UNIT-5

Framework Class Basics

Array List Collection

- It allows you to maintain multiple elements
- It allows the insertion of elements anywhere in the collection and removal of any element.
- It is a dynamic structure.
- That can be grow automatically as you add elements
- Array list's element can be sorted and searched.
- You can remove element by value, not only by index
- It is not strictly typed.

❖ **Creating an ArrayList**

- To use an arraylist ,you must first create an instance of the arraylist class with the new keyword.
- When you declare an arraylist you need to specify any dimension.
- **Syntax:** Dim alist as new ArrayList.
- Alist variable can hold only 16 elements (default size).
- You can set the initial capacity of the arraylist by setting capacity property.

Capacity Property

Use: to set initial capacity of ArrayList.

Syntax: Alist . Capacity = value

Parameter: Alist: instance of arraylist class

Value: numeric value

Count Property

Use : to find exact number of items in the arraylist.which is always less than or equal to the capacity property.

Syntax: Count=alist.count

Parameter: Count: integer variable store number of items

Example

```
Private Sub Button1_Click(…)
Dim myList As New ArrayList()
myList.Add("item1")
myList.Add("item2")
Dim c As Integer
c = myList.Count
MsgBox("You entered " & c & " ITEMS.")
End Sub
```

- **Add Method**
- **Use :** To add items in ArrayList
- Add method appends the specified item to the collection and returns the index of the new item
- **Syntax:** Index=alist . Add(object)
- **Parameter:** Alist :properly declared arraylist.
  Object: items you want to add to arryaList
  index: index of new item
- **Return:** Index of new item
- **Example**

```
Private Sub Button1_Click(...)
Dim a As Integer
Dim ItemList As New ArrayList()
ItemList.Add("Ball")
ItemList.Add("Balloon")
ItemList.Add("Button")
MsgBox("Shows Added Items")
For a = 0 To ItemList.Count - 1
MsgBox(ItemList.Item(a))
Next
End Sub
```

Contains Method

Use to find out whether an item belongs to the collection already or not.

Syntax: Alist.Conatains (object)

Parameter: Alist: insatance of ArrayList

Object: items you want to check.

Return: Boolean value.

Example

```
Private Sub Button1_Click(…)
Dim alist1 As New ArrayList()
alist1.Add("Hello")
alist1.Add(714)
alist1.Add(#1/1/2003#)
alist1.Add(4.5)
alist1.Add(4.5F)
If alist1.Contains("Hello") Then
    MsgBox("item is already exists")
Else
    alist1.Add("hello")
End If
End Sub
```

Insert Method

Use: to insert an item at specific location.

Syntax: Alist .insert (index , object)

Parameter: Alist : instance of arraylist collection

Index:zero based index at which specified value should be inserted

Object:item to be inserted

Example

```
Private Sub Button1_Click(…)
        Dim a As Integer
        Dim ItemList As New ArrayList()
        ItemList.Add("Ball")
        ItemList.Add("Balloon")
        ItemList.Add("Button")
        ItemList.Add("Bus")
        ItemList.Add("Bucket")
        MsgBox("Shows Added Items")
        ItemList.Insert(2, "Bottle")
        For a = 0 To ItemList.Count - 1
            MsgBox(ItemList.Item(a))
        Next
    End Sub
```

AddRange Method

Use: to add multiple items with a single call.

It appends a collection of items to the arraylist.

Syntax: Alist .addrange(objects)

Parameter: Alist: instance of Arraylist collection

Objects:array or arraylist whose items should be added to the end of arraylist

Example

```
Private Sub Button1_Click(…)
    Dim list1 As New ArrayList
    list1.Add(5)
    list1.Add(7)
    Dim list2 As New ArrayList
    list2.Add(10)
    list2.Add(13)
    list1.AddRange(list2)
    Dim num As Integer
    For Each num In list1
      MsgBox(num)
    Next
  End Sub
End Class
```

Remove Method

Use: to remove items by specifying their object value

- Arraylist allows you to remove item only by value not by index.
- if collection contains multiple instance of the same item ,only the first instance of the object will be removed.
- If you attempt to remove an item that doesn't exist ,an exception will be thrown

Syntax: Alist .remove (object)

Parameter: Alist: instance of arraylist collection

Object: value to be removed.

Example

```
Private Sub Button1_Click(…)
        Dim a As Integer
        Dim ItemList As New ArrayList()
        ItemList.Add("Ball")
        ItemList.Add("Balloon")
        ItemList.Add("Button")
        ItemList.Add("Bus")
        ItemList.Add("Bucket")
        MsgBox("Shows Added Items")
        ItemList.Remove("Bus")
       For a = 0 To ItemList.Count - 1
            MsgBox(ItemList.Item(a))
        Next
    End Sub
```

RemoveAt Method

Use: to remove item by specifying index value

Syntax: Alist. Removeat (index)

Parameter: Alist: instance of arraylist

Index: index of item to be removed, must be less than the number of items currently in the list.

Example

```
Private Sub Button1_Click(…)
        Dim list As New ArrayList
        list.Add("Dot")
        list.Add("Net")
        list.Add("Perls")
        list.RemoveAt(1)
        Dim str As String
        For Each str In list
            MsgBox(str)
        Next
    End Sub
```

RemoveRange Method

Use: to remove more than one item.

Syntax: Alist.removerange(startindex , count)

Parameter: Alist: instance of arraylist

Startindex:index of the first item to be removed

Count:number of element to remove

Example

```
Private Sub Button1_Click(…)
        Dim list As New ArrayList
        list.Add("item1")
        list.Add("item2")
        list.Add("item3")
        list.Add("item4")
        list.Add("item5")
        list.RemoveRange(1, 2)
        Dim str As String
        For Each str In list
            MsgBox(str)
        Next
    End Sub
```

- **GetRange Method**
- It extracts a number of elements from the arraylist and stores them to a new arraylist.
- **Syntax:**

  Newlist=arraylist.getrange(index , count)
- **Parameter:**Newlist: array that store element from the arraylist

  Index: index of the first item to copy

  Count: number of items to copy
- **Example**

```
Private Sub Button1_Click(...)
    Dim list1 As New ArrayList
    list1.Add("item1")
    list1.Add("itme2")
    list1.Add("item3")
    list1.Add("item4")
    list1.Add("item5")
    list1.Add("item6")
    Dim list2 As New ArrayList
    list2 = list1.GetRange(2, 3)
    Dim str As String
    For Each str In list2
        MsgBox(str)
    Next
End Sub
```

- **Repeat**
- It fills an arraylist with multiple instance of the same item.
- **Syntax:**

  Newllist=alist.repeat (item ,count)
- **Parameter:** Item: items to repeat

  count: number of times values should be copied.
- **Example**

```
Private Sub Button1_Click(...)
    Dim list1 As New ArrayList
    list1.Add("a")
    list1.Add("d")
    list1.Add("f")
    list1.Add("b")
    list1.Add("p")
    list1.Add("c")
    Dim list2 As New ArrayList
    list2 = System.Collections.ArrayList.Repeat("f", 3)
    Dim c As Integer
    For c = 0 To list2.Count - 1
        MsgBox(list2.Item(c))
    Next
```

Reverse

It reverse the order of the elements in an arraylist

Syntax: alist .reverse ( )

Parameter: Arrayname : name of array to reverse

Return: Array in reverse order

Example

```
Private Sub Button1_Click(…)
        Dim list1 As New ArrayList
        list1.Add("item1")
        list1.Add("itme2")
        list1.Add("item3")
        list1.Add("item4")
        list1.Add("item5")
        list1.Add("item6")
        list1.Reverse()
        Dim c As Integer
        For c = 0 To list1.Count - 1
            MsgBox(list1.Item(c))
        Next
    End Sub
```

Searching and sorting Array

- Arraylists can be search in two ways:
- (1) Binary Search

  It work on sorted list ,is very fast.
- (2) IndexOf and LastIndexOf

  It works on unsorted list .

  All three methods return an index of element.

- **IndexOf**
- **Use:** to search for particular elements in array.
- **Syntax:**

  Index= alist.IndexOf(Object)

  Index=alist.indexOf(objects, startindex)
- **Parameter:** Index: integer variable store index value

  Objects :item you are searching for.

  Startindex: Starting index of search
- **Return:**

  Index of elements in array.

  -1 if objects is not found
- **Example**

```
Private Sub Button1_Click(...)
    Dim list1 As New ArrayList
    list1.Add("item1")
    list1.Add("itme2")
    list1.Add("item3")
    list1.Add("item4")
    list1.Add("item5")
    list1.Add("item6")
    Dim index As Integer
    index = list1.IndexOf("item5")
    MsgBox(index)
 End Sub
```

Contd..

```
Private Sub Button1_Click(...)
        Dim list1 As New ArrayList
        list1.Add("item1")
        list1.Add("itme2")
        list1.Add("item3")
        list1.Add("item4")
        list1.Add("item5")
        list1.Add("item6")
        Dim index As Integer
        index = list1.IndexOf("item5“,3)
        MsgBox(index)
    End Sub
```

LastindexOf

Use: to search for particular elements in array.

It start search from end of the array.

Syntax: Index=alist.LastIndexOf(Object , StartIndex)

Parameter: Index: integer variable store index value

Objects: item you are searching for.

StartIndex: index value to start search.

Return: Index of elements in array.

-1 if objects is not found

Example

```
Private Sub Button1_Click(…)
    Dim list1 As New ArrayList
    list1.Add("item1")
    list1.Add("itme2")
    list1.Add("item3")
    list1.Add("item4")
    list1.Add("item5")
    list1.Add("item6")
    Dim index As Integer
    index = list1.LastIndexOf("item5“,3)
    MsgBox(index)
  End Sub
```

Searching in the segment of array

Use : to search element from specific segment.

Syntax:

Index=alist .indexOf(objects , startindex, count)

Index=alist.LastindexOf(objects , startindex , count)

Parameter: Index: integer variable store index value

Objects: item you are searching for.

Startindex: Starting index of search

Count: number of element in the section to search

Return:

Index of elements in array.

-1 if objects is not found

Example

```
Private Sub Button1_Click(…)
    Dim list1 As New ArrayList
    list1.Add("item1")
    list1.Add("itme2")
    list1.Add("item3")
    list1.Add("item4")
    list1.Add("item5")
    list1.Add("item6")
    Dim index As Integer
    index = list1.LastIndexOf("item5“,3,5)
    MsgBox(index)
  End Sub
```

Binary Search

It work with sorted arraylist.

Syntax: Index =alist.binarySearch(object)

It search items from arraylist with same datatype

Parameter:

alist : instance of arraylist

Object: item you are searching for

Return:

Integer value ,which is the Index of elements you are searching for in the array.

If the object value is not found it return a negative value.

Which is the negative of the index of the next larger item minus one.

Example

```
Private Sub Button1_Click(…)
    Dim list1 As New ArrayList
    list1.Add("a")
    list1.Add("d")
    list1.Add("f")
    list1.Add("b")
    list1.Add("p")
    list1.Add("c")
    list1.Sort()
    Dim index As Integer
    index = list1.BinarySearch("f")
    MsgBox(index)
  End Sub
```

Sort

Use : to sort elements in arraylist.

Syntax: Alist.sort()

Sort the arraylist alphabetically or numerically ,depending on the datatype of the objects.

Example

```
Private Sub Button1_Click(...)
        Dim list1 As New ArrayList
        list1.Add("a")
        list1.Add("d")
        list1.Add("f")
        list1.Add("b")
        list1.Add("p")
        list1.Add("c")
        list1.Sort()
        Dim c As Integer
        For c = 0 To list1.Count - 1
            MsgBox(list1.Item(c))
        Next
    End Sub
```

HashTable Collection

- Hashtable collection is similar to arraylist , but it allows you to access the items by a key.
- Each item has a value and key.
- The value is the same value you store in an array, key is meaningful entity for accessing the items in the collection.
- Hashtable collection adjust its capacity automatically.
- It create automatically a unique key for each item.
- This key is derived from the item being added, and it's possible that two items will produce the same key.
- You need not import any class to use a hashtable.
- Just declare hashtable variable
- **Declare HashTable Variable**

Dim htable as new HashTable

Property & method

- Hashtable exposes most of the properties and methods of the arraylist.
- Count Property :It returns number of elements in hashtable
- **Syntax:**
  - Item = htable.count
- **Parameter:**
  - Item : integer variable store number of elements
  - Htable: instance of hashtable class

Example

```
Private Sub Button1_Click(…)
    Dim htable As New Hashtable
    htable.Add(16, "item3")
    htable.Add(10, "item9")
    htable.Add(15, "item4")
    htable.Add(17, "item2")
    htable.Add(11, "item8")
    htable.Add(14, "item5")
    htable.Add(12, "item7")
    Dim item As Integer
    item = htable.Count
    MsgBox(item)
 End Sub
```

Values

- **Use:** to retrieve all the values in the hashtable.
- **Syntax:**
  - Htable.values()
- **Parameter:**
  - Htable: instance of hashtable collection
- **Return:**
  - Values from hashtable collection

Example

```
Private Sub Button1_Click(…)
  Dim htable As New Hashtable
        htable.Add(16, "item3")
                    htable.Add(15, "item4")
   htable.Add(17, "item2")
   htable.Add(11, "item8")
   htable.Add(14, "item5")
   htable.Add(12, "item7")
      Dim i As Integer
   For i = 0 To htable.Count - 1
     MsgBox(htable.Values(i))
  Next
 End Sub
End Class
```

Keys

- **Use:** to retrieve all the keys in the hashtable
- **Syntax:**
  - Htable.keys()
- **Parameter:**
  - Htable: instance of hashtable class
- **Return":**
  - keys from hashtable collection

Example

```
Private Sub Button1_Click(…)
Dim htable As New Hashtable
        htable.Add(16, "item3")
        htable.Add(15, "item4")
        htable.Add(17, "item2")
        htable.Add(11, "item8")
        htable.Add(14, "item5")
        htable.Add(12, "item7")

Dim i As Integer
 For i = 0 To htable.Count - 1
   MsgBox(htable.keys(i))
Next
End Sub
```

Add Method

- **Use:** to add item to the Hashtable collection
- **Syntax:**
  - Htable. Add (key , value)
- **Parameter:**
  - Key: is a key of the element to add
  - Value: item you want to add.
  - You can have duplicate values ,but key must be unique.

Example

```
Private Sub Button1_Click(…)
 Dim htable As New Hashtable
        htable.Add(16, "item3")
        htable.Add(15, "item4")
         htable.Add(17, "item2")
         htable.Add(11, "item8")
         htable.Add(14, "item5")
         htable.Add(12, "item7")
End Sub
```

ContainsKey Method

- **Use:** to find out existence of key in hashtable collection
- **Syntax:**
  - Htable. ContainsKey (object)
- **Parameter:**
  - Object: Key to locate in the hashtable collection

Example

```
Private Sub Button1_Click(…)
  Dim htable As New Hashtable
  htable.Add(16, "item3")
  htable.Add(15, "item4")
  htable.Add(17, "item2")
  htable.Add(11, "item2")
  htable.Add(14, "item5")
  htable.Add(12, "item7")
  If htable.ContainsKey(17) Then
     MsgBox("alredy exist")
  Else
     htable.Add(17, "item2")
  End If
End Sub
```

- **Containsvalue Method**
- **Use:** to find out existence of value in hashtable collection.
- **Syntax:** Htable. ContainsValue(object)
- **Parameter:** Object: Value to locate in the hashtable collection.

- **Example**

```
Private Sub Button1_Click(...)
 Dim htable As New Hashtable
 htable.Add(16, "item3")
 htable.Add(15, "item4")
 htable.Add(17, "item2")
 htable.Add(11, "item2")
 htable.Add(14, "item5")
 htable.Add(12, "item7")
 If htable.Containsvalue(“item2”) Then
    MsgBox("alredy exist")
 Else
    htable.Add(17, "item2")
 End If
End Sub
```

Remove

- **Use:** to remove items from a hashtable collection
- **Syntax:**
  - Htable.remove (key)
- **Parameter:**
  - Key: key of the element to be removed
  - htable: instance of hashtable class

Example

```
Private Sub Button1_Click(…)
    Dim htable As New Hashtable
    htable.Add(16, "item3")
    htable.Add(15, "item4")
    htable.Add(17, "item2")
    htable.Add(11, "item2")
    htable.Add(14, "item5")
    htable.Add(12, "item7")
    htable.Remove(12)
    For Each itm In htable.Values
       MsgBox(itm)
    Next
End sub
```

CopyTo

- **Use:** to extract item from a hashtable collection
- It copies the items to a one-dimensional array.
- **Syntax:**
  - Htable.values.copy(arrayname,index)
- **Parameter:**
  - Index: zero based index at which copying begins
  - htable: instance of hashtable class
  - Arrayname: destination array name copy element

Example

```
Dim htable As New Hashtable
htable.Add(1, "sun")
htable.Add(2, "mon")
htable.Add(3, "tue")
Dim s(2) As String
htable.Values.CopyTo(s, 0)
Array.Sort(s)
Dim itm As String
For Each itm In s
   MsgBox(itm)
Next
```

SortedList Class

- It is combination of the Array and HashTable classes.
- Item can be accessed either with an index or with a key.
- Collection is always sorted according to the keys.
- The items of a sortedlist are ordered according to the values of their keys , there is no method for sorting the collection according to the values stored in it.

Create a new SortedList collection

Dim slist as new SortedList

Capacity Property

Use : to set initial Capacity of sorted list

Syntax:

Slist .capacity = value

Parameter: Slist : instance of SortedList Collection

Value: numeric capacity value

Count Property

Use : to find exact number of items in the sortedlist collection

Syntax:

Count=slist.count

Parameter:

Count: integer variable store number of items

Return:

Number of items in collection

Example

```
Private Sub Button1_Click(...)
  Dim slist  As New sortedlist
        slist.Add(16, "item3")
        slist.Add(10, "item9")
        slist.Add(15, "item4")
        slist.Add(17, "item2")
        slist.Add(11, "item8")
        slist.Add(14, "item5")
        slist.Add(12, "item7")
  Dim item As Integer
  item = slist.Count
  MsgBox(item)
End Sub
```

Add Method

- **Use:** to add item to the sortedlist collection
- **Syntax:**
  - Slist. Add (key , value)
- **Parameter:**
  - Key: is a key of the element to add
  - Value: item you want to add.
  - Both arguments are objects.
  - You can have duplicate values ,but key must be unique.

Example

```
Private Sub Button1_Click(…)
    Dim slist  As New sortedlist
                    slist.Add(16, "item3")
                    slist.Add(10, "item9")
                    slist.Add(15, "item4")
                    slist.Add(17, "item2")
                    slist.Add(11, "item8")
                    slist.Add(14, "item5")
                    slist.Add(12, "item7")
End Sub
```

ContainsKey Method

- **Use:** to find out existence of key in sortedList collection
- **Syntax:**
  - slist. ContainsKey(object)
  - **Parameter**Object:Key to locate in the hashtable
  - **Return:** Boolean value

Example

```
Private Sub Button1_Click(…)
    Dim slist  As New sortedlist
                    slist.Add(16, "item3")
                    slist.Add(10, "item9")
                    slist.Add(15, "item4")
                    slist.Add(17, "item2")
                    slist.Add(11, "item8")
                    slist.Add(14, "item5")
                    slist.Add(12, "item7")
    If slist.ContainsKey(14) Then
       MsgBox("alredy exist")
    Else
       slist.Add(14,”item5”)
    End If
  End Sub
```

Containsvalue Method

- **Use:** to find out existence of value in SortedList collection
- **Syntax:**
  - slist. ContainsValue(object)
- **Parameter:**
  - Object: Value to locate in the sortedList collection
- **Return:**
  - Boolean value

Example

```
Private Sub Button1_Click(…)
    Dim slist  As New sortedlist
                        slist.Add(16, "item3")
                        slist.Add(10, "item9")
                        slist.Add(15, "item4")
                        slist.Add(17, "item2")
                        slist.Add(11, "item8")
                        slist.Add(14, "item5")
                        slist.Add(12, "item7")
    If slist.Containsvalue(“item5”) Then
       MsgBox("alredy exist")
    Else
       slist.Add(14,”item5”)
    End If
  End Sub
```

SetByIndex

- **Use :** to replace an existing item.
- **Syntax:**

  Slist. SetByIndex(index, item)
- **Parameter:**Index: index at which the value will be inserted

  Item: new item to be inserted in the collection.
- This object will replace the value that corresponds to the specified index.
- The key remains same.
- There is no method for replacing a key, you must first remove the item and then insert it again with its new key.

Example

```
Private Sub Button1_Click(…)
    Dim slist As New SortedList
    slist.Add(16, "item3")
    slist.Add(10, "item9")
    slist.Add(15, "item4")
    slist.Add(17, "item2")
    slist.Add(11, "item8")
    slist.Add(14, "item5")
    slist.Add(12, "item7")
    slist.SetByIndex(3, "item30")
    Dim c As Integer
    For c = 0 To slist.Count - 1
      MsgBox(slist.Values(c))
    Next
End Sub
```

Remove

- **Use:** to remove items item from an sortedlist
- **Syntax:**
  - slist.remove(key)
- **Parameter:**
  - Key: key of the element to be removed
  - slist: instancce of sortedlist class

Example

```
Private Sub Button1_Click(…)
 Dim slist As New SortedList
 slist.Add(16, "item3")
 slist.Add(10, "item9")
 slist.Add(15, "item4")
 slist.Add(17, "item2")
 slist.Add(11, "item8")
 slist.Add(14, "item5")
 slist.Add(12, "item7")
 slist.Remove(10)
 End Sub
```

RemoveAt

- **Use:** to remove items item from an sortedlist
- **Syntax:**
  - slist.remove(index)
- **Parameter:**
  - Index ;index of the element to be removed
  - slist: instancce of sortedlist class

Example

```
Private Sub Button1_Click(…)
 Dim slist As New SortedList
 slist.Add(16, "item3")
 slist.Add(10, "item9")
 slist.Add(15, "item4")
 slist.Add(17, "item2")
 slist.Add(11, "item8")
 slist.Add(14, "item5")
 slist.Add(12, "item7")
 slist.RemoveAt(4)
 End Sub
```

- **clear**
- **Use:** to remove All items from an sortedlist
- **Syntax:**
  - slist.clear()
- **Parameter:**
  - slist: instance of sortedlist class

- **Example**

```
Private Sub Button1_Click(...)
 Dim slist As New SortedList
 slist.Add(16, "item3")
 slist.Add(10, "item9")
 slist.Add(15, "item4")
 slist.Add(17, "item2")
 slist.Add(11, "item8")
 slist.Add(14, "item5")
 slist.Add(12, "item7")
 slist.clear()
 End Sub
```

TrimTosize

- **Use:** to restore its capacity to the default size 16
- **Syntax:**
  - slist.trimToSize
- **Parameter:**
  - slist: instance of sortedlist class

Example

```
Private Sub Button1_Click(…)
 Dim slist As New SortedList
 slist.Add(16, "item3")
 slist.Add(10, "item9")
 slist.Add(15, "item4")
 slist.Add(17, "item2")
 slist.Add(11, "item8")
 slist.Add(14, "item5")
 slist.Add(12, "item7")

 Slist.trimtozise()
 End Sub
```

IndexOfValue

- **Use:** to find the index of a value in the sorted list.
- Which accepts objects as an arguments
- It perform case sensitive search.
- **Syntax:**
  - Index = slist. IndexOfValue(object)
- **Parameter:**
  - slist: instance of sortedlist class
  - Object: item for which you want index
  - Index: integer variable store index value
- **Return:**
  - Index of value if object is exist
  - -1 if not exist
  - If same value appears more than once In the collection then it return the first instance of the value.

Example

```
Private Sub Button1_Click(…)
    Dim slist As New SortedList
    slist.Add(16, "item3")
    slist.Add(10, "item9")
    slist.Add(15, "item4")
    slist.Add(17, "item2")
    slist.Add(11, "item8")
    slist.Add(14, "item5")
    slist.Add(12, "item7")
    MsgBox(slist.IndexOfValue("item3"))
  End Sub
```

IndexOfKey

- **Use:** to find the index of a key in the sorted list.
- Which accepts key as an arguments
- It perform case sensitive search.
- **Syntax:**
  - Index = slist. IndexOfValue (key)
- **Parameter:**
  - slist: instance of sortedlist class
  - Key : key value for which you want index value
  - Index: integer variable store index value
- **Return:**
  - Index of value if object is exist
  - -1 if not exist
  - If same value appears more than once In the collection then it return the first instance of the value.

Example

```
Private Sub Button1_Click(…)
    Dim slist As New SortedList
    slist.Add(16, "item3")
    slist.Add(10, "item9")
    slist.Add(15, "item4")
    slist.Add(17, "item2")
    slist.Add(11, "item8")
    slist.Add(14, "item5")
    slist.Add(12, "item7")
    MsgBox(slist.IndexOfKey(17))
  End Sub
```

GetKey

- It allow you to retrieve the key for the corresponding index value from the sorted list.
- **Syntax:**
- Index: slist.GetKey(index)
- **Parameter:**
  - slist: instance of sortedlist class
  - index: index of the key to get
  - Index: integer variable store index value
  - **Return**: key for specified index in sorted list.

Example

```
Private Sub Button1_Click(…)
    Dim slist As New SortedList
    slist.Add(16, "item3")
    slist.Add(10, "item9")
    slist.Add(15, "item4")
    slist.Add(17, "item2")
    slist.Add(11, "item8")
    slist.Add(14, "item5")
    slist.Add(12, "item7")
    MsgBox(slist.Getkey(2))
  End Sub
```

Getvalue

- It allow you to retrieve the value for the corresponding index value from the sorted list.
- **Syntax:**
- Index: slist.GetValue(index)
- **Parameter:**
  - slist: instance of sortedlist class
  - index: index of the key to get
  - Index: integer variable store index value
  - **Return**: value for specified index in sorted list.

Example

```
Private Sub Button1_Click(…)
    Dim slist As New SortedList
    slist.Add(16, "item3")
    slist.Add(10, "item9")
    slist.Add(15, "item4")
    slist.Add(17, "item2")
    slist.Add(11, "item8")
    slist.Add(14, "item5")
    slist.Add(12, "item7")
    MsgBox(slist.GetValue(2))
  End Sub
```

GetKeyList

- Use: to retrieve the keys in a sortedlist collection and create another list.
- Syntax:
- Getlist = slist .getkeylist
- Parameter:
- Getlist : array that store keys value
- Slist: instance of sortedlist collection
- Return:All keys fro sorted list collection

Example

```
Private Sub Button1_Click(…)
        Dim slist As New SortedList
        slist.Add(16, "item3")
        slist.Add(10, "item9")
        slist.Add(15, "item4")
        slist.Add(17, "item2")
        slist.Add(11, "item8")
        slist.Add(14, "item5")
        slist.Add(12, "item7")
        slist.SetByIndex(3, "item30")
        Dim c As Integer
        Dim l As IList
        l = slist.GetKeyList
        For c = 0 To slist.Count - 1
            MsgBox(l(c))
        Next
    End Sub
```

GetValueList

- Use: to retrive the values in a sortedlist collection and create another list.
- Syntax:

  Getlist = slist .getValuelist
- Parameter:

  Getlist : array that store keys value

  Slist: instance of sortedlist collection
- Return: All values from sorted list collection

Example

```
Private Sub Button1_Click(…)
        Dim slist As New SortedList
        slist.Add(16, "item3")
        slist.Add(10, "item9")
        slist.Add(15, "item4")
        slist.Add(17, "item2")
        slist.Add(11, "item8")
        slist.Add(14, "item5")
        slist.Add(12, "item7")
        slist.SetByIndex(3, "item30")
        Dim c As Integer
        Dim l As IList
        l = slist.Getvaluelist
        For c = 0 To slist.Count - 1
            MsgBox(l(c))
        Next
    End Sub
```

Char Class

- Is used to manipulate individual character.
- It exposes properties and method to examine char value.
- **Properties**

  MaxValue: returns the largest character value.

  MinValue: returns the smallest character value.
- **Methods**
- Char class exposes method to manipulate single character.
- You need to import system. char namespace to use all the method and properties of char class
- Imports system.char

- **GetNumeric value**
- It returns a numeric value if called with an arguments that is a digit otherwise -1.
- **Syntax:**

  Char.getnumericvalue(c as char)
- **Parameter:**

  c: unicode character
- **Returns:** Numeric value
- **Example**

Dim ch as char =“5”

Msgbox(char.Getnumericvalue(ch))

----returns 5

Dim ch as char = “@”

Msgbox(Char.Getnumericvalue(ch))

--- returns -1

- **IsDigit**
- This method examine whether the specified character is a digit or character.
- **Syntax:**
  Char.Isdigit(c as char) as boolean
- **Parameter:**
  C :unicode character
- **Returns:**Boolean value

- **Example**

```
Private Sub Button1_KeyPress(...)
Dim  c as char
C=e.keychar
If char.isdigit(c) then
Msgbox("digit")
Else
Msgbox("not digit)
End if
End sub
```

- **IsLetter**
- This mehod returns a boolean value indicating whether the specified character is a letter or not.
- **Syntax:**

  Ch.Isletter(c as char) as boolean
- **Parameter:** C :unicode character
- **Returns:**

  Boolean value
- **Example**

```
Private Sub Button1_KeyPress (…)
Dim  c as char
C=e.keychar
If char.isletter(c) then
Msgbox(“letter”)
Else
Msgbox(“not letter)
End if
End sub
```

- **Isletterordigit**
- It check whether the specified character is a letter or digit.
- If allows the user to enter only character and digit. User can't insert symbol.
- **Syntax:**

  Char.Isletterordigit(c as char) as boolean
- **Parameter:**

  C :unicode character
- **Returns:**

  Boolean value
- **Example**

```
Private Sub Button1_KeyPress(...)
Dim  c as char
C=e.keychar
If char.isletterordigit(c) then
Msgbox("digit or letter")
Else
Msgbox("symbol")
End if
End sub
```

- **IsLower/IsUpper**
- It check whether specified character is lowercase or uppercase.
- **Syntax:**

  Char.Islower(c as char) as boolean

  Char.Isupper(c as char)as boolean
- **Parameter:** C :unicode character
- **Returns:** Boolean value
- **Example**

```
Private Sub Button1_KeyPress(...)
Dim ch As Char
ch = e.KeyChar
If Char.IsLower(ch) Then
MsgBox("lowercase")
Else
MsgBox("uppervase")
End If
End sub
```

- **IsNumber**
- It check whether specified character is a number or not.
- **Syntax:**

  Char.Isnumber(c as char) as boolean
- **Parameter:**

  C :unicode character
- **Returns:**

  Boolean value
- **Example**

```
Private Sub Button1_KeyPress(...)
dim ch As Char
ch = e.KeyChar
If Char.IsNumber(ch) Then
MsgBox("number")
Else
MsgBox("not number")
End If
End sub
```

- **IsPunctuation**
- It check whether the specified character is a punctuation mark or not.
- **Syntax:**
  Char.IsPunctuation(c as char) as boolean
- **Parameter:**
  C :unicode character
- **Returns:**
  Boolean value
- **Example**

```
Private Sub Button1_KeyPress(...)
Dim ch As Char
ch = e.KeyChar
If Char.Ispunctuation(ch) Then
MsgBox(“punctuation")
Else
MsgBox(“not punctuation")
End If
End sub
```

IsSeperator

- It check whether the charatcer is categorized as a separator
- space , newline character and so on
- Semicolon is not a seperator.

Syntax:

Char.IsSeperator(c as char) as boolean

Parameter:

C :unicode character

Returns:

Boolean value

Seperator:

)	closing parenthesis	:	colon
,	comma	!	Exclamation mark
(	Opening parenthesis	.	Period
#	pound sign		

Example

```
Private Sub Button1_KeyPress(…)
Dim ch As Char
ch = e.KeyChar
If Char.IsSeparator(ch) Then
MsgBox(“separator")
Else
MsgBox(“not separator")
End If
End sub
```

- **ToLower / Toupper**
- It converts arguments to a lowercase or uppercase character .
- **Syntax:**

  Char.ToLower(c as char) as char

  Char .ToUpper(C as char) as char
- **Parameter:**

  C :unicode character
- **Returns:**

  Lowercase if arguments is uppercase

  Uppercase if arguments is losercase
- **Example**

```
Private Sub Button1_Click(..)
Dim ch As Char
ch = "ABC"
Dim ch1 As Char
ch1 = Char.ToLower(ch)
MsgBox(ch1)
End sub
```

Working with String

- New .net framework provides two classes for manipulating text.
  - String class
  - Stringbuilder class
- **String Class**
  - The string object represent fixed length string, which you can not edit.
  - Once you assign a value to a string object,you can examine the string ,locate the word in it,parse it,but you can't edit it.
  - String class implements string data type
  - You need to import System.String namespace to manipulate string.
-
- **StringBuilder Class**
- It's similar to the string class,Only difference is that it can be manipulated in place.
- Difference between two classes is that the string class is for static strings and stringbuilder class is for dynamic strings

String Class Properties

- **Length**
- It retunes the number of character in the string .and it's a read only property to count number of character in a string.
- **Syntax:**
  String.Length() As Integer
  Length= str. length
- **Return:**
- Length: integer value representing length
  Integer : The number of characters in the specified String
- **Example**

```
Private Sub Button1_Click(..)
Dim str As String
str = "abcd“
Dim str1 As String
Str1 = str.Length
MsgBox(str1)
End sub
```

Chars

- The chars property is an array of characters that holds all the characters in the string.
- This property is used to read individual character form a string based on their location in the string.

Syntax: Character = chr .chars(index)

Example

```
Private Sub Button1_Click(..)
Dim str As String
str = "abcd"
Dim str1 As String
str1 = str.Chars(0)
MsgBox(str1)
End sub
```

- **Compare**
- **Use:** to compare two String Objects.
- **Syntax:** String.Compare(String str1,String str2)
  String.Compare(str1,str2,case)
  String.Compare(str1 ,index1,str2, index2,length)
- It allows you to compare segment of a string.
- **Parameters:** str1 : Parameter String
- str2 : Parameter String
- Case:Boolean True/False Indication to check the String with case sensitive or without case sensitive
- Index1 and index 2 : are string location of the segments to be compared in each string.
  Two segments must have same length which is specified by last arguments.
- **Returns:** returns less than zero, zero or greater than zero.
  Less than zero : str1 is less than str2
  zero : str1 is equal to str2
  Greater than zero : str1 is grater than str2
- **Example**

```
Public Class Form1
Private Sub Button1_Click(...)
Dim str1 As String
Dim str2 As String
str1 = "vb.net"
str2 = "VB.NET"
Dim result As Integer
result = String.Compare(str1, str2)
MsgBox(result)
result = String.Compare(str1, str2, True)
MsgBox(result)
End Sub
End Class
```

Concat

Use: to concat specified string object.

Syntax: String.Concat(str1 As String, str2 As String) As String

Parameters:

String str1 : Parameter String

String str2 : Parameter String

Returns:

A new String returns with str1 Concat with str2

Example

```
Public Class Form1
Private Sub Button1_Click(…)
Dim str1 As String
Dim str2 As String
str1 = "Concat() "
str2 = "Test"
MsgBox(String.Concat(str1, str2))
End Sub
End Class
```

Copy

Use: to create a new String object with the same content

Syntax: String.Copy(str As String) As String

Parameters:

str : The argument String for Copy method

Returns:

A new String as the same content of argument String

Example

```
Public Class Form1
Private Sub Button1_Click(…)
Dim str1 As String
Dim str2 As String
str1 = "VB.NET Copy() test"
Str2 = String.Copy(str1)
MsgBox(str2)
End Sub
End Class
```

- **EndsWith/startsWith**
- check if the Parameter String EndsWith or start with the Specified String .
- **Syntax:**

  String.EndsWith(String suffix) as Boolean

  String.startsWith(String suffix) as Boolean

  Parameters: suffix - The passing String for it EndsWith /startsWith
- **Returns:**
- Boolean - Yes/No
- If the String EndsWith /startsWith the Parameter String it returns True
- If the String doesnot EndsWith/Startswith the Parameter String it return False
- **Example**

```
Public Class Form1
Private Sub Button1_Click(...)
Dim str As String
str = "VB.NET TOP 10 BOOKS“
If str.EndsWith("BOOKS") = True Then
MsgBox("The String EndsWith 'BOOKS' ")
Else
MsgBox("The String does not EndsWith 'BOOKS'")
End If
End Sub
End Class
```

Insert

Will insert a String in a specified index in the String instance.

Syntax: String.Insert(Integer ind, String str) as String

Parameters:

ind - The index of the specified string to be inserted.

str - The string to be inserted.

Returns:

String - The result string.

Example

```
Public Class Form1
Private Sub Button1_Click(…)
Dim str As String = "This is VB.NET Test"
Dim insStr As String = "Insert “
Dim strRes As String = str.Insert(15, insStr)
MsgBox(strRes)
End Sub
End Class
```

Split

- Function returns an array of String containing the substrings delimited by the given System.Char array.

Syntax: Public Function Split(ByVal ParamArray separator() As Char) As String()

Parameters:

separator - the given delimiter.

Returns:

An array of Strings delimited by one or more characters in separator

Example

```
Public Class Form1
Private Sub Button1_Click(…)
Dim str As String
Dim strArr() As String
Dim count As Integer
str = "vb.net split test"
strArr = str.Split(" ")
For count = 0 To strArr.Length - 1 MsgBox(strArr(count))
Next
End Sub
End Class
```

- **Contains**
- **Use:** to check the specified parameter String exist in the String or not
- **Syntax:** String.Contains(String str) As Boolean
- **Parameters:**
- String str - input String for search
- **Returns:**
- Boolean - Yes/No
- If the str Contains in the String then it returns true
- If the str does not Contains in the String it returns False
- **Example**

```
Public Class Form1
Private Sub Button1_Click(...)
Dim str As String
str = "VB.NET TOP 10 BOOKS"
If str.Contains("TOP") = True Then
MsgBox("The string Contains() 'TOP' ")
Else
MsgBox("The String does not Contains() 'TOP'")
End If
End Sub
End Class
```

- **SubString**
- returns a new string that is a substring of this string.
- The substring begins at the specified given index and extended up to the given length.
- **Syntax:** string.Substring(ByVal startIndex As Integer, ByVal length As Integer) As String
- **Parameters:**
- startIndex: The index of the start of the substring.
- length: The number of characters in the substring.
- **Returns:**
- The specified substring.
- **Example**

```
Public Class Form1
Private Sub Button1_Click(...)
Dim str As String
Dim retString As String
str = "This is substring test"
retString = str.Substring(8, 9)
MsgBox(retString)
End Sub
End Class
```

CopyTo

Copies a specified number of characters from a specified position in this instance to a specified position in an array of characters.

Syntax:

String.CopyTo(sourceIndex As Integer, destination() As Char, destinationIndex As Integer, count As Integer)

Parameters:

Integer sourceIndex : The starting position of the source String

Char destination() : The character Array

Integer destinationIndex : Array element in the destination

Integer count : The number of characters to destination

Returns:

Integer - If the parameter String occurred as a substring in the specified String it returns position of the first character of the substring .

If it does not occur as a substring, -1 is returned.

Example

```
Public Class Form1
Private Sub Button1_Click(…)
Dim str1 As String = "CopyTo() sample"
Dim chrs(5) As Char
str1.CopyTo(0, chrs, 0, 6)
MsgBox(chrs(0) + chrs(1) + chrs(2) + chrs(3) + chrs(4) + chrs(5))
End Sub
End Class
```

Equals

Use: to check the specified two String Object values are same or not

Syntax: String.Equals(String str1, String str2) As Boolean

Parameters:

str1 : Parameter String

str2 : Parameter String

Returns: Boolean : Yes/No

It return the values of the two String Objects are same

Example

```
Public Class Form1
Private Sub Button1_Click(….)
Dim str1 As String = "Equals“
Dim str2 As String = "Equals"
If String.Equals(str1, str2) Then MsgBox("Strings are Equal() ")
Else MsgBox("Strings are not Equal() ") End If
End Sub
End Class
```

Join

This method joins two or more strings and returns a single string with a separator between the original strings.

Syntax: Newstring=String .join (separator , Strings)

Parameter:

Separator: is the string that will be used as separator.

Strings: String is an array with the string to be joined.

Returns:

New string

- If you have an array of many strings and you want to join few of them, then you can specify the index of the first string in the array and the number of strings to be joined with the following form of the join method.

Syntax:

newstring=string.join(separator,strings,startindex,count)

Parameter:

Separator: is the string that will be used as separator.

Strings: String is an array with the string to be joined.

Startindex: first array element in the value to use.

Count: number of elements of value to use.

Example

```
Dim s as string
Dim a() as string={“hello”,”how,”are”,”you”}
S=string.join(“/”,a)
Msgbox(s)
```

- **Remove**
- It removes number of characters from a string, starting at a specific location and returns the result as a new string
- **Syntax:** Newstring=string.remove(startindex,count)
- **Parameter:**

  Startindex: is the index of the first character to be removed in the string variable
- Count is the number of characters to be removed.
- **Return:**New String
- **Example**

```
Dim str as string =“kskv kachchh university”
Dim nstring as string
Nstring=str.remove(6,5)
Msgbox(Nstring)
```

- **Replace**
- This method replaces all instance of a specified character in a string with a new one
- It create a new instance of the string, replace the characters as specified by its arguments and returns this string
- **Syntax:** Newstring=string.replace(oldchar,newchar)
- **Parameter:**
- Oldchar is the character in the string variable to be replaced
- Newchar is the character to be replace the occurrence of old char.
- **Example**

```
Dim txt ,Ntxt as string
Txt="welcome to vb6"
Ntxt=txt.replace("vb7","vb.net")
```

- **PadLeft / PadRight**
- These method align the string left or right in a specified field.
- They both return a fixed length string with spaces to the right or to the left.
- Both methods accept the length of the field as arguments and return a new string.
- **Syntax:**

  Newstring=string.padleft(totalwidth)

  Newstring=string.padleft(totalwidth,character)

  Newstring=string.padRight(totalwidth)

  Newstring=string.padRight(totalwidth,character)
- **Parameter:**

  Totalwidth is additional padding character

  Character is the character to be pad
- **Return:**
- If the string is shorter than the specified field length then the padleft and padright methods return original string.
- These two method don't trim the string to fit in the specified field.
- **Example**

```
Dim s1 As String
s1 = TextBox1.Text
Dim s2 As String
s2 = s1.PadLeft(10)
TextBox2.Text = s2
```

DateTime Class

- It contains properties and method to manipulate date and time.
- Is used to store date and time value.
- Date and time value are stored internally as double numbers.
- Integer parts of the value correspond to the date and the fraction part correspond to the time.
- To convert a date variable to a double value,use the method ToOADatetime.
- **Properties**
- DateTime class exposes properties to manipulate date and time objects.

Parse/ParseExact

- If you have a strings that represent a date and you want to assign it to a date variable for further processing then parse and parseExact method is used.
- Parse method parses a strings and returns a date value.

Syntax:

Str = String.parse(date)

Parameter:

Str: date variable store date value

Date : string to convert in date.

Example

```
Private Sub Button1_Click(…)
Dim d As String
d = "2/3/2020"
Dim s As Date
s = Date.Parse(d)
MsgBox(Date.Parse(s))
End Sub
```

- **Date**
- It returns the date from a date/time value .
- **Syntax**: Ndate=dateobj.Date
- **Parameter:**
  Ndate: variable of type date
  Dateobj: object of date class
- **Return:** Date
- **Example**

```
Private Sub Button1_Click(...)
Dim d As Date
d = Now()
Dim a As Date
a = d.Date
MsgBox(a)
End Sub
```

- **DayOfWeek / DayOfYear**
- These two property return the day of week (a number from 1 to 7) and the number of the day in the year (an integer from 1 to 365 or 366 for leap year).
- **Syntax:**

  Nday = dateobj . DayOfWeek

  Nday = dateobj . DayOfYear
- **Parameter:**

  Nday: integer variable store week and year number

  Dateobj: object of date class
- **Return:** Day of week and day of year respectively
- **Example**

```
Private Sub Button1_Click(...)
Dim d As Date
d = Now()
Dim dw As Integer
dw = d.DayOfWeek
Dim dy As Integer
dy = d.DayOfYear
MsgBox(dw)
MsgBox(dy)
End Sub
```

Hour / minute / Second / Millisecond

These properties return the corresponding time part of the date value passed as arguments.

Syntax:

H = dateobj. Hour
M = dateobj. Minute
S = dateobj. Second
Milli = dateobj. Millisecond

Parameter:

H,M,S,Milli: are integer variable store corresponding time part.
Dateobj: object to date class

Example

```
Private Sub Button1_Click(…)
Dim d As Date
d = Now()
Dim h  ,m , s, milli As Integer
h = d.Hour
MsgBox(h)
m = d.Minute
MsgBox(m)
s = d.Second
MsgBox(s)
milli = d.Millisecond
MsgBox(milli)
End Sub
```

Day / Month / Year

These three property return day of the month,month of the year and year of the date value passed as an arguments.

Syntax:

D = objdate .Day
M = objdate .Month
Y = objdate.Year

Parameter:

D,M,Y: are integer variable that store day , month and year number
objdate : object of datetime class

Example

```
Private Sub Button1_Click(…)
Dim d As Date
d = Now()
Dim da, m, y As integer
da = d.Day
m = d.Month
y = d.Year
MsgBox("day" & "---" & da & "    " & "month" & " ---" & m & "   " & "year" & "---" & y)
End Sub
```

- **Ticks**
- This property returns the number of ticks from a date/time value.
- Each tick is 100 nanoseconds.
- To convert ticks to millisecond multiply them by 10000 .
- **Syntax: T** = objdate. Tick
- **Parameter:**
- T : variable of long data type store tick value
- Objdate: object of date class
- **Example**

```
Private Sub Button1_Click(...)
Dim d As Date
d = Now()
Dim t As Long
t = d.Ticks
MsgBox(t)
End Sub
```

- **TimeOfDay**
- It returns time from the date/time value.
- **Syntax:** Tod = objdate. TimeOfDay
- **Parameter:**
- Tod : date variable store time value.
- Objdate: object of date class.
- **Example**

```
Private Sub Button1_Click(...)
Dim d As Date
d = Now()
Dim t As Long
MsgBox(d.TimeOfDay)
End Sub
```

- **Methods**
- The datetime class exposes several method to manipulate date.
- You need to import system.datetime namespace to access datetime class method

- **Compare**
- Compare is a shared method that compare two dates /time value .
- **Syntax:** Order=datetime.compare(date1 as date, date2 as date) As integer
- **Parameter**:
- date1 and date 2 : two date value to be compared.
- **Returns:**
- It return -1 if date1 is less than date2
- 0 if they are equal
- 1 if date1 is greater than date2.
- **Example**

```
Imports System.DateTime
Public Class Form1
Private Sub Button1_Click(...)
Dim a As Integer
a = Compare("2/10/2010", "3/10/2010")
MsgBox(a)
End Sub
End Class
```

- **DaysInMonth**
- This shared method returns the number of days in a specific month
- Because february contains a variable number of days depending on the year,the DaysInMonth method accepts as arguments both the month and the year.
- **Syntax:** Monthdays=datetime.daysinmonth(year as integer ,month as integer)
- **Parameter:**

  Monthday: variable of type integer store month value

  year , month : integer value represent month and year.
- **Return:**
- Numbers of days in appropriate month
- **Example**

```
imports System.DateTime
Public Class Form1
Private Sub Button1_Click(...)
Dim monthday As Integer
monthday = DaysInMonth(2010, 3)
MsgBox(monthday)
End Sub
End Class
```

- **IsLeapYear**
- This shared method check whether specified year is leap or not?
- **Syntax:** datetime.IsleapYear(year as integer) as boolean
- **Parameter:**
- Year: 4- digit year
- **Return:**
- Boolean value
- **Example**

```
Imports System.DateTime
Public Class Form1
Private Sub Button1_Click(...)
MsgBox(IsLeapYear (2010)
End Sub
End Class
```

- **Adding Intervals to Dates**
- Various methods add specific intervals to a date/time value.
- Each method accepts the numbers of intervals to add (days , hours , milliseconds , and so on).
- For example. AddYears , AddMonths, AddDays, AddHours , AddMinutes, And AdSeconds, AddMilliseconds and AddTick.
- If arguments is a negative value ,the corresponding intervals are subtracted from the current instance of the class.
- **Syntax:**

Newdate= objdate. AddYears(year as integer) as date

Newdate= objdate. AddMonths(month as integer) as date

Newdate= objdate. AddDays(days as integer) as date

Newdate= objdate. Addhours(hours as integer) as date

Newdate= objdate. Addminute(minute as integer) as date

Newdate= objdate. Addsecond(second as integer) as date

Newdate= objdate. Addmillisecond(millisecond as integer) as date

Newdate= objdate. Addtick(tick as integer) as date

Working with file and folder

- We use file to store data and in many case we have to manipulate files and folder from within application.
- Vb.net exposes directory and file classes to access and manipulate file and folder.
- Directory class provides method for manipulating folder and file class provide method to manipulate file.
- You can store data and you can retrieve data from file using different method.

- **Directory Class**
- It exposes all the methods to manipulate folders .
- you must import System.IO namespace to work with directory.

- **Methods**
- File class expose various method to work with folder.

- **CreateDirectory**
- **Use:** it create a new folder whose path is specified by a string passed as arguments to the method
- **Syntax**: Directory.createDirectory(path as string)
- **Parameter:**Path: fully qualified path of a folder
- **Returns:**
- Direcroty information object which contain information about the newly created folder.
- **Example**

  Directory.createdirectory("c:\mscit")

- Create multiple nested folder in a single call

- **Syntax:** Directory.CreateDirectory("c:\folder1\folder2\folder3")
- It create folder folder1 (if it does not exist) then folder2 (if does not exist) under foledr1 .finally folder3 under folder2 in c:\ drive
- If folder1 already exist then folder2 will automatically creates.
- **Example**

  Directory.createdirectory("c:\kskvku\compsci\mscit\fy")

- **Delete**
- **Use:** it deletes a folder and all the files in it. if the folder contains subfolder the delete method optionally remove the entire directory tree under the node you are removing.
- **Syntax:** directory.delete(path as string)

  --delete the specified path only, if the specified folder contains subfolder they will not be deleted

  directory.delete(path as string , force as boolean)

  --It delete a folder recursively
- **Parameter:**
- Path : path of the folder to be deleted
- Force: is a Boolean value
- True: it remove folder with all its file
- False: it removes only specified folder
- **Example**

Directory.delete("c:\kskv")

Directory.delete("c:\kskv",true)

Exists

- **Use** : it check the existance of folder.
- **Syntax:** directory.exists(path as string)
- **Parameter;**Path:path of the folder
- **Returns:** Boolena value true/false
- **Example**

  Msgbox(Directory.exists("c:\kskv"))

GetCreationtime

- **Use :** it get specific folder's creation time.
- **Syntax:** createon=directory.getcrationtime(path as string)
- **Parameter:**
- Createon : variable store date value
- Path: path of folder
- **Returns**: Date value when specific folder is created
- **Example**

Dim get as date
Get = directory.getcreationtime("c:\kskv")
Msgbox(get)

SetCreationtime

- **Use :** Set the date when user create specific folder
- It accepts a path and date value as arguments and sets specific folder's creation time
- **Syntax:** directory.sercrationtime(path as string , datetime as date)
- **Parameter:**
- datetime : datetime value
- Path: path of folder for which you want to set sate and time
- **Example**

Directory.setCreationTime("c:\kskv", "2\3\2010")

GetCurrentDirectory

- **Use**: it retrieve the path of current directory.
- **Syntax:** Cd=Directory.getcurrentdirectory()
- **Parameter:**
- Cd: variable of type string that store directory value.
- **Return:** Current directory in which application is running
- **Example**
- Dim getdir as string
- Getdir=directory.getcurrentdirectory()
- Msgbox(getdir)

SetcurrentDirectory

- **Use:** it set the current directory.
- **Syntax:** directory .setCurrentDirectory(path as string)
- **Parameter:**
- Path: path of current directory

GetDirectories

- **Use:** it retrieves all the subfolders of a specific folder and return theirs names as an array of strings
- **Syntax:** Dirs = directory.getdirectories(path)
- **Parameters:**
- Path: path of folder which subfoldder you want to retrieve
- Dirs: array that accepts the name of subfolder
- **Returns:** All subfolder of a specific folder as an array
- **Example**

```
Dim dirs() as string
dirs = directory.getdirectories("c:\")
For count = 0 To dirs.length
MsgBox(dirs(count))
Next
```

GetDirectoryRoot

- **Use:** it is used to retrieve the root part of the path.
- **Syntax:** Root=directory.getdirectoryroot(path)
- **Parameter:**
- Path: path of folder
- Root: variable it store root part of path
- **Returns:** Root part of path
- **Example**

Dim root as string
Root=directory.GetDirectoryRoot(c:\kkkv\compsci\mscit")
Msgbox(root)

GetFiles

- **Used:** to get files from specific folder
- **Syntax:**
- Files = directory.GetFiles(path)
- Files=directory.getfiles(path , pattern)
- It allows you to specify a pattern and retrieve only the names of the files that match the pattern.
- **Parameter**:Path: path of folder whose file you want to retrieve
- Files; array of string . Stores file name
- **Returns:**

Name of files in the specific folder as an array of strings
Example
Dim file() as string
File = directory.getfiles("c:\kskv","*.txt")
For count = 0 To file.Length
MsgBox(file(count))
Next

- **GetFileSystemEntries**
- **Use:** to retrieve all items (files and folder)
- **Syntax:**
  Items=directory.getfilesystementrie(path)
  Items=directory.getfilesystementries(path, pattern)
- **Parameter:**
- Items: array that holds all files ans folder
- Path: path of the folder which items you want to retrieve
- **Returns:** Array of all items in a path.
- Example

```
Dim items() as string
items = directory.getfilesystementries("c:\kskv")
For count = 0 To items.Length – 1
MsgBox(items(count))
Next
```

- **GetLastAccessTime**
- **Use**: to retrieve the most recent date and time the file was accessed.
- **Syntax:**
- Gtime=directory.getlastAccesstime(path)
- **Parameter:**
- Gtime: variable it store date time value
- Path: file of folder for which you obtain access date and time
- **Returns:** Most recent date and time the file was accessed.
- Example

Dim getlast as date
Getlast=directory.GetLastAccessTime("c:\kskv")
Msgbox(Getlast)

- **SetLastAccessTime**
- **Use:** to set the most recent date and time the file was accessed.
- **Syntax:** directory.setlastAccesstime(path as string, lastaccesstime as date)
- **Parameter:**
- Path; file of folder for which you obtain access date and time
- Lastaccesstime:date and time you want to set
- Example

Directory.setlastaccesstime("c:\kskv","12\12\2010")

GetLastWriteTime

- **Use:** to retrieve the most recent date and time the file was written.
- **Syntax:** Glwtime=directory.getlastwritetime(path)
- **Parameter:**
- Glwtime: variable it store date time value
- Path; file of folder for which you obtain access date and time
- **Returns:**
- Most recent date and time the file was written
- **Example**

  Dim glwtime as date

  Glwtime=directory.getlastwritetime("c:\kskv")

SetLastWrittenTime

- **Use:** to set the most recent date and time the file was written
- **Syntax:** directory.setlastWrittentime(path as string, lastaccesstime as date)
- **Parameter:**
- Path; file of folder for which you obtain access date and time
- Lastaccesstime:date and time you want to set
- Example

  Directory.setlastwrittentime("c:\kskv","12\10\2010")

GetParent

- **use:** to retrieve parent folder
- **Syntax;** Parent=directory . Getparent(path)
- **Parameter;**
- Parent: string variable store parent folder
- Path: name of file and folder for which you want to find parent folder
- **Returns:**
- Parent folder
- Example

Dim parent as string
Parent=directory.GetParent("c:\mscit")

Move

- **Use:** to move entire folder to another location
- **Syntax:**Directory.move(source, destination)
- **Paramater:**
- Source: path of file or folder you want to move
- Destination: name of the destination folder
- **Example**
  Directory.move("c:\kskv", "d:\")

GetLogicalDrives

This method return name of the logical drive on the comuter.

Syntax: Drives =Directory.GetLogicalDrives()

Paramater:

Drives:array of string store logical drive.

Return:

Logical drive

Example

```
Dim drives() As String
drives = Directory.GetLogicalDrives
Dim idrive As Integer
For idrive = 0 To drives.Length
MsgBox(drives(idrive))
Next
```

File Class

- File class expose methods for manipulating files (copying,moving,opening and closing).
- You need to imports Imports System.IO name space to access all method of file Class.
- **AppendText**
- **Use:** to append text to existing text file .
- If file doesn't exist it create new one and opens it
- **Syntax:** Fstream = file.appendtext(path)
- **Parameter:**Fstream: streamwriter object
- Path: path to the file to append to
- **Returns:**Fsteram sreamwriter object
- **Example**

```
Dim FILE_NAME As String = "D:\test.txt"
    Dim i As Integer
    Dim aryText(4) As String
    aryText(1) = "Had"
    aryText(2) = "Another"
    aryText(3) = "Little"
    aryText(4) = "One"
    Dim objWriter As New System.IO.StreamWriter(FILE_NAME, True)
    For i = 0 To 4 Step +1
       objWriter.WriteLine(aryText(i))
    Next
    objWriter.Close()
    MsgBox("Text Appended to the File")
```

- **Copy**
- **Use :** to copy and existing file to new location
- **Syntax:**

  File.copy(source , destination)

  If destination file exist then copy method will fail

  File.copy(source,destination,overwrite)
- It allows you to specify whether the destination file can be overwritten
- **Parameter:**
- Source: name of the file to be copied,
- Destination : name of destination file
- Overwrite: is a Boolean value
- If true ; file is overwritten if exists.
- **Example**

```
Private Sub Button1_Click(...)
File.Copy("c:\mscit.txt", "d:\kskv\mscit.txt")
End Sub
```

- **Create**
- **Use:** to create a new file. It returns a sream object it is use to read or write form the file. If spefified file exist already its' replaced. New file is opened for read write operations.
- **Syntax:**
- Fstream= file.create(path)
- Fstream=file.create(path,buffersize)
- It create a new file and specify the size of the buffer
- **Parameter:** Path: path and name of file to create
- Buffersize: integer value.
- **Returns:** Stream object to this file.
- **Example**

```
Private Sub Button1_Click(...)
    File.Create("c:\mscit.txt")
End Sub
```

CreateText

- **Use:** to create a text file.
- **Syntax:**Sw= File.createtext(path)
- **Parameter:**
- Path: file to be open for writing.
- **Returns:**
- Steram writer objects for writing to the file.,but used for only text file.

Delete

Use : to remove specified file from the file system.

Syntax: File.delete(path)

Parameter:

Path : path of file you want to delete.

This method will raised exception if the file is open at the time for reading or writing or doesn't exist.

Example

```
Private Sub Button1_Click(…)
        File.Delete("c:\mscit.txt")
End Sub
```

Exists

Use : to check the existance of file.

It is a boolean value that indiactes wheter specific file exists or not.

Syntax: File.exist(path)

Parameter:

Path: name of file to check

Example

```
Private Sub Button1_Click(…)
MsgBox(File.Exists("c:\kskv"))
End Sub
```

- **GetAttributes**
- **Use**: to get to the attribute of specified file.
- **Syntax:** Gattrib=file.getattributes(path)
- **Parameter:**
- Path : path to the file
- Gattrib:stroe attributes of specified file
- **Returns:**

  Attributes of the specified file.

  File may have more than a single attributes
- **Example**

```
Private Sub Button1_Click(...)
Dim attr As String
attr = File.GetAttributes("c:\gauru")
MsgBox(attr)
End Sub
```

GetCreationtime

- **Use :** it get specific files creation time.
- **Syntax:** createon=file.getcrationtime(path)
- **Parameter:**
- Createon : variable store date value
- Path: path of file
- **Returns:** Date value when specific file is created
- **Example**

```
Private Sub Button1_Click(...)
Dim gettime As Date
gettime =File.GetCreationTime("c:\mscit")
MsgBox(gettime)
End Sub
```

SetCreationtime

- **Use :** Set the date when user create specific file
- It accepts a path and date value as arguments and sets specific file's creation time
- **Syntax:** file.sercrationtime(path , datetime)
- **Parameter:**
- datetime : datetime value
- Path: path of file for which you want to set sate and time.
- **Example**

```
Private Sub Button1_Click(...)
file.SetCreationTime("c:\mscit.txt", 3\2\2010")
End Sub
```

GetLastAccessTime

- **Use:** to retrive the most recent date and time the file was accessed.
- **Syntax:**
- Gtime=file.getlastAccesstime(path)
- **Parameter:**
- Gtime: variable it store date time value
- Path; file of folder for which you obtain access date and time
- **Returns:**
- Most recent date and time the file was accessed.

SetLastAccessTime

- **Use:** to set the most recent date and time the file was accessed.
- **Syntax:**
- file.setlastAccesstime(path as string, lastaccesstime as date)
- **Parameter:**
- Path; file of folder for which you obtain access date and time
- Lastaccesstime:date and time you want to set

GetLastWriteTime

- **Use:** to retrive the most recent date and time the file was written.
- **Syntax:** Glwtime=file.getlastwritetime(path)
- **Parameter:**
- Glwtime: variable it store date time value
- Path; file of folder for which you obtain access date and time
- **Returns:** Most recent date and time the file was written

SetLastWrittenTime

- **Use:** to set the most recent date and time the file was written
- **Syntax:** file.setlastWrittentime(path as string, lastaccesstime as date)
- **Parameter:** Path; file of folder for which you obtain access date and time
- Lastaccesstime:date and time you want to set

- **Move**
- **Use :** to move the specified file to a new location.
- Also used to rename a file by simply moving It to another name in the same folder.
- **Syntax:**
- File.move(sourcefilename, destfilename)
- **Parameter:**
- Sourcefilenmae: path of the file to be moved
- Destfilename: path of the destination file.
- **Example**

```
Dim FileToMove As String
Dim MoveLocation As String
FileToMove = "C:\Users\Owner\Documents\test.txt"
    MoveLocation = "C:\Users\Owner\Documents\TestFolder\test.txt"
If System.IO.File.Exists(FileToMove) = True Then
    System.IO.File.Move(FileToMove, MoveLocation)
    MsgBox("File Moved")
    End If
```

Open

- **Use:** to open an existing file for read-write operations.
- **Syntax:**
- Fstream=file.open(path)
- Fstream = file.open(path,filemode)
- It allows you to specify mode in which you want to open file
- Fstream=file.open(path,filemode,accessmode)
- It allows you to specify the access mode
- Fstream=file.open(path,filemode,accessmode,sharemode)
- It spefify how the file will be shared with other applications.
- **parameter:**
- Path:name of file you want to open
- Filemode:mode of file
- Accessmode:mode of access
- **Returs:** Stream objects to this file.

OpenRead

- **Use :** to open existing file in read mode.
- **Syntax:**
  Fstream=file.openread(path)
- **Parameter**
  Path : path of file to open
- **Return :** Stream object (use to read from file)

OpenText

- **Use:** to open an existing text file for reading
- **Syntax:** Fstream=file.opentext(path)
- **Parameter:** Path: path of text file to open for read.
- **Returns:** A streamreader object

OpenWrite

- **Use:**to open an existing file in write mode
- **Syntax:** Fstream=file.openwrite(path)
- **Parameter:** path:path of file to open for writing
- **Returns:** Streamwriter object

DirectoryInfoClass

- Directory Info class is similar to directory class.
- But they must be instantiated before they are used.
- Their constructor specify the folder or file they will act upon. you do not need to specify a folder or file when you call their methods.
- **Methods**
- Directory info class exposes some method to work with directory.
- You need to import Methods of directory info class are similar to directory class.
- Here are couple of methods that are unique to the Directory Info class

CreateSubDirectory

- **Use:** to create a subfolder under the folder specify by the current instance of the class.
- **Syntax:** Directory. CreateSubDirectory(path)
- **Return:** DirectoryInfo object that represent new subfolder.
- **Parameter:** Path of folder
- **Example**

```
Private Sub Button1_Click(...)
Dim di As New DirectoryInfo("c:\kskv")
di.CreateSubdirectory(“mscit")
End Sub
```

GetFileSystemInfos

- **Use:** to retrieve all the information about folder.
- **Syntax:**

  Iteminfo = diObject.GetFileSystemInfos

  Iteminfo = diObject.GetFileSystemInfos(pattern)
- **Parameter:**
- Iteminfo: array that store folder or file info.
- **Return:** An array of FileSystemInfo Objects

PathClass

- Pathclass's methods perform simple task such as retrieving a file's name and extension , returning the full path description of a relative path .
- Path class methods require that you specify the path on which they will act.
- There is no constructor for path class that would allow you instantiate a path objects to represent a specific path.
- **Properties**
- Path class exposes the following properties to work with path.

- **AltDirectorySeparatorChar**
- It return alternate directory separator character.
- **Syntax:**
- Separator= path.AltDirectorySeparatorChar
- **Parameter:**

  Separator : string variable store separator character
- **Return:**
- Separator character.
- **Example**

```
Private Sub Button1_Click(...)
Dim separator As String
Separator = Path.AltDirectorySeparatorChar
MsgBox(s)
End Sub
```

DirectorySeparatorChar

- It returns the directory separator character.
- **Syntax:**
  Separator= path.DirectorySeparatorChar
- **Parameter:**
  Separator : string variable store separator character
- **Return:** Separator character
- **Example**

```
Private Sub Button1_Click(...)
Dim separator As String
Separator = Path.DirectorySeparatorChar
MsgBox(s)
End Sub
```

InvalidPathcharacter

- It returns the list of invalid character in a path as an array.
- **Syntax:**
  invalidpath= path.InvalidPathCharacter
- **Parameter:**
  invalidpath : array that store invalid path character.
- **Return:**
  Invalid path character
- **Example**

```
Private Sub Button1_Click(...)
Msgbox (Path.InvalidPathCharacter )
End Sub
```

- **PathSeparator**
- It returns separator character that may appear between multiple path.
- **Syntax:**
  Separator= path.pathSeparator
- **Parameter:**
  Separator : character variable store separator character
- **Return:**
  Path Separator character
- **Example**

```
Private Sub Button1_Click(...)
Dim c As Char
c = Path.PathSeparator
MsgBox(c)
End Sub
```

VolumeSeparator

- It return volume separator character.
- **Syntax:** volume= path.VolumeSeparator
- **Parameter:** Volume : character variable store volume separator
- **Return:** volume Separator character

Example

```
Private Sub Button1_Click(...)
Dim volume As Char
volume = Path.VolumeSeparatorChar
MsgBox(volume)
End Sub
```

ChangeExtension

- **Use:** to change extension of file.
- **Syntax:**Newext = path. Changeextension(path,extension)
- **Parameter**:Newext: string variable store extension
  Path: path of file
  Extension: files new extension

  If you want to remove file's extension then set the second argument to nothing
- **Example** Private Sub Button1_Click(...)
  Dim next As String
  next = Path.ChangeExtension("c:\mscit.txt", "doc")
  MsgBox(n)
  End Sub

Combine

- **Use**: to combine to path specification into one.
- **Syntax:**Newpath=path.combine(path1,path2)
- **Parameter:**Newpath: string variable store combine new path
  Path1 & path2: path of file
- **Return:** New combine path

Example: Private Sub Button1_Click(...)
Dim newpath As String
newpath = Path.Combine("c:\kskv.txt", "mscit.txt")
MsgBox(newpath)
End Sub

GetDirectoryName

Use: to get directory name of the path

Syntax: Dirname=path.GetDirectoryName(path)

Parameter: Dirname: string variable store directory name
Path: path of file

Return: Directory name of path

Example

```
Private Sub Button1_Click(…)
Dim dirname As String
dirname = Path.GetDirectoryName("c:\mscit.txt")
MsgBox(dirname)
End Sub
```

GetFileName

Use: to get file name of path.

Syntax:

Filename= path. getfilenameI(path)

Parameter:

Filebname: string variable store file name

Path: path of file

Return: File name

Example

```
Private Sub Button1_Click(…)
Dim filename As String
filename = Path.GetFileName("c:\mscit.txt")
MsgBox(filename)
End Sub
```

- **GetFileNameWithoutExtension**
- **Use:** to get file name of path without extension.
- **Syntax:**Filename= path. getfilenameIwithoutextension(path)
- **Parameter:**
- Filebname: string variable store file name
- Path: path of file
- **Return:** File name without extension
- **Example**

```
Private Sub Button1_Click(...)
Dim filename As String
filename = Path.GetFileNamewithoutExtension("c:\mscit.txt")
MsgBox(filename)
End Sub
```

- **GetFullPath**
- **Use:** to get full path of the specified path
- **Syntax:**
- fullpath= path.GetFullPath(Path)
- **Parameter:**
- Fullpath : string variable store full path
- Path : path of file
- **Return:**Full path of file
- **Example**

```
Private Sub Button1_Click(...)
Dim fullpath As String
fullpath = Path.GetFullPath("c:\mscit.txt")
MsgBox(fullpath)
End Sub
```

FileInfo Class

- FileInfo class is similar to file class.
- But they must be instantiated before they used.
- Their constructor specify the file they will act upon, and you do not have to specify a file when you call their method.
- **Properties/ Methods**
- fileInfo class exposes some methods and properties which are equivalents to the member of file class.

- **Length**
- **Use:** to get size of file represented by the fileinfo objects in bytes.
- **Syntax:**

  Len = fibjects.Length
- **Parameter:**

  Len : integer variable store file size

  Fiobjects: objects of fileInfo class.
- **Return:**Size of file in bytes
- **Example**

```
Private Sub Button1_Click(...)
Dim di As New FileInfo("c:\mscit.txt")
Dim len As Integer
len = di.Length
MsgBox(len)
End Sub
```

Creationtime

- **Use :** it get specific files creation time.
- **Syntax:** createon=fiobject.creationtime
- **Parameter:** Createon : variable store date value
  Fiobject: object of fileinfo class.
- **Returns:**Date value when specific file is created
- **Example:** Private Sub Button1_Click(...)
  Dim di As New FileInfo("c:\dhavu.txt")
  Dim createon As Date
  createon = di.CreationTime
  MsgBox(createon)
  End Sub

LastAccessTime

- **Use:** to retrieve the most recent date and time the file was accessed.
- **Syntax:** lastaccesstime= fiobject.lastAccesstime
- **Parameter:** lastaccesstime: variable it store date time value
  Fiobject: object of fileinfo class.
- **Returns:** Most recent date and time the file was accessed.
- **Example**

```
Private Sub Button1_Click(...)
Dim di As New FileInfo("c:\dhavu.txt")
Dim lastaccesstiem As Date
lastaccesstiem = di.LastAccessTime
MsgBox(lastaccesstiem)
End Sub
```

- **LastWriteTime**
- **Use:** to retrieve the most recent date and time the file was written.
- **Syntax:** lwtime=fiobject.getlastwritetime
- **Parameter:**

  lwtime: variable it store date time value

  Fiobject: object of fileinfo class.
- **Returns:**Most recent date and time the file was written
- **Example**

```
Private Sub Button1_Click(...)
Dim di As New FileInfo("c:\dhavu.txt")
Dim lwtime As Date
lwtime = di.LastWriteTime
MsgBox(lwtime)
End Sub
```

- **Name/Fullname/Extension**
- **Use:** to get name, fullname or extension of file
- **Syntax:** Name= FiObject.name
  Fullname=Fiobjects.fullname
  Extension=Fiobjest.Extension
- **Parameter:**
  Name: variable of type string it store name
  Fullname: variable of type string it store fullname
  Extension: variable of type String it store extension
- **Return:** Name, full name and extension of file
- **Example**

```
Private Sub Button1_Click(...)
Dim di As New FILEInfo("c:\dhavu.txt")
MsgBox(di.Extension)
Msgbox(di.Name)
Msgbox(di.FullName)
End Sub
```

Methods

fileInfo class exposes methods for manipulating files and most of them are equivalent to the methods of the file objects.

CopyTo

Use: to copy file represented by the current instance of the fileInfo class.

Syntax:

Fileinfo.copyto(path)

Fileinfo.copyTo(path, force)

Parameter:

Path: destination where the file will be copied.

Force: boolean value.

Return : instance of fileinfo class .

Example

```
Private Sub Button1_Click(…)
Dim di As New FileInfo("c:\mscit.txt")
di.CopyTo("d:\mscit.txt")
End Sub
```

MoveTo

Use: to move file represented by the current instance of the fileInfo class.

Syntax:

Fileinfo.moveTo(path)

Fileinfo.moveTo(path, force)

Parameter:

Path: destination where the file will be moved.

Force: boolean value.

Return : instance of fileinfo class .

Example

```
Private Sub Button1_Click(…)
Dim di As New FileInfo("c:\mscit.txt")
di.MoveTo("d:\mscit.txt")
End Sub
```

Directory

- **Use:** to get information about files parent directory
- **Syntax:** Dir = fiobj. Directory
- **Parameter:** Dir: string variable store directory information
  Fiobj: object of fileinfo class
- **Return:** Information about Files parent directory.

Directoryname

- **Use:** to get file's parent directory name.
- **Syntax:**
- Dirn = FIobj.directoryname
- **Parameter:** Dirn: string variable stroe directory name
  FiObj: object of fileinfo class
- **Return:** File's parent directory name
- **Example**

```
Private Sub Button1_Click(...)
Dim fi As New FileInfo("c:\mscit")
MsgBox(fi.DirectoryName())
End Sub
```

Serialization Class

- You have seen how to store items in various collection, how to access their elements and how to search and sort element.
- But none of the collection exposes a save or similarly name method to store collection to disk .
- Fortunately there is a mechanism that can store arbitrary objects to disk.
- Serialization class exposes two method:
  - Serialized
  - Deserialize
- serialized method which store an object to disk.
- Deserialize method does the opposite ,it reads a file created by serialized method and create the original object.
- To use the method of serialization class you need to import System.Runtime.Serialization.Formatters.

- **Declare an object**
- Dim bformatter as new Binaryformatter()
- Binary formatter class persists objects in binary format.
- **Serialized Method**
- Use: to save an object to disk
- Syntax:
  - Bformatter .serialize(Stream, object)
- Parameter:
- Stream : a variable that represent a stream .
- Object : object you want to serialize.
- Bformatter: instance of serialize class.

- **Serializing individual Object / collection**
- Import System.Runtime.Serialization.Formatters.Binary Namespace
- import system.IO namespace
- Create object to serialize.
- Create instance of filestream .
  - Dim fsobj as Filestream
- Call create method to create file.
  - fsobj= file.create (“file name”)
- Create object of binary formatter class
  - Dim bformatter as new binaryformatter
- Call serialize method
  - Bformatter .serialize(filestream , object)

- **Example**

```
Imports System.Runtime.Serialization.Formatters.Binary
Imports System.IO
Public Class Form2
    Private Sub Button1_Click(...)
        Dim s() As String = {"a", "b", "c", "d"}
        Dim save As FileStream
        save = File.Create("c:\str.bin")
        Dim bformatter As New BinaryFormatter
        bformatter.Serialize(save, s)
    End Sub
End Class
Serializing Collection
Private Sub Button1_Click(...)
        Dim arr As New ArrayList
        arr.Add("sun")
        arr.Add("mon")
        arr.Add("tue")
        arr.Add("wed")
        arr.Add("thu")
        arr.Add("fri")
        arr.Add("sat")
        Dim save As FileStream
        save = File.Create("c:\days.bin")
        Dim bformatter As New BinaryFormatter
        bformatter.Serialize(save, arr)
    End Sub
End Class
```

- **Deserialize Method**
- Use: to read file created by serialize method. And recreate the original object.
- Syntax:
- Object = Bformatter . Deserialize(str)
- Parameter:
- Object: serialized object.
- Str: stream object to the file with the data.

- **Deserializing Objects**
- Import System.Runtime.Serialization.Formatters.
- Binary namespace
- import system.IO namespace
- Create instance of filestream .
  - Dim fs as filestream
- Call openread method to open file.
  - fs= file.openread (“file name”)
- **Deserializing Objects**
- Create instance of serialize class.
  - Dim ser as new binaryformatter
- Create object to store data
  - Dim objectname as <type>
- Call deserialize method
  - Objectname =CType(bformatter.deserialize(streamobject) , <type>) (streamobject)

- **Example**

```
Dim read As FileStream
        read = File.OpenRead("c:\days.bin")
        Dim bformater As BinaryFormatter
        bformater = New BinaryFormatter
        Dim arr As New ArrayList
        arr = CType(bformater.Deserialize(read), ArrayList)
        Dim i As Integer
        For i = 0 To arr.Count-1
            MsgBox(arr(i))
    Next
```

TimeSpan Class

- It represent time interval and can be expressed in many different units, from ticks and millisecond to days.
- Timespan is usually the difference between two date/time value.
- You can create a timespan for a specific interval and use it in your calculation.
- Declare timespan variable:
  - Dim ts as new TimeSpan
- Initialize timespan objects:
  - Dim ts as new timespan(days, hours, minutes, second, millisecond)
  - Dim ts as new timespan(10,2,25,20,300)
- **Properties**
- TimeSpan class exposes some properties to work with time.

- **Example**

```
Private Sub Button1_Click(...)
        Dim ts As New TimeSpan(12, 20, 10, 30,300)
        MsgBox(TimeSpan.TicksPerDay)
        MsgBox(TimeSpan.TicksPerMillisecond)
        MsgBox(TimeSpan.TicksPerHour)
        MsgBox(TimeSpan.TicksPerMinute)
        MsgBox(TimeSpan.TicksPerSecond)
    End Sub
```

- **Duration**
- It returns the duration of the current instance of the timespan.
- Duration is expressed as the number of days, followed by number of hours ,minutes, seconds and milliseconds.
- **Syntax:**Duration = tsobj . Duration
- **Parameter**:
  - Duration : string variable store duration value
  - Tsobj: instance of timespan class
- **Return:**number of days, followed by number of hours ,minutes, seconds and milliseconds.
- **Example**

```
Private Sub Button1_Click(...)
Dim t1 As Date
t1 = Now()
MsgBox("click ok to continue")
Dim t2 As Date
t2 = Now()
Dim ts As TimeSpan
ts = t2.Subtract(t1)
  msgbox(ts.tostring)
Dim dur As String
dur = ts.Duration.ToString
MsgBox(dur)
End Sub
```

- **Methods**
- Timespan class expose various method to manipulate the instance of timespan class

Interval

- It create time span object for a specific duration.
- Each duration is specified as a number of intervals.
- Accurate to a nearest millisecond,

FromDay

- **Use:** to create new timespan .
- **Syntax:** TS=system.timespn.fromdays(value as double)
- **Parameter:** Value: number of days.
- **Returns:** Number of days.
- **Example**

```
Private Sub Button1_Click(...)
 Dim ts As TimeSpan
 ts = System.TimeSpan.FromDays(30)
 MsgBox(ts.Days)
End Sub
```

Fromhours

- Use : to create new timespan
- **Syntax:**
  - TS=system.timespan.fromhours(value as double)
- **Parameter:**
  - Value: number of hours
- Returns:
  - Number of hours.
- **Example**

```
Private Sub Button1_Click(...)
 Dim ts As TimeSpan
 ts = System.TimeSpan.FromHours(3)
 MsgBox(ts.Hours)
End Sub
```

- **fromminutes**
- Use : to create new timespan
- Syntax: TS=system.timespan.fromminutes(value as double)
- Parameter: Value: number of minutes.
- Returns: Number of minutes.
- **Example**

```
Private Sub Button1_Click(...)
        Dim ts As TimeSpan
        ts = System.TimeSpan.Fromminutes(3)
        MsgBox(ts.minutes)
    End Sub
```

FromSeconds

- Use : to create new timespan
- Syntax: TS=system.timespan.fromSeconds(value as double)
- Parameter: Value: number of Secodns.
- Returns: Number of seconds.
- **Example**

```
Private Sub Button1_Click(...)
        Dim ts As TimeSpan
        ts = System.TimeSpan.Fromseconds(3)
        MsgBox(ts.seconds)
    End Sub
```

Frommilliseconds

- Use : to create new timespan
- Syntax: Ts= system.timespan. frommilliseconds(value as double)
- Parameter: Value: number of milliseconds.
- Returns: Number of milliseconds.
- **Example**

```
Private Sub Button1_Click(...)
  Dim ts As TimeSpan
  ts = System.TimeSpan.Frommillisecond(3)
  MsgBox(ts.milliseconds)
  End Sub
```

Fromticks

- Use : to create new timespan
- Syntax: ts=system.timespan.fromticks(value as long)
- Parameter: Value: number of ticks.
- Returns: Number of ticks.
- **Example**

```
Private Sub Button1_Click(...)
      Dim ts As TimeSpan
      ts = System.TimeSpan.FromTicks(3)
      MsgBox(ts.Ticks)
   End Sub
```

Parse(string)

- Use:
  - To create new timespan object from a string.
  - with the timespan format (days followed by period ,hours , minute , seconds separated by colons).
- Syntax:ts= timespan.parse(s as string)
- Parameters:
  - Ts : instance of timespan class
  - S: string with above format.
- Returns: new time span object
- **Example**

```
Private Sub Button1_Click(...)
    Dim sp As TimeSpan
    sp = TimeSpan.Parse("3:12:20:30:300")
    MsgBox(sp)
 End Sub
```

ADD

- **Use:** add a timespan object to the current instance of class.
- **Syntax:**
  - Newts= ts.add(ts1)
- **Parameters:**
  - Ts, ts1,Newts : timespan variable
- **Returns:**
  - Total duration of timaspan

- **Example**

```
Private Sub Button1_Click(...)
    Dim ts1 As New TimeSpan(1, 2, 3, 4, 5)
    Dim ts2 As New TimeSpan(2, 3, 4, 5, 6)
    Dim ts As TimeSpan
    ts = ts1.Add(ts2)
    MsgBox(ts.ToString)
End Sub
```

Subtract

- **Use:** subtract a timespan object from the current instance of the timespan class.
- **Syntax:**Newts= ts.subtract(ts1)
- **Parameters:**Ts, ts1,newts : timespan variable
- **Returns:**Time span after subtraction
- **Example**

```
Private Sub Button1_Click(...)
    Dim ts1 As New TimeSpan(5, 6, 7, 8, 9)
    Dim ts2 As New TimeSpan(2, 3, 4, 5, 6)
    Dim ts As TimeSpan
    ts = ts1.Subtract(ts2)
    MsgBox(ts.ToString)
End Sub
```

CompareTo

- **Use:** it compare the current instance of timespan with another timespan object.
- **Syntax:**Ts. CompareTo(value)
- **Parameter:**Value :object to compare
- **Returns:** 0 : if equal
  -1 : if current instance is longer
  1: if timespan object passed as arguments is longer
- **Example**

```
Private Sub Button1_Click(...)
  Dim ts1 As New TimeSpan(5, 3, 7, 8, 9)
  Dim ts2 As New TimeSpan(2, 6, 4, 5, 6)
  MsgBox(ts1.CompareTo(ts2))
End Sub
```

Equals

- **Use**: indicate whether two timespan objects represent same interval
- **Syntax:** Ts.equal(value)
- **Parameter:** Value: object to compare
- **Return:** True: if equal
  False: if not equal
- **Example**

```
Private Sub Button1_Click(...)
  Dim ts1 As New TimeSpan(5, 3, 7, 8, 9)
  Dim ts2 As New TimeSpan(2, 6, 4, 5, 6)
  MsgBox(ts1.Equals(ts2))
End Sub
```

- **Nagate**
- **Use** : to negate current timespan instance
- **Syntax:** Ts.nagate()
- **Example**

```
Private Sub Button1_Click(...)
        Dim ts1 As New TimeSpan(5, 3, 7, 8, 9)
        Dim ts2 As New TimeSpan
        ts2 = ts1.Negate
        MsgBox(ts2.ToString)
    End Sub
```

The Path Class

The Path class contains an interesting collection of methods, which you can think of as utilities. The Path class's methods perform simple tasks such as retrieving a file's name and extension, returning the full path description of a relative path, and so on. The Path class's members require that you specify the path on which they will act. In other words, there's no constructor for the Path class that would allow you instantiate a Path object to represent a specific path.

Properties

The Path class exposes the following properties. Notice that none of these members apply to a specific path; they're general properties that return settings of the operating system.

AltDirectorySeparatorChar

This property returns an alternate directory separator character. For Windows 2000, the AltDirectorySeparatorChar property returns the slash character (/).

DirectorySeparatorChar

This property returns the directory separator character. For Windows 2000, the DirectorySeparatorChar returns the backslash character (\).

InvalidPathChars

This property returns the list of invalid characters in a path as an array of characters. The following statements print the invalid path characters to the Output window; their output is shown in bold below the code:

```
Dim p As Path
Dim invalidPathChars() As Char
invalidPathChars = p.InvalidPathChars
Dim c As Char
For Each c In invalidPathChars
   Console.Write(c & vbtab)
Next
```

```
/   \   "   <   >   |
```

You can use these characters to validate user input, or pathnames read from a file. If you have a choice, let the user select the files through the Open dialog box, so that their pathnames will always be valid.

PathSeparator

This property returns separator character that may appear between multiple paths. For Windows 2000, this character is the semicolon (;).

VolumeSeparatorChar

This property returns the volume separator character. For Windows 2000, this character is the colon (:).

InvalidPathChars

This property returns the list of invalid characters in a path as an array of characters. The following statements print the invalid path characters to the Output window; their output is shown in bold below the code:

```
Dim p As Path
Dim invalidPathChars() As Char
invalidPathChars = p.InvalidPathChars
Dim c As Char
For Each c In invalidPathChars
   Console.Write(c & vbtab)
Next
```

/ \ " < > |

You can use these characters to validate user input, or pathnames read from a file. If you have a choice, let the user select the files through the Open dialog box, so that their pathnames will always be valid.

PathSeparator

This property returns separator character that may appear between multiple paths. For Windows 2000, this character is the semicolon (;).

VolumeSeparatorChar

This property returns the volume separator character. For Windows 2000, this character is the colon (:).

METHODS

The most useful methods exposed by the Path class are like utilities for manipulating file and path-names, and they described in the following sections. Notice that the methods of the Path class are shared: you must specify the path on which they will act.

ChangeExtension

This method changes the extension of a file, and its syntax is

```
newExtension = Path.ChangeExtension(path, extension)
```

The return value is the new extension of the file (a string value). The first argument is the file's path, and the second argument is the file's new extension. If you want to remove the file's extension, set the second argument to Nothing. The following statement changes the extension of the specified file from "BIN" to "DAT":

```
Dim path As String = 'c:\My Documents\NewSales.bin'
Dim newExt As String = ".dat"
Path.ChangeExtension(path, newExt)
```

The FileStream Object

The StreamReader/StreamWriter and BinaryReader/BinaryWriter objects allow you to read from or write to text and binary files through a FileStream object. To prepare your application to write to a text file, you must set up a FileStream object, which is the channel between your application and the file. There are many ways to set up a FileStream object and associate it with a file, and they're described in the following sections. All the objects are contained in the `System.IO` namespace, so don't forget to import the `System.IO` namespace in your projects that perform file input/output. It's the same namespace that exposes the Directory, File, Path, and other classes discussed so far.

The FileStream object exposes a few members, that convey information about the file you're accessing through a FileStream variable. We'll cover these members here, and then we'll discuss the Reader and Writer objects that actually write data to or read it from files.

The FileStream object's constructor is overloaded; its most common forms require that you specify the path of the file and the mode in which the file will be opened (for reading, appending, writing, and so on). The simpler form of the constructor is

```
Dim FS As New FileStream(path, fileMode)
```

The *fileMode* argument is a member of the FileMode enumeration (see Table 13.3). It's the same argument used by the Open method of the File class. Also similar to Open method of the File class, another overloaded form of the constructor allows you to specify the file's access mode, and the syntax of this method is

```
Dim FS As New FileStream(path, fileMode, fileAccess)
```

The last argument is a member of the FileAccess enumeration (see Table 13.4). The last overloaded form of the constructor accepts a fourth argument, which determines the file's sharing mode:

```
Dim FS As New FileStream(path, fileMode, fileAccess, fileShare)
```

The *fileShare* argument's value is a member of the FileShare enumeration (see Table 13.5).

Properties

You can use the following properties of the FileStream object to retrieve information about the underlying file.

CanRead

This read-only property determines whether the current stream supports reading. If the file associated with a specific FileStream object can be read, this property returns True.

CanSeek

This read-only property determines whether the current stream supports seeking. A seek operation in the context of files doesn't locate a specific value in the file. It simply moves the current position to any location within the file.

CanWrite

This read-only value determines whether the current stream supports writing. If the file associated with a specific FileStream object can be written to, this property returns True.

Length

This read-only property returns the length of the file associated with the FileStream in bytes.

Position

This property gets or sets the current position within the stream. You can compare the Position property to the Length property to find out whether you have reach the end of an existing file. When these two properties are equal, there are no more data to read.