

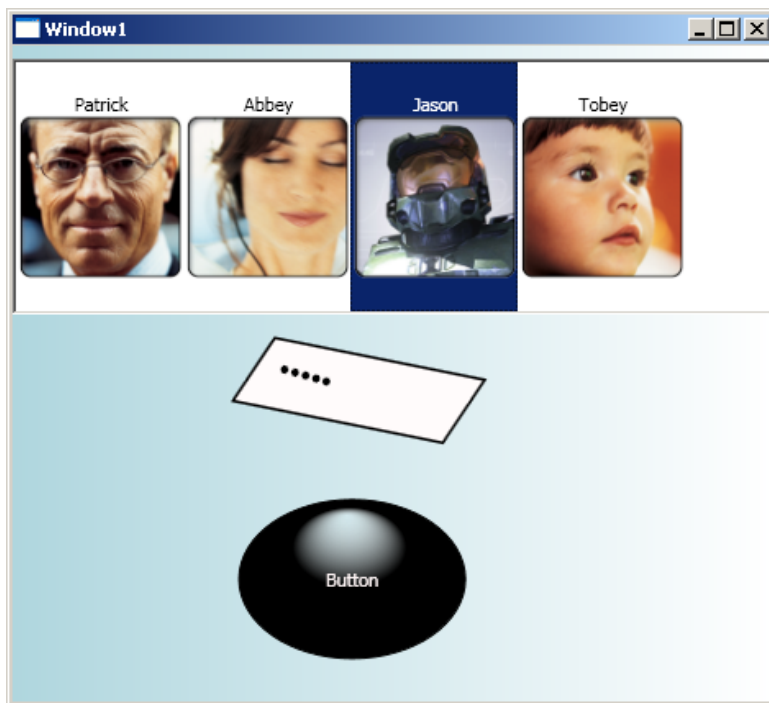
Building the Logon Screen in WPF

Demo Script

Prepared by: Jaime Rodriguez

Version: 0.1

9/17/07



This is a developer centric point demo in building a styled window with control, data templates and styles.

The application will show case workflow between VS 2008 and Expression Blend.

At the end of the exercise, we will have a data driven, heavily customized app and we will have written no code at all; we will have shown seamless transitions and integration between Blend and Cider; this is a canonical example of the workflow we expect developers and designers to go through when building WPF apps.

Key Messages:

1. A designer and developer can use Expression Blend and Microsoft Visual Studio 2008 to create compelling, rich, styled WPF applications.
2. Expression Blend is great for styling, templating, and other visual editing of a WPF application.
3. VS 2008 “Cider” is great for plugging in code, debugging, basic XAML editing, etc.
4. Both Blend and VS 2008 complement each other and seamlessly integrate for a developer and designer to easily and frictionless create rich WPF application.

Key Technologies:

The following technologies are utilized within this demo:

Technology / Product	Version
1. Windows Presentation Foundation	3.0
2. Expression Blend 2	Alpha Preview
3. Visual Studio 2008	RTM

Opening Statement

During the next 20 minutes we will build a few of the controls and features needed to create the Logon Screen Demo. We are not going to build all the screens that would be boring, but let’s build enough to learn the how-to’s.

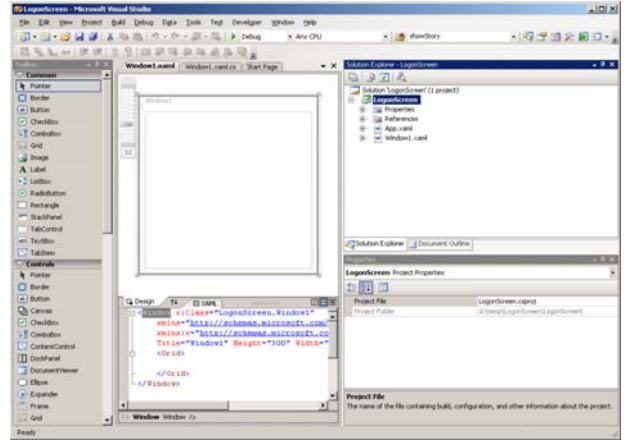
What you will see is how Visual Studio 2008 and Expression Blend seamlessly share the VS 2008

Step-by-step Walkthrough

Estimated time for setting up and configuring the demo: 0 minutes.

Estimated time to complete the demo: 15 minutes.

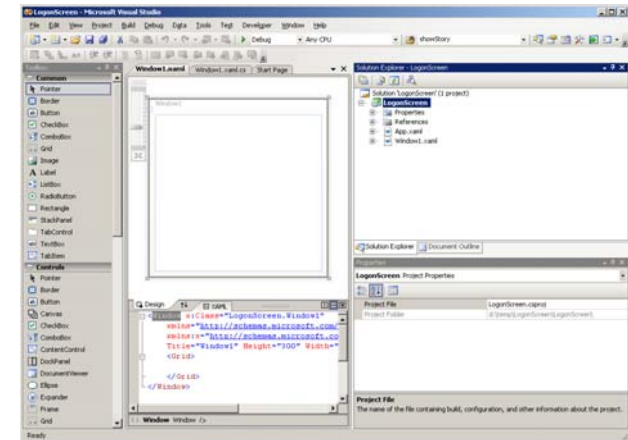
Creating the Logon Screen

Action	Script	Screenshot
<ol style="list-style-type: none">1. Launch VS 2008.2. File -> New -> Project3. Select WPF Application4. Enter a location to save5. Click OK	<ul style="list-style-type: none">• We begin from scratch by creating a new WPF project.• The WPF project template selected here came with WPF.• Here you see the default view in “Cider”; it is called a SplitView. It is very handy because I while I drag & drop visual elements I can still see the XAML that is being generated, tweak it by hand, etc.	

[Showing off Split View]

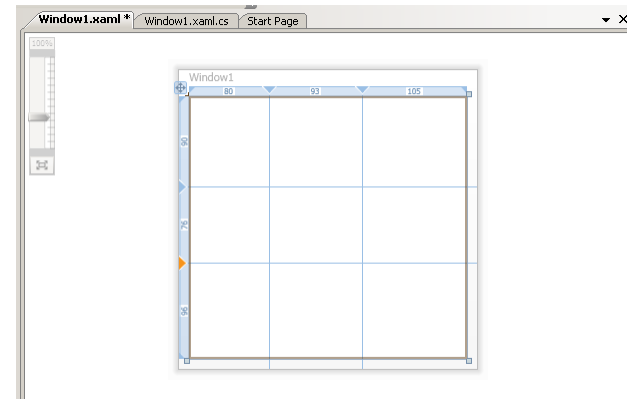
6. Click on the Collapse button in SplitView
7. Click on “Design” tab to show design view
8. Click on “XAML” tab to show XAML Editor
9. Click on the Arrow to “Swap Panes”
10. Click on the Grid in the Design surface
11. Click on the Grid in XAML so you can show SelectionSync as you go back and forth.

- If I need space, I can choose to view XAML or Design view only.
- The XAML editor is a rich editor with reflection-based intellisense, which means any user control or custom I create will still be found by the intellisense if I load it in the designer.
- Notice that as I change selection anywhere (e.g. the design surface) the XAML Editor and the property editor are synced with my selection. This feature is called Selection Sync and is incredibly handy.



12. Click twice at the top of the grid to add three columns.
13. Click twice at the left of the grid to add three rows.

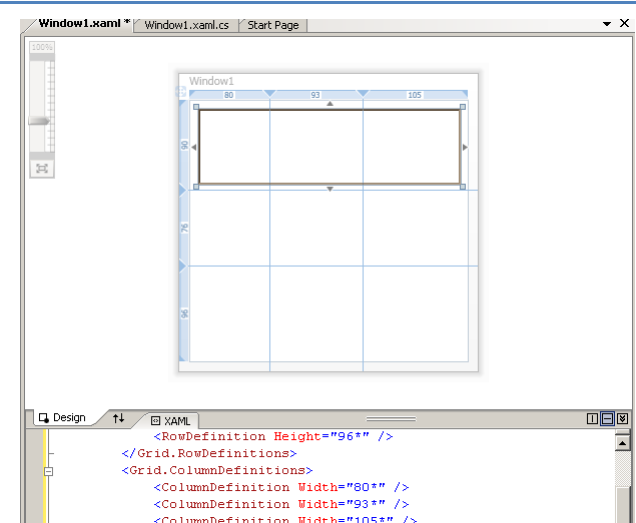
- Let’s create the basic layout for the app.
- We will reuse the Grid that the template added for us, as you can see, to add columns and rows all I have to do here is drag & drop; same to resize the columns and rows.
- When set to stretch the Grid resizes with our window, so we will reuse that for our demo.
- Here we will create three rows and three columns.



14. Drag a ListBox from the Toolbox to the Grid's first row.

15. Have it expand 3 columns wide.

- Let's begin by putting a ListBox on our grid. We drag & drop from the toolbox.
- As I resize the ListBox here in Row0, Cider is smart enough to know how many columns the Grid is taking. It is generating the right code for us, in this case Grid.Column="0".

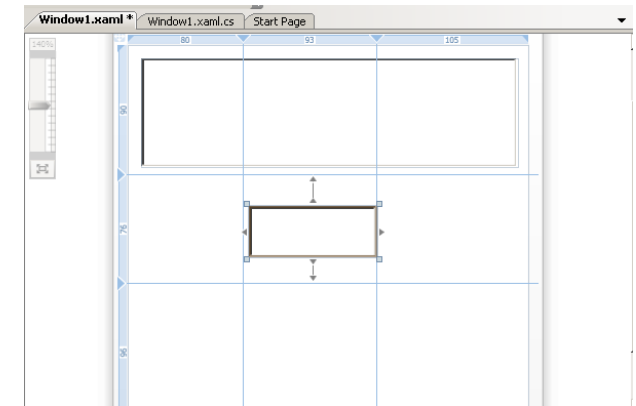


16. Drag a PasswordBox from the Toolbox

17. Position it in the Grid in Row=1, Column=1

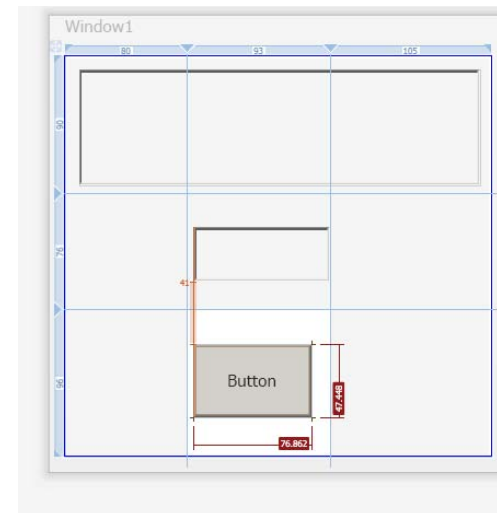
18. Play with alignment by clicking on the arrows around the PasswordBox ultimately you want the PasswordBox to stretch (so all 4 arrows showing)

- We now Drag & Drop the password box, we position it in Row=1, Column="1".
- Notice these arrows or guides around the PasswordBox these are used to control alignment.
- If I click on the Top to not have a guide, that means the control will align to the bottom; if I have arrows on top & bottom that means it stretches instead of using margin. Since this is a grid and we did not set a fixed size for this column, the column will grow/shrink with our window, and if we set alignment to stretch the column will stretch too.



19. Drag a button from the toolbox to the grid. Row =2 Column=1
20. Helps if the button has a squarish look since we are going to build a round button.
21. When dragging the button, move it around so it aligns with the PasswordBox and SnapLines show up.

- We now drag a button to the last row.
- Notice as I move the button and it aligns with the PasswordBox I get SnapLines to guide me for alignment.



22. Double Click on the button
23. In the handler enter anything in a MessageBox
MessageBox.Show ("works");

- We have basic layout done.
- Let's now wire up our event for sign-in.
- All we have to do is double click on the button and the designer will do the wiring for us.
- Let's put a MessageBox.Show () in the handler so we know it works.

```

using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace LogonScreen
{
    /// <summary>
    /// Interaction logic for Window1.xaml
    /// </summary>
    public partial class Window1 : Window
    {
        public Window1()
        {
            InitializeComponent();
        }

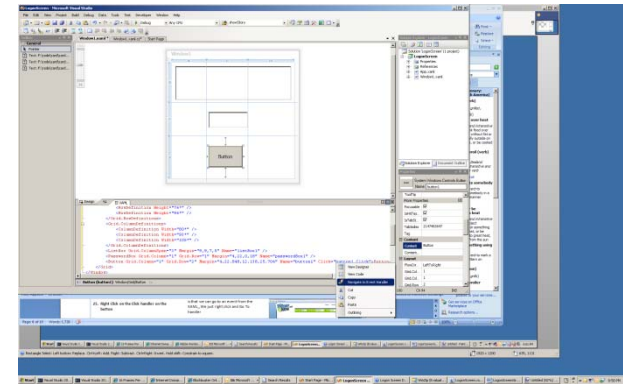
        private void button1_Click(object sender, RoutedEventArgs e)
        {
            MessageBox.Show ( "works" );
        }
    }
}

```

24. Go back to XAML Editor

25. Right Click on the Click handler on the button

- Another nice feature from the XAML Editor is that we can go to an event from the XAML. We just right select “Navigate To Event handler”.



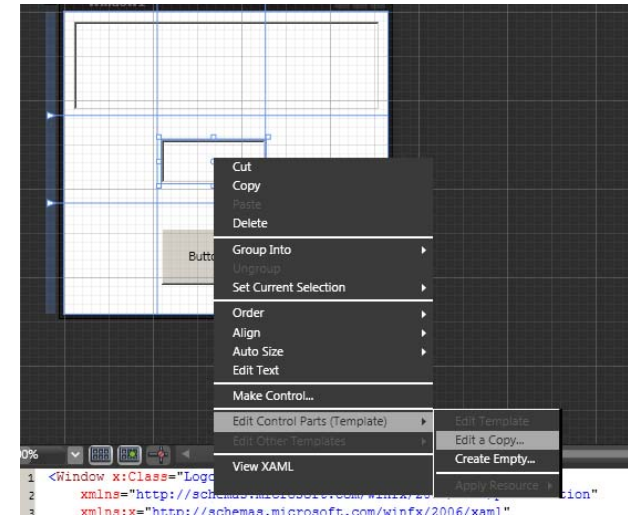
26. Launch Expression Blend 2

27. Open the solution you created in our very first step.

- At this point we have a basic app; we did all the layout and event wiring inside Visual Studio. I did not show you debugging or a lot of XAML editing but of course all that works.
- Let's now move to Expression Blend where we will do the styling and templating so we get the look and feel from the logon screen.
- Notice that we are opening the same project we created in Visual studio; the C#proj and the xaml files actually all the files are shared by the tools.

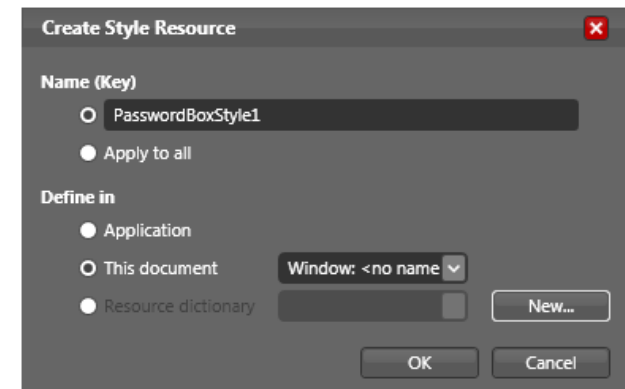
28. Right click on the PasswordBox and select Edit Control Parts -> Edit a Copy.

- Let's first create a PasswordBox like the one we saw on Tobey's screen.
- We will do this by Editing the ControlTemplate for the PasswordBox.



29. When the dialog comes up, Click New to create a ResourceDictionary

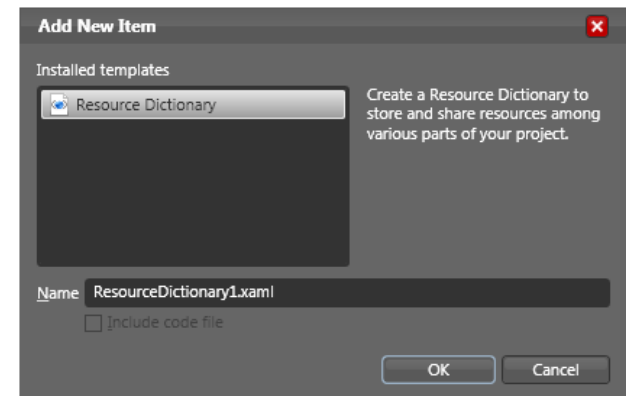
- We will put our template in a ResourceDictionary, this is not a requirement but it helps keep the code clean and readable, so let's click New to create a dictionary and then save our template there.



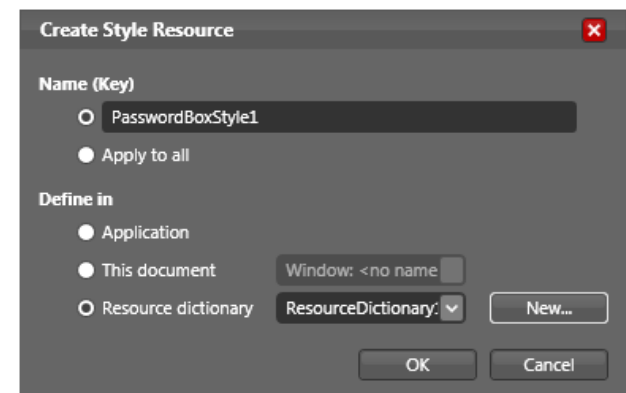
30. Accept the defaults

- We take the defaults.

31. Click OK



32. Click OK to create the template

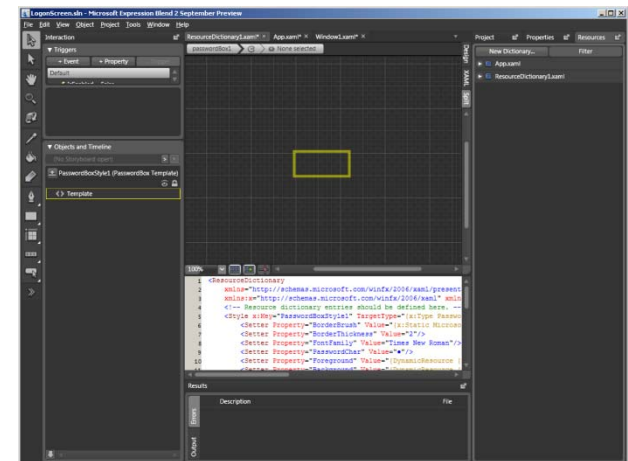


33. Select the ContentHost

34. Click Ctrl-X [to cut it]

35. Select the Border (Bd)

- What we see now is the Control Template for PasswordBox.
- Notice the template is just a border with a ContentHost inside it.
- Let's Cut the ContentHost and delete the Border.
- Our template is now empty the password box has no look.



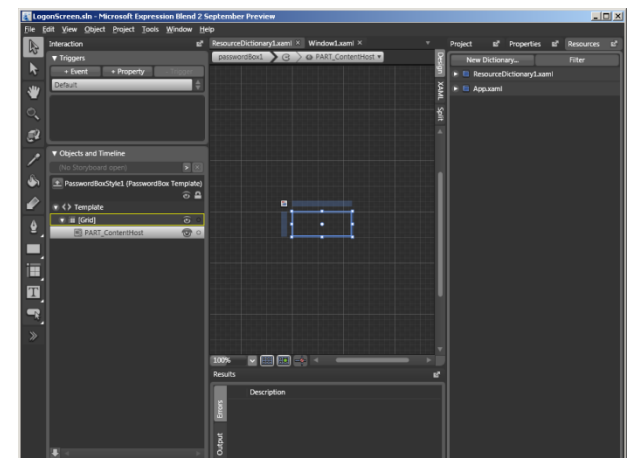
36. Select Template in the Objects tree

37. Double Click on Grid from the Toolbox to add a Grid to the template

38. Select the Grid in Objects tree

39. Press Ctrl-V to paste the previously cut ContentHost

- We now add a grid as the container. The reason we are adding a grid is because it stretches dynamically to fill its available area and it allows for its children to overlap (or overlay).
- Into the grid we paste our previously cut ContentHost.



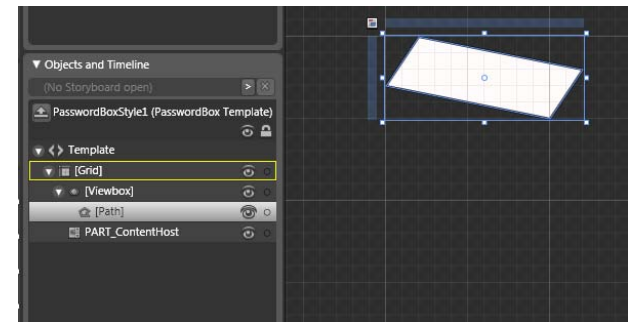
40. Select the Grid
41. Select the Pen tool
42. Draw a Path that looks like the rectangle in the PasswordBox
43. From the toolbox, add a ViewBox into the grid
44. Move the ViewBox backwards so it is behind the PasswordBox.
45. Make sure ViewBox is set to stretch both horizontal and vertical (that is the default).
46. Drag the Path into the ViewBox.



I realize it is counter intuitive to add the path first into the Grid and then move it into the ViewBox; the problem is the ViewBox has no size, so you will have a harder time drawing the Path in try it and decide.

Your visual tree should look like the image on Right hand side.

- Since we want the PasswordBox to have this slanted look, let's draw a Path that we will use as the background.
- To scale the background, let's add a ViewBox into the grid.
- Now we move the ViewBox so it is behind the ContentHost.
- Now we move the path into the ViewBox.



47. Select the PART_ContentHost in the template

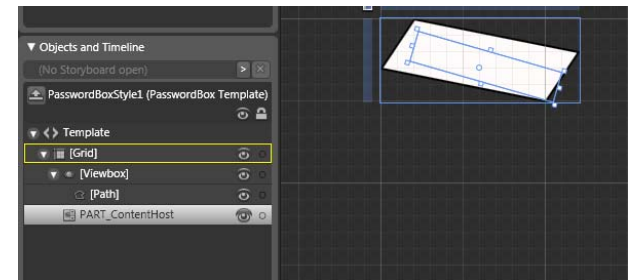
48. Apply a RotateTransform to it so it aligns with our Background Path

49. I recommend you make it fit inside the Path but don't set a Width/Size keep it to stretch with a margin. Just center it nicely and minor stretching will work fine.



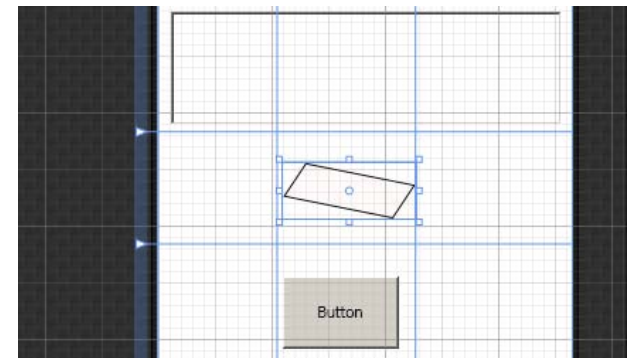
If you will be resizing the PasswordBox a lot you should put the ContentHost in a ViewBox too.

- At last we just need to rotate our content host so it fits within the background we added.



50. Click the “return to scope” button in the Objects tree to finish editing the template.

- We are now done with our template for the PasswordBox, notice we have a very different look & feel from where we started and we have not written any code.
- Notice on our main screen we now have the password box with new look & feel.



51. Right Click on the button

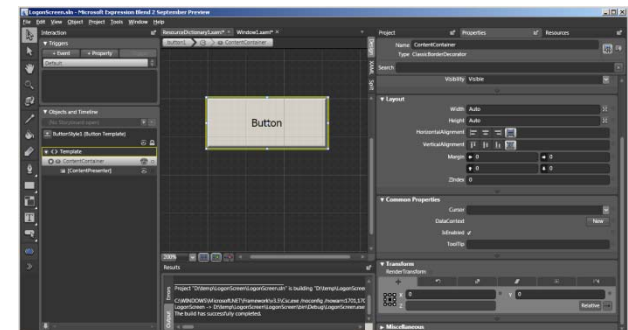
52. Select Edit Control Parts (Template)

53. Select Edit a Copy

54. Click on Resource Dictionary to add it to the existing RD

55. Click OK

- Let's Edit the Template for the button.
- We will add it to our existing ResourceDictionary.
- Here is the template for button, it is specialized Chrome and a ContentPresenter (or in XP it might be a ContentControl plus ContentPresenter).



56. Cut the ContentPresenter like we did earlier

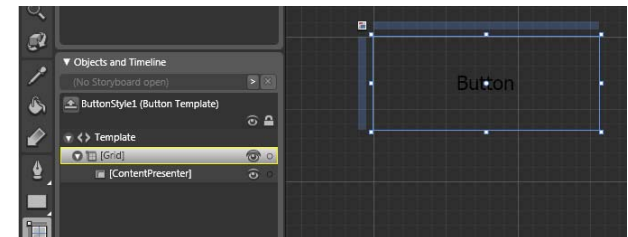
57. Delete the Container

58. Add a Grid from the toolbox

59. Select the grid so it is active

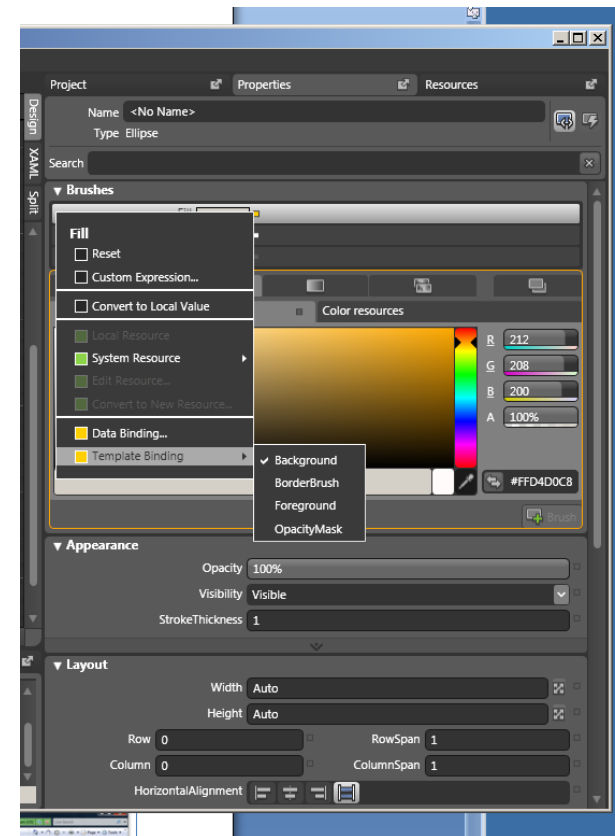
60. Paste the content presenter into Grid.

- Let's cut the presenter; we will use it shortly just got to replace the container.
- We will again add a grid so we benefit from resizing.
- Let's now add back our content presenter to the Grid.



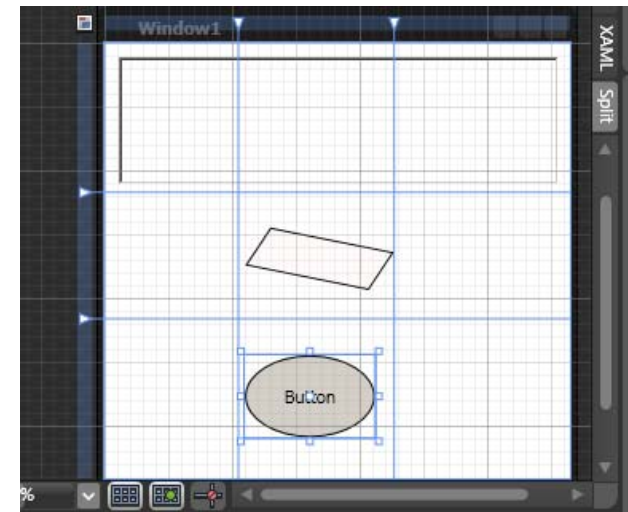
61. Select the Grid
62. Add an Ellipse into the grid
63. Make sure the Ellipse is set to stretch on all directions
64. Make sure the Margins are 0 on all directions
65. Right click on ellipse to send it back behind the content presenter (order - > send to back)
66. With Ellipse selected, Go to properties window
67. Find Fill property
68. Click on the little square next to the Color rectangle in the Fill field
69. Select TemplateBinding - > Background

- Since we want a round button, lets add a ellipse as the background for the button.
- We make sure the Ellipse will stretch to the whole space available, remove any margins.
- We move it back so it is behind our presenter.
- And now we TemplateBind the ellipse's Fill button to the control's background; this way if at design-time we want to change the background we can do so in the control and the ellipse in the template will pick up the changes.



**70. On object tree, Click
Return To Scope (Window)**

- Even though we are missing the glow effects, let's say we are done with the button. I think we have customized the look & feel enough for you to get the idea.
- Our main screen now has the template controls, look how our button is now round but it does not have yet the right background.



71. On object tree, Select the button

72. Go to properties Window

73. Select Background and add a Gradient Brush. Any brush will do If you want a highlight like the brush in the demo, try this:

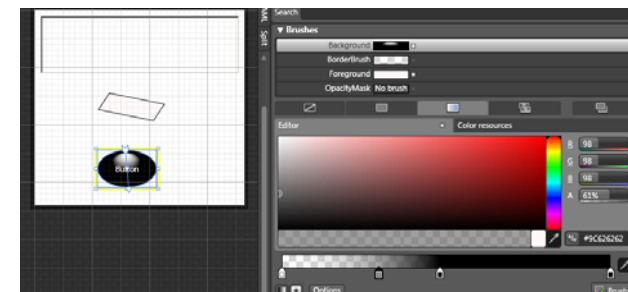
```
<RadialGradientBrush  
GradientOrigin="0.478,0.101">
```

```
<GradientStop Color="#FF000000"  
Offset="1"/>
```

```
<GradientStop Color="#FF000000"  
Offset="0.481"/>
```

```
<GradientStop Color="#00FFFFFF"  
Offset="0"/>
```

- Let's just give our button a background.
- I am going to cheat here and use a RadialGradientBrush since I did not earlier add a second ellipse into the ControlTemplate. We will just make a black brush with some transparent area and use the Brush Transform tool to off-center it.

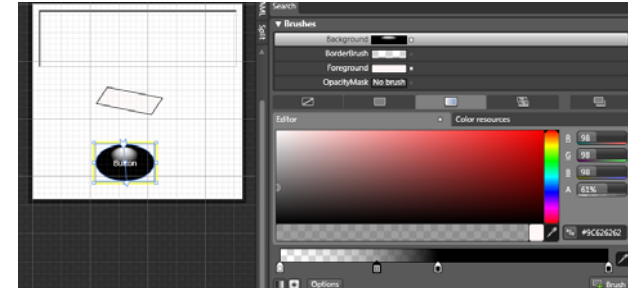


```
<GradientStop Color="#9C6262"
Offset="0.295"/>
</RadialGradientBrush>
```

74. Click on Button

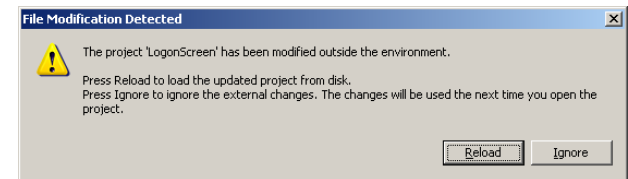
75. On properties Window, Change
Foreground to White

- At last let's just put a white foreground on our button.



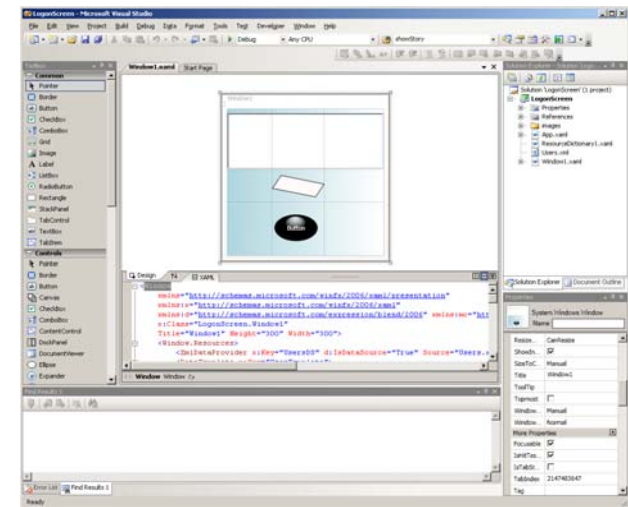
76. Alt-Tab back to Visual Studio.

- Lets Alt-Tab back to Visual Studio.
- Notice VS is saying our project was modified. So let's let it reload.



77. Let it reload, or recompile it

- Notice all the changes we did in Blend are here too. Everything is in sync.
- I can resize the button or the PasswordBox can change my handler, etc.
- This seamless transition from the tools is critical because it allows us to create tools that target the specific roles (developers versus designers) yet make sure they are in sync and working in parallel in some cases.
- Back to regular programming. Once we add the DataBinding code, don't come back to Visual Studio or at least don't open the XAML file in the designer you can still debug, change code, etc. but for the most part you won't need to you can complete the rest of the demo inside Blend.



78. In Blend

79. Click on the Project Tab

80. Click on the Project

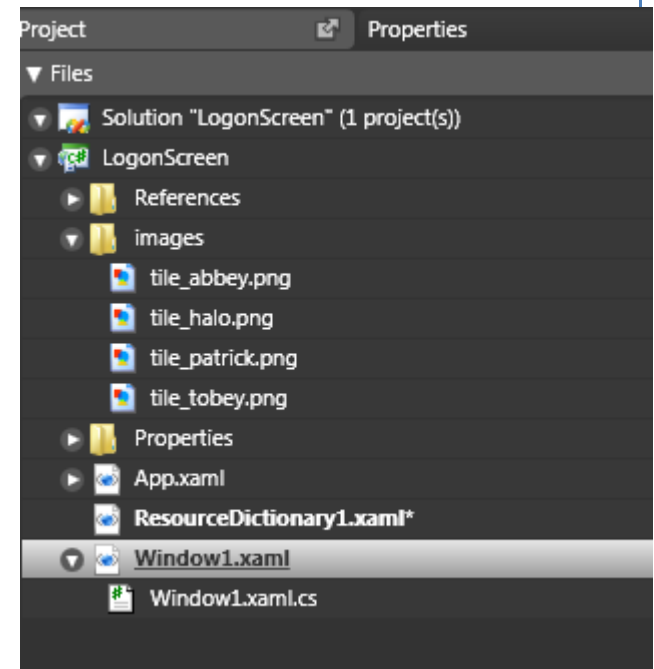
81. Select New Folder

82. Rename the new Folder to “images”

83. Add Existing Item

84. Navigate to the Assets folder Assets, look in the images directory and select all the images.

- At last, let’s just populate our ListBox.
- We first have to add the images for the ListBox.
- We create a new Folder.
- Navigate to our existing images and add them.



85. Again, Add Existing Item

86. Navigate to the Assets folder Assets and select the Users.xml file.

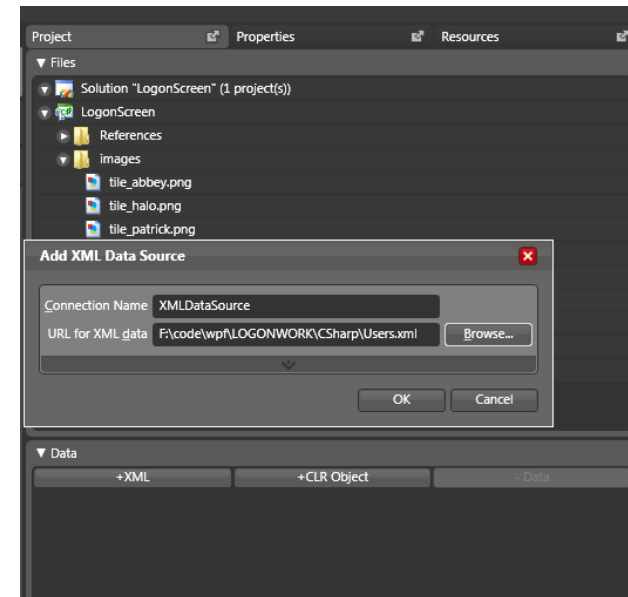
87. On project Tab, under Data section

Click in + XML button

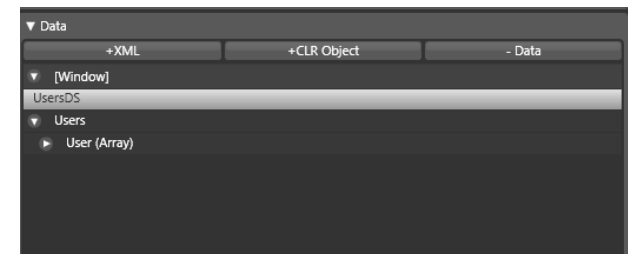
The prompt comes up, select the Users.xml file in the solution

Click OK to select it

- Now we cheat a little and bring a canned XML file that has the list of users and their images. This of course could have come from a database or CLR business object. In this case an XML file is portable for a demo.

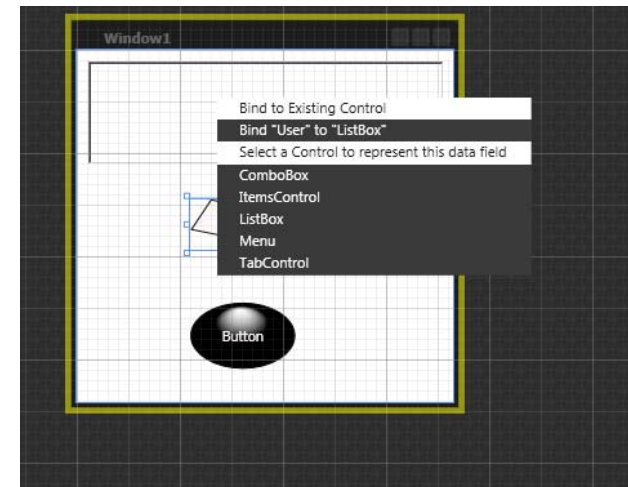


- Notice how this added a DataSource to our project.
- We will of course use this do drag & drop DataBinding like we do in Windows Forms.



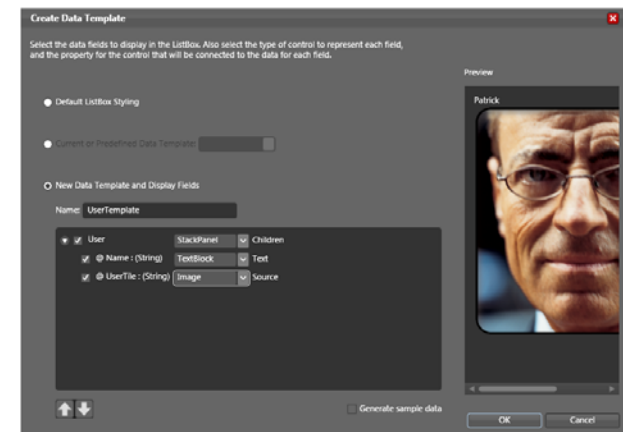
88. Click on the User (Array) collection in Data section, and drag it into the ListBox
89. Select Bind "user" to ListBox"

- Now we cheat a little and bring a canned XML file that has the list of users and their images. This of course could have come from a database or CLR business object. In this case an XML file is portable for a demo.



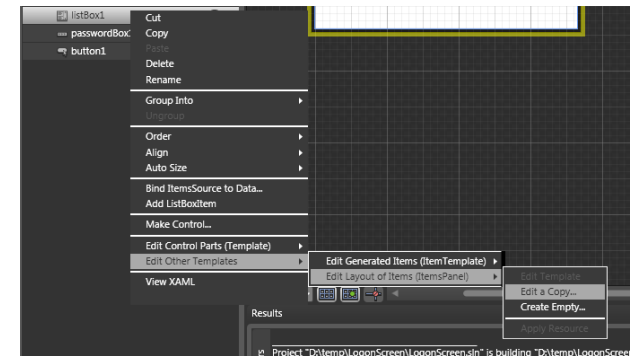
90. Click on the User (Array) collection in Data section, and drag it into the ListBox
91. Select Bind "user" to ListBox"
92. Accept the Select Field "ItemsSource"
93. Click OK
94. The data template window comes up, Change the UserTitle from TextBox to Image and make sure it binds to property Source
95. Click OK

- Now we cheat a little and bring a canned XML file that has the list of users and their images. This of course could have come from a database or CLR business object. In this case an XML file is portable for a demo.
- ItemsSource is a special property in ListBox used for databinding.
- Here we are defining a DataTemplate each item in the Users array will be expanded to this template.
- Let's change UserTitle to Image so we can see the picture.
- Notice the nice preview here in the RHS.
- Click OK to accept the template.



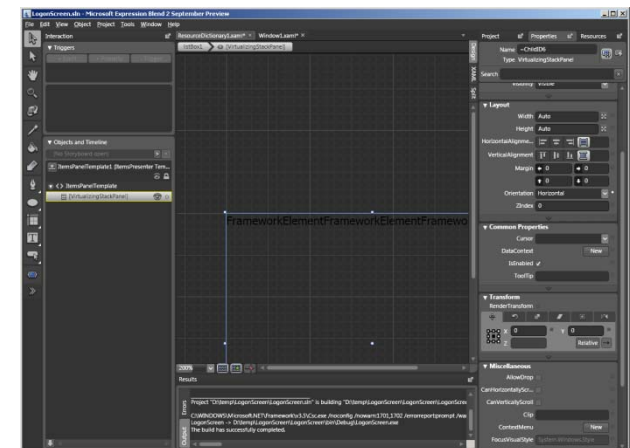
96. On object tree, select the ListBox,
97. Right click to select “Edit Other template”
98. Select “Edit Layout (ItemsPanel)”
99. Select “Edit copy”
100. On new dialog, Click “Resource Dictionary” to add this template in our ResourceDictionary

- Notice our ListBox is vertical, but we need it to be horizontal. Let’s again edit a template for a ListBox so we can change the orientation of the ItemsPanel in the ListBox.



101. On object tree, select the VirtualizingStackPanel.
102. On properties window, in Layout group, change orientation to horizontal.
103. Click the “back to scope” button to finish editing the layout.

- Here you can see the richness we get with templates and composition in WPF. The ListBox has a StackPanel to do its layout. If we just change the Panel’s orientation to Horizontal, we will be set to go.



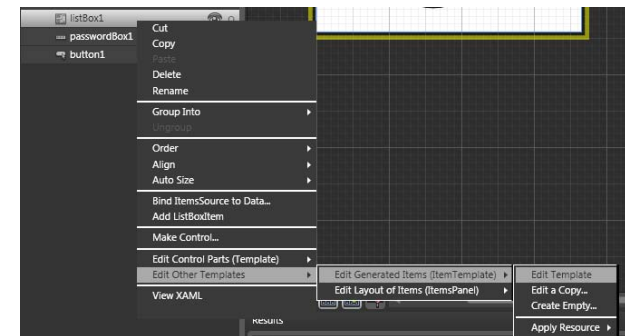
104. Right Click on the ListBox.

105. Select “Edit other templates”

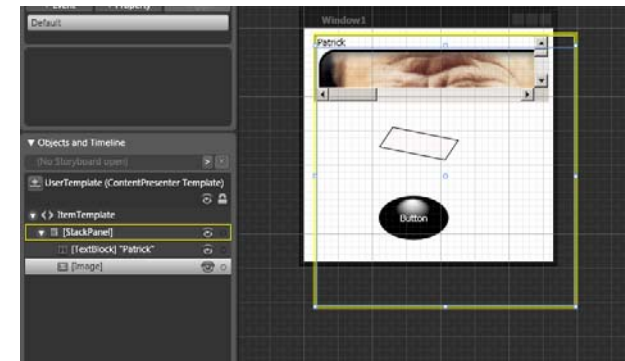
106. Select “Edit Generated Items”

107. Select “Edit Template”

- OK we have a horizontal ListBox, but our images are still too big.
- Let’s edit the data template to change the Image’s width to something reasonable like 100.



- This is the data template we created. Has a Stack Panel with Text and an Image.

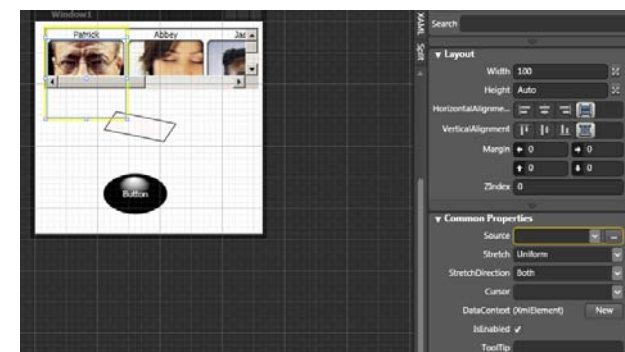


108. On the data template, Click on the Image

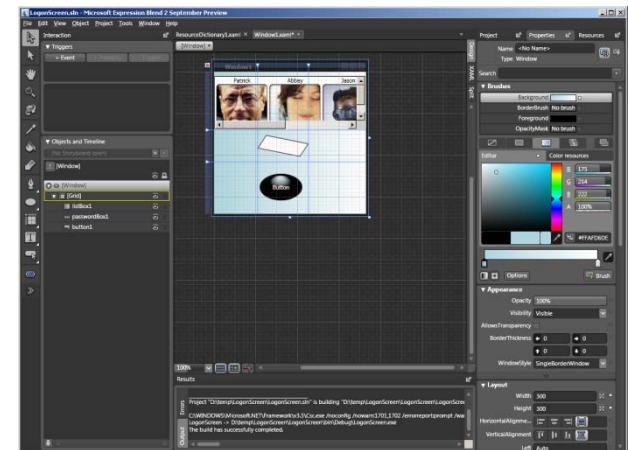
109. Go to properties window, set Width=100

110. Click the “back to scope” button to finish editing the template.

- Let’s center the text and let’s give the image a width of say 100.
- By giving it a width, the image will also resize height, because the stretch is uniform.

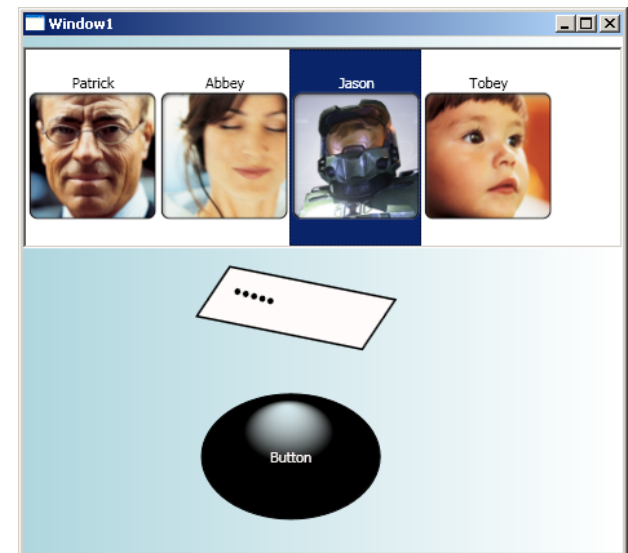


- OK our app is not perfect but I think it is complete enough.
- A couple of minor touches we could do here is add a background to the window maybe a light bluish.
- We should also check that our controls have no margins so this thing resizes nicely since we have not done a lot of static sizing.



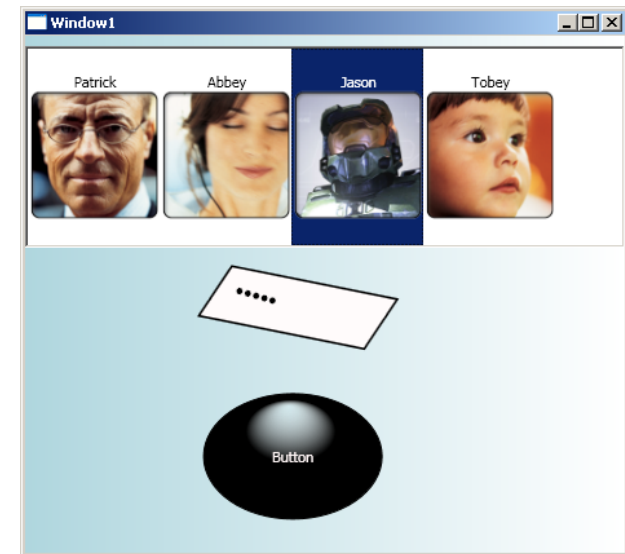
111. Select Project - > Test solution.

- We are now ready to run the app.
- Let's launch it.
- We resize it so everything fits nicely.
- We click on the button, notice it works.
- We enter on the password box it works too.
- We change selection in the ListBox. The whole thing is functional.



112. Back to Visual Studio

- We are now ready to run the app.
- Let's launch it.
- We resize it so everything fits nicely.
- We click on the button, notice it works.
- We enter on the password box it works too.
- We change selection in the ListBox the whole thing is functional.



113. Select Project - > Test solution.

- We still have a bit of work to do we could style the ListBox further add the other buttons, etc. but I think we have gotten the points across so let's summarize.
- We used Visual Studio 2008 for basic layout of our app, we wired the code, we could use debugger, etc. A developer would feel very comfortable there.
- We then use blend to style, build templates, etc. A designer would feel familiar inside Blend.
- These two products shared the project files we were always working in the same files. As we go back and forth, each product

picks the changes.

- We wrote a data driven heavily customized app and we only had to write one line of code, which had nothing to do with the app.