Fixing DPI Scaling Issues in Qt Designer Projects

This guide explains how to fix DPI scaling issues in Qt projects that were developed using Qt Designer with many `.ui` files.

Understanding the Problem

Qt Designer makes it easy to drag-and-drop widgets and position them visually, which often results in:
- **Absolute positioning** (x, y coordinates) instead of layouts
- **Fixed pixel sizes** for widgets
- **Hard-coded margins and spacing**

All of these break on different DPI screens.

Strategy Overview

You have three main options:

1. **Fix the .ui files in Designer** (Best for maintainability)
2. **Override in C++ code after loading UI** (Quick fix, harder to maintain)
3. **Hybrid approach** (Most practical for large projects)

Option 1: Fix in Qt Designer (Recommended)

This is the most maintainable approach. You'll need to restructure your UI files to use layouts.

Step-by-Step Process

1. Open the .ui file in Qt Designer

```
designer mywindow.ui
```

2. Identify widgets using absolute positioning

Look in the property editor for widgets. If you see:
- **geometry** property set (x, y, width, height)
- No parent layout

These need to be fixed.

3. Convert to layout-based design

For a simple form:

1. **Delete the fixed geometry:**
- Select the widget
- In Property Editor, find "geometry"
- You can't delete it directly if it's using absolute positioning

2. **Select all widgets** that should be in a layout (Ctrl+Click multiple widgets)

3. **Right-click** → **Lay Out** and choose:
- **Lay Out Horizontally** (for side-by-side widgets)
- **Lay Out Vertically** (for stacked widgets)
- **Lay Out in a Grid** (for form-like layouts)
- **Lay Out in a Form Layout** (for label/input pairs)

4. **Select the parent widget** (the dialog/window itself)

5. **Apply a layout to the parent:**
- Right-click the parent → **Lay Out** → Choose appropriate layout
- Or click the "Lay Out Vertically/Horizontally" toolbar button

4. Set proper size policies

For each widget:
- Select the widget
- Find **sizePolicy** in Property Editor
- Set **Horizontal Policy** and **Vertical Policy** appropriately:
- **Expanding** for widgets that should grow (text edits, lists)
- **Preferred** for buttons, labels that can resize
- **Fixed** only if absolutely necessary

5. Remove fixed sizes where possible

- Find **minimumSize**, **maximumSize**, **sizeHint** properties
- Clear them unless specifically needed
- Let the layout calculate sizes automatically

6. Adjust margins and spacing

- Select the layout (click the parent widget, then look for the layout object)
- Set **layoutLeftMargin**, **layoutTopMargin**, **layoutRightMargin**, **layoutBottomMargin**
- Set **layoutSpacing**
- Start with values like 9-12 for margins, 6 for spacing (Designer uses logical pixels)

Example Transformation in Designer

Before (absolute positioning):

```
Window (no layout)
  ■■ QPushButton at (10, 10, 100, 30)
  ■■ QLabel at (10, 50, 200, 20)
  ■■ QLineEdit at (10, 80, 200, 25)
```

After (layout-based):

```
Window
  ■■ QVBoxLayout
      ■■ QPushButton (no fixed geometry)
      ■■ QLabel (no fixed geometry)
      ■■ QLineEdit (no fixed geometry)
```

Option 2: Override in C++ After Loading UI

If you have many UI files and can't redesign them all immediately, you can apply fixes in C++ after loading the UI.

Basic Approach

```
// mainwindow.h
class MainWindow : public QMainWindow {
    Q_OBJECT
public:
    MainWindow(QWidget *parent = nullptr);

private:
    Ui::MainWindow ui;
    void applyDpiAwareSizing();
};
```

```
// mainwindow.cpp
MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
{
    ui.setupUi(this);

    // Apply DPI-aware fixes AFTER setupUi()
    applyDpiAwareSizing();
}

void MainWindow::applyDpiAwareSizing() {
    // Get the base unit for DPI-aware sizing
    QFontMetrics fm(QApplication::font());
    int em = fm.height();

    // Fix button sizes
    ui.submitButton-&gt;setMinimumSize(em * 6, em * 2);
    ui.cancelButton-&gt;setMinimumSize(em * 6, em * 2);

    // Fix input field sizes
    ui.nameInput-&gt;setMinimumWidth(em * 15);
    ui.emailInput-&gt;setMinimumWidth(em * 15);

    // Fix icon sizes
    ui.iconButton-&gt;setIconSize(QSize(em, em));

    // Fix margins if there's a layout
    if (ui.centralWidget-&gt;layout()) {
        ui.centralWidget-&gt;layout()-&gt;setContentsMargins(em, em, em, em);
        ui.centralWidget-&gt;layout()-&gt;setSpacing(em / 2);
    }
}
```

Advanced: Dynamically Create Layouts

If your UI file uses absolute positioning, you can create layouts programmatically:

```
MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
{
    ui.setupUi(this);

    // Reparent widgets into a layout
    QVBoxLayout *layout = new QVBoxLayout(ui.centralWidget);

    // Remove widgets from absolute positioning and add to layout
    ui.titleLabel-&gt;setParent(nullptr);  // Detach from parent
    layout-&gt;addWidget(ui.titleLabel);

    ui.nameInput-&gt;setParent(nullptr);
    layout-&gt;addWidget(ui.nameInput);

    ui.submitButton-&gt;setParent(nullptr);
    layout-&gt;addWidget(ui.submitButton);

    // Apply DPI-aware margins
    QFontMetrics fm(QApplication::font());
    int em = fm.height();
    layout-&gt;setContentsMargins(em, em, em, em);
    layout-&gt;setSpacing(em / 2);
}
```

Option 3: Hybrid Approach (Most Practical)

For large projects with many UI files, use a phased approach:

Phase 1: Enable DPI Scaling Globally

```
// main.cpp
int main(int argc, char *argv[]) {
    // CRITICAL: Do this first
    QApplication::setAttribute(Qt::AA_EnableHighDpiScaling);
    QApplication::setAttribute(Qt::AA_UseHighDpiPixmaps);

    QApplication app(argc, argv);

    // ... rest of your code
}
```

This alone will help significantly, even if your UI files aren't perfect.

Phase 2: Create a Base Class for Common Fixes

```
// dpiaware_widget.h
#pragma once
#include &lt;QWidget&gt;
#include &lt;QDialog&gt;
#include &lt;QMainWindow&gt;
```

```
class DpiAwareHelper {
public:
    static void applyToWidget(QWidget *widget);
    static void applyToLayout(QLayout *layout);
    static int getEm();
};


// dpiaware_widget.cpp
#include "dpiaware_widget.h"
#include &lt;QApplication&gt;
#include &lt;QFontMetrics&gt;
#include &lt;QLayout&gt;
#include &lt;QPushButton&gt;
#include &lt;QLineEdit&gt;
#include &lt;QLabel&gt;

int DpiAwareHelper::getEm() {
    QFontMetrics fm(QApplication::font());
    return fm.height();
}

void DpiAwareHelper::applyToWidget(QWidget *widget) {
    int em = getEm();

    // Apply to all child widgets
    for (QObject *child : widget-&gt;children()) {
        QWidget *w = qobject_cast&lt;QWidget*&gt;(child);
        if (!w) continue;

        // Fix buttons
        if (QPushButton *btn = qobject_cast&lt;QPushButton*&gt;(w)) {
            btn-&gt;setMinimumHeight(em * 2);
            if (btn-&gt;minimumWidth() &lt; em * 5) {
                btn-&gt;setMinimumWidth(em * 5);
            }
        }

        // Fix line edits
        if (QLineEdit *edit = qobject_cast&lt;QLineEdit*&gt;(w)) {
            edit-&gt;setMinimumHeight(em * 1.5);
        }

        // Recurse into child widgets
        applyToWidget(w);
    }

    // Fix the layout
    if (widget-&gt;layout()) {
        applyToLayout(widget-&gt;layout());
    }
}

void DpiAwareHelper::applyToLayout(QLayout *layout) {
    int em = getEm();

    layout-&gt;setContentsMargins(em, em, em, em);
    layout-&gt;setSpacing(em / 2);

    // Recurse into nested layouts
    for (int i = 0; i &lt; layout-&gt;count(); ++i) {
        QLayoutItem *item = layout-&gt;itemAt(i);
        if (item-&gt;layout()) {
            applyToLayout(item-&gt;layout());
        }
    }
}
```

Now use this helper in all your windows:

```
// mainwindow.cpp
MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
{
    ui.setupUi(this);

    // One line to fix DPI issues!
    DpiAwareHelper::applyToWidget(this);
}
```

Phase 3: Fix Critical UI Files in Designer

Prioritize fixing the most important/visible UI files:
1. Main window
2. Frequently used dialogs
3. Forms with the worst scaling issues

Leave less critical dialogs for later or rely on the helper class.

Automated Approach: Script to Modify .ui Files

`.ui` files are XML, so you can write a script to modify them programmatically. This is useful for bulk changes.

```
#!/usr/bin/env python3
import xml.etree.ElementTree as ET
import sys
import os

def fix_ui_file(ui_file):
    """
    Remove fixed geometries and add layouts to a .ui file
    WARNING: This is a simplified example - real UI files are complex!
    """
    tree = ET.parse(ui_file)
    root = tree.getroot()

    # Find all widgets with geometry property
    for widget in root.findall('.//widget'):
        # Remove geometry property
        for prop in widget.findall('.//property[@name="geometry"]'):
            widget.remove(prop)
            print(f"Removed geometry from {widget.get('name')}")

    # Save backup
    backup = ui_file + '.backup'
    os.rename(ui_file, backup)

    # Save modified file
    tree.write(ui_file, encoding='utf-8', xml_declaration=True)
    print(f"Modified {ui_file} (backup saved as {backup})")

if __name__ == '__main__':
    if len(sys.argv) &lt; 2:
        print("Usage: fix_ui.py &lt;ui_file&gt;")
        sys.exit(1)

    fix_ui_file(sys.argv[1])
```

Warning: This script is simplistic. Real `.ui` files have complex structures, and removing geometries without adding proper layouts will likely break the UI. Use it as a starting point and test thoroughly.

Practical Workflow for Large Projects

Here's the recommended workflow if you have 50+ UI files:

Week 1: Foundation

1. Enable high DPI scaling in `main.cpp`
2. Create the `DpiAwareHelper` class
3. Apply helper to all windows/dialogs after `setupUi()`
4. Test on different DPIs

Week 2-3: Critical UIs

5. Identify the 10 most important UI files
6. Redesign them in Qt Designer with proper layouts
7. Remove `DpiAwareHelper` calls for these (no longer needed)
8. Test thoroughly

Week 4+: Gradual Migration

9. Fix remaining UI files one by one during regular development
10. When fixing bugs or adding features to a window, take the time to fix its layout
11. Eventually phase out the helper class

Testing Your Changes

Test in Designer

Qt Designer has a preview mode:
1. **Form** → **Preview** (or Ctrl+R)
2. Change the preview style to see how it looks on different platforms
3. Resize the preview window to test layout flexibility

Test at Runtime

```
// In your main window constructor, add debug output
MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
{
    ui.setupUi(this);

    // Debug: print screen DPI info
    qreal dpr = qApp-&gt;devicePixelRatio();
    QScreen *screen = QGuiApplication::primaryScreen();
    qreal dpi = screen-&gt;logicalDotsPerInch();

    qDebug() &lt;&lt; "Device Pixel Ratio:" &lt;&lt; dpr;
    qDebug() &lt;&lt; "Logical DPI:" &lt;&lt; dpi;
    qDebug() &lt;&lt; "Physical DPI:" &lt;&lt; screen-&gt;physicalDotsPerInch();
}
```

Test with environment variables:

```
QT_SCALE_FACTOR=1.5 ./myapp
QT_SCALE_FACTOR=2.0 ./myapp
```

Common Designer Mistakes and Fixes

Mistake 1: Using Spacers with Fixed Sizes

```
Bad: QSpacerItem with fixed width/height
Good: QSpacerItem with sizeType = Expanding
```

Fix in Designer:
- Select the spacer
- Set **sizeType** to "Expanding" (not "Fixed")

Mistake 2: Nested Widgets Without Layouts

```
Bad:
Window
  ■■ QWidget (no layout)
      ■■ QPushButton (absolute position)

Good:
Window
  ■■ QWidget
      ■■ QVBoxLayout
          ■■ QPushButton
```

Fix: Right-click the QWidget → Lay Out → Choose layout type

Mistake 3: Fixed Dialog Sizes

In Designer, dialogs often have fixed sizes set in the **geometry** property.

Fix:
- Select the dialog (click outside all widgets)
- Clear **minimumSize** and **maximumSize** properties
- Set only **sizeHint** if needed
- Or set **minimumSize** to a reasonable default, but not **maximumSize**

Mistake 4: Hard-Coded Font Sizes

```
Bad: font.pointSize = 14 (fixed)
Good: Leave default or use relative sizes
```

Fix:
- Select widgets with custom fonts
- In **font** property, clear **pointSize** (or set to -1)
- Use stylesheet for relative sizing if needed

Example: Converting a Complete UI File

Let's say you have this dialog designed for 1366×768:

Before (dialog.ui in absolute positioning):

```
&lt;ui version="4.0"&gt;
 &lt;class&gt;Dialog&lt;/class&gt;
 &lt;widget class="QDialog" name="Dialog"&gt;
  &lt;property name="geometry"&gt;
   &lt;rect&gt;
    &lt;x&gt;0&lt;/x&gt;
    &lt;y&gt;0&lt;/y&gt;
    &lt;width&gt;400&lt;/width&gt;
    &lt;height&gt;300&lt;/height&gt;
   &lt;/rect&gt;
  &lt;/property&gt;
  &lt;widget class="QLabel" name="titleLabel"&gt;
   &lt;property name="geometry"&gt;
    &lt;rect&gt;
     &lt;x&gt;10&lt;/x&gt;
     &lt;y&gt;10&lt;/y&gt;
     &lt;width&gt;380&lt;/width&gt;
     &lt;height&gt;30&lt;/height&gt;
    &lt;/rect&gt;
   &lt;/property&gt;
   &lt;property name="text"&gt;
    &lt;string&gt;Title&lt;/string&gt;
   &lt;/property&gt;
  &lt;/widget&gt;
  &lt;widget class="QLineEdit" name="nameEdit"&gt;
   &lt;property name="geometry"&gt;
    &lt;rect&gt;
     &lt;x&gt;10&lt;/x&gt;
     &lt;y&gt;50&lt;/y&gt;
     &lt;width&gt;380&lt;/width&gt;
     &lt;height&gt;25&lt;/height&gt;
    &lt;/rect&gt;
```

```
      &lt;/property&gt;
     &lt;/widget&gt;
     &lt;!-- More widgets with fixed geometry... --&gt;
    &lt;/widget&gt;
   &lt;/ui&gt;
```

After (with layouts):

Open in Designer and:
1. Select `titleLabel` and `nameEdit` (Ctrl+Click)
2. Right-click → **Lay Out Vertically**
3. Select the Dialog itself
4. Right-click → **Lay Out Vertically** (to apply layout to the dialog)
5. Adjust margins: Select layout → Set **layoutMargin** to 12
6. Adjust spacing: Set **layoutSpacing** to 6

The XML becomes:

```
   &lt;ui version="4.0"&gt;
    &lt;class&gt;Dialog&lt;/class&gt;
    &lt;widget class="QDialog" name="Dialog"&gt;
     &lt;layout class="QVBoxLayout" name="verticalLayout"&gt;
      &lt;property name="leftMargin"&gt;
       &lt;number&gt;12&lt;/number&gt;
      &lt;/property&gt;
      &lt;!-- ... other margins ... --&gt;
      &lt;property name="spacing"&gt;
       &lt;number&gt;6&lt;/number&gt;
      &lt;/property&gt;
      &lt;item&gt;
       &lt;widget class="QLabel" name="titleLabel"&gt;
        &lt;property name="text"&gt;
         &lt;string&gt;Title&lt;/string&gt;
        &lt;/property&gt;
       &lt;/widget&gt;
      &lt;/item&gt;
      &lt;item&gt;
       &lt;widget class="QLineEdit" name="nameEdit"/&gt;
      &lt;/item&gt;
      &lt;!-- More widgets without fixed geometry... --&gt;
     &lt;/layout&gt;
    &lt;/widget&gt;
   &lt;/ui&gt;
```

Notice: No `geometry` properties on the widgets!

Quick Reference: Designer Workflow

1. **Open UI file** in Qt Designer
2. **Select all widgets** that should be grouped (Ctrl+Click)
3. **Right-click** → **Lay Out** → Choose layout type
4. **Select parent container**
5. **Right-click** → **Lay Out** → Apply layout to container
6. **Select layout object** (may need to click parent, then switch to layout in Object Inspector)
7. **Set properties:**

- layoutLeftMargin, layoutTopMargin, etc.
- layoutSpacing
8. **For each widget, check sizePolicy:**
- Horizontal Policy
- Vertical Policy
9. **Save and test**
10. **Preview** with Form → Preview (Ctrl+R)

Summary Table: When to Use Each Approach

Approach	Best For	Pros	Cons
Fix in Designer	New projects, critical UIs	Most maintainable, proper solution	Time-consuming for many files
C++ Override	Quick fixes, legacy code	Fast to implement	Harder to maintain, code duplication
Hybrid (Helper Class)	Large projects (50+ UIs)	Balance of speed and quality	Temporary solution, needs cleanup later
Python Script	Bulk modifications	Can process many files quickly	Risk of breaking UIs, needs testing

---

Summary

For a Designer-based project with many UI files:

Best approach:

1. Enable DPI scaling in `main.cpp` first
2. Create a helper class to apply fixes programmatically
3. Gradually convert critical UI files to use layouts in Designer
4. Phase out the helper class as files are fixed

Key principle:

Layouts > Absolute positioning. If Designer created absolute positions, you must convert to layouts for proper DPI scaling.

The helper class buys you time to do the proper fixes gradually while still improving the user experience immediately.

Complete Example: Full Implementation

Here's a complete example showing all three approaches working together:

main.cpp

```
#include &lt;QApplication&gt;
#include &lt;QGuiApplication&gt;
#include "mainwindow.h"

int main(int argc, char *argv[]) {
    // Step 1: Enable DPI scaling BEFORE QApplication
    QApplication::setAttribute(Qt::AA_EnableHighDpiScaling);
    QApplication::setAttribute(Qt::AA_UseHighDpiPixmaps);
    QGuiApplication::setHighDpiScaleFactorRoundingPolicy(
        Qt::HighDpiScaleFactorRoundingPolicy::PassThrough
    );

    QApplication app(argc, argv);

    MainWindow w;
    w.show();

    return app.exec();
}
```

dpihelper.h

```
#pragma once
#include &lt;QWidget&gt;
#include &lt;QLayout&gt;

class DpiAwareHelper {
public:
    static void applyToWidget(QWidget *widget);
    static void applyToLayout(QLayout *layout);
    static int getEm();
};
```

dpihelper.cpp

```
#include "dpihelper.h"
#include &lt;QApplication&gt;
#include &lt;QFontMetrics&gt;
#include &lt;QPushButton&gt;
#include &lt;QLineEdit&gt;
#include &lt;QTextEdit&gt;
#include &lt;QComboBox&gt;

int DpiAwareHelper::getEm() {
    return QFontMetrics(QApplication::font()).height();
}

void DpiAwareHelper::applyToWidget(QWidget *widget) {
    int em = getEm();

    for (QObject *child : widget-&gt;children()) {
        QWidget *w = qobject_cast&lt;QWidget*&gt;(child);
        if (!w) continue;

        if (QPushButton *btn = qobject_cast&lt;QPushButton*&gt;(w)) {
            btn-&gt;setMinimumHeight(em * 2);
            btn-&gt;setMinimumWidth(em * 5);
        } else if (QLineEdit *edit = qobject_cast&lt;QLineEdit*&gt;(w)) {
            edit-&gt;setMinimumHeight(em * 1.5);
        } else if (QComboBox *combo = qobject_cast&lt;QComboBox*&gt;(w)) {
            combo-&gt;setMinimumHeight(em * 1.5);
        }

        applyToWidget(w);
    }

    if (widget-&gt;layout()) {
        applyToLayout(widget-&gt;layout());
    }
}

void DpiAwareHelper::applyToLayout(QLayout *layout) {
    int em = getEm();
    layout-&gt;setContentsMargins(em, em, em, em);
    layout-&gt;setSpacing(em / 2);

    for (int i = 0; i &lt; layout-&gt;count(); ++i) {
        if (QLayout *nested = layout-&gt;itemAt(i)-&gt;layout()) {
            applyToLayout(nested);
        }
    }
}
```

mainwindow.cpp

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include "dpihelper.h"

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
{
    ui-&gt;setupUi(this);

    // Apply DPI fixes to UI created by Designer
    DpiAwareHelper::applyToWidget(this);
}

MainWindow::~MainWindow() {
    delete ui;
}
```

This complete setup ensures your Designer-created UIs work across all screen resolutions and DPIs while you gradually migrate to proper layouts.