What's the simplest .NET equivalent of a VB6 control array?

Asked 16 years, 3 months ago Modified 3 years, 10 months ago Viewed 12k times

10

Maybe I just don't know .NET well enough yet, but I have yet to see a satisfactory way to implement this simple VB6 code easily in .NET (assume this code is on a form with N CommandButtons in array Command1() and N TextBoxes in array Text1()):

```
Private Sub Command1_Click(Index As Integer)

   Text1(Index).Text = Timer

End Sub
```

I know it's not very useful code, but it demonstrates the ease with which control arrays can be used in VB6. What is the simplest equivalent in C# or VB.NET?

.net vb6 vb6-migration control-array

Share
Improve this question
Follow

edited Mar 26, 2009 at 12:35
MarkJ
30.4k ● 5 ● 71 ● 113

asked Sep 2, 2008 at 13:44
raven
18.1k ● 17 ● 82 ● 114

See also stackoverflow.com/questions/5497403/control-array-in-vb-net and stackoverflow.com/questions/5738092/vb6-control-arrays-in-net – MarkJ Apr 21, 2011 at 11:42

Maybe you could check in the future my idea: stackoverflow.com/questions/5738092/vb6-control-arrays-in-net/…, I think it provides a solution to this tricky question.
– Federico Navarrete Mar 7, 2017 at 10:09

9 Answers

Sorted by: Highest score (default)

4

Make a generic list of textboxes:

```
var textBoxes = new List<TextBox>();

// Create 10 textboxes in the collection
for (int i = 0; i < 10; i++)
{
    var textBox = new TextBox();
    textBox.Text = "Textbox " + i;
    textBoxes.Add(textBox);
}

// Loop through and set new values on textboxes in collection
for (int i = 0; i < textBoxes.Count; i++)
{
    textBoxes[i].Text = "New value " + i;
    // or like this
    var textBox = textBoxes[i];
    textBox.Text = "New val " + i;
}
```

Share
Improve this answer
Follow

edited Sep 15, 2016 at 10:37

answered Sep 2, 2008 at 13:51
Seb Nilsson
26.4k ● 30 ● 106 ● 131

4

Another nice thing that VB .NET does is having a single event handler that handles multiple controls:

```
Private Sub TextBox_TextChanged(ByVal sender As System.Object, ByVal e As
System.EventArgs) _
        Handles TextBox1.TextChanged, _

        TextBox2.TextChanged, _

        TextBox3.TextChanged

End Sub
```

Share Improve this answer Follow

answered Sep 2, 2008 at 13:49
Mark Biek
151k ● 54 ● 158 ● 201

3

There is no *real* 1 to 1 analog in .Net. Sure, you can make arrays or lists of controls of a specific type, but there's nothing that will do that for you automatically.

However, I've never seen a control array that couldn't be refactored in .Net to something *better*. A case in point is your example. In the scenario you posted, you're using control arrays to pair a button up with a textbox. In .Net, you would probably do this with a custom control. The custom control would consist of a button, a textbox, and maybe a shared/static timer. The form uses several instances of this custom control. You implement the logic needed for the control once, and it's isolated to it's own source file which can be tracked and edited in source control without requiring a merge with the larger form class, or easily re-used on multiple forms or even in multiple projects. You also don't have to worry about making sure the command button index matches up with the textbox index.

Using a custom control for this instead of a control array is loosely analogous to using class to group data instead of an array, in that you get names instead of indexes.

Share
Improve this answer
Follow

edited Nov 7, 2008 at 14:33

answered Sep 2, 2008 at 14:11
Joel Coehoorn
415k ● 114 ● 577 ● 813

2

There are two aspects.

.NET readily supports arrays of controls, VB6 just had to use a workaround because otherwise, wiring up events was really hard. In .NET, wiring up events dynamically is easy.

However, the .NET form designer does not support control arrays for a simple reason: arrays of controls are created/extended at run time. If you know how many controls you need at compile time (the reasoning goes) then you give them different names and don't put them in an array.

> I know it's not very useful code

That's exactly the point. Why have a feature if it's useless?

If needed, you can also access a control by name, resulting in something like this:

```
Private Sub Command_Click(sender As Object, e As EventArgs) Handles
Command1.Click, Command2.Click …
    Dim name As String = DirectCast(sender, Control).Name
    Dim index As Integer = Integer.Parse(name.Substring("Command".Length))
    Controls(String.Format("Text {0}", index)).Text = Timer.Value.ToString()
End Sub
```

Share Improve this answer Follow

answered Sep 2, 2008 at 13:50
Konrad Rudolph
545k ● 139 ● 956 ● 1.2k

I was referring to my example when I said, "it's not very useful code". I have found control arrays to be extremely useful in VB6, which is why I posed the question. I found your proposed solution pretty clever, but it just seems like a bit of a kludge. – raven Sep 17, 2008 at 14:00

VisualBasic .NET's compatibility library contains strong typed control arrays. This is what the upgrade wizard uses to replace the current VB6 control arrays.

1

However, A control array in VB6 is just a collection of objects with VB6 doing some syntax magic on the surface. In the .NET world, by removing this, they are forcing better practices.

In closing, with the advent of generics, there is nothing stopping you from using

```
List<YourControl> MyControlArray.
```

Share
Improve this answer
Follow

edited Sep 2, 2008 at 13:58

answered Sep 2, 2008 at 13:50
FlySwat
175k ● 75 ● 248 ● 314

1

The two main benefits of control arrays in VB6 were: (1) They provided a way for you to iterate through a collection of controls (2) They allowed you to share events between controls

(1) can be accomplished in .Net using an array of controls (2) can be accomplished by having one event handle multiple controls (the syntax is a little different because you use the `sender` argument instead of `myArray(index)`).

One nice thing about .Net is that these features are decoupled. So for instance you can have controls that share events even if they aren't part of an array and have different names and even a different type. And you can iterate through a collection of controls even if they have totally different events.

Share Improve this answer Follow

answered Jan 20, 2010 at 5:20
Tim Goodman
24k ● 7 ● 66 ● 85

0

Make an array of controls.

```
TextBox[] textboxes = new TextBox[] {
    textBox1,
    textBox2,
    textBox3
};
```

Share Improve this answer Follow

answered Sep 2, 2008 at 13:45
Lasse V. Karlsen
391k ● 106 ● 646 ● 844

0

The same click event can handle the button presses from multiple buttons in .Net. You could then add the the text box to find in the Tag property?

```
Private Sub AllButton_Click(sender As Object, e As EventArgs) Handles
Button1.Click, Button2.Click, Button3.Click
  Dim c As Control = CType(sender, Control)
  Dim t As TextBox = FindControl(CType(c.Tag, String))
  If t Is Not Nothing Then
     t.Text = "Clicked"
  End If
End Sub
```

Share Improve this answer Follow

answered Sep 2, 2008 at 13:54

samjudson
56.8k ● 7 ● 60 ● 69

0

I know that my answer is quite late, but I think I found the solution. I'm not the only former VB6 developer who has struggled with this limitation of VS. Long ago, I tried to migrate a CRM that I designed, but I failed because I had a tough dependency on control arrays (hundreds in one form). I read many forums and I was able to write this simple code:

VB.NET:

```
'To declare the List of controls
Private textBoxes As List(Of TextBox) = New List(Of TextBox)()

Private Sub Form1_Load(ByVal sender As Object, ByVal e As EventArgs)
    'To get all controls in the form
    For Each control In Controls

        'To search for the specific type that you want to create the array
        If control.[GetType]() = GetType(TextBox) Then
            textBoxes.Add(CType(control, TextBox))
        End If
    Next

    'To sort the labels by the ID
    textBoxes = textBoxes.OrderBy(Function(x) x.Name).ToList()
End Sub
```

C#:

```
//To declare the List of controls
private List<TextBox> textBoxes = new List<TextBox>();
private void Form1_Load(object sender, EventArgs e)
{
    //To get all controls in the form
    foreach (var control in Controls)
```

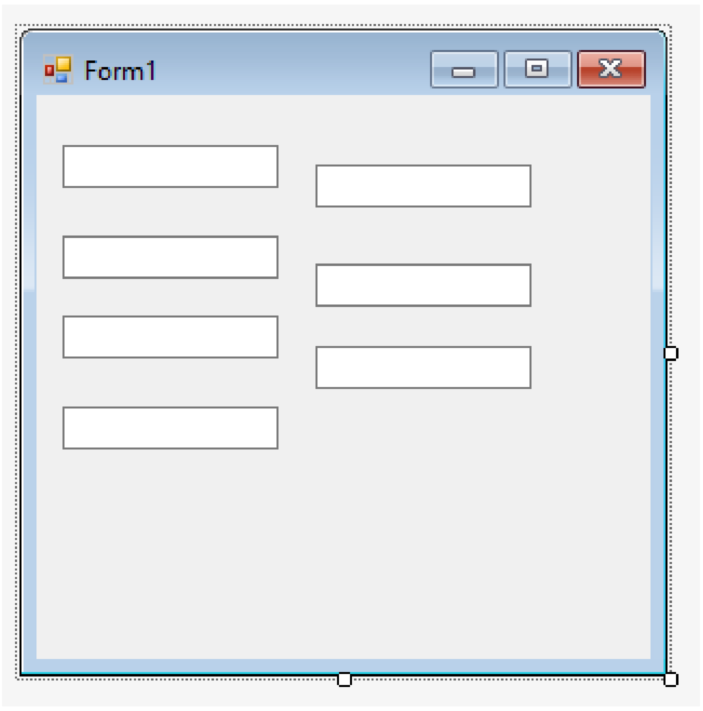
```
        {
            //To search for the specific type that you want to create the array
            if (control.GetType() == typeof(TextBox))
            {
                //To add the control to the List
                textBoxes.Add((TextBox)control);
            }
        }

        //To sort the labels by the ID
        textBoxes = textBoxes.OrderBy(x => x.Name).ToList();
    }
```

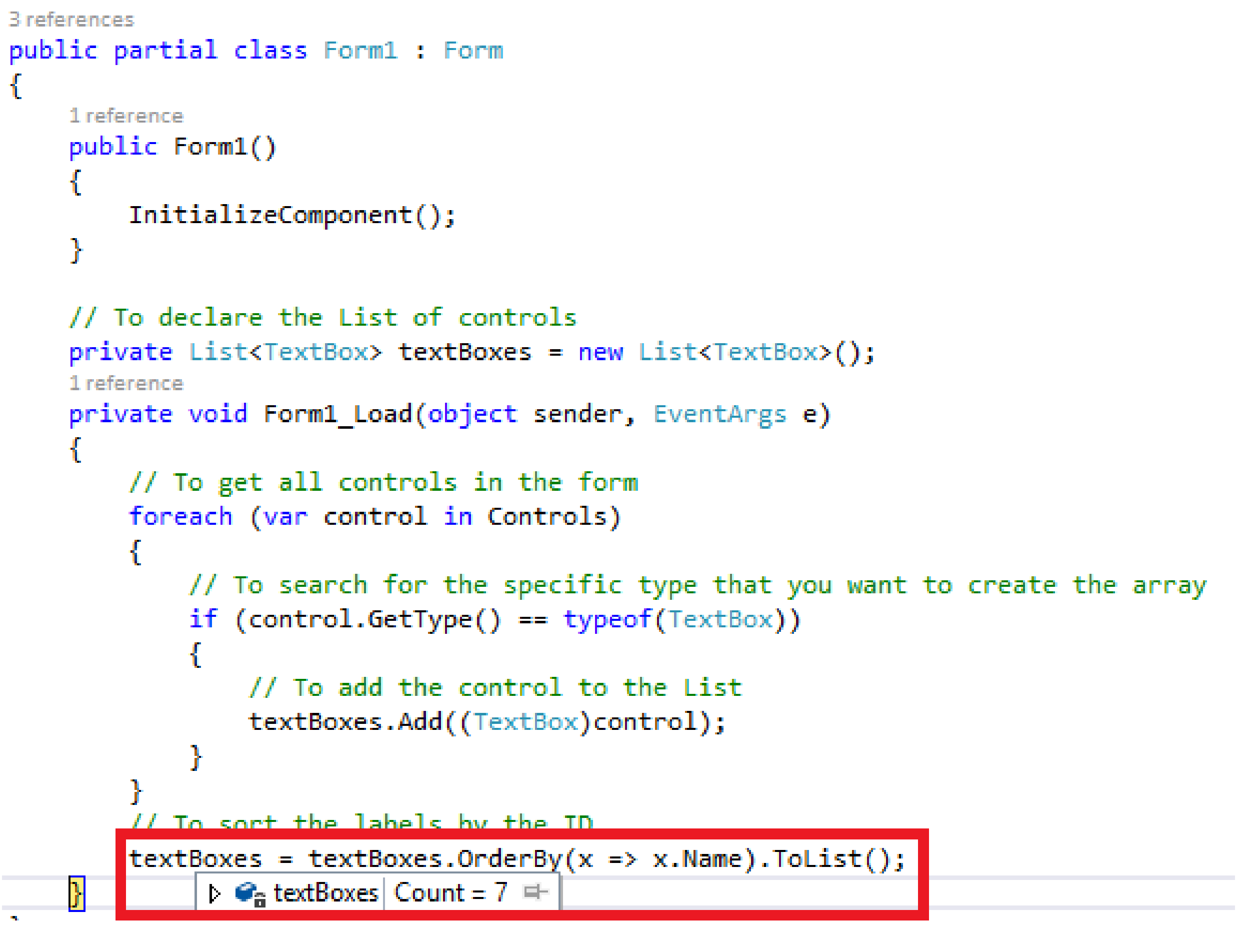
There are 3 points to take into consideration:

1. A `List` will help you to emulate the large collection of controls.
2. `typeof(Control)` will help you define the type of the control to add to the list.
3. While you keep the **"index"** as the last character *(textBox**1**, textBox**2**, ..., textBox**N**)* you can create a logical order.

Example in **design mode**:



Running:



```
3 references
public partial class Form1 : Form
{
    1 reference
    public Form1()
    {
        InitializeComponent();
    }

    // To declare the List of controls
    private List<TextBox> textBoxes = new List<TextBox>();
    1 reference
    private void Form1_Load(object sender, EventArgs e)
    {
        // To get all controls in the form
        foreach (var control in Controls)
        {
            // To search for the specific type that you want to create the array
            if (control.GetType() == typeof(TextBox))
            {
                // To add the control to the List
                textBoxes.Add((TextBox)control);
            }
        }
        // To sort the labels by the ID
        textBoxes = textBoxes.OrderBy(x => x.Name).ToList();
    }
```

A **similar logic** could be potentially used in other technologies like WPF, ASP.NET (Web Forms), or Xamarin (Forms) -in my opinion-. I hope this piece of code is going to help more programmers in the future.

Share

Improve this answer

Follow

edited Feb 13, 2021 at 20:55

answered Aug 6, 2019 at 9:18

Federico Navarrete

3,264 ● 5 ● 46 ● 86