

OmahaMTG July Newsletter

July 28, 2010

Inside This Issue

- 1-2** An ASP.NET Comet Implementation
- 3-4** Silverlight an RIA Power Tool
- 5** New Features in Silverlight 4
- 5-9** UI Updates With Databinding and INotifyPropertyChanged
- 10** A Message From The Editor

An ASP.NET Comet Implementation

By Kyle Kolander

Web applications employ an HTTP request / response model. The browser requests a resource from the server and the server responds to the browser by supplying the requested resource. But what happens when you want to update client browsers as a result of something changing on the server (e.g., an updated stock price or a new chat message)? The browser doesn't know what's happening on the server. How do you immediately tell the browser about this new event?

That's where Comet comes into play. According to Wikipedia, Comet describes a web application model in which a long-held HTTP request allows a web server to push data to a browser, without the browser explicitly requesting it. A Comet implementation will most likely employ the Javascript XMLHttpRequest object, since Javascript is supported by all major browsers.

If you've used the ASP.NET AJAX UpdatePanel and Timer controls, then you've likely achieved Comet-like functionality. In this polling model, the Timer will periodically tick, which will fire off a server-side event asking the server if anything has changed. Regardless of whether the server has any new information for the client, it will respond by rendering a partial page update. Sometimes the UpdatePanel contents will change after this tick event, but most of the time the request / response cycle of the Timer's tick event will be wasteful as nothing of note will have happened on the server. What's worse is that as the number of clients increases, the server will find itself inundated with frequent and unnecessary requests.

To provide an analogy, imagine yourself driving on a family vacation and every two seconds each of your kids asks you "Are we there yet?" Wouldn't it be nice if at the start of the trip your kids each let you know that they'll wait and listen for you to tell them when you arrive at your destination? That's what we can achieve with an ASP.NET Comet implementation.

Let's take a look at a simple chat application in which one person (Kyle) chats with another person (Jake), both of whom are logged in and request the ChatApp.aspx page. On this page there is a multiline TextBox (ID = DisplayMessagesText) to display all of the chat messages sent back and forth along with a TextBox to enter the message and a Button to send the message. If Kyle sends a message to Jake, Jake should see the message instantly and only Jake should see it – no other users should see this message. We'll put the enter message TextBox and the send Button inside of an UpdatePanel in order to prevent a full page refresh whenever a user sends a message.

By clicking the send Button, we're telling the server to push our message out to Jake's browser and render it inside of his DisplayMessagesText TextBox. We can handle the send Button's Click event, but how can we actually push the message out to Jake's browser? If only there was a function we could call that would do this for us. As it turns out, there is... sort of. First we need to do a little more work to make the function available to us in the Button's Click event handler.

Earlier I alluded to most Comet implementations using the Javascript XMLHttpRequest object and this implementation is no different. For those of you unfamiliar with XMLHttpRequest, it's the mechanism by which the UpdatePanel allows for partial page updates. Basically you create an instance of the XMLHttpRequest object, use the object to send a message to the server via a URL, receive and interpret the XML response, and modify the HTML DOM accordingly. In order to do that, we need to register a couple javascript functions with the ChatApp.aspx page.



In the page's Load event handler, we'll build our Javascript string and register it using the ClientScript.RegisterStartupScript function. We aim to accomplish the following:

- 1.) Define CreateServerRequest function
 - a. Create the XmlHttpRequest object
 - b. Handle the XmlHttpRequest.OnReadyStateChange event
 - i. Confirm that request object's response is complete and successful
 - ii. Parse the response message
 - iii. Modify the HTML DOM according to the message contents
 - iv. Call the ExecuteServerRequest function
 - c. Open the request connection and send the message
- 2.) Define ExecuteServerRequest function
 - a. Call the CreateServerRequest function
- 3.) Call the ExecuteServerRequest function

The most important thing to take away from the above steps is that we are essentially creating a recursive cycle of asking the server to tell us if anything important happens, waiting for the response, and updating the browser accordingly. It is through this approach that we are able to simulate an always-on open connection similar to using sockets in a client / server application. The part we still need to explore is step 1.c above. We need to open a connection – but to what? The key is an asynchronous HTTP handler.

To create an asynchronous HTTP handler in ASP.NET, we need to implement the IHttpAsyncHandler interface and configure a mapping between an ASHX URL and our handler class in our web.config file. To create our class, we focus on the BeginProcessRequest and EndProcessRequest methods. When the Javascript XmlHttpRequest object makes its connection with the server, it does so via the ASHX URL, which causes IIS / ASP.NET infrastructure to invoke the BeginProcessRequest method. Once the EndProcessRequest method is called, the response will be sent back to the XmlHttpRequest object.

The BeginProcessRequest method is provided with the HttpContext and an AsyncCallback, which is how we invoke the EndProcessRequest method and send the response. At this point, we need to somehow invoke Jake's AsyncCallback when Kyle sends his message.

There are a few approaches that will allow for this to happen, but the choice of which approach to use really depends on the requirements of the application. For simplicity, we'll look at using the HttpApplicationState object exposed via the HttpContext.Application property (you could accomplish nearly the same thing with the Cache object). A more robust and scalable, but also more complex, solution could involve a shared polling thread in the handler that would continually query a database that gets updated with every message that is sent. Regardless, we need to somehow hold on to each AsyncCallback and invoke it whenever a new message arrives.

That brings us to the next question of uniqueness. We need to somehow distinguish one AsyncCallback from another, so that when a message from Kyle arrives for Jake, we'll invoke the correct callback, and the message will be routed accordingly. Again, this is largely an implementation detail. Regardless of approach, we need to ensure the Javascript request provides the unique information to the asynchronous handler. The easiest way to do this is through the use of a URL query string.

One implementation would be to ensure each DisplayMessagesText TextBox has a unique ClientID (e.g. "DisplayMessagesText_Kyle_Jake" to signify only messages from Kyle and to Jake). We can accomplish this by renaming the control in the page Load event handler. The Javascript URL must include the ClientID in the query string (e.g. "http://localhost/ChatApp.ashx?DisplayMessagesText_Kyle_Jake"). The BeginProcessRequest method would then somehow track and associate that unique key with the supplied AsyncCallback. In our simple example, we'll just store it in the application state. This means that the callback will be exposed to the send button's click event handler. Now it is only a matter of constructing the unique key string in order to retrieve the callback and send the message.

Comet can provide a great approach for pushing data to one or more client browsers. An ASP.NET application can make use of the Javascript XmlHttpRequest object and an asynchronous HTTP handler in order to achieve a Comet implementation. If carefully designed, this approach can provide a better user experience and a more efficient implementation.

About The Author

Kyle Kolander

Software Engineer

Kyle has ten years of software development experience and is currently a software engineer performing application development for LeaseTeam. He earned his Masters degree in Computer Science from Colorado State University specializing in Artificial Intelligence. Kyle and his wife Danielle recently had their first child, William, and look forward to watching him grow up. Outside of work and family, it has been a frustrating summer watching the Chicago Cubs consistently lose, but this fall looks promising as long as Brett Favre returns to the Vikings and Ricky Stanzi doesn't throw too many interceptions for the Hawkeyes. I know we're in Nebraska, but I grew up in Iowa so GO HAWKEYES!

Silverlight - An RIA Power Tool

By Dustin Horne

Introduction

I wanted to kick off my blog with a different kind of Silverlight introduction. The internet is buzzing with news about the new kid on the block and chock full of comparisons to Flash/Flex. While many of these comparisons are valid, both platforms offer features and functionality that make them much more than just tools to make your website look pretty.

I want to primarily focus on the features outside of the realm of animation as these are the elements that make Silverlight a viable platform for building business-centric applications. This post is meant to give you a very broad overview of Silverlight. In future posts I will concentrate on specific features that make Silverlight a powerful business application platform and show you how to leverage those.

.NET – A Common Platform

What I see as one of Silverlight's biggest assets is that it leverages a version of the .NET framework. Microsoft's choice of .NET as a development platform for Silverlight seems like a no-brainer, but it was an extremely important decision for the future of the new RIA technology.

If you decided as I did to give Silverlight development a shot when it first hit the scene you may have been impressed at the power that was available for a first release, but at the same time disappointed at the overall lack of options for the development model.

Programming against the first version of Silverlight was purely javascript. Getting the application to run on the page was a bit cumbersome and the biggest benefit seemed to be the ability to interact with the page DOM. Animations were simple to create but as a whole it lacked the power of what we expect from a Microsoft technology.

Fortunately, Microsoft didn't waste any time pushing out a CLR driven version of Silverlight. While it presented a complete change in thought process over the javascript based version 1.0, it offered significant improvements in performance and power, as well as opening the platform to the enormous existing .NET developer base, giving the community at large the opportunity to quickly start creating simple RIA applications.

Collaborative Focus

Microsoft has never had a large draw in the design community. Silverlight has posed some challenges for the company as it tries to appeal to a completely new user base. The philosophy behind the development process for Silverlight encourages collaboration and team based development while not limiting developers to that model.

Assets can be easily created using Expression Design and exported as XAML to be used in the Silverlight application. UI Designers can mock up the user interface, create design elements, and even create behaviors directly in Expression Blend. Expression Blend projects are solutions that can be opened and coded in Visual Studio. You're not limited to these tools however. If a developer wanted to create the entire application in Visual Studio from start to finish it is certainly possible and with the improved design view tools of Visual Studio 2010 it's a painless process. Many developers are not designers however, making the focus on collaboration an important element of Silverlight development.

While I've touted the tools and collaborative abilities of Silverlight, I think it's only fair for me to point out a couple of places I think Microsoft can improve. Expression Design, as feature rich as it is, still is no Adobe Illustrator replacement. It does have uses beyond Silverlight development, however I find myself using it very infrequently. In addition, currently the Expression product line is strictly Microsoft Windows products. I am a PC user myself, however Macs are not only prominent, but also very powerful in the design community. If Microsoft is serious about attracting designers for Silverlight it needs to make a better effort of providing the tools to a wider range of operating systems (i.e. OS X).

About The Author

Dustin Horne
Software Engineer

Dustin has over 10 years of programming experience in Visual Basic. He specializes in dynamic and database driven web development with ASP .NET and Silverlight using VB .NET and C#. He also develops in PHP, Classic ASP, HTML, CSS, and Javascript; layout and design using Photoshop, Fireworks, and Dreamweaver; has database experience in Access, MySQL, MSSQL, and Firebird. In his free time (what's that?), Dustin enjoys playing Softball and Golf.

Under the Hood

Alright, so it's nice and shiny on the outside. It looks fast and slick but does it have any power? The answer is definitely yes. Microsoft has built Silverlight to run on a revamped version of the .NET Framework. I say revamped because they didn't just remove unnecessary full-framework features to make for a smaller runtime download, they also retooled the class libraries for optimal in and out of browser performance. Some libraries and items are Silverlight specific and some have been moved around a bit, but in general you'll find that many of the same concepts, classes and features in the full .NET Framework are just as familiar in Silverlight.

What I found most compelling about Silverlight from a business application perspective was the rich set of tools available for working directly with data. Leveraging the power of LINQ you can quickly and effectively work with data such as XML to produce rich and dynamic applications. Many of Silverlight's built-in controls support direct data binding and creating your own is by no means a difficult task.

A Practical Example

The best way to illustrate the above would be to provide you with an example. I was tasked with building a gallery for a local artist. A local design shop created the design and UI and all that was needed was to make it function. The specifications required that I not only use Silverlight's DeepZoom technology to create zoom-able high resolution images, but also give the artist the ability to manage her own. In addition to images, she needed the ability to provide titles, pricing and mark the items as sold as well as manage multiple galleries.

The first step was to build an administrative area where she could manage these items. I won't bore you with the details of this step, but it involved giving her the ability to create multiple galleries, upload very large photos, manage the display order of these images in the gallery, and provide additional information about each piece. The images are uploaded through a web form, image pyramids are created for the DeepZoom functionality, and XML files are generated / updated to store information about the galleries and images.

Using the power of LINQ, I was able to read the XML files and update Silverlight controls to reflect it, as well as dynamically updating the DeepZoom component with the current selected image once the thumbnails are clicked. If you'd like, you can check out the application at: www.janeteskrige.com/works.

RIA vs HTML 5... Oh No!

HTML 5 is on the horizon and will pose great challenges for RIA platforms moving forward. It will make many graphics and animation functions possible without the need for a browser plugin or separate RIA platform. I've chosen Silverlight as my client side RIA development platform because I feel that Microsoft's strategy is on track to keep the platform viable.

In all of the heated discussion and mud-slinging (from both camps to be perfectly fair) in the comparison of Flash and Silverlight, the majority of the comparisons seem to be centered around graphics and animation. Implementing 3D graphics and animation is much more powerful in Flash and many of the complaints are about features that are missing from Silverlight in this realm.

Take a look at the new feature set in Silverlight 4, however. While there were some graphics and 3D improvements, the majority of the changes and additions were centered around data driven and business logic functionality. In a world where HTML 5 will make many of the graphics features in both platforms much less relevant, Microsoft is beefing up the engine that drives Silverlight and improving the ability to do meaningful things with data, integrate with standards based web services and create powerful applications well beyond photo galleries and banner ads.

Conclusion

If you've made it this far, I applaud you. While I anticipated this post being much shorter, there are too many exciting things about Silverlight to cover in a mere few paragraphs. I hope you've enjoyed this post and found the information provided to be useful. Stay tuned for future posts where I'll start diving into the technical world of Silverlight and not only demonstrate, but hopefully teach you how you can leverage the power of the platform.

About The Author

Jonathan Atkins
Systems Analyst

Jonathan Atkins is a systems analyst for Mutual of Omaha developing software and database solutions for customer facing and internal systems. He also performs IT consulting services as Atkins Professional Services.

New Features in Silverlight 4

By Jonathan Atkins

Silverlight 4 delivers a new set of tools that enhance developer productivity and enable new levels of user interactivity for the next wave of rich internet applications. The design uses the familiar Extensible Application Markup Language (XAML) and the code uses either C# or VB.net.

New features include built-in printing support, elevated security permissions for trusted application for running Out-of-browser with direct access to the client PC allowing for file access and component access, including web cams and microphones. Right-click support and drag-and-drop are now enabled as well which really help create an entire full-featured application delivered over the wire. The new rich text box allows for quickly and easily creating fully editable and formattable content on the fly. The new WebBrowser control allows displaying HTML pages and content from outside pages inside of a Silverlight painless. Implicit styling allows styles to be defined once at the application level to create a consistent look and feel to your applications that is easily configurable or customizable by users. Developers can use the ChildWindow and NotificationWindow to create popups easily and notification windows(toasts) to alert the user of issues or informational messages. Using the community developed Silverlight Toolkit, a rich set of charting controls are available for displaying interactive charts and graphs that can be bound to datasets and configured to allow filtering and drilling down to the base data. The last major improvement to Silverlight 4 is the simplified use of WCF RIA services which allows for two-way data binding along with server and client side data validation. Validation rules can be setup on both the server and client side in a declarative format and bubbled up to the user with friendly error messages. Microsoft is positioning Silverlight to be the future of rich internet applications and has consistently shown its commitment to this new technology with rapid releases.

UI Updates With Databinding and INotifyPropertyChanged

By Dustin Horne

Today I'm going to show you how easy it is to leverage the power of property binding in Silverlight. I'm going to provide a very simplistic example that, while not very useful in itself, I hope will spark your imagination and show you how easy it is. In my example I'm going to show you how to capture the position of the mouse over a canvas and keep the live updated mouse position in a separate class. In addition, I'll show you how to update your UI directly (moving a box with the mouse and using labels to show the current mouse position) without directly wiring a single event to those controls in your page codebehind!

Before the MVVM pattern folks start foaming at the mouth over this article, I'd like to point out that this is meant to be a very simple demonstration and is not meant to be an MVVM application. However, I will introduce you to a few concepts along the way that you will leverage in the MVVM design pattern. I'm going to briefly introduce you to the DataContext object and I'll be storing the mouse data in a separate class but I will not create a complete Model-View-ViewModel separation for this example. I will, however be encouraging a separation of UI and logic that will serve as a good starting place when you decide to dive into the world of MVVM. Now, on with the show...

What the Heck is INotifyPropertyChanged?

INotifyPropertyChanged is an interface provided in Silverlight and WPF. For the purposes of this article, we will concentrate solely on Silverlight. Traditional ASP .NET applications rely on an event based model. Since the web is really a stateless demon, ASP .NET pages rely on the state of controls being preserved on the page and passed back to the application. While implementing INotifyPropertyChanged still requires an event hook, it does so in a stateful manner by allowing your user interface to interact with a connected data model. So let's take a second to step back and talk about what exactly MVVM means. This will help me keep the usefulness of property binding in context.

MVVM stands for Model-View-ViewModel. While the pattern seems daunting at first, it's really quite simplistic. It provides a means of creating separation between your user interface, your business logic, and your data model. This allows for a very nice separation of design and development processes. The Model is simply a class that provides access to your data. The Model can get its data from a variety of sources, whether it be static or dynamic such as pulling data from a database, XML, or some other arbitrary location. The Model is in charge of exposing that data, but not directly to the View. This allows also for easily switching your data store. Maybe you've started with XML but as demand grows you want to migrate to a database and pull the data from a web service. This can be done by simply switching out the model without a concern of how it will affect your UI.

The View is just as it states, a view. It's a "page" of sorts that holds your controls. Really it is a user control itself, but it is a container for the other controls. The View is responsible only for controlling the display of your controls and instantiating the ViewModel for the controls to bind to. The ViewModel is a bridge between the View and the Model. The ViewModel exposes the data to the View and handles any business logic such as performing actions based on changes to the View. It can be a one way or two way bridge, but it is a live connection. If data in the Model is changed, that data can move through the ViewModel and update the content of the controls on the View, and if a user changes data on the View, that data can move through the ViewModel and back into the Model, ultimately being updated in your actual data store if necessary. The details of MVVM are really beyond the scope of this article, but understanding the basics will help you understand the importance and usefulness of property binding.

Setting Up The Data For Property Binding

Ok, so we want to bind some data to some control properties but we need data to bind. For the purposes of this demo we are going to create a class called "MouseMonitor". The MouseMonitor class will implement `INotifyPropertyChanged` and will expose the position of the mouse to the controls on the page. Why would I want to expose them through a class instead of just setting the properties directly you ask? Well, if we were only using the data for a single property on a single control, you could easily update it from within your view if you're not worried about losing flexibility. However, what happens if you want to bind the same data to multiple controls, or even add controls later on without updating your code? I'm going to show you how to define your `DataContext` for your "View" and expose that data. This allows you to drop new controls into your XAML and bind to the `DataContext` without having to add code to update those controls. In a future article I'll show you how to bind that data programmatically which is even more useful as you may want to dynamically add objects to the page and not in your XAML.

What we need to do first is define our base class that will hold the mouse data that we'll be exposing. This class will implement `INotifyPropertyChanged`, which means it will have to contain a `PropertyChanged` event. The class is very simple. It contains an `OnPropertyChanged` subroutine which raises the `PropertyChanged` event, an `OnMouseMove` subroutine which we'll use as an entry point to hook our UI mouse event, and the `MouseX` and `MouseY` properties which will expose the current mouse position to the controls that are bound. The setter on each property will call our `OnPropertyChanged` method. I've also marked each setter as "Private" since we don't want the values changed outside of our class.

Take a deep breath. That seemed like a lot of steps but let's put it all together with a quick workflow description and some sample code. Here's how it breaks down:

1. Mouse movement fires the `OnMouseMove` subroutine in our class.
2. The `OnMouseMove` Subroutine updates the `MouseX` and `MouseY` properties.
3. The `MouseX` and `MouseY` properties fire the `PropertyChanged` event through `OnPropertyChanged` when updated.
4. The `PropertyChanged` event notifies bound controls that the properties have changed and they should update accordingly.

Now on with the source code:

```

Imports System.ComponentModel

Public Class MouseMonitor
    Implements INotifyPropertyChanged

    Private _mouseX As Double
    Private _mouseY As Double

    Public Property MouseX As Double
        Get
            Return _mouseX
        End Get

        Private Set(ByVal value As Double)
            If Not _mouseX = value Then
                _mouseX = value
                OnPropertyChanged(Me, "MouseX")
            End If
        End Set
    End Property

    Public Property MouseY As Double
        Get
            Return _mouseY
        End Get

        Private Set(ByVal value As Double)
            If Not _mouseY = value Then
                _mouseY = value
                OnPropertyChanged(Me, "MouseY")
            End If
        End Set
    End Property

    Public Event PropertyChanged As PropertyChangedEventHandler Implements INotifyPropertyChanged.PropertyChanged

    Private Sub OnPropertyChanged(ByVal sender As Object, ByVal propertyName As String)
        RaiseEvent PropertyChanged(sender, New PropertyChangedEventArgs(propertyName))
    End Sub

    Public Sub OnMouseMove(ByVal sender As Object, ByVal e As MouseEventArgs)
        Me.MouseX = e.GetPosition(sender).X
        Me.MouseY = e.GetPosition(sender).Y
    End Sub
End Class

```

I'm Gonna Tell You Where to Stick It

That's right, I'm just going to come right out and tell you where to stick it. It being our class and where being our friend DataContext. Getting back to the basics of MVVM, the user interface (view) really just defines how your application will look. The business logic, including that which reacts to events from your UI resides in the ViewModel, in this case our simple MouseMonitor class. While DataContext seems like a handy place to dump objects, it is really designed for a specific purpose, to expose your business logic and data to your view and provide a context for which the controls in your view can access that data.

For my sample application I wanted something simple, so I got rid of the default Grid and replaced it with a Canvas that is now my LayoutRoot object. I'll give you the XAML later when I tie it all together. For now, all you need to know is that everything in the view is contained within a Canvas with a name of "LayoutRoot". What we want to do now is detect mouse movement over the LayoutRoot UI element and handle it. This is done using the MouseMove event. The catch is that we will be handling the event with our MouseMonitor class (this is the reason for the OnMouseMove event in the code above) and updating the properties to expose the mouse location to the bound controls.

The first step is to put into DataContext and instance of our class. DataContext is of type Object and is created automatically an already there for us to use. We'll create the instance of the MouseMonitor class in the default constructor for our view. In this case I've left it as MainPage.xaml.vb. To do this we merely have to say: DataContext = New MouseMonitor and we brush off our hands and call it a day, well almost. We still have to hook our event to our MouseMonitor class. To do this, we simple add an event handler to our LayoutRoot and use the instance in DataContext like so:

```
AddHandler LayoutRoot.MouseMove, AddressOf CType(DataContext, MouseMonitor).OnMouseMove
```

NOTE: Make sure you're hooking your event after InitializeComponent() is called, otherwise you'll receive a null reference exception because the LayoutRoot object has not yet been initialized.

And that's all there is to it. In a few simple lines of code we've exposed our MouseMonitor class to the view and added a handler to track mouse movement over the LayoutRoot canvas. Going back to the MouseMonitor class code above you'll notice the following lines in the OnMouseMove routine:

```
Me.MouseX = e.GetPosition(sender).X
Me.MouseY = e.GetPosition(sender).Y
```

These lines get the position of the mouse from the supplied event arguments. Furthermore, the GetPosition() routine takes an object reference. We've supplied "sender" here so it gets the mouse position relative to the object that fired the event which is the LayoutRoot. Here is all of the code that is required in our View codebehind (MainPage.xaml.vb):

```
Partial Public Class MainPage
    Inherits UserControl

    Public Sub New()
        InitializeComponent()

        DataContext = New MouseMonitor
        AddHandler LayoutRoot.MouseMove, AddressOf CType(DataContext, MouseMonitor).OnMouseMove
    End Sub

End Class
```

And that's it. We are now ready to bind the MouseX and MouseY properties of our MouseMonitor class to any suitable control property we choose. We can also do this by purely modifying our XAML code which I'll show you next.

Binding Through XAML

Now that our code is prepared and we've made our data available through DataContext, we're ready to bind it to our controls. I'm going to start with a simple example of a Rectangle. The Rectangle is contained within the LayoutRoot canvas, so we can bind it's Canvas.Left and Canvas.Top properties to our MouseX and MouseY MouseMonitor properties. This will cause the upper left corner of the Rectangle to follow the mouse pointer around our canvas. You can drag and drop in the IDE of your choice, but the XAML is as follows:

```
<UserControl x:Class="DustinHorne.INPCDemo.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d"
    d:DesignHeight="500" d:DesignWidth="500" xmlns:dataInput="clr-
namespace:System.Windows.Controls;assembly=System.Windows.Controls.Data.Input">
    <Canvas x:Name="LayoutRoot" Background="Yellow" Width="500" Height="500">
        <Rectangle Height="84" HorizontalAlignment="Left" Stroke="Black"
```

```

    Canvas.Left="{Binding Path=MouseX}" Canvas.Top="{Binding Path=MouseY}" Width="102" Fill="#FF190000" />
</Canvas>
</UserControl>

```

Why So Much Work To Move One Object?

This is an easy question to answer. As I've previously mentioned, the purpose of this property binding demonstration is to show you how easy it is to make simple changes without a rewrite of your application. You'll see in the above XAML we have a Rectangle with its Canvas.Left and Canvas.Top properties bound. Imagine if this was a client project. The client now says, "Hey this is great! The black square follows the mouse around on the gawdy yellow background just like we wanted! But, wouldn't it be nice if we could show the user the location of the mouse cursor?"

At this point you slap your forehead, grit your teeth and (hopefully) ask the client for more money since we know that your requirements were perfectly documented and this new feature wasn't accounted for, right? Let's just say for the sake of keeping the client happy, you want to honor the request. Since we've implemented binding, we can do this with a simple addition to the XAML. This is crucial if you're a shop that splits the responsibilities between UI and Business Logic developers (and even data modellers).

So let's throw a few label controls in our XAML. We're going to include 4 labels. Two of these labels are not databound. They are simply identifiers to tell the user where you are displaying the X: location of the mouse and the Y: location of the mouse. The other two labels will have their Content property bound in the exact same way as the Rectangle's Canvas.Left and Canvas.Top properties. By slapping in 4 labels, dragging them where you want, and telling them what you want to bind to, you've honored the client's request with little to no effort. Our revised XAML now looks like this:

```

<UserControl x:Class="DustinHorne.INPCDemo.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d"
    d:DesignHeight="500" d:DesignWidth="500" xmlns:dataInput="clr-
namespace:System.Windows.Controls;assembly=System.Windows.Controls.Data.Input">
    <Canvas x:Name="LayoutRoot" Background="Yellow" Width="500" Height="500">
        <Rectangle Height="84" HorizontalAlignment="Left" Stroke="Black"
            Canvas.Left="{Binding Path=MouseX}" Canvas.Top="{Binding Path=MouseY}" Width="102" Fill="#FF190000" />
        <dataInput:Label Canvas.Left="12" Canvas.Top="450" Height="17" Width="18" Content="x:" />
        <dataInput:Label Canvas.Left="27" Canvas.Top="449" Height="22" Width="46" Content="{Binding Path=MouseX}" />
        <dataInput:Label Canvas.Left="12" Canvas.Top="470" Height="18" Width="19" Content="y:" />
        <dataInput:Label Canvas.Left="27" Canvas.Top="468" Height="23" Width="46" Content="{Binding Path=MouseY}" />
    </Canvas>
</UserControl>

```

Closing Notes / Demo

And that's it folks. Property binding in Silverlight is a breeze. While this demo doesn't serve any practical purpose, I hope it serves as a base for you to come up with your own creative ideas. Have a look at the demo on my blog:

<http://www.dustinhorne.com/post/UI-Updates-With-Databinding-and-INotifyPropertyChanged.aspx>

I've compiled it against Silverlight 3.

A Message From The Editor

By Jake Johnson

What is the purpose of the OmahaMTG newsletter?

If you have made it all of the way down to this page, I am sure that you have a good understanding on what the newsletter is all about. If you don't, then I can fill you in. The OmahaMTG team was looking for a way to share and distribute information about different Microsoft technologies to users in the Omaha community. What better way than having fellow users write little articles describing various technologies that they have experience with.

Why do I keep receiving e-mails about writing an article?

Without the contribution of those writing articles, we wouldn't have a newsletter. The same goes for speakers at the User Group events. There are many positives to pitching in and writing an article. You not only will help out the community, but you will also learn the technology that you are writing about a little bit better. It is also a great way to get your name out there and have a conversation with someone that you may not have known before.

Who is going to read this anyway?

You just did!

Upcoming Events...

Microsoft PDC 2010, October 28-29, <http://www.microsoftpdc.com/>

Heartland Developer Conference, September 8 – 10, <http://www.heartlanddc.com/>

Next .Net UG meeting, August 26th