MultiSelect ListBoxes in Visual Basic 6

Most beginner Visual Basic programmers are familiar with one of my favorite controls--the ListBox.

The ListBox permits the programmer to load it up with a number of items, allow the user to make a selection, and then proceed accordingly. ListBoxes are ideal controls for presenting a list of choices to the user---whether a few items or a large number, ListBoxes are an ideal control to use.

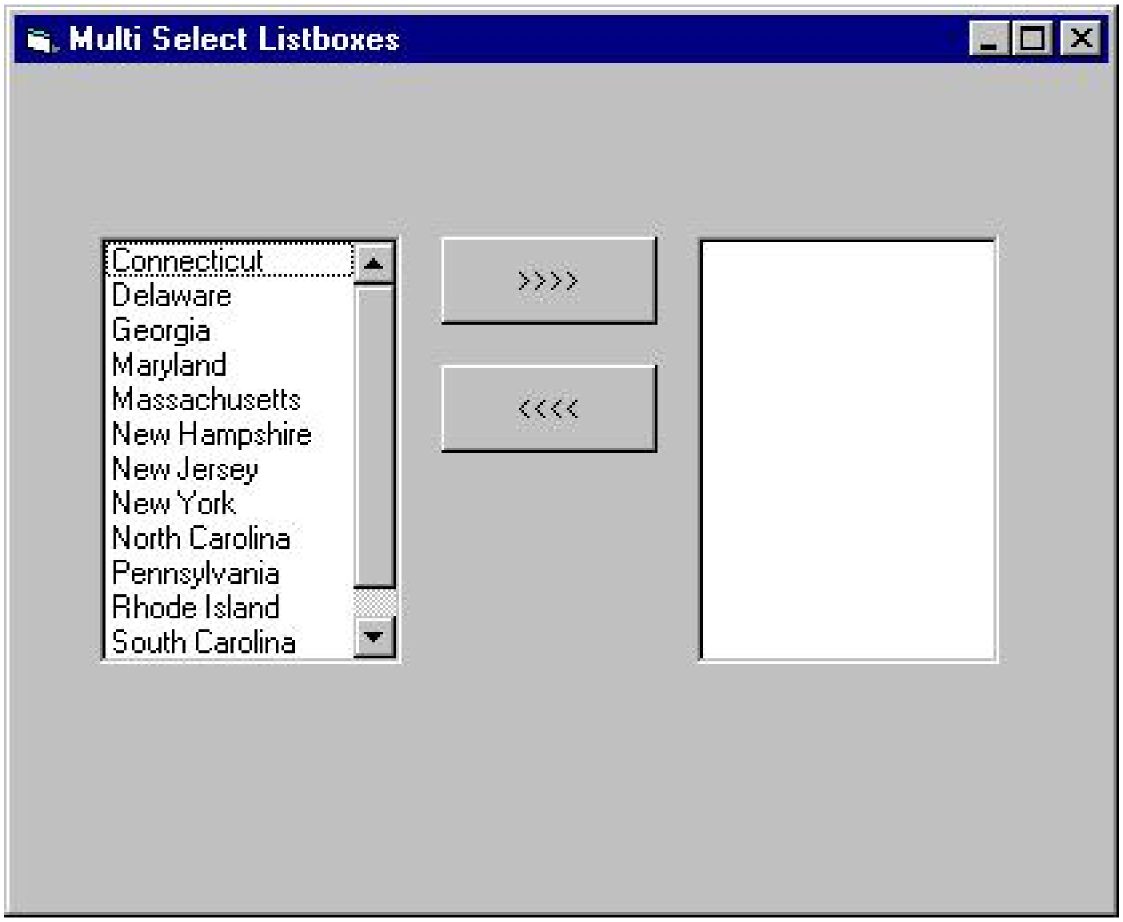
The AddItem Method

Most beginner programmers believe that the user's selection in a ListBox is limited to a single item. After all, when the user makes a selection in a ListBox, if they click on another item, the currently selected item is 'unselected'. Let's create a form with two ListBoxes and two command buttons, and use this code in the Load Event Procedure of the form to load into the first ListBox the original 13 colonies of the United States…

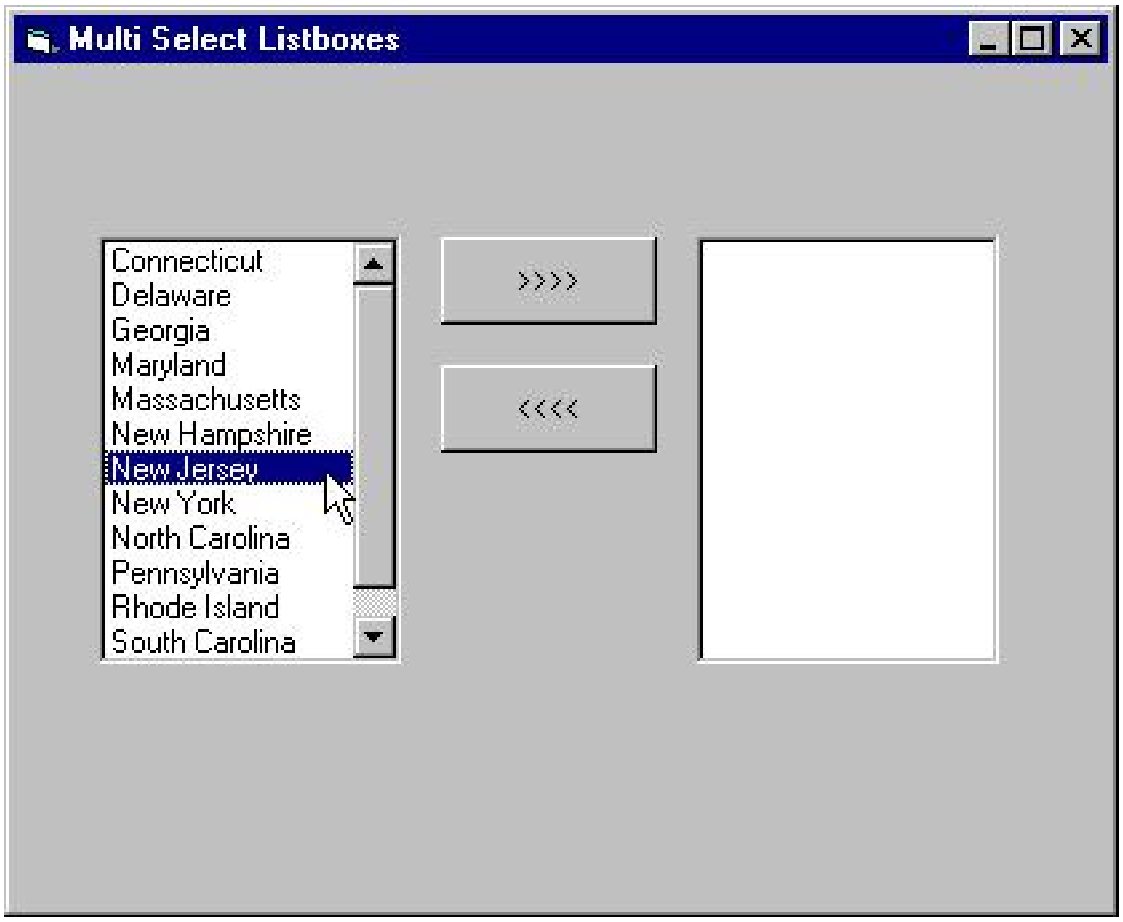
```
Private Sub Form_Load()

With List1
   .AddItem "Connecticut"
   .AddItem "Delaware"
   .AddItem "Georgia"
   .AddItem "Maryland"
   .AddItem "Massachusetts"
   .AddItem "New Hampshire"
   .AddItem "New Jersey"
   .AddItem "New York"
   .AddItem "North Carolina"
   .AddItem "Pennsylvania"
   .AddItem "Rhode Island"
   .AddItem "South Carolina"
   .AddItem "Virginia"
End With

End Sub
```

AddItem, for those of you unfamiliar with it, is a method that will add an item to a ListBox (You can also add items via the List Property of the ListBox at design time, but using the method is a lot more dynamic…)

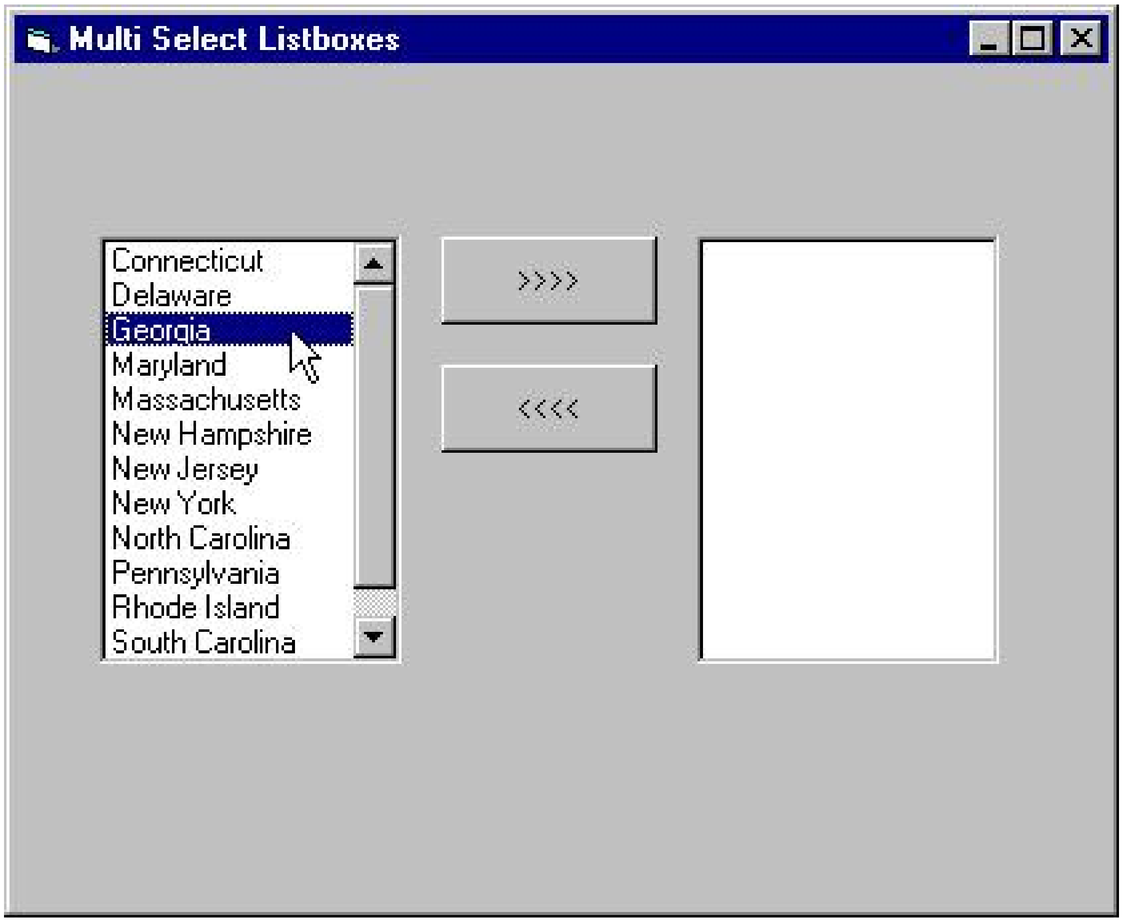
When we run the program and our form appears, this is what we'll see…



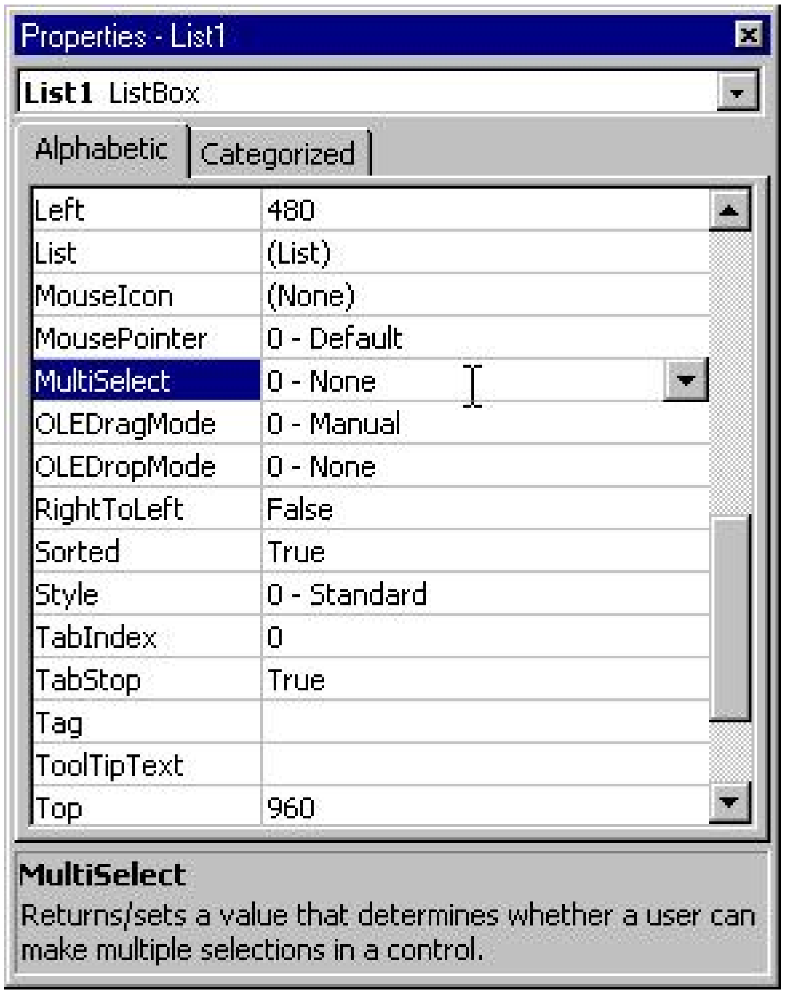
Notice that when we click on an item in the ListBox, the item is selected…



and when we click on a second item in the ListBox, that item is selected, while the originally selected item is then ‘unselected’…



This is the standard behavior of the ListBox—and is in fact dictated by the fact that the default value of the MultiSelect Property of the ListBox is set to 'None'…



Left	480
List	(List)
MouseIcon	(None)
MousePointer	0 - Default
MultiSelect	0 - None
OLEDragMode	0 - Manual
OLEDropMode	0 - None
RightToLeft	False
Sorted	True
Style	0 - Standard
TabIndex	0
TabStop	True
Tag	
ToolTipText	
Top	960

MultiSelect
Returns/sets a value that determines whether a user can make multiple selections in a control.

A little ListBox magic…

Before I discuss how to change the value of the MultiSelect Property of the ListBox to permit multiple items to be selected at once, let's take a look at a pretty common task concerning ListBoxes, and how we could accomplish it—that is, 'moving' selected items from one ListBox to another.

In reality, there is no built in way to accomplish this task---moving a selected item from one ListBox to another involves adding that item to the second ListBox using the AddItem Method of the ListBox and then removing the selected item from the first ListBox by using the RemoveItem Method of the ListBox.

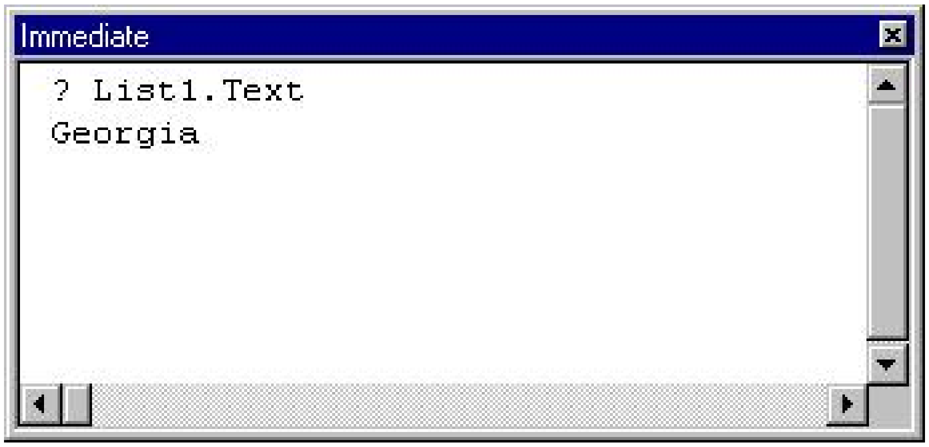
The trick here is to be able to detect which item in the ListBox is selected. With a ListBox in which only one item can be selected, there are actually two ways to identify the selected item.

The Text Property of the ListBox

First, you can use the Text property of the ListBox, which contains the text corresponding to the selected item in the ListBox. For instance, with 'Georgia' selected in the ListBox, if we pause the program and type this statement into the Immediate Window

? List1.Text

we'll discover that the Text Property of the ListBox is 'Georgia'…



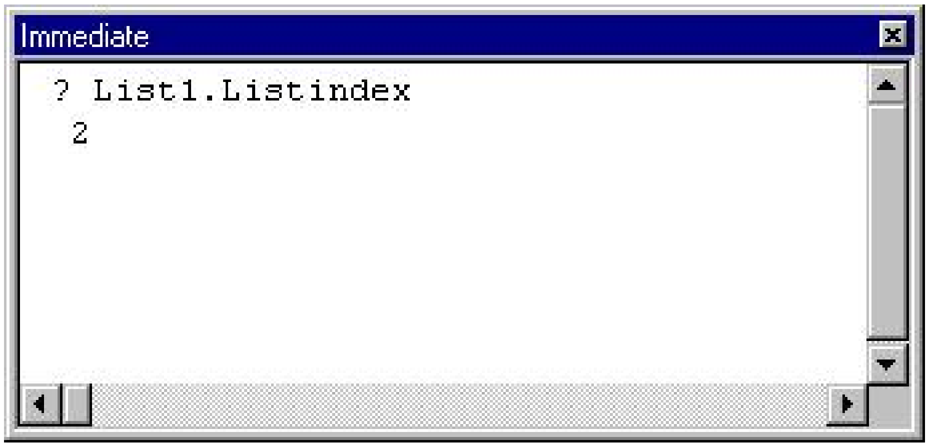
```
? List1.Text
Georgia
```

The ListIndex Property of the ListBox

A second way is detect the item selected in the ListBox is to use the ListIndex property of the ListBox. If the word fragment 'Index' triggers the recollection of an array, you're right---as it turns out, as each item is added to a ListBox, a corresponding 'behind the scenes' array is created, with an element added for each item in the ListBox. A ListBox containing 13 entries creates a 13 element array, with elements numbered from 0 to 12. The ListIndex property of the ListBox corresponds to the 0 based element number of the selected item in this array---therefore, if 'Georgia' is selected in the ListBox, since it's the third item in the ListBox, the ListIndex property of the ListBox is 2 (remember, the element numbers start with 0). With Georgia selected in the ListBox, let's go back to the Immediate Window, and type this statement into it

? List1.ListIndex

As you can see, VB is telling us that the item selected in the ListBox is element number 2 (the third item) corresponding to 'Georgia'



```
? List1.Listindex
 2
```

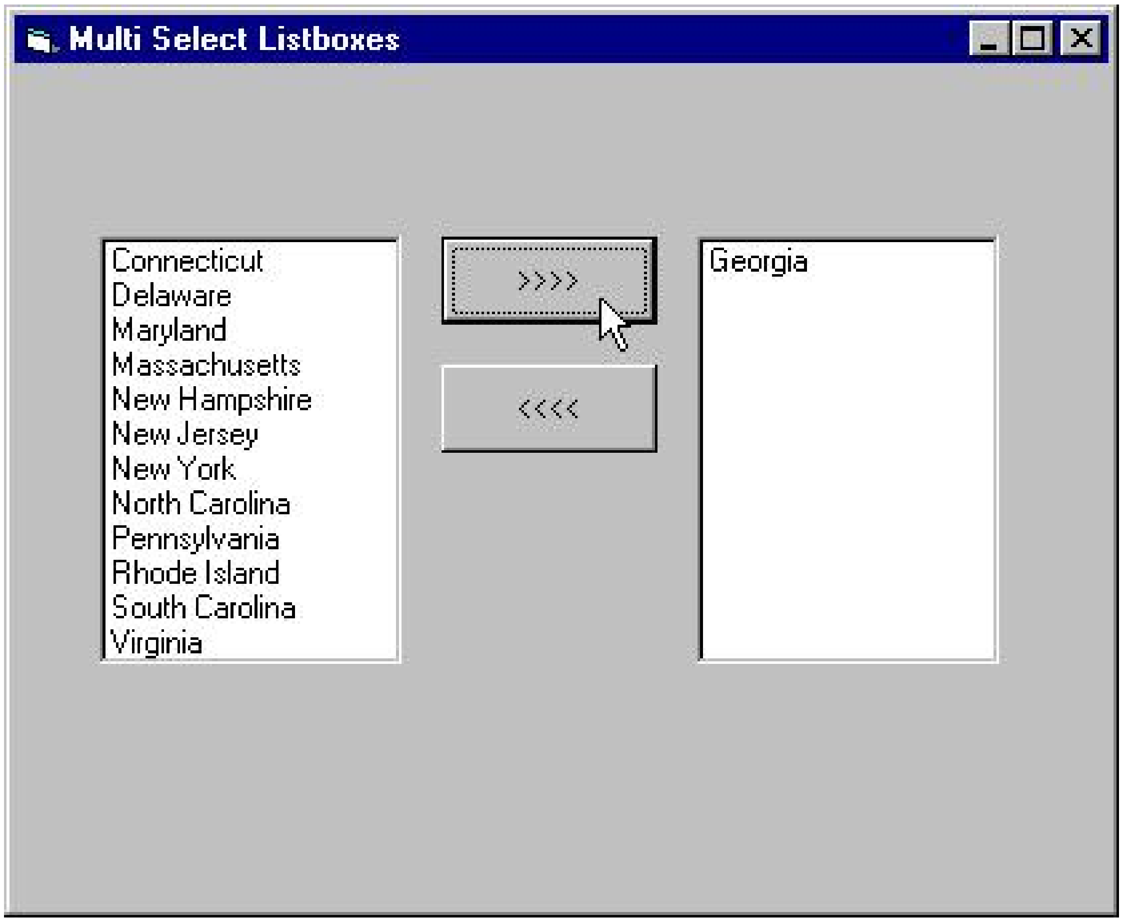
Using these two properties of the ListBox, we can now place code into our top most command button to 'move' the selected item from the ListBox on the left to the ListBox on the right. As I mentioned previously, we'll execute the ***AddItem*** Method of the second ListBox, using the Text property of the first ListBox as an argument. Then we'll execute the ***RemoveItem*** method of the first ListBox, using the ListIndex property of the ListBox as an argument (the RemoveItem method requires the numeric Index of the item in the ListBox). Here's the code…

```
Private Sub cmdMovetoRight_Click()

If List1.ListIndex = -1 Then Exit Sub

List2.AddItem List1.Text
List1.RemoveItem List1.ListIndex

End Sub
```

The first thing we do is check to see if the ListIndex property of the ListBox is -1. A Minus One (-1) is Visual Basic's way to telling us that no item is selected in the ListBox, and it that's the case, we exit the procedure by executing the Exit Sub statement.

If an item is selected, then we add that item to the right-hand ListBox, and then remove it from the left-hand ListBox using the RemoveItem method. Notice that we add the item to the right-hand ListBox BEFORE removing it from the left-hand ListBox—otherwise our program will bomb.

Now when we execute the program, select 'Georgia' in the left-hand ListBox, and then click on the 'move to right' command button, the selected item in the left-hand ListBox is moved to the right ListBox.



By the way, to maintain the alphabetical order of the entries in the ListBox when you start moving items around like this, be sure to set the Sorted Property of each ListBox to True.

We can write similar code to ‘move’ items from the right-hand ListBox to the left, like this..

```
Private Sub cmdMoveToLeft_Click()

If List2.ListIndex = -1 Then Exit Sub

List1.AddItem List1.Text
List2.RemoveItem List2.ListIndex

End Sub
```

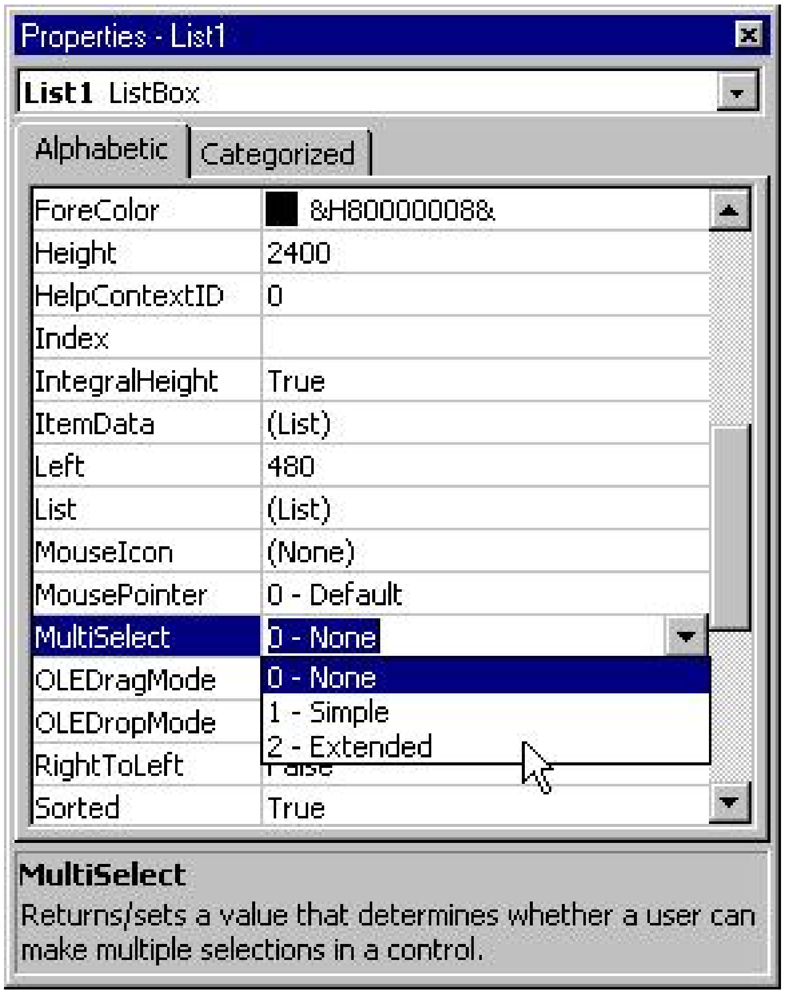
If we now execute the program, we’ll be able to ‘move’ items from one ListBox to another easily.

The MultiSelect Property

As I mentioned earlier, the MultSelect Property of the ListBox allows you to permit the user to make multiple selections in a ListBox. As soon as you permit this to happen, the complexity of determining which (if any) items in the ListBox have been selected increases dramatically. You can no longer depend on the Text Property of the ListBox to determine the item selected, because the Text Property contains the value of the LAST item selected. In the same way, the ListIndex property is of no help, because it too contains the number of the LAST item selected in the List.

Earlier, I stated that a ListBox creates a 'behind the scenes' array of entries (yes, the dreaded Array!!!), and that's what we'll need to use in order to work with multiple selected items in a ListBox.

Before we look at that code (and it's not that bad, believe me), let's take a look at the possible values for the MultiSelect Property.



Properties - List1

List1 ListBox

Alphabetic | Categorized

ForeColor	■ &H80000008&
Height	2400
HelpContextID	0
Index	
IntegralHeight	True
ItemData	(List)
Left	480
List	(List)
MouseIcon	(None)
MousePointer	0 - Default
MultiSelect	0 - None
OLEDragMode	
OLEDropMode	
RightToLeft	False
Sorted	True

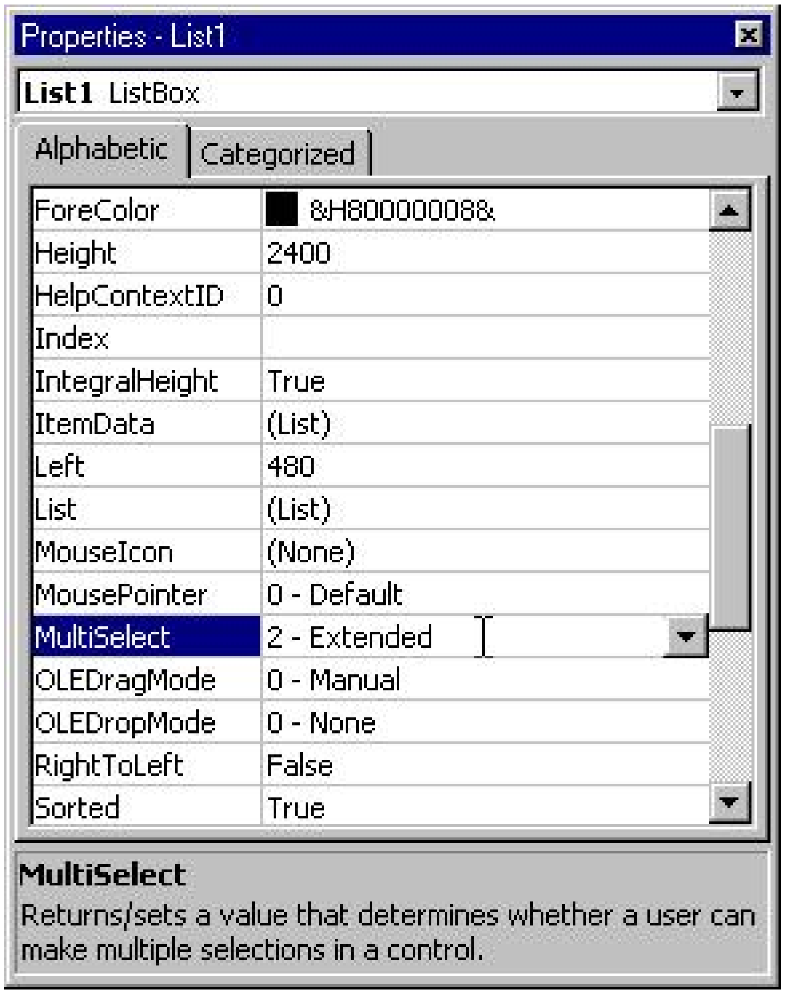
0 - None
1 - Simple
2 - Extended

MultiSelect
Returns/sets a value that determines whether a user can make multiple selections in a control.

As I mentioned earlier, the default value of the MultiSelect property of a ListBox is 0-None—meaning that it's a single selection ListBox. The two MultiSelect values are 1-Simple and 2-Extended. Both permit the user to make multiple selections, but their behavior is slightly different.

2-Extended provides more of a true Windows feel. With 2-Extended, if the user selects an item in the ListBox, presses and holds the Shift key down, and then selects another item in the ListBox, everything in between is selected as well. On the other hand, if the user selects an item, presses and holds the Control key, and then selects another item, both items are selected, and the user can continue to select discrete items like this just by holding the Control key down. This is the behavior of most ListBoxes that you see in Windows programs. With the 1-Simple setting, a mouse click or pressing the SPACEBAR selects or deselects an item in the list.

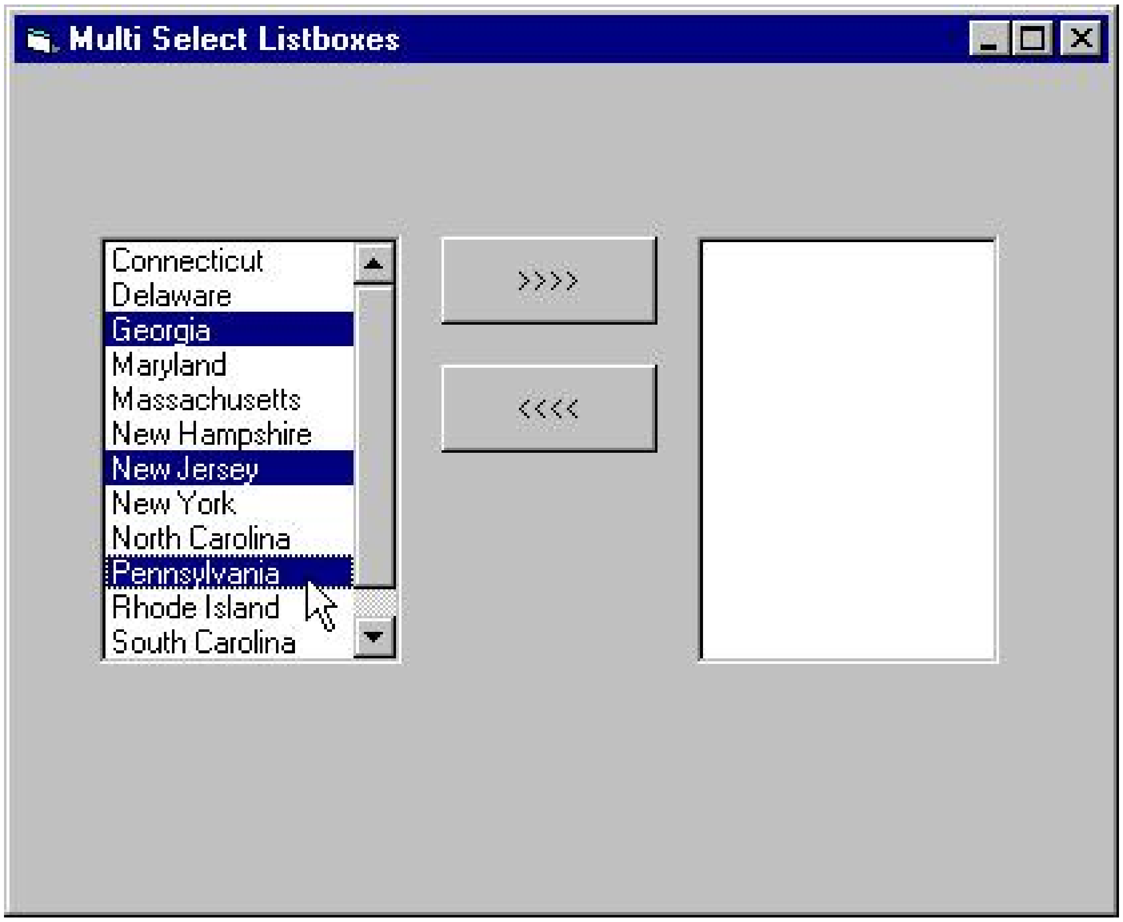
Let’s change the MultiSelect property of both ListBoxes to 2-Extended, and check out the new behavior…



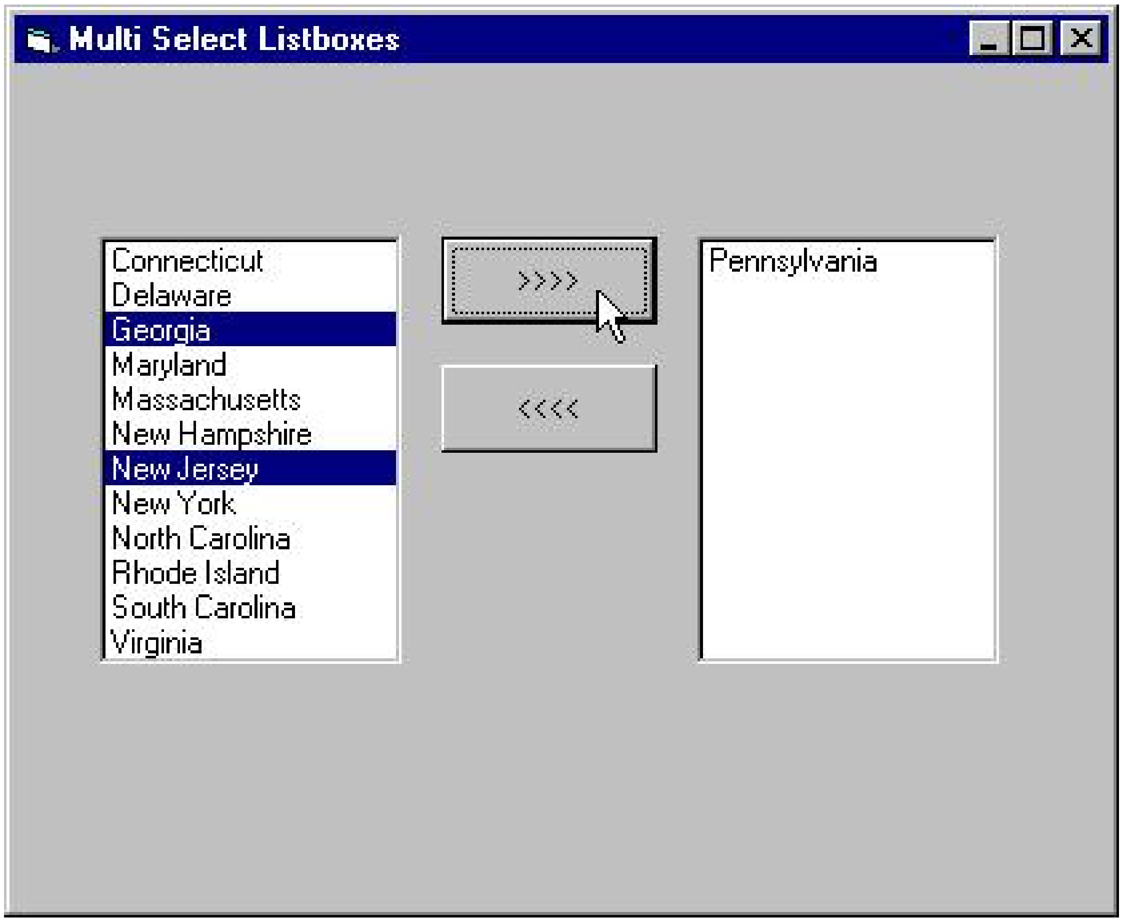
ForeColor	■ &H80000008&
Height	2400
HelpContextID	0
Index	
IntegralHeight	True
ItemData	(List)
Left	480
List	(List)
MouseIcon	(None)
MousePointer	0 - Default
MultiSelect	2 - Extended
OLEDragMode	0 - Manual
OLEDropMode	0 - None
RightToLeft	False
Sorted	True

MultiSelect
Returns/sets a value that determines whether a user can make multiple selections in a control.

Now if we run the program, by using the Ctrl Key plus the mouse button, we’ll be able to make multiple selections…



Unfortunately, if we try to move ALL of the items to the right-hand ListBox by clicking on the ‘movetoright’ Command Button, only the last item selected moves…



Never fear! We can make this work, we just need to modify the code to use that ‘behind the scenes’ array I keep talking about. What we’ll do is use Loop processing to take advantage of a property of the ListBox I haven’t mentioned yet, the Selected Property. I’ll display the code first, and then I’ll explain it…

```
Private Sub cmdMovetoRight_Click()

Dim i As Integer

If List1.ListIndex = -1 Then Exit Sub

For i = List1.ListCount - 1 To 0 Step -1
    If List1.Selected(i) = True Then
      List2.AddItem List1.List(i)
      List1.RemoveItem i
    End If
Next i

End Sub
```

First of all, those of you who have read my books know that I never name variables with a single character, but I’m trying to make sure that the code sits into the width of a code window (the variable ‘I’ really should be ‘intCounter’).

With that out of the way, what we’re doing here is using a loop (in this case a For…Next Loop) to move through the elements of that ‘behind the scenes’ array I’ve been mentioning. The first element in the array is 0, and the last element is equal to the total number of elements in the array, minus 1 (that’s because the array is numbered starting with the number 0). In other words, if the ListBox contains 5 items, it’s last element is number 4.

How can we determine the total number of items in the ListBox (or the number of elements in the array?)—through the ListCount Property of the ListBox. In effect, this line of code…

For i = List1.ListCount - 1 To 0 Step -1

tells Visual Basic to execute a For…Next loop starting with the last element of the array and work our way backwards to 0—the first element of the array representing the ListBox. Working backwards through the array is necessary because if we started from the 0 element of the array, Visual Basic would re-sequence the elements in the Array, and eventually we would attempt to remove an item number that no longer exists).

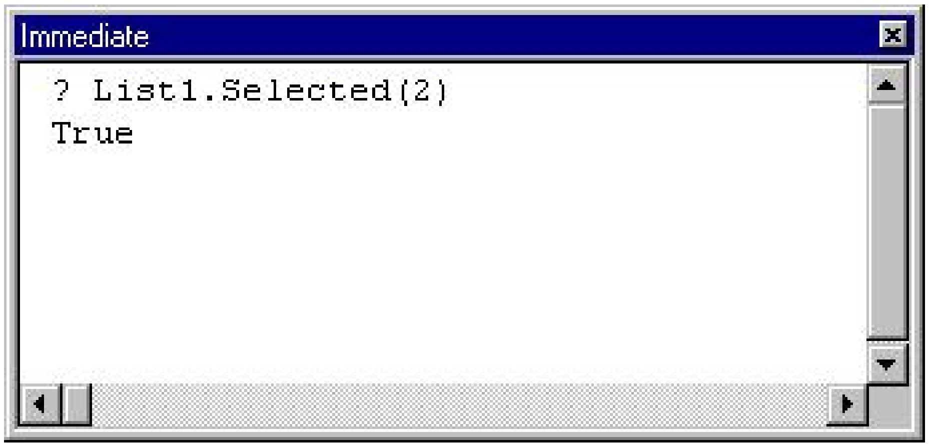
What we need now is a way to determine which element of the array is selected---and that’s where the Selected Property of the ListBox comes into play.

The Selected Property

The Selected Property (a Boolean True/False Property) can be used in conjunction with an element or Index value to determine if an item is selected or not---in other words we can type this entry into the Immediate Window

? List1.Selected(2)

to determine if element 2 (Georgia) of the ListBox is selected…



```
? List1.Selected(2)
True
```

but we can’t type this entry

? List1.Selected

or we’ll receive a compile error. In other words, the Selected Property only exists for items in the ‘behind the scenes’ array.

If this is beginning to make sense to you, then the rest of the code starts to fall into place. This line of code asks whether the item in the array (or ListBox) currently being examined within our For..Next loop is selected.

If List1.Selected(i) = True Then

If it is, we add that item to the right-hand ListBox using the AddItem method of the ListBox.

The List Property

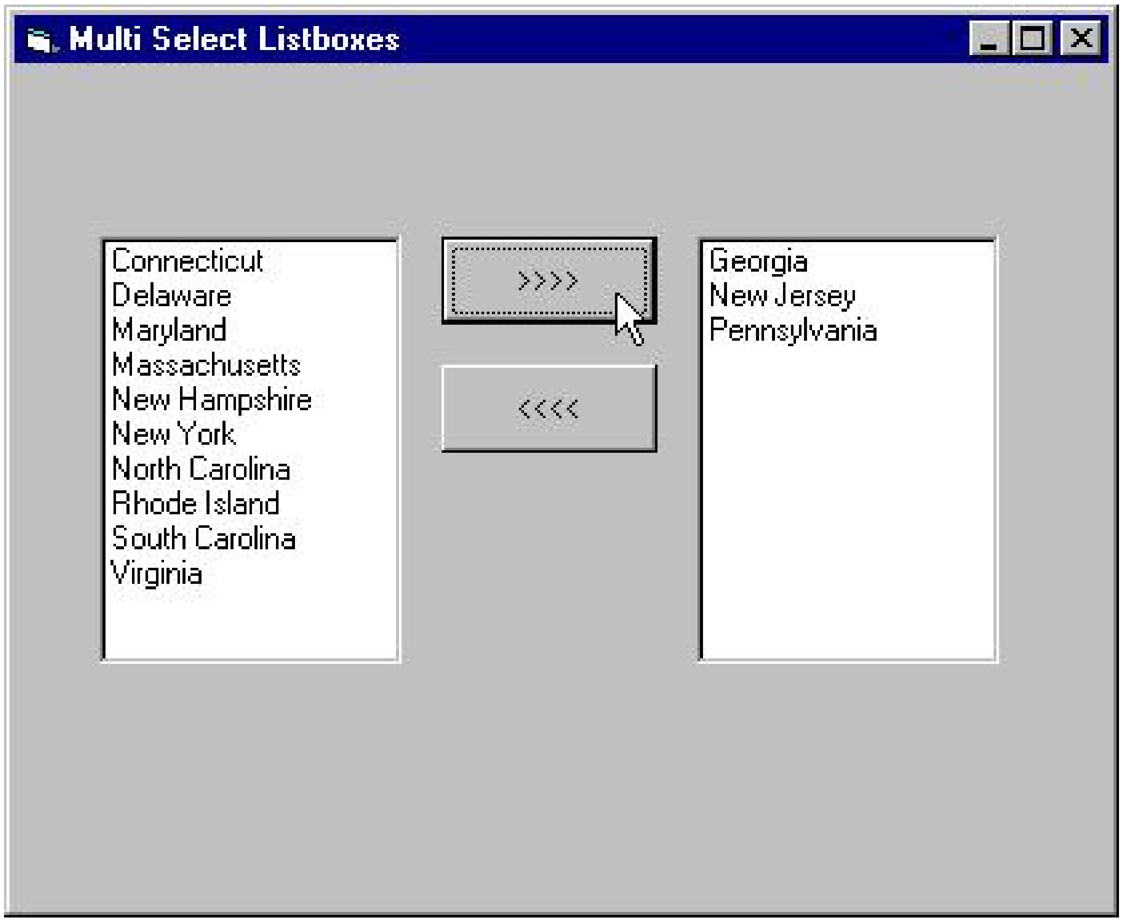
List2.AddItem List1.List(i)

Notice that this time instead of using the Text Property of the ListBox as the argument to the AddItem Method, we use another property—the List Property. Like the Selected Property, the List property exists only for items in this ‘behind the scenes’ array we’ve been talking about. It contains the text value of the item that is selected.

Finally, if the item is selected, we remove it from the first ListBox using the RemoveItem method. Since the RemoveItem method requires the index value of the item in the ListBox, the value of the variable ‘i’ can be used as the argument.

List1.RemoveItem I

Now if we run the program, and make multiple selections in the left-hand ListBox, clicking on the ‘movetoright’ command button will move each selected entry to the right-hand ListBox, and also remove the selections from the left-hand ListBox.



Here’s the code for the ‘movetoleft’ command button ….

```
Private Sub cmdMoveToLeft_Click()

Dim i As Integer

If List2.ListIndex = -1 Then Exit Sub

For i = List2.ListCount - 1 To 0 Step -1
  If List2.Selected(i) = True Then
    List1.AddItem List2.List(i)
    List2.RemoveItem i
  End If
Next i

End Sub
```

By the way, we’ve covered just about every ListBox method there is—in case you’re curious, if you wanted to remove every item from the ListBox at one time, just use the CLEAR method of the ListBox, like this…

```
List1.Clear
```

Summary

Working with MultiSelect ListBoxes is much more complex than the default Single Select ListBox. But the benefits far outweigh the complexity, as you permit the user of your program much greater flexibility.