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

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

The purpose of this short manual is to provide a quick learning path to programming in Microsoft VB.Net for the students who already programming experiences in c, c++ or other programming languages. VB.Net is a powerful programming tool on the contrary to the name suggests, and I found that students can quickly learn the language if they had c or Java experiences. According to my observation, the student's learning was much faster if the basic utility routines and classes along with an example are given. This manual was written to provide example utilities that students can quickly lookup and modify or copy to their programs. Unlike c, VB.net can be written much more quickly if you know many tricks and utilities, but remembering is often the problem. Therefore, I attempted to collect most frequently used utility routines, based on my own programming experience.

For learning more extensive list of techniques involved in VB.net programming, I recommend students to read "Programming Microsoft Visual Basic .Net" written by Fransesco Balena.

I will regularly update this manual, and any suggestion to improve this manual would be appreciated. Please don't hesitate to email me at tkwon@d.umn.edu.

Last Updated: Feb 1, 2007

Table of Contents

1. Basics

1.0 Windows Control Prefix Convention

For easy identification of Windows form controls, (a prefix + function) is recommended to be used for all control names. Whenever a control is placed on the form, the (name) property should be changed to follow this convention. 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
Label	lbl
Button	btn
Textbox	txt
Menu	mnu
CheckBox	chk
PictureBox	pic
Panel	pnl
DataGrid	dg
ListBox	lst
CheckedListBox	clst
ComboBox	cbo
ListView	lv
TreeView	tv
Timer	tmr
OpenFileDialog	ofd
SaveFileDialog	sfd
FolderBrowserDialog	fbd
ColorDialog	cld
FontDialog	fnd

1.1 First Few Lines

At the top of the program, always declare the option as “Explicit On” so that the compiler checks for undefined variables.

```
Option Explicit On
```

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

```
Imports System.Net          ‘ for all network programming
Imports System.Text         ‘ for binary array to ASCII or vice versa conversion routines
Imports System.IO           ‘ for file operations such as stream
Imports System.Math         ‘ for math functions such as sin, cos, log
```

1.2 Declarations

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

```
Dim x, y, z As Single
Dim i As Integer, x As Single, s As String
```

Variables can be initialized where declared using an equal sign.

```
Dim x As Single = 100.5, Name as String = “Tony”
```

In VB Hexadecimal numbers are expressed using &H####.

```
Dim flag As Integer = &HA3CB
```

1.2.1 Array Declaration

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

```
Dim xarray(3) As Single   ‘ declares 4 elements xarray(0), xarray(1),  xarray(2),  xarray(3)
Dim buff(1020) As Byte    ‘ declare a byte array with 1021 elements, it is important to remember
                           ‘ that every array index starts from 0 and ends with the declared index.
                          ‘ In this example, the buff array has elements from buff(0) to buff(1020).
```

If you do not know the number of elements or it is undetermined, a variable array can be declared without defining the size. The array must be re-dimensioned using ReDim before it is used.

```
Dim buff() As Byte            ‘ define variable array
ReDim buff(1020)              ‘ ReDim can be used many times.
ReDim Preserve buff(2040)     ‘ Extend the array size while keeping the old content.
```

A multi-dimensional array is defined by separating each dimension by a comma.

```
Dim a(1,1)  As Integer   ‘ it allocates four elements: a(0,0), a(0,1), a(1,0), a(1,1)
```

For array initialization, curly braces are used.

```
Dim A() As Integer = {1, 2, 3 ,4}   'one dimensional array initialization
Dim B(,) As Integer = { {1, 2, 3}, {4,5,6} }  'two dimensional array initialization
```

1.2.2 Constant Declaration

Constants can be declared using the "Const" statement.

```
Public Const myPi As Single = 3.14 'Declare myPi as a constant 3.14.
Area = myPi * r^2
```

1.2.3. String Constants

Commonly used string constants are:

```
VbCrLf
VbCr
VbLf
VbTab
VbRed, VbGreen, VbBlue, …
```

These are inherited old VB6, but they still works in .net. In the native .Net, some of these characters are defined in the ControlChars class and can be used as:

```
Dim crlf As String = ControlChars.CrLf
```

The ControlChars class contains: Back, Cr, CrLf, FormFeed, NewLine, NullChar, Quote, Tab, and VerticalTab.

The color constants are now in the System. Drawing class and more varieties are available. For example, above VbRed can be replaced with:

```
System.Drawing.Color.Red
```

1.2.4 String manipulations

Insert a string into a string

```
s = "ABCDEF"
s = s.Insert(2, "999")        ' returns s = "AB999CDEF"
```

Pad characters

```
s = "56.3"
s = s.PadRight(6, "0"c)     ' 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

```
Dim s As String = "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)
‘results: s = “The values are 123.45, 123.456”
```

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. The available formatting characters are:

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

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, e.g., X4: 65534 becomes FFFF

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

```
{0: ##.00}      ‘ it formats, for example, number 23.3456 into a string “23.34”
```

1.2.5. Date Time

The type “Date” includes date and time, year, month, day, hour, minute, second.

```
Dim d As New Date(2006, 3, 5)  ‘March 5, 2006
Dim d As New Date(2006, 3, 5, 14, 20, 40) ‘March 5, 2006, 2:20:40 PM


Dim d As New Date.Now   ‘Returns system date and time
Dim d As New Date.Today  ‘Returns date only, and time is set 12:00:00 AM
```

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

```
Dim d As New Date.Today.AddDays(1)   ‘Tomorrow
Dim d As New Date.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.
Dim t2 As Date = Date.Now.Add(New TimeSpan(2, 5, 20, 30)
Conversely, time span can be computed using the subtract method.
Dim startTime As New Date(2005, 4, 6)
Dim  timeTook As TimeSpan = Date.Now.Subtract(startTime)
```

Suppose that you wish to create a directory using today and the file name with the current time. This can be done using a predefined variable "Now". First, the directory is created using:

```
Dim DataDir As String
DataDir = Application.StartupPath
DataDir += "\" + CStr(Now.Year) + Format(Now.Month, "00") + Format(Now.Day, "00")
      If Not Directory.Exists(DataDir) Then
         Directory.CreateDirectory(DataDir)
      End If
```

Next, the file is created using a binary stream as an example.

```
Dim st As Stream
Dim binStream As BinaryWriter
Dim filename As String
filename = Format(Now.Hour, "00") + Format(Now.Minute, "00") + Format(Now.Second, "00")
st = File.Open(DataDir + "\" + filename, FileMode.Create, FileAccess.Write)
binStream = New BinaryWriter(st)
```

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

```
Dim GMT As String = Date.Now.ToUniversalTime
Dim CST As String = Data.Now.ToLocalTime
```

1.2.6 Line Continuation

A long line code can be broken into multiple lines by simply appending underscore "_" where you want to break the line, e.g.,

```
timeMiDelta = (CDbl(txtSensorDistance.Text) * 3600) / _
      ((CDbl(txtSpeed.Text) + CDbl(txtSpeedError.Text)) * 5280)
```

1.2.7 Structures (user defined types)

The user defined types in old VB was created using the Type…End block. This is now supported in .Net using the Structure…End block, but it goes more than replacement. Structure now supports methods, and it is nearly identical to classes. A simple example is given below.

```
Structure Person
        Dim FirstName As String  'Dim means Public here
        Dim LastName As String
        Function FullName() as String
                FullName = FirstName & " " & LastName
        End Function
End Structure
```

The defined structure is used as the same way as you use other types of variable, i.e.,

```
Dim p1 As Person
```

1.3 Operators

The basic arithmetic operators are same as c or c++, i.e.,

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

One of the differences is in the Not Equal operation. In VB, it uses the following symbol:
<> same as "!=" in c++.

Also, "=" is used for both an assignment and for "= =" in c++.

Bit shifting of binary is done using ">>" and "<<". However, a caution must be given, ">>" is an arithmetic shift to right, i.e., it retains the sign bit.

```
Dim h as Short = &H80     ' h = 1000 0000 0000 0000
h >> 2                    ' h = 1110 0000 0000 0000
h = 3                     ' h = 0000 0000 0000 0011
h << 2                    ' h = 0000 0000 0000 1100
```

Shorthand operations are same as c++:

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

Power operations:

```
x = 2 ^ 3       ' produces x = 2 * 2 * 2
y = x ^ 2.5     ' produces x=5.656854
```

Integer mod operations:

```
x = 7 \ 3       ' produces quotient,  x = 2
x = 7 Mod 3     ' produces remainder,  x=1
```

1.4 Math Functions

All math functions are in the following name space.

```
Imports 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

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

1.4.4 Constants

A random number with a seed 1234 is generated by

```
Dim rand As New Random(1234)
```

To get 100 random numbers between 100 and 1000, try

```
Dim randomValue As Integer
For i=1 to 100
  randomValue = rand.Next(100,1000)
Next
```

1.5 Arrays, Collections and Structure

1.5.1 Array operations

Empty array is checked using “Is Nothing”.

```
If Arr Is Nothing then
    Redim Arr(20)
End If
```

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

```
Dim a(2,5,7) as Integer
a.GetLength(0)      ‘returns  3
a.GetLength(1)      ‘returns 6
a.GetLength(2)      ‘returns 7
```

Create a copy of array using DirectCast.

```
Dim arr(4,3) As integer
Dim ArrayCopy(,) As Ingeter = DirectCast( arr.Clone, Integer())
```

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.

```
Dim sourceArr() as Integer = {1, 2, 3, 4, 5}
Dim destArr(20) as Integer
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, …"
```

The CopyTo method can be only useful if the copying array is one dimensional.

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.

```
Dim strArray() As String = {"A", "B", "C", "D", "E"}
i =  Array.IndexOf(strArray, "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"

```
Dim arr()() As String = { 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.

```
Dim al As new ArrayList(100)  ' ArrayList must be instantiated using new before it is used.
                              ' It then allocates a default amount of elements.
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.

```
Dim sAry() as String = {"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 Queue Class

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

```
Dim q As New Queue(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.Deque               ‘ i = 20
‘ Check how many items are still left in the queue
i = q.Count               ‘ i = 1
```

1.5.6 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.

```
Dim lblPBbit As System.Windows.Forms.Label()
 ' Set the lable names as the values of array elements
 lblBit = New System.Windows.Forms.Label(2) {}
 lblBit.SetValue(lblBit0, 0)
 lblBit.SetValue(lblBit1, 1)
 lblBit.SetValue(lblBit2, 2)        ‘ Can be retrieved 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.

```
Dim index As Integer = lblPAbit.IndexOf(lblPAbit, sender)
If lblPAbit(index).Text = "0" Or lblPAbit(index).Text = "" Then
    lblPAbit(index).Text = "1"
Else
    lblPAbit(index).Text = "0"
End If
```

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

```
Dim ctl As Control
Dim strData As New ArrayList
 Dim strHeader As New ArrayList

    For Each ctl In Me.Controls
      If ctl.GetType Is GetType(Label) Then
        If IsNumeric(ctl.Text.Substring(0, 1)) Then
          strData.Add(ctl.Name + "=" + ctl.Text)
        Else
          strHeader.Add(ctl.Text)
        End If
      End If
    Next
```

1.5.7 Structures

Structures in general should be claimed as public and placed in a separate module, since they define a new type of variables. The following shows an example.

```
Public Structure Person
    Public firstName As String
    Public lastName As String
    Public birthDate As Date
End Structure
```

After the structure is built, it can be used in the program as

```
Dim Dave As Person
Dave.firstName=”Dave”
Dave.lastName=”Johnson”
Dave.birthDate=#1/2/1980#
```

Methods and properties can be included in the structure similarly to classes in VB.net. Please consult helps in the VS.

1.6 Conditional and Loop Statements

1.6.1 If-then-else Conditional statements:

```
If a > 0 then
  MsgBox(“ a > 0”)
Elseif a < -3 then
  MsgBox (“ a < -3 and a <=0”)
Else
  MsgBox(“  3<= a <=0”)
End If
```

Use a short circuit statement for more than one if conditions. From VS 2003, AndAlso and OrElse are available.

```
If a1 > 0 AndAlso a1 >  b then ok = True
```

In this case, if the first condition is OK then it tests the second condition.

```
If a1 > 0 OrElse Log(a1) > 3 then ok = True
```

1.6.2 Select Case Statement

When you need to execute one of several groups, depending on the value of an expression, use *Select-Case* statements.

```
Dim Number As Integer = 8
Select Case Number   ' Evaluate Number.
  Case 1 To 5   ' Number between 1 and 5, inclusive.
    Debug.WriteLine("Between 1 and 5")
   Case 6, 7, 8   ' Numbers 6, 7, and 8.
    Debug.WriteLine("Numbers 6, 7 and 8")
  Case 9 To 10   ' Number is 9 or 10.
    Debug.WriteLine("Greater than 8")
  Case Else   ' Other values.
    Debug.WriteLine("Not between 1 and 10")
End Select
```

Another trick you can use is the following. Suppose that you want to select a range of numbers for each execution, then you can use the following example.

```
Dim sn As Single =5.6
Select Case True
  ' The following is the only Case clause that evaluates to True.
   Case  sn > 1 And sn < 5
    Debug.WriteLine("Between 1 and 5")
   Case sn > 6 And sn < 8  ' Number between 6 and 8.
    Debug.WriteLine("Between 6 and 8")
  Case sn > 9 And sn < 10   ' Number is 9 or 10.
    Debug.WriteLine("Greater than 8")
  Case Else   ' Other values.
    Debug.WriteLine("Not between 1 and 10")
End Select
```

1.6.3. For/Do loops:

```
Dim i, c As Integer
For i=0 to 10  ‘ 11 loops
   c += 1
Next

‘ loop index can be defined within the loop, then the scope of variable is only valid within the loop
For i as Integer = 0 to 10
   c += 1
Next
```

For-Each loop can be used for all of the elements in an array or a collection.

```
Dim ar() As Integer = {1, 2, 3}, i As Integer
For Each i in ar
     MsgBox( Cstr(i) )
Next
```

In Do … Loop structure, While or Until tests can be added at the beginning or end.

```
Do While x  >  0
  x = x \ 2
Loop
‘
Do
  x = x \ 2
Loop While x > 0
‘
Do
  x = x \ 2
Loop Until x < 0
‘
```

1.7 Commands

To run “Notepad.exe” from your program and wait until the user terminates it, use the Shell command.

```
Shell (“notepad”, AppWinStyle.NormalFocus, True)
```

Running a Notepad this way is inconvenient, since every other function has to wait. You can make it only wait until a certain amount of time.

```
‘ Run Notepad, and then wait only 5 seconds and then move on to the next statements
Dim taskID As Long
taskID = Shell(“notepad”, AppWinStyle.NormalFocus, True, 5000)
If taskID = 0 Then
    MsgBox (“Notepad was closed within 5 seconds”)
```

```
Else
    MsgBox (“Notepad is still running.”)
End If

‘  Open a text file in the notepad
Shell (“notepad filename.txt”, AppWinStyle.NormalFocus, True, 100)
```

1.8 Error Handling

```
Dim x, y as Single
Try
  x = x/y
Catch ex as Exception
  MsgBox ( ex.Message )
End Try
```

Throwing an exception

```
Throw New System.IO.FileNotFoundException()
‘
‘ or
Dim msg As String = “File Not Found”
Throw New System.IO.FileNotFoundException(msg)
```

This statement is equivalent to the old way of raising error. The following is still valid code, but should be avoid.

```
Err.Raise 345,  , “File not found”
```

1.9 String Functions

Search string
IndexOf

2. Files, Directories, Stream

Always include the following name space.

```
Imports System.IO
```

2.1 Files and Stream

2.1.1 Old way but convenient way of saving/retrieving binary data

The following code saves a simple 3x3 matrix in a binary format.

```
Dim fn As Integer = FreeFile()
      FileOpen(fn, Application.StartupPath & "\data.bin", OpenMode.Binary, OpenAccess.Write)
      Dim mat(,) As Integer = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}}
```

```
    FilePut(fn, mat)  ' Using FilePutObject(fn, mat) is better and ensures to save the object
                      ' information
    FileClose(fn)
```

The saved data can be retrieved using "FileGet()".

```
Dim fn As Integer = FreeFile()
    FileOpen(fn, Application.StartupPath & "\data.bin", OpenMode.Binary, OpenAccess.Read)
    Dim mat(2, 2) As Integer
    FileGet(fn, mat)   'reads the original data back
    FileClose(fn)
    Dim i, j As Integer
    For i = 0 To 2
       For j = 0 To 2
          txt.Text &= mat(i, j) & " "  'txt is a textbox
       Next
       txt.Text &= vbCrLf
    Next
```

The main caution during the retrieval should be given to the specification array dimensions and sizes. They must exactly match with the original dimensions and sizes stored, otherwise, it causes an error. To avoid this, you can use FilePutObject() and FileGetObject().

2.1.2 Using File Stream

```
'Read a line from a text file
Dim sr As New StreamReader("C:\file.txt")
Dim txtData As String = sr.ReadLine
'You can also read until the end of file using seek
Do Until sr.Peek = -1
   txtData += sr.ReadLine
Loop
'At the end make sure you close the stream
sr.Close()

'Write a text file
Dim sw As New StreamWriter(Application.StartupPath + "\file.txt")
sw.Write("This is a test")
sw.Close()
```

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.

```
Dim st As Stream = File.Open("C:\test.dat", FileMode.Create, FileAccess.Write)
Dim bw As New BinaryWriter(st)

For i as Integer = 1 to 10
     bw.Write ( data(i) )
Next
bw.Close()

' Reading back the data
Dim st As Stream = File.Open("C:\test.dat", FileMode.Open, FileAccess.Read)
```

```
Dim br As New BinaryReader(st)

Do Until br.PeekChar=-1
      data(i) = br.ReadDouble
Loop
br.Close()
st.Close()
```

2.1.3 Reading and Writing from Strings

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

```
Dim LString, s As String
Dim strR As New StringReader(LString)
Do Until strR.Peek = -1
          s = strR.ReadLine
Loop
```

For writing StringWriter class is used.

2.2 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”.

```
For Each fname As String In Directory.GetFiles(“C:\myfiles”, “*.txt”)
      Textbox.Text = fname + vbCrLF
Next
```

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

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

2.3 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.

```
Dim dname, fname As String
For Each dname In Directory.GetDirectories(txtFolder.Text)
   txt1.Text += dname + vbCrLf
   For Each fname In Directory.GetFiles(dname, "*.txt")
      txt1.Text += fname + vbCrLf
    Next
Next
```

2.4 Extraction of Path and filename

In the process of file and path handling, we often need to extract the filename or path only 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
```

2.5 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
Dim dirPath As String
dirPath = Application.StartupPath + "\Daylets"
If Not Directory.Exists(dirPath) Then
    Directory.CreateDirectory(dirPath)
End If
```

The whole directory is deleted by,

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

3. Frequently Used Utilities

3.1 Variable Type Conversions(Casting)

There are several three different ways of converting one type to another, for example, converting a string to an integer. The first method is using the old VB conversion routines that are still available. These are very convenient, and I encourage you to use them. Below are the complete list.

```
CBool(expression)
CByte(expression)
CChar(expression)
CDate(expression)
CDbl(expression)
CDec(expression)
CInt(expression)
CLng(expression)
CObj(expression)
CShort(expression)
CSng(expression)
CStr(expression)
```

and here is an example.

```
‘ convert string “12.5” to single 12.5
Dim s As Single = CSng(“12.5”)
```

The second method uses the CType(expression, typename) conversion function. The advantage is that you don't have to remember the conversion name, but select it from the pop up list in the editor. CType works with both reference and value types.

```
Dim s As Single =  CType("12.5", Single)
```

The third method is using the DirectCast keyword. DirectCast only works with references.

```
Dim s As Single =  DirectCast ("12.5", Single)
```

Since string is a reference, it works. However, the following fails.

```
Dim Q As Object = 2.37   ' Requires Option Strict to be Off.
Dim J As Integer = DirectCast(Q, Integer)   ' Fails.
Dim I As Integer = CType(Q, Integer)   ' Succeeds.
```

3.2 Splitting a string into an array of strings

Often we need to divide a string according to delimiters in order to control each element. For that the Split() command is very convenient.

```
Dim sa() as String
Dim s as String = "d1, d2, d3"
sa = Split(s, ",")
' sa then contains sa(0) = "d1"  sa(1) = "d2" sa(2)="d3"
```

3.3 OpenFileDialog/FolderBrowserDialog/Set Attributes/Set Access Time

It is convenient when you want users to choose files using a built in dialog windows.

```
' 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"
If OpenFileDialog1.ShowDialog() = DialogResult.OK Then
     fname = OpenFileDialog1.FileName
Else
    Exit Sub
End If
```

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 Then
         textbox1.Text += fbDialog.SelectedPath
 End If
```

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.

```
If fbDialog.ShowDialog = DialogResult.OK Then
        selectedPath = fbDialog.SelectedPath
        txtOutput.Text += selectedPath + vbCrLf    'display the list of files in that directory

        Dim fname As String
        Dim attr As FileAttributes
        For Each fname In Directory.GetFiles(selectedPath, "*.*")
           txtOutput.Text += fname + vbCrLf
           attr = File.GetAttributes(fname)
           attr = attr Or FileAttributes.ReadOnly  'attributes are bits and must use Or to maintain
the rest of existing setting
           File.SetAttributes(fname, attr)
        Next
 End If
```

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

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

3.4 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()
```

3.5 Form-to-Form Communication Using Events

Since the form traps its own events within the form class, it is tricky to trap the event of other forms. For example, when a peripheral form is closed which collected input from the user, we wish take an action from the main form based on the results obtained from the peripheral form. I find that using event handling techniques work nicely for this purpose. Read the section “4. Generating and Trapping Events” for the basic technique.

3.6 Run Notepad from a program at run time.

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

```
Imports System.Diagnostics
      Dim proc As New Process
      proc.StartInfo.FileName = "Notepad.exe"
      proc.StartInfo.Arguments = Application.StartupPath + "\TToutput.txt"
```

```
proc.Start()
```

3.7 Ubound() of an array.

Frequently, you will need to know the number of elements in an array. In particular, if an array is passes to a subroutine and does not use a fixed length, the number of elements in the array is important information. The utility routine Ubound() provides the highest available index of the indicated dimension. An example is given below.

```
 Dim Highest, MyArray(10, 15, 20), AnyArray(6) as Integer
Highest = UBound(MyArray, 1)   ' Returns 10.
Highest = UBound(MyArray, 2)   ' Returns 15
Highest = UBound(MyArray, 3)   ' Returns 20.
Highest = UBound(AnyArray)   ' Returns 6.
```

4. Generating and Trapping Events

4.1 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.,

```
Dim btnExit As New System.Windows.Forms.Button
```

Next, you add event handler that processes the btnExit click event as:

```
AddHandler btnExit.Click, Addressof ProcBtnExit
```

The event handling routine can then be written using the address defined by the AddHandler as:

```
Sub ProcBtnExit(ByVal sender as Object, ByVal e as EventArgs)
   Application.Exit()
End Sub
```

If you use the design time GUI to generate a click event, Windows generates the event handling routine as:

```
Private Sub btnExit_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles btnExit.Click
      your code
End Sub
```

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

```
btnExt(Me, e)
```

4.2 Creating and Trapping Custom Events

In the previous example (4.1), the event was created by Win32 control. Sometime, you will need to create your own event and the corresponding event handler. It can be done using the following example.

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

```
Class DataCollection
    Event GotData(ByVal data as String)   'define event
    '
    ' Data collection unit
   Sub RecData
       'read it from a source such as a serial port
        Dim msg As String = ReadLineFromSerial()
        RaiseEvent GotData(msg)
   End Sub
End Class
```

This class is capable of generating an event whenever data is received from the serial port assuming that ReadLineFromSerial() 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 Class Form1
   Inherits System.Windows.Forms.Form

  Public Sub New()
      MyBase.New()
      'This call is required by the Windows Form Designer.
      InitializeComponent()
      'Add any initialization after the InitializeComponent() call
      AddHandler dc.GotData, AddressOf OnReceive
      Timer1.Enabled = True
   End Sub
   ' lines of codes generated by Windows will be here

   Public dc As New DataCollection

   Private Sub OnReceive(ByVal msg As String)
         MsgBox(msg)
   End Sub
End Class
```

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

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:

```
Imports System.Drawing
Imports System.Drawing.Drawing2D
Imports  System.Drawing.Imaging
Imports 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 Sub Form1_Paint(ByVal sender As Object, ByVal e As
System.Windows.Forms.PaintEventArgs) Handles MyBase.Paint
    'Get graphics object for the form's surface
    Dim gr As Graphics = e.Graphics
    gr.DrawEllipse(Pens.Red, 0, 0, Me.ClientSize.Width, Me.ClientSize.Height)
  End Sub
```

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 Sub Form1_Resize(ByVal sender As Object, ByVal e As System.EventArgs) Handles
MyBase.Resize
    Dim gr As Graphics = Me.CreateGraphics
    gr.DrawEllipse(Pens.Red, 0, 0, Me.ClientSize.Width, Me.ClientSize.Height)
    gr.Dispose()
    Me.Refresh()
  End Sub
```

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

5.2 Imaging

5.2.1 Loading and Saving Images

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

6. Regular Expression

```
Imports System.Text.RegularExpressions
```

Regular expressions are awkward and contorted looking language, but can be useful. For example, the following removes digits followed by an "a" character.

```
Dim data As String = "a1  a2   a3"
Dim rex As New Regex("a\d")
Writeline(rex.Replace(data, "a"))   ' should print, "a  a  a"
```

Suppose you wish to import data from a semicolon delimited file. The data looks like the following line.

```
"Andrew";"Fuller";1/19/1952;"908 W. Capital Way";"Tacoma"
```

```
Dim re As New
Regex("""(?<fname>[^""]+)"";""(?<lname>[^""]+)"";(?<bdate>[^;]+);"&"""(?<addr>[^""]+)"";""(<city>[^"
"]+)"""
Dim ma As Match
Dim table(,) As String

i=0
For Each ma In re.Matches(fileText)

    table(i, 0) = ma.Groups("fname").Value
    table(i, 1) = ma.Groups("lname").Value
    table(i, 2) = ma.Groups("bdate").Value
    table(i, 3) = ma.Groups("addr").Value
    table(i, 4) = ma.Groups("city").Value
     i +=1
Next
```

As you see, the syntax of the regular expression looks dizzy but not overly complex.

The main syntax for the above example is:

(?<name>substr)	Means, capture the substring and assign it a name. The name must not contain any punctuation symbols.
[^""]	Means, any character except those in the list between the square bracket.
+	Means one or more matches

The syntax of regular expressions can be easily learned, but I recommend to avoid it in Windows programming if possible. It unnecessarily adds complexity in reading the code.

The above could have easily accomplished using simple coding as below.

```
Dim items() As String
items = Split(TextLine, ";")
```

And then simply you could remove the unnecessary characters using the String object.

7. Threading

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

```
Imports 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.

```
Dim th As New Thread(New ThreadStart(AddressOf GetData))
th.Start()    'run the sub GetData as a new thread
              'This thread, th, is terminated when it exits the GetData() sub.
              'If you wish to force the termination, you can issue th.Abort().
              'It is important to remember that threads start asynchronously, i.e., it may not
              'immediately  start executing the sub GetData().
```

Somewhere in your code should have the subroutine for GetData()

```
Sub GetData()
    Dim s as String
    GetSerialData(s)
    PassToBuffer(s)
    Thread.CurrentThread.Sleep(10)  ' wait for 10ms
End Sub
```

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, SyncLock statements is used. For the above example, SyncLock 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.

```
Sub GetData()
    Dim s as String
    SyncLock Me
        GetSerialData(s)
```

```
        PassToBuffer(s)
    End SyncLock
    Thread.CurrentThread.Sloop(10)  ' wait for 10ms
End Sub
```

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

```
Dim m As New Mutex
Sub GetData()
    Dim s as String
    m.WaitOne()
        GetSerialData(s)
        PassToBuffer(s)
    m.ReleaseMutex()
    Thread.CurrentThread.Sloop(10)  ' wait for 10ms
End Sub
```