Create Visual Basic 6 Controls At Runtime

Did you know that it's possible to create controls at runtime?

That's right. If you wish, you can add additional controls to your form at runtime. This capability gives your Visual Basic program the ultimate in flexibility---allowing you to dynamically control the appearance of your form at runtime---not only the placement of controls, but also the type and number of controls that appear on the form.

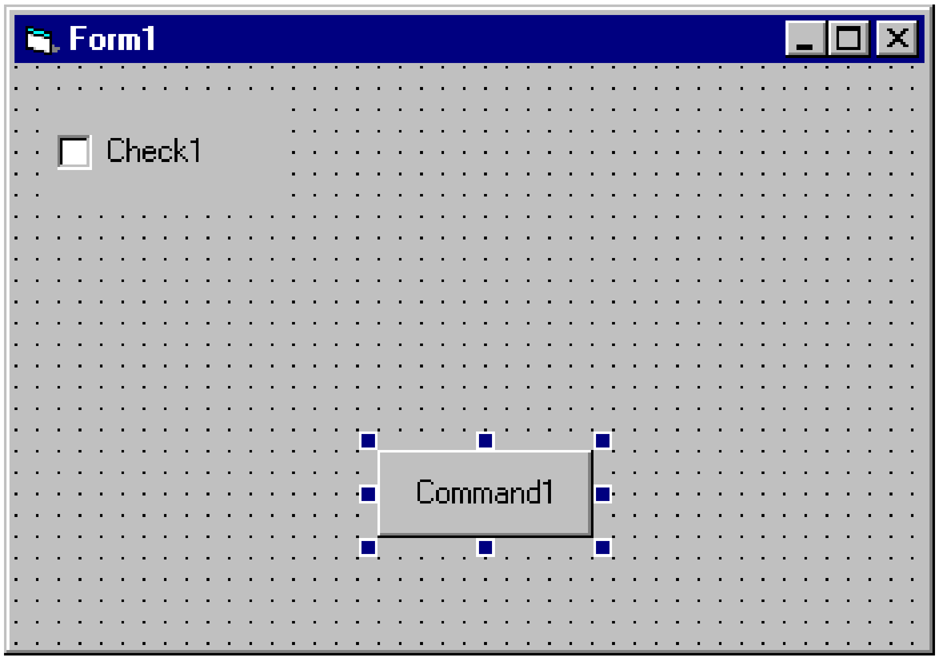
Many Visual Basic programmers are aware that it's possible to create controls at runtime by first creating a control array, and then adding additional members to the control array at runtime by using the Load Statement. This is the first method I'll be examining in this article. In addition, I'll also show you how can create controls at runtime entirely from scratch, without the need to first create a control array.

Method 1---Using a Control Array to create controls

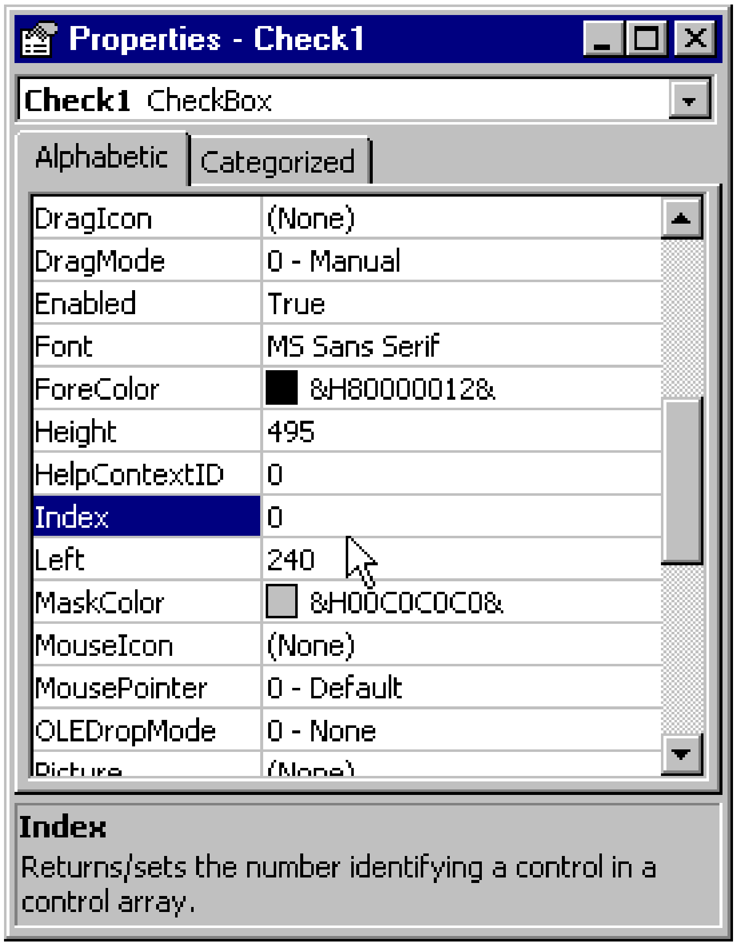
To create a control at run time using this method you first must create a control array for the control you wish to dynamically create. In other words, if you want to create checkboxes at run time, you first must create a control array of checkboxes.

For those of you unfamiliar with the term, a control array is a collection of controls on a form, all having the same name, and possessing unique Index property values. It's possible to create a Control array that has just a single 'member'---and that's what I'll do now.

I'll begin by placing a Checkbox and a Command button on a form.



In the Command Button, I'll be placing code to dynamically create a checkbox on the form at run time. More on that in a few minutes. First, I need to tell Visual Basic that the Checkbox is a member of a Control Array---I do that merely by changing its Index property from the default blank value to a number---in this case 0.



Check1 CheckBox

DragIcon	(None)
DragMode	0 - Manual
Enabled	True
Font	MS Sans Serif
ForeColor	■ &H80000012&
Height	495
HelpContextID	0
Index	0
Left	240
MaskColor	□ &H00C0C0C0&
MouseIcon	(None)
MousePointer	0 - Default
OLEDropMode	0 - None
Picture	(None)

Index
Returns/sets the number identifying a control in a control array.

Once the Index property has been set to 0, the Checkbox is now a member of the Check1 Checkbox Control Array---which makes creating a new checkbox at runtime very easy.

All we need to do is tell Visual Basic that we want to create a new checkbox, using the existing Checkbox as a template. We do this by executing the Visual Basic Load Statement within the Click Event Procedure of the Command Button---like this.

```
Private Sub Command1_Click()

Load Check1(1)

Check1(1).Caption = "New Checkbox"

End Sub
```

The Load statement

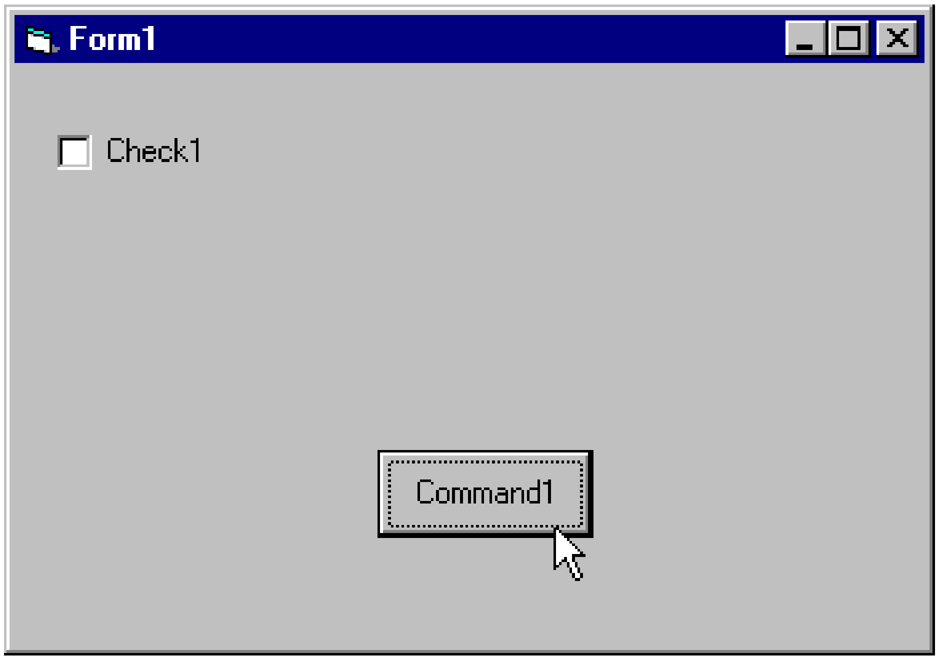
```
Load Check1(1)
```

tells Visual Basic to create a new member of the Check1 checkbox array---and to create it with an Index property of 1. This statement

```
Check1(1).Caption = "New Checkbox"
```

gives the Checkbox, whose index property is equal to 1, a unique caption to make it 'stand out' form the original checkbox placed on the form at design time.

If we now run the program, and then click on the Command Button we'll see this screenshot

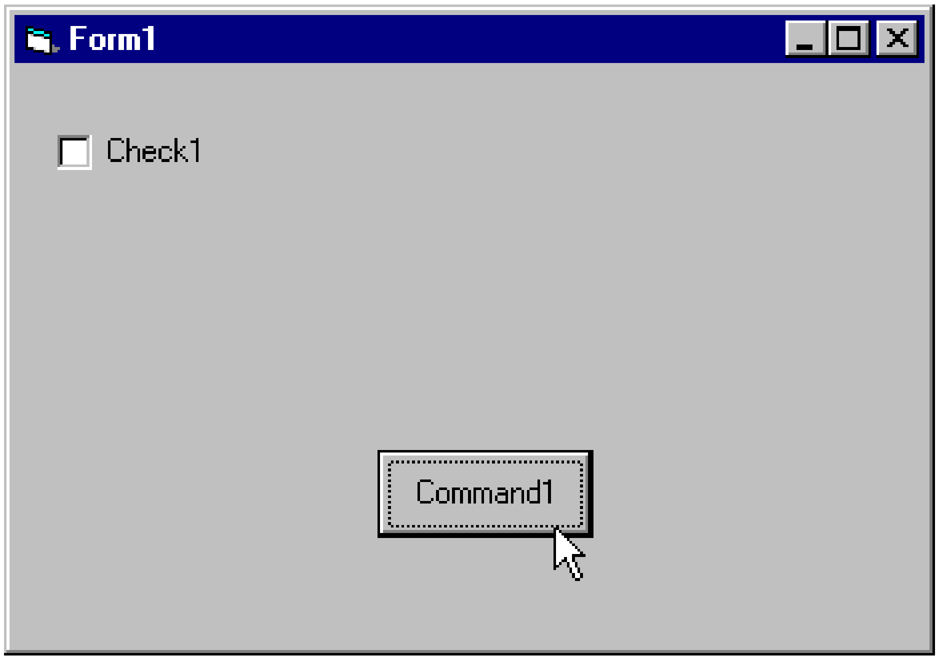


Oops…something's wrong.

What I didn't tell you is that when you create a control at runtime, by default, the Visible property of the new control is set to False. To see the new control, we need to explicitly set its Visible property to True. Let's modify the code in the Click event procedure to look like this…

```
Private Sub Command1_Click()

Load Check1(1)

Check1(1).Caption = "New Checkbox"

Check1(1).Visible = True

End Sub
```

Now let's run the program again, and click on the Command Button once more…



Something is still wrong---there's still just the single checkbox!

The problem is this: when the new control is created, the properties of the new control are identical to the properties of the 'template' control used to create it---with the exception of the Index property which we set with the Load statement, and the Visible property which we know is initialized to False. Because of that, the new checkbox is on the form---it just so happens to be sitting 'under' the first control, since it has identical Top, Left, Height and Width properties.

All we need to do to see the new control is to move it away from the first control---and we can do that by adding a line of code to the Click Event procedure to change the Top property of the new control. Like this…

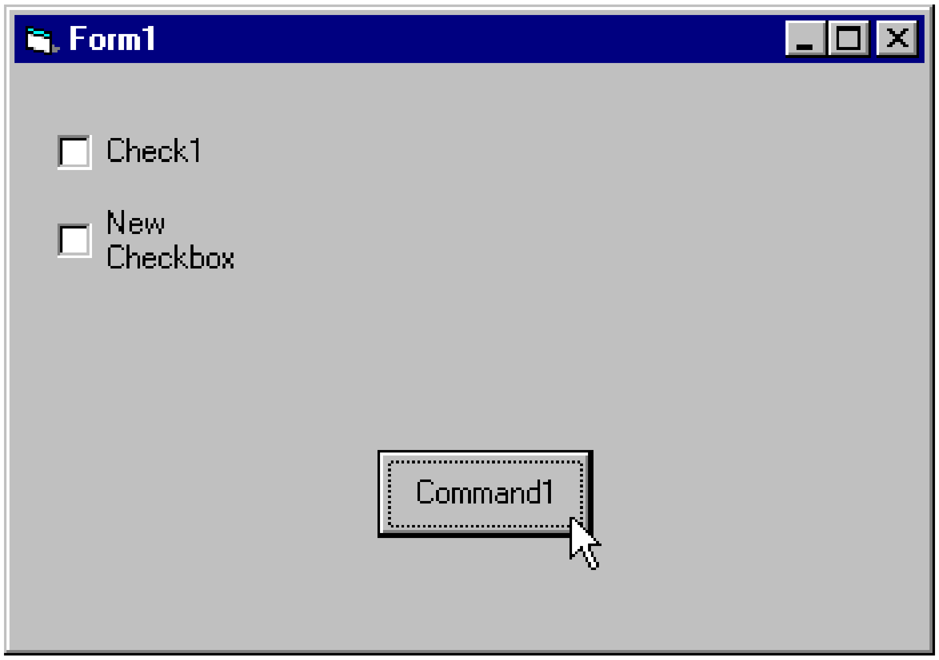
```
Private Sub Command1_Click()

Load Check1(1)

Check1(1).Caption = "New Checkbox"

Check1(1).Visible = True

Check1(1).Top = Check1(0).Top + Check1(0).Height

End Sub
```

This line of code

Check1(1).Top = Check1(0).Top + Check1(0).Height

tells VB to take the current value of the Top property of the existing checkbox, and to add to that the value of its Height Property (remember, the first control has an Index property of 0). The result of this addition is a Top property for the new control that is just under the first control.

If we now run the program, and click on the Command Button, we'll see this screen shot…



Works like a charm!

I should also point out that if we want to give the user the impression that we are creating controls totally from scratch, we can place the template control on the form---and then set its Visible property to False---that way, at run time, the controls are created---seemingly out of nowhere.

To review, here's a summary of the steps necessary to create a new control using the Control Array method.

1. Create a control array of the control type you wish to create at runtime. If you want to create a textbox at runtime, create a Textbox control array. If you want to create a Command Button, create a Command Button Control array. Remember, to create a Control Array, all you need to do is to change the Index property

of the control to something other than its default empty value.

2. Use the Load Statement, with a unique Index property, to create the new control.

3. Change the Visible property of the new control to True in order to make it visible.

4. Change the coordinate properties (Left or Top) to bring the new control out from under the original.

Method 2---Creating controls from scratch using the Controls Collection

There's a second method to create a control at runtime, and that's to use the Add Method of the Controls collection. This method is easier to use than the Control Array method, but harder to understand since it requires some familiarity with the Visual Basic Controls Collection. (I discuss the Controls Collection in my latest book, Learn To Program Objects with Visual Basic 6.)

In short, each control that is placed on the form either at Design time or runtime is made a member of the intrinsic Visual Basic Collection called the Controls Collection. For those of you not familiar with Visual Basic Collections, a Collection is similar to a one dimensional array. Each control on the form has a reference placed on the Controls collection when it is placed there at design time. In the same way, a control that is placed on the form at run time (the way we just did using the Control Array Method) also has a reference placed on the Controls Collection.

It's also possible to create a control at runtime by adding a reference to a control directly to the Controls Collection. Doing so avoids the necessity of first having to create a 'template' control on the form at design time---the reason for that is that VB maintains templates for all of the controls in the hidden Visual Basic Global Object called VB (again, more on this in my Objects Book).

Suffice to say that all that is required to create a control at runtime using this method is to execute four lines of code, like this…

```
Private Sub Command1_Click()

Dim ctlName As Control
```

```
Set ctlName = Form1.Controls.Add("VB.TextBox", "Text1", Form1)

ctlName.Visible = True

ctlName.Top = Check1(0).Top + Check1(0).Height

End Sub
```

This line of code

```
Dim ctlName As Control
```

declares something known as an Object Variable---which is nothing more than a variable that contains, as a value, a reference to an Object (in this case, a Textbox). You can declare an Object variable as a specific control type (such as Textbox) or elect to declare it as the more generic Control type, which is what we did here.

This line of code

```
Set ctlName = Form1.Controls.Add("VB.TextBox", "Text1", Form1)
```

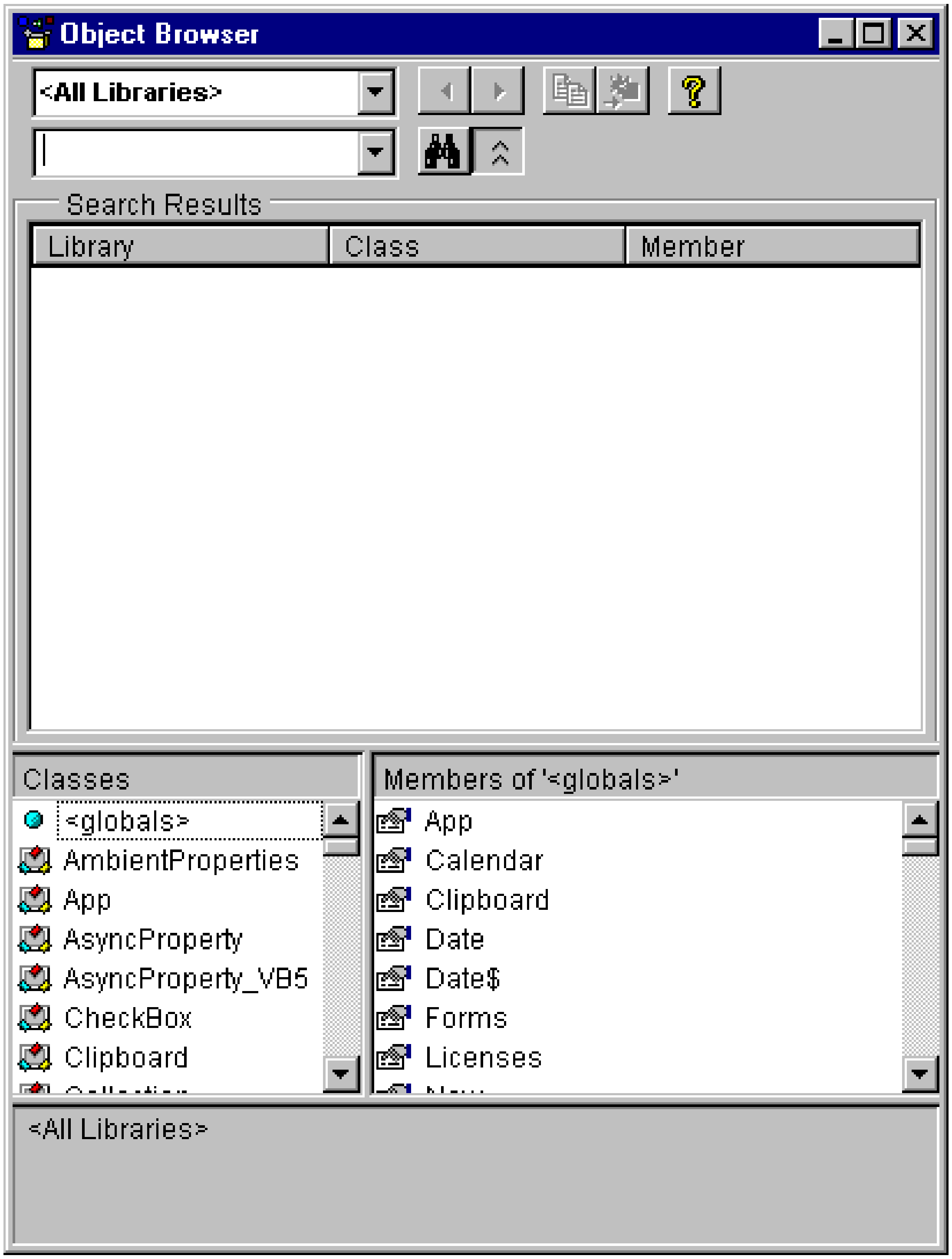
tells Visual Basic to add a Textbox control called Text1 to the Controls Collection of Form1, and to use the Object Variable ctlname to 'point to it'. Once this control has been added to the Controls Collection, thereafter whenever we refer to the control using code we must refer to it by the Object Variable Name. That's why, when we then set the Visible property of the new textbox to True, and adjust its Top property, we use the Object Variable name instead.

```
ctlName.Visible = True

ctlName.Top = Check1(0).Top + Check1(0).Height
```

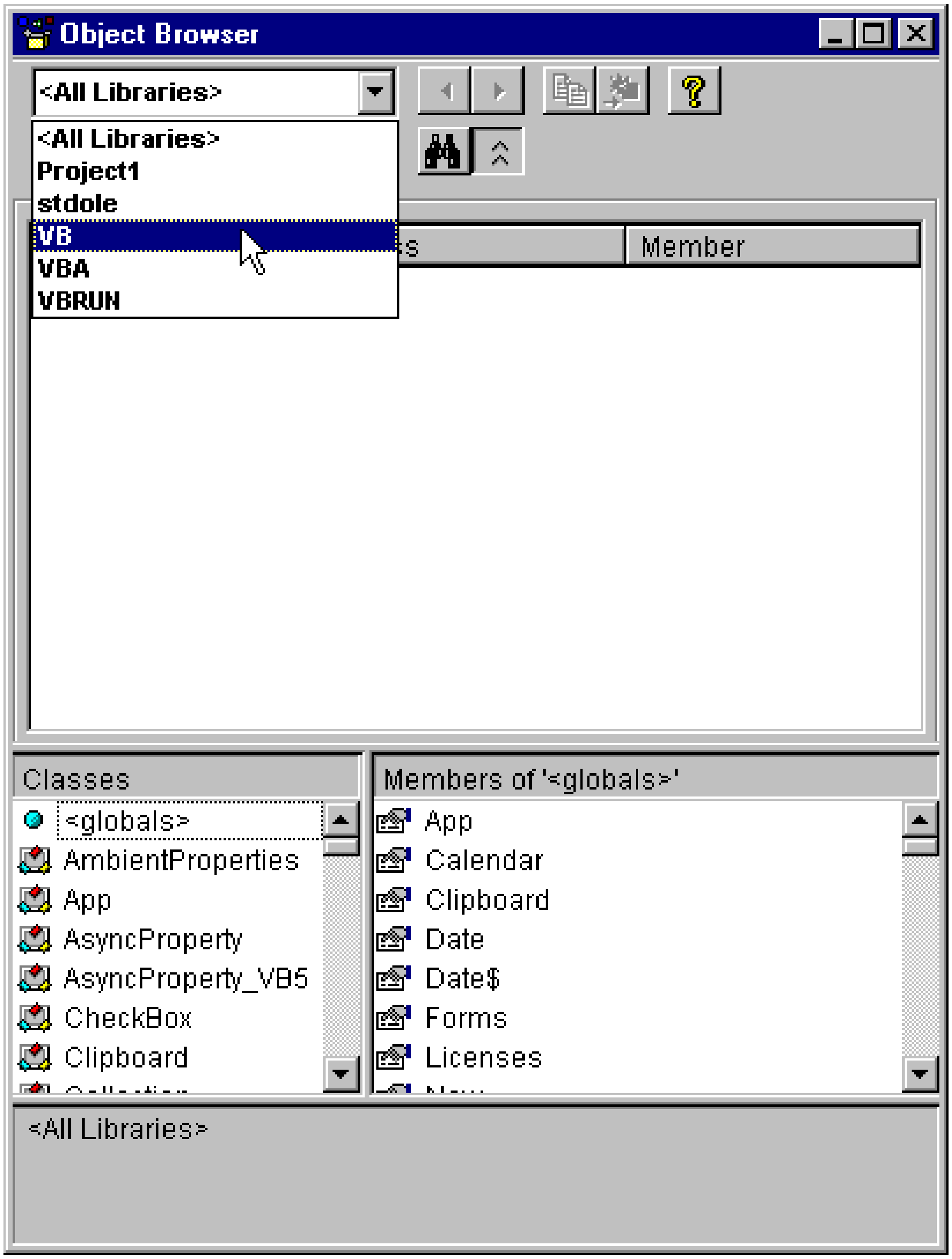
More on the Add Method.

The Add Method has three arguments---the first is the name of the template for the control you are creating, the second is the name of the control as it will appear in the Controls Collection, and the third argument is the control's container (ordinarily the form, but it could be the name of a Frame Control if you wanted the control to be placed 'within' a Frame on the form).

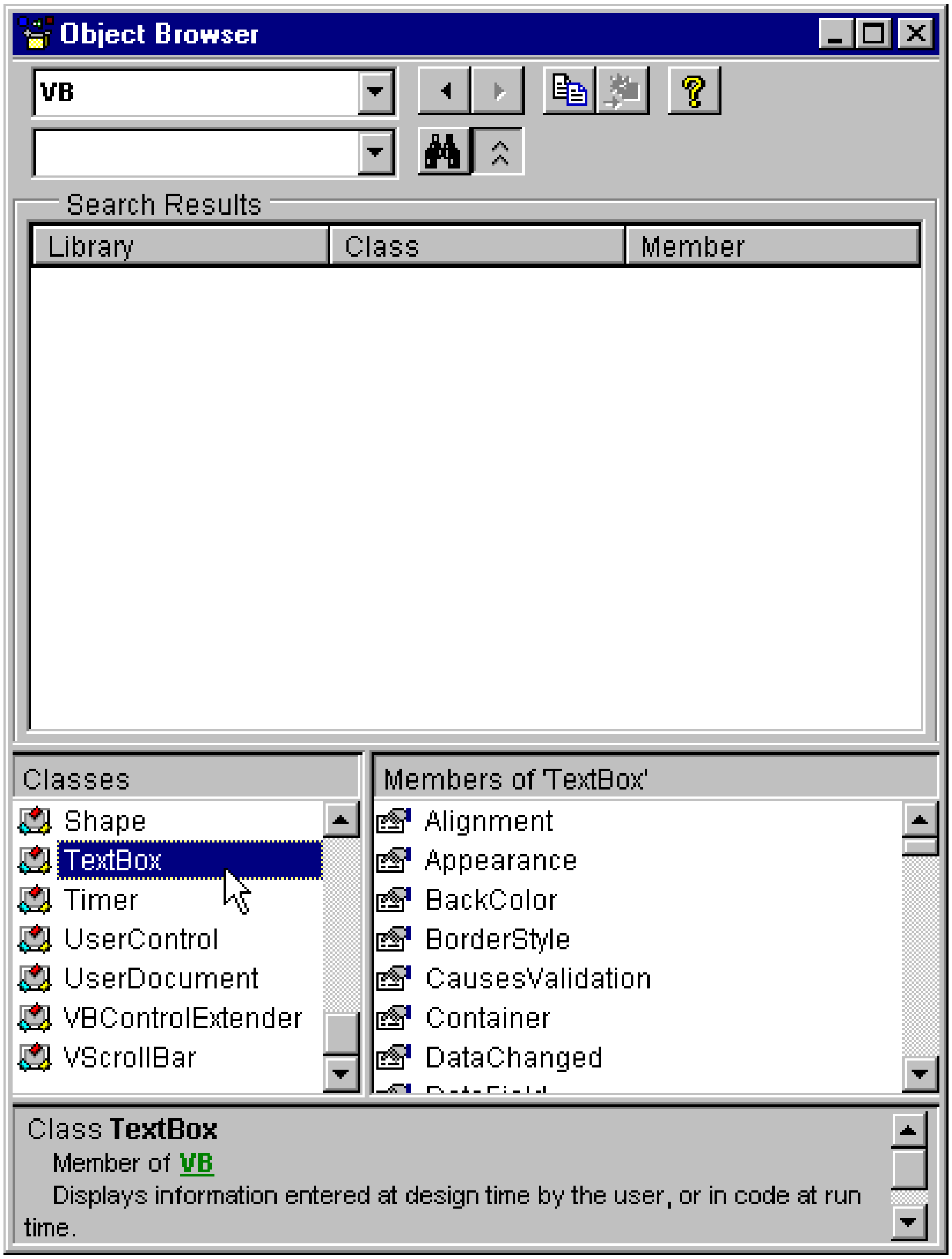
The Textbox control is called Textbox, the Command Button control is called CommandButton. If you open the Visual Basic Object Browser (View-Object Browser from the Visual Basic Menu Bar)



and select the VB Global Object in the library listbox…



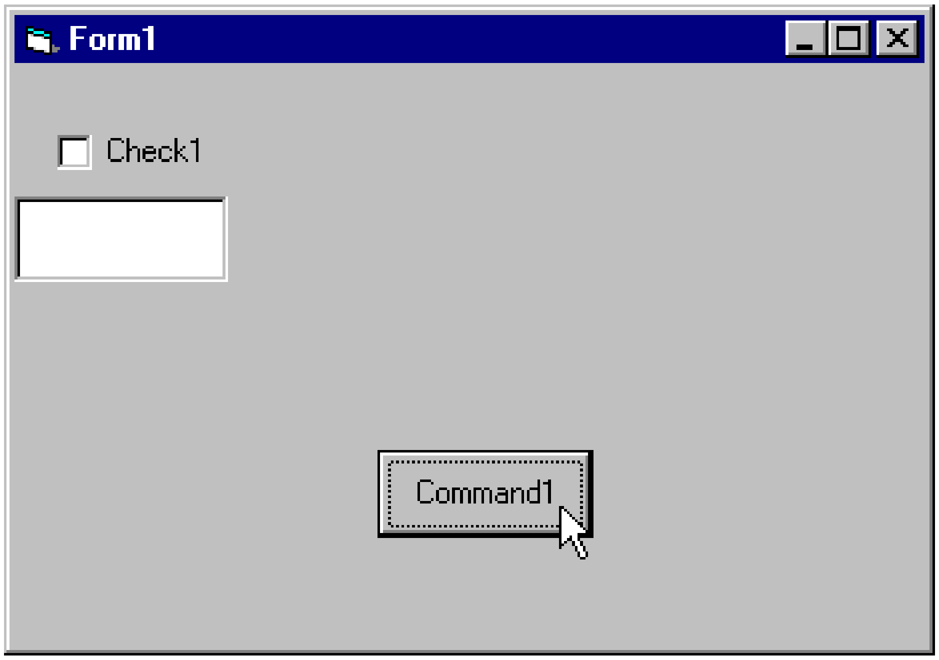
the resulting display will show you the names for the controls that you can create at runtime…



Class **TextBox**
Member of **VB**
Displays information entered at design time by the user, or in code at run time.

Again, for more information on the Visual Basic Object Browser, check out my Objects book.

If we now run the program, and click on the command button, we'll see this screenshot…



As you can see, the new Textbox control has been placed on the form.

Summary

The ability to add controls to a form at run time can produce incredibly dynamic forms.