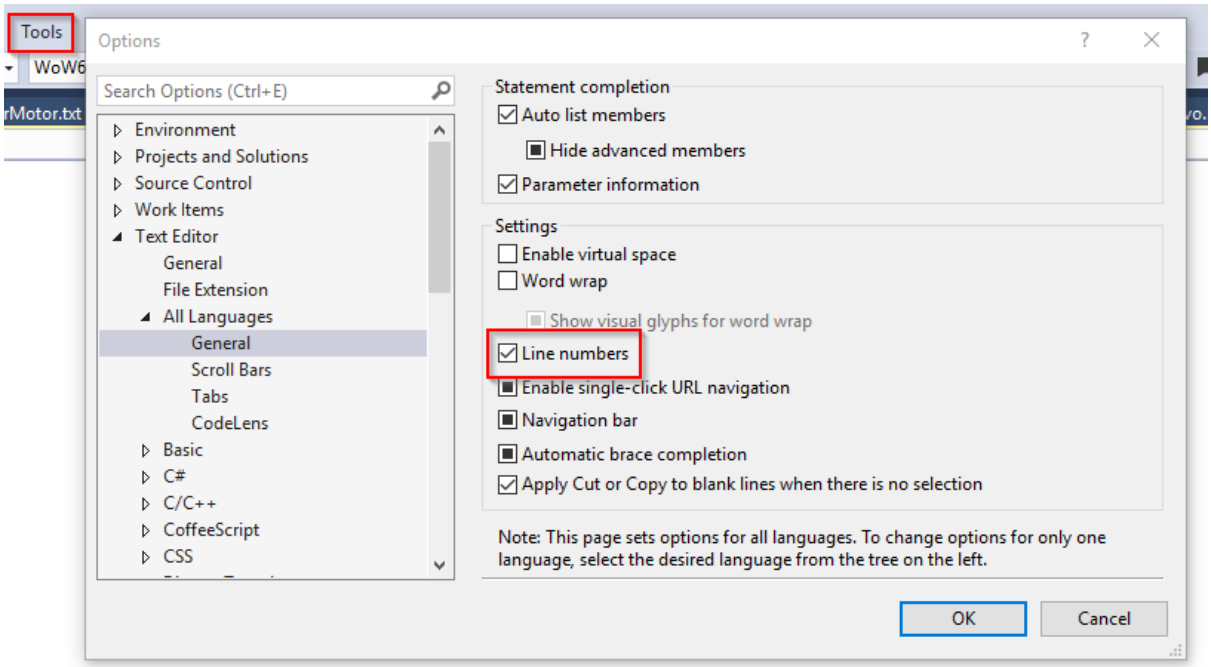


Figure 9 – Add Line Numbers to your Program.cs legend.

3.4 Changing your Project File Location

A Project is a folder which contains your program, along with the files needed to make your program work. When you later get to the stage of performing a 'Build' of your program, you are instructing the Visual Studio compiler to build an executable file, which contains code copied from your .cs file. This is where you will have typed your C# code, which has been translated into machine code – ready to be run. To do this, the Visual Studio compiler will sort through the Project folder and look for the files needed to create the executable. If you run your program and errors are produced, ensuring that the correct assembly files are located in your Project folder is the first thing to check.

The first thing you will do in Visual Studio is create a **Project**. This will produce a **Program.cs** file which will appear in Solution Explorer, which you can type code into via the Code Editor window in Visual Studio. By default you will find that your project is saved into a subset of folders which may be difficult to find at a later stage.

To change your Project location from the Visual Studio Toolbar:

Access **Tools** > **Options** > **Projects and Solutions** > **Locations** (or **General**).

Select the Projects location button (highlighted).

Press **Ctrl** + **N** to create a New Folder.

We would recommend calling this 'C#Programs' as below.

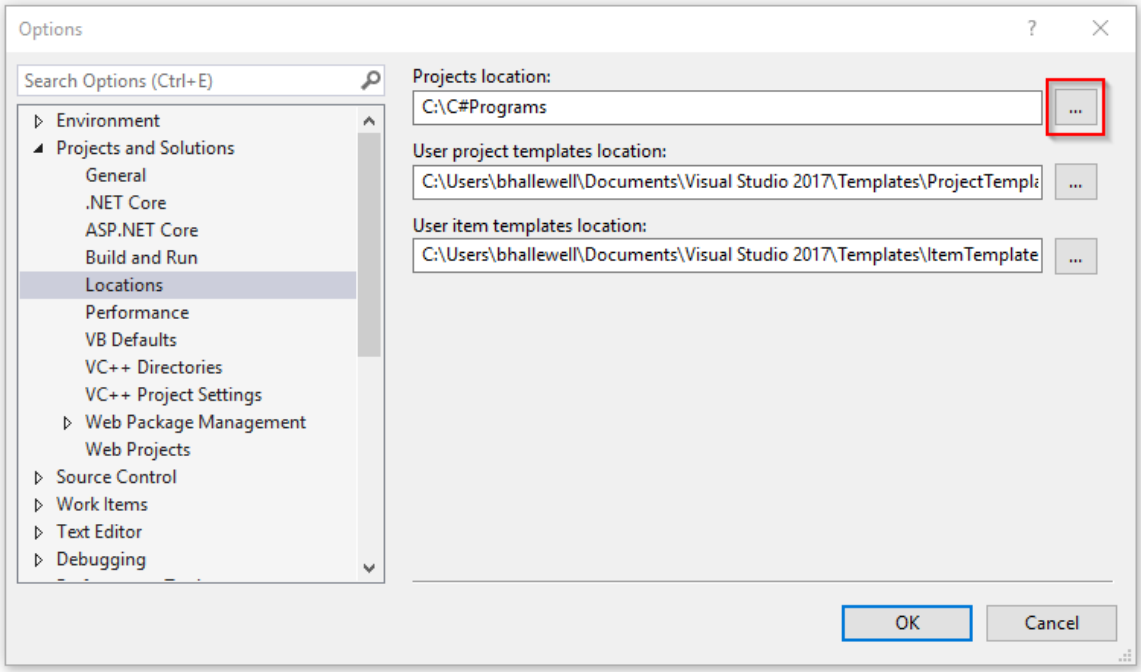


Figure 10 – Change the Projects Folder Location.

Chapter 4 Examples

4.1 Example 1 - Connect, Home, Move and Disconnect

The following example will show users the simplest way to connect to, home, move the device and safely disconnect communications with a KDC101 controller and connected stage/actuator. A KDC101 is a Thorlabs K-Cube DC Servo Controller which controls a range of DC servo motorised actuators and stages. The K-Cube is a range of controller types which have a compact footprint and can be controlled by the Kinesis software.

The following example is a Console Application, the simplest type of project in Visual Studio. Console Applications run in the Console in Windows known as the Command Window.

NOTE: This example omits try/catch statements which are used in exception error handling.

1) Create the Console Application

Within Visual Studio toolbar choose **File** > **New** > **Project**… > **Visual C#** > **Windows Desktop**.

Create a **Console App** and name this **KDC101Console**.

NOTE: It is worth making a note of the Project 'Location'. By default this file path is 'C:\Users\.....\source\repos\' which you may wish to change. See **'Changing the Project Folder Location'** in Section 3.4 for details.

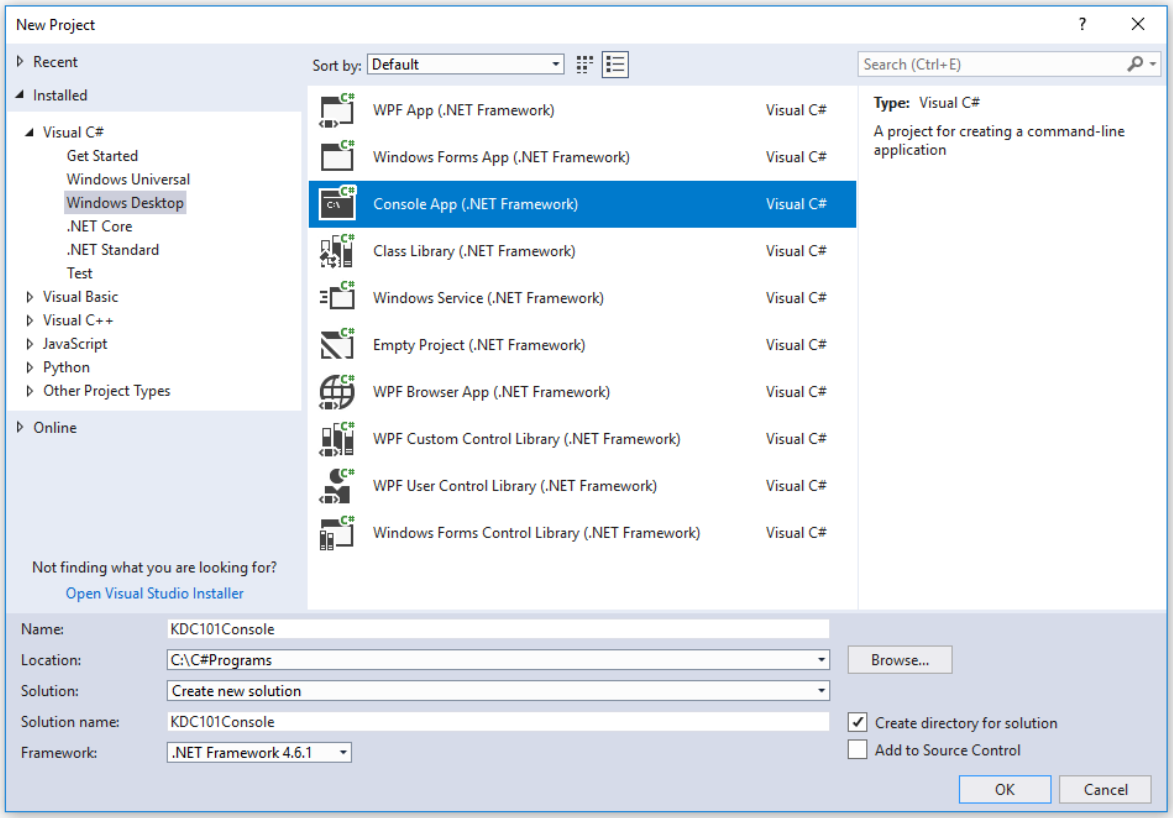


Figure 11 – Create a Console App from with the New Project Window.

2) Within Windows File Explorer, copy the Thorlabs .dll file you're your Project Folder (…KDC101Console\bin\Debug)

Highlight and press **Ctrl** + **C** to copy the .dll files you require (see DotNetAPI help file included within the Kinesis software folder) or if you're unsure, copy all Kinesis.dlls.

NOTE: The 64-bit .dlls are located in **C:\Program Files\Thorlabs\Kinesis** (32-bit .dlls are located in C:\Program Files (x86)\Thorlabs\Kinesis).

Within your Project Folder, press **Ctrl** + **P** to paste the dlls in **…bin\Debug**.

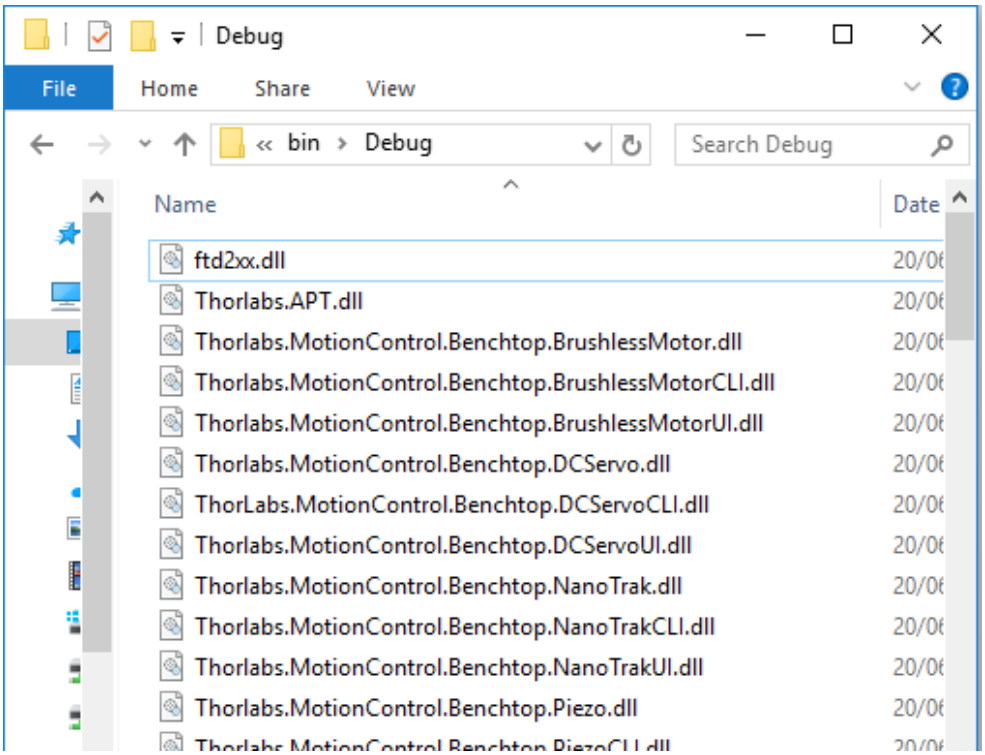


Figure 12 – The Kinesis dlls have been copied into the Debug folder within the Project Folder.

3) Set the Project Platform Target

In **Solution Explorer** right-click **KDC101Console** > **Properties**.

Under **Build** select Platform target to match the .dlls. In our case this is **x64**.

Click **File** > **Save All**.

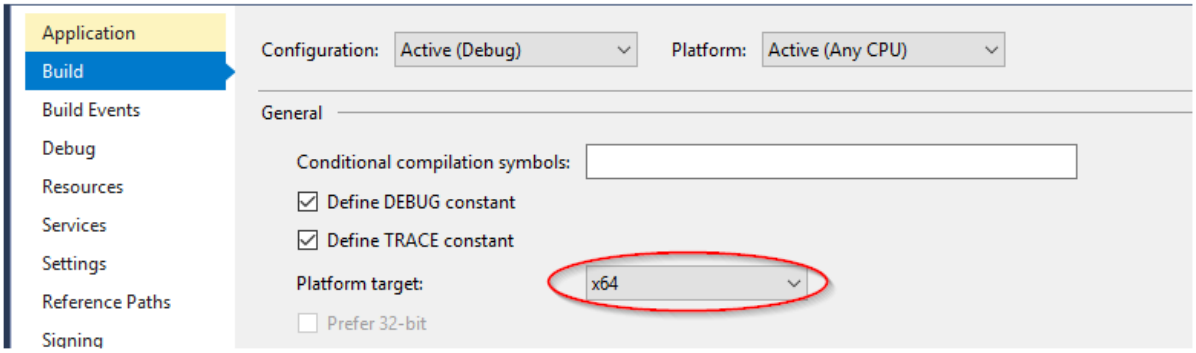


Figure 13- The Platform Target is set as x64, matching the type property of the C++ dlls in the Kinesis Software folder.

4) Add References to your Project.

The .NET API holds a list of Dependencies (or component programs) needed to access the functionality of a device.

In **Solution Explorer** right-click **KDC101Console** > **Add** > **Reference**

Click **Browse…** select the dlls below that were copied into your Project in Step 2).

- *Thorlabs.MotionControl.DeviceManagerCLI*
- *Thorlabs.MotionControl.GenericMotorCLI*
- *Thorlabs.MotionControl.Tools.Common*
- *Thorlabs.MotionControl.Tools.Logging*

Also add reference to the specific CLI assembly which applies to your device.

- *Thorlabs.MotionControl.KCube.DCServoCLI* (for KDC101)

5) Reference Namespaces in your MainWindow class.

Code **using** directives of the .dll namespaces you wish to access by inserting the following code in at line 6. in **Program.cs**.

```
using System.Threading;  //enables use of Thread.Sleep() "wait" method
using Thorlabs.MotionControl.DeviceManagerCLI;
using     Thorlabs.MotionControl.GenericMotorCLI.Settings;     //this     will
specifically target only the commands contained within the .Settings sub-class
library in *.GenericMotorCLI.dll.
using Thorlabs.MotionControl.KCube.DCServoCLI;
```

NOTE: 'Using' directives allow access to types from within a namespace within your program, such that you don't have to continually qualify the use of a type from within the namespace in the body of your program.

6) Specify the device serial number

Copy the following code after "static void Main(string[] args) {".

```
// We create the serial number string of your connected controller. This will
// be used as an argument for LoadMotorConfiguration(). You can replace this
// serial number with the number printed on your device.
string serialNo = "27000423";
```

NOTE: Assign the device serial number string matching your device e.g. *27000423*.
The serial number is a unique number to identify every controller unit made and is typically printed on your device hardware.

7) Enter the rest of the annotated code

Enter the following code below "string serialNo = "********";".

```
// This instructs the DeviceManager to build and maintain the list of
// devices connected.
DeviceManagerCLI.BuildDeviceList();

// This creates an instance of KCubeDCServo class, passing in the Serial
Number parameter.
KCubeDCServo device = KCubeDCServo.CreateKCubeDCServo(serialNo);

// We tell the user that we are opening connection to the device.
Console.WriteLine("Opening device {0}", serialNo);

// This connects to the device.
device.Connect(serialNo);

// Wait for the device settings to initialize. We ask the device to
// throw an exception if this takes more than 5000ms (5s) to complete.
device.WaitForSettingsInitialized(5000);

// This calls LoadMotorConfiguration on the device to initialize the
DeviceUnitConverter object required for real world unit parameters.
MotorConfiguration motorSettings = device.LoadMotorConfiguration(serialNo,
DeviceConfiguration.DeviceSettingsUseOptionType.UseFileSettings);

// This starts polling the device at intervals of 250ms (0.25s).
device.StartPolling(250);

// We are now able to Enable the device otherwise any move is ignored. You
should see a physical response from your controller.
device.EnableDevice();
Console.WriteLine("Device Enabled");

// Needs a delay to give time for the device to be enabled.
Thread.Sleep(500);

// Home the stage/actuator.
Console.WriteLine("Actuator is Homing");
device.Home(60000);
```

```
// Move the stage/actuator to 5mm (or degrees depending on the device
connected).
Console.WriteLine("Actuator is Moving");
device.MoveTo(5, 60000);

// Stop polling the device.
device.StopPolling();

// This shuts down the controller. This will use the Disconnect() function to
close communications & will then close the used library.
device.ShutDown();

// Click any key at the end of the program to exit.
Console.WriteLine("Process complete. Press any key to exit");
Console.ReadKey();
```

8) Save the Solution.

Click **File** > **Save All**

9) Build the Solution

Right-click your Project in **Solution Explorer** > click **Build**.

This will create **KCubeDCServo.exe** within your Project Folder.

10) Run the Program

Right-click your Project in **Solution Explorer** > **Debug** > click **Start new instance**.

Command Prompt

You can also run your console application from the command prompt.

1. Click the Windows Start Menu > type **cmd** to open **Command Prompt**.
2. In Command Prompt type
   cd C:\C#Programs\KCubeDCServo\KCubeDCServo\bin\Debug\KCubeDCServo
3. Press Enter.
4. In Command Prompt type **KCubeDCServo.exe** > press enter to run the program.

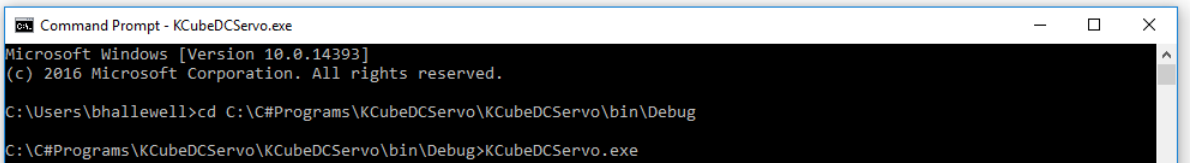


Figure 14 – How to run your program from within the Command Prompt.

4.2 Example 2 – Running a Kinesis Instrument Panel in Visual Studio

This simple example will enable you to load an instance of a Kinesis device UI panel in a WPF application within Visual Studio.

The .NET Framework provides several development frameworks that you can use to build common application types. Windows Presentation Foundation (WPF) is a framework for building Desktop client applications, which would have a Graphical User Interface (GUI). You will need to code the back-end of this application in C# and the user interface in .xaml – a mark-up language which simplifies creating a UI for a .NET Framework application.

The example below is based on loading a KDC101 DC Servo Controller connected to a MTS50-Z8 50mm linear stage.

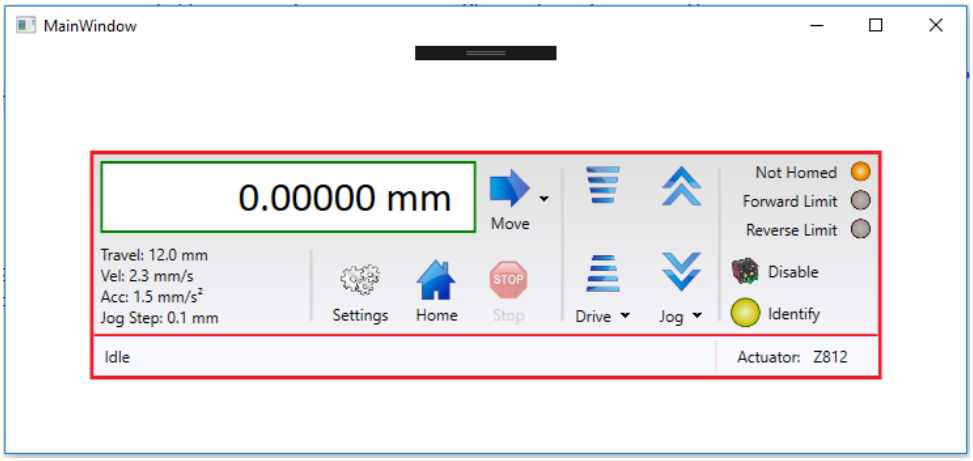


Figure 15 – The KDC101 Kinesis GUI hosted in a WPF application.

1) Create the WPF Application

Within Visual Studio toolbar choose **File** > **New** > **Project**… > **Visual C#** > **Windows Desktop**

Create New WPF Application.

Create a **WPF App** and name this **WPFKcubeUI**.

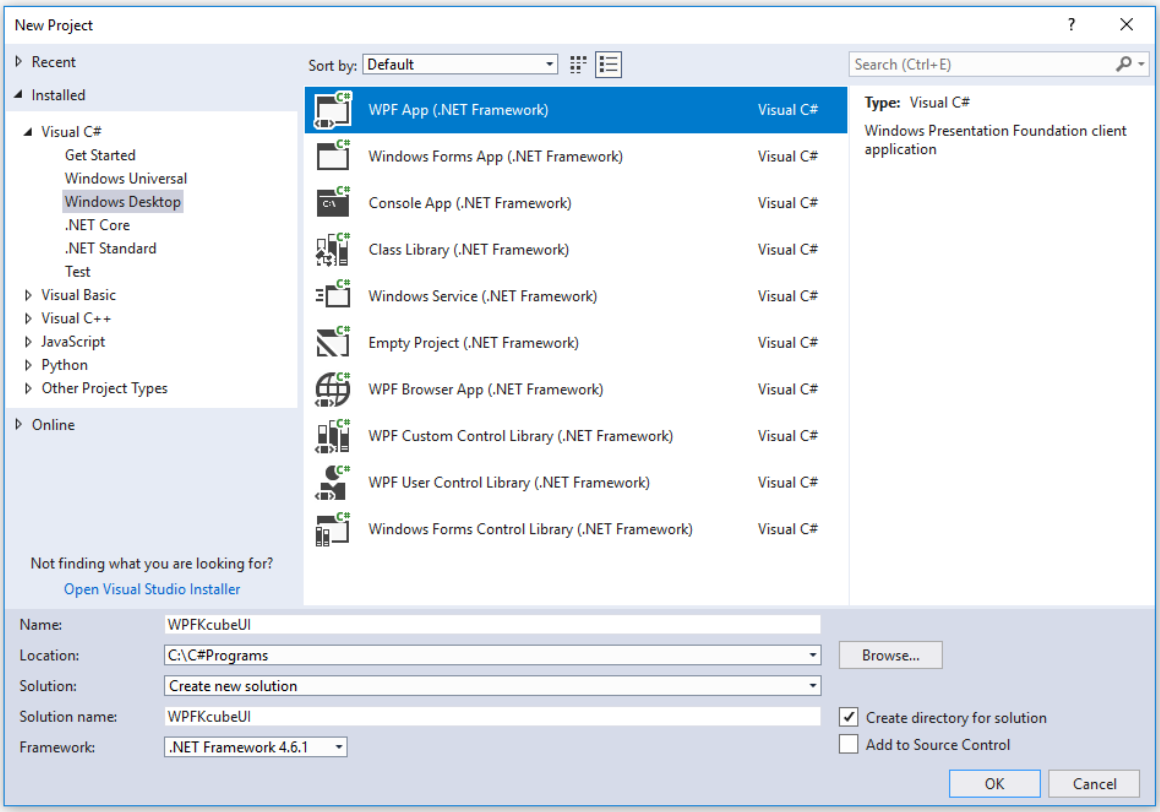


Figure 16 - Create a WPF App from with the New Project Window.

2) Within Windows File Explorer, copy the Thorlabs .dll file into WPFKCubeUI\bin\Debug.

Highlight and press **Ctrl** + **C** to copy the .dll files you require (see DotNetAPI help file included within the Kinesis software folder) or if you're unsure, copy all Kinesis.dlls.

NOTE: The 64-bit .dlls are located in **C:\Program Files\Thorlabs\Kinesis** (32-bit .dlls are located in C:\Program Files (x86)\Thorlabs\Kinesis).

Within your Project Folder, press **Ctrl** + **P** to paste the dlls in **…bin\Debug**.

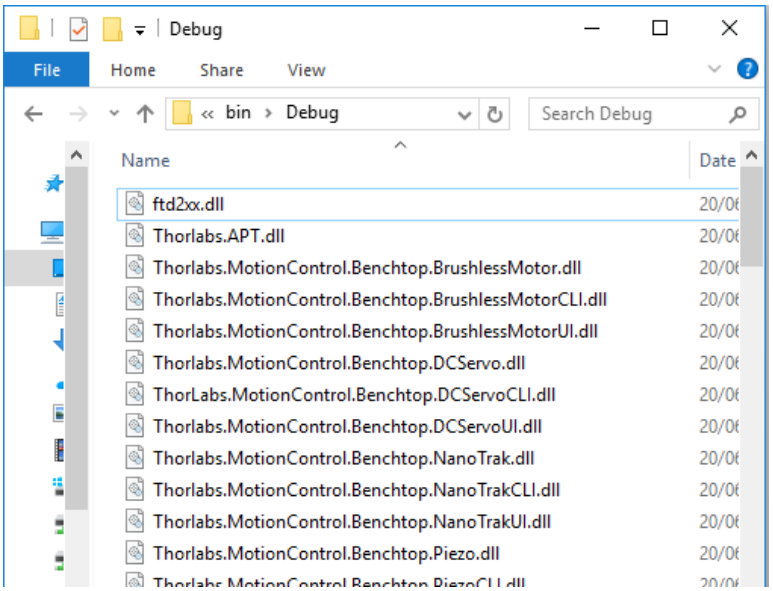


Figure 16 - The Kinesis dlls have been copied into the Debug folder within the Project Folder.

3) Set the Project Platform Target

In **Solution Explorer** right-click on **WpfApp1** > **Properties**.

Under **Build** select Platform target to match the .dlls. In our case this is **x64**.

Click **File** > **Save All**.

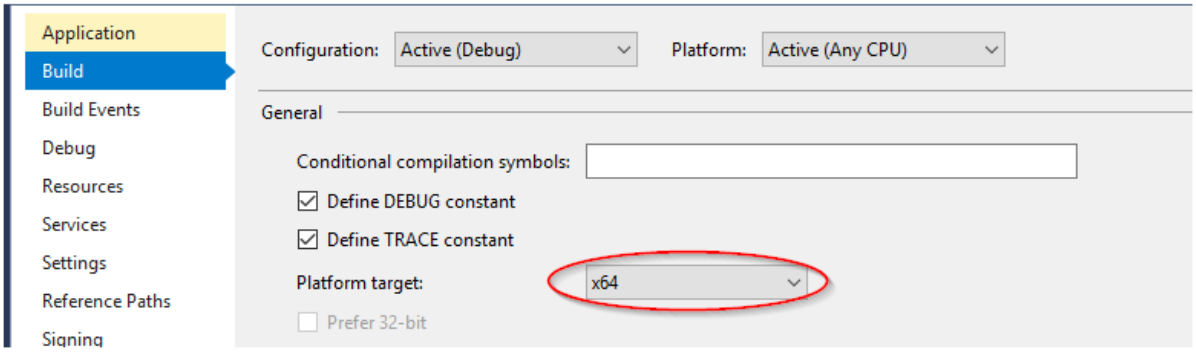


Figure 17 - The Platform Target is set as x64, matching the type property of the C++ dlls in the Kinesis Software folder.

4) Add References to your Project.

The .NET API holds a list of Dependencies (or component programs) needed to access the functionality of a device.

Right-click **WpfApp1** within **Solution Explorer** > **Add** > **Reference**

Click **Browse…** select the dlls below that were copied into your Project in Step 2).

- *Thorlabs.MotionControl.DeviceManagerCLI*
- *Thorlabs.MotionControl.DeviceManagerUI*
- *Thorlabs.MotionControl.GenericMotorCLI*
- *Thorlabs.MotionControl.GenericMotorUI*
- *Thorlabs.MotionControl.Tools.Common*
- *Thorlabs.MotionControl.Tools.Logging*
- *Thorlabs.MotionControl.Tools.WPF*

Also add reference to the specific CLI and UI assemblies which apply to your device.

- *Thorlabs.MotionControl.KCube.DCServoCLI**
- *Thorlabs.MotionControl.KCube.DCServoUI**

*Applies to use of KDC101

5) Reference Namespaces in your MainWindow class.

MainWindow inherits functionality from System.Windows.Window, a class which provides the ability to configure and control dialogue boxes and user interface windows. This class contains the code which will interact with the Kinesis Instrument Panel which is designed in MainWindow.xaml.

In **Solution Explorer**, under MainWindow.xaml open **MainWindow.xaml.cs** class.

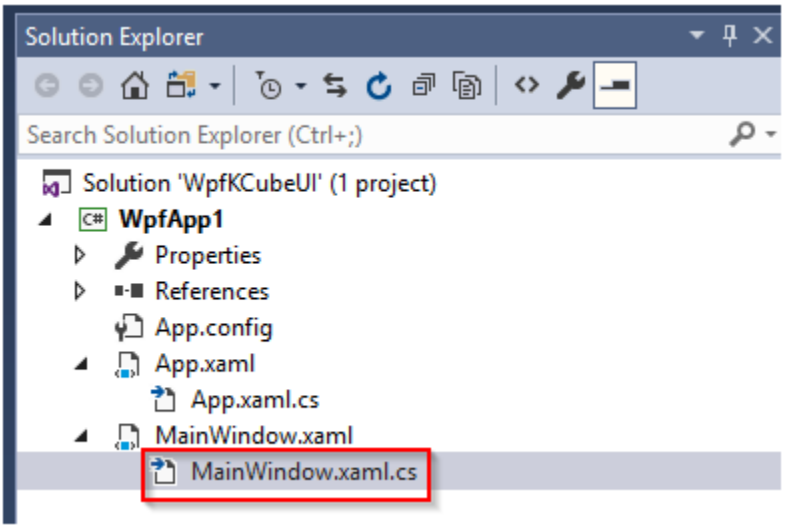


Figure 18 – Solution Explorer showing the location of MainWindow.xaml.cs within the WPF Project.

Code **using** directives of the .dll namespaces you wish to access by inserting the following code in at line 16 in **MainWindow.xaml.cs**

```
using Thorlabs.MotionControl.DeviceManagerCLI;
using Thorlabs.MotionControl.KCube.DCServoCLI;
using Thorlabs.MotionControl.KCube.DCServoUI;
```

NOTE: Depending on the device you're connecting, will affect the using statements you've written. These are a shortcut for accessing Classes, Functions and Properties within the Kinesis class libraries as outlined within the Kinesis .NET API document.

6) Code in .XAML to design the appearance and functionality of the interface.

Open **MainWindow.xaml** > edit and type in the following code.

```
Title="MainWindow" Height="350" Width="750" Loaded="MainWindow_OnLoaded"
Closed="MainWindow_OnClosed">

 <Grid Margin="50,75,50,50">
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto" />
            <RowDefinition Height="Auto" />
        </Grid.RowDefinitions>
    <ContentControl Name="_contentControl" Margin="5" Grid.Row="1"/>
 </Grid>
</Window>
```

7) Create the MainWindow_OnLoaded event

Right-click "MainWindow_OnLoaded" on Line 8 in **MainWindow.xaml.**

Select **Go To Definition**.

This will create and register a **MainWindow_OnLoaded** event within **MainWindow.xaml.cs**.

NOTE: The code within this event will run when the UI window is generated.

8) Create the MainWindow_OnLoaded event

Repeat Step 7) for "MainWindow_OnClosed".

NOTE: The code within this event will run when the UI window is closed.

You are now ready to code the back end functionality of the device which is coded within the **MainWindow.xaml.cs** class.

9) Create Thorlabs.MotionControl.KCube.DCServoCLI.KCubeDCServo class reference

In **MainWindow.xaml.cs** type the following code beneath `public MainWindow() { InitializeComponent(); }`.

```
        //The field _kCubeDCServo is an instance field of KCubeDCServo. This
builds an empty reference _kCubeDCServo.
        //This reference will later be assigned to create an instance of the
KCubeDCServo class object variable which we will interact
//within MainWindow_OnLoaded() & MainWindow_OnClosed()
        KCubeDCServo _kCubeDCServo = null;
```

_kCubeDCServo is a reference variable of the *Thorlabs.MotionControl.KCube.DCServoCLI.**KCubeDCServo*** class.

In C# you can assign null to any reference variable such that it doesn't yet refer to an object in memory. An object, being the instance of this **KCubeDCServo** class, is later created and assigned to this reference when the **CreateKCubeDCServo()** function is called.

The **Private** Access Modifier makes **KCubeDCServo** class functions and properties available to the **MainWindow.xaml.cs** class and component events as shown below.

10) Code the Device Connection

Copy the following code into your MainWindow_OnLoaded() event 'private void MainWindow_OnLoaded(object sender, RoutedEventArgs e)'

```
{
// This instructs the DeviceManager to build and maintain the list of
// devices connected. We then print a list of device name strings called
// "devices" which contain the prefix "27"
DeviceManagerCLI.BuildDeviceList();
List<string> devices = DeviceManagerCLI.GetDeviceList(27);

// IF statement - if the number of devices connected is zero, the Window
// will display "No Devices".
if (devices.Count == 0)
{
         MessageBox.Show("No Devices");
         return;
}
// Selects the first device serial number from "devices" list.
string serialNo = devices[0];

// Creates the device. We assign an instance of the device to _kCubeDCServo
KCubeDCServo _kCubeDCServo = KCubeDCServo.CreateKCubeDCServo(serialNo);


// Connect to the device & wait for initialisation. This is contained in a
// Try/Catch Error Handling Statement.
try
{
         _kCubeDCServo.Connect(serialNo);
```

```
// wait for settings to be initialized
            _kCubeDCServo.WaitForSettingsInitialized(5000);
}
catch (DeviceException ex)
{
            MessageBox.Show(ex.Message);
            return;
}
// Create the Kinesis Panel View for KDC101
_contentControl.Content = KCubeDCServoUI.CreateLargeView(_kCubeDCServo);
}
```

11) Code Device Disconnection

Copy the following code into your MainWindow_Closed() method private void MainWindow_Closed(object sender, RoutedEventArgs e)

```
// Disconnect device after closing the Window.
if ((_kCubeDCServo != null) && _kCubeDCServo.IsConnected)
{
      _kCubeDCServo.Disconnect(true);
}
```

12) Save the Solution.

Click **File** > **Save All**

13) Build the Solution

Right-click **WpfApp1** in **Solution Explorer** > click **Build**.

14) Run the Program

Right-click **WpfApp1** in **Solution Explorer** > click **Debug** > **Start new instance**.

OR

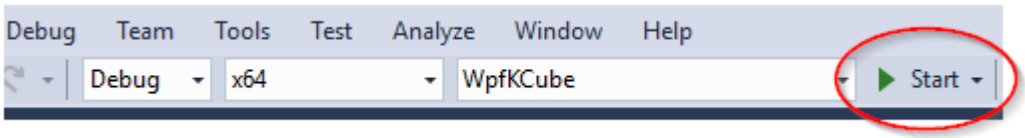


Figure 19 – Run your program from the Visual Studio toolbar.

NOTE: There are a range of Console Application and WPF Application code examples for each type of Kinesis compatible device in the .NET API help file.

Glossary

DLL	Dynamically linked library, It is a type of assembly which contains classes and functionality which can be invoked in your program. These are stored in your Project
Build	The primary function of Build is to compile your program into Machine Code which can be ran by a computer. This will also organise files such as DLLs in your project folder.
Framework	A layered software structure containing components such as programs, rules & support that can be built on by Software Developers. The .NET Framework is a Windows framework developed by Microsoft which includes a large class library which can be accessed by code in different languages.
IDE	Integrated Development Environment. A software application used to write software. This will typically contain a source code editor, build automation tools and a debugger.
Console Application	A lightweight application which can be output through a command-line interface such as Command Prompt.
WPF	Windows Presentation Foundation. A .NET development framework (template) for building Desktop client applications. WPF applications are versatile frameworks for building interfaces and are typically programmed in C# and .xaml.
Solution	A structure for organising projects in Visual Studio. The solution maintains the state information for projects in .sln files.
Project	The group of files which are needed to build & run a program or collection of programs. These files are typically contained in a single folder.
Namespace	A way to organise classes hierarchically. These are typically .NET Framework namespaces, such as the System.Windows namespace where Windows is a class within the System Namespace. These can also be user-defined namespaces such as Thorlabs.MotionControl.DeviceManager from Thorlabs. By adding reference to an assembly, you can list the namespace you wish to use in a using statement to qualify types being used in your program.
Class	A template for creating or instantiating objects. This also contains functions and properties which can changed or run by an object.
Device	The Kinesis software term for the Kinesis compatible electronics controller.
K-Cube	The product line of compact Thorlabs electronic controllers compatible with Kinesis software e.g. KDC101 - K-Cube Brushed DC Servo Motor Controller
Stage/Actuator	Motorised translation device driven by a Thorlabs controller.

Thorlabs Worldwide Contacts

USA, Canada, and South America
Thorlabs, Inc.
56 Sparta Avenue
Newton, NJ 07860
USA
Tel: 973-300-3000
Fax: 973-300-3600
www.thorlabs.com
www.thorlabs.us (West Coast)
Email: sales@thorlabs.com
Support: techsupport@thorlabs.com

UK and Ireland
Thorlabs Ltd.
1 Saint Thomas Place
Ely CB7 4EX
Great Britain
Tel: +44 (0) 1353-654440
Fax: +44 (0) 1353-654444
www.thorlabs.com
Email: sales.uk@thorlabs.com
Support: techsupport.uk@thorlabs.com

Europe
Thorlabs GmbH
Hans-Böckler-Str. 6
85221 Dachau / Munich
Germany
Tel: +49-(0) 8131-5956-0
Fax: +49-(0) 8131-5956-99
www.thorlabs.de
Email: europe@thorlabs.com

Scandinavia
Thorlabs Sweden AB
Bergfotsgatan 7
431 35 Mölndal
Sweden
Tel: +46-31-733-30-00
Fax: +46-31-703-40-45
www.thorlabs.com
Email: scandinavia@thorlabs.com

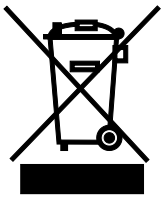
France
Thorlabs SAS
109, rue des Côtes
78600 Maisons-Laffitte
France
Tel: +33 (0) 970 444 844
Fax: +33 (0) 825 744 800
www.thorlabs.com
Email: sales.fr@thorlabs.com

Brazil
Thorlabs Vendas de Fotônicos Ltda.
Rua Riachuelo, 171
São Carlos, SP 13560-110
Brazil
Tel: +55-16-3413 7062
Fax: +55-16-3413 7064
www.thorlabs.com
Email: brasil@thorlabs.com

Japan
Thorlabs Japan, Inc.
3-6-3 Kitamachi,
Nerima-ku, Tokyo 179-0081
Japan
Tel: +81-3-6915-7701
Fax: +81-3-6915-7716
www.thorlabs.co.jp
Email: sales@thorlabs.jp

China
Thorlabs China
Room A101, No. 100, Lane 2891,
South Qilianshan Road
Putuo District
Shanghai 200331
China
Tel: +86 (0) 21-60561122
Fax: +86 (0) 21-32513480
www.thorlabschina.cn
Email: chinasales@thorlabs.com

Thorlabs verifies our compliance with the WEEE (Waste Electrical and Electronic Equipment) directive of the European Community and the corresponding national laws. Accordingly, all end users in the EC may return “end of life” Annex I category electrical and electronic equipment sold after August 13, 2005 to Thorlabs, without incurring disposal charges. Eligible units are marked with the crossed out “wheelie bin” logo (see right), were sold to and are currently owned by a company or institute within the EC, and are not dissembled or contaminated. Contact Thorlabs for more information. Waste treatment is your own responsibility. “End of life” units must be returned to Thorlabs or handed to a company specializing in waste recovery. Do not dispose of the unit in a litter bin or at a public waste disposal site.



Annex I

THORLABS
www.thorlabs.com