Build a Light Intensity Logger Using Visual Basic 2005

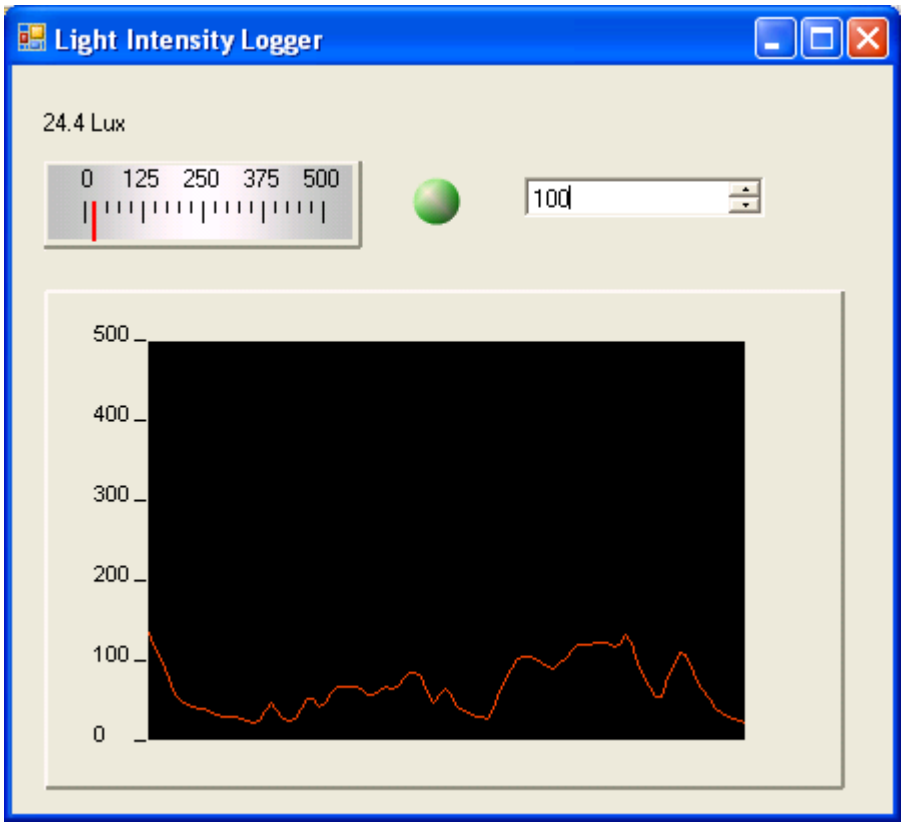


Table of Contents

Learning Objectives....5
Important Note....5
Introduction....6
Types of problems that are solved by measuring light intensity....6
Computer based Measurement System....7
Visual Basic....8
Exercise 1 – Use a Computer based Light Intensity Data logger....9
Objective....9
Exercise 2 – First Visual Basic Program....11
Objective....11
Program 2.1 Hello World....13
Namespace....16
Class....16
Main....16
Important Notes....17
Exercise 3 – Variables, Expressions & Statements....18
Objective....18
Program 3.1 Calculate Equivalent Resistance....18
Variable Declaration....18
Data Types....19
Assignment of Values into Variables....19
Expressions....19
Statements....20
Using Integral Data Types....20
Program 3.2 Calculate Equivalent Resistance using Integral Data Type....21
Explicit Casting....21
How to measure resistance (optional exercise)....22
Exercise 4 – Console Input and Output....23
Objective....23
Program 4.1 Interactive Calculator....23
Data Type String....24
Reading from Console....24
Convert from string to double....24
How to measure current (optional exercise)....25
Exercise 5 – Analog Input (Measure Light Intensity)....26
Objective....26
Class, objects, methods, properties....26
Program 5.1 Measure Light Intensity....27
Referencing Assemblies....27
Enumerations....30
Create the EMANT300 object....30
Open method....30
Read Analog Voltage....30
Close method....30
Imports Directive....31
Using the Simulator....31
Exercise 6 – Analog Output....32
Objective....32
Problem....32
Solution....32
Program 6.1 Measure Temperature using Thermistor....35
Constants....36

Analog Output....36
Math.Log Method....36
Math.Pow Method....36
Exercise 7 – Decision Making Statements....37
Objective....37
Program 7.1 Night Light....37
Relational and Boolean Operators....38
If statement....38
Digital Output....39
Measure temperature in oF or oC (optional exercise)....39
Exercise 8 – For Loop....40
Objective....40
Program 8.1 Measure Light Intensity 10 times....40
For Loop....41
Delay....41
Traffic Lights using LEDs (optional exercise)....42
Exercise 9 – Digital Input....43
Objective....43
Program 9.1 Read Switch....43
Data Type Boolean....43
Read Digital Input....44
Exercise 10 – Do Loop....45
Objective....45
Program 10.1 Measure Light Intensity Until Switch Pressed....45
Do statement....46
Pedestrian Crossing (optional exercise)....46
Exercise 11 – Array....47
Objective....47
Program 11.1 Basic Statistics....47
Array....48
Statistics....49
Console Formatting....49
Exercise 12 – File IO....50
Objective....50
Program 12.1 Log to File....50
StreamWriter Class....50
Exercise 13 – Create a Windows Application....52
Objective....52
Label Control....54
NumericUpDown Control....55
Button Control....55
Events....56
Exercise 14 - Install the EMANT300 Components and Instrument and Controls....61
Objective....61
Exercise 15 – Create an Instrument User Interface....64
Objective....64
Program 15.1 Modify Timer Event Handler....65
Assigning Values to the Controls....67
Appendix A – Using the Simulator....68
Appendix B - Emant300 Class Reference....69
Example....69
Requirements....69
List of Members....70

Emant300 Constructor..... 70
Emant300.Simulation Property..... 70
Emant300.HwId Property..... 70
Emant300.CommPort Property..... 70
Emant300.Open Method..... 71
Emant300.Open Method (bool, string)..... 71
Emant300.Close Method..... 71
Emant300.Reset Method..... 71
Emant300.ConfigDIO Method (Int32)..... 72
Emant300.ConfigPWMCounter Method (Emant300.PWMORCNT, Emant300.EVENTORTIMED, Int32, Int32)..... 72
Emant300.ConfigAnalog Method (Double, Emant300.POLARITY,Int32)..... 72
Emant300.ConfigAnalogAdvance Method (Emant300.POLARITY, Emant300.FILTER, Emant300.CALIBRATION, Boolean, Emant300.REF, Emant300.VREF, Boolean, Emant300.PGA, Int32, Int32, Int32)..... 73
Emant300.ReadAnalog Method (Emant300.AIN, Emant300.AIN)..... 74
Emant300.ReadAnalogWaveform Method (Emant300.AIN, Emant300.AIN, Int32)..74
Emant300.WriteAnalog Method (double)..... 74
Emant300.ReadDigitalBit (Int)..... 75
Emant300.ReadDigitalPort..... 75
Emant300.WriteDigitalBit Method (int, bool)..... 75
Emant300.WriteDigitalPort Method (int)..... 75
Emant300.ReadCounter (out Double)..... 76
Emant300.WritePWM (Double, Double)..... 76
Emant300.AIN Enumeration..... 77
Emant300.VREF Enumeration..... 77
Emant300.POLARITY Enumeration..... 77
Emant300.FILTER Enumeration..... 77
Emant300.CALIBRATION Enumeration..... 79
Emant300.REF Enumeration..... 79
Emant300.PGA Enumeration..... 79
Emant300.PWMORCNT Enumeration..... 79
Emant300.EVENTORTIMED Enumeration..... 81
Appendix C - Emant Instrument Controls Kit Class Reference..... 82
AnalogMeter Class..... 82
LED Class..... 82
Thermometer Class..... 82
LineGraph Class..... 82

Learning Objectives

Following our step by step Instruction Guide, the user will create a Light Intensity Logger, a Night Light, and Universal Thermometer. After completion of these exercises, the user would have learnt the following

Sensor / Actuator

- Photodiode
- Switch
- LED
- Thermistor

Data Acquisition (DAQ)

- Analog Input
- Analog Output
- Digital Input
- Digital Output

Visual Basic

- Basic Program Structure
- Variables & Statements
- Console Input/Output
- Branching Statement if else
- for Loop
- do Loop
- Array
- File IO
- Windows Forms
- Simple Instrument Controls

The exercises will take about 6 hours to complete. The additional optional exercises are intended for the faster learners.

Important Note

The PC must be correctly setup in order for you to complete the exercises in this instructional guide. **Microsoft Visual Basic .NET 2005** must already be installed. Copy the entire *EmantVB2005* folder to your PC, preferably to your *My Documents* folder. The exercises may be completed without the Data Acquisition Module by using the software simulator.

Introduction

"In problem-based learning, the starting point for learning should be a problem, a query or a puzzle that the learner wishes to solve"
Boud. D, 1995, p.13. Enhancing Learning through Self Assessment

Types of problems that are solved by measuring light intensity

- In photography or film production, creative manipulation of light can create stunning images from a low cost consumer camcorder, while poor lighting will cripple the most expensive, state-of-the-art broadcast camera.
- Light intensity is an important management factor for breeder type poultry. There is evidence suggesting a minimum threshold intensity is needed to obtain optimal reproduction performance.
- The amount of light received by a plant is an important factor, because plants' growth may suffer if they do not receive sufficient light. They may lose their characteristic shape and grow thin, leggy stems. New leaves may also be smaller and turn yellow. Some plants may lose their color or turn dark green. Plants exposed to excessive light, on the other hand, may not flower properly or may turn pale green. Different species of plants require different amounts of light.

Light intensity decreases dramatically with distance from the light source. An object located 15 cm from two 40-watt fluorescent lamps receive about 9300 lux, but only 5400 lux if they are 30 cm from the lamps.

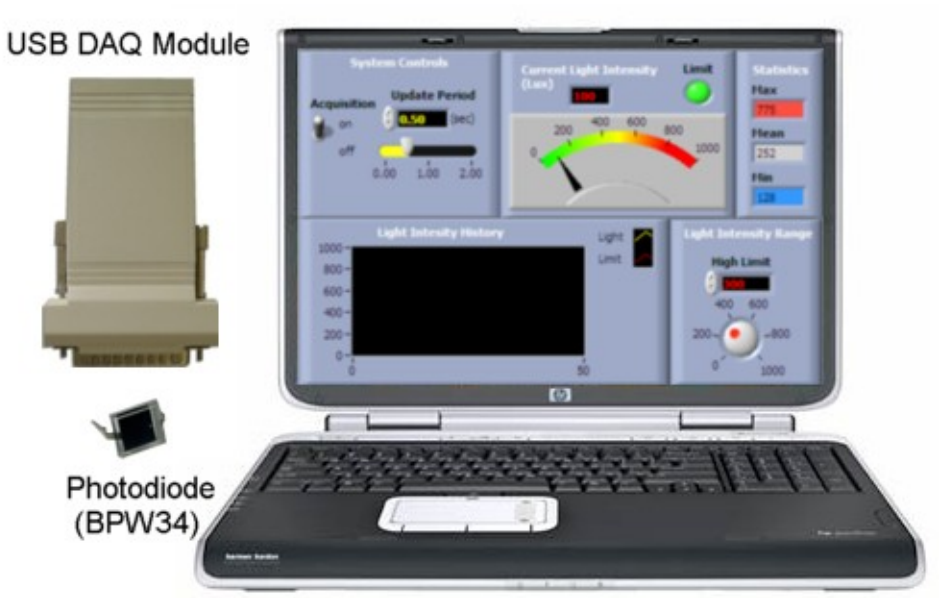
The human eye is a very poor "instrument" for measuring light intensity, because the pupil adjusts constantly in response to the amount of light it receives. To accurately measure the light intensity in a given spot, it is best to use a light meter.

Light intensity may be measured in lux (metric system) or foot-candles (Imperial system). Note that 1 foot-candle = 10.76 lux

To provide some points of reference:

- Full sunlight 11000 lux
- Morning sunlight 6000 lux
- A bright office has about 400 lux
- Moonlight represents about 1 lux

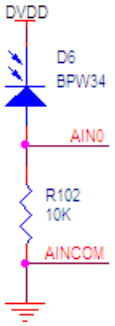
Computer based Measurement System



A computer based light intensity logger is made up of the following components

- Sensor - a photodiode (BPW34)
- Data acquisition module
- Computer with programming software (Visual Basic)

The light sensor used is the BPW34. This is a high speed and high sensitive silicon PIN photodiode in a miniature flat plastic package. A photodiode is designed to be responsive to optical input. Due to its water clear epoxy the device is sensitive to visible and infrared radiation. The large active area combined with a flat case gives a high sensitivity at a wide viewing angle.



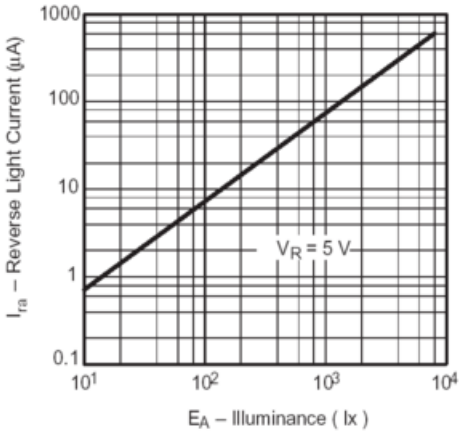


Photo diodes can be used in either zero bias or reverse bias. Diodes have extremely high resistance when reverse biased. This resistance is reduced when light of an appropriate frequency shines on the junction. Hence, a reverse biased diode can be used as a light detector by monitoring the current running through it. Coupled to a 10Kohm resistor, and given the specification of the BPW34 a simple relationship between lux (light intensity) and voltage is given by

$$lux = 1333 * V_o$$

With the Data Acquisition Module (DAQ module), a regular computer can now be used to measure this voltage. The DAQ module contains an 24-bit ADC (analog to digital converter). During analog to digital conversion, a digital value can correspond to a range of analog values. Any analog signal within the zone of one least significant bit (LSB) will have the same digital value. This error is known as quantization error. The relationship between this error and the bit resolution is given by

error = 1 / 2^n where n is the resolution in bits of the ADC

For an 24-bit ADC operating over a 2.5V range, the accuracy you obtain cannot be better than 2.5/16777216 V or 0.15 uV. However due to noise, this is only achievable through careful wiring. In a typical application, a noise floor of about 18 bits or 9 uV is usually achievable.

The DAQ module used (the EMANT300) has up to 6 differential ADC channels, one 8 bit current DAC channel, 8 digital I/Os and either a 16 bit PWM or a Counter. It is connected to the PC via the USB port of the computer.

The Light Application Adaptor for the EMANT300 has the photodiode BPW34 connected to AIN0 (input analog) of the DAQ module and the resistor connected to AINCOM

Visual Basic

In the early years of computing for Engineers, FORTRAN was a popular language. With the introduction of the PCs, Pascal became the language of choice for many programming classes in Engineering. This was followed by C, Visual Basic and C++. All these programming languages have strengths in different areas. Some (like C++) are powerful but difficult to work with while others (Visual Basic) are simpler but limiting in functionality or performance.

Visual Basic was originally created to make creating graphical user interfaces easy. Visual Basic .NET is a very capable object oriented programming language that retains some of the easy to use features of the earlier versions.

In the next few exercises, you will learn how to combine the Data Acquisition Module, Light Application Adaptor and Visual Basic to create a Light Intensity Data Logger.

Web Resource

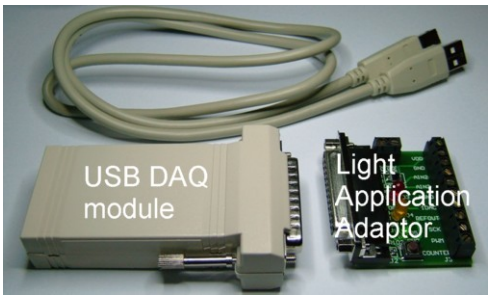
www.emant.com EmAnt Pte Ltd

Exercise 1 – Use a Computer based Light Intensity Data logger

Objective

- Use a computer based measurement system

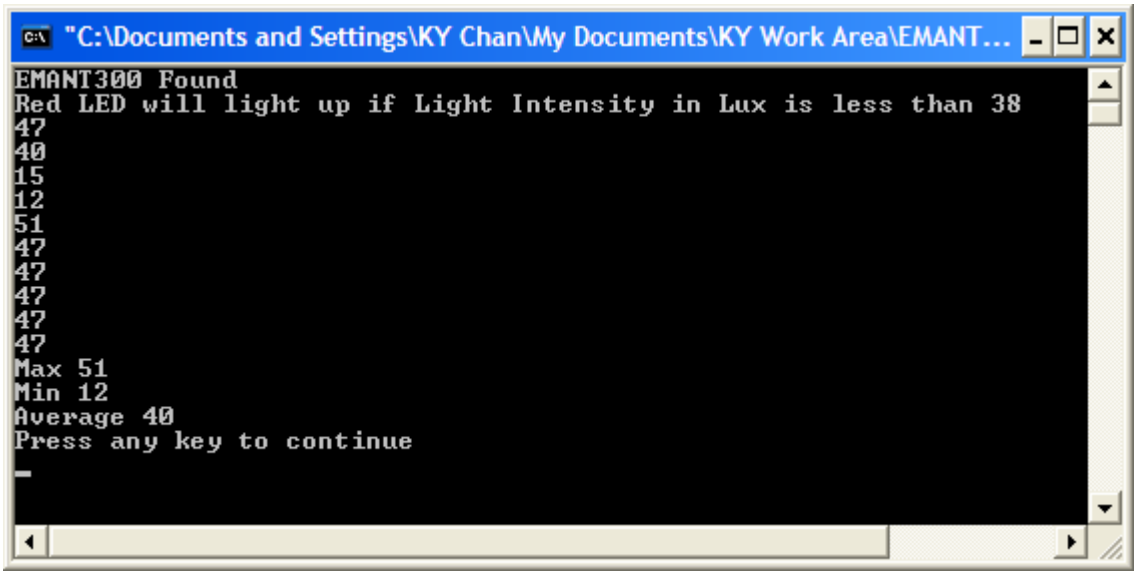
Over the next few exercises, you will learn how to use Visual Basic, the Low Cost Data Acquisition (DAQ) Training kit with a Photodiode to build a Light Intensity Logging System. In this exercise, you will run a Visual Basic program called *Logger.exe* so that you will have an idea of what is a simple console light intensity logger.



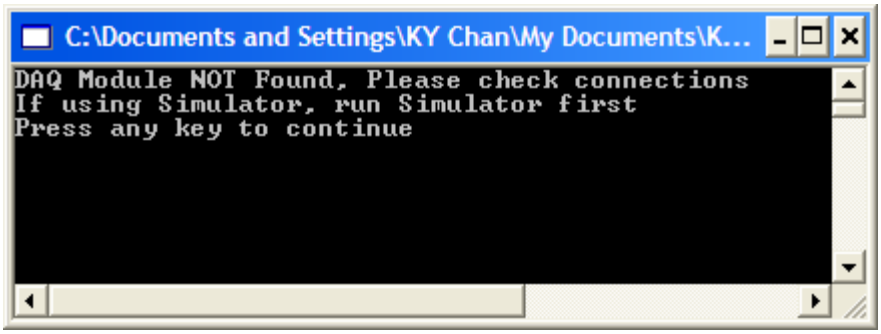
1. Connect the DAQ Training Kit, comprising the USB DAQ Module (EMANT300), Light Application Adaptor and the USB cable, to your computer. *If you do not have the hardware and will be using the simulator for your learning, please read Appendix A before proceeding to step 2.*
2. Browse the *EmantVB2005* folder. Click on *Logger.exe* to run the program
3. This Visual Basic program uses the Photodiode on the Light Application Adaptor to measure the light intensity. The result is shown in Lux
4. The program measures the ambient light intensity and sets the alarm limit to 80% of the value.

5. The program then takes 10 light intensity values in Lux at the rate of one measurement per second. If the Lux level is below the limit, the Red LED will light up. You can change the light intensity measured by covering the Photodiode with your hands.
6. Before the program ends, it calculates the maximum, minimum and average light intensity values in Lux.

3.



Did you see the following instead? Please check your connections if you are using hardware or run LightApp.exe first if you are using the Simulator. Run the Logger.exe again.



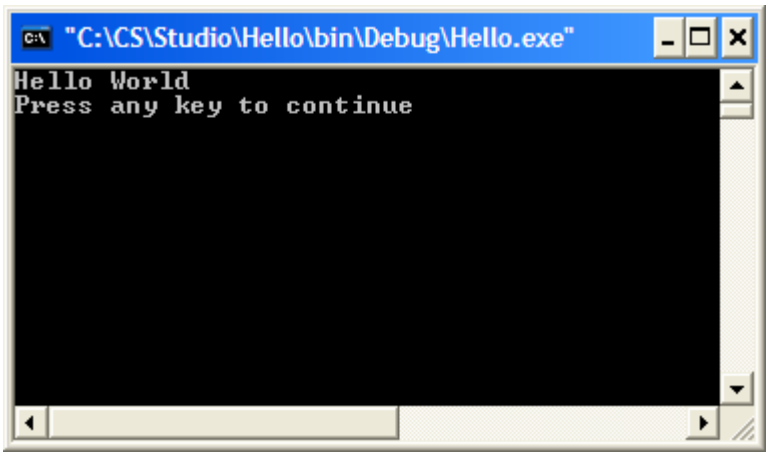
End of Exercise 1

Exercise 2 – First Visual Basic Program

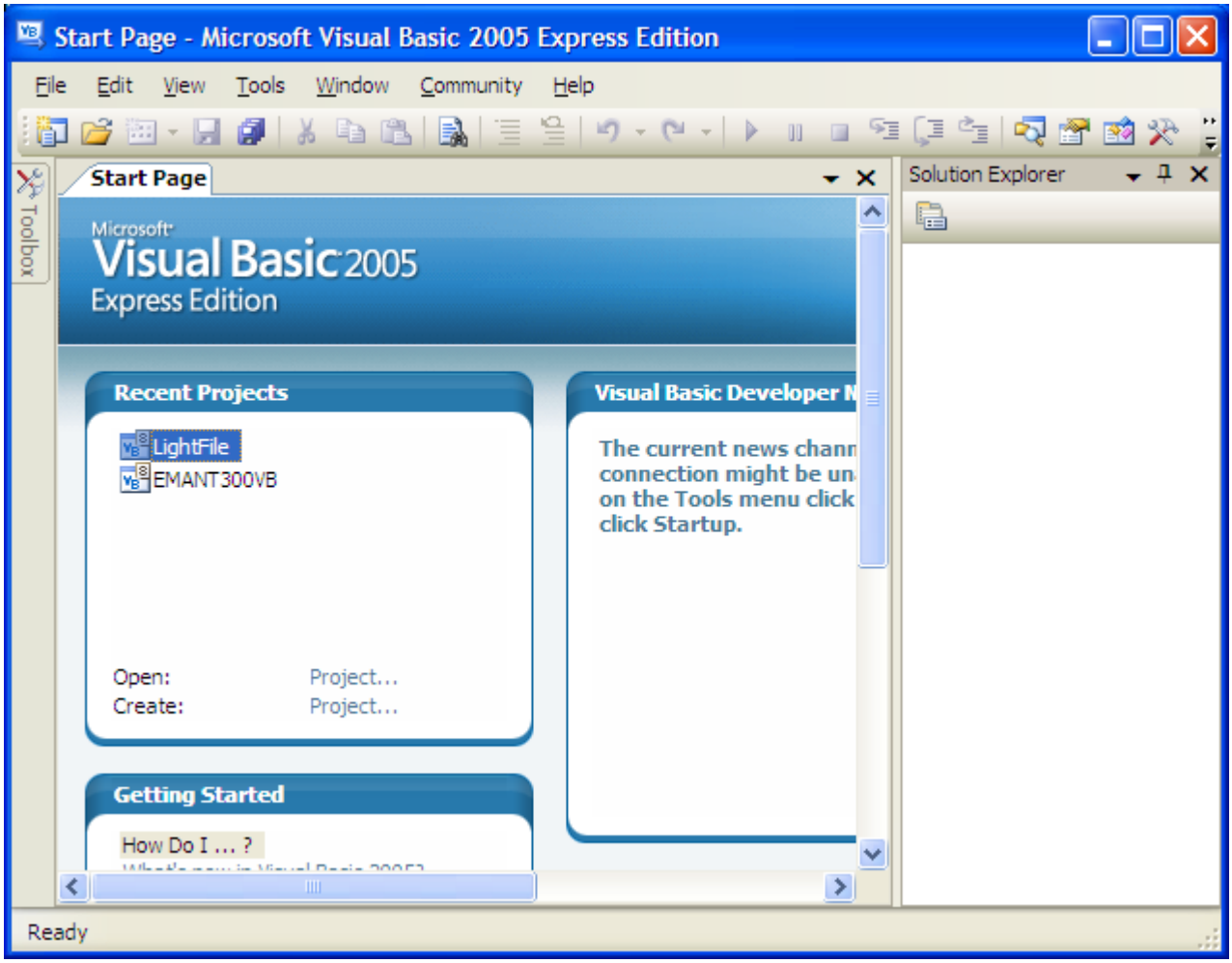
Objective

- Familiarize with the Visual Basic development environment
- Write your first Visual Basic program

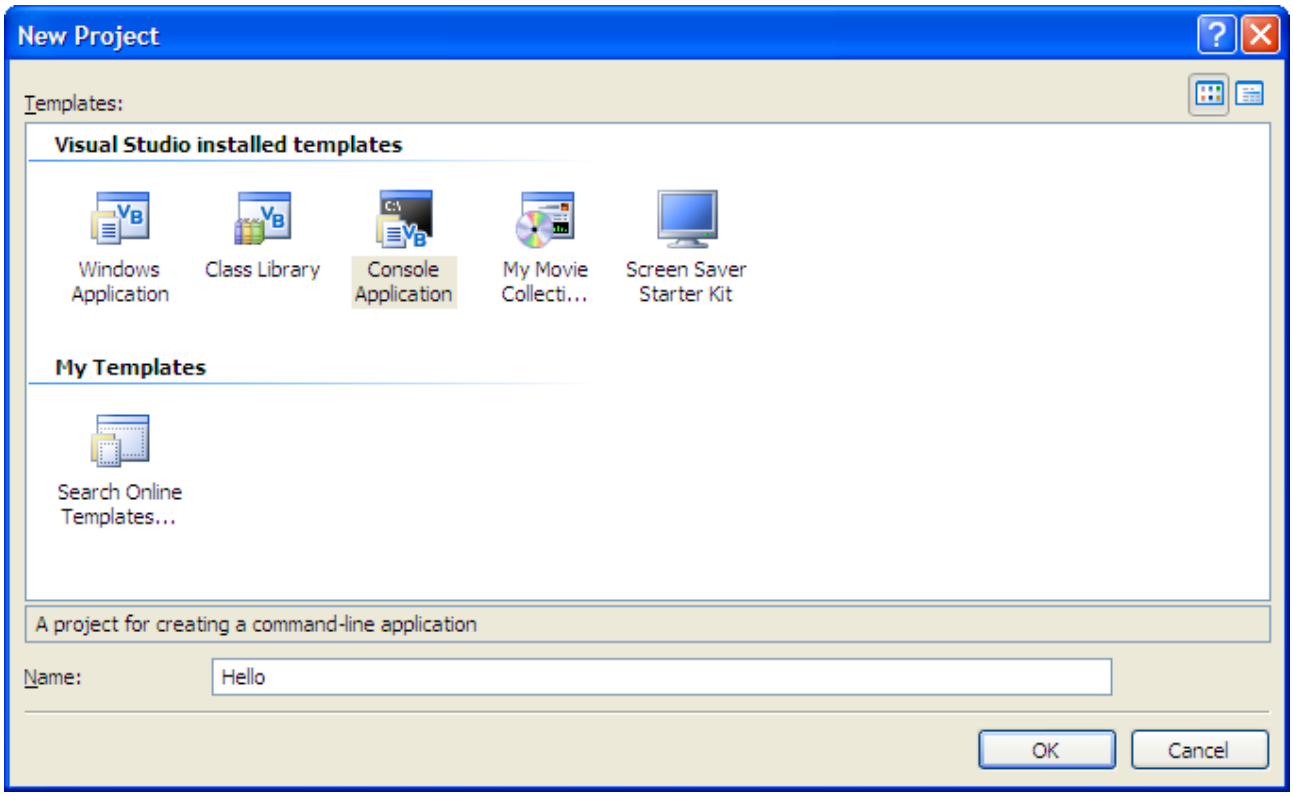
We will create a simple program that displays the message **Hello World** on the screen.



1. Start the Microsoft Visual Basic .NET 2005, the Integrated Development Environment (IDE)

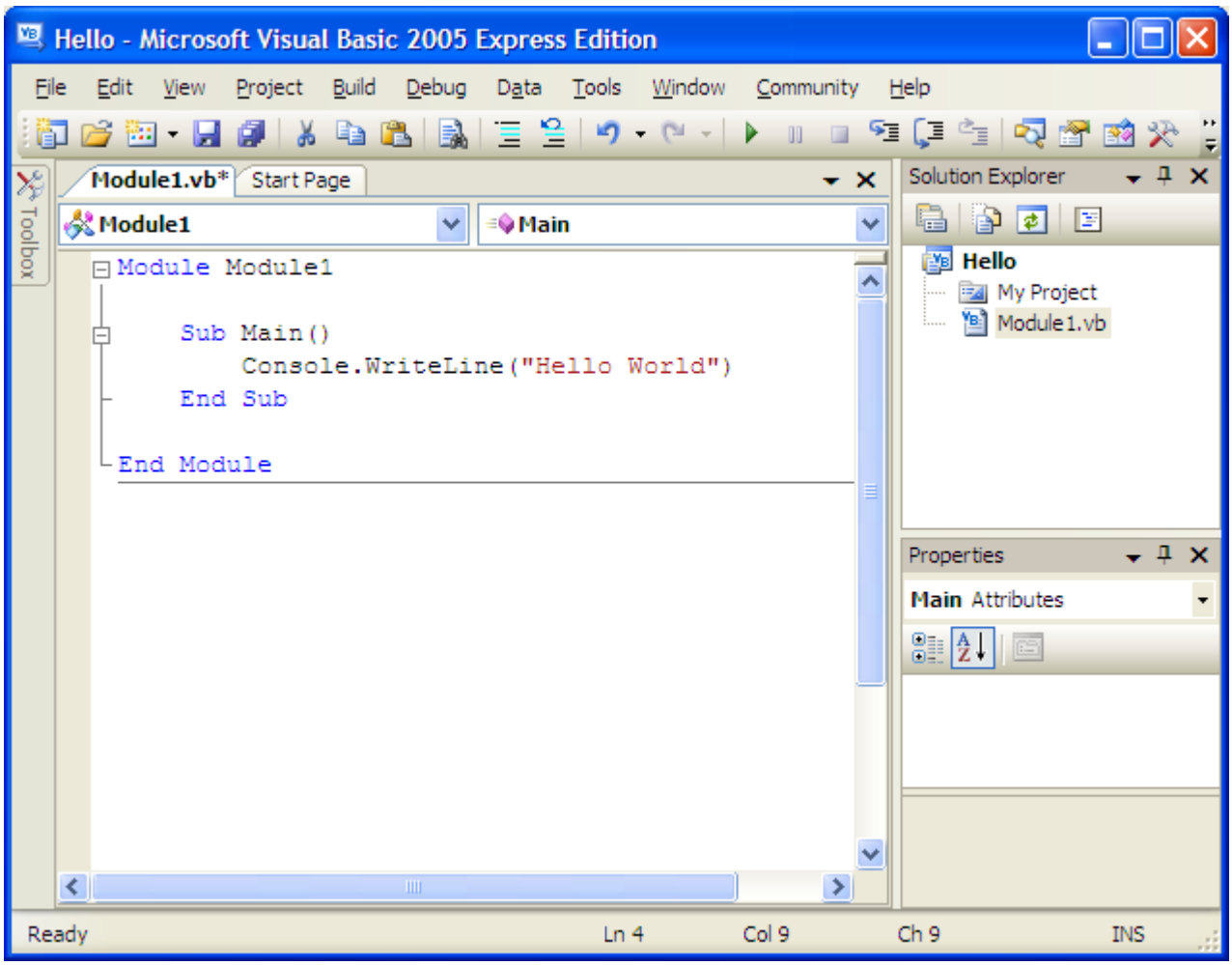


2. Select **File -> New Project**



3. Select **Visual Basic Projects** and click on **Console Application**
4. Call the project name **Hello**.
5. Click on the **OK** button to create the project. The source code for a Visual Basic program is typically stored in one or more text files with a file extension of *.vb* In this example, it is called *Module1.vb.*
6. Add the following line to the generated source code.

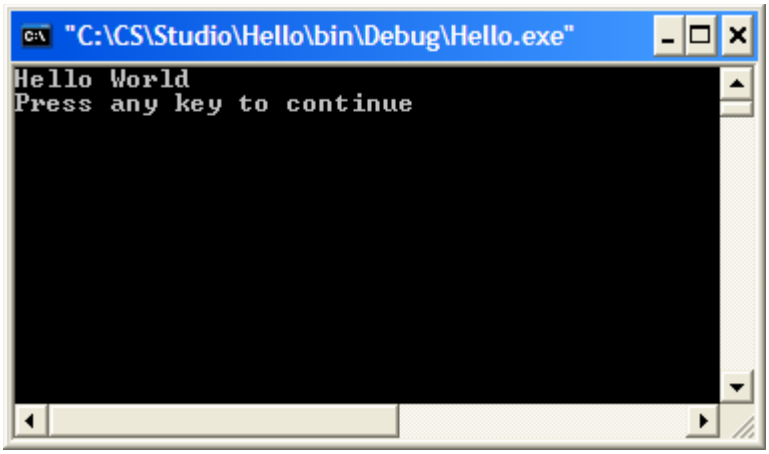
```
Console.WriteLine("Hello World")
```



Program 2.1 Hello World

```
Module Module1
    Sub Main()
        Console.WriteLine("Hello World")
    End Sub
End Module
```

7. Press **Ctrl+F5** to **Start without Debugging** to run your first program. (Press the Ctrl and F5 function keys at the same time)



8. Press any key to return to the development environment.
9. Save the project using **File->Save All**. The project will be stored in the location shown. To store in a different directory, click on the **Browse** button and select the directory you want. Please store your projects in the *EmantVB2005* folder. As some exercises will be built on the earlier exercises to save time, knowing where you saved your projects is important

Namespace

Namespaces provide a hierarchical means of organizing the elements of one or more programs. An analogy would the naming of cities. Do you know that there are two cities named Austin in the world? One is in the USA and the other in Argentina. If you want to send mail to either city, you would write either Austin, USA or Austin, Argentina to be unambiguous. Country names like USA and Argentina in Visual Basic are called namespaces. In Visual Basic, the default namespace is the filename you gave. For your first program, the namespace is *Hello*

Class

```
Module Module1
```

A namespace is usually made up of one or more classes. **A class is a definition for a specific kind of object.** In subsequent exercises you will noticed that all the automatically generated codes have the same class called `Module1`. Thanks to the namespace difference, there is no confusion.

To make the transition from VB6 to VB .NET easier, the *Module* keyword is used instead of *Class* keyword. VB .NET converts the modules to classes automatically.

Fortunately for us, many classes are already predefined for the more common tasks that we need to do. In this program we make use of the `Console` class.

```
  Console.WriteLine("Hello World")
```

The **hello, world** output is produced using the `WriteLine` method of the `Console` Class in the `System` namespace.

Console.WriteLine Method (String)

Writes the specified string value, followed by the current line terminator, to the standard output stream.

Main

```
    Sub Main()

    End Sub
```

The `Main` method is a member of the module `Module1`. The entry point of this application is the method named `Main`.

Important Notes

Please note the following while editing your Visual Basic code to avoid compilation errors

- Visual Basic is not case–sensitive. If you type `Console.Writeline` as `Console.writeline` they are interpreted as the same. Visual Basic applies the appropriate case so if you typed `writeline` it is changed to `Writeline` when you hit enter.
- The line continuation character _ allows you to split a statement over several lines
- ' marks those lines for commenting and are ignored by the compiler

```
' This is a comment
```

End of Exercise 2

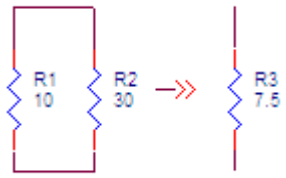
Exercise 3 – Variables, Expressions & Statements

Objective

Learn about

- Variables
- Expressions
- Statements

We will create a program to calculate the equivalent resistance of two resistors in parallel. When you have two resistors in parallel, the equivalent resistance is given by



$$Req = \frac{R1 * R2}{(R1 + R2)}$$

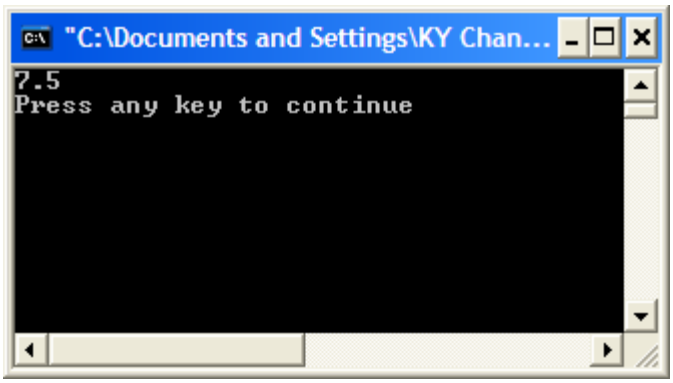
$$Req = 7.5$$

1. Follow the steps that you have learnt in the previous exercise to create a new Visual Basic console project. Name the project *Variables*.
2. Add the highlighted lines to the generated source code.

Program 3.1 Calculate Equivalent Resistance

```
Module Module1

    Sub Main()
        Dim R1, R2, Req As Double
        R1 = 10
        R2 = 30
        Req = R1 * R2 / (R1 + R2)
        Console.WriteLine(Req)
    End Sub

End Module
```

3. Press **Ctrl+F5** to **Start without Debugging** to run your calculator program



4. Press any key to return to the development environment.

Variable Declaration

```
Dim R1, R2, Req As Double
```

- R1, R2 and Req are called variables and they hold values in the computer memory.
- A variable declaration gives it a name and the data type (in this case double). A declaration begins with the keyword `Dim`
- A variable must be declared before using it.

Data Types

- The eight integral types provide support for 8-bit, 16-bit, 32-bit, and 64-bit values in signed or unsigned form.
- The two floating point types, float and double, are represented using the 32-bit single-precision and 64-bit double-precision IEEE 754 formats.
- The decimal type is a 128-bit data type suitable for financial and monetary calculations.
- Visual Basic's bool type is used to represent boolean values—values that are either true or false.
- Character and string processing in Visual Basic uses Unicode encoding. The char type represents a 16-bit Unicode code unit, and the string type represents a sequence of 16-bit Unicode code units.

The following table summarizes Visual Basic's numeric types.

Category	Bits	Type	Range/Precision
Signed integral	16	`Short`	–32,768...32,767
	32	`Integer`	–2,147,483,648...2,147,483,647
	64	`Long`	–9,223,372,036,854,775,808...9,223,372,036,854,775,807
Unsigned integral	8	`Byte`	0...255
Floating point	64	`Double`	5.0×10^{-324} to 1.7×10^{308}, 15-digit precision
Decimal	128	`Decimal`	1.0×10^{-28} to 7.9×10^{28}, 28-digit precision

Assignment of Values into Variables

Once we have declared our variables, we can place values in them by way of the assignment operator =

```
R1 = 10
R2 = 30
```

R1 now holds 10.0 and R2 holds 30.0

The general form of the assignment statement is

```
variable = expression
```

Expressions

Expressions are constructed from operands and operators. The operators of an expression indicate which operations to apply to the operands.

- Examples of operators include +, -, *, /.
- Examples of operands include literals, fields, local variables, and expressions.

When an expression contains multiple operators, the precedence of the operators controls the order in which the individual operators are evaluated. For example, the expression (R1 + R2) is evaluated first because the ( ) operator has higher precedence than the * or / operators.

```
R1 * R2 / (R1 + R2)
```

The following table summarizes arithmetic Visual Basic's operators, listing the operator categories in order of precedence from highest to lowest. Operators in the same category have equal precedence.

Category	Expression	Description
Multiplicative	`x * y`	Multiplication
	`x / y`	Division (float)
	`i \ j`	Division (integer)
	`x Mod y`	Modulus
Additive	`x + y`	Addition, string concatenation, delegate combination
	`x - y`	Subtraction, delegate removal
Assignment	`x = y`	Assignment

Statements

The actions of a program are expressed using statements. Visual Basic supports several different kinds of statements, a number of which are defined in terms of embedded statements.

- Declaration statements are used to declare local variables and constants.
- Expression statements are used to evaluate expressions. Expressions that can be used as statements include method invocations, object allocations using the `New` operator, assignments using = and the compound assignment operators.

```
    Req = R1 * R2 / (R1 + R2);
```

Using Integral Data Types

Integral variables like `Integer` hold whole numbers like 10, 20, 7 compared to floating point variables like `Double` which can hold values like 10.1, 7.5

1. Replace

```
    Dim R1, R2, Req As Double
```

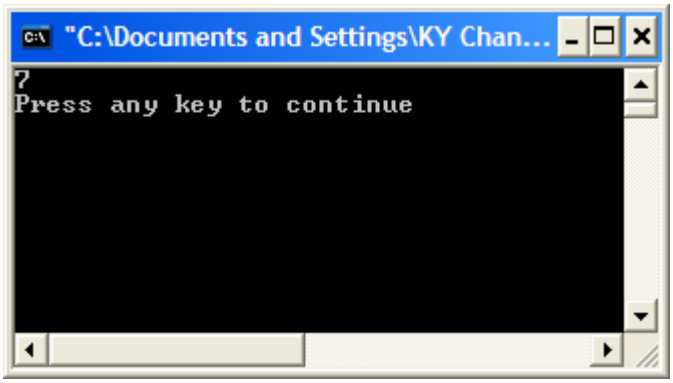
   with

```
    Dim R1, R2, Req As Integer
```


Program 3.2 Calculate Equivalent Resistance using Integral Data Type

```
Module Module1

    Sub Main()
        Dim R1, R2, Req As Integer

        R1 = 10
        R2 = 30
        Req = R1 * R2 \ (R1 + R2)
        Console.WriteLine(Req)
    End Sub

End Module
```

2. Run the program again. The result shows 7 rather than 7.5 because the answer is truncated. So be careful when selecting the data type for your variables to prevent errors.



Explicit Casting

```
  Dim R1 As Double
  Dim R2 As Integer
  R1 = 10.5
  R2 = R1
```

If you add the following line to the top of your source code

```
Option Strict On
```

the following line will report a compilation error because you try to assign a Double to an Integer

```
  R2 = R1
```

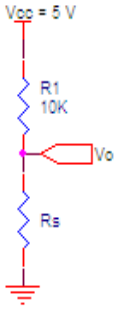
To avoid the compilation error, use **casting** where the term `CType(expression, typename)` tells the compiler to convert the value within into an integer value. However, it will still result in a loss of data - R2 is 10 instead of 10.5. You can use this type of conversion to convert any numeric data types.

```
  R2 = CType(R1, Integer)
```

How to measure resistance (optional exercise)

Data Acquisition Cards normally measure only voltages. Often it is necessary to measure resistances if the sensor used changes its resistance with respect to the physical phenomenal measured.

Examples include the Light Dependent Resistor where the resistance changes with Light Intensity and the Thermistor where resistance changes with temperature. One simple solution is to connect a resistor in series with the sensor.



$$Vo = \frac{Rs}{(R1 + Rs)} * Vcc$$

$$Rs = \frac{R1 * Vo}{(Vcc - Vo)}$$

The following Visual Basic code calculates the sensor resistance given the voltage across the sensor is 3.0 V.

```
Dim R1, Rs, Vcc, Vo As Double

R1 = 10000.0
Vo = 3.0
Vcc = 5.0
Rs = (R1 * Vo) / (Vcc - Vo)
Console.WriteLine(Rs)
```

End of Exercise 3

Exercise 4 – Console Input and Output

Objective

- Learn Console Input
- Revise Console Output
- Revise Variables, Statements and Expressions

In the previous exercise, if we want to calculate the resistance given the voltage we need to edit the Visual Basic source code, recompile and rerun. It is easier if we can run the same program and interactively enter the voltage.

In this exercise, we will learn how to enter the voltage interactively using Console Input.

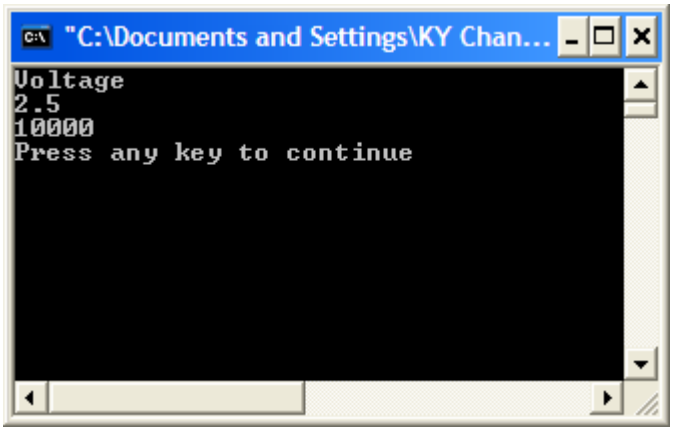
1. Create a new Visual Basic console project. Name the project *Consoleio*.
2. Add the highlighted lines to the generated source code.

Program 4.1 Interactive Calculator

```
Module Module1

    Sub Main()
        Dim R1, Rs, Vcc, Vo As Double
        Dim myinput As String
        Console.WriteLine("Voltage")
        myinput = Console.ReadLine
        Vo = Convert.ToDouble(myinput)
        R1 = 10000
        Vcc = 5
        Rs = (R1 * Vo) / (Vcc - Vo)
        Console.WriteLine(Rs)
    End Sub

End Module
```

3. Press **Ctrl+F5** to **Start without Debugging** to run your calculator program. Key in a voltage between 0 and 4.5. The program will return the resistance value.

4. .



4. Press any key to return to the development environment.

5. Run the program several times. Enter a different value of voltage each time and find the resistance value.

Data Type String

```
Dim myinput As String
```

In the earlier exercise we have looked at the use of numeric variables. However, the processing of text data is also needed. Visual Basic provides the string data type and string variables can hold any characters. An example of a string is “Hello World”. If you want to assign a string variable, you must enclose the string in quotes

```
myinput = "Hello World"
```

You can join or concatenate strings

```
myinput = "Hello" + " World"
```

Reading from Console

```
  myinput = Console.ReadLine
```

The `ReadLine` method of the `Console` class prompts the user for input and returns the string that was entered and assign it to the string variable `myinput`.

Console.ReadLine Method

Reads the next line of characters from the standard input stream.

Convert from string to double

```
Vo = Convert.ToDouble(myinput)
```

To convert the string variable to another data type, Visual Basic provides a `Convert` class in the `System` namespace that supports just about every conversion between intrinsic types, such as int, double and string. Our code converts the string to a double and assigns it to `Vo`.

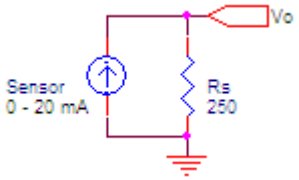
The `Convert` class returns a type whose value is equivalent to the value of a specified type. Some of the common methods.

Method	Conversion
ToBoolean	Converts a specified value to an equivalent Boolean value.
ToDateTime	Converts a specified value to a DateTime.
ToDouble	Converts a specified value to a double-precision floating point number.
ToInt32	Converts a specified value to a 32-bit signed integer.
ToString	Converts the specified value to its equivalent String representation.
ToUInt32	Converts a specified value to a 32-bit unsigned integer.

How to measure current (optional exercise)

In the process industry where sensors are located long distances from the measurement unit, the sensor is often connected to a signal conditioning unit which output a current that corresponds to the physical measurement.

A typical sensor could be a CO (carbon monoxide) sensor which outputs a 0 - 20 mA current that corresponds to a 0 – 300 ppm CO concentration. To measure this current, a simple solution would be to add a shunt resistance of 250 ohm.



$$Io = \frac{Vo}{R5}$$

The following Visual Basic code calculates the current input given the voltage across the load resistance.

```
Dim R1, Io, Vo As Double
Dim myinput As String
Console.WriteLine("Voltage")
myinput = Console.ReadLine
Vo = Convert.ToDouble(myinput)
R1 = 250
Io = Vo / R1
Console.WriteLine(Io)
```

End of Exercise 4

Exercise 5 – Analog Input (Measure Light Intensity)

Objective

- Learn Analog Input
- Learn more about Class and Object

In this exercise, we will measure the light intensity using a Photodiode and the EMANT300, a Data Acquisition Module.

Class, objects, methods, properties

We will use cars to illustrate the above concepts.

- Cars have registrations numbers to differentiate one car from another.
- Cars can be described by their make or their color
- Cars can be driven forward and in reverse.

We have two cars, SFL8772 is a Grey Toyota while SCU7056 is a Gold Honda.

In Visual Basic terminology, we would say that

- `Car` is a class
- `SFL8772` and `SCU7056` are objects and instances of the `Car` Class.
- `SFL8772.Color = Grey; SCU.Color = Gold`
  `Color` is property of the `Car` Class
- `SFL8772.Make = Toyota; SCU.Make = Honda`
  `Make` is another property of the `Car` Class
- `SFL8772.Drive(forward); SCU7056.Drive(reverse)`
  `Drive` is a method of the `Car` Class.
- To identify the property, we put a dot or '.' between the object and the property and assign an appropriate value.
- The difference between method and property is that a method is associated with a task.
- A method may or may not require parameters like `forward` or `reverse` to be passed.

In your previous exercises, you have used the `Console` class and the methods `ReadLine` and `Writeline` to allow your program to interact with the user using the computer monitor and keyboard.

To perform Data Acquisition, we have provided a class called `EMANT300` which works with the EMANT300 Data Acquisition Module and allows your program to interact with the physical world..

5. Create a new Visual Basic console project. Name the project *ReadLight*
6. Add the highlighted lines to the generated source code.
7. Uncomment (remove ') the following line if you are using the simulator

```
' DAQ.Simulation = True
```

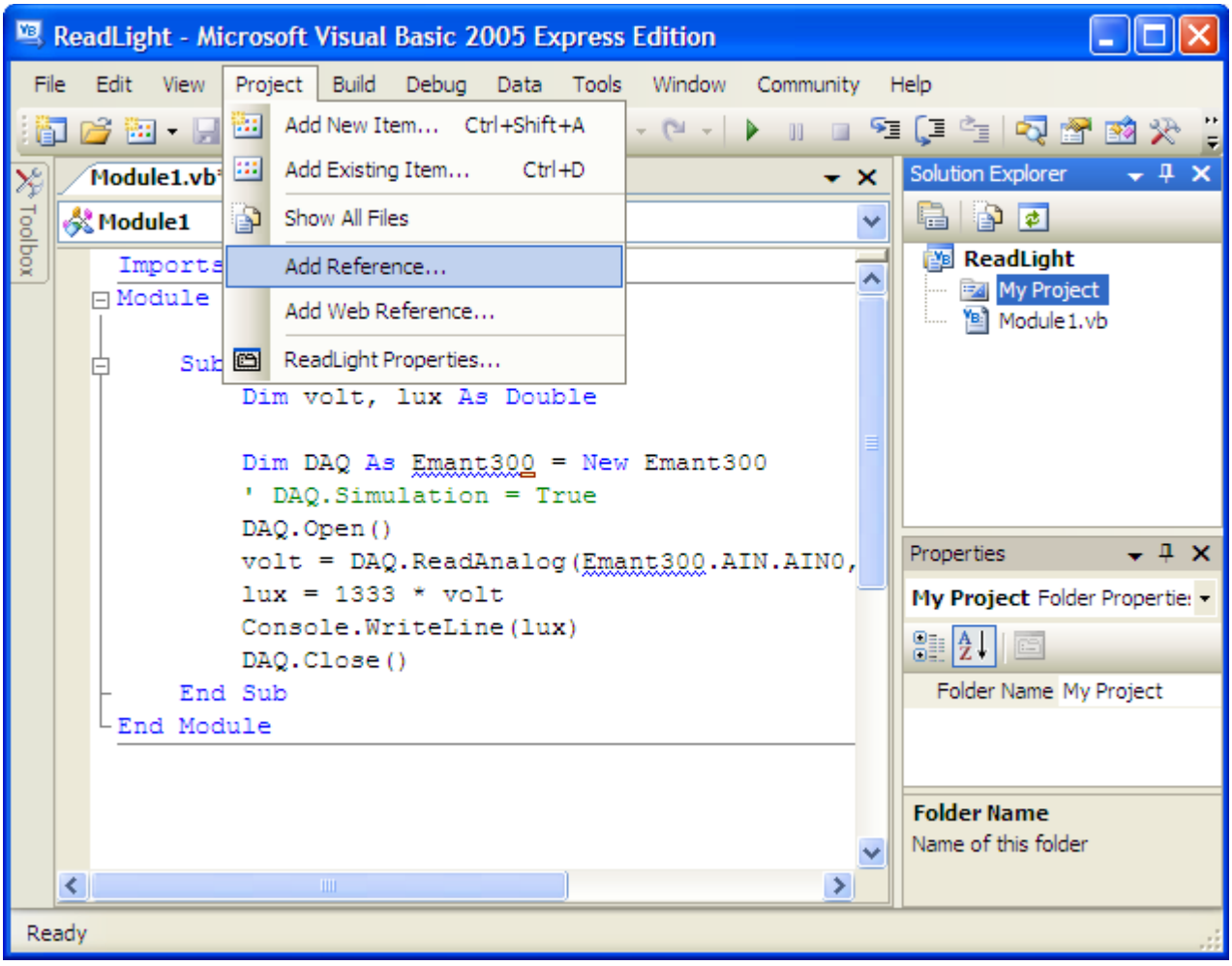
*If you are using the **simulator** rather than the actual hardware, please read Appendix A now on instructions to program the simulator. The simulator must be running before you start your own program.*

Program 5.1 Measure Light Intensity

```
Imports Emant
Module Module1

Sub Main()
  Dim volt, lux As Double

  Dim DAQ As Emant300 = New Emant300
' DAQ.Simulation = True
  DAQ.Open()
  volt = DAQ.ReadAnalog(Emant300.AIN.AIN0, Emant300.AIN.COM)
  lux = 1333 * volt
  Console.WriteLine(lux)
  DAQ.Close()
End Sub

End Module
```

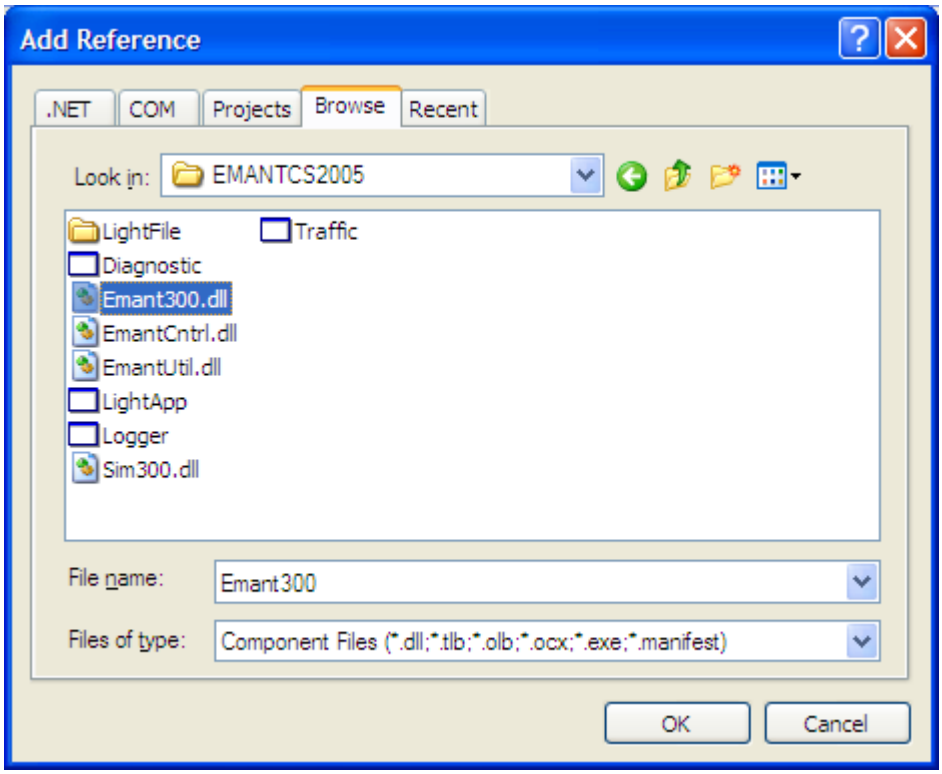
Referencing Assemblies

4. Every project contains a **References** folder for identifying physical assemblies the code in the project uses. In order to use the `Emant300` class, the program must reference the assembly *Emant300.dll* . Right Click on the **Project-> Add Reference...**



8.

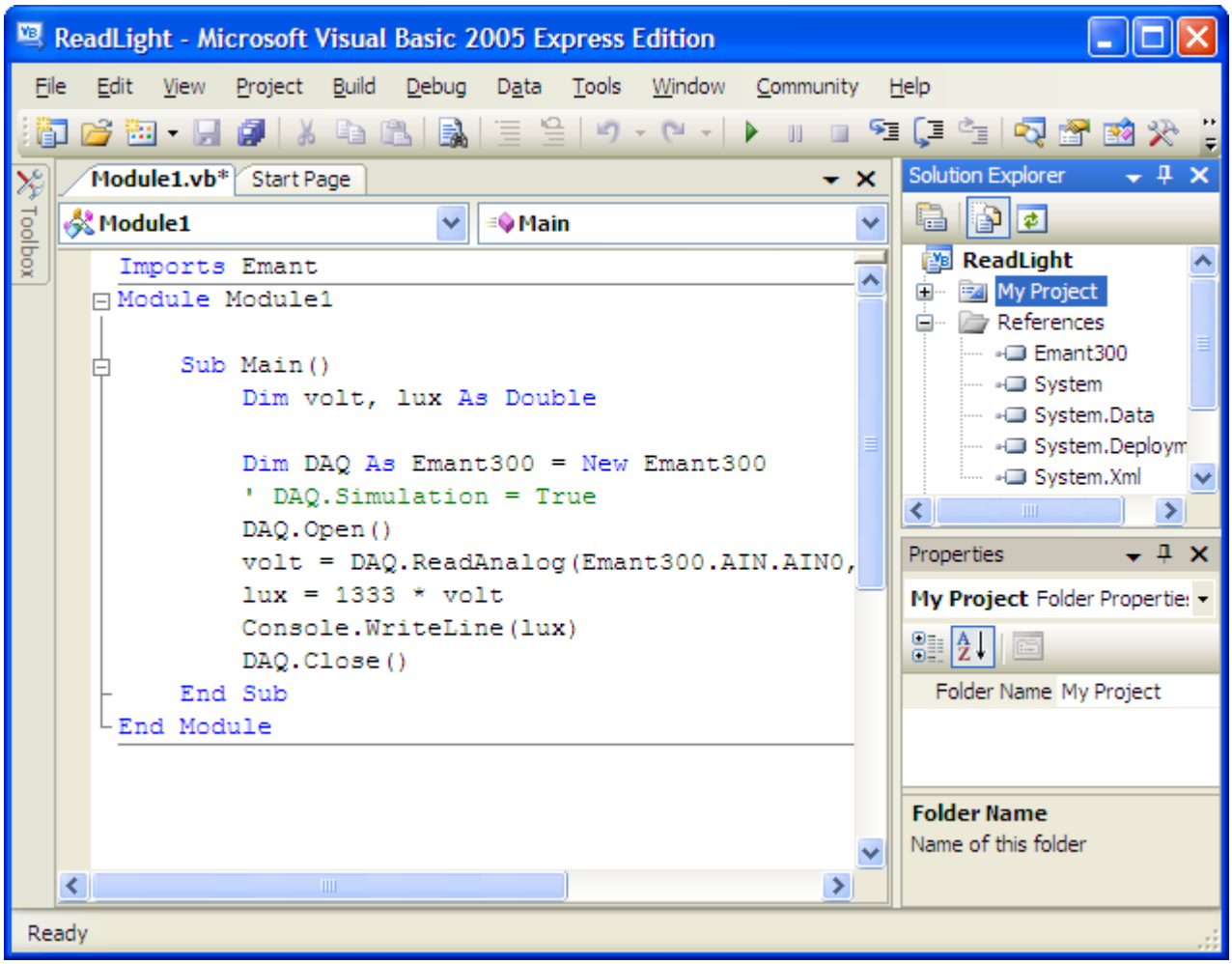
5. Click on the **Browse** tab



6. Go to the *EmantVB2005* folder. Select the assembly file *Emant300.dll*
7. Click OK to add *Emant300.dll*

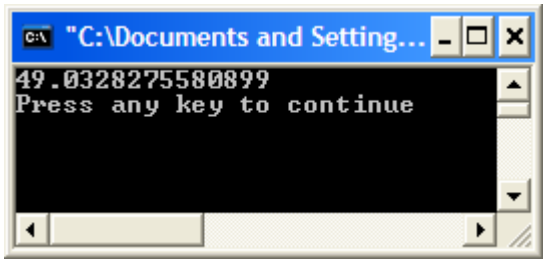
9.

8. Click **Project->Show All Files** You should see the *Emant300* Assembly included in the *References* folder.



9. Press **Ctrl+F5** to **Start without Debugging** to run your light measurement program.

10.



10. Press any key to return to the development environment.

11. Run the program several times. Cover the Photodiode with your hand to observe the change in light intensity measured.

12. Save your project by clicking on **File -> Save All**. You will be using this program in a later exercise.

Enumerations

Enumerations are strongly typed of constants that help to make programming more meaningful and safe. In this example, the analog inputs for the EMANT300 module are fixed by hardware. The corresponding enumeration is

Member name	Description
AIN0	AIN0 – Analog Input 0
AIN1	AIN1 – Analog Input 1
AIN2	AIN2 – Analog Input 2
AIN3	AIN3 – Analog Input 3
AIN4	AIN4 – Analog Input 4
AIN5	AIN5 – Analog Input 5
COM	AINCOM – Common Analog Input
DIODE	DIODE – Temperature Sensing Diode

Create the EMANT300 object

```
Dim DAQ As Emant300 = New Emant300
```

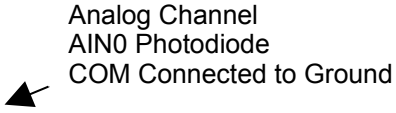
An instance of `EMANT300` is created and called `DAQ`. The variable `DAQ` is the equivalent of the car registration number. All future references to this object will use this name. See Appendix B for the full description of this Class.

Open method

```
DAQ.Open()
```

Open is a method that instructs the program to connect to the DAQ module that is physically connected to USB port.

Read Analog Voltage




```
volt = DAQ.ReadAnalog(Emant300.AIN.AIN0,Emant300.AIN.COM)
```

The analog voltage across AIN0 and GND (COM) is read. The 10K resistor reading the Photodiode current is connected to AIN0 and GND (COM). `Emant300.AIN.AIN0` and `Emant300.AIN.COM` are the respective enumeration.

```
lux = 1333 * volt
```

The voltage is converted to Lux and then displayed on the console output. See page 7 for theory behind the calculations.

Close method

```
DAQ.Close()
```

Finally the `DAQ` connection is closed. To ensure that your programs end correctly, always call the Close method before you exit your programs.

Imports Directive

```
Imports Emant
```

Allow unqualified reference to `Emant300`. If you don't include this `Imports` directive, all references to the `Emant300` and its object will have to full

```
Dim DAQ As Emant.Emant300 = New Emant.Emant300
```

is shortened to

```
Dim DAQ As Emant300 = New Emant300
```

Using the Simulator

```
' DAQ.Simulation = True
```

Recall from exercise 2 that ' marks those lines for commenting and are ignored by the compiler. Therefore if you have entered the code with the ' you have commented out the line of code. The Emant300 component defaults to actual hardware when the Simulation property is not set. If you are using the simulator, then you should uncomment the line as below

```
  DAQ.Simulation = True
```

If you uncomment the line and use the hardware later, then just set the property to false.

```
  DAQ.Simulation = False
```

End of Exercise 5

Exercise 6 – Analog Output

Objective

- Analog Output

Problem

Severe acute respiratory syndrome (SARS) is a respiratory or airways infection that broke out in Asia in early 2003. It is highly infectious and death from the disease is not uncommon. Fever is one of the first signs of SARS. On June 1, 2003, in Taiwan, a National Temperature Monitoring Campaign was launched. In the campaign, fever was defined as forehead or axillary temperature >37°C. You have been tasked to create a fever detector.

Solution

Warning: This is intended as proof of concept exercise and not intended for real life use.

A thermistor will be used to measure the forehead temperature. Thermistors are widely used in industrial applications because of their sensitivity, small size, ruggedness and low cost. Thermistors have an electrical resistance that varies non-linearly with temperature. The R-T characteristics of most thermistors can be described by the Steinhart-Hart equation:

$$1/T = A + B*(\text{Ln } R) + C*(\text{Ln } R)^3$$

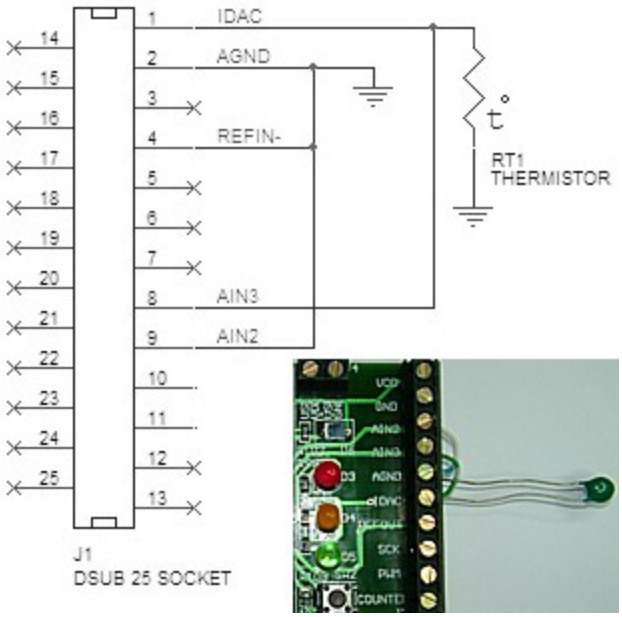
T is the absolute temperature (in Kelvin) and A, B, and C are constants which can be determined by measuring three sets of resistance and temperature values during calibration.

Most thermistors have a negative temperature coefficient (NTC), their resistance decreases with increasing temperature. Thermistors are specified according to its nominal resistance at 25 ^{o}C and commonly available thermistors range from 250 ohms to 100 kohms

The thermistor that we are using has the following characteristics

- Nominal resistance @ 25 ^{o}C: 10 kohms
- negative temperature coefficient (NTC)
- Steinhart-Hart equation parameters:
    - A= 0.001129148
    - B= 0.000234125
    - C= 8.76741E-8

As the DAQ module Analog Input measures only voltage, we will need to provide a current source to convert the resistance to voltage. The EMANT300 has an 8 bit current DAC (digital to analog converter). As the DAC has 8 bits resolution, we can drive the resistance from 0 to 1mA in 255 steps with increments of about 39uA. In our exercise, we will drive 0.1mA into the thermistor. As the thermistor has a nominal value of 10 kohm at 25 ^{o}C, at this temperature the voltage across the thermistor will be (0.1mA * 10 kohm) = 1V.



- Connect the thermistor to the Light Application Adaptor screw terminals labeled IDAC and AGND
- Connect a wire from IDAC to AIN3
- Connect a wire from AGND to AIN2

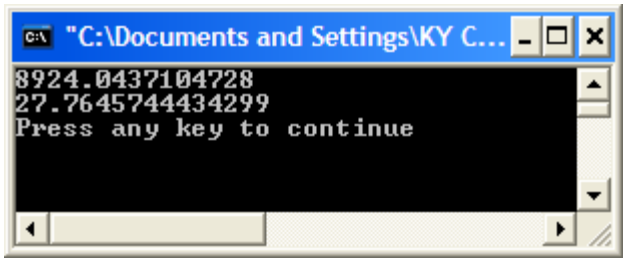
1. Create a new Visual Basic console project. Name the project `Temperature`
2. Add the highlighted lines to the generated source code
3. Uncomment (remove ') the following line if you are using the simulator

```
' DAQ.Simulation = True
```

Program 6.1 Measure Temperature using Thermistor

```
Imports Emant
Module Module1

    Sub Main()
     Const A As Double = 0.001129148
     Const B As Double = 0.000234125
     Const C As Double = 0.0000000876741
     Dim volt, temp, R As Double

     Dim DAQ As Emant300 = New Emant300
     'DAQ.Simulation = True
     DAQ.Open()
     DAQ.WriteAnalog(0.1)
     volt = DAQ.ReadAnalog(Emant300.AIN.AIN3, Emant300.AIN.AIN2)
     R = volt / 0.0001
     Console.WriteLine(R)
     temp = 1 / (A + B * Math.Log(R) + C * Math.Pow(Math.Log(R), 3))
     temp = temp - 273
     Console.WriteLine(temp)
     DAQ.Close()
    End Sub

End Module
```

4. In order to use the `Emant300` class, add the class library *Emant300.dll* to the references folder. Repeat steps 4 to 8 from exercise 5.
5. Press **Ctrl+F5** to **Start without Debugging** to run the program. Observe the thermistor resistance and temperature value.



6. Press any key to return to the development environment.
7. Use your finger to touch the thermistor. Rerun the program. The temperature should change to reflect the higher temperature of your body.

Constants

Constants are used when you value you have assigned is fixed throughout the program. Unlike **Variables**, the value cannot be reassigned. Thus it is suitable for A, B and C found in the Steinhart-Hart equation.

```
Const A As Double = 0.001129148
Const B As Double = 0.000234125
Const C As Double = 0.0000000876741
```

Analog Output

```
DAQ.WriteAnalog(0.1)
```

The parameter 0.1 (variable is double data type) sets the current output to 0.1 mA. Value must be between 0 to 1 mA

Math.Log Method

Returns the natural (base e) logarithm of a specified number.

```
Math.Log(R)
```

It is one of the methods from the `Math` Class. The `Math` Class provides constants and static methods for trigonometric, logarithmic, and other common mathematical functions. The other method we used in this example is

Math.Pow Method

Returns a specified number raised to the specified power..

```
Math.Pow(Math.Log(R),3)
```

The line perform the following calculation $(\text{Ln } R)^3$

Information on the `Math` Class can be found at the Microsoft website. Search using "Math.Log MSDN"

End of Exercise 6

Exercise 7 – Decision Making Statements

Objective

- Learn `If` statement
- Revise analog input
- Learn digital output

Night lights are normally used in children's bedrooms to reduce their fear of the dark and to provide some light in a darkened room. But even without children, a home should use night lights for safety. The lights use built-in photo sensors so that they turn on only at night.

In this exercise, we will program the light sensor and the Green LED to create a simple night light. We use the digital outputs to turn on and off the LED. To do this we will use the if statement

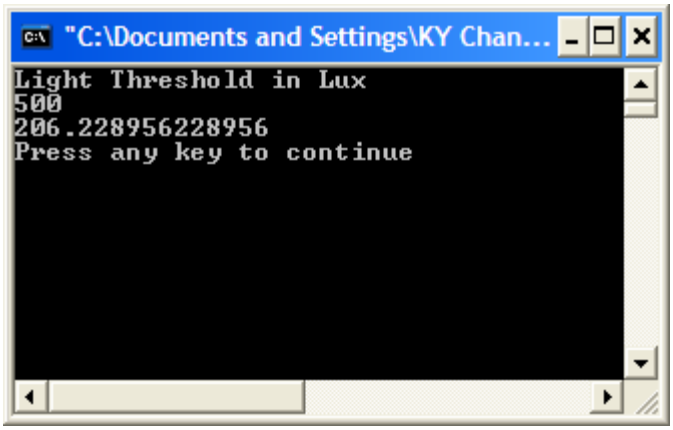
1. Create a new Visual Basic console project. Name the project *Ifled*
2. Add the highlighted lines to the generated source code
3. Add the following line if you are using the simulator

```
DAQ.Simulation = True
```

Program 7.1 Night Light

```
Imports Emant
Module Module1

    Sub Main()
        Dim volt, lux, compare As Double
        Dim input As String
        Console.WriteLine("Light Threshold in Lux")
        input = Console.ReadLine
        compare = Convert.ToDouble(input)
        Dim DAQ As Emant300 = New Emant300
        DAQ.Open()
        volt = DAQ.ReadAnalog(Emant300.AIN.AIN0, Emant300.AIN.COM)
        lux = 1333 * volt
        Console.WriteLine(lux)
        If lux < compare Then
            DAQ.WriteDigitalBit(0, True)
        Else
            DAQ.WriteDigitalBit(0, False)
        End If
        DAQ.Close()
    End Sub

End Module
```

4. In order to use the `Emant300` class, add the class library *Emant300.dll* to the references folder. Repeat steps 4 to 8 from exercise 5.
5. Press **Ctrl+F5** to **Start without Debugging** to run the program. Enter the Lux threshold and observe the bulb brightness. The Light Intensity measured in Lux is also displayed.



6. Press any key to return to the development environment.
7. Run the program several times. Choose a Lux threshold that is between the light intensity when the Photodiode is covered and when it is not. Observe the Green LED turning on when the light level drops below the threshold.

Relational and Boolean Operators

Relational Operators are used to compare data. They are used to determine the result of a comparison to `True` or `False`

```
lux < compare
```

The following table summarizes some of Visual Basic's relational and boolean operators, listing the operator categories in order of precedence from highest to lowest. Operators in the same category have equal precedence.

Category	Expression	Description
Relational and type testing	`x < y`	Less than
	`x > y`	Greater than
	`x <= y`	Less than or equal
	`x >= y`	Greater than or equal
	`x Is T`	Return `true` if `x` is a `T`, `false` otherwise
Equality	`x == y`	Equal
	`x != y`	Not equal
Logical AND	`x And y`	Integer bitwise AND, boolean logical AND
Logical XOR	`x Xor y`	Integer bitwise XOR, boolean logical XOR
Logical OR	`x Or y`	Integer bitwise OR, boolean logical OR

If statement

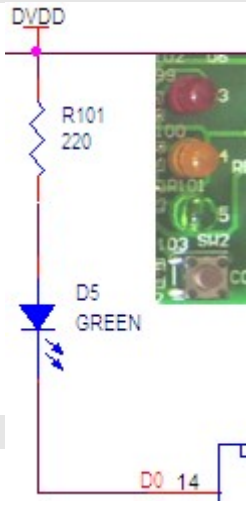
The If statement selects a statement for execution based on the value of a boolean expression.

If statement:
`If` boolean-expression `Then` statement `End If`
`If` boolean-expression `Then` statement `Else` statement `End If`

An if statement is executed as follows:

1. The boolean-expression is evaluated.
2. If the boolean expression yields true, control is transferred to the statement following **The**n. When and if control reaches **Else**, control is transferred to the end point of the if statement.
3. If the boolean expression yields false and an **Else** part is present, control is transferred to the statement following **Else**.
4. If the boolean expression yields false and if an **Else** part is not present, control is transferred to the end point of the if statement.

```
If lux < compare Then
   DAQ.WriteDigitalBit(0, True)
Else
   DAQ.WriteDigitalBit(0, False)
End If
```



In our example, assuming compare = 100 and two measurements are taken, one with lux = 50 and the second with lux = 200

1. the boolean expression is `lux < compare`
2. lux = 50: DAQ.WriteDigitalBit(0, True); is executed and 5 V is output to the Green LED. The LED turns off.
3. lux = 200: DAQ.WriteDigitalBit(0, False); is executed and 0 V is output to the Green LED. The LED turns on.

Digital Output

State

```
   DAQ.WriteDigitalBit(0, True)
```

DIO Bits Assignment
0 Green LED (output)
1 Orange LED (output)
2 Red LED (output)
3 Switch (input)

The above code turns the Green LED.

Measure temperature in °F or °C (optional exercise)

Temperature can be measured using either the Centigrade scale or Fahrenheit scale. The Fahrenheit scale is common in the USA whereas most of the other countries adopt the Centigrade or metric scale. If you are developing a temperature measurement solution that will be used worldwide, you have to allow the user the option of selecting either scale.

Develop a Visual Basic program that measures temperature and allows the reading in either °F or °C

Hints:

1. Use the temperature sensor program developed earlier
2. Use the if statement
3. To convert F to C use the following equation

$$F = \frac{C * 9}{5} + 32$$

End of Exercise 7

Exercise 8 – For Loop

Objective

- For Loop

When Sam was doing the light measurement exercise, he thought that there must be a better to measure the light intensity repeatedly without him having to run the program each time he needs a reading.

Computers are capable of doing things over and over again without getting tired or bored like Sam. An interactive or repetitive loop allows a set of statements to be repeated.

In this exercise, we will create a program that measures the light intensity 10 times at the rate of one measurement per second.

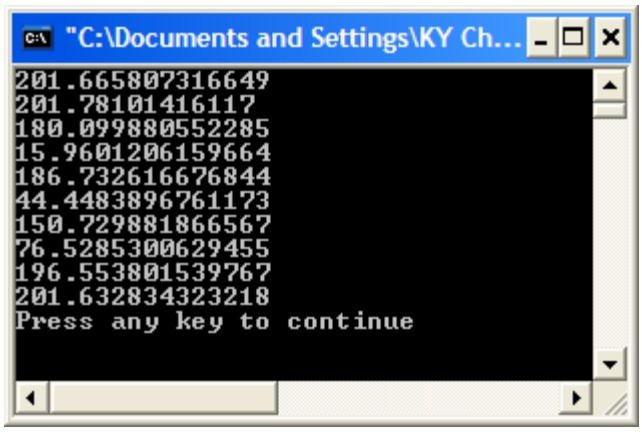
1. Open your previously saved project *Readlight* from exercise 5. Look for it in the *EmantVB2005\Readlight* folder If you cannot find the project, then build a new project from scratch.
2. Add the highlighted lines to the generated source code if you are building on the *Readlight* project. Add the following line if you are using the simulator

```
DAQ.Simulation = True
```

Program 8.1 Measure Light Intensity 10 times

```
Imports Emant
Module Module1

    Sub Main()
        Dim volt, lux As Double
        Dim i As Integer
        Dim DAQ As Emant300 = New Emant300

        DAQ.Open()
        For i = 0 To 9
            volt = DAQ.ReadAnalog(Emant300.AIN.AIN0, Emant300.AIN.COM)
            lux = 1333 * volt
            Console.WriteLine(lux)
            DAQ.Delay(1000)
        Next
        DAQ.Close()
    End Sub

End Module
```

3. Press **Ctrl+F5** to **Start without Debugging** to run the program. 10 Light Intensity measurements in Lux will be displayed at the rate of 1 measurement per second.



4. Press any key to return to the development environment.
5. Save your project by clicking on **File -> Save All**. You will be using this program in a later exercise.

For Loop

The For statement is of the form:

`For` variable **=** expression1 `To` expression2 [`Step` expression3]

statement1
statement2
..
`Next`

```
        For i = 0 To 9
            volt = DAQ.ReadAnalog(Emant300.AIN.AIN0, Emant300.AIN.COM)
            lux = 1333 * volt
            Console.WriteLine(lux)
            DAQ.Delay(1000)
        Next
```

In the above for loop, the light intensity is measured and displayed 10 times.

- expression1: this is executed when the loop is entered. `i` is assigned `0` in the above example
- expression2: if this is evaluated true, the statements in the loop are executed otherwise the loop is exited. Thus as long as `i` is less or equal to `9`, light intensity is measured
- expression3: this is executed each time after the statements in the loop are executed. `i` is incremented `Step`. If omitted, the incremented by 1.

Delay

In the `Emant300` class, we have a method called `Delay`. The program is delayed by the time in msec. The following line delays the program by 1000 ms before continuing with the next statement.

```
      DAQ.Delay(1000)
```

Delay in msecs

Traffic Lights using LEDs (optional exercise)

1. Goto to *EmantVB2005* folder and run the *Traffic.exe* program. When the program runs, the Green, Yellow and Red LEDs turn on in sequence like the traffic lights.
2. Using the for loop, write your own Visual Basic program that simulate the traffic lights.

Hint:

- use a for loop
- turn on LED (0)
- delay 1 second
- turn off LED (0)
- repeat steps 2-4 for LED (1,2)

End of Exercise 8

Exercise 9 – Digital Input

Objective

- Digital Input

In a car, an alarm light is turned on whenever a door is opened. This is implemented by using a switch to indicate whether is door is opened or closed. The Light Application Adaptor has one switch. We will modify our earlier exercise 7 replacing the light sensor with the switch to turn on and off the LED. We will also practice using the For Loop we learnt in the previous earlier.

1. Create a new Visual Basic console project. Name the project *ReadSwitch*
2. Add the highlighted lines to the generated source code. If you are using the simulator, don't forget to add the additional line of code.

Program 9.1 Read Switch

```
Imports Emant
Module Module1

    Sub Main()
        Dim i As Integer
        Dim swstate As Boolean
        Dim DAQ As Emant300 = New Emant300
        DAQ.Open()
        For i = 0 To 9
            swstate = DAQ.ReadDigitalBit(3)
            DAQ.WriteDigitalBit(0, swstate)
            Console.WriteLine(swstate)
            DAQ.Delay(1000)
        Next
        DAQ.Close()
    End Sub

End Module
```

3. In order to use the `Emant300` class, add the class library *Emant300.dll* to the references folder. Repeat steps 4 to 8 from exercise 5.
4. Click on **Debug-> Start** to run the program. The Console Window opens and then closes automatically after 10 seconds when the program ends. **Start without Debugging** is useful if the console displays some information as you are required to enter a key to close the window.
5. Press and hold the switch to turn the Green LED on. Release to turn the Green LED off.

Data Type Boolean

```
  bool swstate;
```

Boolean data types have 2 states, true or false. You assign a value as follows

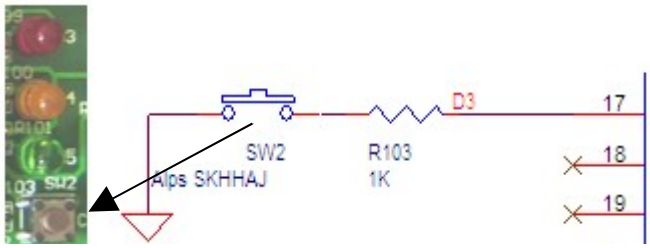
```
swstate = True
```

Read Digital Input

```
swstate = DAQ.ReadDigitalBit(3)
```

DIO Channel
0 – 2: LED (output)
3: Switch (input)

The `ReadDigitalBit` method returns a boolean value reflecting the switch status (true /false)



End of Exercise 9

Exercise 10 – Do Loop

Objective

- Do Loop

Although the For loop solves Sam's tedium of making repeated light intensity measurement, he needs to know beforehand the number of measurements to take. He wonders if Visual Basic will allow him to stop measurements when the switch is closed.

In this exercise, we will make use of the do loop to make repeated measurements every second until a switch is pressed.

1. Create a new Visual Basic project and name it *ReadLightdo*. Add the highlighted lines to the generated code. Don't forget to add *Emant300.dll* to the references and set the **Simulation** property if you are not using hardware.

Program 10.1 Measure Light Intensity Until Switch Pressed

```
Imports Emant
Module Module1

    Sub Main()
        Dim swstate As Boolean
        Dim volt, lux As Double
        Dim DAQ As Emant300 = New Emant300
        DAQ.Open()
        Do
            volt = DAQ.ReadAnalog(Emant300.AIN.AIN0, Emant300.AIN.COM)
            lux = 1333 * volt
            Console.WriteLine(lux)
            swstate = DAQ.ReadDigitalBit(3)
            DAQ.Delay(1000)
        Loop While swstate
        DAQ.Close()
    End Sub

End Module
```

2. Press **Ctrl+F5** to **Start without Debugging** to run the program. Light Intensity measurements in Lux will be displayed at the rate of 1 measurement per second. To stop the measurements, press this button.
3. Press any key to return to the development environment.

Do statement

The do statement conditionally executes the statement(s) in the loop one or more times.

```
Do
statement1;
statement2;
..
Loop Until boolean-expression
```

Do statements in Loop until the condition is true

```
Do
statement1;
statement2;
..
Loop While boolean-expression
```

Do statements in Loop while the condition is true

```
Do
  volt = DAQ.ReadAnalog(Emant300.AIN.AIN0, Emant300.AIN.COM)
  lux = 1333 * volt
  Console.WriteLine(lux)
  swstate = DAQ.ReadDigitalBit(3)
  DAQ.Delay(1000)
Loop While swstate
```

Our do statement is executed as follows:

- Control is transferred to the statements in the do loop. In our example, the program
    1. measures the voltage,
    2. convert to Lux,
    3. displays the value,
    4. delays 1 sec and
    5. read the switch state. If the switch is not pressed, `swstate` is **true**. If switch is pressed, `swstate` is **false**.
- When and if control reaches the end point of the statements in the loop, the boolean-expression is evaluated.
- If the boolean expression yields true, control is transferred to the beginning of the do statement. Otherwise, control is transferred to the end point of the do statement.

Pedestrian Crossing (optional exercise)

Modify the traffic lights program you created earlier into a pedestrian crossing, In a pedestrian crossing, the light stays green until the crossing switch is closed. The green light then turns off, and the yellow and red light turns on in sequence.

End of Exercise 10

Exercise 11 – Array

Objective

- Learn about Arrays

Sam has been asked to help out in a Greenhouse project. He was told that light intensity and duration are important for crop growth and development as photosynthesis uses light. He has the following information

- Low light causes plants to be long and spindly, have small leaves, bud blades, poor pollination and poor fruit quality.
- Photosynthesis is stopped at high light intensity depending on species.

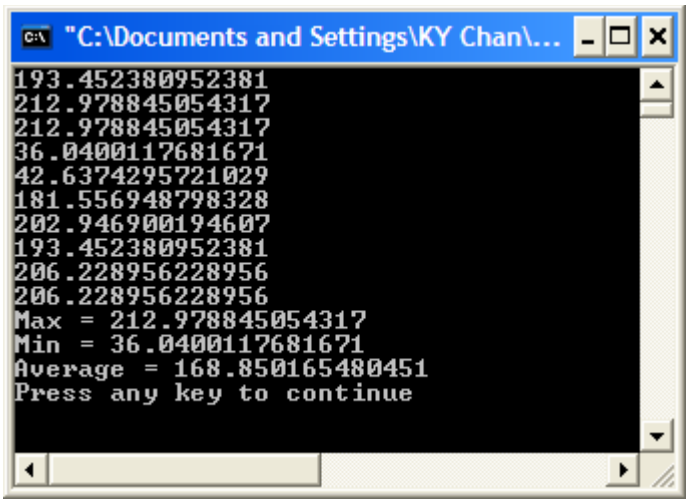
Sam decided that on top of taking light intensity measurements, he would like to know some basic statistics of the measurements like the maximum, minimum and average light intensities. One way to do this is to use arrays.

1. Open your previously saved project *Readlight* from exercise 8. Look for it in the *EmantVB2005\Readlight* folder If you cannot find the project, then build a new project from scratch.
2. Add the highlighted lines to the generated source code if you are building on the *Readlight* project
3. The `Math` Class that you used earlier does not include the basic statistics like maximum, minimum and average. We have provided the `Stat` Class which has these methods. The `Stat` Class is found in the class library *EmantCntrl.dll* which is located in the *EmantVB2005* folder. Add the class library *EmantCntrl.dll* to the references folder. To do this, repeat steps 4 to 8 from exercise 5.

Program 11.1 Basic Statistics

```
Imports Emant
Module Module1

    Sub Main()
        Dim i As Integer
        Dim volt, lux As Double
        Dim luxarray(10) As Double
        Dim s As Stat = New Stat
        Dim DAQ As Emant300 = New Emant300
        DAQ.Open()
        For i = 0 To 9
            volt = DAQ.ReadAnalog(Emant300.AIN.AIN0, Emant300.AIN.COM)
            lux = 1333 * volt
            Console.WriteLine(lux)
            luxarray(i) = lux
            DAQ.Delay(1000)
        Next
        Console.WriteLine("Max = {0}", s.Max(luxarray))
        Console.WriteLine("Min = {0}", s.Min(luxarray))
        Console.WriteLine("Ave = {0}", s.Average(luxarray))
        DAQ.Close()
    End Sub

End Module
```

11. Press **Ctrl+F5** to **Start without Debugging** to run the program. 10 Light Intensity measurements in Lux will be displayed at the rate of 1 measurement per second. At the end of the measurement, basic statistics like maximum, minimum and average light intensities will be displayed.



5. Press any key to return to the development environment.
6. Save your project by clicking on **File -> Save All**. You will be using this program in a later exercise.

Array

Array length

Array data type

```
Dim luxarray(10) As Double
```

An array is a data structure that contains a number of variables which are accessed through computed indices.

```
luxarray(i) = lux
```

The variables contained in an array, also called the elements of the array, are all of the same type, and this type is called the element type of the array. **luxarray elements are of the double data type**

An array has a rank which determines the number of indices associated with each array element. The rank of an array is also referred to as the dimensions of the array. An array with a rank of one is called a single-dimensional array. An array with a rank greater than one is called a multi-dimensional array. Specific sized multi-dimensional arrays are often referred to as two-dimensional arrays, three-dimensional arrays, and so on. **luxarray is a one dimensional array**

Each dimension of an array has an associated length which is an integral number greater than or equal to zero. The dimension lengths are not part of the type of the array, but rather are established when an instance of the array type is created at run-time. The length of a dimension determines the valid range of indices for that dimension: **luxarray array length is 10**

For a dimension of length N, indices can range from 0 to N – 1 inclusive. The total number of elements in an array is the product of the lengths of each dimension in the array. If one or more of the dimensions of an array have a length of zero, the array is said to be empty.

The element type of an array can be any type, including an array type.

Statistics

The `Stat` Class has methods for calculating Maximum, Minimum and Average. You pass to the methods an array and they return the respective values as a double.

```
Dim dmax, dmin, dave As Double
Dim s As Stat = New Stat

dmax = s.Max(luxarray)
dmin = s.Min(luxarray)
dave = s.Average(luxarray)
```

Console Formatting

```
Console.WriteLine("Max = {0}", s.Max(luxarray))
```

The console replaces {0}with the result of the method `s.Max` More information on the formatting can be found from the Microsoft website. Search using the keywords “Console.WriteLine MSDN”

End of Exercise 11

Exercise 12 – File IO

Objective

- File IO

Instead of writing to the console, a `StreamWriter` class is available for those who wish to write to file instead of displaying the result on the console.

1. Use **File -> Open -> Existing Project**, to open the *LightFile* project in the folder *EmantVB2005\LightFile* folder. The following program has been created for you.

Program 12.1 Log to File

```
Imports Emant
Imports System.IO
Module Module1

    Sub Main()
        Dim volt, lux As Double
        Dim i As Integer
        Dim DAQ As Emant300 = New Emant300
        Dim sw As StreamWriter = New StreamWriter("TestFile.txt")
        DAQ.Simulation = False
        DAQ.Open()
        For i = 0 To 9
            volt = DAQ.ReadAnalog(Emant300.AIN.AIN0, Emant300.AIN.COM)
            lux = 1333 * volt
            Console.WriteLine(lux)
            sw.WriteLine(lux)
            DAQ.Delay(1000)
        Next
        DAQ.Close()
        sw.Close()
    End Sub

End Module
```

2. Modify the `DAQ.Simulation` property if necessary
3. Click on **Debug-> Start without Debugging** to run the program. 10 Light Intensity measurements in Lux will be displayed and written to a text file named *TestFile.txt* at the rate of 1 measurement per second.
4. Press any key to return to the development environment.
5. You can find *TestFile.txt* in the *EmantVB2005\LightFile\bin\Debug* folder.

StreamWriter Class

```
Imports System.IO
```

Allows unqualified reference to the `StreamWriter` objects

```
Dim sw As StreamWriter = New StreamWriter("TestFile.txt")
```

Create a `StreamWriter` object called `sw` that writes to a text file called `TestFile.txt`

```
sw.WriteLine(lux)
```

`StreamWriter` method that writes the lux value to the file


```
sw.Close()
```
Close the `StreamWriter` object called `sw`

Information on the `StreamWriter` Class can be found at the Microsoft website. Search using the keywords **StreamWriter Class**

End of Exercise 12

Exercise 13 – Create a Windows Application

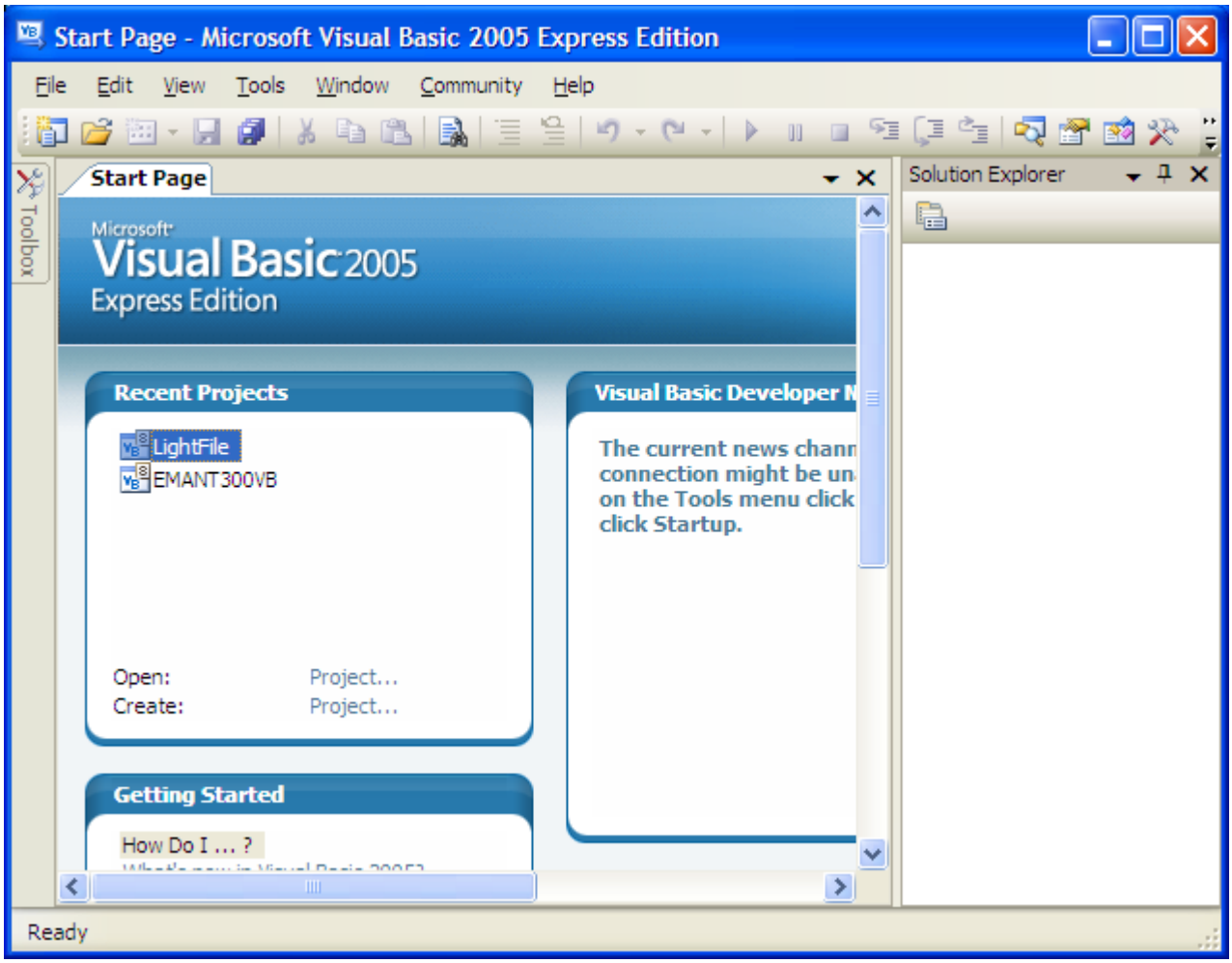
Objective

- Create a Windows Application
- Learn about Controls, Components and Properties
- Learn about Events

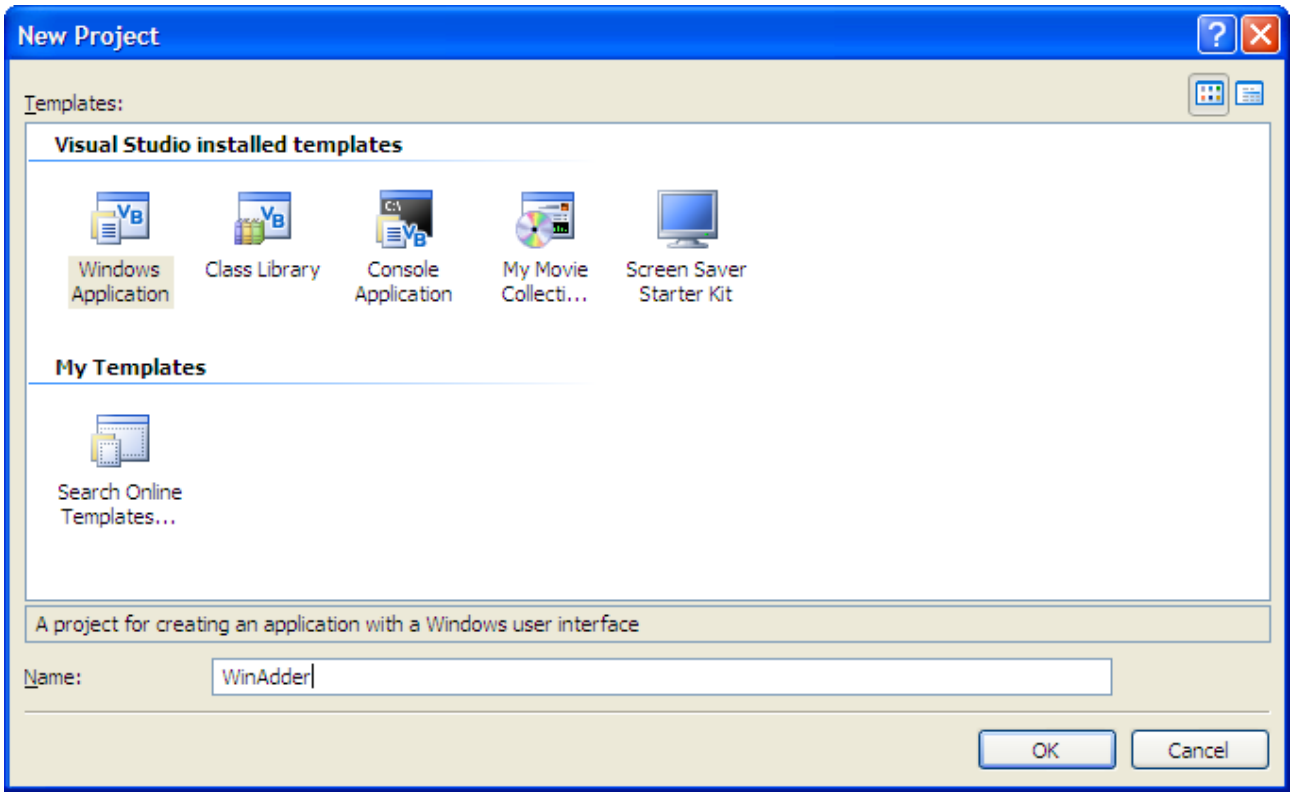
The Windows operating system provides a graphical user interface (GUI). Compared to the console programming, it makes the user's life (but not necessarily the programmer's life) easier. It forms the basis of most modern programs. Manipulating a mouse and clicking on screen elements is definitely more intuitive than a text based menu.

If you incorporate the Standard Graphical controls like text boxes, list boxes etc, your users will be familiar with their use. We will now create a Windows Application that adds two numbers and displays the result.

1. Select **File -> New Project**

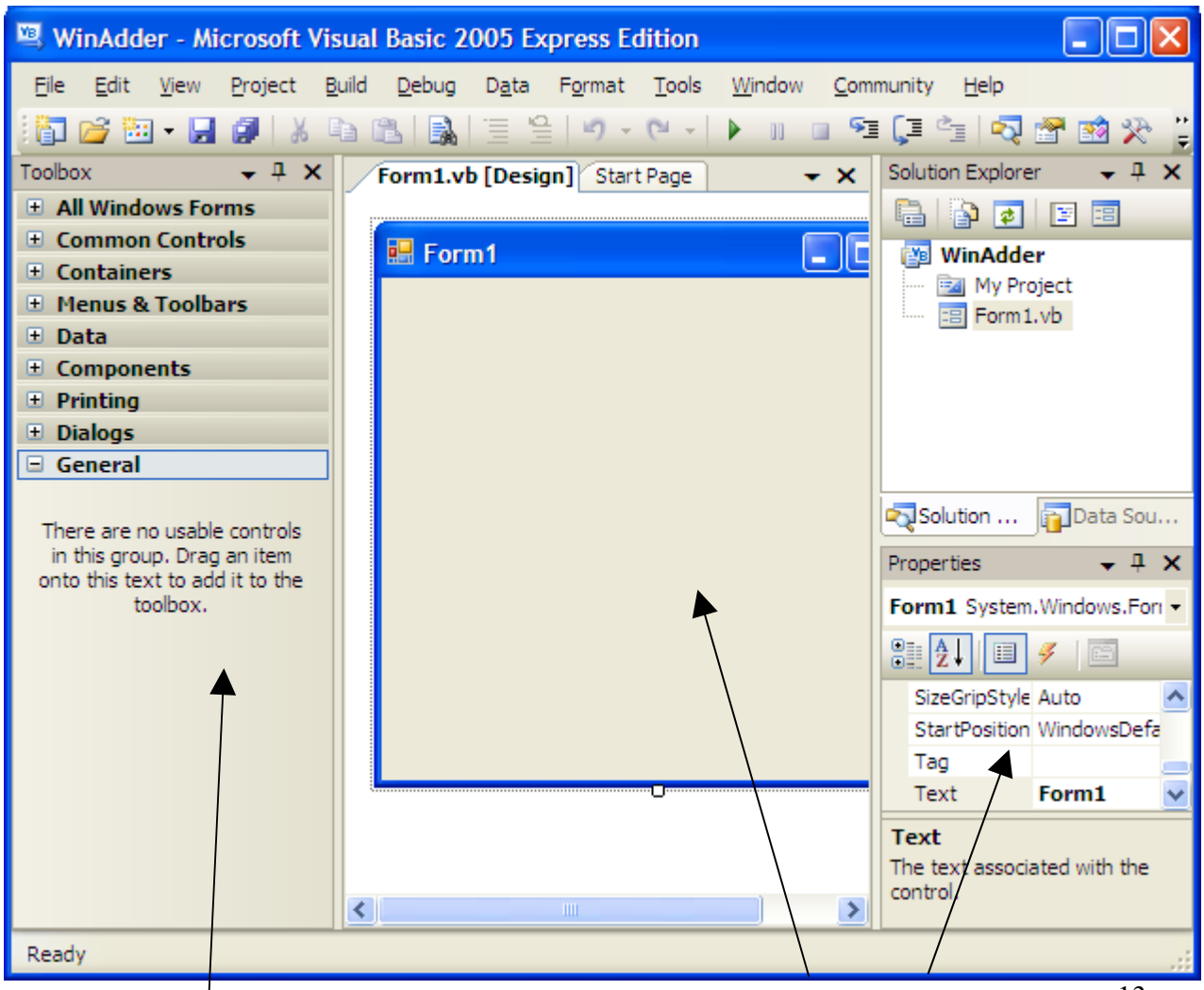


2. Select Visual Basic Projects and click on **Windows Application**



3. Call the project name *WinAdder*. Click OK

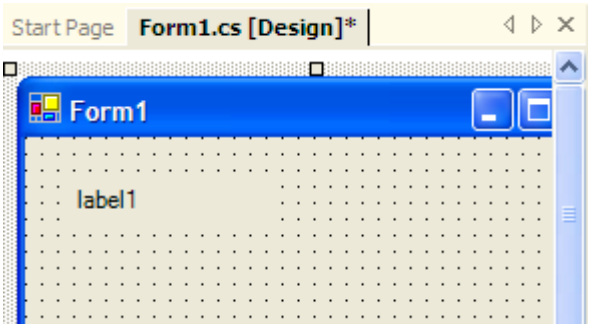
12.

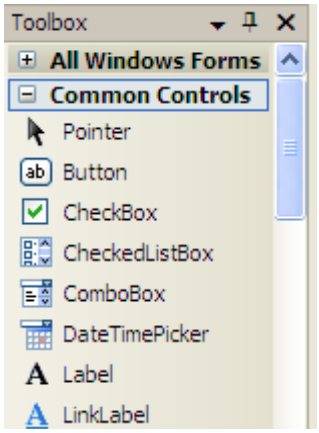


Your screen should be different from the console application programs you created earlier. You have the **Toolbox** and the **Form** besides the **source code**. The **Toolbox** contains the **controls** and **components** that you will place on the **Form** which is your user interface. Since we need to display the result of the addition, we will use of the `Label` control.

Label Control

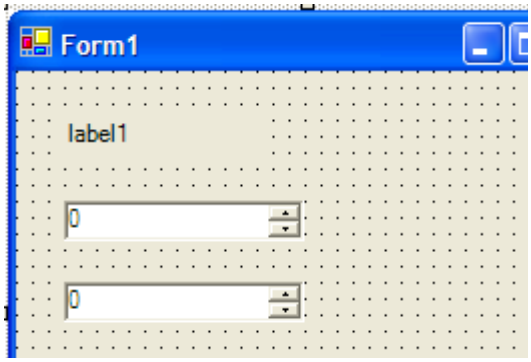
4. Locate the Toolbox and click on `Label`. The Toolbox has many tabs, `Label` is located in the **Common Controls** tab. While holding on the mouse left button, drag the **`Label`** to the form to place the `Label` to the Form.





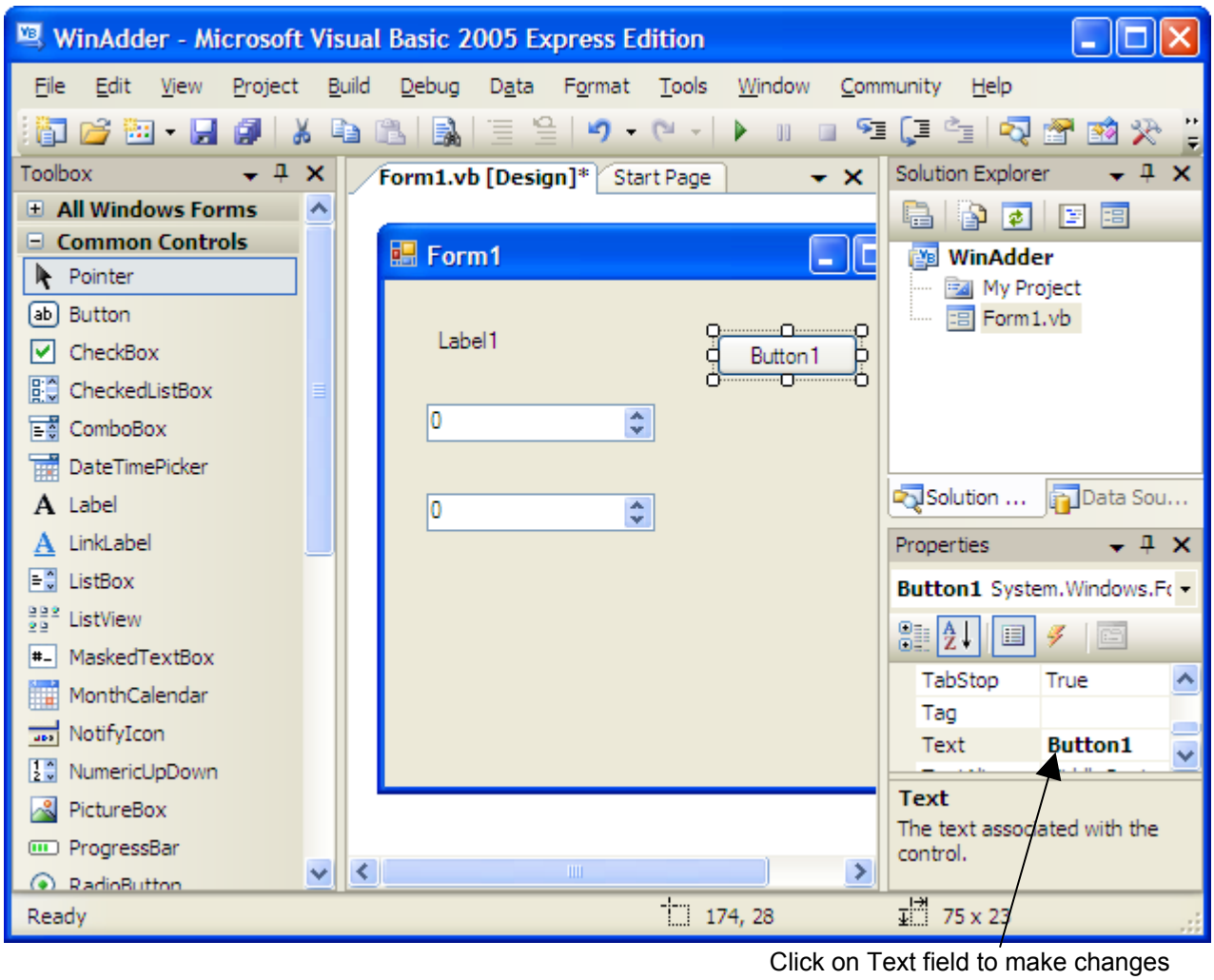
NumericUpDown Control

5. From Toolbox, locate and click on `NumericUpDown`. The Toolbox has many tabs, `NumericUpDown` is located in the **Windows Forms** tab. You may have to scroll down to find the control. While holding on the mouse left button, drag the `NumericUpDown` to the form to place the control to the Form. Since we are adding two numbers, we will need two of these controls. Repeat the operation to place the second control. Your Form is now like the one on the right.

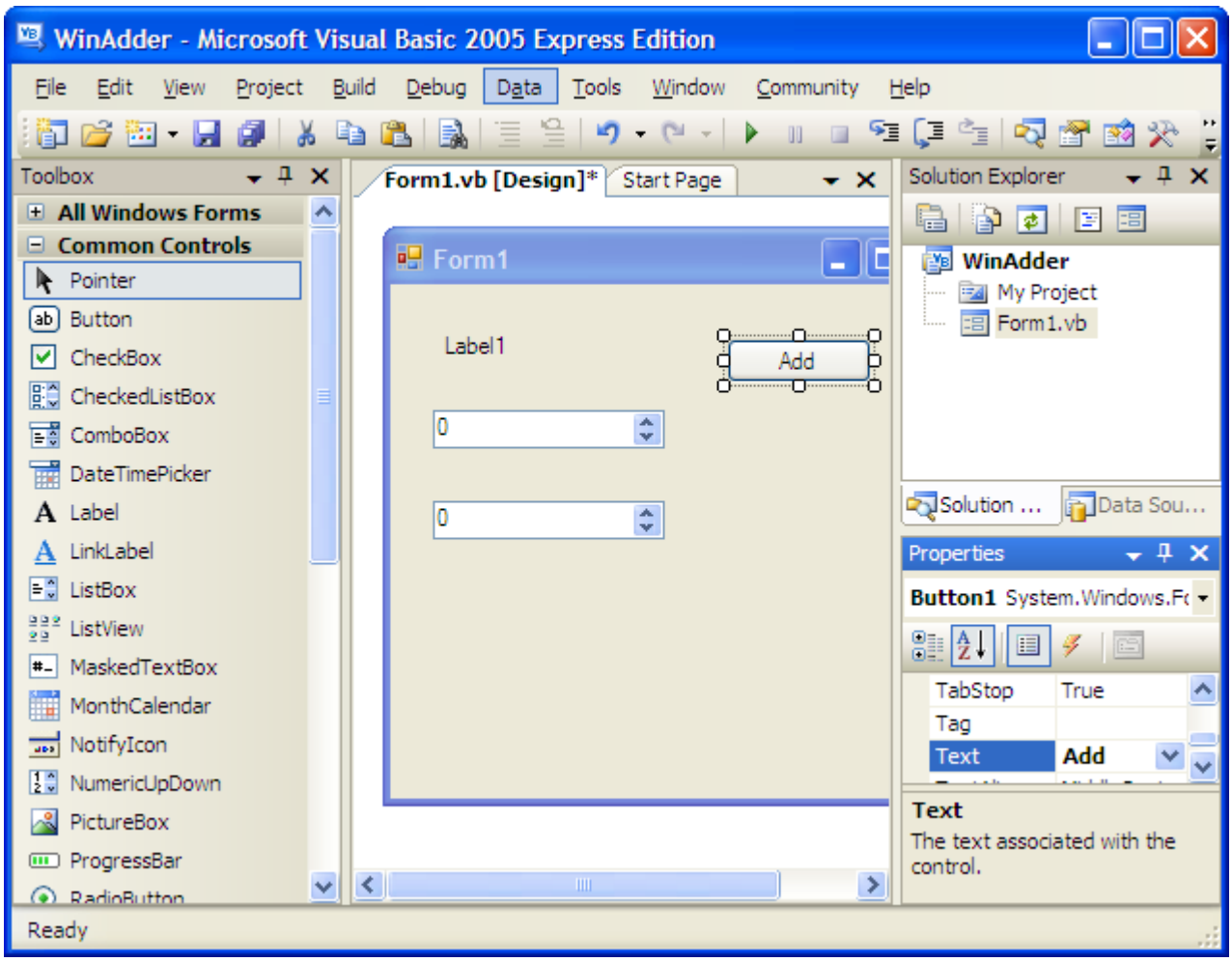


Button Control

6. From the same **Windows Forms** tab, find and place a `Button`. Your form now looks like below



7. All controls have **properties**. Let's make the `button` more meaningful to the user by changing the text on the button to **Add**. The `button` component has a property called `Text`. To change the button text during design, we can modify the **Text** field in the **Properties** Window. Click on the **Text** field, change the value to `Add`, hit the <enter> key and the `button` now displays `Add`. Your form now looks like the one below



Events

Design of a Windows Application is different from the console application. The Windows program is one main loop that waits for the user or the system to do something. When the user or system does something, an event is triggered. Most of the code in Windows Application comprise responses to events.

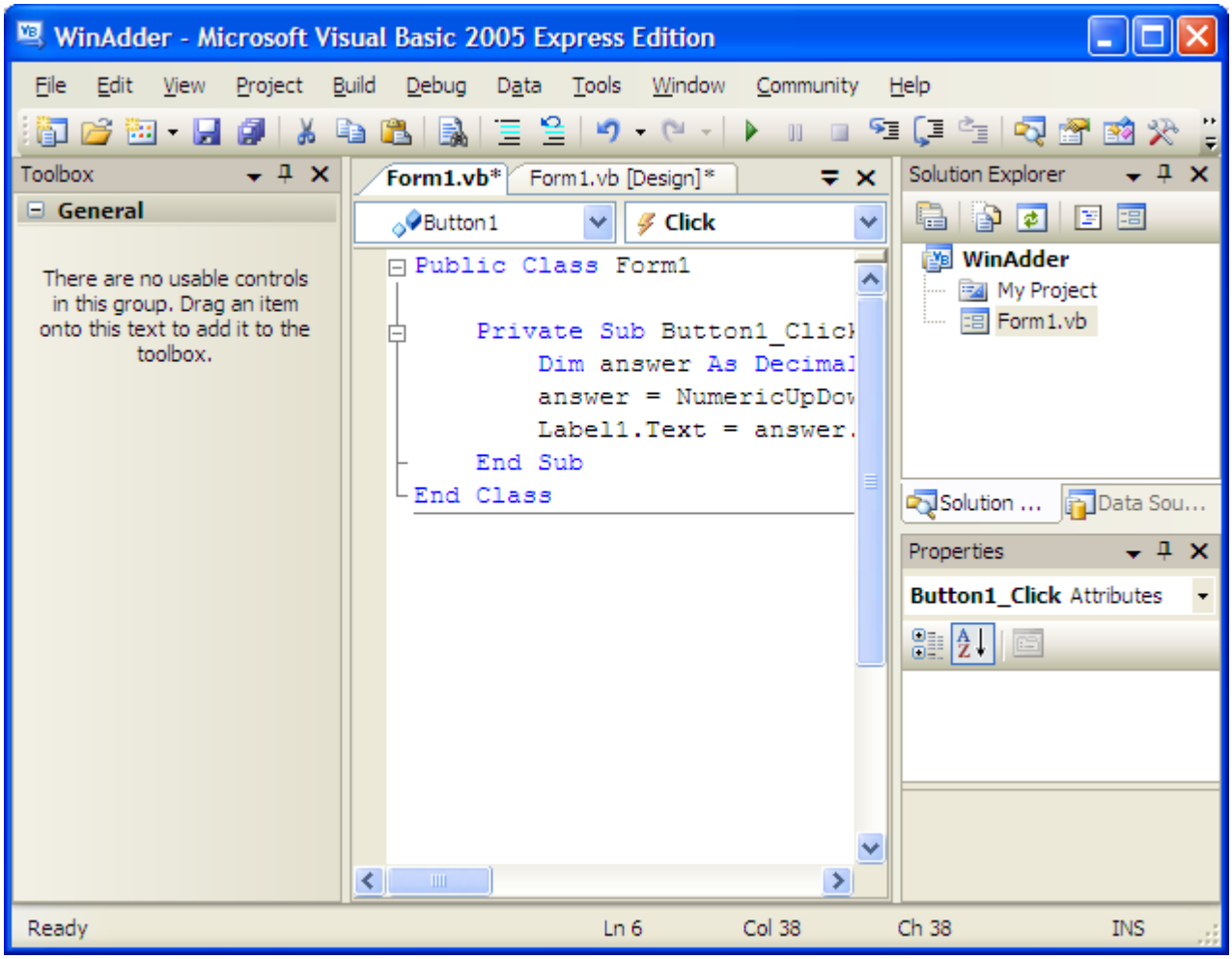
An Event in Windows can be activated by the user by clicking on a button. In our case, we want to add the two numbers.

14. Double click on the control (the button) will cause its default **event handler** to be created. An event handler is a method that is called when an event occurs. In this case, when the user clicks on the button, the following method is called and whatever code in the method will be executed.

```
Private Sub Button1_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles Button1.Click

End Sub
```

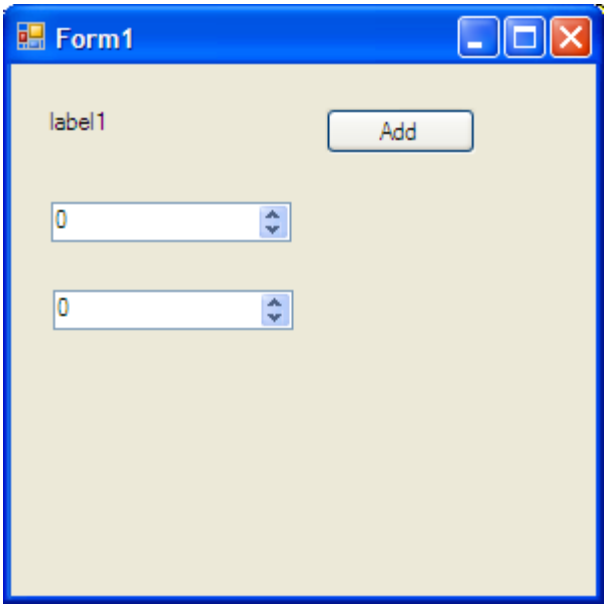
9. Type in the highlighted code to read the `NumericUpDown` controls' value , add them up and display on the form using `label1` . Your `Button1_Click` method should look like the following

```
Private Sub Button1_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles Button1.Click
    Dim answer As Decimal
    answer = NumericUpDown1.Value + NumericUpDown2.Value
    Label1.Text = answer.ToString
End Sub
```



Note that when you were typing in the code, a different window displays the code. The user interface window is shown with the extension **[Design]**

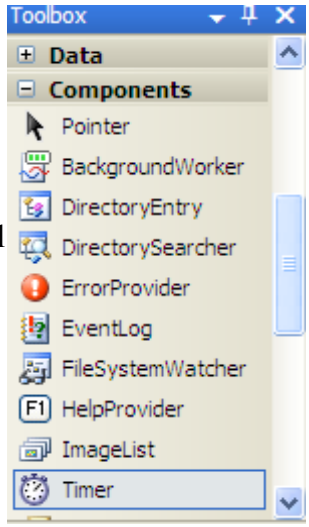
10. Click on **Debug-> Start** to run your first Windows program. If you did not have any typing errors, the program will run and the following window appears.



Change the values in the numeric controls and click the `Add` button. `Label1` will change to reflect the sum of the numbers.

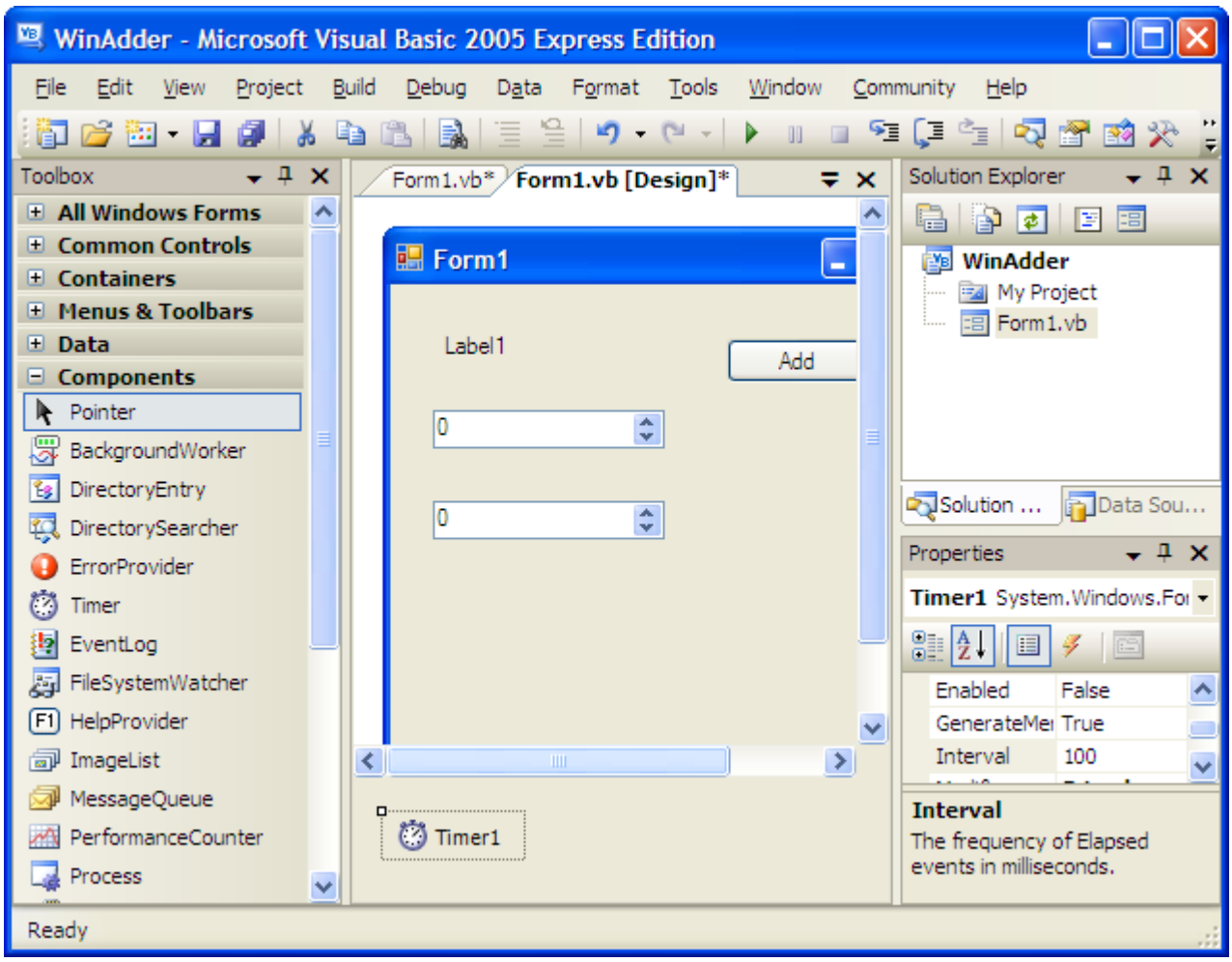
11. Click on  to return to the development environment.

12. Save your project by clicking on **File -> Save All**.



What if the user wants the answer to be updated when the inputs are changed rather than wait until the add button is clicked. There are several ways to do this. One way is to use the timer component.

13. In .NET jargon, a **control** is a **visual component** whereas a **component** is **non-visual**. Thus, the `Label`, `NumericUpDown` and `Button` are all controls whereas the `Timer` which we will use now is a component. A component does not appear (or is invisible) during runtime.
14. Goto the **Toolbox**, click on the **Components** tab to display the components as shown on the right
15. Click on `Timer`. While holding on the mouse left button, drag the `Timer` to the form and release the left button to get the component placed as shown in the figure above. **You must release the component on the form**. When you have successfully placed the `Timer` component, the **Form** looks as below. As a non-visual component, it resides in a component tray which is at the bottom of the form designer.



The `Timer` object is called `Timer1` and some of its default properties are

- `timer1.Enabled is false`
- `timer1.Interval is 100 ms`

16. On the `Timer1` properties window, set the **Enabled** property to **True**. Double click on the `Timer1` component to create the `Timer1`'s default event called Tick Type in the highlighted code to the **event handler** created

```
Private Sub Timer1_Tick(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Timer1.Tick
    Dim answer As Decimal
    answer = NumericUpDown1.Value + NumericUpDown2.Value
    Label1.Text = answer.ToString
End Sub
```

You will realize that this is the same code we type in earlier for the `Button1_Click` event. As the timer elapsed event is fired every 100ms, the answer is therefore recalculated and displayed every 100ms.

```
        Label1.Text = answer.ToString
```

In Exercise 4, you learnt about the `Convert` class in the `System` namespace. In this example, we convert a decimal to a string before displaying it using label1.

Click on **Debug-> Start to** run your modified Windows program. If you did not have any typing errors, the program will run. `Label1` changes to reflect the sum of the numbers. Change the values in the numeric controls and `Label1` will change to reflect the sum of the numbers.

17.Click on  to return to the development environment.

18.Save your project by clicking on **File -> Save All**.

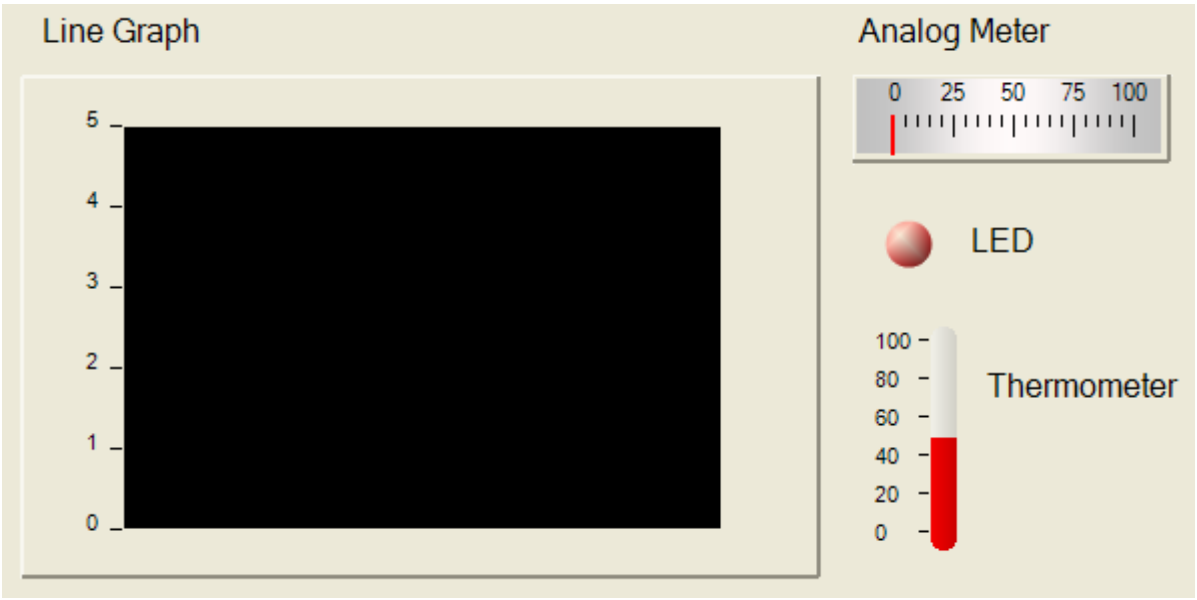
Exercise 14 - Install the EMANT300 Components and Instrument and Controls

Objective

- Learn to install third party controls to the Visual Basic IDE

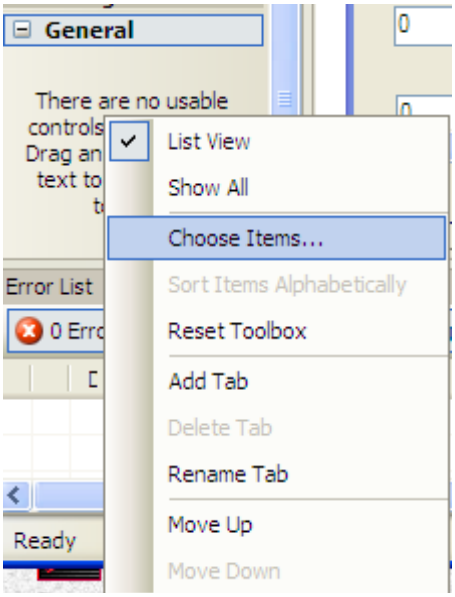
The standard controls provided are better suited for the Office Automation applications rather than Measurement Automation applications. In Engineering, we normally display data using charts, meters and LEDs. Fortunately, these controls are available from third party suppliers like 9Rays.Net. They cost from a few hundred US$ to several thousand US$.

For your exercise, we have included a FREE simple instrument control kit that contains the following controls

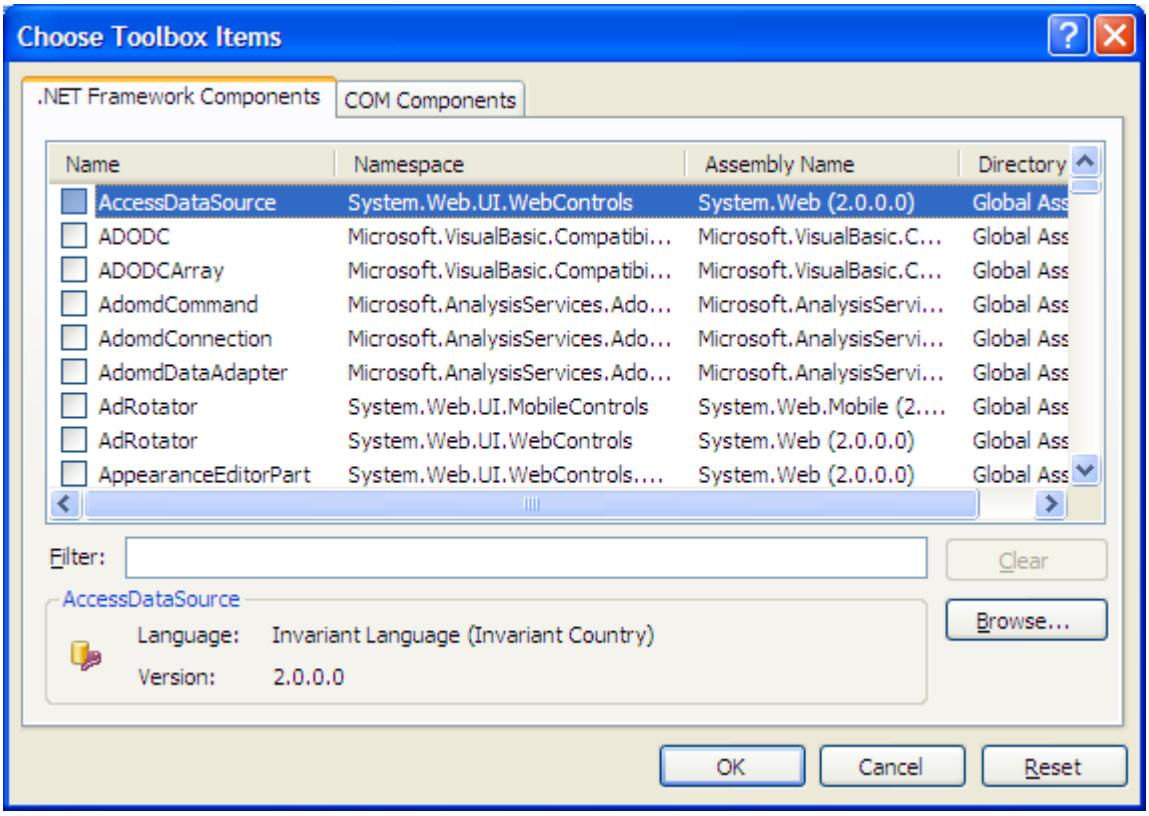


To integrate the **Emant Instrument Controls** in Visual Studio .NET:

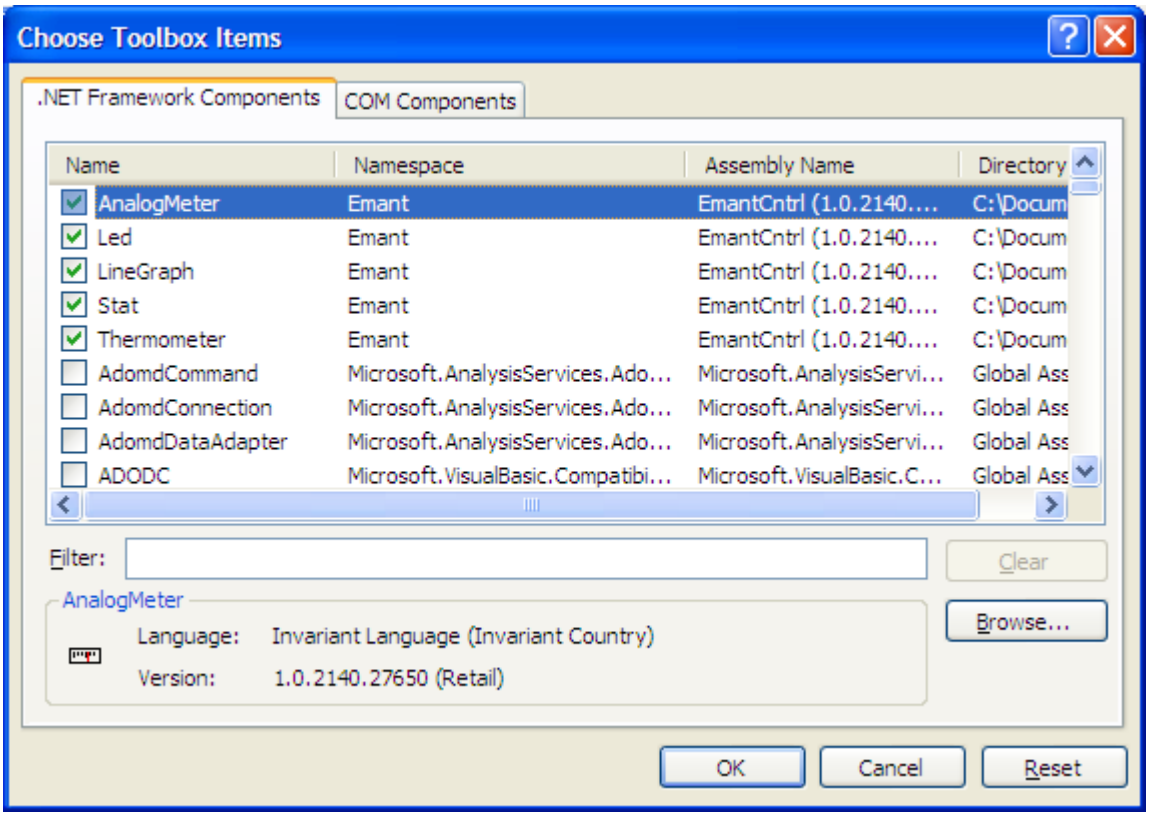
1. Select the **General** tab in the **Toolbox**
2. Right-click on the **Toolbox** background
3. Select **Choose Items...**
4. On the **Choose Toolbox Items Dialog**, select the **NET Framework Components** tab



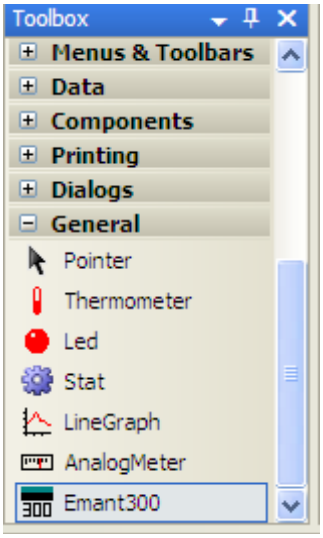
5. Press the **Browse** Button
6. Locate and select the *EmantCntrl.dll* assembly in the *EmantCS2005* folder



7. Click on the **Namespace** tab to sort the controls by Namespace to see the dialog as below. Click **OK**



8. If your controls are successfully installed, your **General** tab will display the controls. Note that you need to install your components only once. It will be available for subsequent projects until you remove them.
9. Repeat the steps for the `Emant300` component. The `Emant300` component is required to program the EMANT300 USB DAQ module. You have used it in your earlier exercises as the assembly *Emant300.dll*



10. See Appendix B & C for the properties associated with these controls.

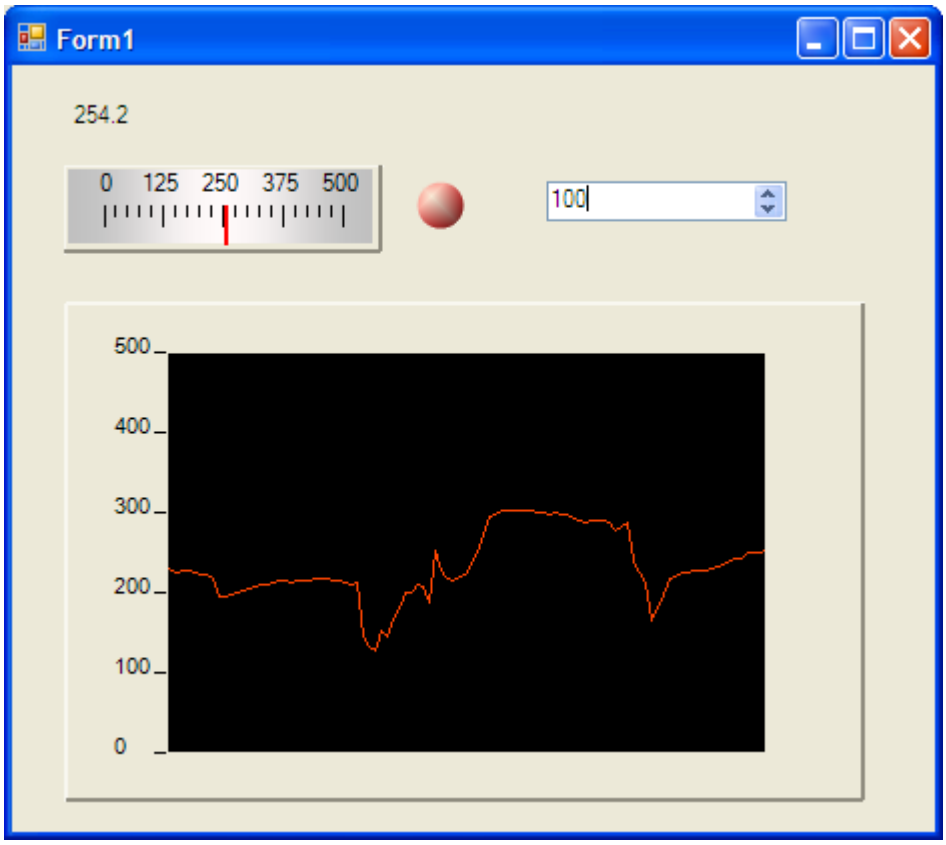
End of Exercise 14

Exercise 15 – Create an Instrument User Interface

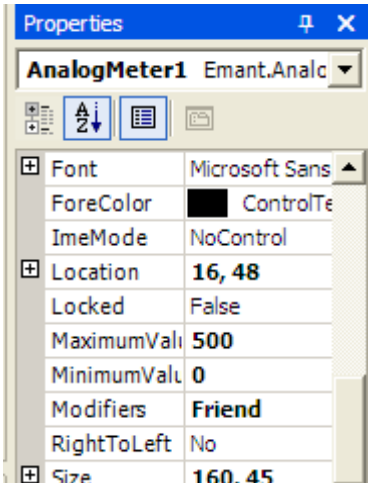
Objective

- Use third party controls
- Use Emant300 component

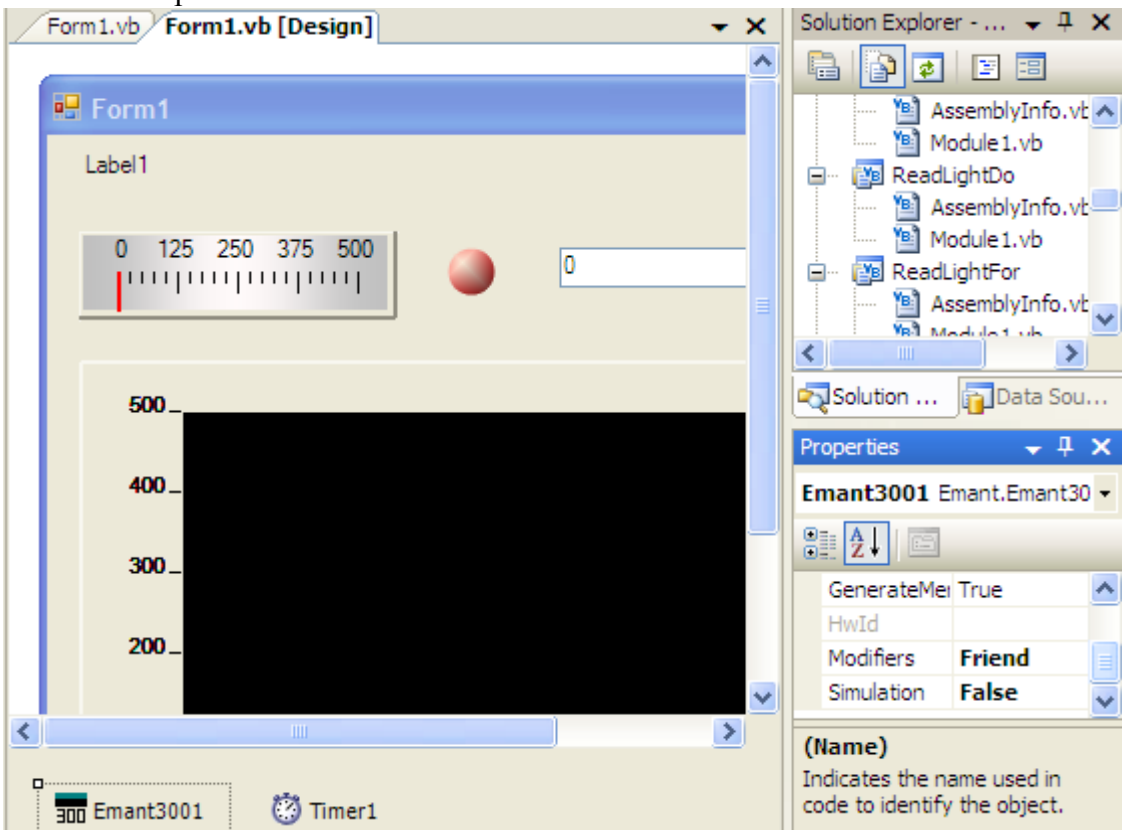
In this final exercise, you will build a computer based light logger with a modern GUI.



1. Select **File -> New Project**
2. Select Visual Visual Basic Projects and click on Windows Application
3. Call the project name *WinLogger*. Store your projects in the *EmantVB2005* folder. Click on the OK button to create the project.
4. Add the `AnalogMeter, LED, LineGraph` & `NumericUpDown` controls shown.
5. From the **Toolbox** and **General** tab, click and drag the **AnalogMeter** to place it on the form.
6. On its **Properties** window, set the **MaximumValue** to **500**.
7. Click and drag the `LineGraph` to place it on the form.
8. On its **Properties** window, set the **YMax** to **500**.
9. Click and drag the `LED` to place it on the form.
10. Add the `NumericUpDown` control you learnt from the previous exercise. On its **Properties** window, set the **Maximum** to **500**
11. Add the `Label` control you learnt from the previous exercise.
12. Now add the two components `Timer` and `Emant300`. Click on `Timer`. While holding on the mouse left button, drag the `Timer` to the form and release the left button to get the component placed as shown in the figure above. **You must release the component on the form**. The `Timer` component, as a non-visual component,



resides in a component tray which is at the bottom of the form designer. Repeat for the `Emant300` component. The Form should look as below.



13. If you select the `emant3001` object (an instance of the `Emant300` component), you can see its properties. If you are using the simulator, then you should change its **Simulation** property to true.
14. Double click on `Timer1` component will cause its event handler to be created. Add the highlighted code below to the `timer1_Tick` event handler.

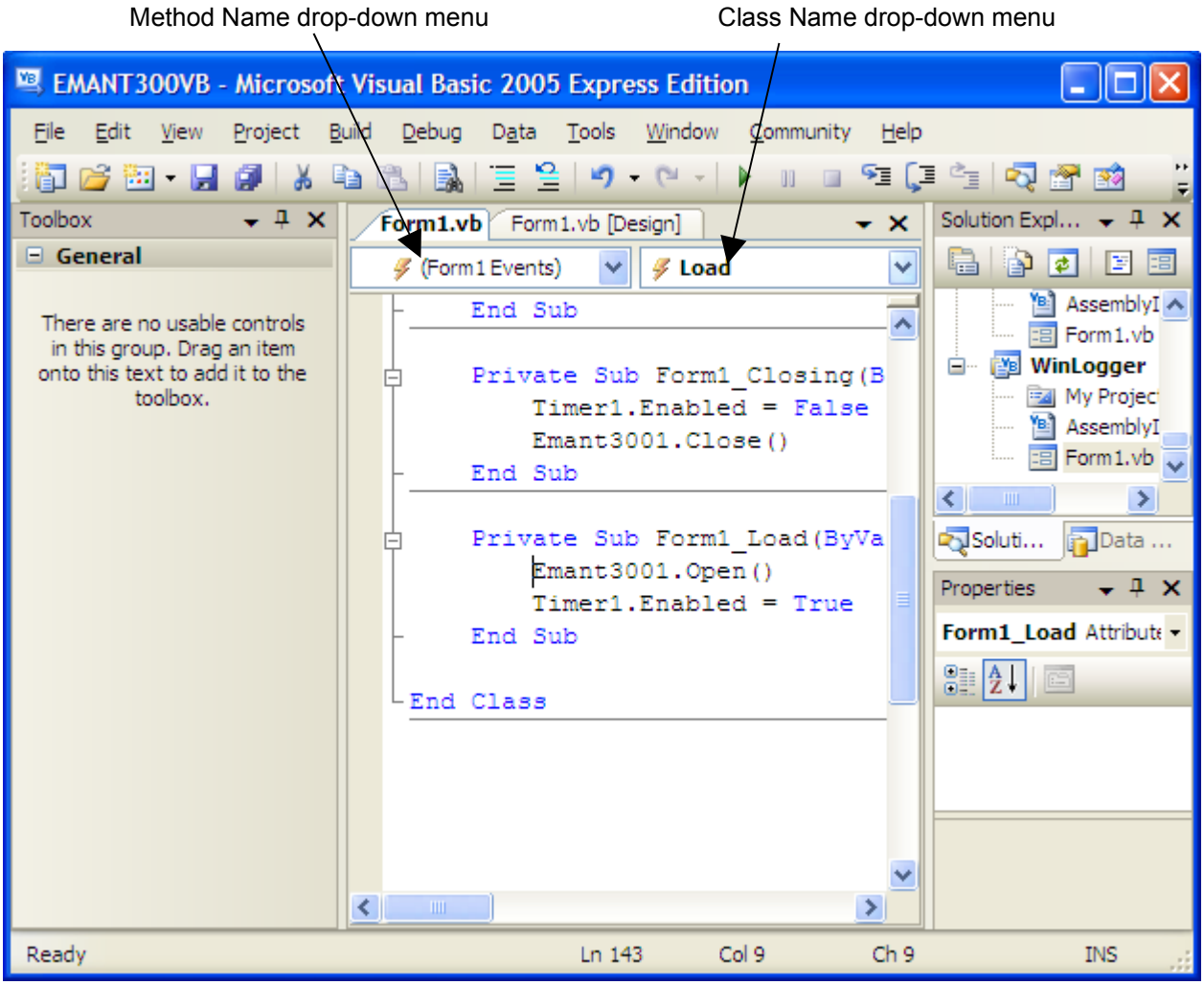
Program 15.1 Modify Timer Event Handler

```
Private Sub Timer1_Tick(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Timer1.Tick
  Dim volt, lux As Double
  volt = Emant3001.ReadAnalog(Emant.Emant300.AIN.AIN0, Emant.Emant300.AIN.COM)
  lux = 1333 * volt
  AnalogMeter1.Value = lux
  LineGraph1.Value = lux
  Led1.Value = lux > NumericUpDown1.Value
  Label1.Text = lux.ToString("0.0")
End Sub
```

In exercise 5, we learnt that we need to use the `Open` method to connect to the DAQ module and the `Close` method to close the DAQ connection before exiting the program. The `Form` Class has two events that you can use to automate the process. When the Form loads, there is a `Form.Load` event generated. Likewise when the user closes the Form, a `Form.Closing` is generated.

15.To create the `Form.Load` event,

- click on the **Code** tab to select the Code View
- select `Form1Events` from the **Class Name drop-down menu**.
- from the **Method Name drop-down menu**, select the `Load` event.
- Add the highlighted code below to the `Form.Load` event handler created.




```
Private Sub Form1_Load(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles MyBase.Load
  Emant3001.Open()
  Timer1.Enabled = True
End Sub
```

16.Repeat the above steps for the `Form Closing` Event. Add the highlighted code.

```
Private Sub Form1_Closing(ByVal sender As Object, _
ByVal e As System.ComponentModel.CancelEventArgs) Handles MyBase.Closing
  Timer1.Enabled = False
  Emant3001.Close()
End Sub
```

Assigning Values to the Controls

```
AnalogMeter1.Value = lux
```
assign the lux value to the AnalogMeter1

```
LineGraph1.Value = lux
```
assign the lux value to the LineGraph1

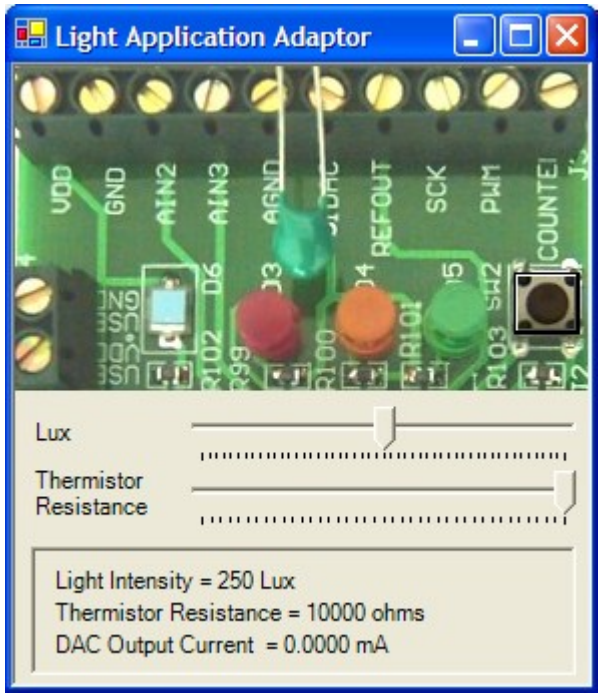
```
Led1.Value = lux > NumericUpDown1.Value
```
the Led1 state depends on whether the measured lux value exceeds the `NumericUpDown` setting

```
Label1.Text = lux.ToString("0.0")
```
In Exercise 4, you learnt about the `Convert` class in the `System` namespace. In this example, we convert a double to a string. The optional format "0.0" instructs the method to display the string with one decimal place.

17. Click on **Debug-> Start** to run your light measurement program. Cover or shine on the photodiode to observe the change in light intensity measured and how it is displayed on your interface. Modify the `NumericUpDown` value and observe the change in LED state.
18. Click on  to return to the development environment.
19. Save your project by clicking on **File -> Save All**.

End of Exercise 15

Appendix A – Using the Simulator



Learning to program in Visual Basic and DAQ, like most worthwhile endeavors, is inefficient and requires effort and time. The paradox of learning is that you learn the most when you make mistakes.

The simulator allows you to work on exercises even though the hardware is not available. Therefore you can complete your exercises outside the laboratory and use the DAQ hardware only to verify your error free programs.

To use the simulator, browse the *EmantVB2005* folder and look for the simulator program called *LightApp.exe* Click on the program to start the simulator.

Your simulator must be running before you run your own program.

- To change the analog inputs, adjust the sliders
- To operate the switch, click on the switch.
- The LEDs will turn on / off according to your program.

The following information is needed from exercise 5

When you are using the DAQ module, you need the `Emant300` class. To use the simulator just add the line `DAQ.Simulation = true` between creating the `Emant300` object and `Emant300.Open` method (add the highlighted line)

```
    Dim DAQ As Emant300 = New Emant300
    DAQ.Simulation = True
    DAQ.Open()
```

When you have access to the DAQ hardware, either comment out the `Simulation` line using `'` or set the property to `false` and your program will use the hardware

```
'DAQ.Simulation = True
```

or

```
  DAQ.Simulation = False;
```

If you have created your own DAQ solution using the EMANT300 hardware and you wish to create a simulator for your system, a `Sim300` component is available for you to do so. Please email support@emant.com for information.

Appendix B - Emant300 Class Reference

Represents the EMANT300, a low cost USB data acquisition module from EMANT PTE LTD. The USB DAQ Module measures analog input voltages, output analog current and perform digital input and outputs, PWM or Counter functions.

Example

The following example reads in the analog voltage connected to Analog Input 0 and the Common Analog Input.

```
Imports Emant
Module Module1

Sub Main()
  Dim volt As Double
  Dim DAQ As Emant300 = New Emant300
  DAQ.Open()
  volt = DAQ.ReadAnalog(Emant300.AIN.AIN0, Emant300.AIN.COM)
  Console.WriteLine(volt)
  DAQ.Close()
End Sub
```

Requirements

Namespace: Emant
Assembly: Emant300.dll, CommBase.dll, Sim300.dll, EmantUtil.dll

List of Members

Emant300 Constructor

Initializes a new instance of the Emant300 class.

```
public Emant300();
```

Example

Creates an instance called DAQ

```
  Dim DAQ As Emant300 = New Emant300
```

Emant300.Simulation Property

Gets or sets the boolean to determine to use the hardware or simulator. Must set Simulation before calling Open method.

```
public boolean Simulation {get; set;}
```

Property Value

true to connect to simulator, false to connect to hardware. Default is false.

Example

Use simulator

```
    Dim DAQ As Emant300 = New Emant300
    DAQ.Simulation = True
    DAQ.Open()
```

Emant300.HwId Property

Gets the Hardware Identity of the Emant300 DAQ module connected. The string gives the model and firmware version

```
public string HwId {get;}
```

Property Value

If the following string is returned, Emant300 000001, it indicates that the firmware version is 000001.

Emant300.CommPort Property

Gets the virtual Comm Port, the Emant300 DAQ module is connected to.

```
public string CommPort {get;}
```

Property Value

If the module is connected to the virtual serial comm port 2, then COM2 is returned. This information is useful when you need to connect several Emant300 to the same computer.

Emant300.Open Method

Opens the connection to the Emant300.

```
public boolean Open();
```

Return Value

Returns true if DAQ module is found.

Example

Finds and opens the connection to the DAQ module.

```
DAQ.Open()
```

Emant300.Open Method (bool, string)

Opens the connection to the Emant300.

```
public boolean Open( bool find, string port);
```

Parameters

Port The virtual serial comm. Port the Emant300 is connected to.

Return Value

Returns true if DAQ module is found.

Example

Do not find but open the DAQ module that is connected to COM2. Use this to connect to several Emant300 to the same computer.

```
DAQ.Open(false,"COM2")
```

If *find* is true, it will find the Emant300 connected with the lowest Virtual Comm Port Number.

Emant300.Close Method

Close the Emant300 connection. IMPORTANT - the Close method must be called before the program exits if Open was successful.

```
public void Close();
```

Example

```
DAQ.Close()
```

Emant300.Reset Method

Reset the Emant300.

```
public void Reset();
```

Emant300.ConfigDIO Method (Int32)

Sets the 8 bits of the DIO indicating the bit should be configured as input or output. Must set configuration before calling Open method

public Boolean ConfigDIO(Int32 *Value*)

Parameters

Value 0 to set bit as output and 1 to set bit as input.

Return Value

True if configuration is successful otherwise false.

Example

The light application adaptor has bit 0-2 as outputs and bit 3 as input.

```
Emant3001.ConfigDIO(&H08)
```

Emant300.ConfigPWMCounter Method (Emant300.PWMORCNT, Emant300.EVENTORTIMED, Int32, Int32)

Configures the PWM or Counter

public Boolean ConfigPWMCounter (Emant300.PWMORCNT *PWMOrCnt*, Emant300.EVENTORTIMED *EventOrTimed*, Int32 *MSInt*, Int32 *SetCount*)

Parameters

PWMOrCnt Specify if PWM or Counter in operation.
EventOrTimed Specify if Counter is Timed or Event Counting.
MSInt For Timed Counting, specifies the period in mSec to count pulses.
SetCount For Event Counting, used to clear or set count to specific start value.

Return Value

True if configuration is successful otherwise false.

Example

Strain gauges connected in the bridge configuration have very low output voltage. The following line configures the analog input range to -0.01 to 0.01 and sampling rate to 10Hz

```
Emant3001.ConfigAnalog(0.01,Emant.Emant300.POLARITY.Bipolar, 10)
```

Emant300.ConfigAnalog Method (Double, Emant300.POLARITY,Int32)

Configures the Analog Input for Input Range, Polarity and Sampling Rate

public Boolean ConfigAnalog(Double *InputLimit*, Emant300.POLARITY *Polarity*, Int32 *SampleFreq*)

Parameters

InputLimit The input range of the input signal.
Polarity Specify if input signal is unipolar or bipolar.
SampleFreq Sampling frequency. The default is 100 samples/sec.

Return Value

True if configuration is successful otherwise false.

Example

For Strain gauges, set analog input range to -0.01 to 0.01 and sampling rate to 10Hz

```
Emant3001.ConfigAnalog(0.01,Emant.Emant300.POLARITY.Bipolar, 10)
```

Emant300.ConfigAnalogAdvance Method (Emant300.POLARITY, Emant300.FILTER, Emant300.CALIBRATION, Boolean, Emant300.REF, Emant300.VREF, Boolean, Emant300.PGA, Int32, Int32, Int32)

Configures the Analog Input - for advanced users

public Boolean ConfigAnalogAdvance (Emant300.POLARITY *Polarity*, Emant300.FILTER *Filter*, Emant300.CALIBRATION *Calibration*, Boolean *BOD* ,Emant300.REF *Reference* , Emant300.VREF *VRef*, Boolean *Buffer*, Emant300.PGA *Gain*, Int32 *ACLK*, Int32 *Decimation* , Int32 *ODAC*)

Parameters

Polarity	Specify if input signal is unipolar or bipolar.
Filter	Set ADC Filter.
Calibration	Set ADC Calibration.
BOD	When the Burnout Detect is set, two current sources are enabled. The current source on the positive input channel sources approximately 2µA of current. The current source on the negative input channel sinks approximately 2µA. This allows for the detection of an open circuit (full-scale reading) or short circuit (small differential reading) on the selected input differential pair. Enabling the buffer is recommended when BOD is enabled. Set true to enable and false to disable
Reference	Specify internal or external reference.
VRef	Set the internal reference to either 1.25V or 2.5V
Buffer	Set true to enable input Buffer Amplifer, false to disable buffer.
Gain	Set PGA gain.
ACLK	Analog clock frequency
Decimation	Decimation Ratio
ODAC	The analog output from the PGA can be offset by up to half the full-scale input range of the PGA by using the ODAC register. The ODAC (Offset DAC) register is an 8-bit value; the MSB is the sign and the seven LSBs provide the magnitude of the offset.

The data rate for the ADC is determined by ACLK and Decimation Ratio. First, the ACLK register divides the system clock; that value is then divided by 64 to give us the modulation clock. The data output rate is determined by the Decimation Ratio. The eleven bits in the decimation ratio divide the modulation clock to calculate the data output rate.

$$Data\ Rate = \frac{22118400}{((ACLK+1)*64*Decimation)}$$

Return Value

True if configuration is successful otherwise false.

Emant300.ReadAnalog Method (Emant300.AIN, Emant300.AIN)

Measures the analog voltage of any two of the six plus one common analog inputs of the DAQ module. Measures the internal temperature sensing diode voltage if both inputs set to DIODE.

public double ReadAnalog(*Emant300.AIN PositiveInput, Emant300.AIN NegativeInput*);

Parameters

PositiveInput The positive analog input channel.
NegativeInput The negative analog input channel.

Return Value

The value of the voltage read.

Example

Measure voltage difference between analog input channel 0 and analog input common.

```
Dim volt As Double
volt = DAQ.ReadAnalog(Emant300.AIN.AIN0,Emant300.AIN.COM)
```

Emant300.ReadAnalogWaveform Method (Emant300.AIN, Emant300.AIN, Int32)

Measures the waveform of any two of the six plus one common analog inputs of the DAQ module. Measures the internal temperature sensing diode voltage if both inputs set to DIODE.

public double[] ReadAnalogWaveform (Emant300.AIN *PositiveInput* , Emant300.AIN *NegativeInput* , Int32 NumberOfSamples)

Parameters

PositiveInput The positive analog input channel.
NegativeInput The negative analog input channel.
NumberOfSamples Specify the number of samples

Return Value

The waveform read.

Example

Measure a 300 point waveform of the voltage difference between analog input channel 0 and analog input common.

```
Dim mywave(300) As Double
mywave = emant3001.ReadAnalogWaveform(Emant300.AIN.AIN3,Emant300.AIN.COM,300)
```

Emant300.WriteAnalog Method (double)

Set the analog output current of the DAQ module.

public void WriteAnalog(double *mA*)

Parameters

mA The analog current value to be output (0 to 1mA).

Example

Set Analog Output Current to 0.1 mA

```
DAQ.WriteAnalog(0.1)
```

Emant300.ReadDigitalBit (Int)

Reads in the state of the Digital Input of the DAQ module.

public boolean ReadDigital(int32 *channel*)

Parameters

channel The digital channel to read.

Return Value

The return bool represents the state of the digital input.

Example

Read the state of the digital bit 4.

```
Dim swstate as Boolean
swstate = DAQ.ReadDigitalBit(4)
```

Emant300.ReadDigitalPort

Reads in the state of the Digital Input of the DAQ module.

public int32 ReadDigital()

Return Value

The return int32 represents the 8 bits of the digital port.

Emant300.WriteDigitalBit Method (int, bool)

Set state of the Digital Ouput of the DAQ module. Does not affect bits configured as Digital Inputs.

public boolean WriteDigital(int32 *channel*, boolean *DAQDO*)

Parameters

channel The digital channel to set.
DAQDO The state to be set.

Example

Set the output of bit 0 of the DIO to high.

```
DAQ.WriteDigitalBit(0, True)
```

Emant300.WriteDigitalPort Method (int)

Set state of the Digital Ouput of the DAQ module. Does not affect bits configured as Digital Inputs.

public boolean WriteDigitalPort(int32 *value*)

Parameters

value 8 bit value of the output.

Emant300.ReadCounter (out Double)

Reads the counter value.

```
public Int32 ReadCounter(out Double Period)
```

Return Value

The return int32 represents the count.
Period returns the period if counter is configured for Timed Counting

Example

Read the count and period of the input digital waveform.

```
Dim Period, temp As Double
temp = Emant3001.ReadCounter(out Period);
```

Emant300.WritePWM (Double, Double)

Reads the counter value.

```
public Boolean WritePWM (Double Period , Double DutyCycle )
```

Parameters

Period Period of the PWM in uS (100 to 35000).
DutyCycle The duty cycle (0 to 100%).

Example

Set up a 10ms period (100 Hz) , 40% duty cycle PWM.

```
Emant3001.WritePWM(10000, 40);
```

Emant300.AIN Enumeration

Specifies the inputs of the ReadAnalog method.

Members

Member name	Description
AIN0	AIN0 – Analog Input 0
AIN1	AIN1 – Analog Input 1
AIN2	AIN2 – Analog Input 2
AIN3	AIN3 – Analog Input 3
AIN4	AIN4 – Analog Input 4
AIN5	AIN5 – Analog Input 5
COM	AINCOM – Common Analog Input
DIODE	DIODE – Temperature Sensing Diode

Emant300.VREF Enumeration

Specifies the internal reference voltage.

Members

Member name	Description
V2_5	Sets internal reference voltage to 2.5V
V1_5	Sets internal reference voltage to 1.5V

Emant300.POLARITY Enumeration

Specifies the polarity of the input voltage.

Members

Member name	Description
Unipolar	The input voltage range is unipolar (example 0 to 2.5V)
Bipolar	The input voltage range is bipolar (example -2.5V to 2.5V)

Emant300.FILTER Enumeration

At the ouput of the ADC is a Digital Filter. The Digital Filter can use either the Fast Settling, Sinc2, or Sinc3 filter. In addition, the Auto mode changes the Sinc filter after the input channel or PGA changed. When switching to a new channel, it will use the Fast Settling filter. It will then use the Sinc2 followed by the Sinc3 filter to improve noise performance.

Members

Member name	Description
Auto	Auto mode

Member name	Description
Fast_Settling	Fast Settling
Sinc_2	Sinc2
Sinc_3	Sinc3

Emant300.CALIBRATION Enumeration

The offset and gain errors in Emant300, or the complete system, can be reduced with calibration. Each calibration process takes seven acquisition clock cycles to complete. Therefore, it takes 14 clock cycles to complete both an offset and gain calibration..

Members

Member name	Description
No_Cal	No Calibration
Self_Cal_Offset_Gain	Self Calibrate Offset and Gain
Self_Cal_Offset	Self Calibrate Offset Only
Self_Cal_Gain	Self Calibrate Gain Only
Sys_Cal_Offset	System Calibrate Offset Only
Sys_Cal_Gain	System Calibrate Gain Only

Emant300.REF Enumeration

The Emant300 can use either an internal or external voltage reference.

Members

Member name	Description
Internal	Internal voltage reference
External	External voltage reference

Emant300.PGA Enumeration

Specifies the gain of the Programmable Gain Amplifier.

Members

Member name	Description
G1	Sets Gain to 1
G2	Sets Gain to 2
G4	Sets Gain to 4
G8	Sets Gain to 8
G16	Sets Gain to 16
G32	Sets Gain to 32
G64	Sets Gain to 64
G128	Sets Gain to 128

Emant300.PWMORCNT Enumeration

Use the Counter or the PWM.

Members

Member name	Description
Count	Use Counter
PWM	Use PWM

Emant300.EVENTORTIMED Enumeration

The Counter can be configured to either measure frequency/period of a digital waveform or count events.

Members

Member name	Description
Timed	Measure Frequency / Period
Event	Count Events

Appendix C - Emant Instrument Controls Kit Class Reference

AnalogMeter Class

Property	Data Type	Description
Value	Double	Assign meter reading
MaximumValue	Double	Set meter maximum limit
MinimumValue	Double	Set meter minimum limit

```
AnalogMeter1.Value = 2.5
```

LED Class

Property	Data Type	Description
Value	Boolean	Assign LED state
ColorOn	Color	Color of LED when it is on
ColorOff	Color	Color of LED when it is off

```
Led1.Value = True
Led1.ColorOn = Color.Yellow
```

Thermometer Class

Property	Data Type	Description
Value	Double	Assign thermometer reading
Maximum	Double	Set thermometer maximum limit
Minimum	Double	Set thermometer minimum limit

```
Thermometer1.Value = 25.0
```

LineGraph Class

Property	Data Type	Description
Value	Double	Assign LineGraph reading
YMax	Double	Set Y Axis maximum limit
YMin	Double	Set Y Axis minimum limit
XMax	Int32	Set number of points on the X axis

```
LineGraph1.Value = 2.0
```