BuilderHMI.Lite - Simple drag-and-drop WPF Layout

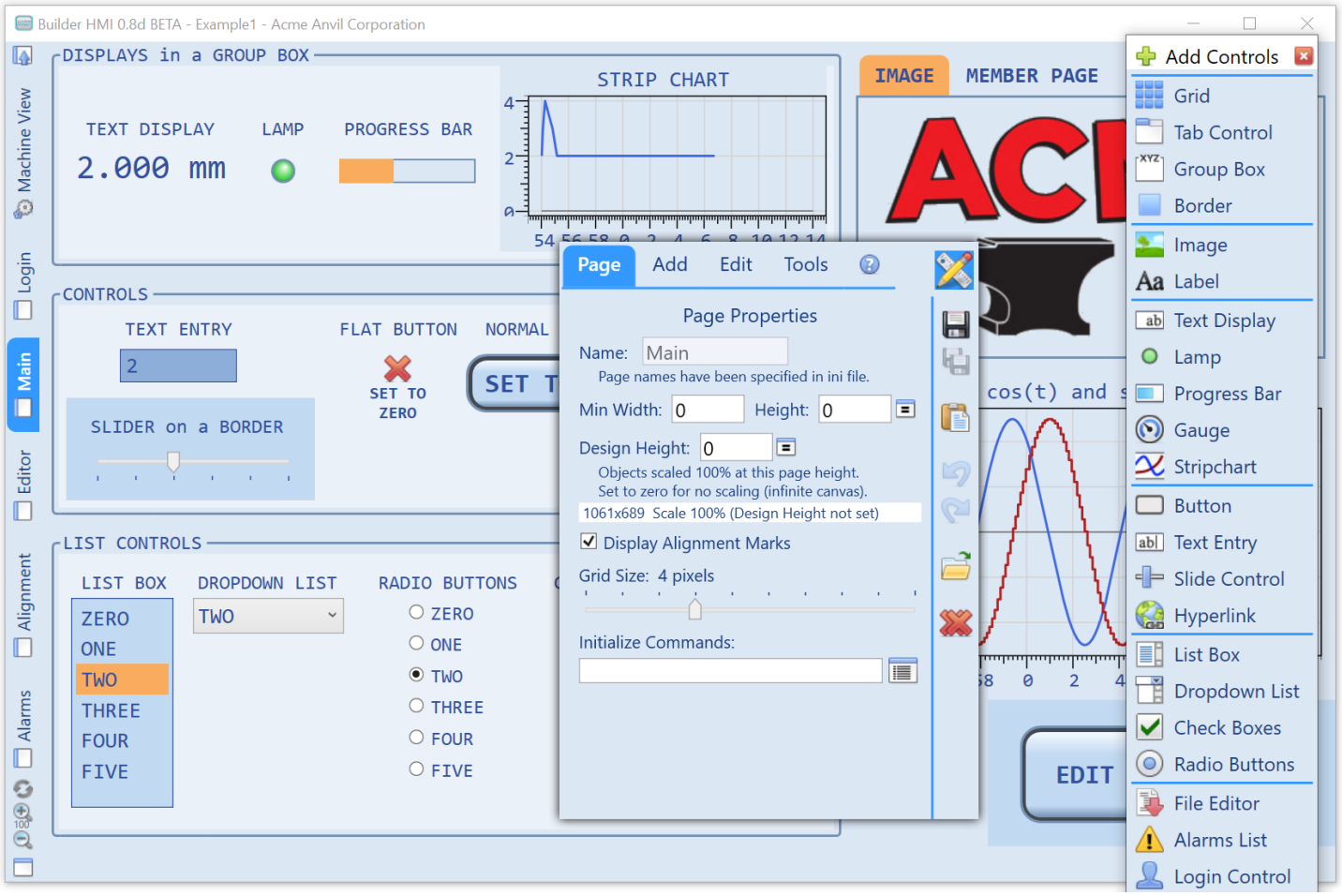
WPF UI design as quick and intuitive as Windows Forms! Did you hear that MICROSOFT ??

Background

BuilderHMI is a low-code and no-code *Rapid Application Development* (RAD) tool for industrial operator interface.

NO Code
UI pages can be quickly and easily designed in drag-and-drop fashion and then run directly by the app.

LOW Code
A complete WPF project with full C# source code can be generated for customization in MS Visual Studio.



Features

- Multiple project support with a New Project wizard.
- Ability to run standalone (Design mode and Run mode).
- Undo/redo stack for all UI design actions.
- Unlimited UI pages with navigation.
- The full set of standard WPF controls.
- Nested Grids and Tab Controls.
- Stripchart and other specialized controls for industrial HMI.
- Motion program file editor control with download.
- Choice of styles for Button, TextBlock, Border, etc.
- Save controls and complete pages to a Library for reuse.
- Robust communication with industrial motion controls.
- Controls send commands to motion control when clicked, etc.
- Alarms, message logging, user login system.
- Internationalization support for foreign language users.
- Specify styling and color scheme (Skin support).
- Browser-style Ctrl +/- zoom for UI screens.
- Live XAML window with direct edit feature.
- Customizable via external assemblies (plugins).

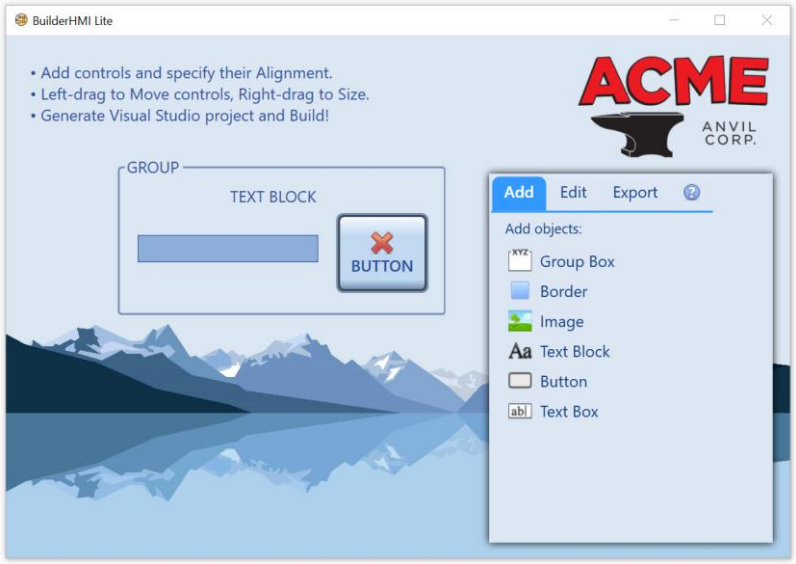
BuilderHMI.Lite

BuilderHMI.Lite is a subset of the full BuilderHMI app that focuses on WPF screen design and Visual Studio project generation.

CodeProject article:
https://www.codeproject.com/Articles/5283954/BuilderHMI-Lite-Simple-Drag-and-Drop-WPF-Layout

GitHub repository:
https://github.com/BruceGreene/BuilderHMI.Lite

YouTube introductory video:
https://youtu.be/xgmCmU76PFQ



Features

- WPF controls: Button, TextBox, TextBlock, GroupBox, Border and Image.
- Add controls then Left-drag to Move and Right-drag to Size.
- Left/Right/Center/Stretch horizontal alignment and Top/Bottom/Center/Stretch vertical.
- Editing functions: Cut/Copy/Paste/Delete, ToFront and ToBack.
- WPF Style support for the main window and all controls.
- Visual Studio WPF/C# project generation from a template.

The Controls

The BuilderHMI.Lite app includes a basic list of standard WPF controls: Button, TextBox, TextBlock, GroupBox, Border and Image. Each of the supported control types is implemented as an “Hmi” derived class that contains additional design-time information such as initial size and what resizing is possible. The derived class is also responsible for managing the control’s property pane and generating its XAML.

Each derived class must implement the IHmiControl interface which MainWindow uses to manage the collection of controls that have been added by the user.

```
public class HmiTextBlock : TextBlock, IHmiControl
{
    public HmiTextBlock()
    {
        Text = "TEXT BLOCK";
        SetResourceReference(StyleProperty, "TextBlockStyle");
    }

    public FrameworkElement fe { get { return this; } }
    public MainWindow OwnerPage { get; set; }
    public string NamePrefix { get { return "text"; } }
    public Size InitialSize { get { return new Size(double.NaN, double.NaN); } }
    public ECtrlFlags Flags { get { return ECtrlFlags.None; } }

    private static HmiTextBlockProperties properties = new HmiTextBlockProperties();
    public UserControl PropertyPage { get { properties.TheControl = this; return properties; } }

    public string ToXaml(int indentLevel, bool vs = false)
    {
        var sb = new StringBuilder();
        for (int i = 0; i < indentLevel; i++) sb.Append("    ");
        sb.Append(vs ? "<TextBlock Style=\"{DynamicResource TextBlockStyle}\"" : "<HmiTextBlock");
        sb.AppendFormat(" Name=\"{0}\"", Name);
        if (!string.IsNullOrEmpty(Text))
            sb.AppendFormat(" Text=\"{0}\"", WebUtility.HtmlEncode(Text).Replace("\n", "&#10;"));
        OwnerPage.AppendLocationXaml(this, sb);
        sb.Append(" />");
        return sb.ToString();
    }
}
```

XAML Generation

The ToXaml() method in each of the "Hmi" derived classes must be capable of generating XAML for the derived class (for cut/copy/paste) as well as XAML for the base class (for VS project generation). If you copy a button on the BuilderHMI.Lite design surface and then paste it into a text editor you will see XAML like this:

```
<HmiButton Name="button1" Text="DELETE" ImageFile="delete.png" HorizontalAlignment="Left"
  VerticalAlignment="Top" Margin="192,344,0,0" Width="92" Height="72" />
```

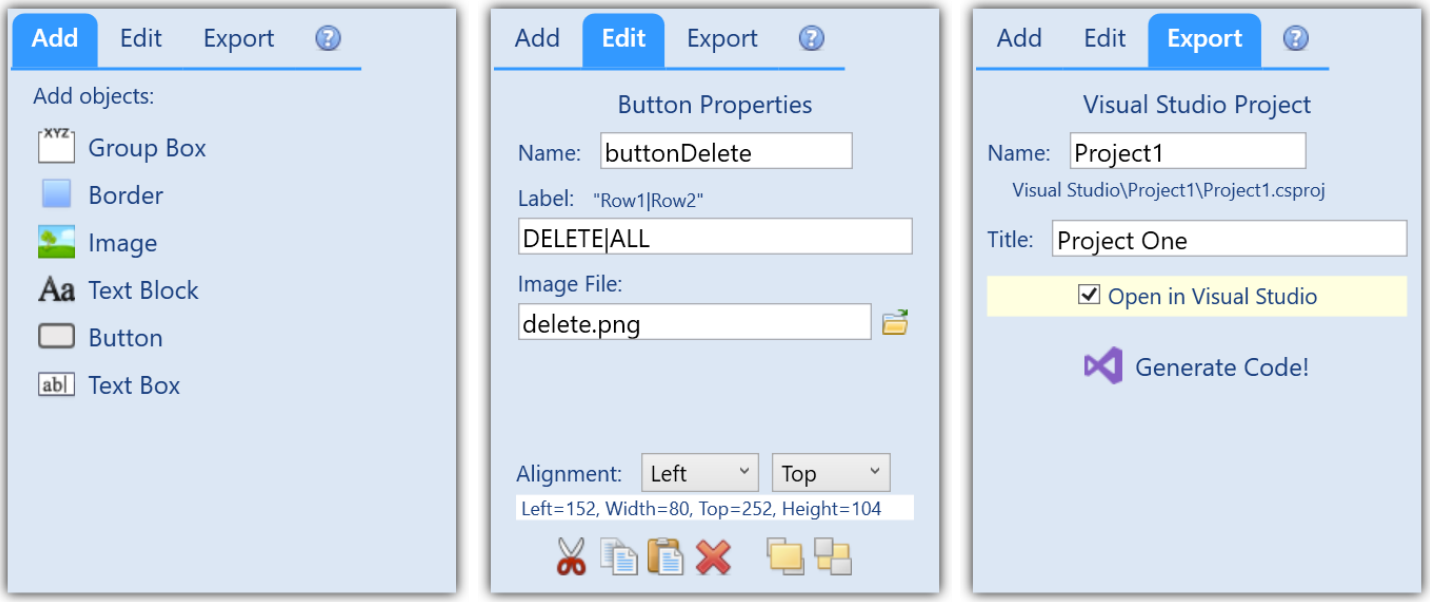
However, if you generate a Visual Studio project and open it you will see XAML like this:

```
<Button Name="button1" Style="{DynamicResource ButtonStyle}" HorizontalAlignment="Left"
  VerticalAlignment="Top" Margin="192,344,0,0" Width="92" Height="72">
  <StackPanel>
    <Image Source="Images/delete.png" Stretch="None" />
    <TextBlock Text="DELETE" />
  </StackPanel>
</Button>
```

Only "Hmi" derived classes can be pasted to the BuilderHMI.Lite design surface. However, the standard WPF control classes are used for Visual Studio project generation.

Designer Tool Window

The Designer tool window is pretty straight forward: An Add tab with the list of available controls, an Edit tab that hosts the selected control's property pane, and an Export tab for Visual Studio project generation. This window is tightly coupled to MainWindow and primarily just calls MainWindow methods when buttons are clicked.



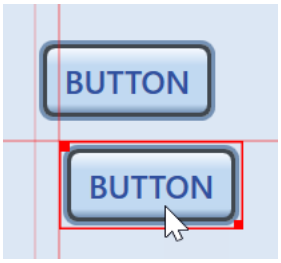
The Design Surface

The design surface is implemented by the code-behind of the MainWindow class.

- Adding new controls and generating unique initial names.
- Control selection, Left-drag-to-move, Right-drag-to-size and Alignment changes.
- Hotkey functions Cut/Copy/Paste, Delete, ToFront, ToBack and arrow key nudge.
- XAML generation and Visual Studio project generation.

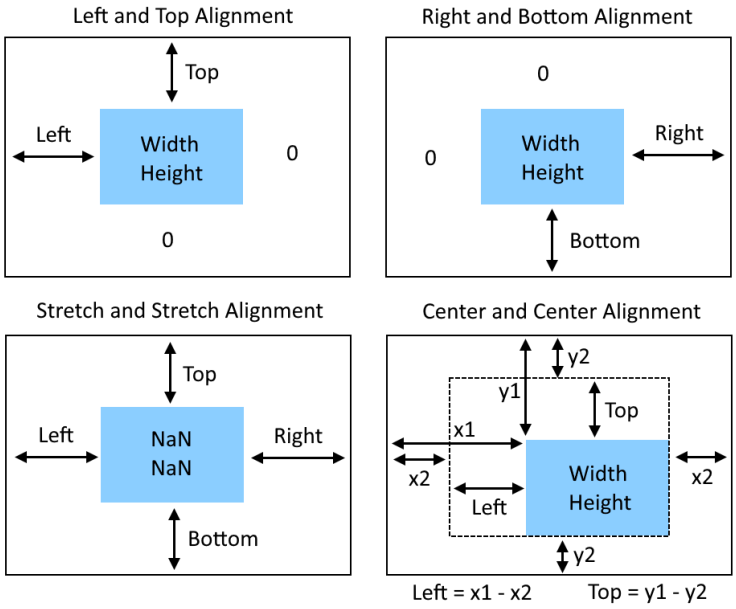
Left-Drag to Move and Right-Drag to Size

I chose the left-drag-to-move and right-drag-to-size method over grab handles because I wanted an absolute minimum of screen clutter to surround the selected control. The Marker class provides the box surrounding the selected control as well as the alignment marks while moving. The Location0 class is responsible for moving or sizing a control relative to its initial control and mouse cursor locations.



Alignment

WPF utilizes the Horizontal and Vertical Alignment, Margin, Width and Height properties to position a control within a Grid cell. The BuilderHMI.Lite app manages Margin, Width and Height to seamlessly enable the control to be moved or sized on the design surface regardless of the values of its alignment options. A control's horizontal or vertical location can always be described by two values. However, these values are different for each alignment option:



Styling

Styles for the supported control types are in “Styles.xaml” with a Colors section at the top of the file.

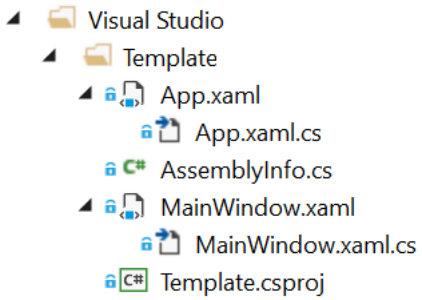
```
<Style x:Key="TextBoxStyle" TargetType="TextBox">
    <Setter Property="Foreground" Value="{DynamicResource TextBrush}"/>
    <Setter Property="Background" Value="{DynamicResource TextBoxBackgroundBrush}"/>
    <Setter Property="BorderBrush" Value="{DynamicResource TextBrush}" />
    <Setter Property="MinWidth" Value="35" />
    <Setter Property="Padding" Value="2"/>
</Style>
```

All resource references are dynamic which prepares the app to be upgraded to allow skin changes on the fly.

```
public HmiTextBox()
{
    SetResourceReference(StyleProperty, "TextBoxStyle");
}
```

Visual Studio Project Generation

When you first contemplate writing an app that can generate fully buildable Visual Studio WPF/C# projects, it sounds like a daunting task. The happy news is that it’s very simple. First use VS to create a representative project, then add its various source files to your app’s project under a Template folder. The key is that these source files all have their Build Actions set to None and Copy to Output Directory set to Copy if Newer.



Replace the actual project name, project namespace, etc with tags such as “__PROJECT_NAME__”. Now all your VS project generator has to do is copy these template files to the new project folder while performing a few text replacements.

```
// Generate AssemblyInfo.cs from the template:
string path = Path.Combine(templateFolder, "AssemblyInfo.cs");
string text = File.ReadAllText(path).Replace("__PROJECT_NAME__", projectName);
path = Path.Combine(projectFolder, "Properties");
Directory.CreateDirectory(path);
path = Path.Combine(path, "AssemblyInfo.cs");
File.WriteAllText(path, text);
```

The critical step for BuilderHMI.Lite of course is the generation of MainWindow.xaml. Each control that the user has added must be written out in XAML format and in Z-order.

```
// Generate xaml for all controls in Z-order:
var sortedChildren = new SortedList<int, IHmiControl>(gridCanvas.Children.Count);
foreach (object child in gridCanvas.Children)
{
    if (child is IHmiControl control)
        sortedChildren[Panel.GetZIndex(control.fe)] = control;
}
var sb = new StringBuilder();
foreach (IHmiControl child in sortedChildren.Values)
    sb.AppendLine(child.ToXaml(2, true));
```

Points of Interest

From a WPF developer's perspective, I feel that these are the most interesting aspects of the project:

- Intuitive mouse drag-and-drop layout of WPF controls in a Grid cell.
- Ability to change control alignment properties independent of location.
- XAML generation that supports both copy and paste operations and VS project generation.
- Surprisingly simple Visual Studio project generation via template files.

Author's Note: I began development of drag-and-drop layout in the obvious way – by using a WPF Canvas. That's what it's for right? However, as I ventured further down the rathole of implementing alignment beyond Left-Top I realized that I was duplicating the elegant alignment system already present in a WPF Grid! I then went back to the drawing board (literally) and worked out the relationships between alignment, margin and control dimensions in order to decouple alignment from location. The result of that work is what you have here!

Conclusion

Let's imagine that your business regularly needs to create customized versions of a core WPF app. An example from my field would be a manufacturer of industrial machinery where each machine can be ordered with different options such as cutting heads, sensors or the number of axes of motion.

This project becomes very helpful under the above scenario. It can easily be modified to generate your business app's core source code and additional control types can be added to the UI design framework.

- Customized versions of the core app could then be generated much more rapidly and reliably.
- UI screens could then be designed by people who are not experienced WPF/C# developers.

The full version of BuilderHMI extends this idea by supporting a Library of previously saved controls and entire UI pages for reuse.