How to configure an app to run correctly on a machine with a high DPI setting (e.g. 150%)?

Asked 12 years, 1 month ago Modified 1 year, 8 months ago

Viewed 67k times

119

I've created a simple Winforms application in C#. When I run the application on a machine with high DPI settings (e.g. 150%), the application gets scaled up. So far so good! But instead of rendering the fonts with a higher font size, all texts are just scaled up, too. That of course leads to very blurry text (on all controls like buttons etc.).

Shouldn't windows take care of rendering the texts correctly? For example my application's title bar is rendered crisp & clear.

c# winforms fonts dpi dpi-aware

Share

Improve this question

Follow

edited Aug 13, 2018 at 8:52

Wollmich

1,646 ● 1 ● 19 ● 51

asked Nov 5, 2012 at 8:08

Boris

8,911 ● 26 ● 71 ● 128

7 Answers

Sorted by: Highest score (default)

152

Once you go past 100% (or 125% with the "XP-style DPI scaling" checkbox ticked), Windows by default takes over the scaling of your UI. It does so by having your app render its output to a bitmap and drawing that bitmap to the screen. The rescaling of that bitmap makes the text inevitably look fuzzy. A feature called "DPI virtualization", it keeps old programs usable on high resolution monitors.

You have to explicitly let it know that you can handle higher DPI settings by adding the `<dpiAware>` element to your manifest. The MSDN page is here but it isn't complete since it is omitting the UAC settings. Project + Add New Item, pick "Application Manifest File". Edit the manifest text or copy/paste this:

```
<?xml version="1.0" encoding="utf-8"?>
<assembly xmlns="urn:schemas-microsoft-com:asm.v1" man
xmlns:asmv3="urn:schemas-microsoft-com:asm.v3" >
    <assemblyIdentity version="1.0.0.0" name="MyApplic
    <trustInfo xmlns="urn:schemas-microsoft-com:asm.v2
        <security>
            <requestedPrivileges xmlns="urn:schemas-mi
                <requestedExecutionLevel level="asInvo
            </requestedPrivileges>
        </security>
    </trustInfo>
    <asmv3:application>
        <asmv3:windowsSettings
xmlns="http://schemas.microsoft.com/SMI/2005/WindowsSe
            <dpiAware>true</dpiAware>
        </asmv3:windowsSettings>
```

```
    </asmv3:application>
</assembly>
```

You can also pinvoke SetProcessDPIAware() in your Main() method, necessary for example if you deploy with ClickOnce:

```
    [STAThread]
    static void Main() {
        if (Environment.OSVersion.Version.Major >= 6)
        Application.EnableVisualStyles();
        Application.SetCompatibleTextRenderingDefault(
        Application.Run(new Form1());              // E
    }

    [System.Runtime.InteropServices.DllImport("user32.
    private static extern bool SetProcessDPIAware();
```

UPDATE, this common need is finally a bit easier if you use VS2015 Update 1 or higher. The added manifest already has the relevant directive, just remove the comments.

Keyword for search so I can find this post back: dpiAware

Share Improve this answer

edited Nov 15, 2017 at 23:19

Follow

answered Nov 5, 2012 at 8:37

Hans Passant

940k ● 148 ● 1.7k ● 2.6k

Thanks, your solution works fine. The only problem left is that all images are now at their original sizes. I guess I'll have to find a way to add additional "retina"-icons to my GUI... – Boris Nov 5, 2012 at 8:53

@HansPassant I have the same problem with blurry fonts, and after applying this solution, my controls don't scale and they can't fit. How to get both to work? – gajo357 May 30, 2014 at 13:03

4 For anyone investigating this Win8 madness, `SetProcessDPIAware` is deprecated, and it also doesn't work properly (at least not in Win8.1), causing unpredictable scaling on different controls. I strongly recommend using the manifest approach instead. – Jason Williams Aug 27, 2014 at 22:20

5 Hmya, Windows 8.1 acquired per-monitor DPI. I wasn't aware that I needed that. – Hans Passant Aug 27, 2014 at 22:26

1 If you are going to use ClickOnce for deployment, you cannot use the dpiAware option in manifest, use SetProcessDPIAware() instead. – Tute Mar 17, 2015 at 13:10

19

Applications can be developed in two different mode.

The first one is to declare our application to be non-DPI-aware (not declaring anything will default to this). In this case the operating system will render our application under the **expected 96 DPI** and then will do to the bitmap scaling that we discussed before. The result will be a blurry looking application, but with a correct layout.

The second option is to declare the application as DPI-aware. In this case the OS will not do any scaling and will let your application render according to the original DPI of the screen. In case of a per-monitor-DPI environment, your application will be rendered with the highest DPI of all the screens, then this bitmap will be scaled down to the proper size for each monitor. Downscaling results in a better viewing experience than upscaling but you might still notice some fuzziness.

If you want to avoid that, you must declare your application as per-monitor-DPI-aware. Then you must detect when your application is dragged across different monitors and render according to the DPI of the current one.

Declaring the DPI awareness is done in a manifest file.

refer the following link [stackoverflow]

Share Improve this answer Follow

edited May 23, 2017 at 12:26

Community Bot

1 ● 1

answered Dec 4, 2014 at 5:58

shanthi_karthika

915 ● 2 ● 8 ● 23

what if users have a window in *restored* size and move it so parts of it are on different monitors? do we need to render everything twice and use the monitor boundaries as bounding boxes? how much of that is covered by the winforms libraries? – Cee McSharpface Oct 11, 2018 at 10:42

8

Using .NET Framework 4.7 and Windows 10 Creators Update (1703) or newer you must do the following things to configure high DPI support for your Windows Form application:

Declare compatibility with Windows 10.

To do this, add the following to your `manifest` file:

```
<compatibility xmlns="urn:schemas-microsoft.com:compat
  <application>
    <!-- Windows 10 compatibility -->
    <supportedOS Id="{8e0f7a12-bfb3-4fe8-b9a5-48fd50a1
  </application>
</compatibility>
```

Enable per-monitor DPI awareness in the `app.config` file.

Windows Forms introduces a new System.Windows.Forms.ApplicationConfigurationSection element to support new features and customizations added starting with the .NET Framework 4.7. To take advantage of the new features that support high DPI, add the following to your application configuration file.

```
<System.Windows.Forms.ApplicationConfigurationSection>
  <add key="DpiAwareness" value="PerMonitorV2" />
</System.Windows.Forms.ApplicationConfigurationSection
```

Important

In previous versions of the .NET Framework, you used the manifest to add high DPI support. This approach is no longer recommended, since it overrides settings defined on the app.config file.

Call the static EnableVisualStyles method.

This should be the first method call in your application entry point. For example:

```
static void Main()
{
    Application.EnableVisualStyles();
    Application.SetCompatibleTextRenderingDefault(fals
    Application.Run(new Form1());
}
```

The advantage of this is the support for dynamic DPI scenarios in which the user changes the DPI or scale factor after a Windows Forms application has been launched.

Source: [High DPI support in Windows Forms]

Share Improve this answer edited Dec 4, 2019 at 16:12

Follow

answered Aug 13, 2018 at 8:10

Wollmich

1,646 ● 1 ● 19 ● 51

5

None of these suggestions worked for me but, something happened after I removed the `Form.Font = new` ... from the `Form.Design.cs`, the form started to re-scale properly, it works if the Font is defined in the constructor or not at all. Why? somebody else may be able to explained, I just can talk about the changed I made and took me a few minutes to figured out it was the root cause for the form I was working on. Hope it helps.

Share Improve this answer

Follow

edited Mar 6, 2018 at 18:57

Xantium

11.6k ● 12 ● 70 ● 93

answered Mar 6, 2018 at 18:48

Vasco Bonilla

51 ● 1 ● 1

3

Since at least Visual Studio 2017 you just have to add a manifest file and uncomment this section:

```
<application xmlns="urn:schemas-microsoft-com:asm.v3">
    <windowsSettings>
        <dpiAware
xmlns="http://schemas.microsoft.com/SMI/2005/WindowsSe
    </windowsSettings>
</application>
```

Share Improve this answer

Follow

answered Oct 21, 2019 at 0:22

Aranxo

1,163 ● 8 ● 17

2 This did Not Work for me in Windows 11, Visual Studio 2022 (17.1.6) – gridtrak May 10, 2022 at 12:43

@gridtrak Thanks for your remark. I will have an eye on it. But I didn't install yet Windows 11. – Aranxo May 10, 2022 at 14:55

0

This is not an answer. This is my work around. None of the above answers or comments worked for me. I also searched for and tried other methods.

I have been using Visual Studio.NET with C# and Windows.Forms since it was originally released. Until VS 2022 and Windows 11 this year, setting the scale mode seemed to work fine. For some reason, *some* of my Form.Height values get reduced at run time. No problems so far with Form.Width being changed. For me, this problem started April 1, 2022 - so I first thought it was an April Fool's prank!

Anyway, I have given up trying *solutions* for now and decided it is more practical for me to just set the Form.Size in the constructor code.

I observe the Designer UI uses **Size** which it converts to `ClientSize` in its generated code as follows:

```
  this.AutoScaleMode = System.Windows.Forms.AutoScaleM
  this.ClientSize = new System.Drawing.Size(744, 109);
  this.ControlBox = false;
  this.DoubleBuffered = true;
  this.FormBorderStyle = System.Windows.Forms.FormBord
  this.StartPosition = System.Windows.Forms.FormStartP
```

My workaround in my Form's constructor looks like:

```
/// <summary>
/// Constructor
/// </summary>
public MyForm()
{
  // In the designer, MyForm.Size was entered and disp
  InitializeComponent();

  // At runtime, MyForm.Size is changed to 760, 111
  // I will Reset this form's Size here so I can get f
  this.Size = new Size(760, 148);
}
```

Platform:

- Windows 11 Professional
- Microsoft Visual Studio Professional 2022
- Version 17.1.6
- VisualStudio.17.Release/17.1.6+32421.90
- Microsoft .NET Framework version 4.8.04161
- C# Tools 4.1.0-5.22165.10+e555772db77ca828b02b4bd547c318387f11d01f
- HDMI 1920x1080 video (100% or no scaling)

Share Improve this answer

Follow

answered May 9, 2022 at 22:40

gridtrak

807 • 9 • 20

0

I've been battling this for years. Sometimes I make a change that works, sometimes it doesn't. Now I finally found a setting that worked for all monitors:

In your app.config file, add the following section. Please note the value says "SystemDPIAware"

your screen will look the same on every monitor

```
    <System.Windows.Forms.ApplicationConfigurationSect
    <add key="DpiAwareness" value="SystemDPIAware"/>
  </System.Windows.Forms.ApplicationConfigurationSecti
```

Share Improve this answer

Follow

answered Apr 10, 2023 at 16:20

Don P

539 • 6 • 13