Quick Learning of C# .Net for Students Who Already Know C or Java

By Prof . Taek Kwon
Department of Electrical and Computer Engineering
University of Minnesota Duluth

This short note is to provide students a quick learning path and also a reference for programming in Microsoft C#. Since most students already had programming experiences in c, java, or other programming languages, C# can be learned by simple examples and lookup lists, which is the intent of this note.

I will regularly update this note to make it more readable as time permits.

Last Updated: 2/21/ 2011

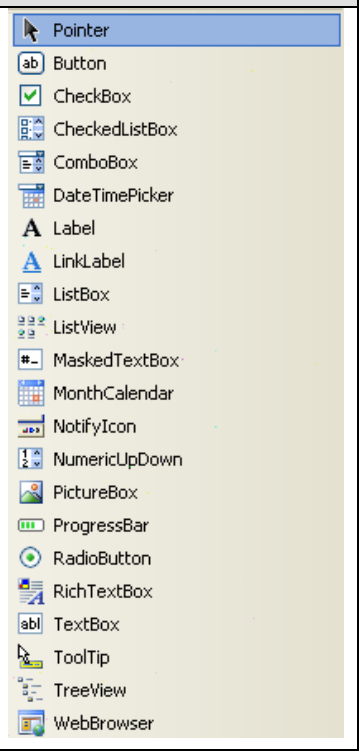
Table of Contents

1. Data Types and Basics

1.0 Windows Control Prefix Convention

For easy identification of Windows form controls, use of "a prefix + function name" is recommended for all control names. Whenever a control is placed on the form, you should change the (name) property to follow this convention, instead of using the default name. For example, after an Exit button is created, its (name) property should be changed to **btnExit**, which clearly indicates that it is an Exit button. This makes the code much more meaningful and readable than the Windows default name **Button1**. Below summarizes the prefix conventions for windows controls.

Windows Name	Prefix
Form	frm
Button	btn
CheckBox	chk
CheckedListBox	clst
ComboBox	cbo
Label	lbl
ListBox	lst
ListView	lvw
DataGrid	dgr
PictureBox	pic
Textbox	txt
TreeView	trv
Menu	mnu
Timer	tmr
OpenFileDialog	opf
SaveFileDialog	svf
FolderBrowserDialog	fbr
ColorDialog	cld
FontDialog	fnd
RadioButton	rdo



1.1 First Few Lines

Name spaces are declared first. The followings are the frequently included in the name spaces.

```
using System;
using System.Diagnostics;
using System.Globalization;
using System.Runtime.InteropServices;
using System.Security;
using System.Threading;
using System.Windows.Forms;
using System.Net;           // for all network programming
```

```
using System.Text;          //for binary array to ASCII or vice versa conversion
routines
using System.IO;             //for file operations such as stream
using System.Math;         //for math functions such as sin, cos, log
```

If you start your project with windows form, the following name spaces are includes as default.

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

1.2 Data Types

Integer Types:

sbyte	1 byte, System.SByte
short	2 bytes, System.Int16
int	4 bytes, System.Int32
long	8 bytes, System.Int64
byte	1 byte, unsigned, System.Byte
ushort	2 bytes, unsigned, System.UInt16
uint	4 bytes, unsigned, System.UInt32
ulong	8 bytes, unsigned, System.UInt64
char	2 bytes, Unicode, System.Char

Floating-Point Types

float	4 bytes, System.Single
double	8 bytes, System.Double

Bool Type

bool	1byte/2byte, System.Boolean

String Type

string	20bytes minimum, System.String

You can declare multiple variables of the same type in one line or different types by separating each by comma.

```
float x, y, z;
float x=1, y=2, z=3;  //with initialization
in i,j,k;
string s1 = “String type”;

float x  = 100.5f;  //tailer f indicates a floating point number
```

Hexadecimal assignment.

```
int flag = 0xffa0;  // prfix 0x is appended for a Hex number specification
```

1.2.1 Type Conversions

To convert types, use System.Convert class. Here are examples.

```
//convert string “12.5” to a double
double dd = Convert.ToDouble(“12.5”);

// convert the double to int
int ii = Convert.ToInt32(dd);

//convert from a binary string to a int
int num = Convert.ToInt32(“110110101”, 2);
// convert from an octal string
int num = Convert.ToInt32(“767676”, 8);
// convert from a hex string
int num = Convert.ToInt32(“AB78DF”, 16);
```

Convert to binary or hex string from a number.

```
string txt = Convert.ToString(27, 2);  // “11011”
string txt = Convert.ToString(255, 16); // “FF”
```

1.2.2 Constant Declaration

Constants can be declared using the “const” modifier.

```
public const double x = 1.0, y = 2.0, z = 3.0;

public const float myPi = 3.14; //Declare myPi as a constant 3.14.
```

1.2.3 Array Declaration

If you know the number of elements, a fixed array can be declared.

```
int[] vals = new int[4];  //declares 4 elements vals(0), vals(1),  vals(2),  vals(3)
```

or you may use two lines of code for the same thing, i.e.,

```
int[] vals;
vals = new int[4];
```

Arrays are initialized using curly a bracket {}.

```
int[] vals = new int[] {0, 1, 2, 3};
```

Multidimensional arrays can be created using [,,,] form.

```
int[,] numbers = new int[,] { {1, 2}, {3, 4}, {5, 6} };
numbers[1,1]=7  //assignment of an element
```

Arrays have a length property that can be used for for- loops.

```
for(int i = 0;  i < vals.Length;  i++) ;
```

For getting the length of multidimensional array, use the GetLength property.

```
for(int i = 0; I < numbers.GetLength(1); i++) ;  //Gets the inner dimension length
```

Regarding the inner/outer dimension definition, the left side is the outer, so the outermost dimension is always 0.

```
numbers[outer, inner]

int [,,] a = new int [2,5,7]
a.GetLength(0)        //returns 2
a.GetLength(1)        //returns 5
a.GetLength(2)        //returns 7
```

1.2.4. Special Characters

Most commonly used escape sequence characters in C# are

Char	Meaning	Value
\’	Single quote	0x27
\’’	Double quote	0x22
\\	Backslash	0x5C
\0	Null	0x00
\a	Alert	0x07
\b	Backspace	0x08
\f	Form feed	0x0C
\n	New line	0x0A
\r	Carriage return	0x0D
\t	Horizontal tab	0x09
\v	Vertical tab	0x0B

1.2.5 Strings

Insert a string into a string

```
string s = “ABCDEF”;
s = s.Insert(2, “999”) ;          //returns s = “AB999CDEF”
```

Pad characters

```
s = “56.3”;
s = s.PadRight(6, ‘0’);      //returns s = “56.300”, i.e. pads two zeros
```

Extract substring from the given string

`s.Substring(start[, length]),` *start* is the starting index (0 is the first) to be extracted and *length* is the number of characters from start. If length is omitted, the substring is extracted to the end of the string.

```
string s = "D34567";
s = s.Substring(1);   // returns s=”34567”
s = s.Substring(1, 2);  // returns =”34”
```

Another useful string function is the format of numerical numbers within a text string.

```
s = String.Format("The values are {0}, {1}, {2}", x, y, z);
s = String.Format("The values are {0:F2}, {1:F3}", 123.4567, 123.4567);
//results: s = "The values are 123.45, 123.456"

Hex and binary format can be displayed using Convert to string.
b1 = 56;
txt1.Text = b1.ToString("X") + "\r\n" ;
txt1.Text += Convert.ToString(b1,2); //2 specifies base=2
```

The format is specified using {#: $} where # is the index of variables after the comma starting 0, and $ is the formatting string. In the above case, F3 tells to print only three digits after the decimal point of the number. The available formatting characters are:

Format String	Description
G	General, formats numbers to a fixed point or exponential depending on the number
N	Number, it converts to a comma format, e.g., 12000 becomes 12,000
D	Decimal, 23.56
E	Scientific
F	Fixed point
P	Percent, 0.234 becomes 23.4%
R	Round-trip, converts to a string containing all significant digits it is used when you need to recover the number with no loss
X	Hexadecimal, converts to hex string, e.g., "X4" would display 65534 to FFFF

For custom formats, use the place-hold character # for digit or space and '0' for digit or 0.

```
{0: ##.00}    // it formats, a number 23.3456 into a string "23.34"
```

Searching a particular letter or word within a string is one of the frequently needed routines that could simplify programming. Among string methods, IndexOf or LastIndexOF is used.

```
string str  = "This is the string to be searched.";
int Position;
Position = str.IndexOf("is", 0, str.Length);   //Returns 2
```

The IndexOf method searches the word "is" starting from index 0 for the entire string and returns the index of the first occurrence. If it is not found, -1 is returned. The staring index plus

the count should be the position within the string *str*. Since the default is searching the entire string, it can be simplified to.

```
Position = str.IndexOf(“is”, 0) ;   //Returns 2
```

String is a 0 indexed array.

```
string s = “Going Down”;
for (int i=0; i<s.Length; i++)
    Console.WriteLine(s[i]);    //displays “Going Down” vertically
string s = “ABCDEF”;
Console.WriteLine(s[3]);  // displays “D”

//You may create reapeated characters using:
string s =  new string(‘A’,10);  // AAAAAAAAAA
```

Frequently, data is entered into a text file separated by delimiters such as comma or space. In order to read the data items, the string.Split() method can be used.

```
string data = “apple, pear, orange”;
string[] strArr = data.Split(‘,’);
// The array contains strArr[0]=”apple”, strArr[1]=” pear”, strArr[2]=” organge”
// Two items will include a space.
// Remove the space using Trim() method.
for (int i=0; i< strArr.Length; <i++)
   strArr[i] = strArr.Trim[i];
```

Sometime, the data is entered using more than one types of delimiters. In such a case multiple delimiters can be specified using Char[];

```
string txt = "aa, bb; cc dd"; //the delimeters are “,”, “;” and “ “
char[] delimeters = { ',', ' ', ';' };

//removing the empty items can be done using the RemoveEmptyEntries option
string[] strArr = txt.Split(delimeters, StringSplitOptions.RemoveEmptyEntries);
for (int i = 0; i < strArr.Length; i++)
{
    textBox1.Text += strArr[i] + "\r\n";
}
// This should display
aa
bb
cc
dd
```

1.2.5. DateTime

The type "DateTime" includes date and time, year, month, day, hour, minute, second.

```
DateTime d  = New DateTime(2006, 3, 5);  //March 5, 2006
DateTime d = New DateTime(2006, 3, 5, 14, 20, 40);  //March 5, 2006, 2:20:40 PM
```

There is no easy and simple way of changing time only from date type. A simple way is using a temporary string.

```
string tstr = "";
tstr = sDate.ToString("d") + " 12:00:00 AM";
sDate = Convert.ToDateTime(tstr);
```

Here are some DataTime format examples.

```
tstr = sDate.ToString("d");     // 2/28/2009
tstr = sDate.ToString("MMM");   // Feb
tstr = sDate.ToString("MMMM");  // Feburary
tstr = sDate.ToString("ddd");    // Sat
tstr = sDate.ToString("dddd");    // Saturday
```

Padding:

When text data needs to be aligned at the column, PadLeft or PadRight can be used to align them. The following example pads five spaces.

```
sw.Write("  {0}", oneWeek(j).ToString.PadLeft(5));
```

The system default integers for Day of week are:

0=Sunday, 1=Monday, 2=Tuesday, 3=Wednesday, 4=Thursday, 5=Friday, 6=Saturday.

Daylight saving time is retrieved using TimeZone and DaylightTime Objects.

```
TimeZone tz = TimeZone.CurrentTimeZone;
System.Globalization.DaylightTime dlc = tz.GetDaylightChanges(2005);
txt1.Text = dlc.Start;
```

Years, months, days, hours, minutes, seconds can be added or subtracted by a negative number.

```
DateTime d = DateTime.Today.AddDays(1);   //Tomorrow
DateTime d = DateTime.Today.AddDays(-1);   //Yesterday
```

It also exposes `Add` and `Subtract` methods. The object TimeSpan is convenient to use with these methods.

```
//Add 2 days, 5 hours, 20 minutes, and 30 seconds to Now.
DateTime t2  = DateTime.Now.Add(New TimeSpan(2, 5, 20, 30);
//Conversely, time span can be computed using the subtract method.
DateTime startTime = New DateTime(2005, 4, 6);
TimeSpan  timeTook  = DateTime.Now.Subtract(startTime);
```

Finding number of days between two dates can be done by using:

```
TimeSpan ts1 = endDate.Subtract(beginDate);
numberOfdays = ts1.Days    'This can be used as index since it is zero based
```

Get the number of days in a given month

```
DateTime.DaysInMonth(2009,5);  //number of days in 2009, May
```

Date and time can be displayed using GMT or local time.

```
string GMT = DateTime.Now.ToUniversalTime();
string CST  = Data.Now.ToLocalTime();
```

1.3 Operators

The basic arithmetic operators are:

```
+   addition
-   subtraction
*   multiplication
/   division
```

Integer Mod operations:

```
x = 7 % 3     //produces remainder,  x=1
```

Bitwise operations:

```
<<   Shift left
>>   Shift right
|    Or
&    And
^    Xor
```

Logical Boolean

```
==   Equal
!=   Not equal
&&   And
||   Or
<    less than
<=   less than equal
>    greater than
>=   greater than equal
```

Shorthand operations:

```
x += 1              // x = x + 1
x -= 2              // x = x - 2
x *= 2              // x = x * 2
x /= 10             // x = x / 10
```

1.4 Math Functions

All math functions are in the following name space.

```
using System.Math;
```

All of the available functions in .Net can be categorized in three groups.

1.4.1 Arithmetic functions

```
Abs, Ceiling, Floor, Min, Max, Sqrt, Exp, Log, Log10, Round, Pow, Sign, IEEERemainder

Ceiling(3.2)      // Ceiling(3.2)=4
Floor(3.2)        // Floor(3.2)=3
Max(2,5)          // Max(2,5)=5
Min(2,5)          // Min(2,5)=2
Sqrt(4)           // square root of 4 = 2
Exp(3)            // e^3
Log(5)            // returns natural log (base-e)
Log10(5)          // returns log with base-10
Round()           // round to nearest, 3.45 = 3.4, 3.46=3.5
Math.Pow(2, 3.5)  // 2^^(3.5), base is 2, and the power is 3.5
```

1.4.2 Trig and inverse trig functions

```
Sin, Cos, Tan, Asin, Acos, Atan, Atan2
```

1.4.3 Hyperbolic trig functions

```
Sinh, Cosh, Tanh
```

1.4.4 Constants

```
    E, PI
```

1.5 Arrays, ArrayList, Queue

1.5.1 Array operations

GetLengh(i) where i is dimension, returns the number of elements.

```
int[,,] a = new int[2,5,7];
a.GetLength(0)        //returns 2
a.GetLength(1)        //returns 5
a.GetLength(2)        //returns 7
```

Arrays can be copied partially using the Array.Copy method. In this case, the destination array size must be bigger than the size of source array.

```
int[] sourceArr =  {1,2,3,4,5};
int[] destArr = new int[20];
Array.Copy(sourceArr, destArr, 4);  // 4 indicates count starting from index=0
                //The content in destArr is now "1, 2, 3, 4, 0, 0, 0, 0, …"
```

You can sort a partial elements [5,100] of an array arr[100]:

```
Array.Sort(arr, 5, 96) ;     // 5 is the starting index, 96 is the length
```

You can also clear (set to 0) a part of array.

```
Array.Clear(arr, 10, 91);     //clear elements [10, 100]
```

Search the index of an element from an array. It is particularly useful for searching string arrays. The search is case sensitive.

```
String[] strArr = { "Aa", "B", "C", "D", "E" };
int i = Array.IndexOf(strArr, "C"); // i=2
```

1.5.2 Jagged arrays (array of arrays)

Jagged array is used when the size of array is not constant. The following is an example of two dimensional jagged array.

"00"
"10" "11"
"20" "21" "22"

```
String[][] arr = { New String[] {"00"},
                   New String[] {"10", "11"},
                   New String[] {"20", "21", "22"} };
Arr[2][1]                  // it contains "21"
Arr[1][0]                  // it contains "10"
```

1.5.3 ArrayList

ArrayList is similar to array but has collection functions. It is useful when the array size changes as you add the elements.

```
ArrayList al = new ArrayList(100);  // ArrayList must be instantiated before using it.
                                    // A default number of elements is 4.
al.Add("1");          // "1" is added to the list
al.Add("2");          //  "2" is added to the list
al.Add("3");           //  "3" is added to the list
al.RemoveAt(1);        // "2" is removed
al.Clear();            // empties all elements
```

After constructing an ArrayList, each element can be retrieved using the normal indexing techniques of an array. The number of elements can be retrieved using the count property.

```
al.Count();       // count is not index, it is always one bigger than the last index
```

1.5.4 Searching a value from array

Use the Array.IndexOf method. The search is case sensitive.

```
string[] sAry = {"Bob", "Joe", "Sue", "Ann"};
index = Array.IndexOf(sAry, "Joe");     // returns index=1
index = Array.IndexOf(sAry, "Kim");     //if search fails, it returns index=-1
```

1.5.5 Array of Controls

Suppose that you have three labels in the form and wish to control them using an array. The labels can be declared using a label array and the values can be set using the SetValue method.

```
System.Windows.Forms.Label[] lblPBbit = new System.Windows.Forms.Label[2];
lblBit.SetValue(lblBit0, 0);
lblBit.SetValue(lblBit1, 1);
lblBit.SetValue(lblBit2, 2) ;   // Can retrieve its properties by, e.g., lblBit[2].Name
```

From a label event, which label was clicked can be identified. The following example toggles the label text from "0" to "1" or vice versa whenever the label is clicked.

```
int index = lblPAbit.IndexOf(lblPAbit, sender);
If (lblPAbit(index).Text == "0") || (lblPAbit(index).Text = "")
  {
      lblPAbit(index).Text = "1";
   }
Else
  {
      lblPAbit(index).Text = "0";
  }
```

When controls are mixture of different types, it can be identified using GetType. The following slice of code show an example usage.

```
Control ctl;
 ArrayList strData = New ArrayList();
 ArrayList strHeader = New ArrayList();
        foreach (ctl in this.Controls)
         {
            if (ctl == Label)
            {
                if (Char.IsNumeric(ctl.Text.Substring(0, 1)))
                {
                    strData.Add(ctl.Name + "=" + ctl.Text);
         }
                else
              {
                    strHeader.Add(ctl.Text);
                 };
            };
        };
```

1.5.6 Queue Class

When you need FIFO memory or need a circular queue, use the Queue class.

```
Queue<int> q = new Queue<int>(30);        // Set a queue with 30 elements
q.Enqueue(10);
q.Enqueue(20);
q.Enqueue(30);
//Extract the first value
i = q.Dequeue();               // i = 10
//Read the next value but don't extract
i = q.Peek();                  // i = 20
// Extract it
i = q.Dequeue();               // i = 20
// Check how many items are still left in the queue
i = q.Count();                  // i = 1
```

2. If-then-else, Loops, Classes, Function

2.1 Conditional Statements

2.1.1 If-then-else statements:

```
The basic if-then example is

if (a > b)
{
   c = 1;
}
else
{
   c = 2;
}



// else-if structure can be used to check multiple conditions.

int Compare(int a, int b)
 {
    if (a > b)
         return 1;
    else if (a < b)
          return -1;
    else
          return 0;
}
```

2.1.2 Switch statement

Switch statement can be used for a selection of possible values in a clean way.

```
int x = award;
switch( x )
{
  case 1:
    Console.WriteLine("Winner!");
    break;
  case 2:
    Console.WriteLine("Runner-up!");
    break;
  case 3:
    Console.WriteLine("Second Runner-up!");
    break;
  default:
    Console.WriteLine("Not this time!");
    break;  // break must be specified, no default fall-through
}
```

The switch statement can only evaluate a predifiend type that includes string tyhpe and enum.

Unlike in Java and C++, the end of a case statement must explicitly state where to go to next. There is no error-prone default fall through behavior; not specifying a break results in the next case statement being executed.

```
string title = “Mr. Thompson”;
switch (title.Substring(0,2))
{
  case null:
    Console.WriteLine(“Unknown”);
    goto default;
  case “Mr”:
    Console.WriteLine(“A male member!”);
   // error, should specify “break”
  default:
    Console.WriteLine(“What’s your ID?”);
    break;
}
```

2.2 Loops

For loop has three parts: (1) statement executed before the loop begins, (2) Boolean expression that, if it is true, the loop block is executed, (3) a statement executed after iteration of the loop block.

```
int c;
for (int i=0; i<10; i++) // 10 loops
{
   c += 1;
}
```

Any of the three parts of the statement may be omitted. The following causes an infinitive loop.

```
for (;;)
{
  Console.WriteLine(“Infinite loop!”);
}
```

For collections, foreach statement is used.

```
foreach (Control ctrl in Form1.Controls)
{
     Console.WriteLine(ctrl.Text);

}
```

While loop:
Boolean expression is tested before iteration of the statement block.

```
int x = 0;
while (true)
```

```
{
   x++;
   if (x > 5 ) break;  //break from the loop
}

// example of using “continue” statement
int x = 0;
int y = 0;
while (y < 100)
{
  x++;
  if ( (x % 7) == 0)
     continue; // continue with next iteration
  y++;
}
```

Do-while loop

Boolean expression is tested after iteration of the statement block.

```
int x = 0;
do
{
    x++;
} while (x < 100);
```

2.3 Passing arguments in functions

By default, arguments are passed by values.

```
static int doubleVal (int a)
{
     return a*2;
}

int a = 5;
inr b = doubleVal(a);
Console.WriteLine(a.ToString()); // displays “5”
```

To pass the value by reference, use the modifier “ref”.

```
static void int doubleVal (ref int a)
{
     a = a*2;
}

int a = 5;
doubleVal(a);
Console.WriteLine(a.ToString()); // displays “10”
```

2.4 Classes

A typical class format is illustrated using an example.

```
class RandomNum
{
      float prob;
      int min, max; //random number range
      Random rand;

  // constructor, it has the same name as the class name
  public RandomNum ( int seed)
  {
    rand = new Random(seed);
    min = 0;  //default min
    max = 500; //default max
  }

  //property Min
  public int Min
  {
    get
    {
       return min;
    }
    set
    {
       min = value;
    }
  }

  //property Max
  public int Max
  {
    get
    {
      return max;
    }
    set
    {
      max = value;
     }
   }

// method
public int GetOne()
{
   return rand(min, max);
}

}
```

2.5 struct

```
struct Simple
{
   public int Position;
   public bool Exists;
   public double LastValue;
 };


// application of struct simple
Simple s;
s.Position = 1;
s.Exists = false;
s.LastValue = 5.5;
```

2.6 enum

```
public enum Direction
{
      North,
      East,
      West,
      South
}


// usage
if (d == Direction.North)
{
  …
}
```

2.7 Error Handling

```
single x, y;

try
{
   x = x/y
}
catch (Exception ex)
{
   MessageBox.Show( ex.Message );
}
```

The throw statement throws an exception to indicate an abnormal condition has occurred. The thrown exception is caught by Try-Catch.

```
throw new System.IO.FileNotFoundException()

// or
string msg = “File Not Found”
throw new System.IO.FileNotFoundException(msg)
```

```
// another example
float PEratio(float P, float E)
{
  if (E == null)
   {
     throw new ArgumentException(“E can’t be null”);
   }
  else
   {
     return P/E;
   }
}


// application program
try
 {
   Console.WriteLine( PEratio(x, y).ToString());
 }
catch (ArgumentException ex)
{
   Console.WroteLine(ex);
}
```

3. Files, Directories, Stream

Always include the following name space.

```
using System.IO
```

3.1 Files and Stream

3.1.1 Using File Stream (Text, Binary)

Read and write the entire text file:

```
string text = File.ReadAllText(@"C:\file.txt");
File.WriteAllText(@"C:\file1.txt", text.ToUpper()); // write the text as upper case
//Note that the @ character says "ignore" the back slash character, i.e. do not read as
an escape character.
```

Read a text file into an array of strings

```
String[] lines = File.ReadAllLines(@"C:\file.txt");
//Frequently, we need to remove empty lines when the file is a data file and it is
//supplied by a user or program. The following code slice is useful for such
//applications.
int count = 0;
for (int i=0; i<lines.Length; i++)
{
  If ( lines[i].Trim().Length > 0)
  {
    lines[count++] = lines[i];
   }
}
string[] lines2 = new string[count];
Array.Copy(lines, lines2, count); //remove the excess lines and copy to lines2
File.WriteAllLines(Application.StartupPath + @"\file2.txt");
```

When a log file is created, all you are doing is appending lines. The following code can be used.

```
string msg = string.Format("Program run at {0}\n", DateTime.Now);
File.AppendAllText(Application.StartupPath + @"\log.txt", msg);
```

A simple way of reading all bytes:

```
byte[] bytes = File.ReadAllBytes("bin.dat");
```

3.1.2 Using File Streams (Text, Binary)

Read a line from a text file

```
StreamReader sr = New StreamReader("C:\file.txt");
string txtData = sr.ReadLine();
sr.Close();
```

Other alternative way of opening file streams.

```
Stream st = File.Open(filename, FileMode.Open, FileAccess.ReadWrite,
FileShare.ReadWrite);
StreamReader sr = new StreamReader(st);
```

Or,

```
FileStream fs = new FileStream(filename, FileMode.Open);
StreamReader sr = new StreamReader(fs);
```

You may read text until the end of file using Peek

```
while (sr.Peek() != -1)
{
    txtData += sr.ReadLine() + "\r\n";
}
sr.Close();
```

The same code can be written using built in EndOfStream property.

```
while (! sr.EndOfStream)
{
    txtData += sr.ReadLine() + "\r\n";
}
sr.Close();
```

Read the entire contents in a single line of code.

```
txtData = sr.ReadToEnd();
```

Write a text file:

```
StreamWriter sw = New StreamWriter(Application.StartupPath + @"\file.txt");
sw.Write("This is a test");
sw.Close();
```

Another way of opening StreamWriter:

```
Stream st = FileOpen(filename, FileMode.Create, FileAccess.ReadWrite, FileShare.None);
StreamWriter sw = New StreamWriter(st);
```

Yet, another way:

```
FileStream fs = new FileStream(filename, FileMode.Open);
StreamWriter sw = New StreamWriter(fs);
```

For reading and writing binary files, the BinaryReader and BinaryWriter classes are used. However, unlike the text reading and writing, the short form cannot be used. A file stream must be defined before applying the BinaryReader and BinaryWriter. Another important point is that it can write only one datum at a time.

```
Stream st = File.Open(Application.StartupPath + @"\data.bin", FileMode.Create,
FileAccess.Write);
BinaryWriter bw = New BinaryWriter(st)
int[] ary = {1, 2, 3, 4};
int[,] mat = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
Note that binaryWriter can write only one datum at a time.
  for (int i = 0; i< 4; i++)
  {
```

```
            bw.Write(ary[i]);
 }
```

If a two dimensional data is written, the order should be remembered when read.

```
for (int i = 0; i<3; i++)
 {
   for (int j = 0; j < 3; j++)
    {
        bw.Write(mat[i, j]);
     }
  }
 st.Close();
 bw.Close();
```

The following code slice reads the data written back by following the same order it is written.

```
st = File.Open(Application.StartupPath + @"\data.bin", FileMode.Open, FileAccess.Read)
BinaryReader br = New BinaryReader(st);
int[] ary1 = new int [4];
for ( i = 0; i<4; i++)
 {
      ary1[i] = br.ReadInt32;
 }
 int[,] mat1 = new int[3,3];
 for ( i = 0; i<3; i++)
  {
     for (j = 0; j < 3; j++)
      {
          mat1[i, j] = br.ReadInt32;
       }
   }
```

In VS2005, (PeekChar()==-1) causes an error because a character is Unicode and 16 bits. The following solves this problem.

```
while (br.BaseStream.Position < br.BaseStream.Length)
{
   // data reading code here
   data(i) = br.ReadDouble();
}
```

When you pass streams using a subroutine, should you pass as by value or by ref? The answer is that it works both ways.

```
void callsw(ref StreamWriter sw)
{
   sw.WriteLine("This is a test");
   sw.Close();
}
```

3.1.3 Reading and Writing from Strings

Lines in a multi-line text in a long string can be read using StringReader.ReadLine.

```
string LString, s;
StringReader strR = New StringReader(LString);
while ( strR.Peek != -1)
```

```
{
   s = strR.ReadLine();
}
```

For writing StringWriter class is used.

3.2 Getting All of the Filenames in a Directory

Following example is useful when you want to display all of the *.txt files in a directory say, “C:\myfiles”.

```
foreach ( string fname in Directory.GetFiles(@“C:\myfiles”, “*.txt”))
{
       Textbox.Text += fname + “\r\n”;
}
```

You can easily modify file’s last write time using the file class.

```
File.SetCreation(fname, Date.Now);
```

3.3 Getting All of the Directories in a Directory

The following example shows how to get all of the directories in a given directory and the getting files of the directory.

```
string dname, fname;
foreach (dname in Directory.GetDirectories(txtFolder.Text))
{
    txt1.Text += dname + “\r\n”;
    foreach (fname in Directory.GetFiles(dname, "*.txt"))
     {
        txt1.Text += fname + “\r\n”;
     }
}
```

3.4 Extraction of Path and filename

In the process of file and path handling, we often need to extract the filename or path from the complete path. The Path and Directory classes can be used.

```
//Assume that pathFile = “C:\MyFile\file.txt”
pathStr = Directory.GetParent(pathFile).ToString();  //pathStr=”C:\MyFile”
name = Path.GetFileName(pathFile); //name=”file.txt”
name = Path.GetExtension(pathFile); //name=”.txt”
name = Path.GetFileNameWithoutExtension(pathFile);  //name =”file”
name = Path.Root(pathFile);   //name=”C:\”
Path.HasExtension(pathFile);   //returns true/false
```

3.5 How to Check Existence of Directory

Suppose that you need to check existence of a directory before you create a directory. The following example checks the directory before it creates.

```
// Create a directory if it does not exist
string dirPath;
```

```
dirPath = Application.StartupPath + @"\Daylets"
if (! Directory.Exists(dirPath) )
 {
     Directory.CreateDirectory(dirPath);
}
```

The whole directory is deleted by,

```
 Directory.Delete(dirPath);
```

3.6 Get file size, creation time, etc from file or directory: FileInfo, DirectoryInfo, DriveInfo classes

To get the size and creation time of a file, use the properties of FileInfo class.

```
FileInfo sfi = New FileInfo(myPath & "\" & fname);
sfi.Length;
sfi.CreationTime;
```

For getting information about a directory, us

```
DirectoryInfo di = New DirectoryInfo(myPath);
```

4. Frequently Used Utilities

4.1 Random number generator

First, set the random seed.

```
Random rand = new Random(54321);

// random seed using time
int rseed = (int) (DateTime.Now.Ticks & int.MaxValue);
rand = new Random(seed);

// get a value in the range 0 to 50
int r = rand.Next(50);

// get a value between 100 to 500
int r = rand.Next(100, 500);

// get a value between 0 and 1
double dr = rand.NextDouble();

// get an array of 100 random byte values
byte[] buffer = new byte[100];
Rand.NextBytes(buffer);
```

4.2 OpenFileDialog/FolderBrowserDialog/Set Attributes/Set Access Time

It is convenient to get a filename through a windows dialog.

```
// This example allows users to choose files from a list of *.txt
OpenFileDialog1.Title = "Select a Inductance Signature Data File.";
// You can display only certain types of files, e.g., *.txt or *.csv.  Each choice
//should be entered by  "|" separator, and two fields must be provided within each
// field, i.e., "file description | mask".
OpenFileDialog1.Filter = "Data files (*.TXT)|*.TXT|Data files (*.csv)|*.csv";
OpenFileDialog1.FileName = "";
if (OpenFileDialog1.ShowDialog() == DialogResult.OK)
{
      fname = OpenFileDialog1.FileName;
}
```

Multiple files can be loaded by selecting the multiselect property to true from the OpenFileDialog property. Then, the selected file names are returned as an array of strings.

```
OpenFileDialog1.Multiselect = True;
string[] Fnames;
OpenFileDialog1.Filter = "Data files (*.csv)|*.csv| Data files (*.txt)|*.txt| All files
(*.*)| *.*";
 if (OpenFileDialog1.ShowDialog() == Windows.Forms.DialogResult.OK)
  {
      Fnames = OpenFileDialog1.FileNames;
  }
```

You may use the SaveFileDialog control in a similar manner.

Folder browser dialog is used to obtain the folder name.

```
fbDialog.SelectedPath = "C:\MyPrograms";  //it is convenient to set the initial
directory
fbDialog.ShowNewFolderButton = False;   //do not allow to create new folder
if (fbDialog.ShowDialog == DialogResult.OK)
 {
      textbox1.Text += fbDialog.SelectedPath;
 }
```

It will display the selected folder to the textbox.

In the following example, all of the files in the selected directory are set to read-only attribute using the folderBrowserDialog. Attributes can be useful in handling files.

```
fbDialog.SelectedPath = "C:\"; //It is convenient to set the default startup directory
if (fbDialog.ShowDialog == DialogResult.OK)
{
     selectedPath = fbDialog.SelectedPath;
     txtOutput.Text += selectedPath + "\r\n";  //display the list of files in that
directory

     string fname;
     FileAttributes attr;
     foreach fname in Directory.GetFiles(selectedPath, "*.*")
      {
          txtOutput.Text += fname + "\r\n";
          attr = File.GetAttributes(fname);
          attr = attr | FileAttributes.ReadOnly;  //attributes are bits and must use Or
to maintain the rest of existing setting
          File.SetAttributes(fname, attr);
       }
 }
```

Another useful method is setting the last access time of the file. It can be done using:

```
    File.SetLastAccessTime(fname, Date.Now);
```

Folder dialog

The following line is required. If it is not set, the program hangs because it cannot search the directory.

```
FolderBrowserDialog1.RootFolder = Environment.SpecialFolder.MyComputer;

FolderBrowserDialog1.SelectedPath = txtExportDir.Text;  //sets the initial path

if (FolderBrowserDialog1.ShowDialog == Windows.Forms.DialogResult.OK)
 {
     txtExportDir.Text = FolderBrowserDialog1.SelectedPath;
     ini1.setINI("Path", "ExportDir", txtExportDir.Text);
 }
```

4.3 Scrolling the textbox after filling in text

Make sure to set the Scrollbars property to Vertical and use the following three lines of code slice. It will then properly scroll the text up in the textbox.

```
TextBox1.SelectionStart = TextBox1.TextLength + 1;
TextBox1.SelectionLength = 0;
TextBox1.ScrollToCaret();
```

4.4 Run Notepad/Excel from a program at run time.

Use the process class. The following example opens the file in Application.StartupPath + "\TToutput.txt" on a notepad.

```
using System.Diagnostics;

 Process proc = New Process;
 proc.StartInfo.FileName = "Notepad.exe";
 proc.StartInfo.Arguments = Application.StartupPath + "\TToutput.txt";
 proc.Start();
```

Similarly, a csv file can be open using the Excel.

```
  Process proc = New Process;
  proc.StartInfo.FileName = "Excel.exe";
  proc.StartInfo.Arguments = Application.StartupPath + "\data.csv";
  proc.Start();
```

In some computers, above approach does not find the Excel. The following is more safe approach when the path to Excel is unknown.

```
   string exeString = Application.StartupPath + "\loadTemp.csv";
   Process.Start(exeString);
```

4.5 Hash Table for Fast Search

If a fast search is needed for an array of data, a hash table should be used.

```
Hashtable hTable = New Hashtable(200, 1);  // 200 is the capacity, 1 defines
the load factor


     hTable.Add("100", 20); //add items  (key, value)
     hTable.Add("101", 21);
     hTable.Add("102", 221);

      hTable(“100”);   // returns 20
      hTable(“101”);   // returns 21
      hTable(“102”);   // returns 221
```

Note: the values can be any integer.

If the key is not found, it returns null. Therefore, if-null phrase should be tested as shown by below example:

```
int index;
if (hTable(“107”) == null)
{
     Console.WriteLine(“Key \“107\” Not found”);
}
else
{
     index = hTable(“107”);
}
```

Since hTable() does not trigger exception, the try-catch cannot be used to check if the key can be found or not, i.e., **the following does not work**.

```
int index;
Try
{     index = hTable(“107”);
}
catch (Exception ex)
{
     Console.WriteLine("107 Not found");
}
```

4.6 Binary Search

Binary provides a fast search but it can only be used for integer numbers. Floating points do not work and produces exception errors. Array and ArrayList can be used. It is always a good idea to sort the numbers before applying the binary search.

```
int[] dat = {1, 3, 5, 7};
txt1.Text = Array.BinarySearch(dat, 5).ToString(); //returns 2
txt1.Text = Array.BinarySearch(dat, 2).ToString(); // returns -2, when the searching
number does not exist, it returns a negative but the magniture depends on the entry
number.
```

ArrayList requires sorting before BinarySearch is applied. If it is not sorted, it returns a negative number.

```
using System.Collections;


ArrayList dat1 = new ArrayList();
dat1.Add(1);
dat1.Add(5);
dat1.Add(3);
dat1.Add(7);
dat1.Add(9);
dat1.Sort();
txt1.Text = dat1.BinarySearch(5).ToString();   //returns 2
txt1.Text = dat1.BinarySearch(20).ToString(); //returns -6, a negative number
```

4.7 Handling of Windows Generated Events

Events can be trapped using WithEvents variable. However, more simple way of dealing with events is using EventHadler. Suppose that you wish to generate a button dynamically inside the program. First, create a button, i.e.,

```
Button btnExit = new Button();
btnExit.Text = "Exit";
this.Controls.Add(btnExit);
```

Next, add an event handler that processes the btnExit click event.

```
btnExit += new EventHandler(btnExit_Click);
```

The event handling routine can then be added.

```
private void btnExit_Click(Object sender, EventArgs e)
 {
  Application.Exit();
  }
```

Frequently, you will find that you need to call this function without the users' actual click action. If such cases arise, you can simply call this subroutine using:

```
EventArgs evt = new EventArgs();
btnExt_Click(this, evt);
```

4.8 Creating and Trapping Custom Events

In the previous example (4.7), the event was created by Win32 control. Sometime, you will need to create your own event and the corresponding event handler.

First, let's create a class that generates an event when some data is received.

```
Class DataCollection
{
     public event GotData(string data);   //define event

    // Data collection unit
    void RecData()
    {
         'read it from a source such as a serial port
          string msg = ReadFromSerial(); //assume that this routine read a string
          RaiseEvent GotData(msg);
    }
}
```

This class is capable of generating an event whenever data is received from the serial port assuming that ReadFromSerial() reads a line from the serial port. Next we need to trap the event in the application, which can be done by adding an AddHandler.

```
public partial class Form1 : Form
{
       public Form1()
        {
          InitializeComponent();
         //Add event handler
          dc.GotData += new EventHandler(OnReceive);
        }


    public DataCollection dc = New DataCollection();

    private void OnReceive(string msg )
     {
        MessageBox.Show(msg);
     }
}
```

In the above example, a string was passed as a result of an event. However, you do not have to pass a value if using a property is more efficient.

4.9 Some Windows Controls

4.9.1 GroupBox Control

Controls in GroupBox is not exposed in this.controls collection. Suppose that multiple check boxes are in the GroupBox and you want to set the Checked property true for all of them. The following snippet shows how to do it.

```
Control ctl;
foreach (ctl In GroupBox1.Controls)
 {
     if (ctl == CheckBox)
      {
         ctl.Checked = true;

      }
  }
```

4.9.2 CheckedListBox Control

The following gets indeces of the checked items

```
    for (int i = 0; i < ckListSta.CheckedIndices.Count; i++)
     {
        Txt1.Text += ckListSta.CheckedIndices(i) + "\r\n";
     }
```

To get the checked items, use the following:

```
    for (int i = 0; i < ckListSta.CheckedItems.Count; i++)
     {
           Txt1.Text += ckListSta.CheckedItems(i) + "\r\n";
     }
```

5. GDI+

GDI+ is used to produce text and graphic outputs. It also deals with bitmaps and other kind of images. In order to use GDI+, the following namespaces are used:

```
using System.Drawing;
using System.Drawing.Drawing2D;
using  System.Drawing.Imaging;
using System.Drawing.Text;
```

5.1 Graphics Object Reference

In order to draw graphics on a drawing surface, you must first get a reference to a Graphic object. There are two approaches: (1) get a Graphics object from the argument of an event, or (2) get a Graphics object by using the CreateGraphics method.

5.1.1 Getting from the argument of event

In a form, a paint event exposes Graphics reference. Other events do not expose the reference to Graphics object such as the Resize event.

```
private void Form1_Paint(Object sender, PaintEventArgs e)
{
    //Get graphics object for the form's surface
    Graphics gr = e.Graphics;
    gr.DrawEllipse(Pens.Red, 0, 0, ClientSize.Width, ClientSize.Height);
 }
```

If you resize the form, you will notice that the drawings are distorted since repaint is not executed again. The following solves this problem by redrawing the graph object using another Win32 event.

5.1.2 Using CreateGraphics

If the event parameter does not expose a reference to the Graphics object, you can use the CreateGraphics method. After using this object the graphic object must be destroyed to save the resource.

```
private void Form1_Resize(Object sender, EventArgs e)
{
    Graphics gr = this.CreateGraphics;
    gr.DrawEllipse(Pens.Red, 0, 0, ClientSize.Width, ClientSize.Height);
    gr.Dispose();
    this.Refresh();
 }
```

Refresh() is needed to activate the paint action after the drawing.

5.2 Drawing Methods

DrawArc

DrawBezier

DrawCurve

DrawEllipse

DrawLine

DrawPath

DrawPie

DrawPolygon

DrawRectangle

DrawString

FillClosedCurve

FillEllipse

FillPath

FillPie

FillPolygon

FillRectangle

FillRegion

5.3 Imaging

GDI+ can load images from the following formats: bitmaps (BMP), GIF, JPEG, PNG, and TIFF.

```
Graphics g = e.Graphics;
Bitmap bmp = new Bitmap("rama.jpg");
g.DrawImage(bmp, 0, 0);


// scale the image to fit  to the client area of the form
g.DrawImage(bmp, this.ClientRectangle);


// changing resolution of bitmap
bmp.SetResolution(600f, 600f);
bmp.SetResolution(1200f, 1200f);

// resize the image using the lowest quality interpolation mode
int width = bmp.Width;
int height = bmp.Height;
g.InterpolationMode = InterpolationMode.NearestNeighbor;
```

```
//Bilinear (default) or HighQualityBilinear can be selected.
g.DrawImage(bmp,
   new Rectangle(10, 10, 120, 120); // source rectangle
   new Rectangle(0, 0, width, height); // destination rectangle
   GraphicsUnit.Pixel);
```

Cropping an image:

```
Rectangle sr = new Rectangle(80, 60, 400, 400);
Rectangle dr = new Rectangle(0, 0, 200, 200);
g.DrawImage(bmp, sr, dr,   GraphicsUnit.Pixel);
```

6. Regular Expression

```
using System.Text.RegularExpressions;
```

Regular expressions are awkward and contorted looking language, but can be a powerful tool for searching a pattern from a string. For example, the following removes digits (\d) followed by an 'a' character.

```
string data = "a1 a2 a3";
Regex rex = New Regex(@"a\d");        // 'a' followed by a digit (\d)
Console.Writeline(rex.Replace(data, "a"));   // should print, "a  a  a" since it
replaces "a#" with a.
```

Continuing the above example, each match can be analyzed using the MatchCollection object.

```
MatchCollection mc = rex.Matches(data);
foreach (Match m in mc)
{
  Console.WriteLine("'{0}' at position {1}", m.Value, m.Index);
}
```

This should display:

```
'a1' at position 0
'a2' at position 3
'a3' at position 6
```

In regular expression, constructing a search patter is important. Here are few examples.

Extracting date/time

```
string text = "0/02/03 16:20:34";
string p = @"(\d+)/(\d+)/(\d+) (\d+):(\d+):(\d+)";
Match m = Regex.Match(text, p);
```

Detecting IP addresses

```
string text = "55.54.53.52";
string p = "^" + @"([01]?\d\d|2[0-4]\d|25[0-5])\." + @"([01]?\d\d|2[0-4]\d|25[0-5])\."
   + @"([01]?\d\d|2[0-4]\d|25[0-5])\." + @"([01]?\d\d|2[0-4]\d|25[0-5])" + $;
Match m = Regex.Match(text, p);
```

Finding all caps words.

```
string text = "This IS a Test OF ALL Caps";
string p = @"(\b[^\Wa-z0-9_]+\b)";
MatchCollection mc = Regex.Matches(text, p);
```

The examples shown here are just a first step and a lot more needs to be leaned to be able to apply for practical applications. To learn more about Regular Expresions, please refer to the reference [1].

7. Threading

For writing threads, the following name space should be on the top of your program.

```
using System.Threading
```

Threads can be viewed as separate small programs within a program. Let's suppose you are acquiring data from a serial port and you wish to run that function as a thread.

```
//Create a new thread
Thread th = new Thread(new ThreadStart(GetData));
th.Start();
…
th.Abort();
```

The th.Start() runs the function GetData() as a new thread. This thread, th, is terminated when it exits the GetData() sub. If you wish to force the termination, you may issue th.Abort() within your program. It is important to remember that threads start asynchronously, i.e., it may not **immediately** start and executing the function, in this case, GetData().

Somewhere in your code, you should have the function GetData()

```
void GetData()
{
     string s;
     GetSerialData(s);
     PassToBuffer(s);
     Thread.CurrentThread.Sleep(10);  // wait for 10ms
}
```

When you are using a thread, there is a risk that other objects may use the section of the code or break in between the code. To ensure that only one thread use a section of code at a time, lock statements could be used. For the above example, lock can be placed for the segment of getting the data and putting the data into a buffer to ensure that only one thread executes these two routines. This removes any chance that the data is copied to a wrong place.

```
static object mylock = new object();
void GetData()
{
     string s;
      lock (mylock)  //another of writing this is lock(this) using this object
      {
        GetSerialData(s);
        PassToBuffer(s);
      }
      Thread.CurrentThread.Sleep(10);
}
```

Another approach to synchronization is using the Mutex (mutual exclusive) class. The Mutex class can be owned by only one thread at a time.

```
Mutex m = new Mutex();
void GetData()
{
     string s;

      m.WaitOne();
        GetSerialData(s);
        PassToBuffer(s);
      m.ReleaseMutex();

     Thread.CurrentThread.Sleep(10);
}
```

8. Topics to Study Further

There are many topics not covered in this short note but important. The recommended topics to study further include:

- Generics
- Interfaces
- Attributes
- Reflection
- Object Serialization
- PInvoke and COM Interop
- Control Designs

An excellent book to learn C# further is:

[1] Francesco Balena, *Programming Microsoft Visual C# 2005: The Basic Class Library*,Microsft Press, 2006.

If you wish to learn GDI+, the following book is one of the good choices.

[2] Eric White, *Pro .Net 2.0 Graphics Programming*, Apress, 2006.