

## **Maximize Your Programming Experience by Using ArrayList instead of Traditional Array in Visual Basic .NET**

**Sam Hijazi**  
**Professor of Computer Technology**  
**Florida Keys Community College**  
**5901 College Road**  
**Key West, Florida 33040**  
**(305) 809-3541**

### **Abstract**

This paper compares the use of a traditional array, as data a structure, to an ArrayList, as a class with powerful methods and properties, in Visual Basic .Net. One of the most significant drawbacks of using arrays is their inability to adjust to different sizes. ArrayLists can be added or deleted dynamically at run time. This paper should provide the reader with a comprehensive comparison between Arrays and ArrayLists. The obvious benefit of this research is to show the flexibility and the ease of using ArrayLists in order to maximize and produce elegant code using Visual Basic .Net.

### **What is ArrayList?**

The ArrayList class is found in the system.collection namespace. The ArrayList is an extremely effective and helpful class in VB .net (Roith, 2002). Roith continued by stating that ArrayList is dissimilar to an array because the number of its objects (elements) can grow dynamically. For example, a programmer can easily add an object at the end of the list by using the method Add(). Also multiple elements can be added by using the method AddRange(). Also inserting an object somewhere with the list of elements can be added by using either the method Insert() or InsertRange(), which allows for the insertion of multiple values. Finally, Roith stated that similarly a programmer can remove an item or multiple items by using Remove() and RemoveRange() respectively. Obviously from the above discussion ArrayList can expand or shrink.

### **Why ArrayList?**

The most obvious reason for using ArrayList, in comparison to regular array, is that ArrayList can be expanded in runtime. While arrays are helpful and much more effective comparing to individual variables, they are inflexible and fixed in their size (Doke and Williams, 2005). The authors noted that “it is extremely difficult to change the number of array elements as your code is executing.” Further a programmer can resize the number of elements dynamically with an ArrayList.

Since ArrayList is a .NET technology, it is available for C# also. When programmers switch from C or C++ to C#, they realize that there is no dynamic memory allocation (CSharpFriends.com, n.d.). Further, the reason for not having the dynamic memory allocation is

that ArrayList can do the same functionality but easier. As a result, programmers don't have to worry about freeing the memory or concern themselves with "array bound overflow."

## **Helpful Properties and Methods Used in ArrayList**

### **Common Properties**

The following list of ArrayList properties was selected from the Microsoft Developer Network (MSDN). To see the complete list of the properties for the ArrayList class, visit Microsoft at: [http://msdn2.microsoft.com/en-US/library/system.collections.arraylist\\_members\(VS.80\).aspx](http://msdn2.microsoft.com/en-US/library/system.collections.arraylist_members(VS.80).aspx)

<b>Name</b>	<b>Description</b>
<a href="#"><u>Capacity</u></a>	"Gets or sets the number of elements that the <b>ArrayList</b> can contain."
<a href="#"><u>Count</u></a>	"Gets the number of elements actually contained in the <b>ArrayList</b> ."

### **Common Methods**

To give the reader an idea about the enormous functionality of an ArrayList, the following list of ArrayList methods was selected from The Microsoft Developer Network (MSDN). To see the complete list of the methods for the ArrayList class, visit Microsoft at: [http://msdn2.microsoft.com/en-US/library/system.collections.arraylist\\_members\(VS.80\).aspx](http://msdn2.microsoft.com/en-US/library/system.collections.arraylist_members(VS.80).aspx)

<b>Name</b>	<b>Description</b>
<a href="#"><u>Add</u></a>	"Adds an object to the end of the <b>ArrayList</b> ."
<a href="#"><u>AddRange</u></a>	"Adds the elements of an <a href="#"><u>ICollection</u></a> to the end of the <b>ArrayList</b> ."
<a href="#"><u>BinarySearch</u></a>	"Overloaded. Uses a binary search algorithm to locate a specific element in the sorted <b>ArrayList</b> or a portion of it."
<a href="#"><u>Clear</u></a>	"Removes all elements from the <b>ArrayList</b> ."
<a href="#"><u>Clone</u></a>	"Creates a shallow copy of the <b>ArrayList</b> ."
<a href="#"><u>Contains</u></a>	"Determines whether an element is in the <b>ArrayList</b> ."
<a href="#"><u>CopyTo</u></a>	"Overloaded. Copies the <b>ArrayList</b> or a portion of it to a one-dimensional array."
<a href="#"><u>Equals</u></a>	"Overloaded. Determines whether two <a href="#"><u>Object</u></a> instances are equal. (Inherited from <a href="#"><u>Object</u></a> .)"
<a href="#"><u>FixedSize</u></a>	"Overloaded. Returns a list wrapper with a fixed size, where elements are allowed to be modified, but not added or removed."
<a href="#"><u>GetRange</u></a>	"Returns an <b>ArrayList</b> which represents a subset of the elements in the source <b>ArrayList</b> ."
<a href="#"><u>GetType</u></a>	"Gets the <a href="#"><u>Type</u></a> of the current instance. (Inherited from <a href="#"><u>Object</u></a> .)"

<a href="#">IndexOf</a>	“Overloaded. Returns the zero-based index of the first occurrence of a value in the <b>ArrayList</b> or in a portion of it.”
<a href="#">Insert</a>	“Inserts an element into the <b>ArrayList</b> at the specified index.”
<a href="#">InsertRange</a>	“Inserts the elements of a collection into the <b>ArrayList</b> at the specified index.”
<a href="#">LastIndexOf</a>	“Overloaded. Returns the zero-based index of the last occurrence of a value in the <b>ArrayList</b> or in a portion of it.”
<a href="#">ReadOnly</a>	“Overloaded. Returns a list wrapper that is read-only.”
<a href="#">Remove</a>	“Removes the first occurrence of a specific object from the <b>ArrayList</b> .”
<a href="#">RemoveAt</a>	“Removes the element at the specified index of the <b>ArrayList</b> .”
<a href="#">RemoveRange</a>	“Removes a range of elements from the <b>ArrayList</b> .”
<a href="#">Reverse</a>	“Overloaded. Reverses the order of the elements in the <b>ArrayList</b> or a portion of it.”
<a href="#">SetRange</a>	“Copies the elements of a collection over a range of elements in the <b>ArrayList</b> .”
<a href="#">Sort</a>	“Overloaded. Sorts the elements in the <b>ArrayList</b> or a portion of it.”
<a href="#">ToArray</a>	“Overloaded. Copies the elements of the <b>ArrayList</b> to a new array.”
<a href="#">ToString</a>	“Returns a <a href="#">String</a> that represents the current <b>Object</b> . (Inherited from <a href="#">Object</a> .)”
<a href="#">TrimToSize</a>	“Sets the capacity to the actual number of elements in the <b>ArrayList</b> .”

### **Example One**

This first example is to show the dynamic nature of an ArrayList. In line 7 below, myArrayList was created as an ArrayList with four elements. Lines 9 through 11, show how to assign values to the elements of myArrayList using the method Add(). As a result of the adding the three values, the following will take place in the memory:

- myArrayList(0) = “Bit”
- myArrayList(1) = “Byte”
- myArrayList(2) = “Kilobyte”

In order to show a value on the screen in console mode, we need to use console class and WriteLine( ) or Write( ) method. Once the method WriteLine( ) finishes printing a line, it advance the carriage return to the next line, while the method Write( ) doesn't.

Line 14 prints the capacity of myArrayList, using the MyArrayList.Capacity( )method. The result is 4. We reserved 4 items for the ArrayList initially and we used 3, therefore the capacity has not been totally used. It is still 4. When we used the method Count( ) with myArrayList.count(), it shows in the output the value 3. The value 3 indicates that only 3 elements out of the 4 have been used.

Line 18 and 19 added two more values to myArrayList. Wait a minute! Wasn't myArrayList created to hold only 4 elements initially? Yes, but adding two more values will exceed the original size of myArrayList, therefore myArrayList immediately doubled its capacity to 8 elements. This is exactly the reason for calling ArrayList a dynamic structure.

Printing the new capacity and the number of items used in myArrayList seen in line 21 through 24 will prove that the capacity is 8 and the number of the assigned items is 5.

It is interesting to see how ArrayList doubles its own capacity (size) dynamically as soon as we exceed the original size by one more element. This flexibility is very hard to obtain using array, especially during running time.

```
1      Imports System
2
3      Module Module1
4
5          Sub Main()
6
7              Dim myArrayList As New ArrayList(4)
8
9              myArrayList.Add("Bit")
10             myArrayList.Add("Byte")
11             myArrayList.Add("Kilobyte")
12
13             Console.WriteLine("Initial Addition of Elements:")
14             Console.WriteLine("The capacity of MyArrayList is ----> " & MyArrayList.Capacity)
15             Console.WriteLine("The number of used items is -----> " & MyArrayList.Count)
16             Console.WriteLine("-----")
17
18             MyArrayList.Add("Megabyte")
19             MyArrayList.Add("Gigabyte")
20
21             Console.WriteLine("After Adding Two More Elements:")
22             Console.WriteLine("The capacity of MyArrayList is ----> " & MyArrayList.Capacity)
23             Console.WriteLine("The number of used items is -----> " & MyArrayList.Count)
24             Console.WriteLine("-----")
25         End Sub
26
27     End Module
```

### Output Screen for Example One

```
Initial Addition of Elements:
The capacity of ArrayListEx1 is ---> 4
The number of used items is -----> 3
-----

After Adding Two More Elements:
The capacity of ArrayListEx1 is ---> 8
The number of used items is -----> 5
-----

Press any key to continue.
```

### Example 2

This example shows selected methods used by ArrayList. The program starts by adding five elements to myArrayList. The initial capacity of myArrayList is 4, by the time the program executes line 11, the number of the elements will be doubled to 8.

Using the remove ( ) method shown in line 13, it will delete the item with the value “Ghz.” Line 14 utilize the insert ( ) method to insert the value “Byte” in the second position of myArrayList, since ArrayLists, similar to arrays, start with index 0.

Line 15 apply the method IndexOf ( ) to locate which entry is assigned to the value “Megabyte.” The output below shows the result of using IndexOf ( ) method which returns the value 3 since “Megabyte” is the element number 4 in the list. Another helpful method is Contains ( ), which

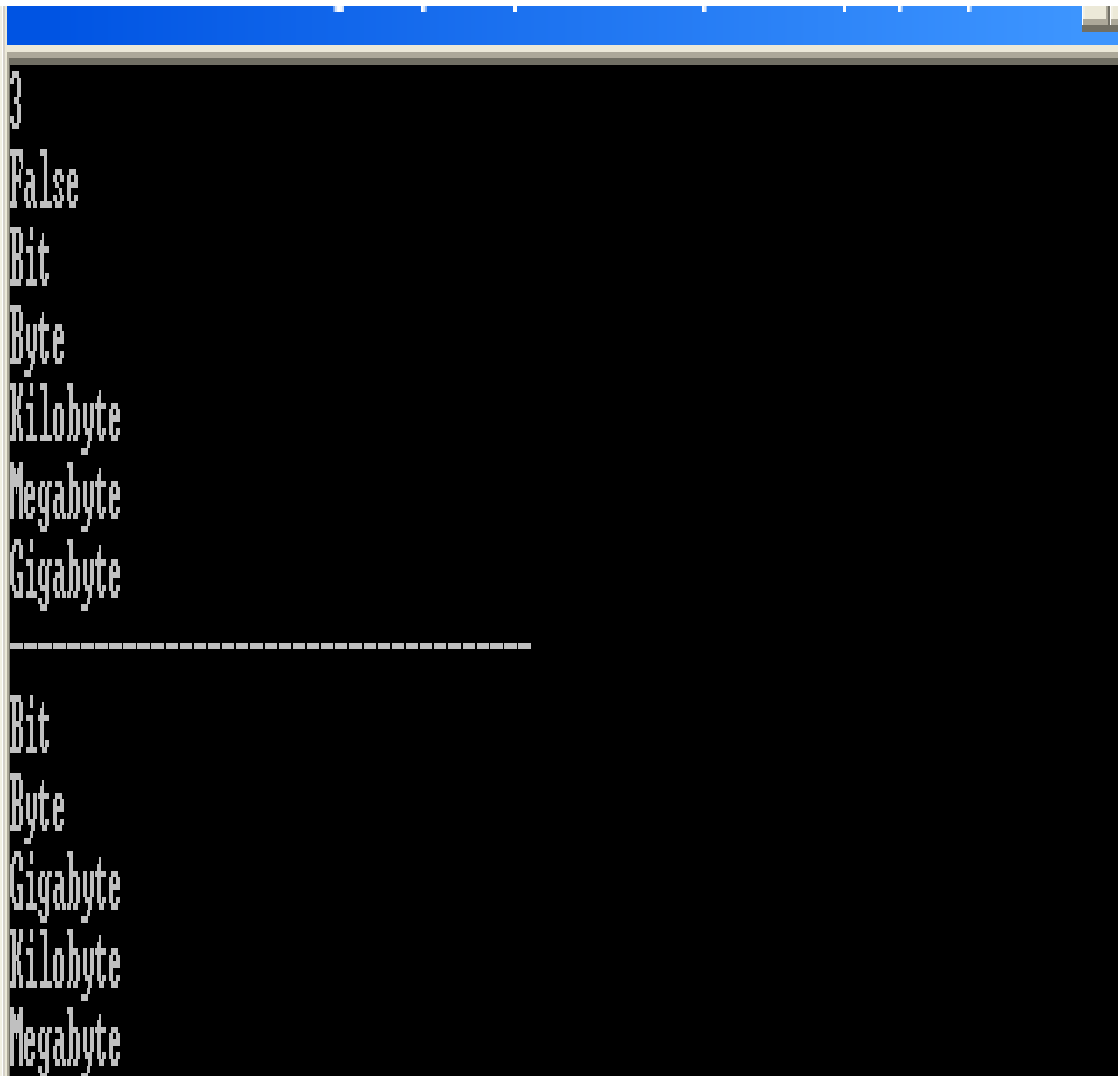
returns true if the value is found, else it will return false. From the output below, the returned value is false since the value “Terabyte” is not contained in myArrayList.

The function printArrayList ( ) receives myArrayList and prints every elements in. This is shown below in the output. Immediately after printing myArrayList elements, the method Sort() was applied to myArrayList. myArrayList elements are sorted in ascending order as shown in the output screen output.

Other methods can be used as easily. ArrayList is a very powerful data structure. The above mentioned method can and should save programmers hours if they were to rely on using traditional arrays.

```
1      Imports System
2
3      Module Module1
4          Sub Main()
5              Dim myArrayList As New ArrayList(4)
6
7              myArrayList.Add("Bit")
8              myArrayList.Add("Kilobyte")
9              myArrayList.Add("Ghz")
10             myArrayList.Add("Megabyte")
11             myArrayList.Add("Gigabyte")
12
13             myArrayList.Remove("Ghz")
14             myArrayList.Insert(1, "Byte")
15             Console.WriteLine(myArrayList.IndexOf("Megabyte"))
16             Console.WriteLine(myArrayList.Contains("Terabyte"))
17
18
19             printArrayList(myArrayList)
20             myArrayList.Sort()
21
22             printArrayList(myArrayList)
23         End Sub
24         Console.WriteLine("-----")
25
26         Sub printArrayList(ByVal arrList As ArrayList)
27             Dim i As Integer
28             For i = 0 To arrList.Count - 1
29                 Console.WriteLine(arrList.Item(i))
30             Next
31         End Sub
32     End Module
```

### Output Screen for Example Two



```
3
False
Bit
Byte
Kilobyte
Megabyte
Gigabyte
-----
Bit
Byte
Gigabyte
Kilobyte
Megabyte
```

## Conclusion

ArrayList is a very powerful structure. It contains helpful properties (characteristics) and powerful methods (processes) that save programmers hours. An ArrayList is dynamic which allows for the expansion or shrinking of the number of its own elements. This ability itself provides programmers with the capacity to maximize their code without having to worry about re-dimensioning their arrays frequently, since ArrayLists have come to the rescue.

## Reference

ArrayList Members (2006). MSDN-Microsoft. Retrieved April 15, 2006, from Web site: [http://msdn2.microsoft.com/en-US/library/system.collections.arraylist\\_members\(VS.80\).aspx#](http://msdn2.microsoft.com/en-US/library/system.collections.arraylist_members(VS.80).aspx#)

CSharpFriends.com (n.d.). ArrayList: Dynamic Array in .NET. Retrieved April 15, 2006, from Web site: <http://www.csharpfriends.com/Articles/getArticle.aspx?articleID=56>.

Doke, R. E. & Wilimas. S. R. (2005). *Microsoft Visual Basic .NET: From Problem Analysis to Program Design*. Printed in Canada: Thomson-Course Technology.

Roith, J. (2002). *ArrayList - Introduction*. Retrieved April 15, 2006, from Web site: [http://www.gotmono.com/docs/base\\_classes/arraylist.html](http://www.gotmono.com/docs/base_classes/arraylist.html)