Introduction to Arrays

The concept of an array goes back to mathematics; it’s either a vector or a matrix. You will find that with an array, it's easy to deal with many values that are the same, but need to be stored and accessed individually.

In the real world you can find instances of "array-like" entities. For example, a locker room has one room for all the lockers. Each locker has different data in it. ALL lockers each have a unique number (index) that allows you to find the locker you want very quickly. You can also think of mail boxes at a post office. Once again, you go to one location for all of the post office boxes. Each box has a unique number (index) to allow you to find the box very quickly.

An array is a group of data that can be accessed under a single variable name. Instead of having to create multiple variables when you wish to reference many different data items that are all similar, you can use an array to store each data item in its own separate compartment as shown in Figure 1.

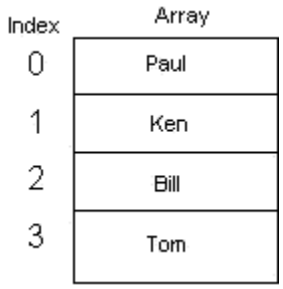

Figure 1. An example of an Array

Each element in an array is referenced with a number, called an index. Array elements start with 0 in any of the .NET languages.

Arrays all have the following characteristics:

- Arrays elements start with the number zero (0).
- All elements of the array must have the same data type.
- A variant array may have different data types in each element.
- Arrays may be public or local in scope.
- Arrays may be a fixed size or dynamically allocated at runtime.

Using Arrays

There are many objects in the .NET Framework that return an array. In the sample, shown in Figure 2, you will learn how to convert a string to an array, how to retrieve a list of files in a folder, how to retrieve a list of folders within a path and how to work with lines of text in a text box.

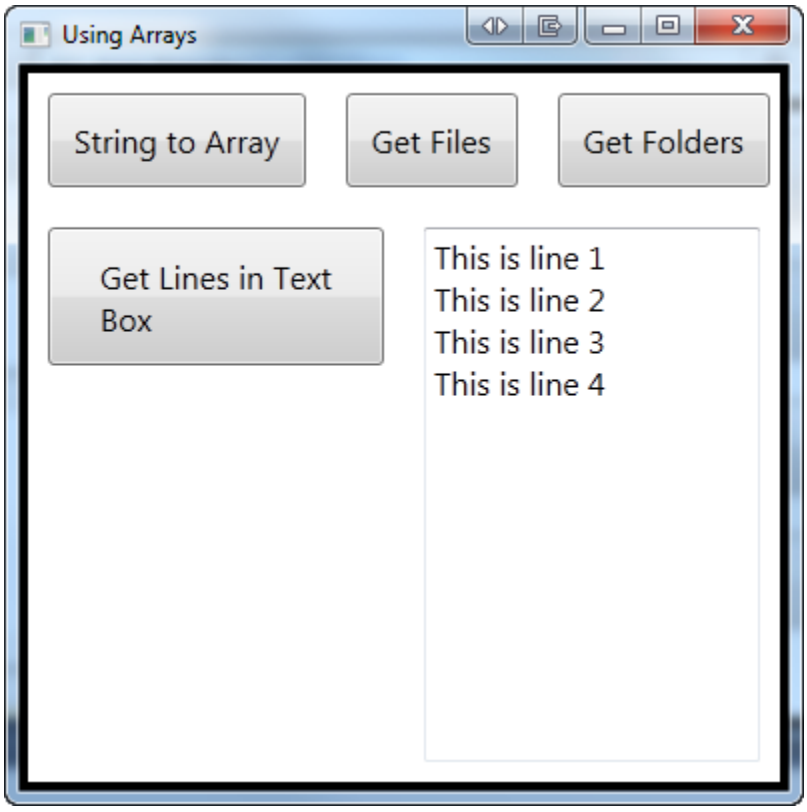

Figure 2. Example of using Arrays

Converting a String to an Array

The first button on the above form will convert a string of comma delimited names to an array.

```
private void StringToArray()
{
  string[] values = null;
  string names = null;
  int index = 0;

  names = "Ken,Michael,Bruce,Paul";

  values = names.Split(",".ToCharArray());

  for (index = 0; index <= values.Length - 1; index++)
  {
    Debug.WriteLine(values[index]);
  }
}
```

In the routine above you use the Split method of the String object to return an array. The Split method requires that you pass a Char array of delimiters to look for. When the Split method finds any of the delimiters passed, it will take the word immediately before that delimiter and add it to an array. In the code above you will pass a comma as the delimiter.

Once you have the array you can loop through each value using a **for** loop. You retrieve the total number of values within the array using the Length property.

Retrieving a List of Files

The "Get Files" button on the form in Figure 2 will retrieve a list of files from a specified path.

```
private void GetFiles()
{
  string[] files = null;

  files = System.IO.Directory.GetFiles("D:\\");

  foreach (string name in files)
  {
    Debug.WriteLine(name);
  }
}
```

The method GetFiles which is part of the Directory class will return an array of all of the files in the specified path you pass to it. In the above example, you are retrieving all files from the root of the D drive. Instead of using a **for** loop

and an incrementing variable use a **foreach** loop. The **foreach** loop is slightly more efficient than using an incrementing number.

Retrieving a List of Folders

The "Get Folders" button on the form will retrieve a list of folders from a specified path.

```
private void GetFolders()
{
  string[] folders = null;

  folders = System.IO.Directory.GetDirectories("D:\\");

  foreach (string name in folders)
  {
    Debug.WriteLine(name);
  }
}
```

This routine is exactly like the previous one except you are now calling the GetDirectories method on the Directory class.

Retrieving Lines of Text

If you create a text box that has the MultiLine property set to True and you place several lines of text in the text box with a carriage-return, line-feed character in between each line, you can use the Lines method on the Text box control to retrieve an array of each line.

```
private void LinesInTextBox()
{
  string[] lines = null;

  // Create Array to size of lines
  lines = new string[txtLines.LineCount + 1];

  for (int index = 0; index <= txtLines.LineCount - 1;
        index++)
  {
    lines[index] = txtLines.GetLineText(index);
  }

  // Display Each Line
  foreach (string line in lines)
  {
    Debug.Write(line);
  }
}
```

Once again, the code should look very familiar at this point. On the form is a text box named *txtLines*. The LineCount property returns how many lines are in the text box. You can use this count to create an array of lines. You then use the GetLineText method to get each line from the text box individually and add them to the array.

Creating Your Own Arrays

Besides using the built-in objects within .NET that return arrays, you can also create your own arrays.

Fixed-Size Arrays

An array can be dimensioned to a certain fixed size at declaration time. To declare a fixed-size array, you place a number within square brackets following the variable name.

```
private void FixedArray()
{
  string[] names = new string[3];

  names[0] = "Ken";
  names[1] = "Paul";
  names[2] = "Michael";

  foreach (string name in names)
  {
    Debug.WriteLine(name);
  }
}
```

The code above declares an array that is declared to have 3 elements numbered 0 to 2. After you have declared an array, you can then place values into each element using the square brackets [] with an index number.

Dynamic Arrays

Many times you do not know the number of elements you will need in an Array until runtime. If this happens you can create an array with no elements, and then re-dimension that array at runtime to a particular size.

```
private void DynamicArrays()
{
 int[] values;

 // Dynamically create 3 elements
 values = new int[3];

 values[0] = 10;
 values[1] = 20;
 values[2] = 30;

 foreach (int value in values)
 {
  Debug.WriteLine(string.Format("value={0}", value));
 }
}
```

Using the square brackets with no number in it lets the compiler know that you will be using an array of **int**s, but you don't know how many elements yet. Later in the code you can then create the number of elements using the **new** keyword. The value in the square brackets can be a hard coded value (as shown), or could be a variable.

Initializing Arrays at Declaration Time

If you have a very small array you can initialize the values when you declare the variable as shown in the code below:

```
private void InitArray()
{
  int[] values = { 10, 20, 30, 40 };

  foreach (int value in values)
  {
    Debug.WriteLine(string.Format("value={0}", value));
  }
}
```

Using the braces {} you can enter as many comma delimited values for the initialization of the array as you want. However many values you have in the braces will become the total amount of the elements in the array.

Summary

Arrays are used when you need to store many elements under a common name. There are many uses for an array, however you should also look into using collection classes as well.