In WPF, setting a fixed Width on a UserControl or an internal child like TextEdit can be overridden by its parent container's layout logic, especially when using **Grid** and **StackPanel**.

The issue in your code is that the **Grid** and **StackPanel** in your LoginSignUpForm are providing the child controls (your FormInput controls) with as much space as they want to take, or they are forcing them to stretch to fill the container's available space.

Why your width is not getting fixed:

1. **Grid Layout Defaults:** In your LoginSignUpForm, you defined a Grid with RowDefinitions and ColumnDefinitions. Since you didn't specify a Width for the ColumnDefinition, it defaults to 1* (star sizing), which tells the column to expand to fill all available horizontal space.
2. **UserControl Constraints:** Even though you set Width="450" inside the FormInput.xaml root, the **parent** (the Grid/StackPanel in LoginSignUpForm) may be overriding this if it's set to stretch.
3. **HorizontalAlignment:** By default, many WPF controls have a HorizontalAlignment of Stretch. This means that even if you request 450, the parent might expand the control further to fill the window if the alignment is not explicitly set to something like Left or Center.

How to fix it:

To ensure your input boxes stay at 450 pixels regardless of the window size, you should set both the Width and the HorizontalAlignment property to prevent "stretching" behavior.

- **In your FormInput.xaml:**
  Set the HorizontalAlignment to Center or Left so it doesn't try to fill its parent's width.

xml

```
<dx1:TextEdit

  Text="{Binding UserName}"

  Height="100"

  Width="450"

  HorizontalAlignment="Center"
```

Label="{Binding LabelText, RelativeSource={RelativeSource AncestorType=UserControl}}"

/>

Use code with caution.

- **In your LoginSignUpForm.xaml:**
  Ensure the StackPanel or Grid doesn't force children to stretch. You can set the MaxWidth of the container or explicitly set the HorizontalAlignment of the FormInput instances.

xml

<Finp:FormInput LabelText="Enter Your Name" HorizontalAlignment="Center" Width="450" />

Use code with caution.

Tip: If you want the form to remain centered and fixed, it is often better to wrap your entire form in a container with a fixed MaxWidth or a centered DockPanel.

Would you like to see how to make the entire form **responsive** so it looks good on both small and large windows without using hard-coded pixel values?

10 sites

- How to set width and height to wrap to content of UserControl

12-Jun-2020 — Is there a way to wrap the width of a UserControl? The current XAML setup causes the UserControl to stretch to the screen width an...

- How to set WPF components transformation to not scale with ...

29-Mar-2025 — The big thing about WPF and xaml is that it can resize itself to fit the screen, and it can't do that if you specify the size like...

- WPF [] Viewbox seems to only scale objects Horizontally, but ...

21-Sept-2025 — When defining grids and columns etc, you use a number followed by asterisk. Two columns, with one having width 8* and the other ha...

yes

To make your WPF application responsive so it scales elegantly across different screen sizes, you should shift away from fixed pixel values like

Width="450" and use **fluid layouts**.

Key Responsive Design Strategies for WPF

1. **Use Proportional Sizing (Star Sizing):**
   In your Grid, replace fixed or default column widths with star (*) sizing. This tells the column to take a proportional fraction of the available space rather than a fixed number of pixels.

1. **Example:** <ColumnDefinition Width="*" /> makes a column fill all remaining space.

2. **Define Size Ranges:**
   Instead of hardcoding a single width, use MinWidth and MaxWidth. This allows your controls to grow or shrink within a safe range, ensuring they don't become unreadably small or excessively wide on large monitors.

1. **Constraint:** Set a MaxWidth on your main form container (e.g., 600px) so it remains readable on ultra-wide screens.

3. **The Viewbox Control for Scaling:**
   If you want your entire UI to scale like an image (getting bigger as the window grows), wrap your main layout in a Viewbox.

1. **Note:** This will scale everything, including text and button sizes, which is great for "kiosk-style" apps but might not always be ideal for standard desktop software.

4. **Adaptive Triggers for Breakpoints:**
   For more advanced responsiveness, use VisualStateManager with AdaptiveTrigger. This allows you to completely change the layout (e.g., switching from two columns to one) when the window width drops below a certain threshold, similar to web media queries.

Implementation Tip

Modify your LoginSignUpForm by removing fixed widths from the child controls and setting a MaxWidth on the central container. Set HorizontalAlignment="Stretch" for inputs so they fill the fluid columns you've created.

Would you like a **code example** showing how to refactor your current XAML into a fluid, star-sized grid?

To align the text to the left in your Finp:SaveButton, you need to update its internal UserControl definition so the dx:SimpleButton correctly receives and uses alignment properties 1.3.10.

Currently, your dx:SimpleButton is inside a Grid with hardcoded dimensions, and it isn't "listening" to the HorizontalContentAlignment you set on the parent Finp:SaveButton.

How to Fix

Modify your **SaveButton.xaml** (the UserControl) with these changes:

1. **Bind HorizontalContentAlignment:** Add a binding to the SimpleButton so it pulls the alignment from the UserControl.
2. **Use Stretch:** Set the SimpleButton to HorizontalAlignment="Stretch" within its internal Grid to ensure it fills the space allowed.
3. **Adjust Padding:** Sometimes DevExpress SimpleButton needs a small left padding to look correctly aligned.

Stack Overflow +2

Update your UserControl XAML to this:

xml

```
<UserControl x:Class="DevX.View.UserControls.FormInputs.SaveButton" ...>
  <Grid>
    <dx:SimpleButton
      Background="Transparent"
      BorderThickness="0"
      Foreground="White"
      Content="{Binding ButtonContent, RelativeSource={RelativeSource
AncestorType=UserControl}}"
```

```
        Command="{Binding MCommand, RelativeSource={RelativeSource
AncestorType=UserControl}}"


        <!-- ADD THESE THREE LINES -->

        HorizontalContentAlignment="{Binding HorizontalContentAlignment,
RelativeSource={RelativeSource AncestorType=UserControl}}"

        HorizontalAlignment="Stretch"

        Padding="5,0,0,0"


        Height="35" />

   </Grid>

</UserControl>
```

Use code with caution.

Why it wasn't working

- **Missing Binding:** The HorizontalContentAlignment="Left" you set in your main page was being ignored because the internal dx:SimpleButton didn't know it should look at that property.
- **Fixed Width:** The internal Width="120" on your SimpleButton was overriding the Width="88" you tried to set on the outside. Removing the internal fixed width allows the external setting to take control.

Microsoft Learn +2

Would you like to also adjust the **vertical alignment** or **padding** for the "Sign U