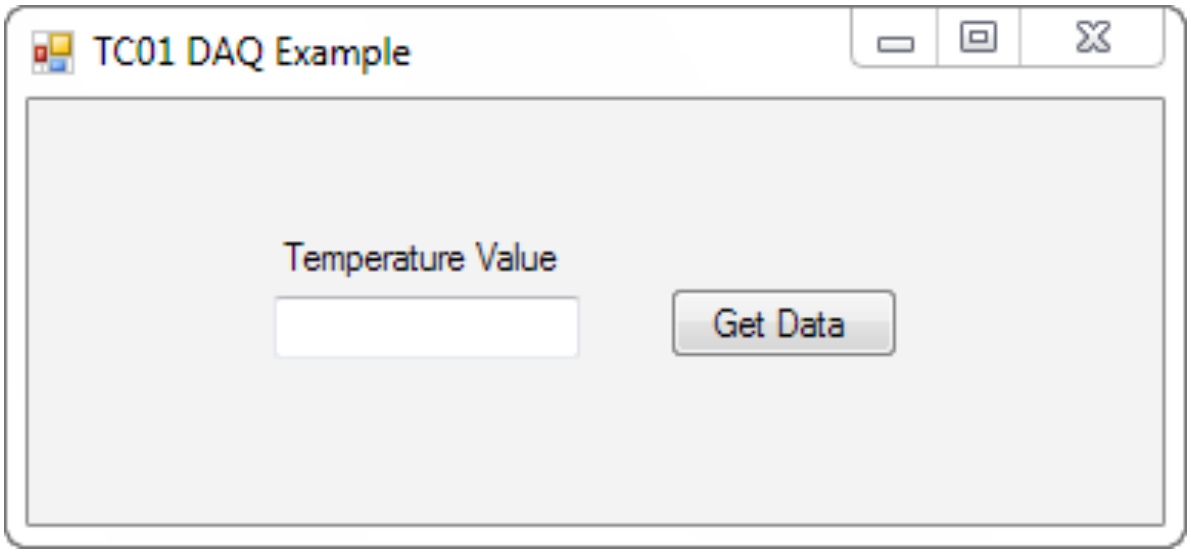
https://www.halvorsen.blog

Data Acquisition in C#

Hans-Petter Halvorsen



Data Acquisition in C#

Hans-Petter Halvorsen

E-Mail: hans.p.halvorsen@usn.no

Web: https://www.halvorsen.blog

https://www.halvorsen.blog

Table of Contents

1 Introduction

In this Tutorial we will learn how to create DAQ (Data Acquisition) applications in Visual Studio and C#. We will use a USB-6008 DAQ device from National Instruments as an example. In order to use DAQ devices from National Instruments in C# and Visual Studio we need to NI-DAQmx driver provides by National Instruments. As part of this installation you can install a .NET API. We will use this API to create a simple DAQ application. In addition, we will use Measurement Studio which is an add-on to Visual Studio which makes it easier to create more advanced DAQ applications.

In this Tutorial we end up with a control application. We will send and read data to a DAQ device, and we will create our own discrete PID controller, low-pass filter and a discrete model of our system. We will also read and write data to an OPC server.

You will find this document and lots of other information in the following web site:

https://www.halvorsen.blog/documents/programming/csharp/

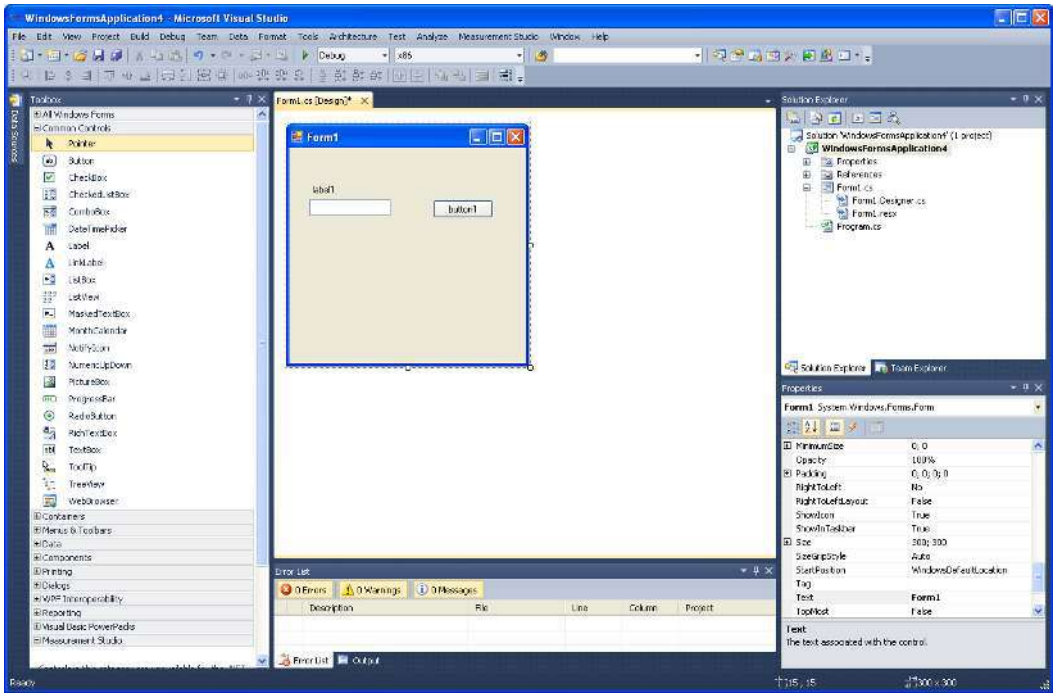
1.1 Visual Studio

Microsoft Visual Studio is an integrated development environment (IDE) from Microsoft. It can be used to develop console and graphical user interface applications along with Windows Forms applications, web sites, web applications, and web services in both native code together with managed code for all platforms supported by Microsoft Windows, Windows Phone, Windows CE, .NET Framework, .NET Compact Framework and Microsoft Silverlight.

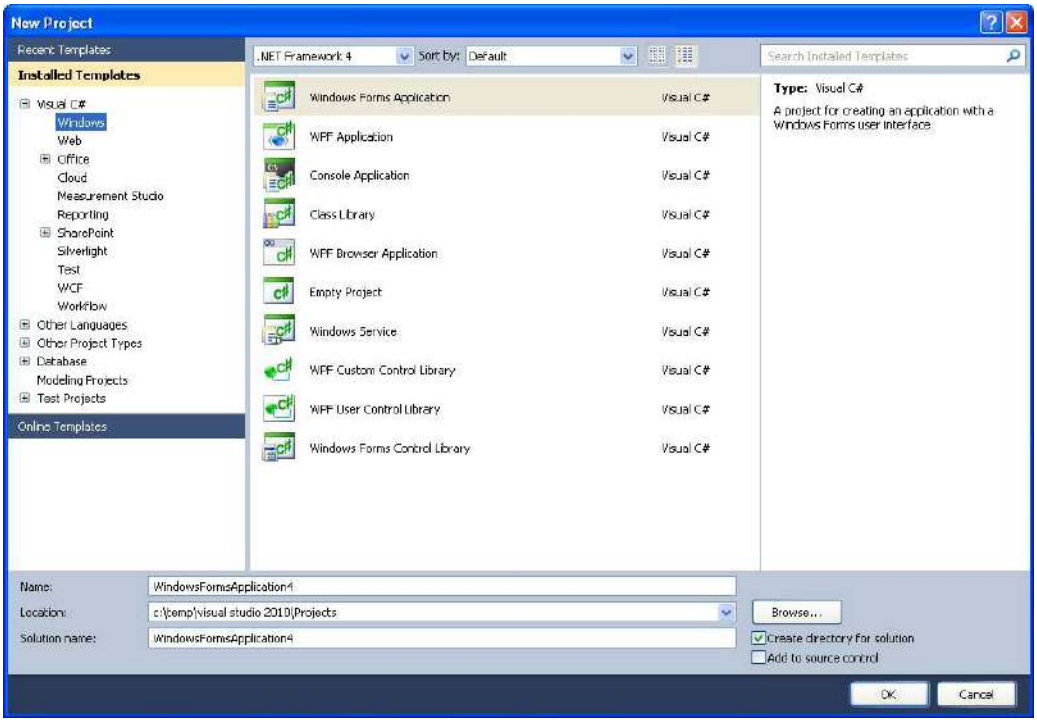
For more information about Visual Studio and C#, visit the following web page:

https://www.halvorsen.blog/documents/programming/csharp/

Below we see the integrated development environment (IDE) in Visual Studio:



New projects are created from the “New Project” window:



1.2 DAQ Hardware

In this Tutorial we will use different hardware available from National Instruments as examples:

- TC-01 Thermocouple device
- USB-6008 DAQ Device
- MyDAQ

1.2.1 NI USB TC-01 Thermocouple Device

Below we see the NI USB-TC01 Thermocouple Measurement device.

We will give code examples of how to use this device in C#. Since this is a DAQmx supported device from National Instruments, the programing structure will be the same as for NI USB-6008.

1.2.2 NI USB-6008 DAQ Device

NI USB-6008 is a simple and low-cost multifunction I/O device from National Instruments.

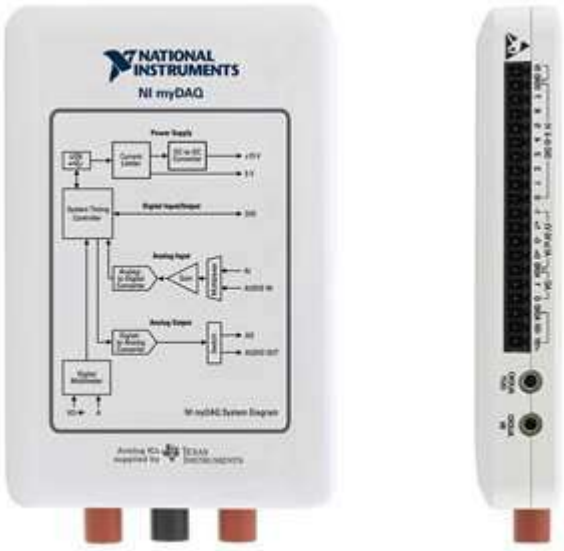
The device has the following specifications:

- 8 analog inputs (12-bit, 10 kS/s)
- 2 analog outputs (12-bit, 150 S/s)
- 12 digital I/O
- USB connection, No extra power-supply neeeded
- Compatible with LabVIEW, LabWindows/CVI, and Measurement Studio for Visual Studio .NET
- NI-DAQmx driver software

The NI USB-6008 is well suited for education purposes due to its small size and easy USB connection.

1.2.3 myDAQ

NI myDAQ is a simple and intuitive DAQ device from National Instruments. NI myDAQ have Analog Inputs (AI), Analog Outputs (AO), Digital Inputs (DI) and Digital Outputs (DO).



Specifications:

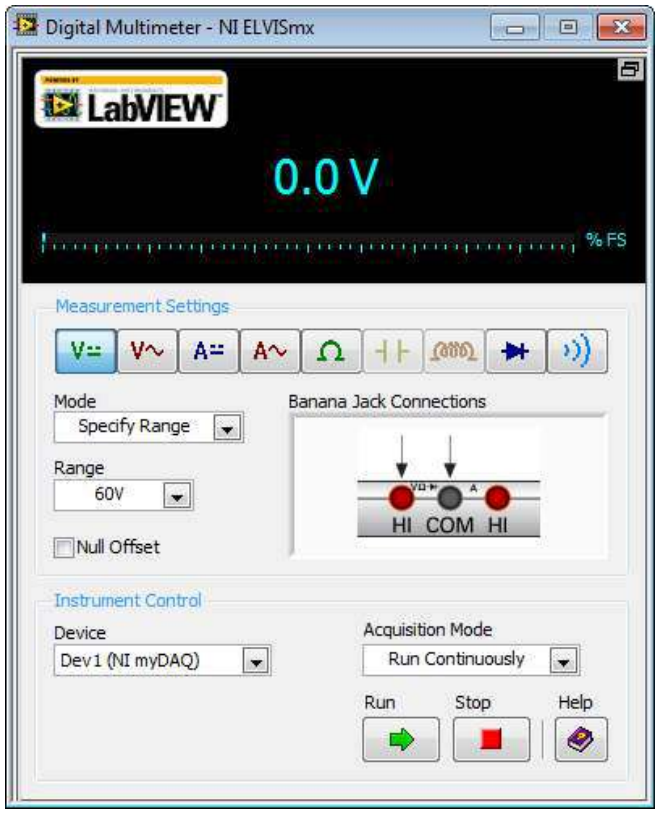
- Two Differential Analog Input and Analog Output Channels (200 ks/s, 16 bit, +/- 10 Volts)
- Eight Digital Input and Digital Output Lines (3.3 Volt TTL-Compatible)
- +5 , +15, and -15 Volt Power Supply Outputs (up to 500m Watts of Power)
- 60 Volt Digital Multimeter (DMM) for Measuring Voltage, Current, and Resistance

In addition to traditional I/O, the myDAQ have a built-in **Digital Multimeter**. The myDAQ can also be used as a **Power Supply**. Using the built-in software the myDAQ can also be used as an **Oscilloscope** and **Function Generator**.

When you plug in the device in the USB connection on your PC, the following will pop-up automatically (NI ELVISmx Instrument Launcher):

Note! You need to install the NI ELVISmx driver software first

If you click on the DMM button, the built-in Digital Multimeter will appear:



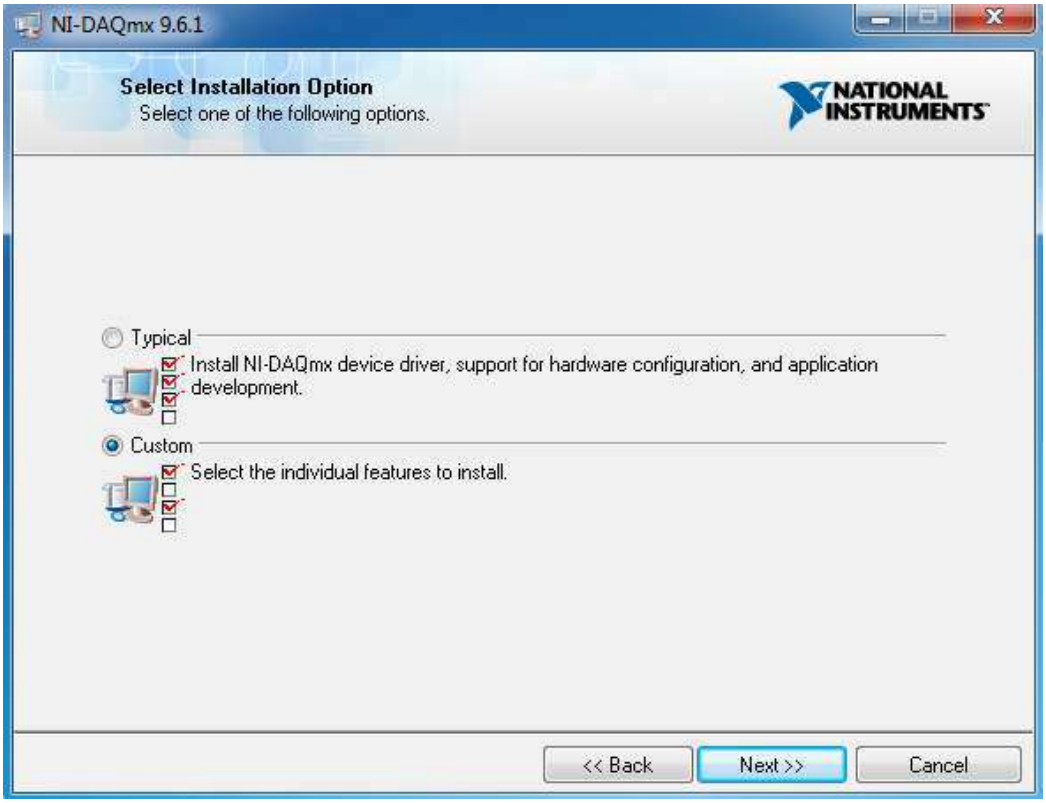
1.3 NI DAQmx driver

National Instruments provides a native .NET API for NI-DAQmx. This is available as a part of the NI-DAQmx driver and does not require Measurement Studio.

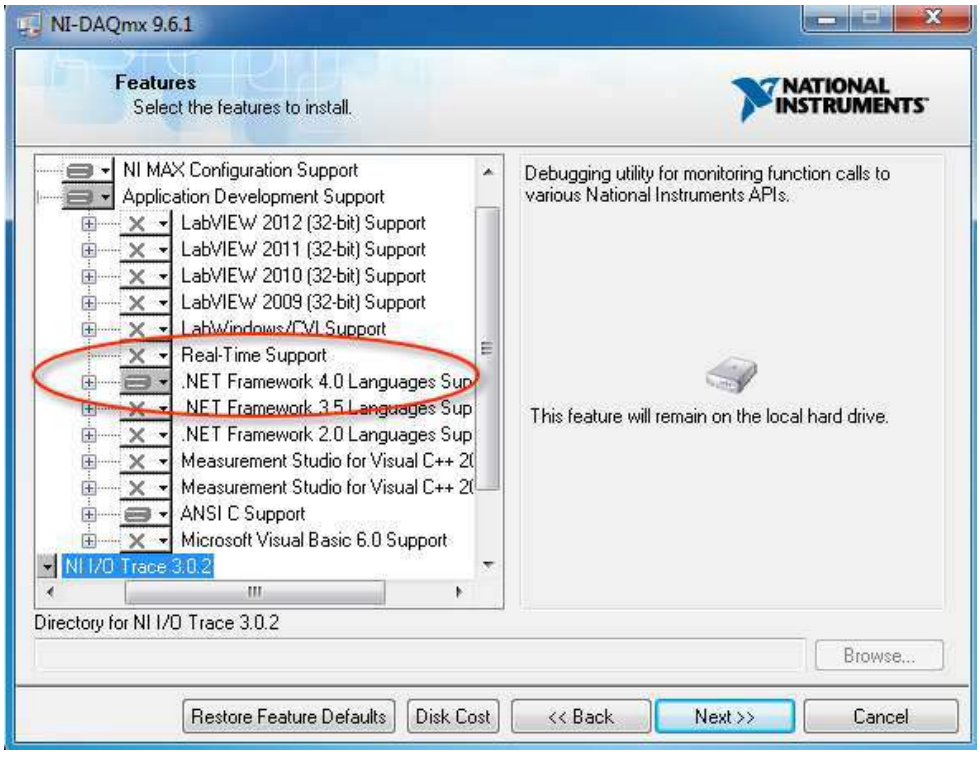
Note! In order to install the DAQmx API for C#, make sure to select ".NET Support" when installing the DAQmx driver.

This application uses the C# API included in the NI DAQmx driver, so make sure that you have installed the NI DAQmx driver in advance.

During the installation make sure to select “Custom”:



Next, make sure that you select .NET Framework X.x Support for the version of .NET that your version of Visual Studio is using:



1.4 Measurement Studio

C# is a powerful programming language, but has few built-in features for measurement and control applications. Measurement Studio is an add-on to Visual Studio which makes it easier to create such applications. With Measurement Studio we can implement Data Acquisition and a graphical HMI.

2 Data Acquisition

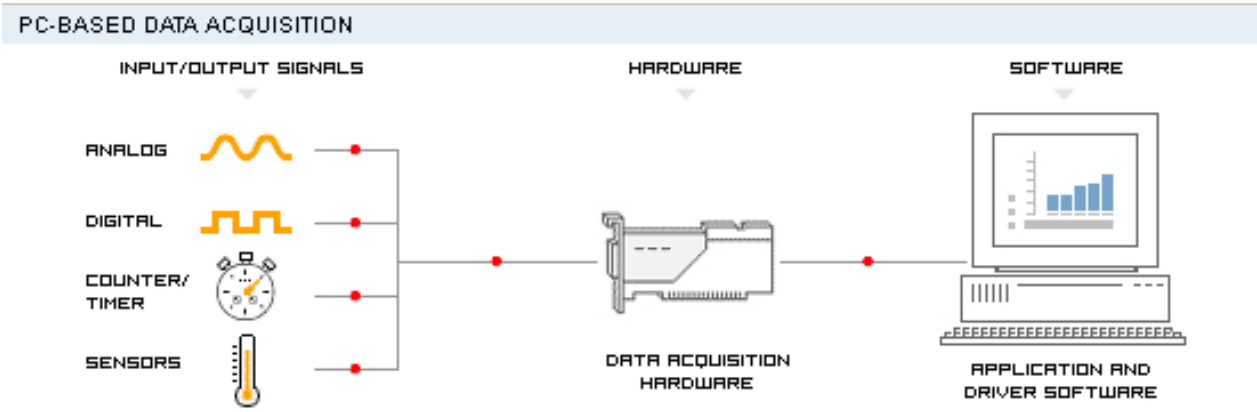
2.1 Introduction

The purpose of data acquisition is to measure an electrical or physical phenomenon such as voltage, current, temperature, pressure, or sound. PC-based data acquisition uses a combination of modular hardware, application software, and a computer to take measurements. While each data acquisition system is defined by its application requirements, every system shares a common goal of acquiring, analyzing, and presenting information. Data acquisition systems incorporate signals, sensors, actuators, signal conditioning, data acquisition devices, and application software.

So summing up, Data Acquisition is the process of:

- Acquiring signals from real-world phenomena
- Digitizing the signals
- Analyzing, presenting and saving the data

The DAQ system has the following parts involved, see Figure:



[Figure: www.ni.com]

The parts are:

- Physical input/output signals
- DAQ device/hardware

- Driver software
- Your software application (Application software)

For an Introduction to Data Acquisition, see this webcast:
http://zone.ni.com/wv/app/doc/p/id/wv-169

2.1.1 Physical input/output signals

A physical input/output signal is typically a voltage or current signal.

2.1.2 DAQ device/hardware

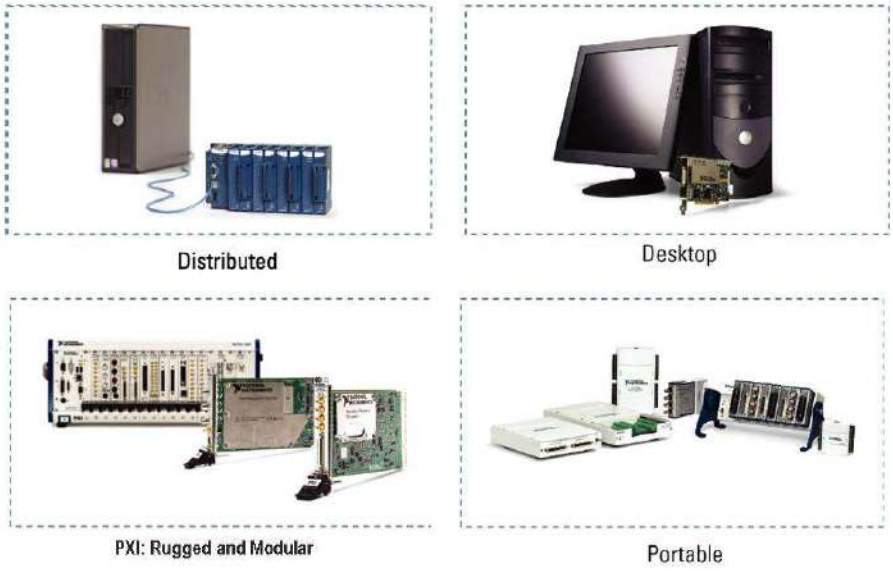
DAQ hardware acts as the interface between the computer and the outside world. It primarily functions as a device that digitizes incoming analog signals so that the computer can interpret them

A DAQ device (Data Acquisition Hardware) usually has these functions:

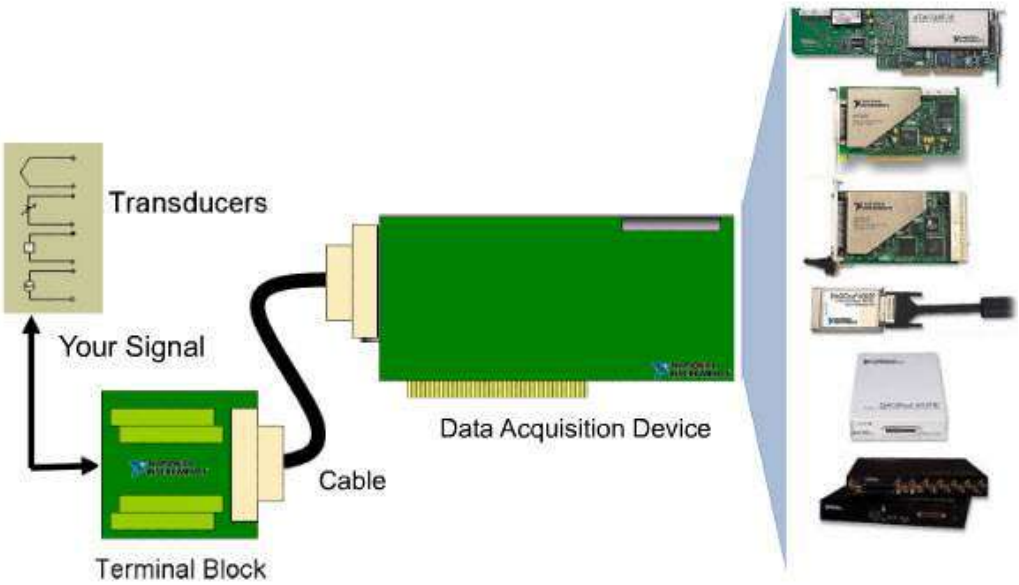
- Analog input
- Analog output
- Digital I/O
- Counter/timers

We have different DAQ devices, such as:

- "**Desktop** DAQ devices" where you need to plug a PCI DAQ board into your computer. The software is running on a computer.
- "**Portable** DAQ devices" for connection to the USB port, Wi-Fi connections, etc. The software is running on a computer
- "**Distributed** DAQ devices" where the software is developed on your computer and then later downloaded to the distributed DAQ device.



[Figure: www.ni.com]



[Figure: www.ni.com]

2.1.3 Driver software

Driver software is the layer of software for easily communicating with the hardware. It forms the middle layer between the application software and the hardware. Driver software also prevents a programmer from having to do register-level programming or complicated commands in order to access the hardware functions.

Driver software from National Instruments: NI-DAQmx

2.1.4 Your software application

Application software adds analysis and presentation capabilities to the driver software. Your software application normally does such tasks as:

- Real-time monitoring
- Data analysis
- Data logging
- Control algorithms
- Human machine interface (HMI)

In order to create your DAQ application you need a programming development tool, such as Visual Studio/C#, LabVIEW, etc..

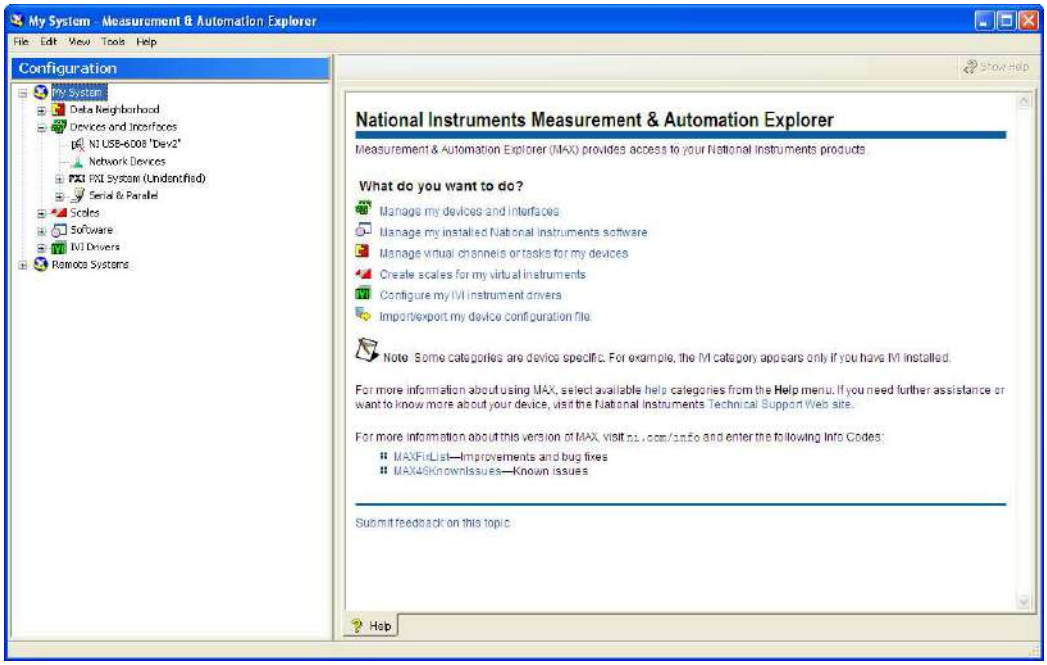
2.2 MAX – Measurement and Automation Explorer

Measurement & Automation Explorer (MAX) provides access to your National Instruments devices and systems.

With MAX, you can:

- Configure your National Instruments hardware and software
- Create and edit channels, tasks, interfaces, scales, and virtual instruments
- Execute system diagnostics
- View devices and instruments connected to your system
- Update your National Instruments software

In addition to the standard tools, MAX can expose item-specific tools you can use to configure, diagnose, or test your system, depending on which NI products you install. As you navigate through MAX, the contents of the application menu and toolbar change to reflect these new tools.



2.3 DAQ in Visual Studio

We can create DAQ applications with or without Measurement Studio. In both situations you need the NI-DAQmx driver library.

2.3.1 NI-DAQmx

National Instruments provides a native .NET API for NI-DAQmx. This is available as a part of the NI-DAQmx driver and does not require Measurement Studio.

In general, data acquisition programming with DAQmx involves the following steps:

- Create a Task and Virtual Channels
- Start the Task
- Perform a Read operation from the DAQ
- Perform a Write operation to the DAQ
- Stop and Clear the Task.

Data acquisition in text based-programming environment is very similar to the LabVIEW NI-DAQmx programming as the functions calls is the same as the NI-DAQmx VI's.

Using NI-DAQmx in Text Based Programming Environments:

http://zone.ni.com/devzone/cda/tut/p/id/5409#toc4

2.3.2 Examples

Examples installed as part of NI-DAQmx:

The location of examples will depend on the version of Visual Studio and is listed in the following Developer Zone Article: Using NI-DAQmx in Text Based Programming Environments. The most common location is:

C:\Documents and Settings\All Users\Documents\National Instruments\NI-DAQ\Examples\DotNET<.NET Framework Version>\

Newest examples for .NET x.x and Visual Studio 20xx:

C:\Documents and Settings\All Users\Documents\National Instruments\NI-DAQ\Examples\DotNETx.x\

Sub-folders named CS contain C# examples. These examples install with NI-DAQmx. Measurement Studio is not required to install the class libraries or the examples.

Note! If the paths above do not exist, be sure you have .NET support installed for NI-DAQmx.

3 My First DAQ App with USB-6008 using DAQmx Driver

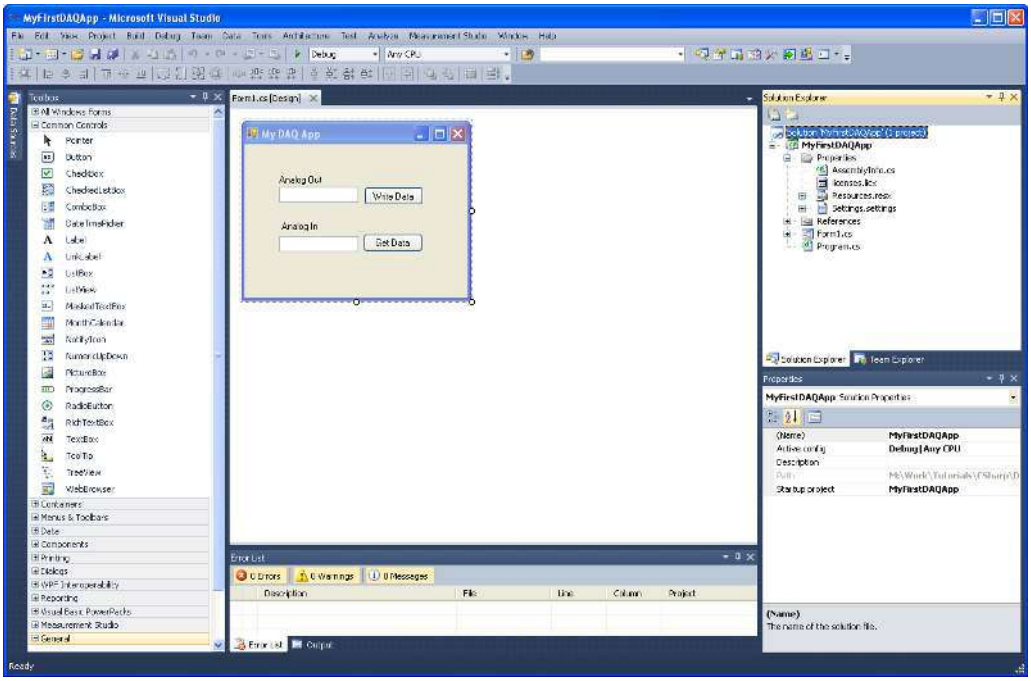
We will create a simple application in Visual Studio that uses a NI USB-6008 DAQ device.

3.1 Introduction

This application uses the C# API included in the NI DAQmx driver, so make sure that you have installed the NI DAQmx driver in advance. You don't need Measurement Studio to create this application.

Start Visual Studio and create a new Windows Forms application.

We will create the following application in Visual Studio 2010 and C#:



The User Interface looks like this:

We start by connecting the Analog In and Analog Out wires together (a so called Loopback test).

When we click the “Write Data” button, the value entered in the text box “Analog Out” will be written to the DAQ device. If we have connected the Analog In and Analog Out wires together we will read the same value in the “Analog In” textbox when we push the “Get Data” button.

3.2 Example

We will go through the different parts of the code in detail.

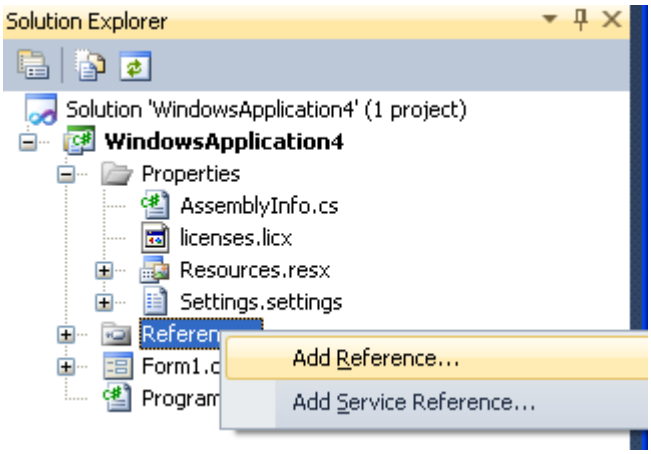
3.2.1 Add References to DAQmx Driver

This application uses the C# API included in the NI DAQmx driver, so make sure that you have installed the NI DAQmx driver in advance.

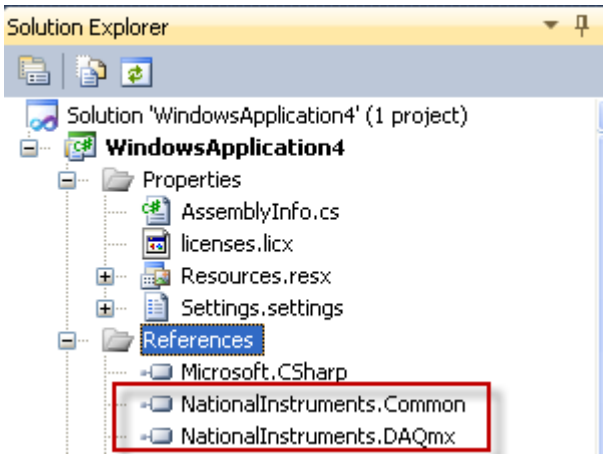
We need to add the following Assembly references:

- NationalInstruments.Common
- NationalInstruments.DAQmx

We add References by right-clicking the References node in the Solution Explorer:



When we have added the necessary References, the Solution Explorer will look like this:



3.2.2 Initialization

We need to add the following Namespaces:

```
using NationalInstruments;
using NationalInstruments.DAQmx;
```

3.2.3 Analog Out

We implement the code for writing to the Analog Out channel in the Event Handler for the "Write Data" button:

```
    private void btnWriteAnalogOut_Click(object sender, EventArgs e)
      {

          Task analogOutTask = new Task();

          AOChannel myAOChannel;

          myAOChannel = analogOutTask.AOChannels.CreateVoltageChannel(
              "dev1/ao0",
              "myAOChannel",
              0,
              5,
              AOVoltageUnits.Volts
              );

          AnalogSingleChannelWriter writer = new
           AnalogSingleChannelWriter(analogOutTask.Stream);

          double analogDataOut;

          analogDataOut = Convert.ToDouble(txtAnalogOut.Text);

          writer.WriteSingleSample(true, analogDataOut);

      }
```

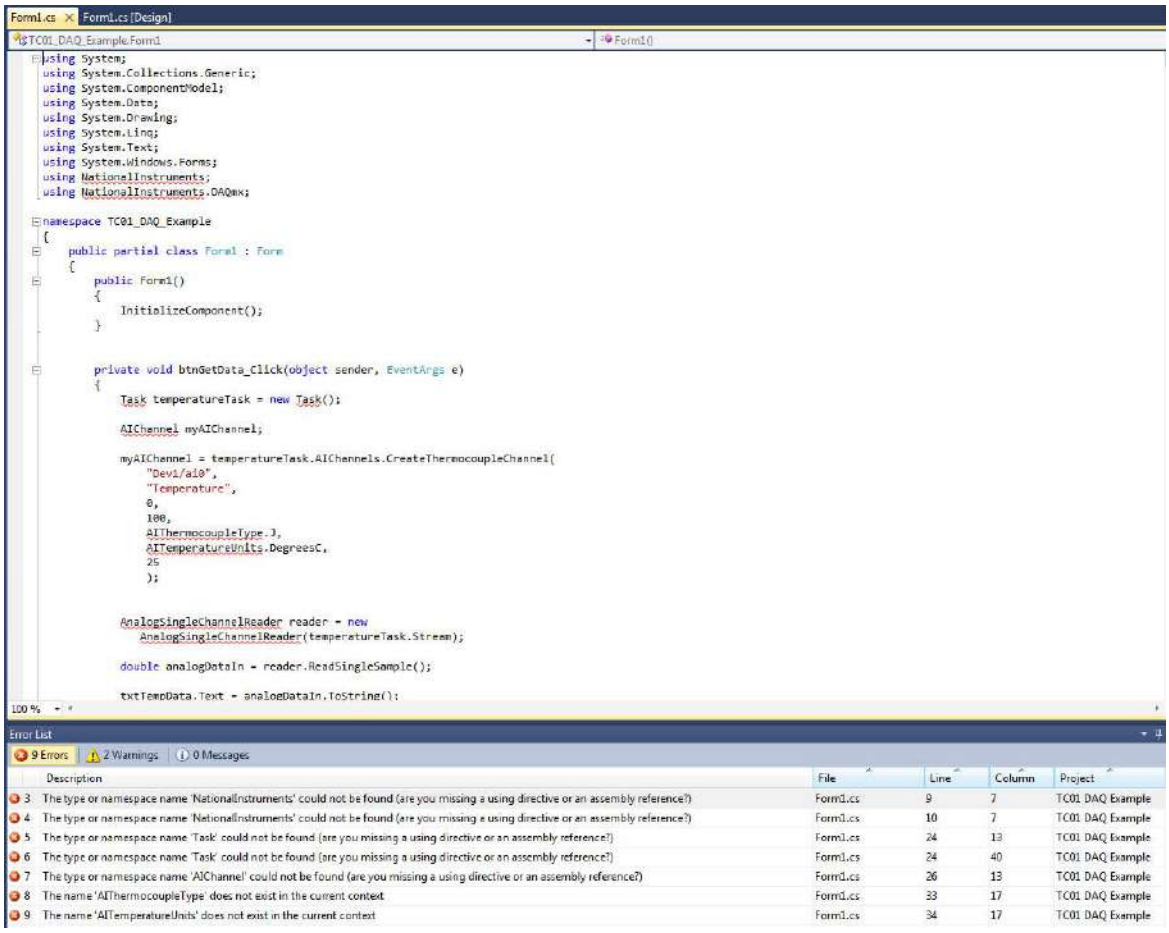
3.2.4 Analog In

We implement the code for reading data from the Analog In channel in the Event Handler for the "Get Data" button:

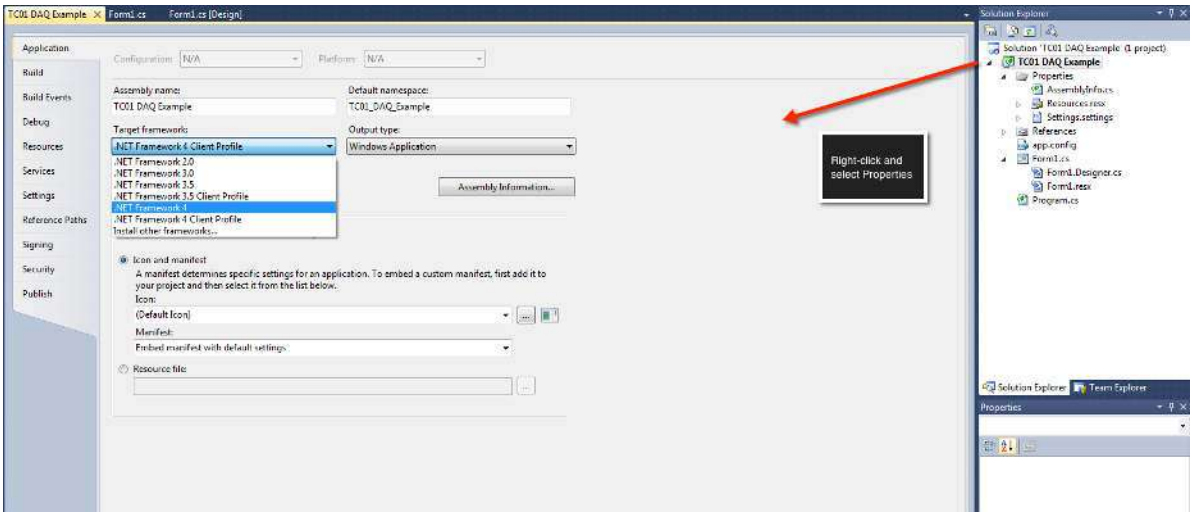
```
private void btnGetAnalogIn_Click(object sender, EventArgs e)
      {

          Task analogInTask = new Task();

          AIChannel myAIChannel;

          myAIChannel = analogInTask.AIChannels.CreateVoltageChannel(
              "dev1/ai0",
              "myAIChannel",
              AITerminalConfiguration.Differential,
              0,
              5,
              AIVoltageUnits.Volts
              );

          AnalogSingleChannelReader reader = new
           AnalogSingleChannelReader(analogInTask.Stream);

          double analogDataIn = reader.ReadSingleSample();

          txtAnalogIn.Text = analogDataIn.ToString();

      }
```

3.2.5 Error?

If your application runs without error that's fine, but perhaps you get the following error:



In order to fix the problem, open the Properties for your project:



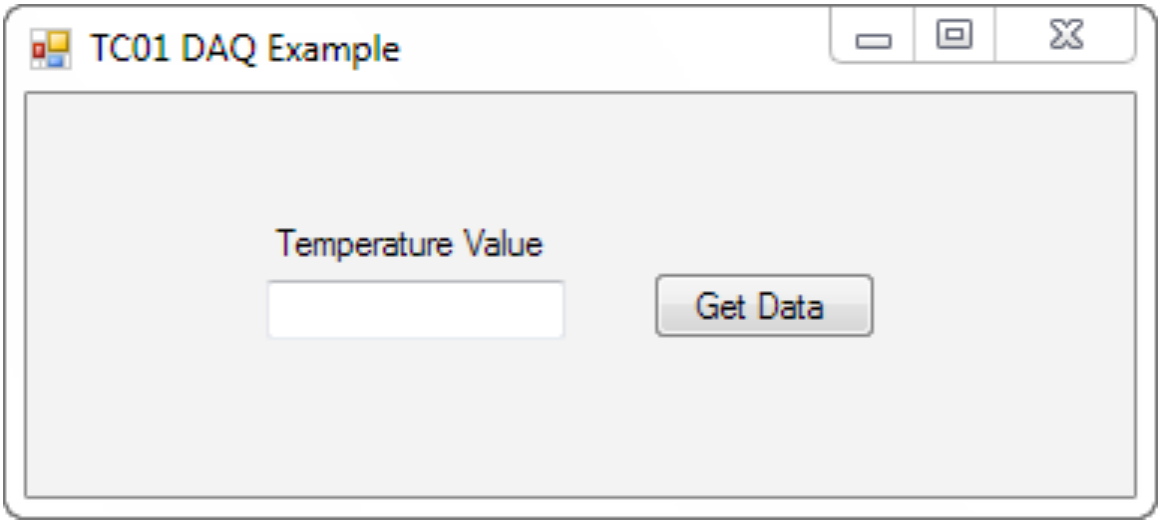
Make sure to select ".NET Framework X" in the "Target framework" drop-down menu.

4 Temperature Logging with TC-01 Thermocouple Device

In this example we will use a NI TC-01 USB Thermocouple device.

4.1 Example

We will create the following simple application:

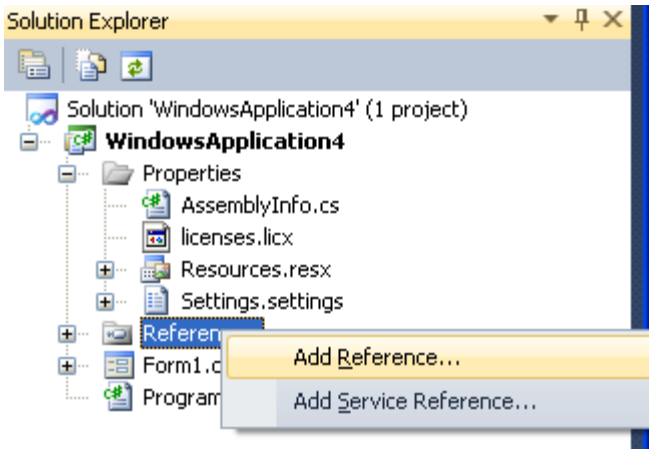


4.1.1 Add References to DAQmx Driver

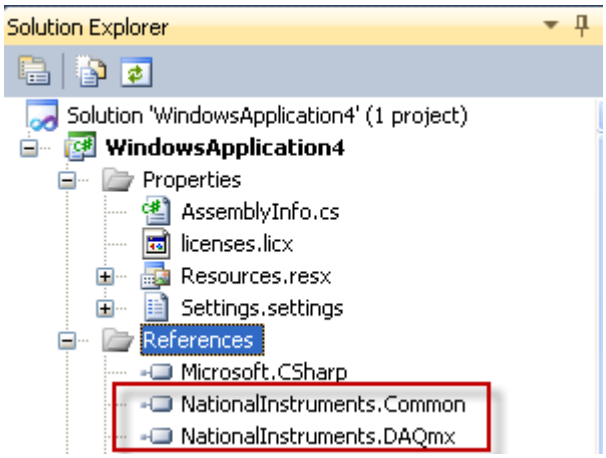
We need to add the following Assembly references:

- NationalInstruments.Common
- NationalInstruments.DAQmx

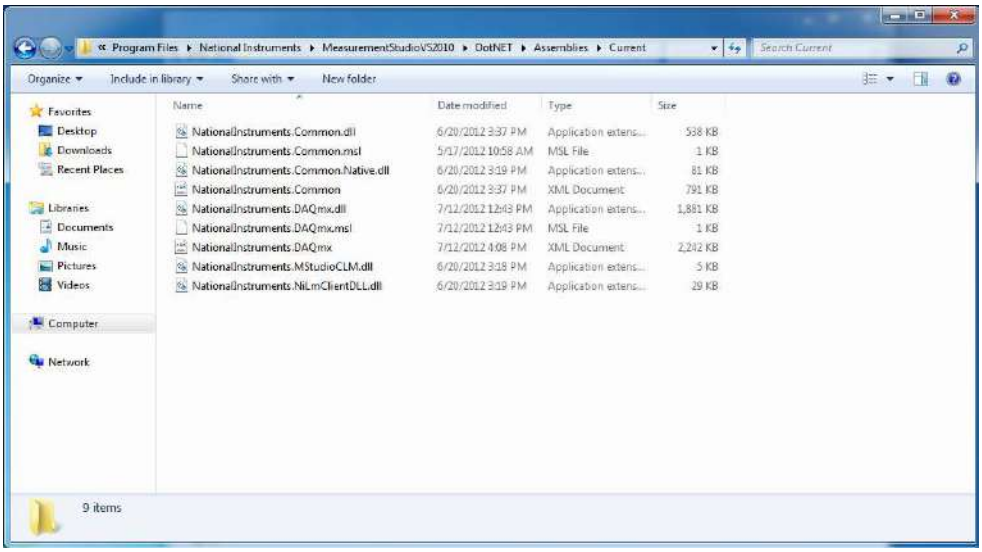
We add References by right-clicking the References node in the Solution Explorer:



When we have added the necessary References, the Solution Explorer will look like this:



The dlls are normally to find here:



4.1.2 Initialization

We need to add the following Namespaces:

```
using NationalInstruments;
using NationalInstruments.DAQmx;
```

4.1.3 Read Temperature Data

Enter the following code lines:

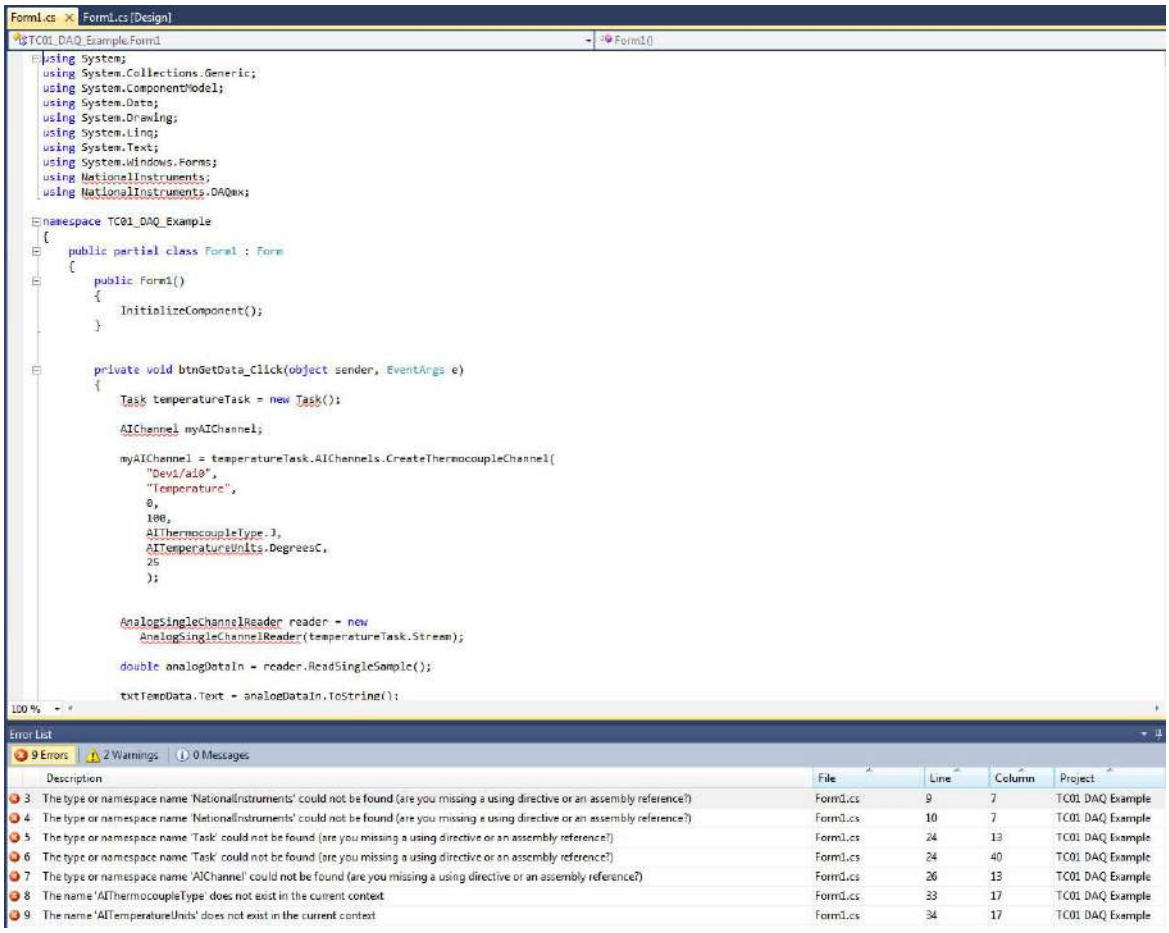
```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using NationalInstruments;
using NationalInstruments.DAQmx;

namespace TC01_DAQ_Example
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }


        private void btnGetData_Click(object sender, EventArgs e)
        {
            Task temperatureTask = new Task();

            AIChannel myAIChannel;

            myAIChannel = temperatureTask.AIChannels.CreateThermocoupleChannel(
                "Dev1/ai0",
                "Temperature",
                0,
                100,
                AIThermocoupleType.J,
                AITemperatureUnits.DegreesC,
                25
                );


            AnalogSingleChannelReader reader = new
              AnalogSingleChannelReader(temperatureTask.Stream);

            double analogDataIn = reader.ReadSingleSample();

            txtTempData.Text = analogDataIn.ToString();

        }
    }
}
```

4.1.4 Test your application

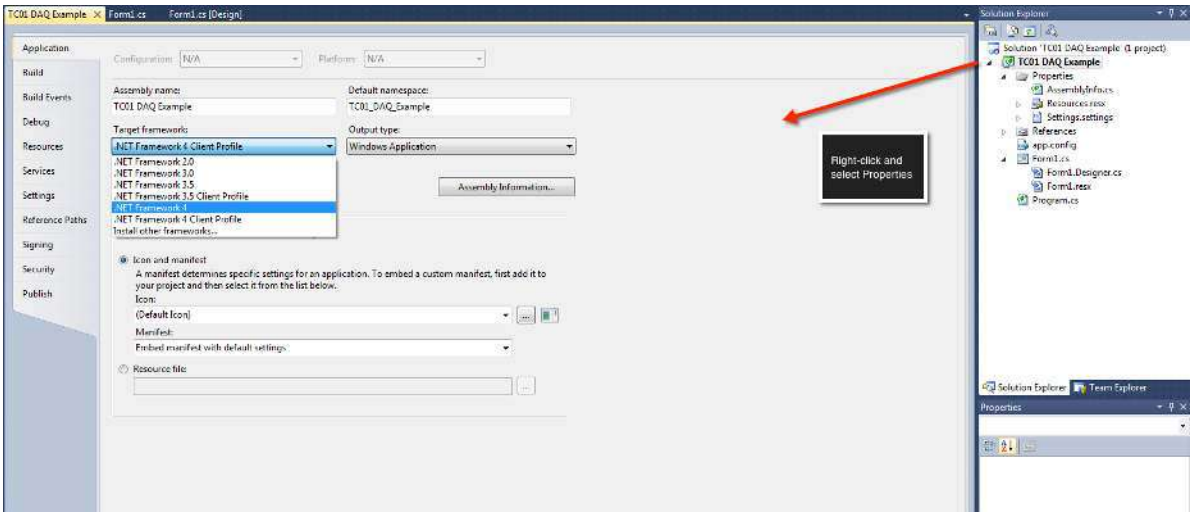
Run your application. When clicking the button, you should now retrieve data from the temoerature device.

4.1.5 Error?

If your application runs without error that's fine, but perhaps you get the following error:



In order to fix the problem, open the Properties for your project:

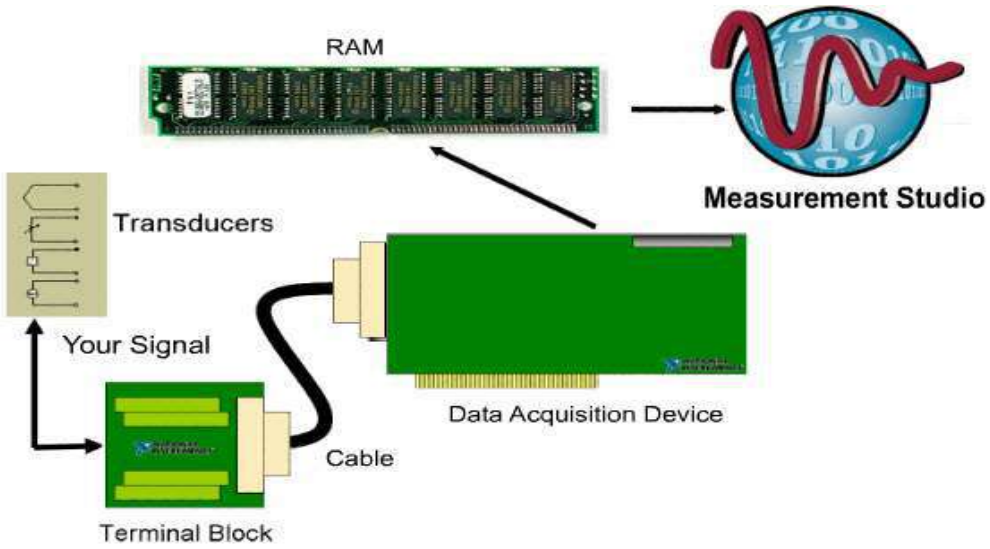


Make sure to select ".NET Framework X" in the "Target framework" drop-down menu.

5 Measurement Studio

5.1 Introduction

C# is a powerful programming language, but has few built-in features for measurement and control applications. Measurement Studio is an add-on to Visual Studio which makes it easier to create such applications. With Measurement Studio we can implement Data Acquisition and a graphical HMI.

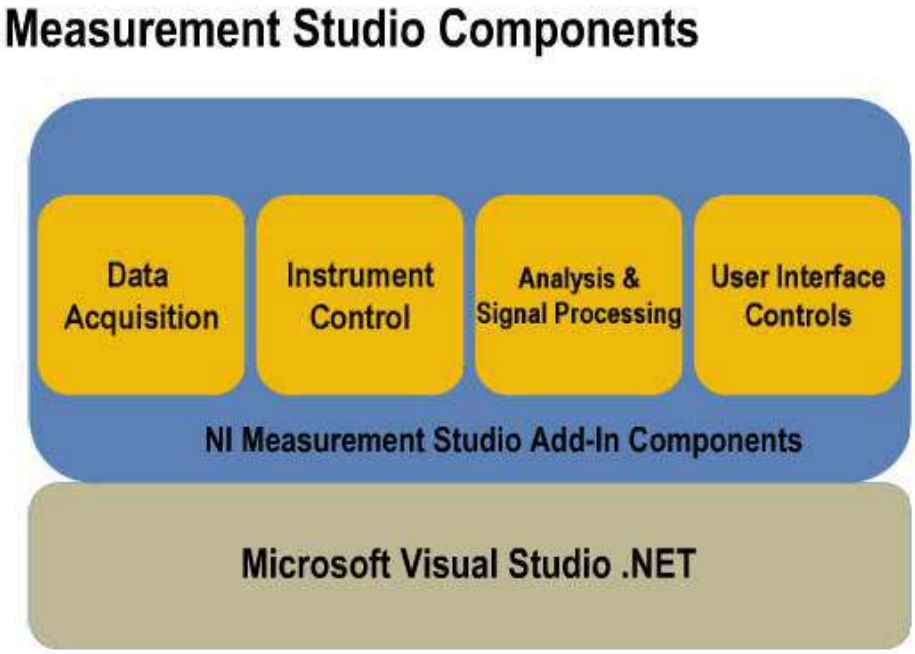


[Figure: www.ni.com]

Measurement Studio for Visual C# .NET provides:

- Managed .NET controls for creating rich Web and Windows GUIs
- Multithreaded API for data acquisition

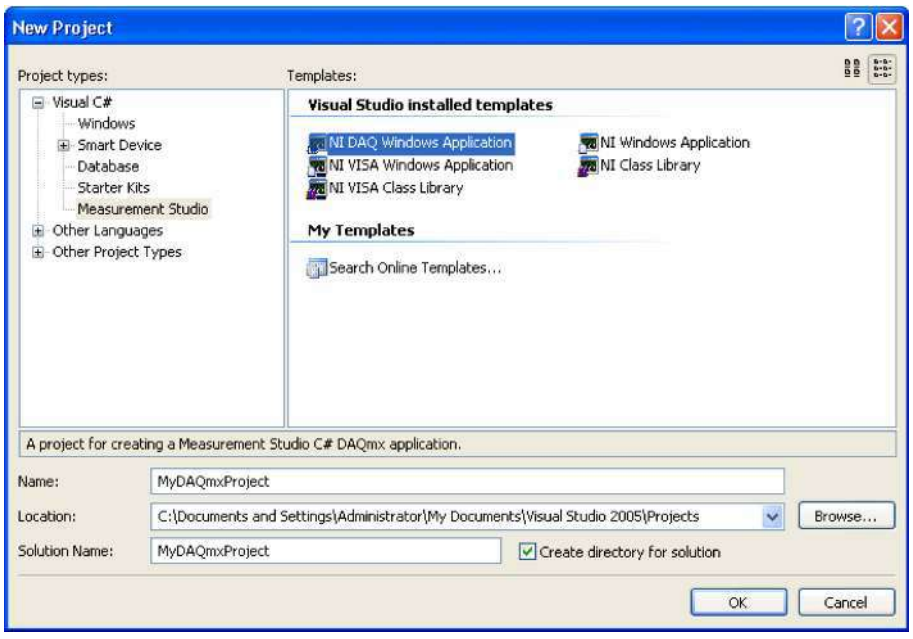
- Instrument control APIs
- Analysis libraries designed for engineers and scientists



[Figure: www.ni.com]

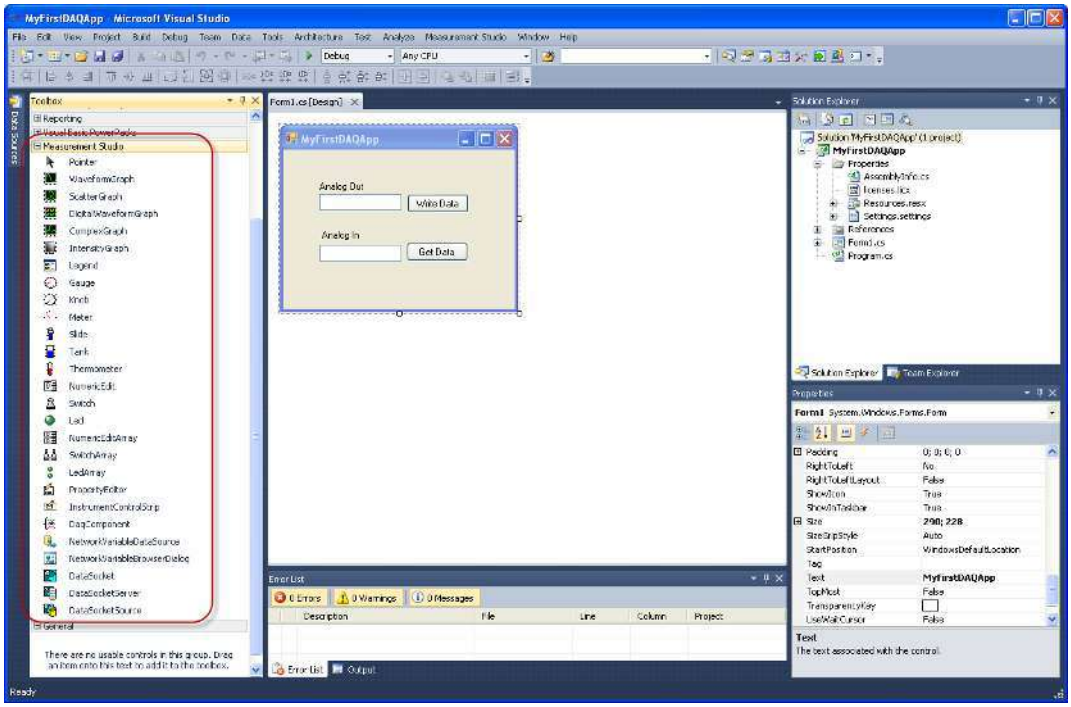
5.2 Templates

Measurement Studio has several Templates that make it easier to build DAQ applications.

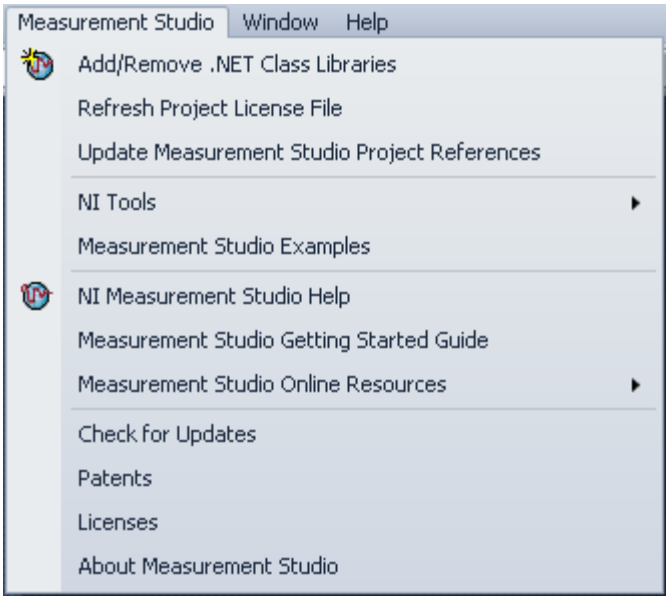


5.3 Toolbox

Below we see the Toolbox in Visual Studio that is installed with Measurement Studio:



In addition to the Toolbox, Measurement Studio also installs the following menu item:

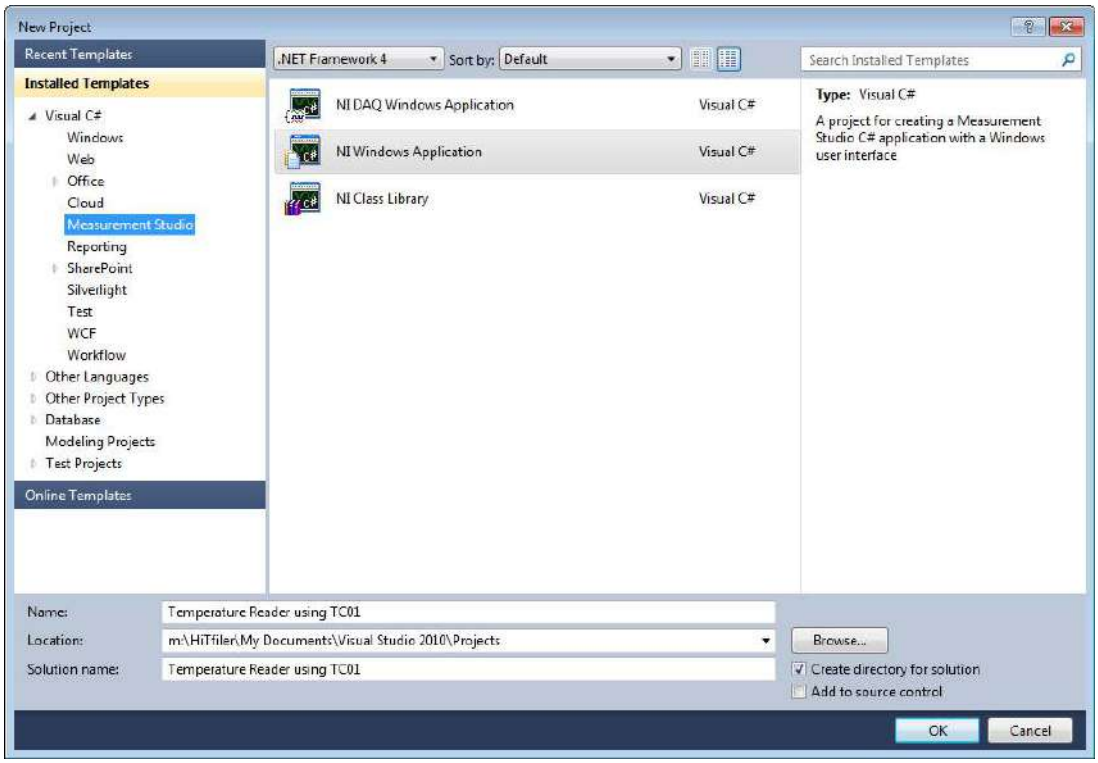


5.4 Logging Temperature Data with TC-01 Thermocouple Example

In this example we will use a NI TC-01 USB Thermocouple device.

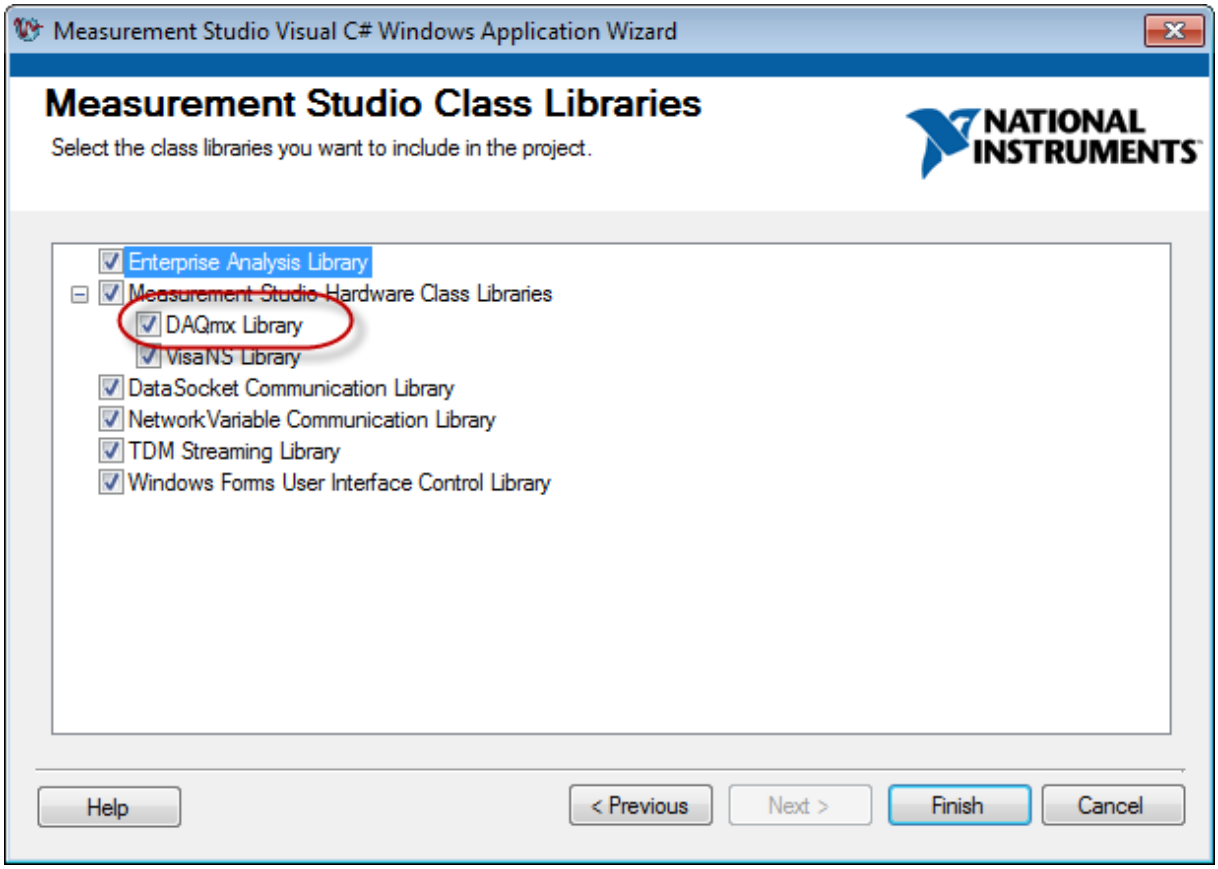
5.4.1 Select Template

In this example, we will select the “NI windows Application” Template in the “New Project” window.



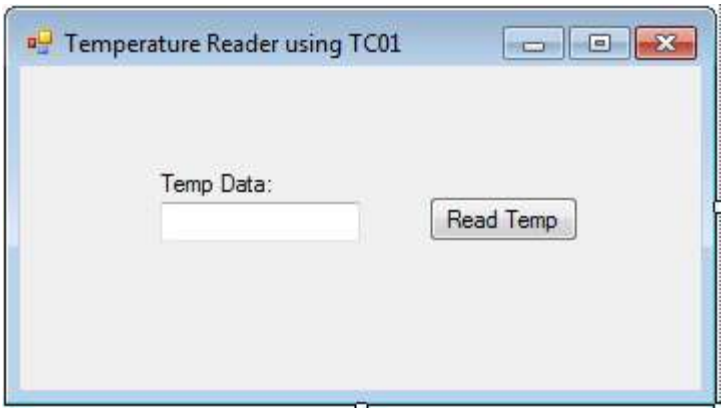
5.4.2 Select Class Libraries

Next we select the Class Libraries we want to include. In our case we need at least to select the “DAQmx Library”.



When we click “Finish” an empty project will be created for us. We are now ready to create our own application.

We start by creating a simple User Interface:



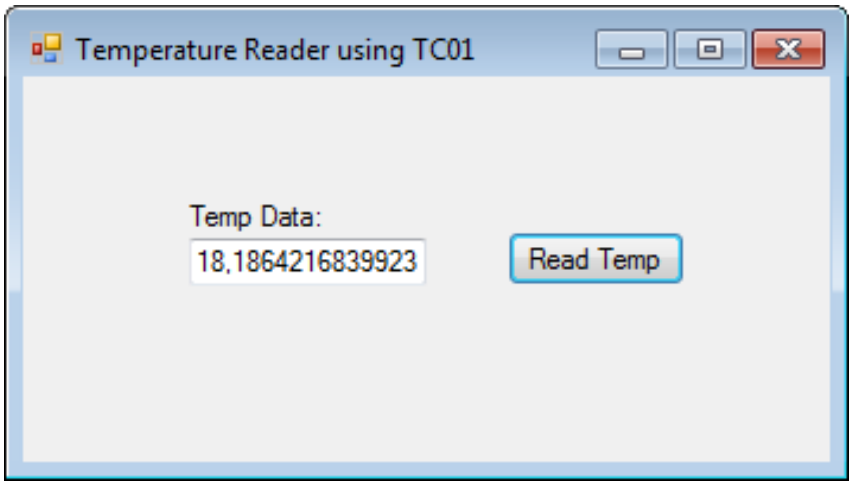
When we click the “Read Temp” button, the Temperature data shall be shown in the “Temp Data” TextBox.

In the event Handler for the “Read Temp” button, we create the following code:

```
private void btnReadTempData_Click(object sender, EventArgs e)
```

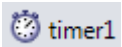
```
        {

            Task analogInTask = new Task();

            AIChannel myAIChannel;

            myAIChannel = analogInTask.AIChannels.CreateThermocoupleChannel(
                "Dev1/ai0",
                "Temperature",
                0,
                100,
                AIThermocoupleType.J,
                AITemperatureUnits.DegreesC,
                25
                );


            AnalogSingleChannelReader reader = new
              AnalogSingleChannelReader(analogInTask.Stream);

            double analogDataIn = reader.ReadSingleSample();

            txtTempData.Text = analogDataIn.ToString();

        }
```

Then we can test our application:



5.4.3 Using a Timer

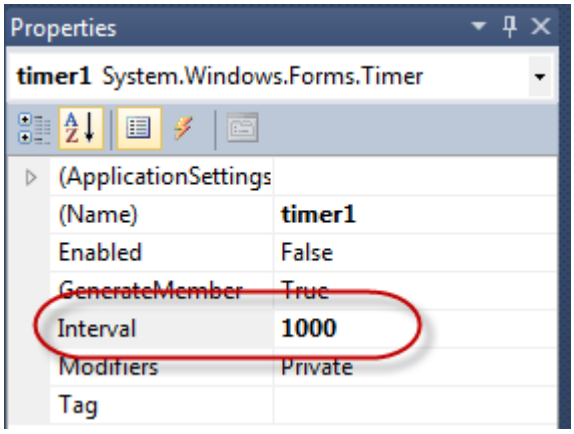
Improvements: We should use a **"Timer"** in order to read Temperature data at specific intervals instead of pushing a button each time we need data.



We drag in a Timer component from the Toolbox. First, we need to start the Timer:

```
public Form1()
        {
            InitializeComponent();

            timer1.Start();
        }
```

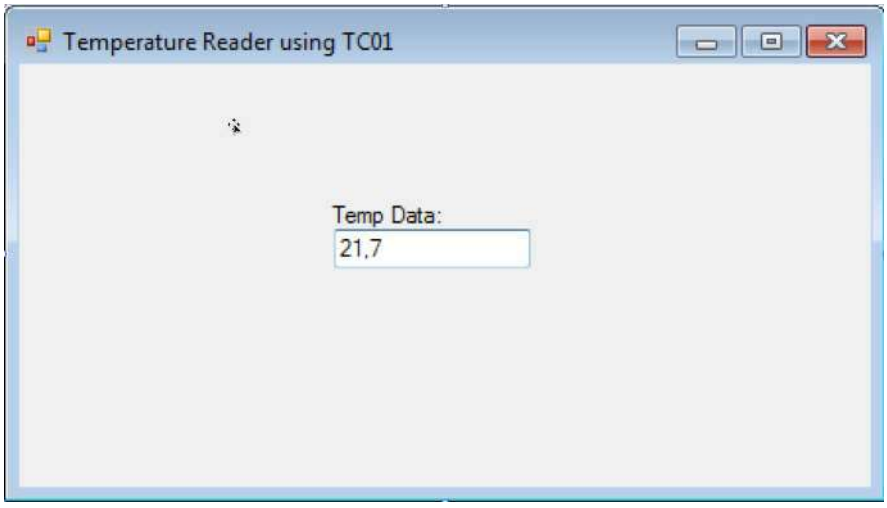
Next, we need to specify the interval. We can do that in the **Properties** window:



In the Timer Event we write the code for reading the Temperature:

```
private void timer1_Tick(object sender, EventArgs e)
       {

          Task analogInTask = new Task();


          AIChannel myAIChannel;


          myAIChannel = analogInTask.AIChannels.CreateThermocoupleChannel(
              "Dev1/ai0",
              "Temperature",
              0,
              100,
              AIThermocoupleType.J,
              AITemperatureUnits.DegreesC,
              25
              );


          AnalogSingleChannelReader reader = new
             AnalogSingleChannelReader(analogInTask.Stream);

          double analogDataIn = reader.ReadSingleSample();


          txtTempData.Text = analogDataIn.ToString("0.0");

       }
```

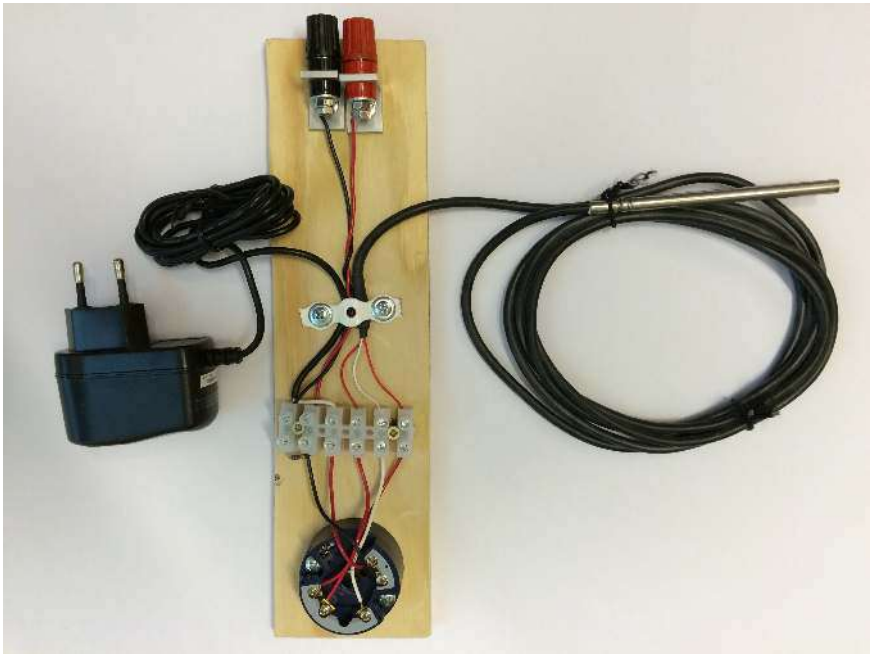
Finally, we test the application:



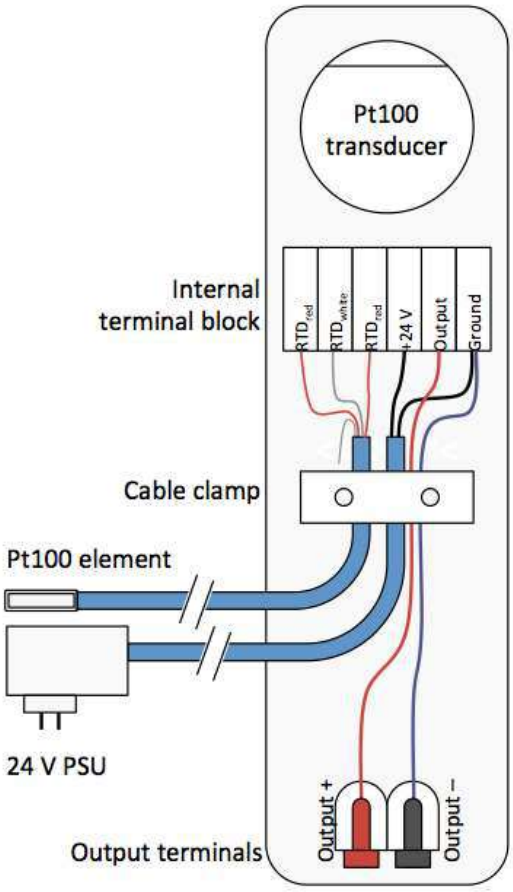
5.5 Logging Temperature Data with USB-6008 Example

In this example we will log temperature data using the USB-6008 device.

In addition we will use a Pt-100 sensor that we connect to one of the Analog Input channels on the USB-6008 device:



The Output from this device is between 1-5V that illustrates a temperature between 0 and 100 degrees Celsius.



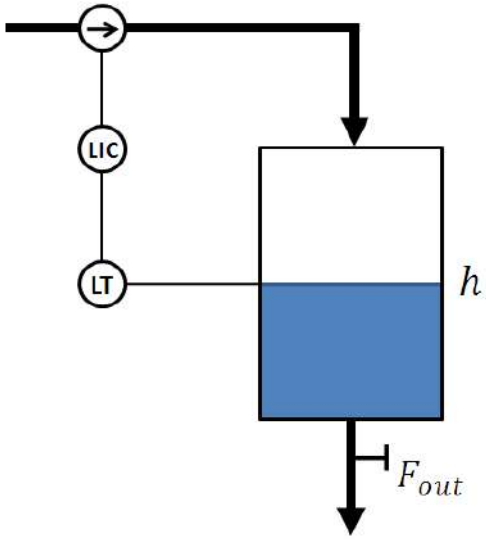
The procedure and code is identical as the previous example, except a small part as shown below:

```
private void timer1_Tick(object sender, EventArgs e)
       {

           Task analogInTask = new Task();


           AIChannel myAIChannel;


           myAIChannel = analogInTask.AIChannels.CreateVoltageChannel(
               "dev1/ai0",
               "myAIChannel",
               AITerminalConfiguration.Differential,
               1,
               5,
               AIVoltageUnits.Volts
               );


           AnalogSingleChannelReader reader = new
              AnalogSingleChannelReader(analogInTask.Stream);

           double analogDataIn = reader.ReadSingleSample();


           txtTempData.Text = analogDataIn.ToString("0.0");

       }
```

6 Control Application

6.1 Introduction to the Example

In this example we will use Measurement Studio to create a simple control application. We will control the level in a water tank using manual control. The process is as follows:

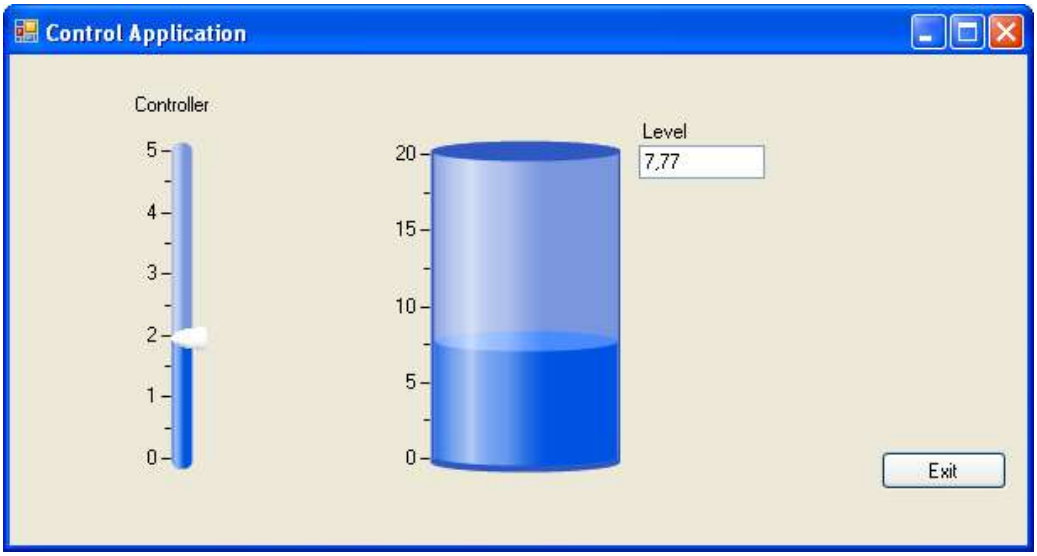


In this example we will use a small-scale laboratory process called LM-900 Level System:

We want to control the level in the water tank using a pump on the inflow. We will read the level using our USB-6008 DAQ device (Analog In) and write the control signal (Analog Out) to the DAQ device.

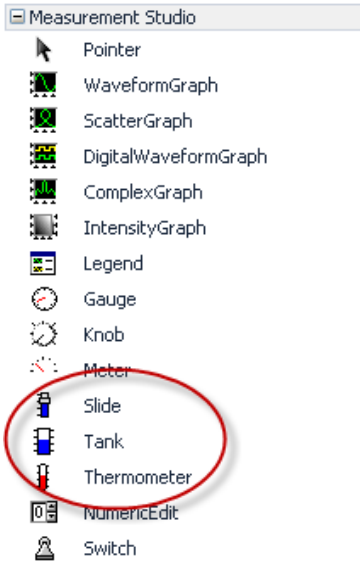
The Analog Out (control signal) will be a signal between $0 - 5V$ and the Analog In (Level) will be a $0 - 5V$ signal that we need to scale to $0 - 20cm$.

We will create the following application:



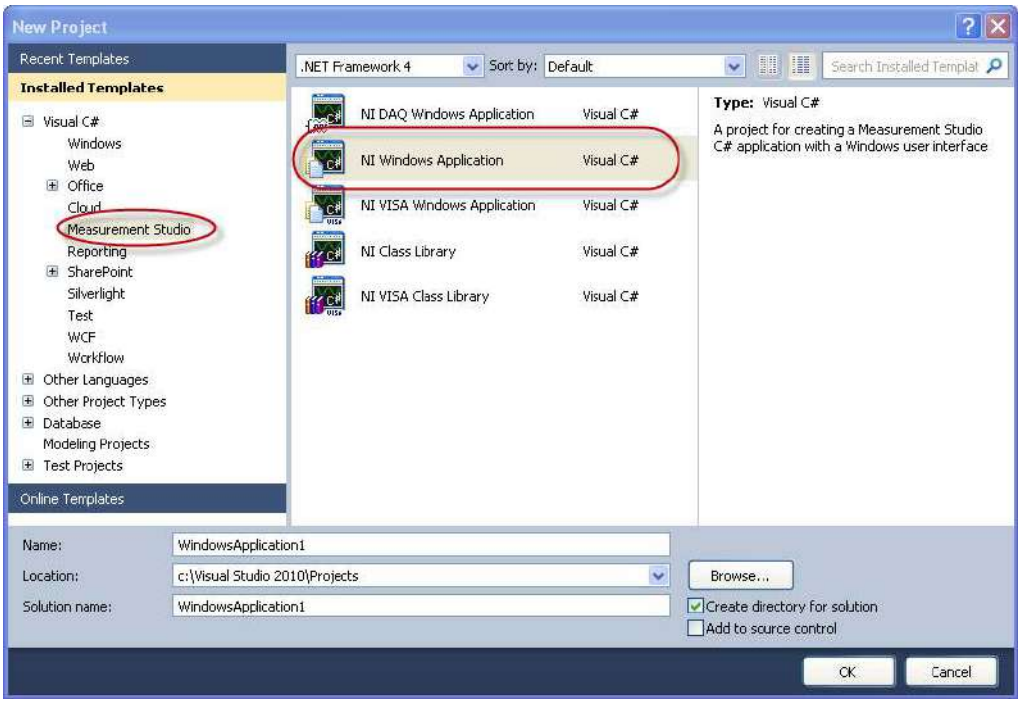
We will use a “Slider” to manually adjust the control signal and a Tank to indicate the level in the real process.

In this example we will use the “Slide” control and “Tank” control that comes with Measurement Studio.



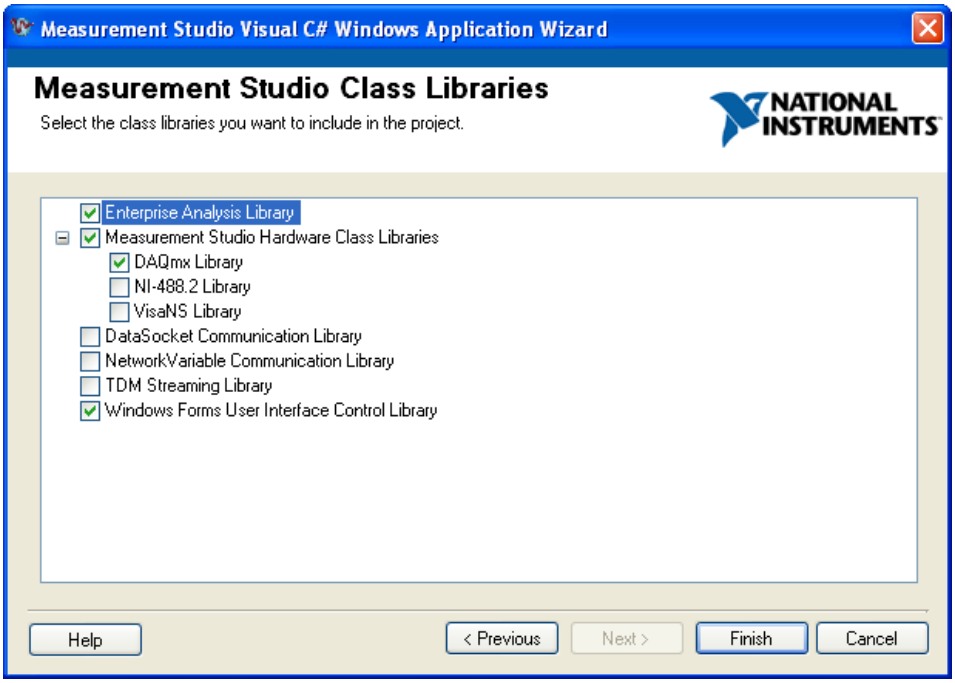
6.2 Coding

Start a New Project in Visual Studio:

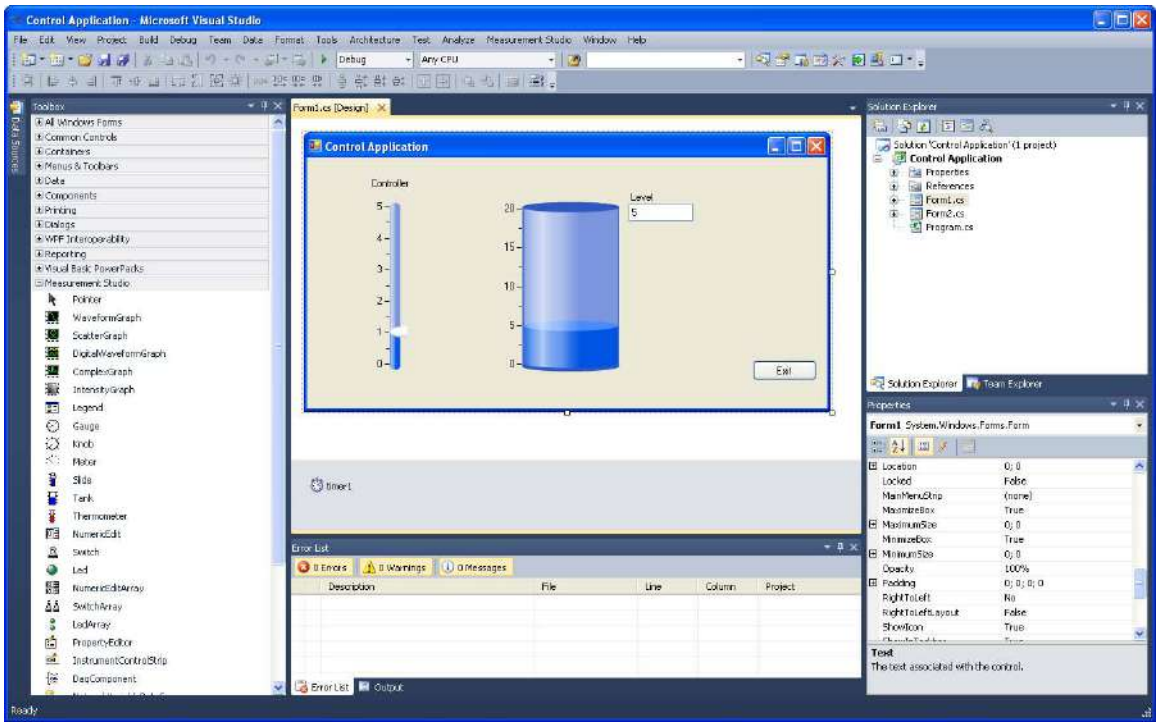


Select the “NI Windows Application” Template in the “Measurement Studio” node.

In the window that appears next, select the Libraries you want to include:



We create the User Interface above in Visual Studio, and it looks like this:



For the read and write operations we have created a simple Class with two methods:

```
public class DaqData
    {

        public double ReadDaqData()
        {
            ...
        }


        public void WriteDaqData(double analogDataOut)
```

```
        {
            ...
        }
    }
```

More about the **ReadDaqData()** and **WriteDaqData()** methods below.

6.2.1 Read Level

The ReadDaqData() method handles the logic for reading from the DAQ device:

```
public double ReadDaqData()
        {

            Task analogInTask = new Task();

            AIChannel myAIChannel;

            myAIChannel = analogInTask.AIChannels.CreateVoltageChannel(
                "dev1/ai0",
                "myAIChannel",
                AITerminalConfiguration.Differential,
                0,
                5,
                AIVoltageUnits.Volts
                );

            AnalogSingleChannelReader reader = new
             AnalogSingleChannelReader(analogInTask.Stream);

            double analogDataIn = reader.ReadSingleSample();

            return analogDataIn;
        }
```

6.2.2 Write Control Value

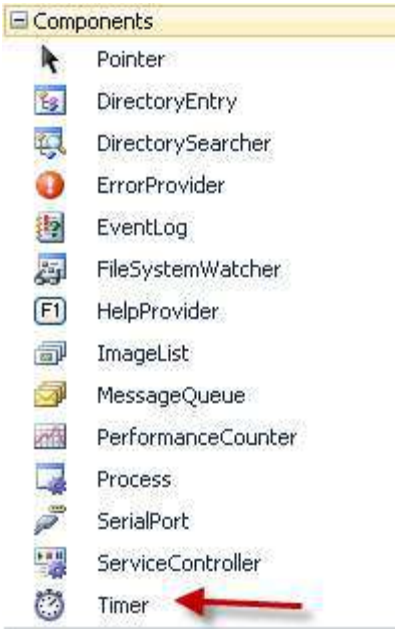
The WriteDaqData() method handles the logic for writing to the DAQ device:

```
  public void WriteDaqData(double analogDataOut)
        {

            Task analogOutTask = new Task();

            AOChannel myAOChannel;

            myAOChannel = analogOutTask.AOChannels.CreateVoltageChannel(
                "dev1/ao0",
                "myAOChannel",
                0,
                5,
                AOVoltageUnits.Volts
                );

            AnalogSingleChannelWriter writer = new
             AnalogSingleChannelWriter(analogOutTask.Stream);

            writer.WriteSingleSample(true, analogDataOut);

        }
```

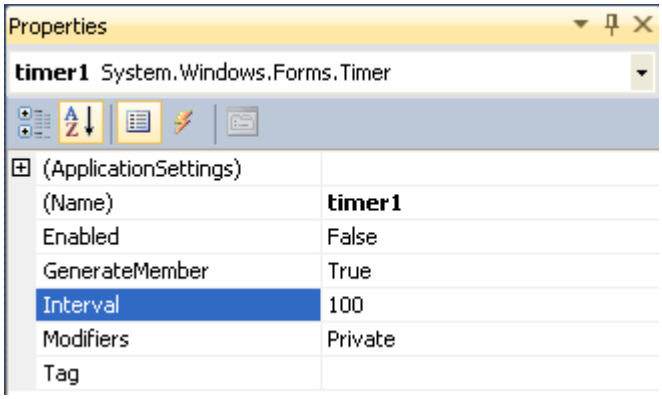
6.2.3 Using a Timer

In the previous example (“My First DAQ App”) we was reading and writing to the DAQ device when clicking a button, but in an ordinary application this is not a good solution. In order to read and write data on regular intervals we will use a “Timer”.

In the “Components” toolbox we find the “Timer” Control:



In the Properties window we can specify the Interval (“Sampling Time”) in milliseconds.



We can start the timer with the following code:

```
public Form1()
       {
           InitializeComponent();

           timer1.Start();
       }
```

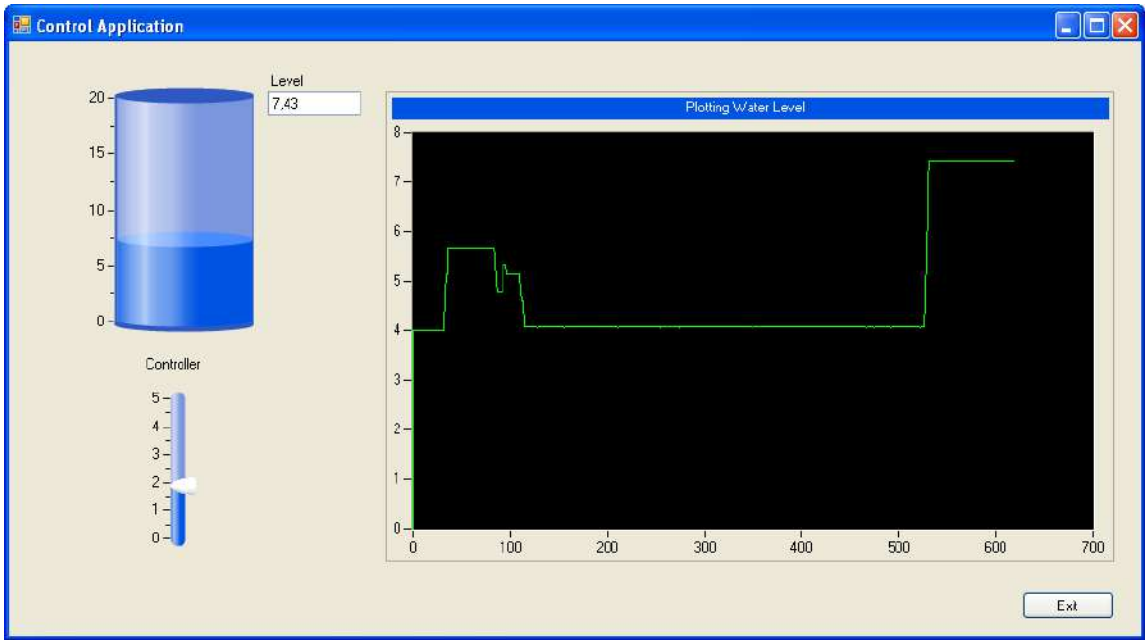
In the Timer Event we create the main logic in our application. We call the ReadDaqData() and WriteDaqData() methods, updates the GUI, etc.

```
private void timer1_Tick(object sender, EventArgs e)
       {

          DaqData myDaqData = new DaqData();

          //Read Data
          double analogDataIn;

          analogDataIn = myDaqData.ReadDaqData();

          if (analogDataIn < 0)
              analogDataIn = 0;
          if (analogDataIn > 5)
              analogDataIn = 5;

          //Scaling:
          analogDataIn = analogDataIn * 4; //0-5V -> 0-20cm


          tank.Value = analogDataIn;

          txtLevelValue.Text = analogDataIn.ToString("0.00");


          //Write Data

          double analogDataOut;

          analogDataOut = sliderControl.Value;

          myDaqData.WriteDaqData(analogDataOut);

       }
```

7 Trending Data

Now we want to extend our application with functionality for viewing historical data using a trend plot.

Below we see the new user interface:



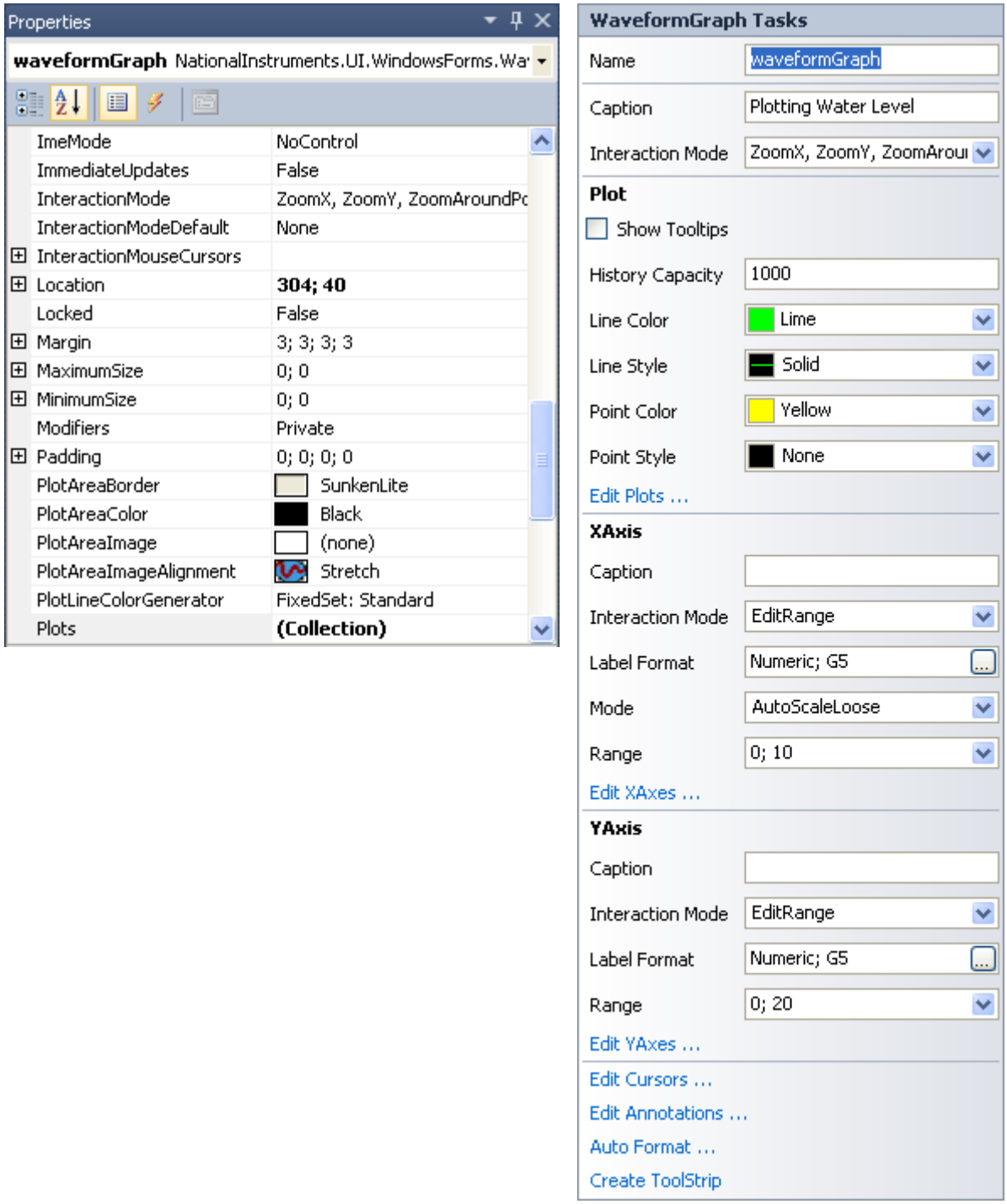
In this example we will use the “WaveformGraph” control in Measurement Studio.

The source code is the same as in the previous example, except for one new line of code in the Timer Event:

```
waveformGraph.PlotYAppend(analogDataIn);
```

The “WaveformGraph” control has lots of functionality you can set in the Properties window or clicking the Smart Tag Anchor (little arrow in the upper right corner of the control).

Below we see the Properties window (left side) and the Smart Tag Panel (right side) for the WaveformGraph control:


Properties
waveformGraph NationalInstruments.UI.WindowsForms.Wa
ImeMode NoControl
ImmediateUpdates False
InteractionMode ZoomX, ZoomY, ZoomAroundPc
InteractionModeDefault None
InteractionMouseCursors
Location 304; 40
Locked False
Margin 3; 3; 3; 3
MaximumSize 0; 0
MinimumSize 0; 0
Modifiers Private
Padding 0; 0; 0; 0
PlotAreaBorder SunkenLite
PlotAreaColor Black
PlotAreaImage (none)
PlotAreaImageAlignment Stretch
PlotLineColorGenerator FixedSet: Standard
Plots (Collection)
WaveformGraph Tasks
Name waveformGraph
Caption Plotting Water Level
Interaction Mode ZoomX, ZoomY, ZoomAroui
Plot
Show Tooltips
History Capacity 1000
Line Color Lime
Line Style Solid
Point Color Yellow
Point Style None
Edit Plots ...
XAxis
Caption
Interaction Mode EditRange
Label Format Numeric; G5
Mode AutoScaleLoose
Range 0; 10
Edit XAxes ...
YAxis
Caption
Interaction Mode EditRange
Label Format Numeric; G5
Range 0; 20
Edit YAxes ...
Edit Cursors ...
Edit Annotations ...
Auto Format ...
Create ToolStrip

8 Discretization

The next improvements to our application would be to implement a Low-pass Filter in order to remove the noise from the signal when reading the level. Another improvement would be to replace the manual control with a PI controller that do the job for us. Finally it would be nice to have a mathematical model of our water tank so we can simulate and test the behavior of the real system without connect to it.

So we need to create discrete versions of the low-pass filter, the PI controller and the process model. We can, e.g., use the Euler Forward discretization method:

$$\dot{x} \approx \frac{x_{k+1} - x_k}{T_s}$$

or the Euler Backward discretization method:

$$\dot{x} \approx \frac{x_k - x_{k-1}}{T_s}$$

T_s is the Sampling Time.

8.1 Low-pass Filter

The transfer function for a first-order low-pass filter may be written:

$$H(s) = \frac{y(s)}{u(s)} = \frac{1}{T_f s + 1}$$

Where T_f is the time-constant of the filter, $u(s)$ is the filter input and $y(s)$ is the filter output.

Discrete version:

It can be shown that a discrete version can be stated as:

$$\boldsymbol{y_k = (1 - a)y_{k-1} + au_k}$$

Where

$$\boldsymbol{a = \frac{T_s}{T_f + T_s}}$$

Where T_s is the Sampling Time.

It is a golden rule that $T_s \ll T_f$ and in practice we should use the following rule:

$$T_s \leq \frac{T_f}{5}$$

Proof:

Given:

$$\frac{y}{u} = \frac{1}{T_f s + 1}$$

This gives:

$$(T_f s + 1)y = u$$

$$T_f s y + y = u$$

Inverse Laplace gives:

$$T_f \dot{y} + y = u$$

We use the Euler Backward discretization method, $\dot{x} \approx \frac{x_k - x_{k-1}}{T_s}$, which gives:

$$T_f \frac{y_k - y_{k-1}}{T_s} + y_k = u_k$$

Then we get:

$$T_f(y_k - y_{k-1}) + y_k T_s = u_k T_s$$

$$T_f y_k - T_f y_{k-1} + y_k T_s = u_k T_s$$

$$y_k(T_f + T_s) = T_f y_{k-1} + u_k T_s$$

This gives:

$$y_k = \frac{T_f}{T_f + T_s} y_{k-1} + \frac{T_s}{T_f + T_s} u_k$$

For simplicity we set:

$$\frac{T_s}{T_f + T_s} \equiv a$$

This gives:

$$y_k = (1 - a)y_{k-1} + a u_k$$

$$a = \frac{T_s}{T_f + T_s}$$

[End of Proof]

8.2 PI Controller

A PI controller may be written:

$$u(t) = u_0 + K_p e(t) + \frac{K_p}{T_i}\int_0^t e d\tau$$

Where u is the controller output and e is the control error:

$$e(t) = r(t) - y(t)$$

Laplace:

$$u(s) = K_p e(s) + \frac{K_p}{T_i s} e(s)$$

Discrete version:

We start with:

$$u(t) = u_0 + K_p e(t) + \frac{K_p}{T_i}\int_0^t e d\tau$$

In order to make a discrete version using, e.g., Euler, we can derive both sides of the equation:

$$\dot{u} = \dot{u}_0 + K_p \dot{e} + \frac{K_p}{T_i} e$$

If we use Euler Forward we get:

$$\frac{u_k - u_{k-1}}{T_s} = \frac{u_{0,k} - u_{0,k-1}}{T_s} + K_p \frac{e_k - e_{k-1}}{T_s} + \frac{K_p}{T_i} e_k$$

Then we get:

$$\boldsymbol{u_k = u_{k-1} + u_{0,k} - u_{0,k-1} + K_p(e_k - e_{k-1}) + \frac{K_p}{T_i} T_s e_k}$$

Where

$$e_k = r_k - y_k$$

We can also split the equation above in 2 different pars by setting:

$$\Delta u_k = u_k - u_{k-1}$$

This gives the following PI control algorithm:

$$\boldsymbol{e_k = r_k - y_k}$$

$$\boldsymbol{\Delta u_k = u_{0,k} - u_{0,k-1} + K_p(e_k - e_{k-1}) + \frac{K_p}{T_i} T_s e_k}$$

$$\boldsymbol{u_k = u_{k-1} + \Delta u_k}$$

This algorithm can easily be implemented in C#.

8.2.1 PI Controller as a State-space model

Given:

$$u(s) = K_p e(s) + \frac{K_p}{T_i s} e(s)$$

We set $z = \frac{1}{s} e \Rightarrow sz = e \Rightarrow \dot{z} = e$

This gives:

$$\dot{z} = e$$

$$u = K_p e + \frac{K_p}{T_i} z$$

Where

$$e = r - y$$

Discrete version:

Using Euler:

$$\dot{z} \approx \frac{z_{k+1} - z_k}{T_s}$$

Where T_s is the Sampling Time.

This gives:

$$\frac{z_{k+1} - z_k}{T_s} = e_k$$

$$u_k = K_p e_k + \frac{K_p}{T_i} z_k$$

Finally:

$$e_k = r_k - y_k$$

$$u_k = K_p e_k + \frac{K_p}{T_i} z_k$$

$$z_{k+1} = z_k + T_s e_k$$

This algorithm can easily be implemented in C#.

8.3 Process Model

The level tank:

A very simple (linear) model of the water tank is as follows:

$$A_t \dot{h} = K_p u - F_{out}$$

or

$$\dot{h} = \frac{1}{A_t}\left[K_p u - F_{out}\right]$$

Where:

- h [cm] is the level in the water tank
- u [V] is the pump control signal to the pump
- A_t [cm2] is the cross-sectional area in the tank
- K_p [(cm3/s)/V] is the pump gain
- F_{out} [cm3/s] is the outflow through the valve (this outflow can be modeled more accurately taking into account the valve characteristic expressing the relation between pressure drop across the valve and the flow through the valve).

We can use the Euler Forward discretization method in order to create a discrete model:

$$\dot{x} \approx \frac{x_{k+1} - x_k}{T_s}$$

Then we get:

$$\frac{h_{k+1} - h_k}{T_s} = \frac{1}{A_t}\left[K_p u_k - F_{out}\right]$$

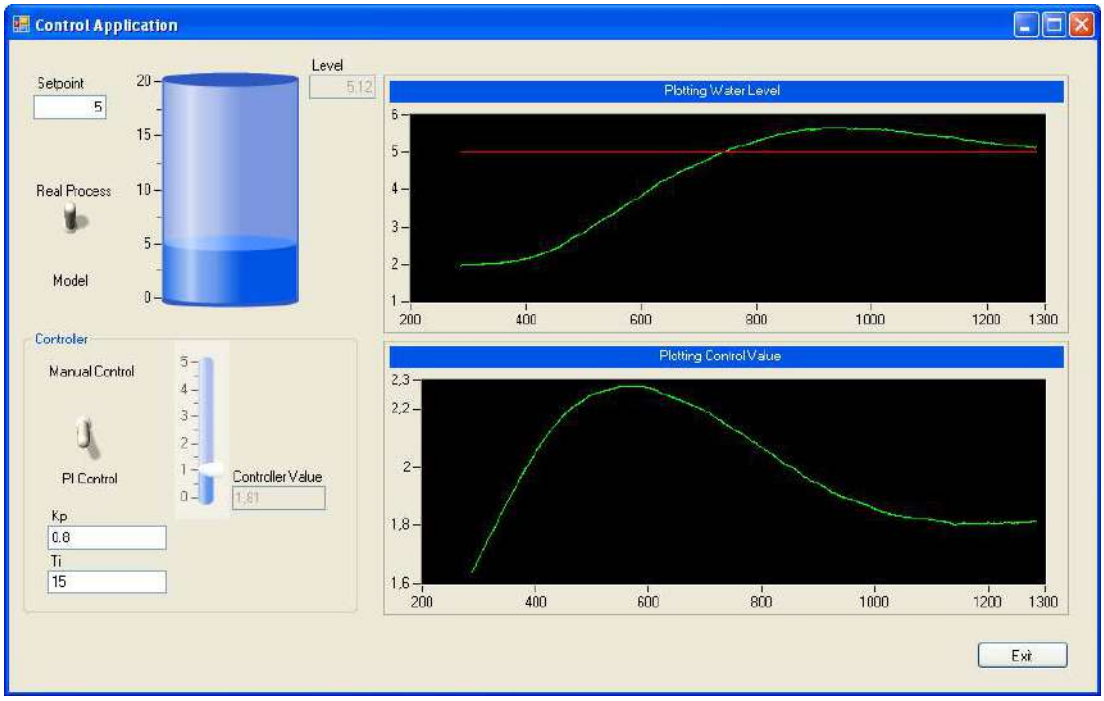
Finally:

$$\boldsymbol{h_{k+1} = h_k + \frac{T_s}{A_t}\left[K_p u_k - F_{out}\right]}$$

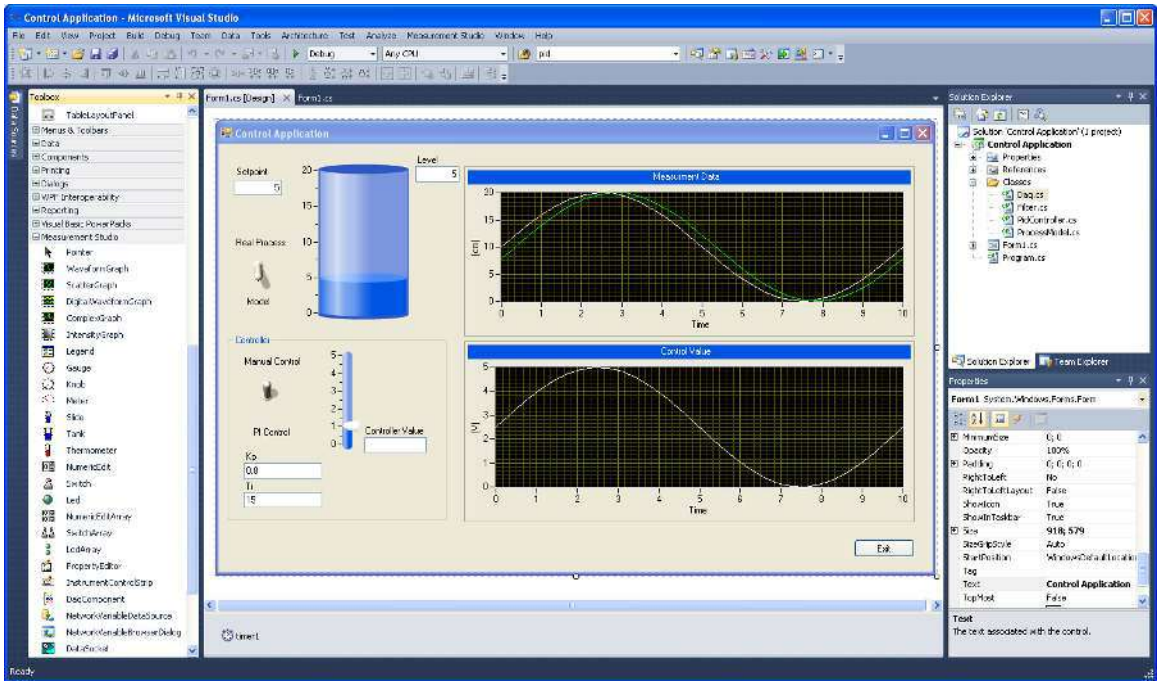
8.4 Final Application

We extend our Control Application with a discrete PI controller, a discrete Low-pass filter and a discrete process model, so we can switch between the real process and a simulator.

Our User Interface is as follows:



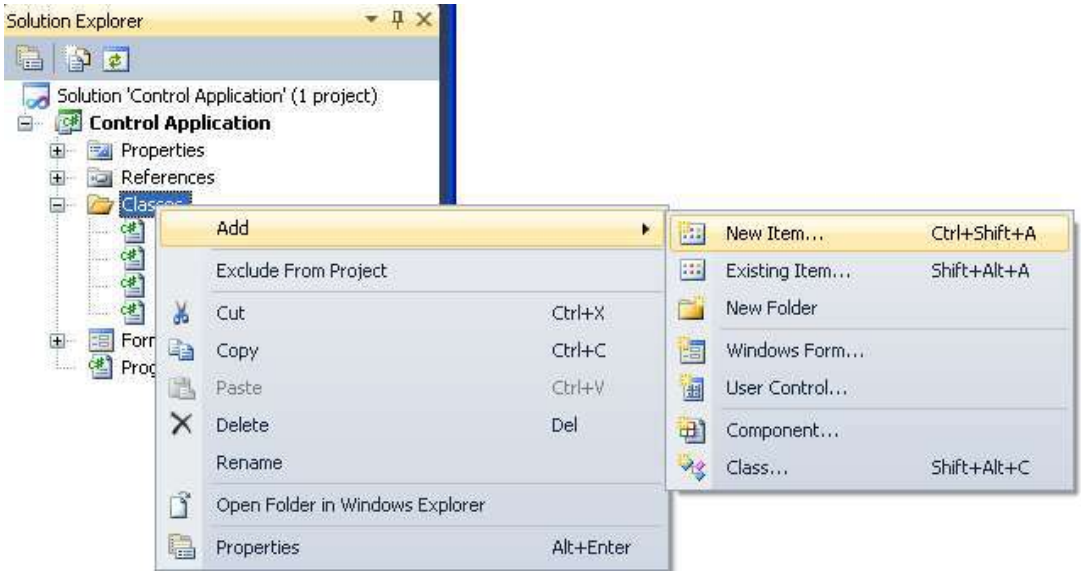
Below we see the Project and Solution in Visual Studio:



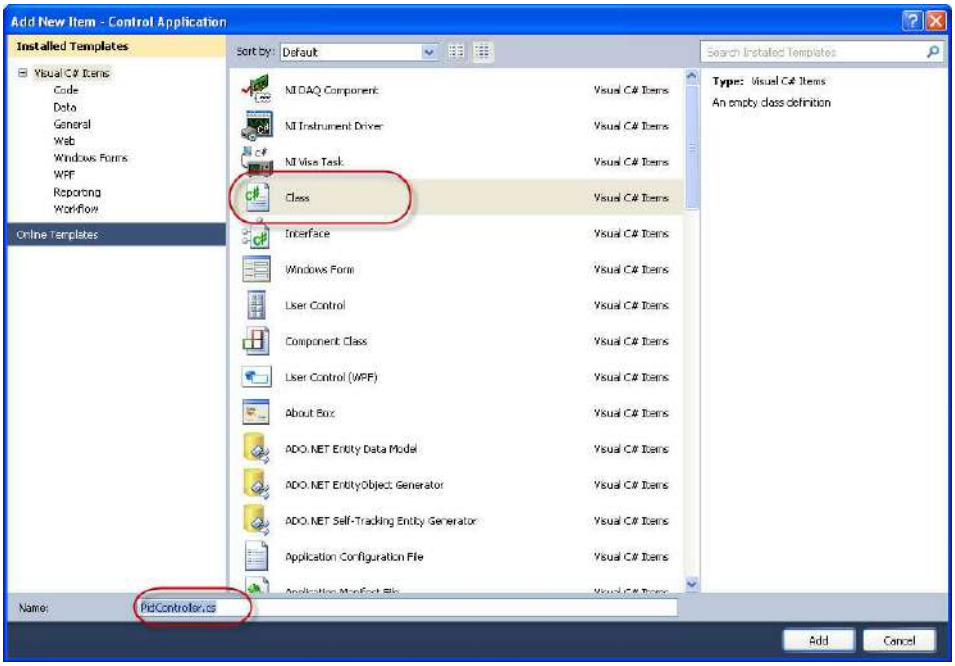
Below we will show and describe the important parts of the code.

PI Controller:

We create a new Class for our PID algorithm, by right-click in the Solution Explorer (Add→New Item…)



The Add New Item window appears:



Select the Class Item and a proper Name, e.g. “PidController”.

The **PidController** Class is as follows:

```
class PidController
{

        public double r; //Reference Value
       public double Kp; //Proportional Gain for PID Controler
       public double Ti; //Integral Time for PID Controler
       public double Ts; //Sampling Time

       private double z; //Internal variable


       public double PiController(double y)
       {
          double e; // Error between Reference and Measurement
```

```
            double u; // Controller Output

            //PID Algoritm
            e = r - y;
            u = Kp * e + (Kp / Ti) * z;
            z = z + Ts * e;

            return u;
        }

}
```

We then initialize the PidController Class:

```
PidController pidControl = new PidController
        {
        Ts=0.1,
             r=5,
             Kp=0.8,
             Ti=15
        };
```

Finally we use the controller:

```
private void ControlSystem()
{

        //Write Control Value

       if (switchController.Value == true) //Use Manual Control
       {

              controllerOutput = sliderControl.Value;

       }
       else // Use PID Control
       {

        controllerOutput = pidControl.PiController(levelMeasurement);

        //Scaling
        controllerOutput = controllerOutput / 4; //0-20cm -> 0-5V

        //Set boundaries
        if (controllerOutput < 0)
                      controllerOutput = 0;
        if (controllerOutput > 5)
                      controllerOutput = 5;
        }

       myDaqData.WriteDaqData(controllerOutput); //Write to DAQ

}
```

Low-pass Filter:

We create the Low-pass Filter as a separate Class to:

```
class Filter
{

       public double yk;
       public double Ts;
       public double Tf;


       public double LowPassFilter(double yFromDaq)
       {
```

```
            double a;
            double yFiltered;

            a = Ts / (Ts + Tf);


            yFiltered = (1 - a) * yk + a * yFromDaq;
            yk = yFiltered;

            return yFiltered;

        }
}
```

We then initialize the filter:

```
Filter filter = new Filter
        {
        Ts = 0.1,
             Tf = 2
       };
```

Finally we use the filter:

```
// Lowpass filtering the Measure Value due to noise
levelMeasurement = filter.LowPassFilter(levelMeasurement);
```

Discrete Model:

We do the same for the discrete model.

We have created a Class and a LevelTankModel Method that we use in our simulation:

```
levelMeasurement = model.LevelTankModel(controllerOutput);
```

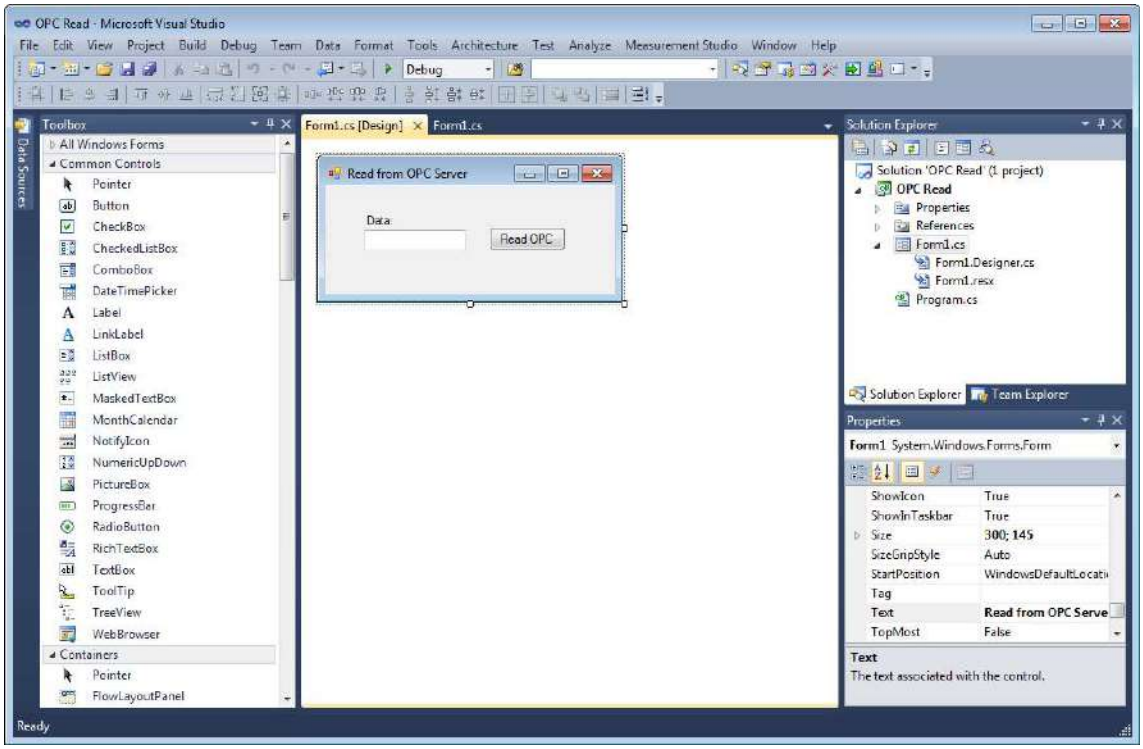
9 OPC

In order to communicate with an OPC Server we can use the DataSocket API that is part of the Measurement Studio. We use the Matrikon OPC Simulation Server.

9.1 Read OPC Data

Below we will go through a very simple example. We will read one value from the OPC Server each time we click a button.

Visual Studio Project:



Code:

We define a DataSocket object:

```
DataSocket dataSocket = new DataSocket();
```

Next, We Connect to the OPC Server:

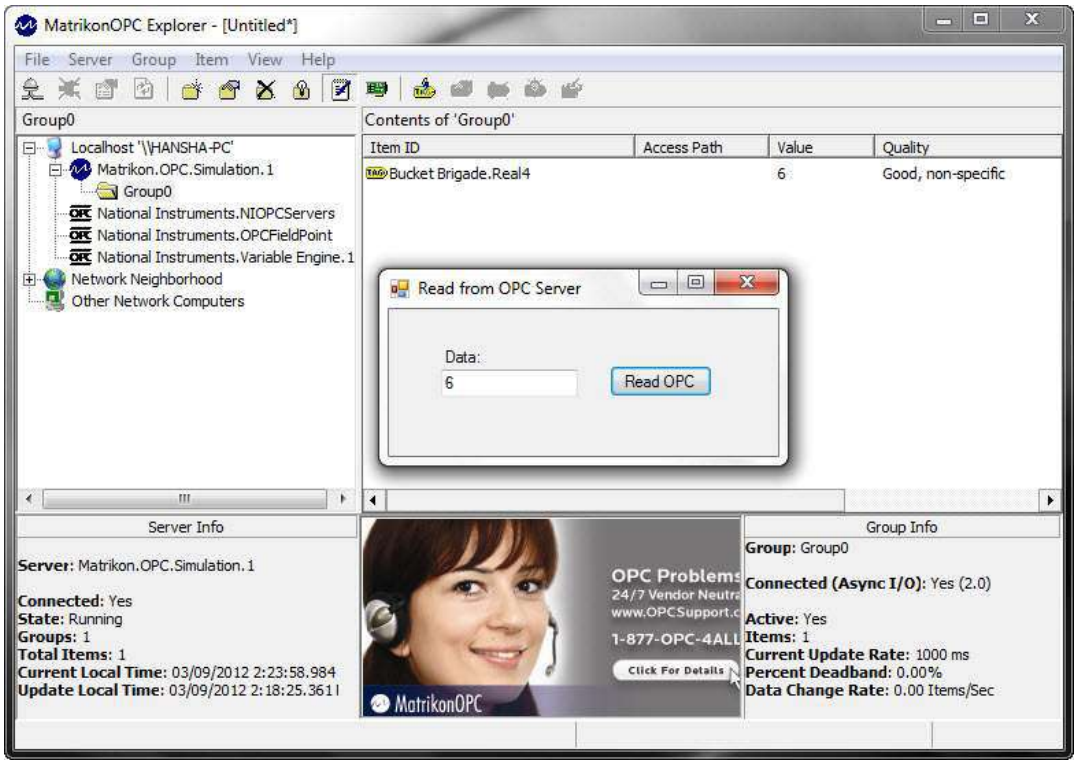
```
string opcUrl;
opcUrl = "opc://localhost/MATRIKON.OPC.Simulation/Bucket Brigade.Real4";
```

```
if (dataSocket.IsConnected)
    dataSocket.Disconnect();

dataSocket.Connect(opcUrl, AccessMode.Read);
```

Finally, we Read OPC Data:

```
private void btnReadOpc_Click(object sender, EventArgs e)
{
    dataSocket.Update();

    txtReadOpcValue.Text = dataSocket.Data.Value.ToString();
}
```

We test the Application using the Matrikon OPC Explorer:

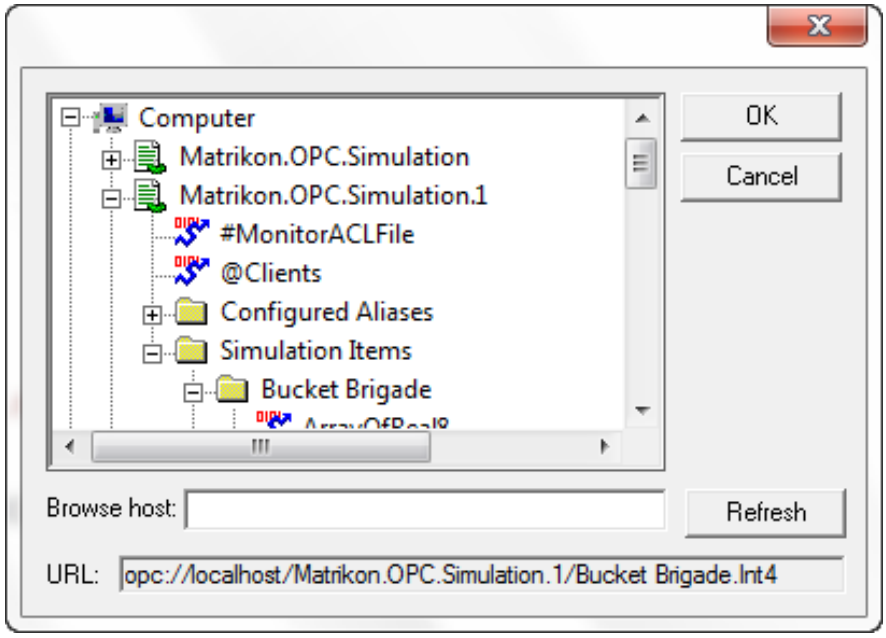


SelectUrl:

We can use the **SelectUrl** method if we want to pick the OPC item from a list of available servers (both local servers and network servers) and items.

```
dataSocket.SelectUrl();
```

The **SelctUrl** method will pop up the following window:

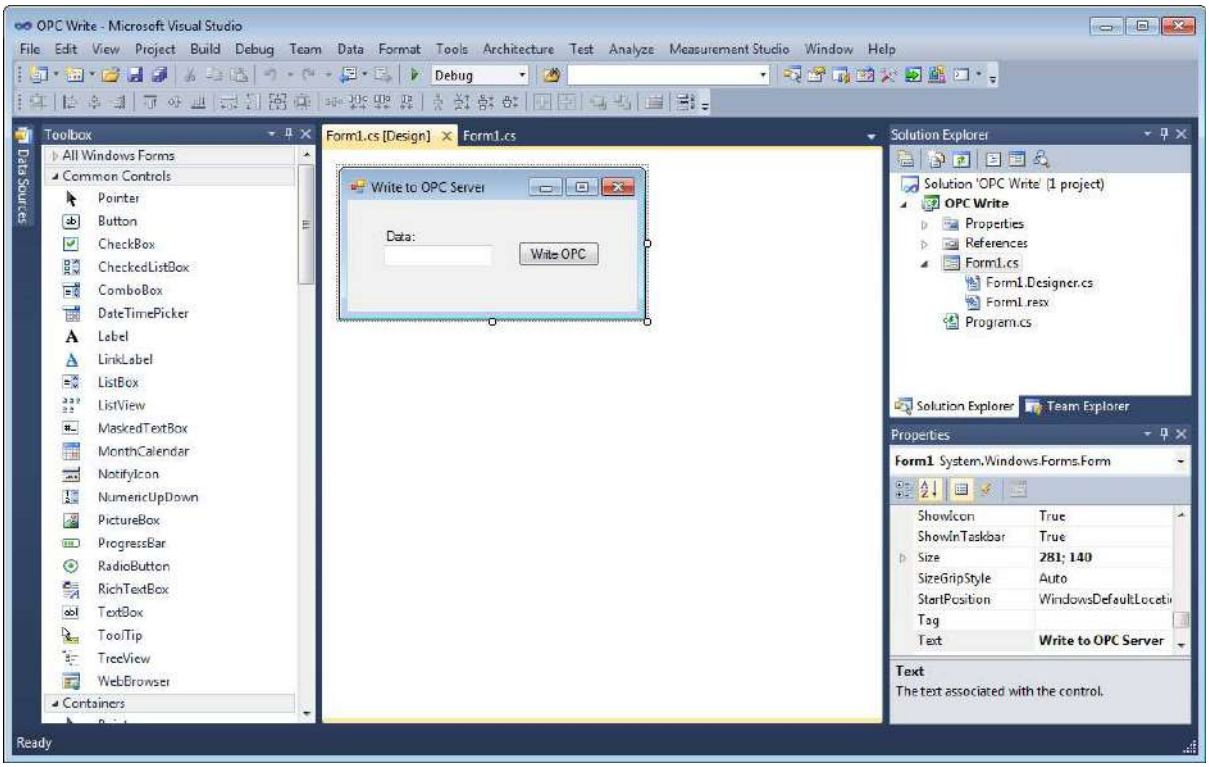


9.2 Write OPC Data

We use the same DataSocket API here.

Visual Studio Project:

Below we will go through a very simple example. We will write one value to the OPC Server each time we click a button.



Code:

We define a DataSocket object:

```
DataSocket dataSocket = new DataSocket();
```

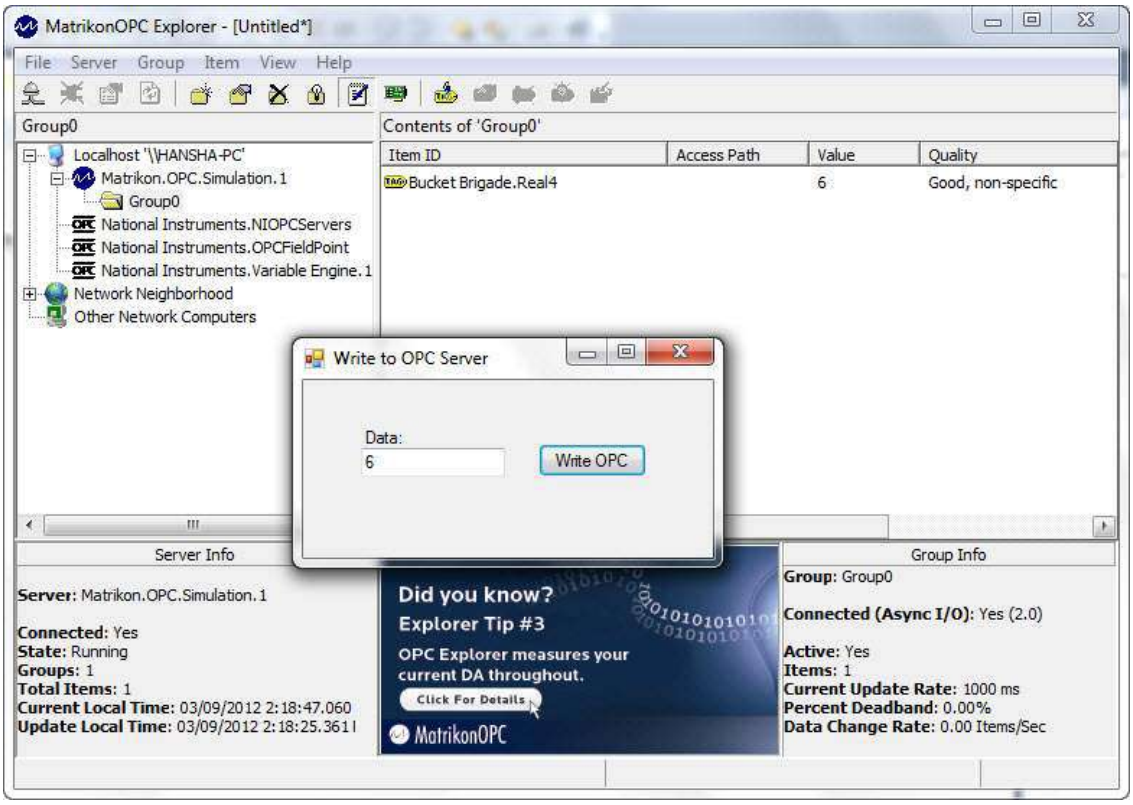
Next, We Connect to the OPC Server:

```
string opcUrl;
opcUrl = "opc://localhost/MATRIKON.OPC.Simulation/Bucket Brigade.Real4";

if (dataSocket.IsConnected)
   dataSocket.Disconnect();

dataSocket.Connect(opcUrl, AccessMode.Write);
```

Finally, we Write OPC Data:

```
private void btnWriteOpc_Click(object sender, EventArgs e)
{

     double opcValue = 0;

     opcValue = Convert.ToDouble(txtWriteOpcValue.Text);

     dataSocket.Data.Value = opcValue;

     dataSocket.Update();
}
```

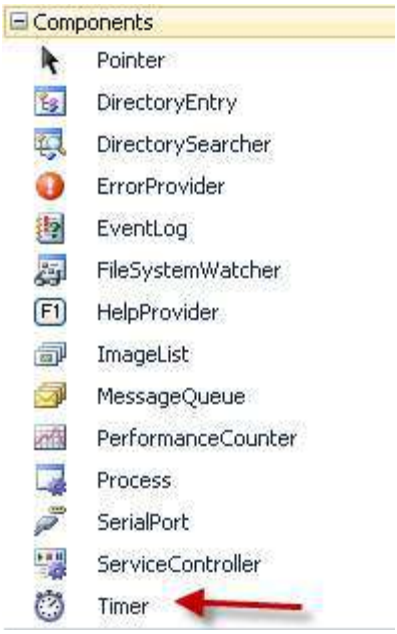
We test the Application using the Matrikon OPC Explorer:



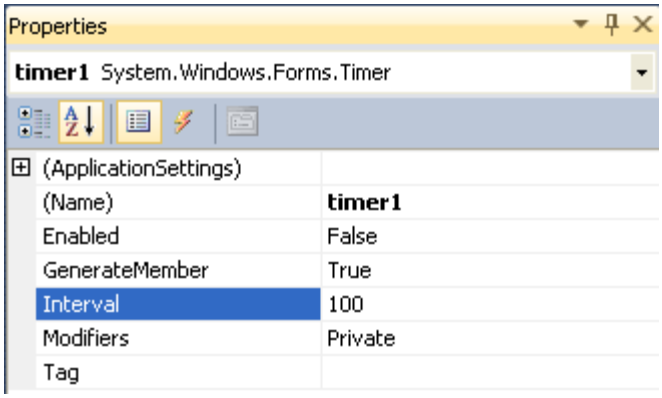
9.3 Using a Timer

We can use a timer in order to read values “continuously”, i.e. at specific intervals.

In the “Components” toolbox we find the “Timer” Control:



In the Properties window we can specify the Interval (“Sampling Time”) in milliseconds.



We can start the timer with the following code:

```
public Form1()
       {
           InitializeComponent();

           timer1.Start();
       }
```

In the Timer Event we create the code in order to read data at this specific interval.

```
private void timer1_Tick(object sender, EventArgs e)
       {

           …
           …

       }
```

10 Using Measurement Studio Templates

In order to use the NI USB-TC01 Thermocouple Measurement device with C# we need to have the **DAQmx driver** and the DAQmx API for C# installed. In order to install the DAQmx API for C#, make sure to select ".NET Support" when installing the DAQmx driver.

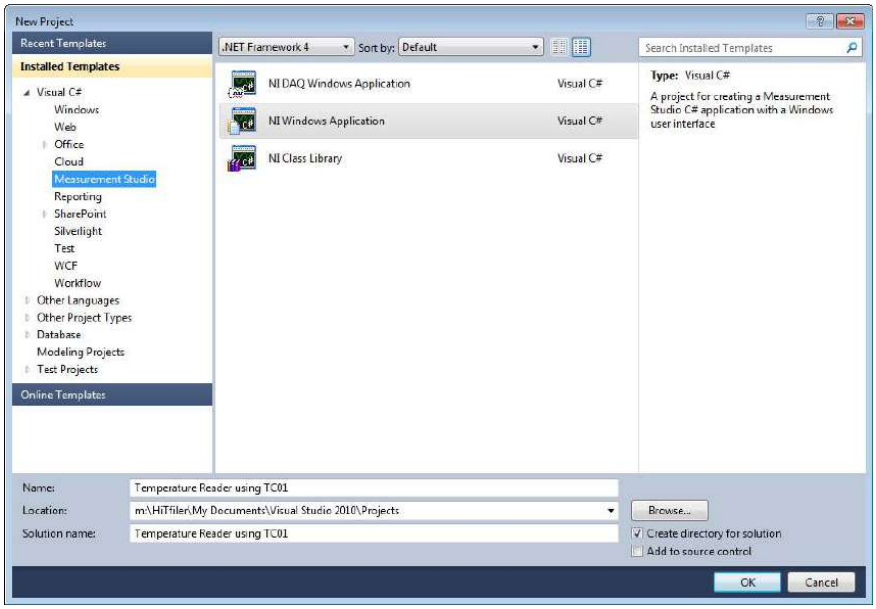
C# is a powerful programming language, but has few built-in features for measurement and control applications. **Measurement Studio** is an add-on to Visual Studio which makes it easier to create such applications. With Measurement Studio we can implement Data Acquisition and a graphical HMI.

You don't need to use the Measurement Studio to create an application where you use the NI USB-TC01 Thermocouple Measurement device, but it is easier.

Here we will use Visual Studio and the Measurement Studio Add-in to create some DAQ examples where we get temperature data from the NI USB-TC01 Thermocouple Measurement device.

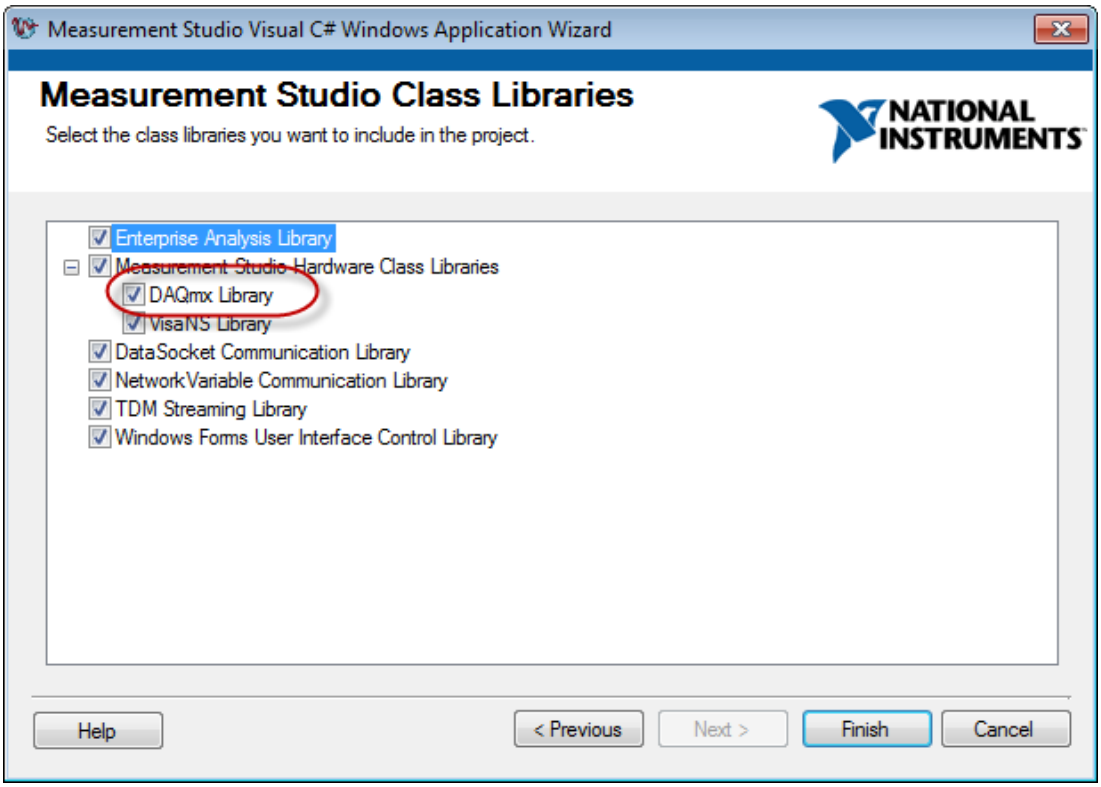
10.1 Create a NI Windows Application

In this example, we will select the "NI windows Application" Template in the "New Project" window.



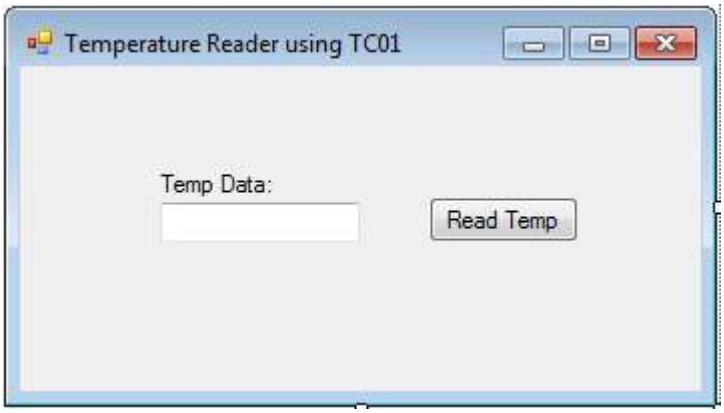
Note! The “New Project” window may look different on your computer, it depends on what features you have installed and which version or edition of Measurement Studio you are using.

Next we select the Class Libraries we want to include. In our case we need at least to select the “DAQmx Library”.



When we click “Finish” an empty project will be created for us. We are now ready to create our own application.

We start by creating a simple User Interface:

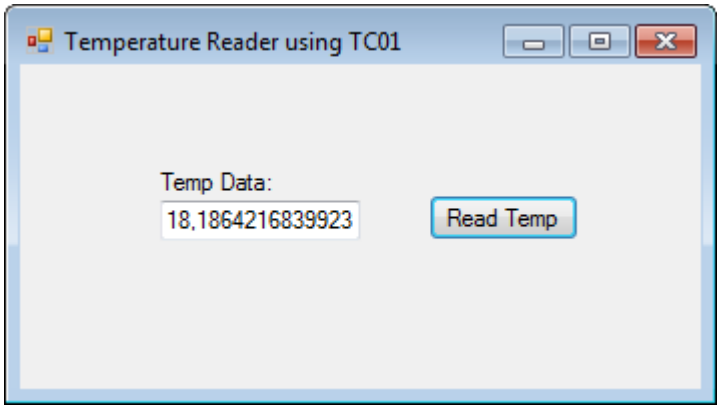


When we click the “Read Temp” button, the Temperature data shall be shown in the “Temp Data” TextBox.

In the event Handler for the “Read Temp” button, we create the following code:

```
private void btnReadTempData_Click(object sender, EventArgs e)
        {

            Task analogInTask = new Task();

            AIChannel myAIChannel;

            myAIChannel = analogInTask.AIChannels.CreateThermocoupleChannel(
                "Dev1/ai0",
                "Temperature",
                0,
                100,
                AIThermocoupleType.J,
                AITemperatureUnits.DegreesC,
                25
                );


            AnalogSingleChannelReader reader = new
               AnalogSingleChannelReader(analogInTask.Stream);

            double analogDataIn = reader.ReadSingleSample();

            txtTempData.Text = analogDataIn.ToString();

        }
```

Then we can test our application:

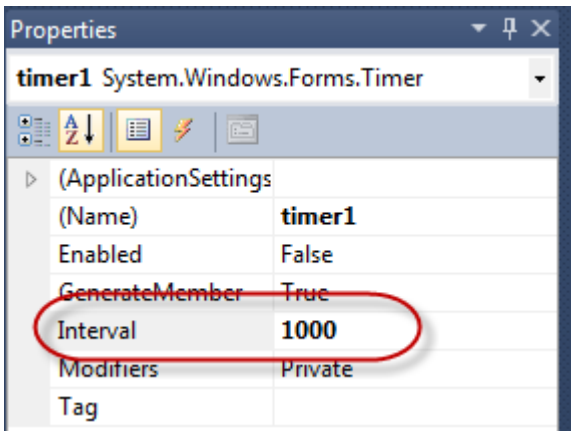


Improvements: We should use a **"Timer"** in order to read Temperature data at specific intervals instead of pushing a button each time we need data.

We drag in a Timer component from the Toolbox. First, we need to start the Timer:

```
public Form1()
       {
           InitializeComponent();

           timer1.Start();
       }
```

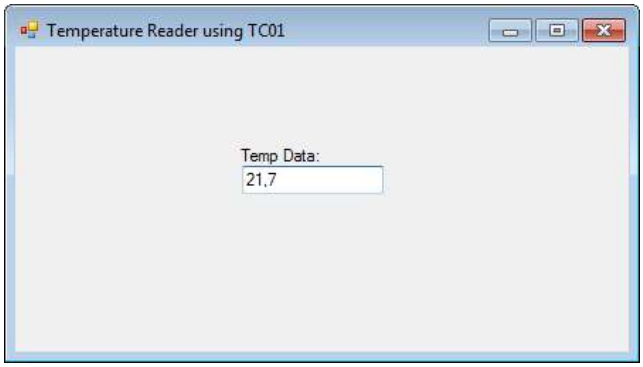
Next, we need to specify the interval. We can do that in the **Properties** window:



In the Timer Event we write the code for reading the Temperature:

```
private void timer1_Tick(object sender, EventArgs e)
       {

           Task analogInTask = new Task();


           AIChannel myAIChannel;


           myAIChannel = analogInTask.AIChannels.CreateThermocoupleChannel(
               "Dev1/ai0",
               "Temperature",
               0,
               100,
               AIThermocoupleType.J,
               AITemperatureUnits.DegreesC,
               25
               );


           AnalogSingleChannelReader reader = new
              AnalogSingleChannelReader(analogInTask.Stream);

           double analogDataIn = reader.ReadSingleSample();


           txtTempData.Text = analogDataIn.ToString("0.0");

       }
```

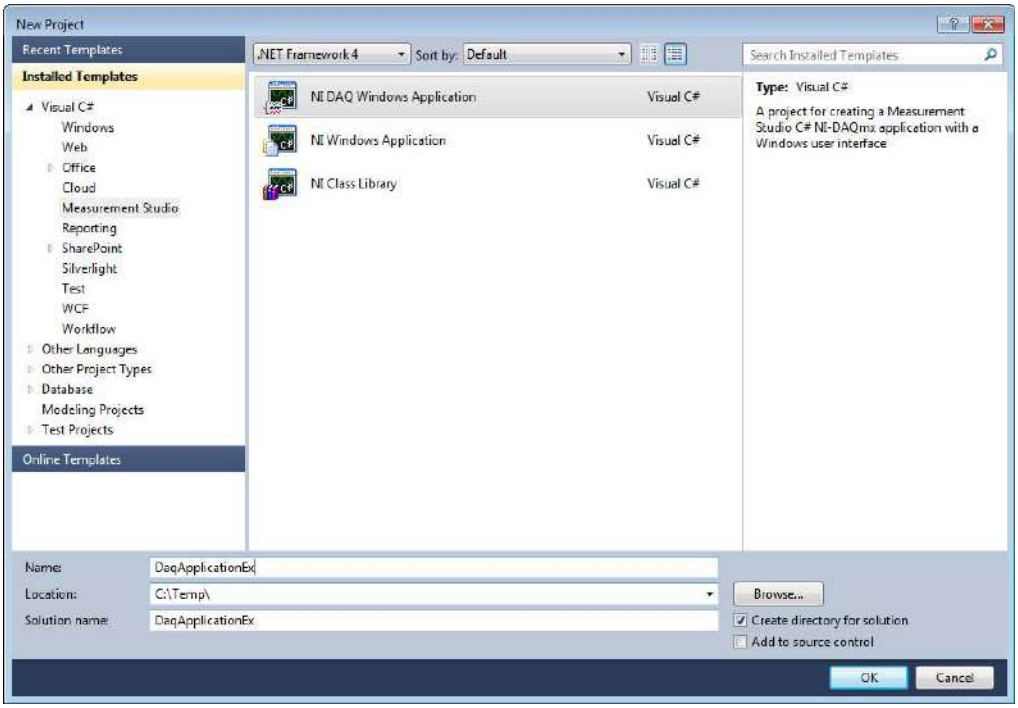
Finally, we test the application:



10.2 Create a NI DAQ Windows Application

Note! This option is only availible if you to have either the Professional or Enterprise version of the Measurement Studio software.

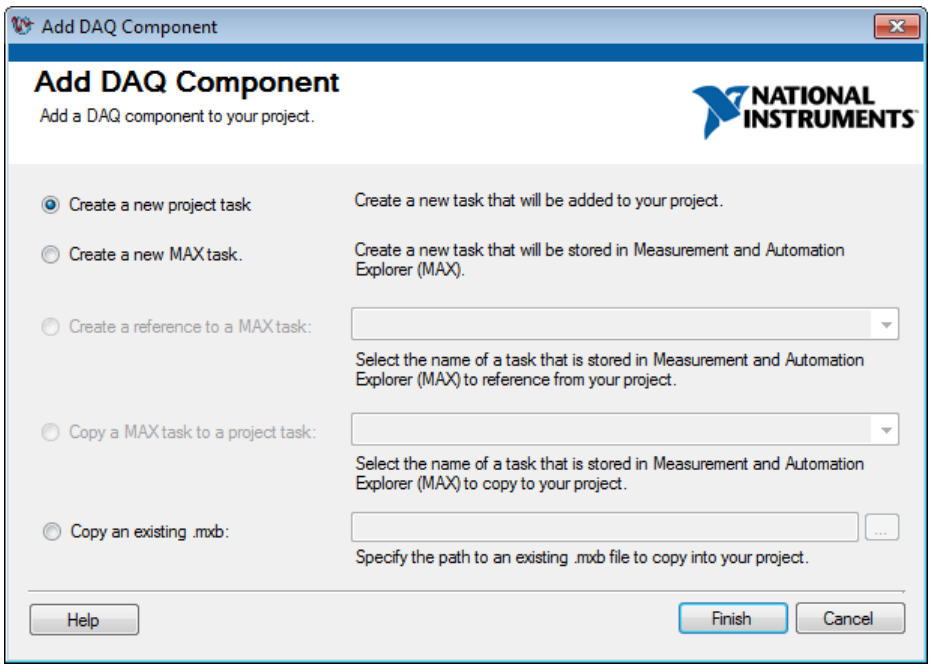
We start by creating a “New Project” from Visual Studio. In the “New Project” window we select “Visual C#” » “Measurement Studio”, and then select the “NI DAQ Windows Application” Template that is part of the Measurement Studio.



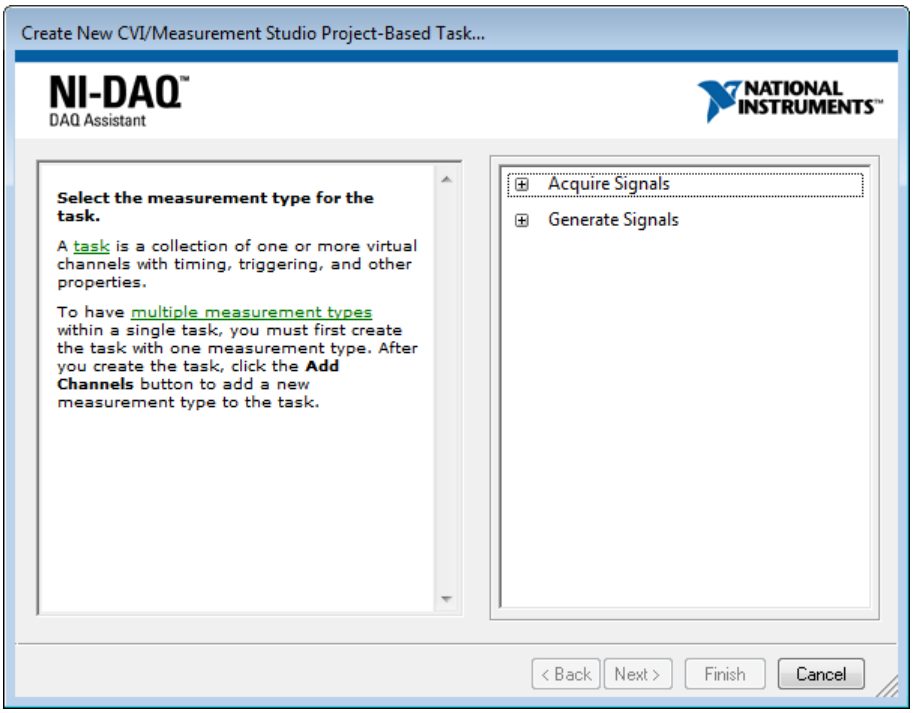
Note! The “New Project” window may look different on your computer, it depends on what features you have installed and which version or edition of Measurement Studio you are using.

Next, we need to go through different steps in a wizard.

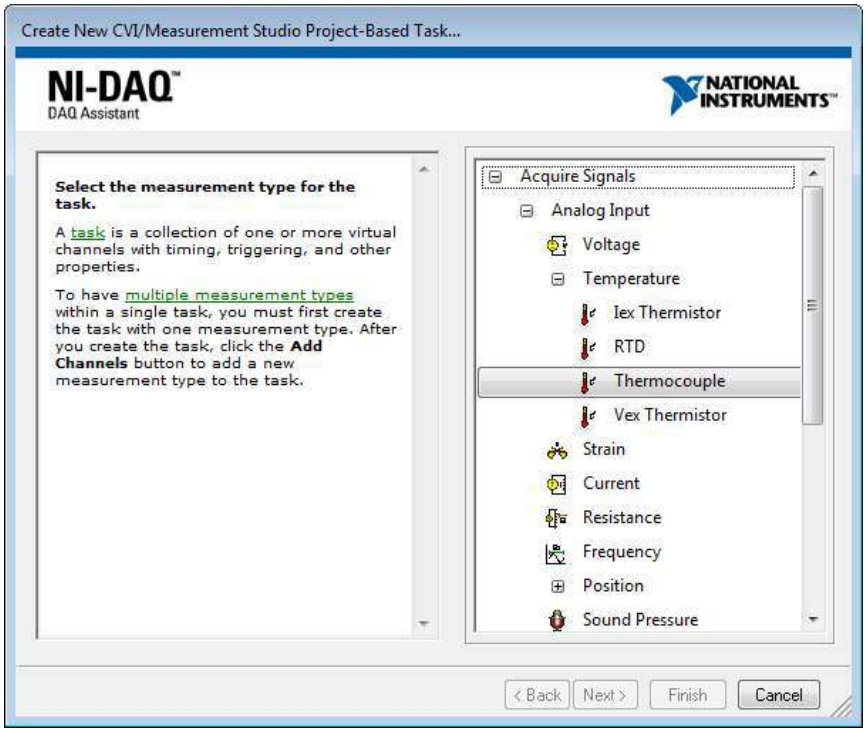
In the first step we select “Create a new project task”:



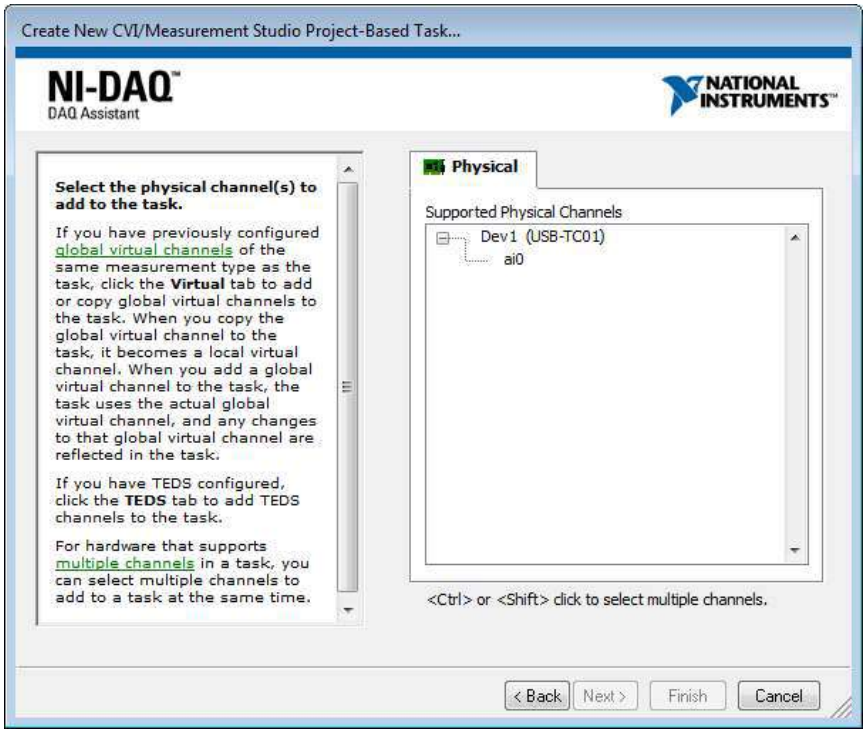
In the next step we select the measurement type, and since the NI USB-TC01 Thermocouple Measurement device is for reading Temperature values, we need to select “Acquire Signals”.



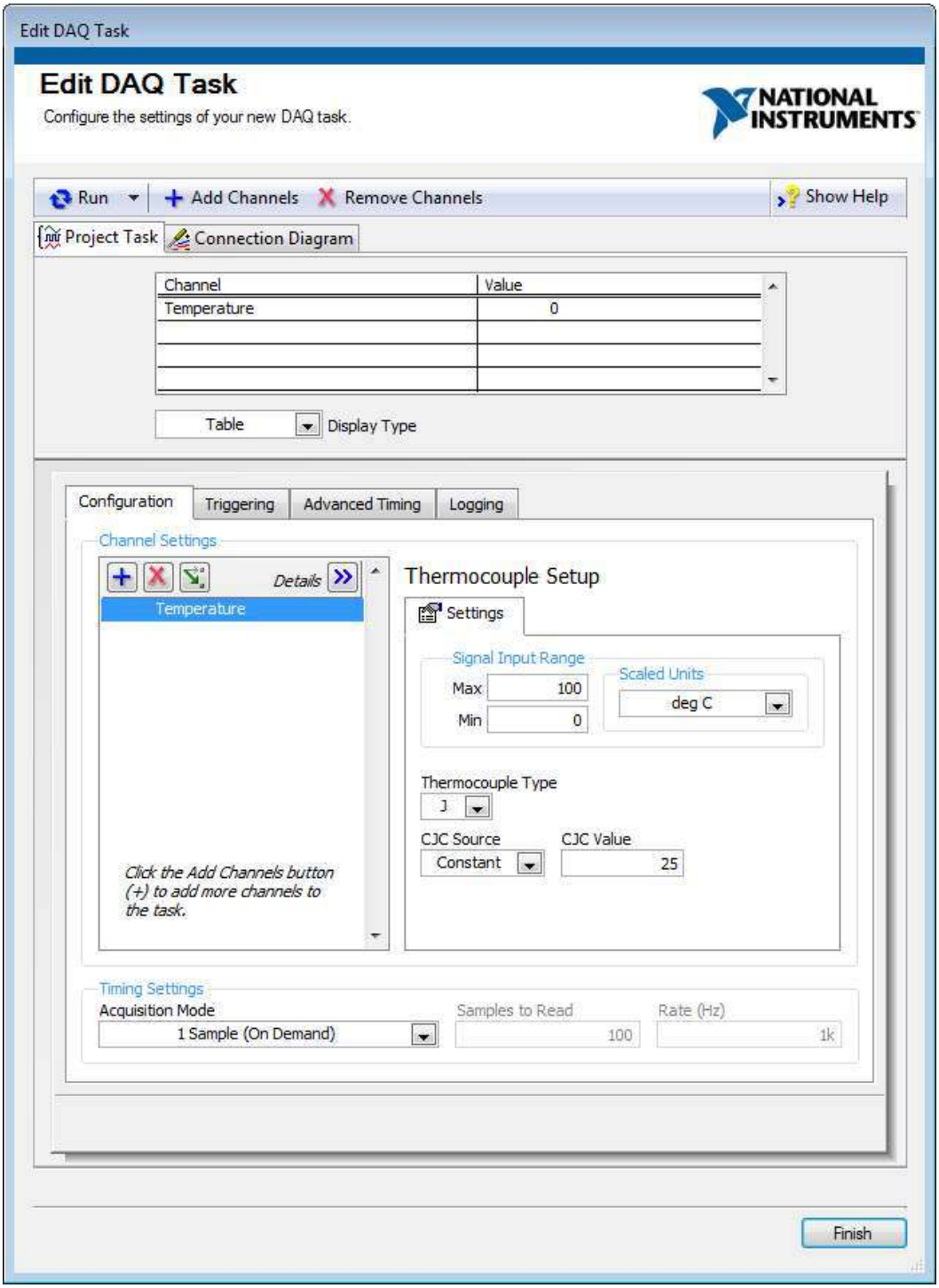
In the “Acquire Signals” node we first select “Temperature” and then select “Thermocouple”.



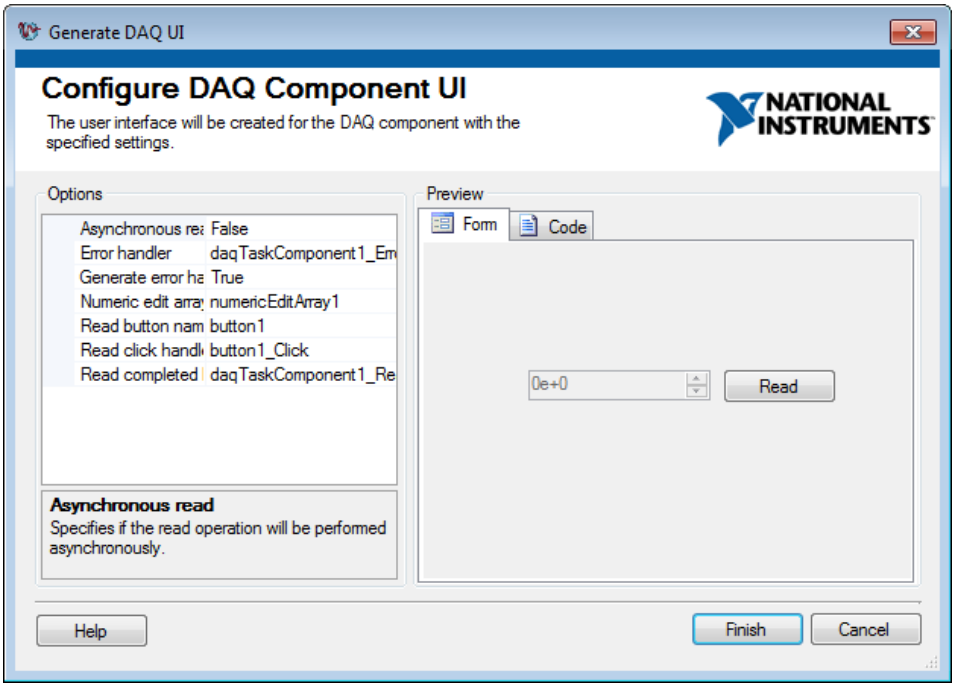
Next we need to select the physical channel(s). Since the NI USB-TC01 Thermocouple Measurement device has only one channel available, we need to select “ai0” (Analog In, Channel 0).



Next we can set different Properties, such as “Input Range”, “Acquisition Mode”, “Thermocouple Type”, etc.



Finally we see a Preview of the User Interface before it is automatically generated by the Measurement Studio:



When we click “Finish”, the Solution, the Project and User Interface will be automatically created.

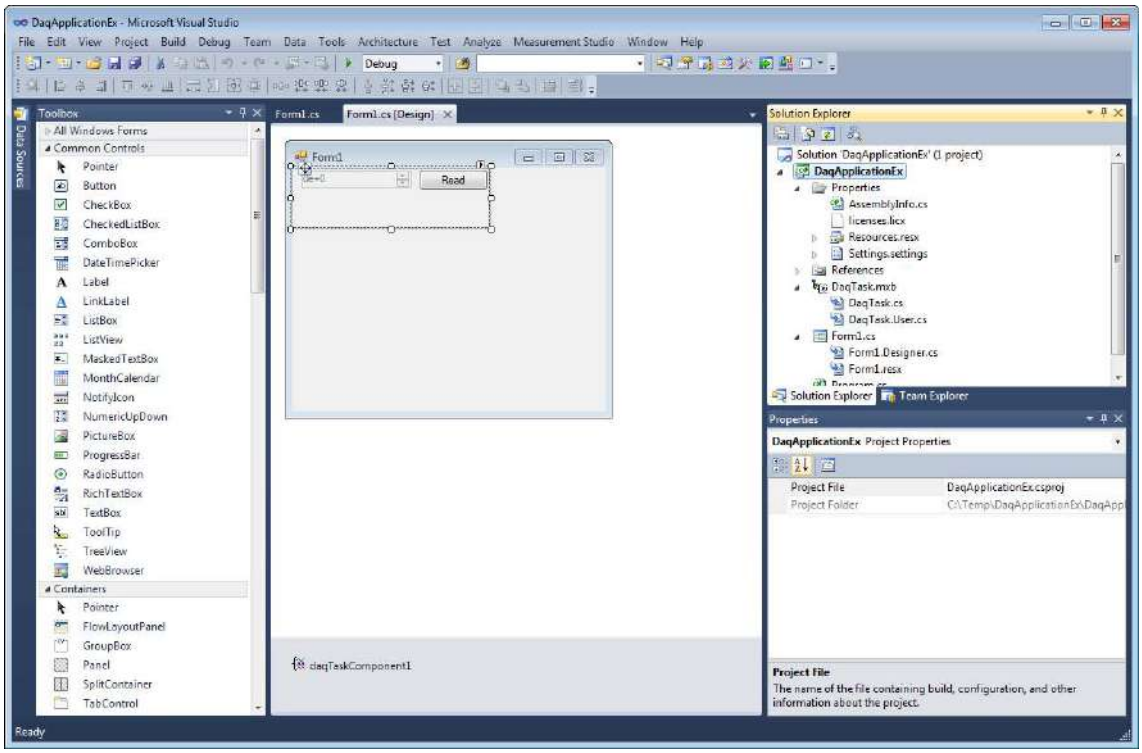
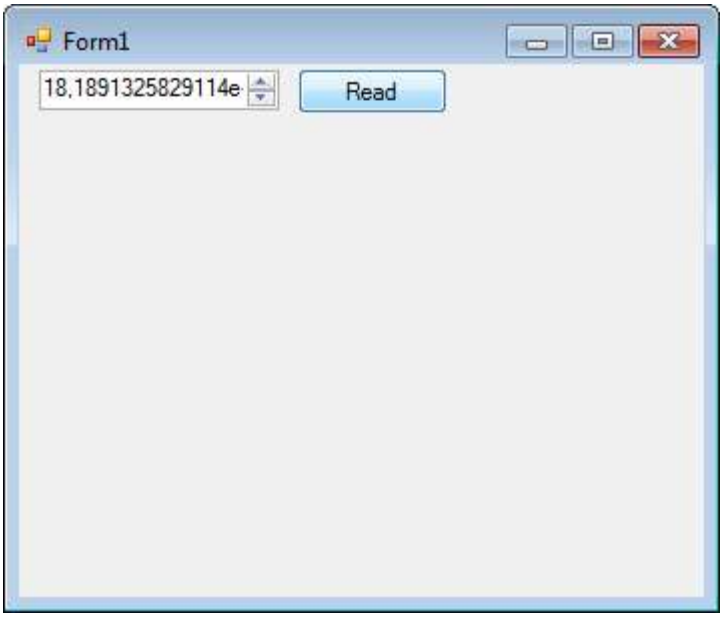


Figure 1: Visual Studio Generated Solution

Finally we can test the Application:



With this approach we get a “flying start” and we can change the code as we please. The drawback with this approach is that the code that is automatically generated is a little bit “messy”.

So actually, this is not a method I will recommend to use.

Appendix A: Source Code

In this Appendix the complete source code for all the examples will be listed.

My First DAQ App

The code for this application is as follows:

```
using NationalInstruments;
using NationalInstruments.DAQmx;
using NationalInstruments.UI;
using NationalInstruments.UI.WindowsForms;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;


namespace MyFirstDAQApp
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }


        private void Form1_Load(object sender, EventArgs e)
        {

        }

        private void btnGetAnalogIn_Click(object sender, EventArgs e)
        {

            Task analogInTask = new Task();


            AIChannel myAIChannel;

            myAIChannel = analogInTask.AIChannels.CreateVoltageChannel(
                "dev1/ai0",
                "myAIChannel",
                AITerminalConfiguration.Differential,
                0,
                5,
                AIVoltageUnits.Volts
                );

            AnalogSingleChannelReader reader = new
             AnalogSingleChannelReader(analogInTask.Stream);

            double analogDataIn = reader.ReadSingleSample();

            txtAnalogIn.Text = analogDataIn.ToString();
```

```
        }

        private void btnWriteAnalogOut_Click(object sender, EventArgs e)
        {


            Task analogOutTask = new Task();

            AOChannel myAOChannel;

            myAOChannel = analogOutTask.AOChannels.CreateVoltageChannel(
                "dev1/ao0",
                "myAOChannel",
                0,
                5,
                AOVoltageUnits.Volts
                );

            AnalogSingleChannelWriter writer = new
             AnalogSingleChannelWriter(analogOutTask.Stream);

            double analogDataOut;

            analogDataOut = Convert.ToDouble(txtAnalogOut.Text);

            writer.WriteSingleSample(true, analogDataOut);



        }


    }
}
```

Control Application

The code for this application is as follows:

```
using NationalInstruments;
using NationalInstruments.DAQmx;
using NationalInstruments.UI;
using NationalInstruments.UI.WindowsForms;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace Control_Application
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();

            timer1.Start();
        }

        private void timer1_Tick(object sender, EventArgs e)
        {

            DaqData myDaqData = new DaqData();
```

```
            //Read Data
            double analogDataIn;

            analogDataIn = myDaqData.ReadDaqData();

            if (analogDataIn < 0)
                analogDataIn = 0;
            if (analogDataIn > 5)
                analogDataIn = 5;

            //Scaling:
            analogDataIn = analogDataIn * 4; //0-5V -> 0-20cm


            tank.Value = analogDataIn;

            txtLevelValue.Text = analogDataIn.ToString("0.00");


            //Write Data

            double analogDataOut;

            analogDataOut = sliderControl.Value;

            myDaqData.WriteDaqData(analogDataOut);

        }

        private void button1_Click(object sender, EventArgs e)
        {
            Application.Exit();
        }
    }


    /// <summary>
    /// Reading and Writing Data from DAQ Device
    /// </summary>
    public class DaqData
    {

        public double ReadDaqData()
        {

            Task analogInTask = new Task();

            AIChannel myAIChannel;

            myAIChannel = analogInTask.AIChannels.CreateVoltageChannel(
                "dev1/ai0",
                "myAIChannel",
                AITerminalConfiguration.Differential,
                0,
                5,
                AIVoltageUnits.Volts
                );

            AnalogSingleChannelReader reader = new
             AnalogSingleChannelReader(analogInTask.Stream);

            double analogDataIn = reader.ReadSingleSample();

            return analogDataIn;
        }


        public void WriteDaqData(double analogDataOut)
        {

            Task analogOutTask = new Task();

            AOChannel myAOChannel;

            myAOChannel = analogOutTask.AOChannels.CreateVoltageChannel(
                "dev1/ao0",
```

```
                "myAOChannel",
                0,
                5,
                AOVoltageUnits.Volts
                );

            AnalogSingleChannelWriter writer = new
             AnalogSingleChannelWriter(analogOutTask.Stream);

            writer.WriteSingleSample(true, analogDataOut);

        }
    }

}
```

10.3 OPC Read

```
using NationalInstruments;
using NationalInstruments.Net;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace OPC_Read
{
    public partial class Form1 : Form
    {
        DataSocket dataSocket = new DataSocket();

        public Form1()
        {
            InitializeComponent();

            string opcUrl;
            opcUrl = "opc://localhost/MATRIKON.OPC.Simulation/Bucket Brigade.Real4";


            if (dataSocket.IsConnected)
                dataSocket.Disconnect();

            dataSocket.Connect(opcUrl, AccessMode.Read);
        }

        private void btnReadOpc_Click(object sender, EventArgs e)
        {

            dataSocket.Update();

            txtReadOpcValue.Text = dataSocket.Data.Value.ToString();
        }
    }
}
```

10.4 OPC Write

```
using NationalInstruments;
using NationalInstruments.Net;
using System;
using System.Collections.Generic;
```

```
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace OPC_Write
{
    public partial class Form1 : Form
    {

        DataSocket dataSocket = new DataSocket();


        public Form1()
        {
            InitializeComponent();

            string opcUrl;
            opcUrl = "opc://localhost/MATRIKON.OPC.Simulation/Bucket Brigade.Real4";

            if (dataSocket.IsConnected)
                dataSocket.Disconnect();

            dataSocket.Connect(opcUrl, AccessMode.Write);
        }

        private void btnWriteOpc_Click(object sender, EventArgs e)
        {

            double opcValue = 0;

            opcValue = Convert.ToDouble(txtWriteOpcValue.Text);

            dataSocket.Data.Value = opcValue;

            dataSocket.Update();
        }
    }
}
```

Data Acquisition in C#

Hans-Petter Halvorsen

E-Mail: hans.p.halvorsen@usn.no

Web: https://www.halvorsen.blog

https://www.halvorsen.blog