

# The Silverlight 2 Tutorials

## #5 User Controls

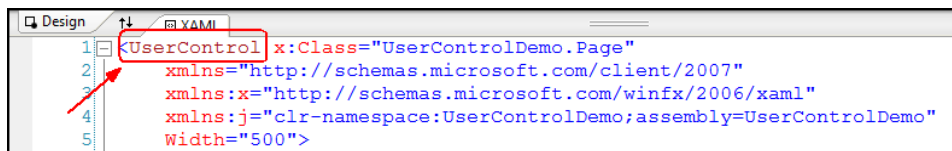
By Jesse Liberty

### User Controls

The best way to think about UserControl is as modules of XAML and code that you can easily re-use throughout your program. That is, they represent a simple form of custom control through aggregation of existing controls and code, much like User Controls in ASP.NET (pagelets).

Key to this idea is that a UserControl is created as a *composite of existing elements*. The pattern is to add a Panel to the UserControl's *Content* property, and then to add additional controls to the Panel's children collection.

In Silverlight 2, UserControls are ubiquitous as they are the containing control generated by Visual Studio 2008 for every page



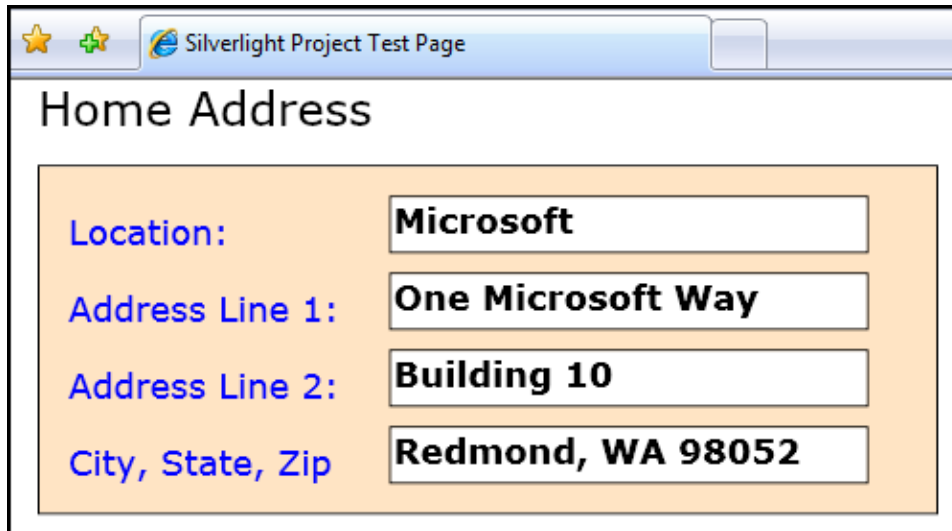
```
1 <UserControl x:Class="UserControlDemo.Page"
2   xmlns="http://schemas.microsoft.com/client/2007"
3   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
4   xmlns:j="clr-namespace:UserControlDemo;assembly=UserControlDemo"
5   Width="500">
```

### Creating UserControls

The simplest way to create a UserControl is to start with working code, scoop it out of your program and put it into its own file using the Silverlight User Control template as we'll see shortly

### Start by Creating Something Worth Reusing

The key to reusability is having something worth reusing. In her Mix 08 presentation [Karen Corby](#) showed reusability based on a bit of Silverlight that uses keyboard shortcuts to fill in an address form (press Control-M and Microsoft's address is entered).



---

This tutorial is created based on her example, with her permission and encouragement.

---

## Creating the Keyboard Project First

This project will assume you've worked your way through the previous tutorials and thus will progress quickly through

- Creating an application
- Adding controls and adding event handlers
- Using styles

So, to get started, fire up Visual Studio 2008 and create a new application called KeyboardFun.

The image above shows a text block with Home Address, and then what amounts to a grid with a border and prompts on the left and text boxes on the right. That suggests that the First control will be a stack panel so that we can easily stack one control on top of another, and within that stack panel a Grid.

```
<UserControl x:Class="KeyboardFun.Page"
  xmlns="http://schemas.microsoft.com/client/2007"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Width="500" Height="500">

  <StackPanel Background="White">
    <TextBlock Text="Home Address" FontFamily="Verdana" FontSize="24"
      HorizontalAlignment="Left" Margin="15,0,0,0"/>

    <Grid x:Name="AddressGrid" >
      <!--More Here-->
    </Grid>
  </StackPanel>
</UserControl>
```

```
    </StackPanel>
</UserControl>
```

Set the background color for the grid to a soft color (I personally like Bisque). While you're at it, surround your Grid with a Black Border. Give your grid 5 rows plus top and borders of 10 and two columns with left and right borders of 10 and a center padding of 10, as shown,

```
<Border BorderBrush="Black" BorderThickness="1" Margin="15">
<Grid x:Name="AddressGrid" Background="Bisque" >
    <Grid.RowDefinitions>
        <RowDefinition Height="10" />
        <RowDefinition Height="auto" />
        <RowDefinition Height="auto" />
        <RowDefinition Height="auto" />
        <RowDefinition Height="auto" />
        <RowDefinition Height="auto" />
        <RowDefinition Height="10" />
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="10" />
        <ColumnDefinition Width="auto" />
        <ColumnDefinition Width="10" />
        <ColumnDefinition Width="auto"/>
        <ColumnDefinition Width="10" />
    </Grid.ColumnDefinitions>
    <!--More Here-->
</Grid>
</Border>
```

## Setting the Styles for the Controls

The grid will be filled with four prompts and four text boxes, but a quick look at the earlier diagram shows that these are styled, as described in the Tutorial [Styles and Templates](#).

To make them available to all pages in the application, the styles will be defined in App.xaml. There are only two styles, as the form is quite simple; one style for the TextBlock to hold the prompt, and one for the TextBox to hold the value.

```
<Application.Resources>
    <Style TargetType="TextBlock" x:Key="TextBlockPrompt">
        <Setter Property="VerticalAlignment" Value="Bottom" />
        <Setter Property="HorizontalAlignment" Value="Left" />
        <Setter Property="FontFamily" Value="Verdana" />
        <Setter Property="FontSize" Value="18" />
        <Setter Property="FontWeight" Value="Medium" />
        <Setter Property="Foreground" Value="Blue" />
        <Setter Property="Margin" Value="5" />
    </Style>

    <Style TargetType="TextBox" x:Key="TextBoxStyle">
        <Setter Property="FontFamily" Value="Verdana" />
        <Setter Property="FontSize" Value="18" />
        <Setter Property="FontWeight" Value="Bold" />
    </Style>
</Application.Resources>
```

```

        <Setter Property="Foreground" Value="Black" />
        <Setter Property="VerticalAlignment" Value="Bottom" />
        <Setter Property="HorizontalAlignment" Value="Left" />
        <Setter Property="Width" Value="250" />
        <Setter Property="Height" Value="30" />
        <Setter Property="Margin" Value="5" />
    </Style>
</Application.Resources>

```

## Creating the Controls in the Grid

We can begin easily enough by adding the prompt for the location (using the TextBlockPrompt style) and the corresponding TextBox using the TextBox style.

```

<TextBlock Text="Location: "
    Style="{StaticResource TextBlockPrompt}"
    Grid.Row="2" Grid.Column="1" />

<TextBox x:Name="Location"
    Style="{StaticResource TextBoxStyle}"
    Text="Silverlight Central"
    Grid.Row="2" Grid.Column="3" />

```

They make a matched set, and we can do the same for the other three controls as well.

```

<TextBox x:Name="Address1"
    Style="{StaticResource TextBoxStyle}"
    Text="100 Main Street"
    Grid.Row="3" Grid.Column="3" />

<TextBlock Text="Address Line 2: "
    Style="{StaticResource TextBlockPrompt}"
    Grid.Row="4" Grid.Column="1" />

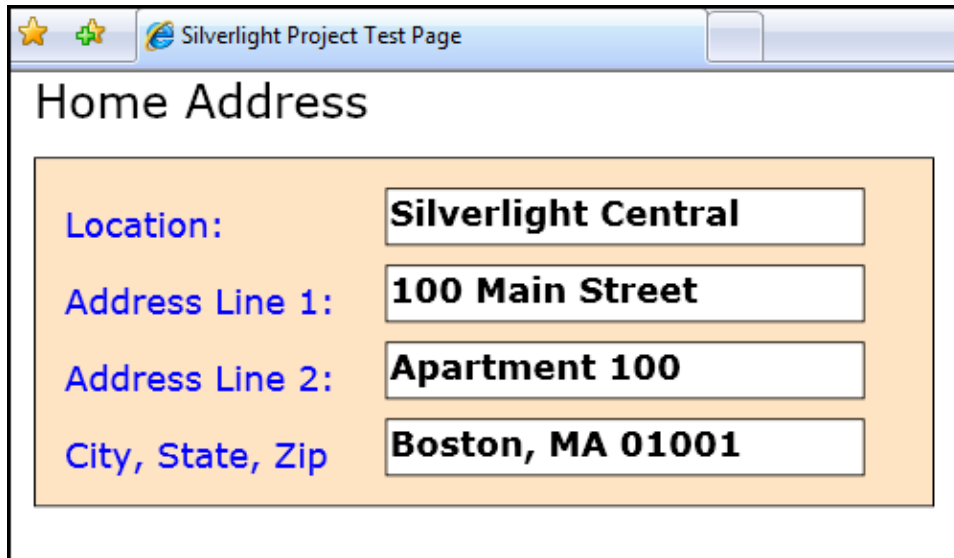
<TextBox x:Name="Address2"
    Style="{StaticResource TextBoxStyle}"
    Text="Apartment 100"
    Grid.Row="4" Grid.Column="3" />

<TextBlock Text="City, State, Zip "
    Style="{StaticResource TextBlockPrompt}"
    Grid.Row="5" Grid.Column="1"/>

<TextBox x:Name="City"
    Style="{StaticResource TextBoxStyle}"
    Text="Boston, MA 01001"
    Grid.Row="5" Grid.Column="3"/>

```

Notice that the Text for each of the TextBox controls is hard coded and displayed as coded when you run the application,



It is fair to say that as written, this application is of limited value.

## Two Way Data Binding

As discussed in the [Tutorial on Data Binding](#) one powerful way to bind the data that will fill the address to data that you might persist in, for example, a database or an XML file or any other storage, is by using a business object to mediate between your User Interface (the Silverlight application) and the storage.

Let's create an address object that will map to the data we wish to display (though a typical business object will probably not be so tightly coupled to a particular UI page). We will do with address exactly what we did with the book object in that tutorial. Rather than making you wade through all that code, I'll simply provide the Properties and their underlying member variables in the next table:

| Private Member Variable | Public Property |
|-------------------------|-----------------|
| location                | Location        |
| address1                | Address1        |
| address2                | Address2        |
| city                    | City            |

The class structure is unchanged from the example shown in the previous tutorial,

```

using System.Collections.Generic;
using System.ComponentModel;

namespace KeyboardFun
{

```

```
public class Address : INotifyPropertyChanged
{
```

Once you've filled in all the public properties you are ready to replace the hard coded Text Block values with the data bound values,

```
<TextBox x:Name="Location"
Style="{StaticResource TextBoxStyle}"
Text="{Binding Location, Mode=TwoWay}"
Grid.Row="2" Grid.Column="3" />

<TextBox x:Name="Address1"
Style="{StaticResource TextBoxStyle}"
Text="{Binding Address1, Mode=TwoWay}"
Grid.Row="3" Grid.Column="3" />

<TextBox x:Name="Address2"
Style="{StaticResource TextBoxStyle}"
Text="{Binding Address2, Mode=TwoWay}"
Grid.Row="4" Grid.Column="3" />

<TextBox x:Name="City"
Style="{StaticResource TextBoxStyle}"
Text="{Binding City, Mode=TwoWay}"
Grid.Row="5" Grid.Column="3"/>
```

## And Now For Something Completely Different

Until now we've essentially been reviewing styles and data binding. But remember, all of this is in service to creating a UI and code behind it worth reusing. Let's review what we have: a "form" that prompts for address information, and an Address object that can either hold or provide that information.

Note that the "form" is implemented as the contents of the Grid whose name is "Address" and which contains the four TextBlocks and the four TextBoxes.

What we'd like to add is the ability for the form itself to respond to two keyboard shortcuts. When we press Control-M we'd like the form to be filled in with the address information for Microsoft, and when we press Control-C we'd like to fill in the address of the [Computer History Museum](#)



## Keyboard Events

To accomplish this, we'll respond to keyboard events received by the stack panel: specifically the `KeyDown` event.

The Silverlight Help files document that the `KeyDown` event is a bubbling event (see [Tutorial #1 on Controls](#) for a full explanation of bubbling) if a keydown is received by any control within the grid, that event will bubble up to the grid and can be responded to in the Grid's `KeyDown` event handler.

## First Get It Working

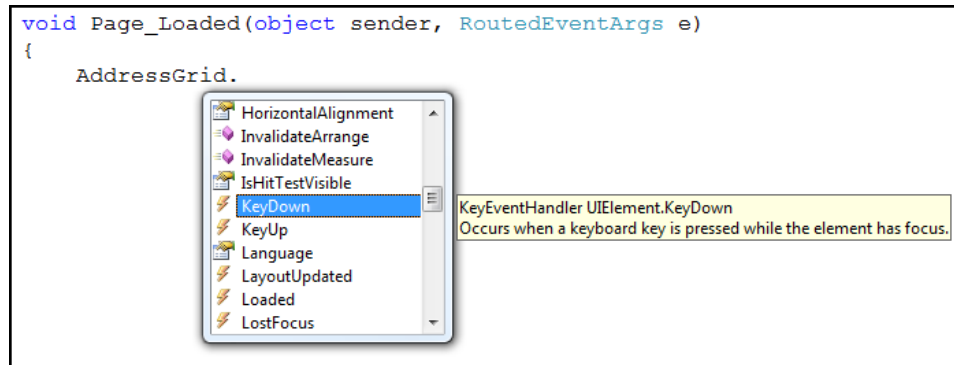
We can start simple and respond to *any* keyboard event by filling the form with the Microsoft address. We'll give the class a private `Address` object which we'll allocate memory for in the constructor. We'll also handle the standard `Loaded` event that fires when the page is loaded.

```
public partial class Page : UserControl
{
    private Address theAddress;
    public Page()
    {
        InitializeComponent();
        theAddress = new Address();
        Loaded += new RoutedEventHandler(Page_Loaded);
    }
}
```

In `Page_Loaded` we want to create an event handler for `KeyDown` when any keydown event is registered on any control within the grid, which you'll remember we named `AddressGrid` in `Page.xaml` like this,

```
<Grid x:Name="AddressGrid" Background="Bisque" >
```

When Page.xaml is saved, that identifier is immediately available in the code behind, and we can access its properties and events, including its KeyDown event,



```
void Page_Loaded(object sender, RoutedEventArgs e)
{
    AddressGrid.KeyDown += new KeyEventHandler(AddressGrid_KeyDown);
}
```

Visual Studio 2008 will offer to create the shell of an event handler for us, which is just what we want. For now, we'll respond to the KeyDown indiscriminately; whatever key is pressed, we'll fill in Microsoft's address.

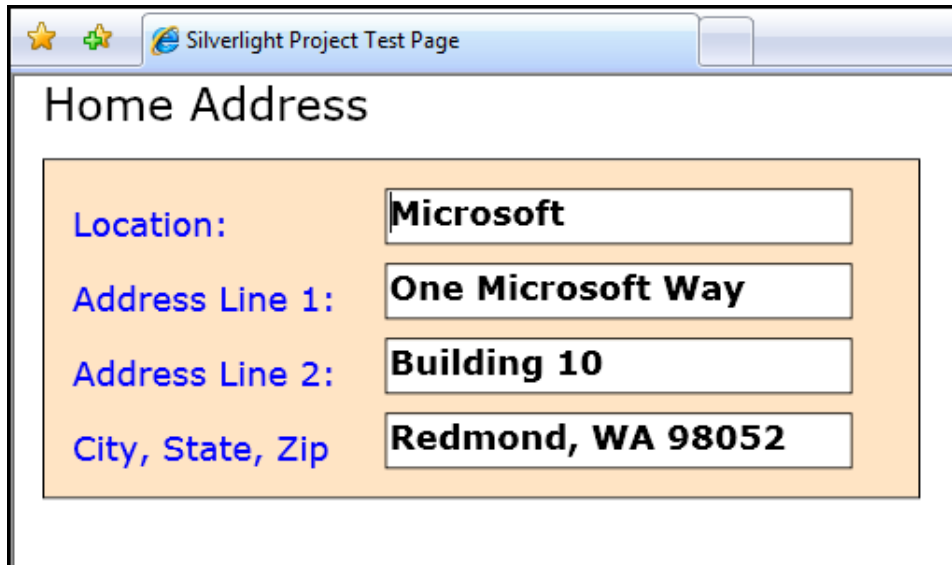
This approach is called by various names, depending on how you feel about it:

- Get it working and keep it working
- Successive approximation
- Wasting time on work you'll just throw away

I'm a big believer in successive approximation, by changing fewer things between test runs, I have fewer things to look at when it doesn't work.

```
void AddressGrid_KeyDown(object sender, KeyEventArgs e)
{
    theAddress.Location = "Microsoft";
    theAddress.Address1 = "One Microsoft Way";
    theAddress.Address2 = "Building 10";
    theAddress.City = "Redmond, WA 98052";
    this.DataContext = theAddress;
}
```

Note that we end the event by setting the DataContext of the page to the Address object we just populated. That has the effect of providing a specific object for the controls to bind to. When you run this application nothing happens. Click inside any of the four text boxes, however, and then press any key at all and the address is filled in. The cleanest way to do this is to press the shift key!



## Discriminating Among Keys

Clearly this isn't quite cooked. We want to fill in the Microsoft Address on Control-M, the Computer Museum's address on Control-C. A small revision to the event handler is needed,

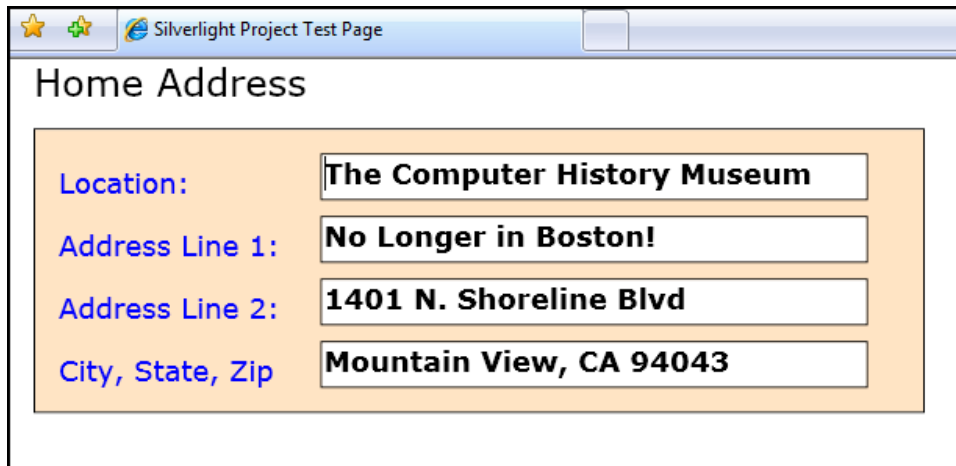
```
void AddressGrid_KeyDown(object sender, KeyEventArgs e)
{
    if (e.Key == Key.C && Keyboard.Modifiers == ModifierKeys.Control)
    {
        theAddress.Location = "The Computer History Museum ";
        theAddress.Address1 = "No Longer in Boston!";
        theAddress.Address2 = "1401 N. Shoreline Blvd";
        theAddress.City = "Mountain View, CA 94043";
    }
    if (e.Key == Key.M && Keyboard.Modifiers == ModifierKeys.Control)
    {
        theAddress.Location = "Microsoft";
        theAddress.Address1 = "One Microsoft Way";
        theAddress.Address2 = "Building 10";
        theAddress.City = "Redmond, WA 98052";
    }
    this.DataContext = theAddress;
}
```

A quick fix to the width of the outermost user control in Page.xaml (changing Width from 500 to 600) and to the Width Setter Property in the TextBox Style in App.xaml, (changing the value from 250 to 350) and we have room for "The Computer History Museum"

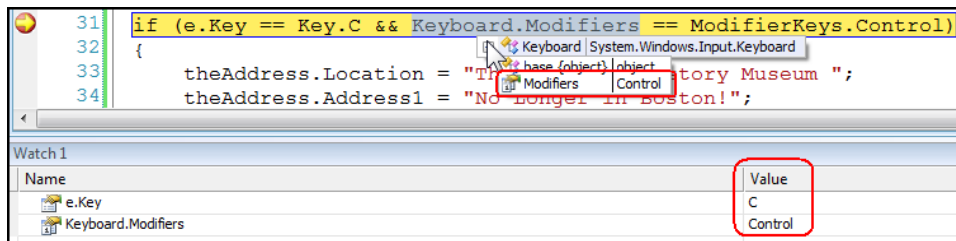
```
<Style TargetType="TextBox" x:Key="TextBoxStyle">
    <Setter Property="Width" Value="350" />
```

| </Style>

Bring up the application, anywhere in the form, and press Control-C; then Control-M and then Control-C. Fun, eh?



By setting a break point where the key is examined, you can actually watch the test for the Control-C combination; either hovering over the values, or looking at the Watch (or QuickWatch) window,



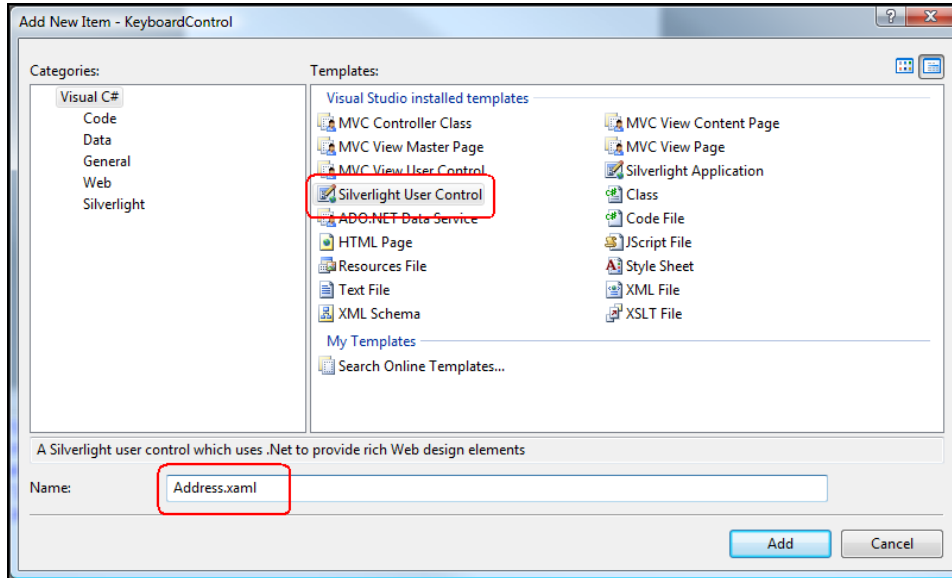
## User Control

This is, arguably, useful. In fact, we might want to have a Home Address, a Work Address and a Billing Address, and it might be nice to have the control-key work for all three. Thus, we have two choices.

1. Duplicate the xaml and code behind for each instance (home, work, billing)
2. Factor out the common xaml and supporting code into a control

I won't waste your time hammering on why we'll choose #2; if you've come this far you have a visceral revulsion at the idea of choice #1 and are eager to get on to seeing how you create the UserControl.

You begin, no surprise, by right clicking on the project and choosing Add. Within the templates section of the dialog box choose Silverlight User Control. Name your new control Address.xaml



Two files are immediately added to your project

- Address.xaml
- Address.xaml.cs

Address.xaml will look amazingly familiar when you first open it.

```
<UserControl x:Class="KeyboardFun.Address"
  xmlns="http://schemas.microsoft.com/client/2007"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Width="400" Height="300">
  <Grid x:Name="LayoutRoot" Background="White">

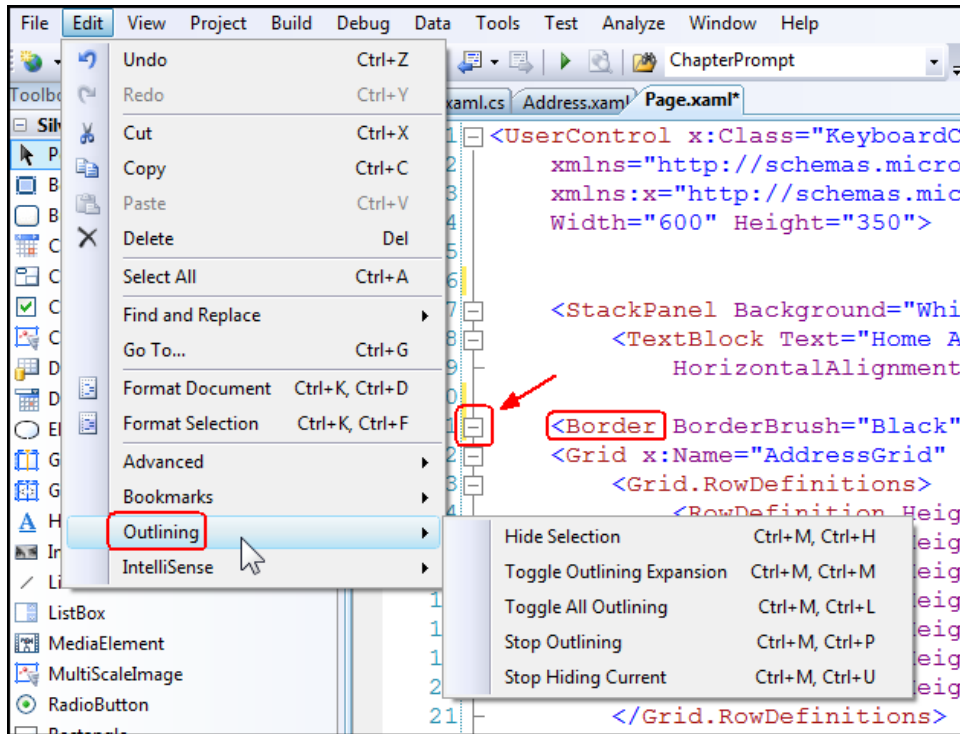
    </Grid>
</UserControl>
```

It is, essentially, the same as Page.xaml, except the names have been changed to protect the innocent namespace.

## Implementation

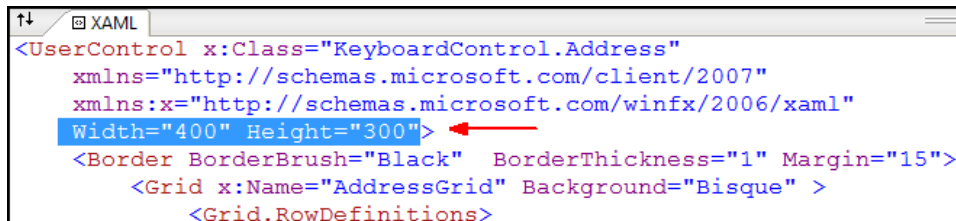
The rest, I'm afraid, is mechanical.

Return to Page.xaml and scoop out the Border and everything within it. The simplest way to do this is to collapse the Border control (if outlining mode is off in Visual Studio 2008 you can easily turn it on from the Edit menu when the design or code window has focus,



Collapse the Border and then use Control-X to cut it from its current position, and Control-V to paste it into the new .xaml file, **replacing the grid that was placed there by Visual Studio 2008**

Notice that your control is in place but a bit narrow. Delete the Width and Height attributes from the outer User Control,



As soon as you delete these the control will be centered and it will expand to fit the controls,

|                  |                      |
|------------------|----------------------|
| Location:        | <input type="text"/> |
| Address Line 1:  | <input type="text"/> |
| Address Line 2:  | <input type="text"/> |
| City, State, Zip | <input type="text"/> |

## Adding Code

Once again, the is entirely mechanical. All the following steps occur in AddressUserControl.xaml.cs:

1. Add the Address member variable as you did earlier
2. In the constructor allocate memory for the Address object and set the Loaded event handler
3. In the Page\_Loaded implementation add an event handler for the KeyDown event on the grid

```
public partial class AddressUserControl : UserControl
{
    private Address theAddress = null;
    public AddressUserControl()
    {
        InitializeComponent();
        theAddress = new Address();
        Loaded += new RoutedEventHandler(Page_Loaded);
    }

    void Page_Loaded(object sender, RoutedEventArgs e)
    {
        AddressGrid.KeyDown += new KeyEventHandler(AddressGrid_KeyDown);
    }
}
```

4. The implementation of the AddressGrid.KeyDown is cut and paste out of Page.xaml.cs,

```
void AddressGrid_KeyDown(object sender, KeyEventArgs e)
{
    if (e.Key == Key.C && Keyboard.Modifiers == ModifierKeys.Control)
    {
        theAddress.Location = "The Computer History Museum ";
        theAddress.Address1 = "No Longer in Boston!";
        theAddress.Address2 = "1401 N. Shoreline Blvd";
        theAddress.City = "Mountain View, CA 94043";
    }
}
```

```

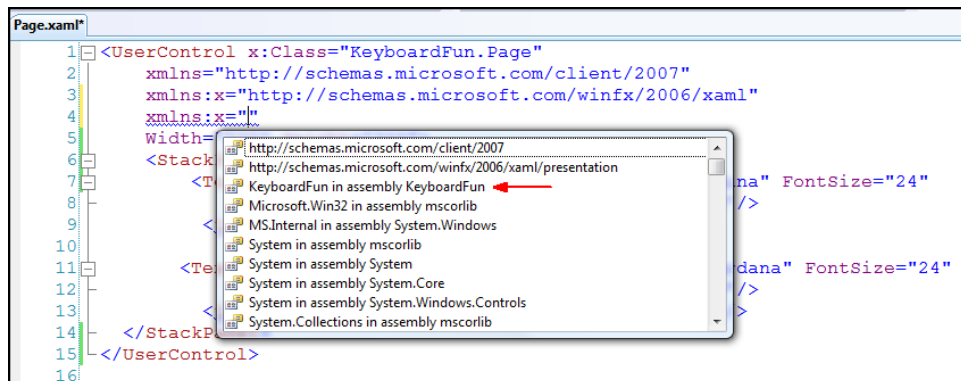
    }
    if (e.Key == Key.M && Keyboard.Modifiers == ModifierKeys.Control)
    {
        theAddress.Location = "Microsoft";
        theAddress.Address1 = "One Microsoft Way";
        theAddress.Address2 = "Building 10";
        theAddress.City = "Redmond, WA 98052";
    }
    this.DataContext = theAddress;
} // end KeyDown event handler
} // end class AddressUserControl

```

## Using The User Control

That's it for creating the User Control, but it isn't much use if you don't place it back into your Page.xaml file. Here's how.

1. Save all your files
2. At the top of Page.xaml add a namespace for your control, with a prefix of your choosing (I'll use jl). Intellisense is incredibly eager to help.



3. Remember to add the assembly as well, so that the entire statement looks like this,

```
xmlns:jl="clr-namespace:KeyboardFun; assembly=KeyboardFun"
```

4. You are now ready to substitute the User control for all the contents of the Border in Page.xaml. If you've not deleted Border already do so now, and replace with your control, just as you might place any other control into the StackPanel,

```
<jl:AddressUserControl x:Name="HomeAddress" />
```

## Reuse, reuse, reuse...

Let's add one more of our user controls to the StackPanel and change the prompt on the first, so that it looks like this,

```

<UserControl x:Class="KeyboardFun.Page"
  xmlns="http://schemas.microsoft.com/client/2007"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

```

```

xmlns:jl="clr-namespace:KeyboardFun;assembly=KeyboardFun"
Width="500" Height="500">
<StackPanel Background="White">

    <TextBlock Text="Event Address" FontFamily="Verdana" FontSize="24"
        HorizontalAlignment="Left" Margin="15,0,0,0"/>
    <jl:AddressUserControl x:Name="HomeAddress" />

    <TextBlock Text="Billing Address" FontFamily="Verdana" FontSize="24"
        HorizontalAlignment="Left" Margin="15,0,0,0"/>
    <jl:AddressUserControl x:Name="BillingAddress" />

</StackPanel>
</UserControl>

```

Note that adding a second instance of the AddressUserControl requires only making sure they have different names. Before you run the program, take a peek at Page.xaml.cs,

```

using System.Windows.Controls;

namespace KeyboardFun
{
    public partial class Page : UserControl
    {
        public Page()
        {
            InitializeComponent();
        }
    }
}

```

All the logic has been exported and encapsulated in the User Control. You can add 2 (or 20) Address User Controls to your UI and add no code whatsoever. Yet, when you run the program, each works independently, and each supports the Control-keys as implemented,

The screenshot shows a web browser window with the title 'Silverlight Project Test Page'. The page content is divided into two sections: 'Event Address' and 'Billing Address'. Each section contains a form with four input fields. The 'Event Address' form has the following values: Location: 'The Computer History Museum', Address Line 1: 'No Longer in Boston!', Address Line 2: '1401 N. Shoreline Blvd', and City, State, Zip: 'Mountain View, CA 94043'. The 'Billing Address' form has the following values: Location: 'Microsoft', Address Line 1: 'One Microsoft Way', Address Line 2: 'Building 10', and City, State, Zip: 'Redmond, WA 98052'.

| Event Address    |                             |
|------------------|-----------------------------|
| Location:        | The Computer History Museum |
| Address Line 1:  | No Longer in Boston!        |
| Address Line 2:  | 1401 N. Shoreline Blvd      |
| City, State, Zip | Mountain View, CA 94043     |

| Billing Address  |                   |
|------------------|-------------------|
| Location:        | Microsoft         |
| Address Line 1:  | One Microsoft Way |
| Address Line 2:  | Building 10       |
| City, State, Zip | Redmond, WA 98052 |

## What Have You Learned Dorothy?

One thing we've learned is that Karen Corby is a rising star at Microsoft; and that I'm not above taking her [brilliant demo](#) from Mix and decomposing it into a tutorial! Keep an eye on her [blog](#); she has a lot to say and it is always worth hearing. And she is an incredibly kind person.

A second thing we've learned, is that if you don't need a full blown custom control (ready to ship to customers), refactoring UI and its supporting code into User Controls is, once you've done it, almost trivial; and incredibly convenient, and makes for much more scalable code.

I will follow with a tutorial on Custom Controls, but there are a few more fundamental topics I want to be sure to cover first.

Note, the source code for this tutorial comes in three parts:

1. KeyboardStarter starts you off with a grid and the basic controls and styles you already know about
2. KeyboardControl is the interim application with the KeyDown event handler

3. UserControlDemo is the final product with the KeyDown event handling logic factored out into a User control.