Why can't you bind the Size of a windows form to ApplicationSettings?

Asked 16 years, 4 months ago Modified 14 years ago Viewed 5k times

4

Update: Solved, with code

I got it working, see my answer below for the code...

Original Post

As Tundey pointed out in his answer to my last question, you can bind nearly everything about a windows forms control to ApplicationSettings pretty effortlessly. So is there really no way to do this with form Size? This tutorial says you need to handle Size explicitly so you can save RestoreBounds instead of size if the window is maximized or minimized. However, I hoped I could just use a property like:

```
public Size RestoreSize
{
    get
    {
        if (this.WindowState == FormWindowState.Normal)
        {
            return this.Size;
        }
        else
        {
            return this.RestoreBounds.Size;
        }
    }
    set
    {
        ...
    }
}
```

But I can't see a way to bind this in the designer (Size is notably missing from the PropertyBinding list).

c# visual-studio data-binding .net-2.0

Share
Improve this question
Follow

edited Jun 20, 2020 at 9:12
Community Bot
1 ●1

asked Aug 20, 2008 at 18:59
Brian Jorgensen
1,228 ●14 ●17

Sorry, coming very late to your question. But I do know the answer :-) – HTTP 410 Nov 4, 2008 at 10:44

5 Answers

Sorted by: Highest score (default)

12

I finally came up with a Form subclass that solves this, once and for all. To use it:

1. Inherit from RestorableForm instead of Form.
2. Add a binding in (ApplicationSettings) -> (PropertyBinding) to WindowRestoreState.
3. Call Properties.Settings.Default.Save() when the window is about to close.

Now window position and state will be remembered between sessions. Following the suggestions from other posters below, I included a function ConstrainToScreen that makes sure the window fits nicely on the available displays when restoring itself.

Code

```
// Consider this code public domain. If you want, you can even tell
// your boss, attractive women, or the other guy in your cube that
// you wrote it. Enjoy!

using System;
using System.Windows.Forms;
using System.ComponentModel;
using System.Drawing;

namespace Utilities
{
    public class RestorableForm : Form, INotifyPropertyChanged
    {
        // We invoke this event when the binding needs to be updated.
        public event PropertyChangedEventHandler PropertyChanged;

        // This stores the last window position and state
        private WindowRestoreStateInfo windowRestoreState;

        // Now we define the property that we will bind to our settings.
        [Browsable(false)]        // Don't show it in the Properties list
        [SettingsBindable(true)]  // But do enable binding to settings
        public WindowRestoreStateInfo WindowRestoreState
        {
            get { return windowRestoreState; }
            set
            {
                windowRestoreState = value;
                if (PropertyChanged != null)
                {
                    // If anybody's listening, let them know the
                    // binding needs to be updated:
```

```
                    PropertyChanged(this,
                        new PropertyChangedEventArgs("WindowRestoreState"));
                }
            }
        }

        protected override void OnClosing(CancelEventArgs e)
        {
            WindowRestoreState = new WindowRestoreStateInfo();
            WindowRestoreState.Bounds
                = WindowState == FormWindowState.Normal ?
                  Bounds : RestoreBounds;
            WindowRestoreState.WindowState = WindowState;

            base.OnClosing(e);
        }

        protected override void OnLoad(EventArgs e)
        {
            base.OnLoad(e);

            if (WindowRestoreState != null)
            {
                Bounds = ConstrainToScreen(WindowRestoreState.Bounds);
                WindowState = WindowRestoreState.WindowState;
            }
        }

        // This helper class stores both position and state.
        // That way, we only have to set one binding.
        public class WindowRestoreStateInfo
        {
            Rectangle bounds;
            public Rectangle Bounds
            {
                get { return bounds; }
                set { bounds = value; }
            }

            FormWindowState windowState;
            public FormWindowState WindowState
            {
                get { return windowState; }
                set { windowState = value; }
            }
        }

        private Rectangle ConstrainToScreen(Rectangle bounds)
        {
            Screen screen = Screen.FromRectangle(WindowRestoreState.Bounds);
            Rectangle workingArea = screen.WorkingArea;

            int width = Math.Min(bounds.Width, workingArea.Width);
            int height = Math.Min(bounds.Height, workingArea.Height);

            // mmm....minimax
            int left = Math.Min(workingArea.Right - width,
                                Math.Max(bounds.Left, workingArea.Left));
            int top = Math.Min(workingArea.Bottom - height,
                                Math.Max(bounds.Top, workingArea.Top));

            return new Rectangle(left, top, width, height);
        }
    }
}
```

Settings Bindings References

- SettingsBindableAttribute
- INotifyPropertyChanged

Share

Improve this answer

Follow

edited Aug 20, 2008 at 23:33

answered Aug 20, 2008 at 23:06

Brian Jorgensen

1,228 ● 14 ● 17

7

The reason why the Form.Size property is not available in the settings binding UI is because this property is marked **DesignerSerializationVisibility.Hidden**. This means that the designer doesn't know how to serialise it, let alone generate a data binding for it. Instead the **Form.ClientSize** property is the one that gets serialised.

If you try and get clever by binding **Location** and **ClientSize**, you'll see another problem. When you try to resize your form from the left or top edge, you'll see weird behaviour. This is apparently related to the way that two-way data binding works in the context of property sets that mutually affect each other. Both **Location** and **ClientSize** eventually call into a common method, **SetBoundsCore()**.

Also, data binding to properties like **Location** and **Size** is just not efficient. Each time the user moves or resizes the form, Windows sends hundreds of messages to the form, causing the data binding logic to do a lot of processing, when all you really want is to store the last position and size before the form is closed.

This is a very simplified version of what I do:

```
private void MyForm_FormClosing(object sender, FormClosingEventArgs e)
{
    Properties.Settings.Default.MyState = this.WindowState;
    if (this.WindowState == FormWindowState.Normal)
    {
       Properties.Settings.Default.MySize = this.Size;
       Properties.Settings.Default.MyLoc = this.Location;
    }
    else
    {
       Properties.Settings.Default.MySize = this.RestoreBounds.Size;
       Properties.Settings.Default.MyLoc = this.RestoreBounds.Location;
    }
    Properties.Settings.Default.Save();
}

private void MyForm_Load(object sender, EventArgs e)
```

```
{
    this.Size = Properties.Settings.Default.MySize;
    this.Location = Properties.Settings.Default.MyLoc;
    this.WindowState = Properties.Settings.Default.MyState;
}
```

Why is this a very simplified version? Because doing this properly is a lot trickier than it looks :-)

Share Improve this answer Follow

answered Nov 4, 2008 at 2:59
HTTP 410
17.6k ● 14 ● 79 ● 129

2

One of the reason I imagine size binding is not allowed is because the screen may change between sessions.

Loading the size back when the resolution has reduced could result in the title bar being beyond the limits of the screen.

You also need to be wary of multiple monitor setups, where monitors may no longer be available when you app next runs.

Share Improve this answer Follow

answered Aug 20, 2008 at 19:52
Martin
40.3k ● 20 ● 100 ● 131

1

Well I have had a quick play with this and you are correct, while there is no way to directly *bind* the size of the form to AppSettings, you can add your own values and change the size on load.

I would perhaps recommend that if this is a common feature, you subclass Form and make it automatically prob the App.Config for the forms size settings.

(Or you could roll your own file.. Get it to query an Xml file "formname.settings.xml" or something? - thinking out loud!)..

Heres what I had (very rough, no error checking etc).

App.Config

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <appSettings>
        <add key ="FormHeight" value="500" />
        <add key ="FormWidth" value="200"/>
    </appSettings>
</configuration>
```

Form Code

```
    private void Form1_Load(object sender, EventArgs e)
    {
        string height = ConfigurationManager.AppSettings["FormHeight"];
        int h = int.Parse(height);
        string width = ConfigurationManager.AppSettings["FormWidth"];
        int w = int.Parse(width);
        this.Size = new Size(h, w);
    }
```

Share Improve this answer Follow

answered Aug 20, 2008 at 19:41
Rob Cooper
28.9k ● 26 ● 105 ● 142

1

I agree with Rob Cooper's answer. But I think Martin makes a very good point. Nothing like having users open your application and the app is off-screen!

So in reality, you'll want to combine both answers and bear in mind the current screen dimensions before setting your form's size.

Share Improve this answer Follow

answered Aug 20, 2008 at 20:04
Tundey
2,965 ● 1 ● 24 ● 28