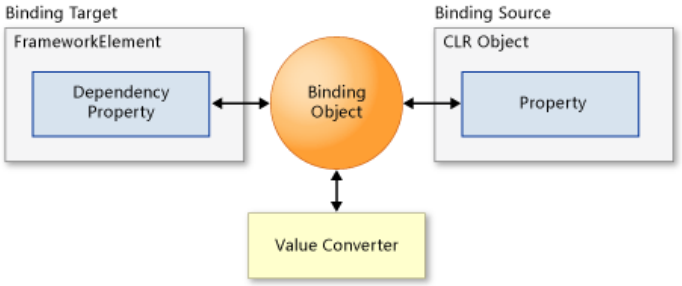
Silverlight 4

پژوهش و نگارش: وحید نصیری
Vahid_Nasiri@Yahoo.Com
بهار ۱۳۸۹

فهرست مطالب

فصل ۸ - آشنایی با سیستم Binding در Silverlight



شکل ۱– انقیاد داده‌ها سبب ارتباط منابع داده و رابط گرافیکی کاربر می‌شود

مقدمه

Data-Binding عملیاتی است که توسط آن بین رابط گرافیکی کاربر و منطق تجاری برنامه ارتباط برقرار می‌شود (شکل ۱). روش‌های انقیاد داده‌ها (Data-Binding) در محصولات مایکروسافت در طی بیش از یک دهه، توسعه‌ی مداوم داشته‌اند. در دهه‌ی ۹۰ میلادی، Classics ASP روشی ساده را برای افزودن داده‌های پویا به صفحات ارائه داد. سپس این روش‌ها در ASP.NET بسیار تکامل یافته و امکان bind داده‌ها به کنترل‌ها و تولید خودکار خروجی HTML از آن‌ها را میسر ساخت. آخرین فناوری موجود در این زمینه را می‌توان در WPF و سپس نمونه‌ی خلاصه شده‌ای از آن را در Silverlight شاهد بود که یکی از مهم‌ترین توانایی‌های این فناوری‌ها به شمار رفته و پایه و اساس برنامه نویسی تجاری با این سیستم‌ها می‌باشند. میزان اهمیت فصل جاری تا آن حد است که می‌توان آن‌را پیشنیاز الزامی مطالعه و درک فصل‌های آتی کتاب درنظر گرفت.

نحوه‌ی تعریف Data-Binding

نحوه‌ی تعریف Data-Binding در Silverlight همانند WPF است؛ هر چند با توجه به حجم کم آن، تنها موارد مهم آن‌را به ارث برده است. لطفا به مثال زیر دقت بفرمائید:

XAML

```
<TextBox x:Name="FirstNameTextBox1" Text="{Binding FirstName}" />
<TextBox x:Name="FirstNameTextBox2" Text="{Binding Path=FirstName}" />
```

```
<TextBox x:Name="FirstNameTextBox3">
   <TextBox.Text>
      <Binding Path="FirstName" />
   </TextBox.Text>
</TextBox>
```

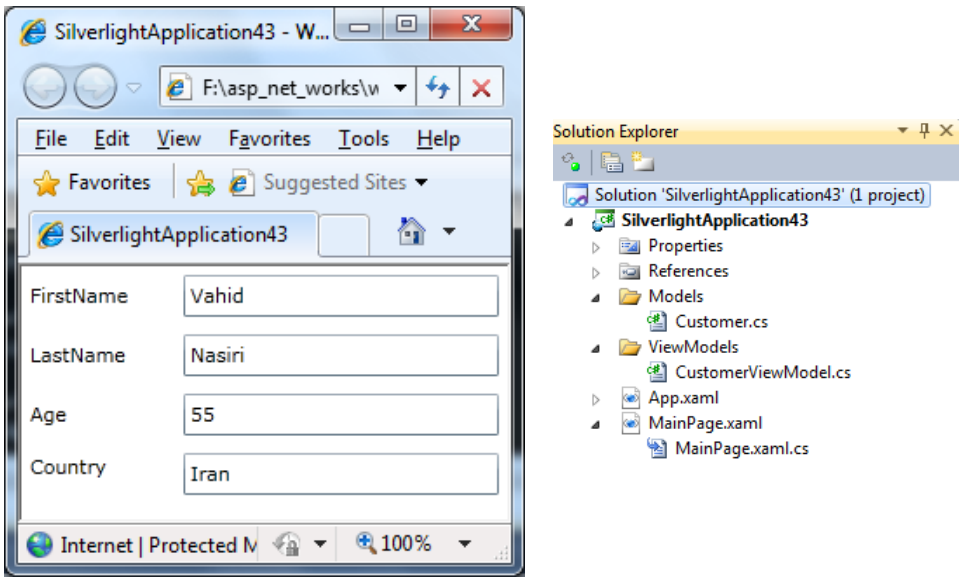
در این مثال خاصیت FirstName یک شیء به خاصیت Text یک شیء TextBox مقید (Bind) شده است. هر سه نحوه‌ی تعریف ذکر شده دقیقا معادل می‌باشند. کنترل FirstNameTextBox1 ، صرفا راه میانبری را جهت روش ذکر شده در کنترل FirstNameTextBox2 ارائه می‌دهد. در FirstNameTextBox3 از روش property-element syntax کمک گرفته شده است که طولانی‌تر است.

تا اینجا تنها انقیاد داده‌ها را تعریف کرده‌ایم. اما سؤال اینجا است که داده‌های این جعبه‌های متنی از کجا تامین خواهند شد؟ برای اینکه عملیات Binding به درستی کار کند باید خاصیت DataContext مربوط به آن کنترل، یا شیء والد آن مقدار دهی گردد. برای مثال:

C#

```
FirstNameTextBox1.DataContext = myCustomer;
```

در این مثال اگر خاصیت FirstName مربوط به شیء myCustomer مورد استفاده قرار گیرد، روش مناسبی جهت مقدار دهی DataContext انتخاب گردیده است. اما در عمل هر شیء، چندین خاصیت داشته و در یک فرم، خواص آن به کنترل‌های متعددی مقید خواهند گردید. لطفا به مثال کاملی در این زمینه دقت بفرمائید (شکل ۲):



شکل ۲– ساختار پوشه‌ها و فایل‌های اولین مثال انقیاد داده‌ها به همراه نمایی از آن در حال اجرا

ابتدا یک پروژه‌ی جدید Silverlight را آغاز کرده و سپس دو پوشه‌ی جدید را به نام‌های Model و ViewModel به آن اضافه نمائید. تعریف کلاس مشتری ما در پوشه‌ی Model قرار خواهد گرفت. مرسوم است که کار دریافت اطلاعات انقیاد داده‌ها توسط صفحه‌ی اصلی برنامه، به کمک کلاسی به نام ViewModel انجام شود (در مورد این الگوی طراحی در طی فصول بعدی بیشتر توضیح داده خواهد شد).

کدها و نحوه‌ی تعریف این دو کلاس را در ادامه مشاهده خواهید کرد:

Customer.cs

```
namespace SilverlightApplication43.Models
{
    public class Customer
    {
        public string FirstName { get; set; }
        public string LastName { get; set; }
        public int Age { get; set; }
        public string Country { get; set; }
    }
}
```

CustomerViewModel.cs

```
using SilverlightApplication43.Models;

namespace SilverlightApplication43.ViewModels
{
    public class CustomerViewModel
    {
        public Customer Customer { get; set; }

        public CustomerViewModel()
        {
            this.Customer = new Customer
            {
                FirstName = "Vahid",
                LastName = "Nasiri",
                Age = 55,
                Country = "Iran"
            };
        }
    }
}
```

اکنون جهت معرفی ViewModel برنامه به View ، مطابق کدهای بعد عمل خواهیم نمود. در اینجا DataContext شیء جاری (یا همان User control) مقدار دهی شده است؛ بنابراین دیگر نیازی نخواهد بود تا به ازای تک تک جعبه‌های متنی تعریف شده در صفحه‌ی اصلی، DataContext را مقدار دهی نمائیم و اطلاعات انقیاد

داده‌ها از شیء والد دریافت می‌گردد. DataContext منبع اطلاعاتی است که در اختیار اشیاء مقصد قرار می‌گیرد و اشیاء فرزند، قابلیت ارث بری از آن را دارند. به همین جهت برای حالت‌هایی که چندین المان قرار است از یک منبع داده‌ای استفاده کنند، بسیار مناسب می‌باشد.

MainPage.xaml.cs

```
using SilverlightApplication43.ViewModels;

namespace SilverlightApplication43
{
    public partial class MainPage
    {
        public MainPage()
        {
            InitializeComponent();

            this.DataContext = new CustomerViewModel();
        }
    }
}
```

در Data-Binding دو شیء دخیل هستند: مقصد یا همان شیء بصری که قرار است اطلاعات را نمایش دهد و منبع که می‌تواند یک مدل تجاری و مانند آن باشد. منبع را می‌توان یک خاصیت تعریف شده در اشیاء .NET در نظر گرفت اما مقصد همواره باید از نوع DependencyProperty که در فصل‌های قبل در مورد آن توضیح داده شد، باشد.

در پایان نحوه‌ی تعریف انقیاد داده‌های برنامه به جعبه‌های متنی به شرح بعد می‌باشند.

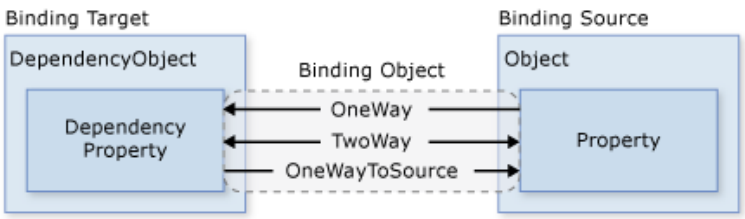
MainPage.xaml

```
<UserControl x:Class="SilverlightApplication43.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="300" d:DesignWidth="400"
    xmlns:sdk=
     "http://schemas.microsoft.com/winfx/2006/xaml/presentation/sdk">
    <Grid x:Name="LayoutRoot" Background="White">
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="126*" />
            <ColumnDefinition Width="274*" />
        </Grid.ColumnDefinitions>
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto" />
```

```
            <RowDefinition Height="Auto" />
        <RowDefinition Height="Auto" />
        <RowDefinition Height="*" />
    </Grid.RowDefinitions>
    <sdk:Label  Margin="5" Grid.Row="0" Content="FirstName" />
    <sdk:Label Grid.Row="1"  Margin="5" Content="LastName" />
    <sdk:Label Grid.Row="2" Margin="5" Content="Age" />
    <sdk:Label Grid.Row="3" Margin="5" VerticalAlignment="Top"
                Content="Country" />
    <TextBox Grid.Column="1"  Grid.Row="0"  Margin="5"
             Text="{Binding Customer.FirstName}"
             />
    <TextBox Grid.Column="1" Grid.Row="1" Margin="5"
             Text="{Binding Customer.LastName}"
             />
    <TextBox Grid.Column="1" Grid.Row="2" Margin="5"
             Text="{Binding Customer.Age}"
             />
    <TextBox Grid.Column="1" Grid.Row="3"  Margin="5"
             VerticalAlignment="Top"
             Text="{Binding Customer.Country}"
             />
  </Grid>
</UserControl>
```

استفاده از کلاس ViewModel مزایای بسیاری را به همراه دارد. برای مثال اکنون شما می‌توانید ظاهر رابط گرافیکی برنامه را تغییر دهید بدون اینکه نیازی باشد تا کلاس‌های Model و ViewModel برنامه تغییر کنند. همچنین با استفاده از سیستم پیشرفته‌ی انقیاد داده‌ای که ملاحظه می‌نمائید، بین رابط گرافیکی برنامه و داده‌های مرتبط با آن گره خوردگی وجود ندارد. برای مثال در کلاس ViewModel هیچ اثری از کنترل‌های TextBox مشاهده نمی‌شود. همین امر آزمودن کلاس ViewModel را سهولت می‌بخشد. این مباحث پایه و اساس الگوی طراحی M-V-VM می‌باشند که در طی چند فصل در مورد آن بحث خواهیم کرد.

انواع انقیاد داده‌ها و یا جهت‌های متفاوت سیلان داده‌ها



شکل ۳– جهت‌های متفاوت سیلان داده‌ها در حین انقیاد آن‌ها

Silverlight سه روش متفاوت انقیاد داده‌ها را تعریف نموده است (شکل ۳) که جهت سیلان داده‌ها و همچنین زمان انجام این عملیات را مشخص می‌کنند:

۱. OneTime : آیا منبع داده مورد استفاده هیچگاه تغییر نمی‌کند؟ اگر پاسخ بلی است از این حالت انقیاد داده‌ها استفاده نمائید. در این حالت مقصد (Target) مقدار دهی شده و عملیات انقیاد داده‌ها فراموش خواهد شد. بنابراین اگر احیانا تغییری در منبع داده‌ها (Source) صورت گیرد، در مقصد منعکس نخواهد گردید.
۲. OneWay : در این حالت به ازای هر بار تغییر منبع داده‌ها، مقصد نیز به روز رسانی خواهد شد (بدون هیچ نوع کد نویسی خاصی) و حالت پیش فرض انقیاد داده‌ها در Silverlight می‌باشد. بهترین حالت برای نمایش فقط خواندنی اطلاعاتی است که احتمال تغییر در منبع آن وجود دارد.
۳. TwoWay : در این حالت انقیاد داده‌ها، اگر منبع تغییر کند بلافاصله تغییرات در مقصد نیز منعکس می‌گردد و برعکس (برای مثال داده‌های وارد شده در رابط گرافیکی کاربر برنامه، بلافاصله در منبع داده‌ها نیز تاثیر داده شود).

نحوه‌ی تعریف این حالات متفاوت سیلان داده‌ها در کدهای XAML برنامه به شکل زیر است:

XAML

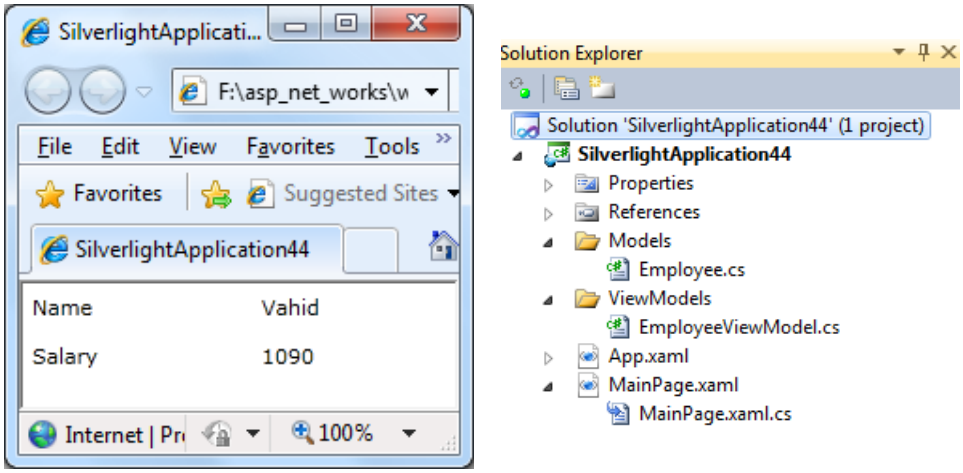
```
<TextBox x:Name="FirstName"
         Text="{Binding Customer.FirstName, Mode=OneTime}" />
```

مطابق توضیحات قبل، اگر مقدار ویژگی Mode ذکر نگردد، حالت OneWay در Silverlight تنظیم خواهد شد.

نکته‌ی بسیار مهم این حالات سیلان اطلاعات، عدم نیاز به کد نویسی جهت به روز رسانی هر یک از طرفین می‌باشد. برای مثال اگر در قسمتی از برنامه شیء Customer تغییر کرد و حالت انقیاد داده‌ها به مقدار OneWay یا TwoWay تنظیم شده بود، کاربر بلافاصله تغییرات را در رابط گرافیکی برنامه مشاهده خواهد نمود. هر چند این مورد نکته‌ای اساسی در تعریف عملیات Binding است، اما شاید این سؤال پیش آید که Silverlight از کجا متوجه خواهد شد که در شیء Customer تغییری حاصل شده است؟ پاسخ به این مورد، قسمت بعدی فصل جاری را تشکیل خواهد داد.

تشخیص خودکار تغییرات در منبع داده‌ها

برای انتشار آنی و خودکار تغییرات در منبع داده‌ها به مقصد یا همان رابط گرافیکی کاربر برنامه، نیاز است تا کلاس مدل مورد استفاده، اینترفیس استاندارد INotifyPropertyChanged را پیاده سازی کند. جهت توضیح کاربردی این روش لطفا به مثال بعد دقت بفرمائید (شکل ۴).



شکل ۴- ساختار پوشه‌ها و فایل‌های دومین مثال انقیاد داده‌ها به همراه نمایی از آن در حال اجرا

در دومین مثال انقیاد داده‌های فصل جاری، ساختار پوشه‌ها همانند مثال قبلی است که در آن دو پوشه‌ی Models و ViewModels به برنامه اضافه شده‌اند. سپس کدهای مدل و ViewModel مورد نظر به شرح زیر می‌باشند:

Employee.cs

```
using System.ComponentModel;

namespace SilverlightApplication44.Models
{
    public class Employee : INotifyPropertyChanged
    {
        string _name;
        public string Name
        {
            get { return _name; }
            set
            {
```

```
            if (_name == value) return;
            _name = value;
            raisePropertyChanged("Name");
        }
    }

    int _salary;
    public int Salary
    {
        get { return _salary; }
        set
        {
            if (_salary == value) return;
            _salary = value;
            raisePropertyChanged("Salary");
        }
    }

    public event PropertyChangedEventHandler PropertyChanged;

    void raisePropertyChanged(string propertyName)
    {
        var handler = PropertyChanged;
        if (handler == null) return;
        handler(this, new PropertyChangedEventArgs(propertyName));
    }
  }
}
```

کلاس کارمند جهت انتشار آنی تغییرات خواص خود به رابط کاربر، اینترفیس استاندارد INotifyPropertyChanged را همانند کدهای فوق پیاده سازی کرده است. این روش، استاندارد بوده و در مورد تمامی کلاس‌های آتی مورد استفاده‌ی شما نیز به همین شکل خواهد بود. اینترفیس استاندارد INotifyPropertyChanged دارای یک رخداد به نام PropertyChanged است که تمامی خواص کلاس جاری باید به این طریق، تغییرات خود را به سیستم Binding موجود در Silverlight اعلام کنند. در غیراینصورت، تغییرات رخداده در داده‌های برنامه به صورت خودکار به رابط کاربر منعکس نخواهند شد.

در ادامه ViewModel برنامه جهت تهیه اطلاعات مورد نیاز View (یا همان صفحه‌ی اصلی برنامه در این مثال)، تعریف شده است. در این کلاس یک Timer از نوع DispatcherTimer جهت افزودن مقادیر آزمایشی به خاصیت حقوق شیء کارمند تعریف شده ، در فواصل زمانی مشخص بکارگرفته شده است. استفاده از کلاس DispatcherTimer ، متداول‌ترین روش تعریف Timer در Silverlight می‌باشد:

EmployeeViewModel.cs

```
using System;
using System.Windows.Threading;
using SilverlightApplication44.Models;

namespace SilverlightApplication44.ViewModels
{
    public class EmployeeViewModel
    {
        public Employee Employee { set; get; }

        public EmployeeViewModel()
        {
            this.Employee = new Employee
                                {
                                    Name = "Vahid",
                                    Salary = 1000
                                };

            startTimer();
        }

        //create a timer in Silverlight
        private void startTimer()
        {
            var myDispatcherTimer =
                new DispatcherTimer
                    {
                        // 1000 Milliseconds
                        Interval = new TimeSpan(0, 0, 0, 0, 1000)
                    };
            myDispatcherTimer.Tick += eachTick;
            myDispatcherTimer.Start();
        }

        void eachTick(object o, EventArgs sender)
        {
            Employee.Salary += 10;
        }
    }
}
```

همانطور که در این ViewModel مشاهده می‌کنید، هیچگونه ارجاعی از View برنامه در آن وجود نداشته و تغییرات رخداده در خاصیت حقوق شیء کارمند (در متد eachTick) به صورت خودکار به View برنامه منعکس می‌گردد و این امر با پیاده سازی اینترفیس INotifyPropertyChanged میسر شده است.

در ادامه باید اطلاعات این ViewModel را در اختیار View برنامه قرار داد :

MainPage.xaml.cs

```
using SilverlightApplication44.ViewModels;

namespace SilverlightApplication44
{
    public partial class MainPage
    {
        public MainPage()
        {
            InitializeComponent();

            this.DataContext = new EmployeeViewModel();
        }
    }
}
```

سپس تعاریف انقیاد داده‌ها در کدهای XAML صفحه‌ی اصلی برنامه به شرح بعد خواهند بود :

MainPage.xaml

```
<UserControl x:Class="SilverlightApplication44.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="300" d:DesignWidth="400">
    <Grid x:Name="LayoutRoot" Background="White">
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="122" />
            <ColumnDefinition Width="*" />
        </Grid.ColumnDefinitions>
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto" />
            <RowDefinition Height="Auto" />
            <RowDefinition Height="*" />
        </Grid.RowDefinitions>
        <TextBlock  Margin="5"  Text="Name"  />
        <TextBlock Grid.Row="1" Margin="5" Text="Salary" />
        <TextBlock Grid.Column="1"  Margin="5"
                   Text="{Binding Employee.Name}"  />
        <TextBlock Grid.Column="1" Grid.Row="1" Margin="5"
                   Text="{Binding Employee.Salary}" />
    </Grid>
</UserControl>
```

انقیاد به لیستی از اشیاء

در عمل اکثر مسایل همراه با انقیاد داده‌ها به شکل لیستی از اطلاعات می‌باشند؛ لیست کارمندان، لیست محصولات و غیره. خوشبختانه در Silverlight و WPF ، یک نوع Generic List ویژه جهت کار با اشیایی که اینترفیس INotifyPropertyChanged را پیاده سازی می‌کنند به نام ObservableCollection<T> وجود دارد. زمانیکه اشیاء مورد نظر ما به یک ObservableCollection اضافه شده یا از آن حذف می‌شوند، به صورت خودکار رخداد CollectionChanged آن فراخوانی می‌گردد. به این صورت دیگر نیازی به هیچگونه مطلع سازی Silverlight در مورد تغییرات رخداده در آن لیست نخواهد بود و تمام عملیات به صورت خودکار انجام می‌شود. برای مثال اگر یک ObservableCollection به خاصیت ItemsSource یک ListBox مقید شود، پس از افزودن شیءایی به آن، نتیجه‌ی حاصل در ListBox نمایش داده شده و اگر شیءایی از این لیست حذف شود، بلافاصله و بدون کد نویسی خاصی، ردیف متناظر با آن از ListBox حذف خواهد گردید.

مفهوم دیگری که حین انقیاد لیستی از اشیاء به کنترل‌های مرتبط حائز اهمیت می‌شود، data template نام دارد. برای مثال ListBox ذیل، به ازای نمایش هر ردیف خود، یکبار کار ساخت قالب داده‌ای تعریف شده را انجام می‌دهد. به این ترتیب می‌توان کنترل‌های بسیار متنوعی را خلق کرد. برای مثال ایجاد لیستی شبیه به لیست‌های صفحات وب که هر ردیف آن شامل یک تصویر، عنوان ، متن و سایر کنترل‌ها است. همچنین امکان بهره‌گیری کامل از مزایای Binding نیز در اینجا مهیا است.

XAML

```
<ListBox ItemsSource="{Binding}">
    <ListBox.ItemTemplate>
        <DataTemplate>
            <TextBlock Text="{Binding Name}"></TextBlock>
        </DataTemplate>
    </ListBox.ItemTemplate>
</ListBox>
```

در این مثال خاصیت ItemsSource مساوی {Binding} قرار گرفته است. این مورد بدین معنا است که تمام اطلاعات والد جهت انقیاد داده‌ها مورد استفاده قرار خواهد گرفت.

تعریف قالب‌های داده‌ای در قسمت منابع یک User control بسیار متداول است. در ادامه مثالی را از این دست به همراه نحوه‌ی بکارگیری این منبع، ملاحظه می‌نمائید :

XAML

```
<UserControl.Resources>
    <DataTemplate x:Key="employeeTemplate">
        <TextBlock Text="{Binding Name}"></TextBlock>
    </DataTemplate>
```

```
</UserControl.Resources>

<ListBox ItemsSource="{Binding}"
         ItemTemplate="{StaticResource employeeTemplate}">
</ListBox>
```

به این ترتیب کدهای XAML کنترل ListBox ما اندکی خواناتر خواهند شد.

باید در نظر داشت که منبع تعریف شده در سطح User control تنها در همان فایل قابل استفاده است. اگر نیاز به تعریف منبعی وجود داشت که در سطح برنامه استفاده شود، می‌توان آن‌را در فایل استاندارد App.XAML تعریف نمود.

بدیهی است امکان استفاده از سیستم‌های طرح بندی نیز جهت تعریف قالب‌های داده‌ای مجاز است. برای مثال:

XAML

```
<ListBox ItemsSource="{Binding}">
    <ListBox.ItemTemplate>
        <DataTemplate>
            <StackPanel Orientation="Horizontal">
                <TextBlock Text="{Binding Name}" Width="90" />
                <TextBlock Text="{Binding LastName}" Width="90" />
            </StackPanel>
        </DataTemplate>
    </ListBox.ItemTemplate>
</ListBox>
```

انقیاد به رخدادها

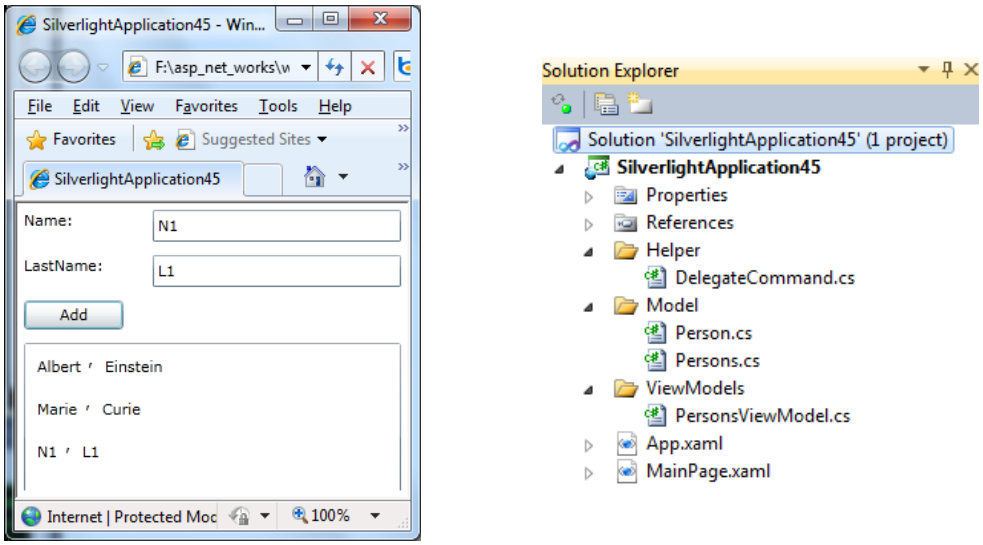
یکی از ویژگی‌های جدید Silverlight 4 ، امکان انقیاد به رخدادها است. امروزه جدا سازی لایه‌های برنامه از یکدیگر یکی از مسایل بسیار مهم طراحی و برنامه نویسی به شمار می‌روند. مشکل اصلی روش کد نویسی متداول در Silverlight و یا WPF ، تعریف روال‌های رخدادگردان در فایل‌های Code behind صفحات XAML برنامه است. این مورد سبب گره خوردگی رابط گرافیکی کاربر به منطق تجاری آن شده و در عمل امکان تهیه‌ی آزمون‌های واحد خودکار آن‌را بسیار مشکل می‌نماید. به همین جهت رسیدن به فایل‌های Code behind خالی از کد، یکی از آرزوهای مهم الگوهای برنامه نویسی شیءگرا به شمار می‌روند.

اولین قدم در پیاده سازی انقیاد به یک رخداد، پیاده سازی اینترفیس استاندارد ICommand می‌باشد. با پیاده سازی این اینترفیس دو متد مهم CanExecute و Execute تعریف خواهند شد. با استفاده از متد CanExecute ابتدا بررسی می‌شود که آیا اطلاعات دریافتی معتبر است یا خیر و در صورت تعیین اعتبار شیء دریافتی، متد

Execute اجرا خواهد شد. از آنجائیکه پیاده سازی این اینترفیس تکراری بوده و در اکثر پروژه‌های شما حضور خواهد داشت، در مثال بعدی، کلاس کمکی DelegateCommand برای سهولت تعاریف مربوطه ایجاد شده است.

در این مثال ابتدا یک شیء از نوع ICommand در ViewModel برنامه جهت انقیاد به رخداد کلیک افزودن اطلاعات تعریف شده و سپس در سازنده‌ی کلاس با کمک کلاس کمکی DelegateCommand ، متدهای CanExecute و Execute به آن معرفی خواهند شد. مرحله‌ی بعد تعریف ویژگی‌های Command و CommandParameter در کدهای XAML برنامه است.

در ادامه در طی یک مثال نسبتا جامع قصد داریم مباحث انقیاد به لیستی از اشیاء و همچنین انقیاد به رخدادها را با جزئیات بیشتری بررسی نمائیم (شکل ۵).



شکل ۵– ساختار پوشه‌ها و فایل‌های برنامه انقیاد به رخدادها به همراه تصویری از آن در حال اجرا

DelegateCommand یک کلاس کمکی جهت سهولت تعاریف اشیاء Command و مدیریت آن‌ها می‌باشد. از این کلاس در ViewModel برنامه برای انقیاد به رخداد کلیک در برنامه استفاده خواهیم کرد. کدهای این کلاس را در ادامه ملاحظه می‌نمائید:

DelegateCommand.cs

```
using System;
using System.Windows.Input;
namespace SilverlightApplication45.Helper
{
    public class DelegateCommand<T> : ICommand
    {
        readonly Func<T, bool> _canExecute;
        readonly Action<T> _executeAction;
        bool _canExecuteCache;
```

```
        public DelegateCommand(Action<T> executeAction,
            Func<T, bool> canExecute)
        {
            _executeAction = executeAction;
            _canExecute = canExecute;
        }

        public bool CanExecute(object parameter)
        {
            bool temp = _canExecute((T)parameter);

            if (_canExecuteCache != temp)
            {
                _canExecuteCache = temp;

                if (CanExecuteChanged != null)
                {
                    CanExecuteChanged(this, new EventArgs());
                }
            }

            return _canExecuteCache;
        }

        public event EventHandler CanExecuteChanged;
        public void Execute(object parameter)
        {
            _executeAction((T)parameter);
        }
    }
}
```

مدل برنامه از دو کلاس Person و Persons تشکیل شده است. جهت استفاده از امکانات پیشرفته‌ی Binding در Silverlight نیاز است تا کلاس Person اینترفیس INotifyPropertyChanged را نیز پیاده سازی کند:

Person.cs

```
using System.ComponentModel;

namespace SilverlightApplication45.Model
{
    public class Person : INotifyPropertyChanged
    {
        string _firstName;
        public string FirstName
```

```
        {
            get { return _firstName; }
            set
            {
                if (_firstName == value) return;
                _firstName = value;
                raisePropertyChanged("FirstName");
            }
        }

        string _lastName;
        public string LastName
        {
            get { return _lastName; }
            set
            {
                if (_lastName == value) return;
                _lastName = value;
                raisePropertyChanged("LastName");
            }
        }

        public event PropertyChangedEventHandler PropertyChanged;
        void raisePropertyChanged(string propertyName)
        {
            var handler = PropertyChanged;
            if (handler == null) return;
            handler(this, new PropertyChangedEventArgs(propertyName));
        }
    }
}
```

سپس برای تعریف گروهی از اشخاص، از اشیاء ObservableCollection کمک گرفته شده است. نحوه‌ی تعریف استاندارد اینگونه لیست‌ها به شکل زیر می‌باشد:

Persons.cs

```
using System.Collections.ObjectModel;

namespace SilverlightApplication45.Model
{
    public class Persons : ObservableCollection<Person>
    {
    }
}
```

در ادامه به مهم‌ترین قسمت برنامه می‌رسیم. در کلاس ViewModel کار ایجاد یک سری اطلاعات پیش فرض (برای مثال افزودن دو شخص به لیست اشخاص) و همچنین کار انقیاد به رخداد کلیک افزودن اشخاص انجام می‌شود:

PersonsViewModel.cs

```
using System.Windows.Input;
using SilverlightApplication45.Helper;
using SilverlightApplication45.Model;

namespace SilverlightApplication45.ViewModels
{
    public class PersonsViewModel
    {
        public ICommand AddDataCommand { get; set; }

        public Person EnteredData { set; get; }

        public Persons Persons { set; get; }
        public PersonsViewModel()
        {
            Persons = new Persons
                        {
                            new Person
                                {
                                    FirstName = "Albert",
                                    LastName = "Einstein",
                                },
                            new Person
                                {
                                    FirstName = "Marie",
                                    LastName = "Curie "
                                }
                        };

            AddDataCommand = new DelegateCommand<Person>(addData,
                                canAddData);
            EnteredData = new Person();
        }

        private bool canAddData(Person enteredPerson)
        {
            return enteredPerson != null;
        }

        private void addData(Person enteredPerson)
        {
            if (enteredPerson == null) return;
            Persons.Add(new Person
```

```
                                {
                                    FirstName = enteredPerson.FirstName,
                                    LastName = enteredPerson.LastName
                                });
        }
    }
}
```

همانطور که در کدهای این ViewModel ملاحظه می‌نمائید، شیء Persons که از نوع ObservableCollection است، برای انقیاد به ListBox در اختیار View برنامه قرار می‌گیرد. همچنین یک شیء دیگر به نام EnteredData جهت دریافت اطلاعات وارد شده در جعبه‌های متنی برنامه تعریف شده است. به این صورت و با کمک گیری از انقیاد دو طرفه، دیگر نیازی نخواهد بود تا اطلاعات وارد شده را بر اساس نام دقیق هر یک از جعبه‌های متنی دریافت کنیم. بنابراین جداسازی قابل توجهی بین تعاریف رابط کاربر و مدل برنامه صورت خواهد گرفت.

در اینجا نیاز است تا از کلیه‌ی اشیاء تعریف شده، وهله‌هایی درسازنده‌ی کلاس ایجاد گردد. به این صورت از Null بودن آن‌ها در حین استفاده جلوگیری به عمل می‌آید.

نکته‌ی مهم دیگر این ViewModel ، انقیاد به رخداد کلیک افزودن اطلاعات در برنامه است. با کمک این توانایی، دیگر نیازی نخواهد بود تا روال رخدادگردان مربوطه را در Code behind صفحه‌ی اصلی برنامه تعریف کرده و منطق تجاری برنامه را با رابط گرافیکی کاربر آن گره بزنیم.

در ادامه کدهای XAML تعاریف رابط کاربر گرافیکی برنامه به همراه نحوه‌ی تعریف Binding به اشیاء و رخدادها را ملاحظه خواهید نمود.

MainPage.xaml

```
<UserControl x:Class="SilverlightApplication45.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"
    xmlns:local="clr-namespace:SilverlightApplication45.ViewModels"
    mc:Ignorable="d" d:DesignHeight="300" d:DesignWidth="400">
    <UserControl.Resources>
        <local:PersonsViewModel x:Key="vm"/>
    </UserControl.Resources>
    <Grid x:Name="LayoutRoot" Background="White"
                DataContext="{Binding Source={StaticResource vm}}">
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="97" />
            <ColumnDefinition Width="303*" />
        </Grid.ColumnDefinitions>
        <Grid.RowDefinitions>
```

```
            <RowDefinition Height="Auto" />
            <RowDefinition Height="Auto" />
            <RowDefinition Height="Auto" />
            <RowDefinition Height="*" />
        </Grid.RowDefinitions>
        <TextBlock Text="Name:" Margin="5" Grid.Row="0" />
        <TextBox Margin="5" Name="txtName" Grid.Row="0" Grid.Column="1"
                 Text="{Binding EnteredData.FirstName, Mode=TwoWay}"
                 />
        <TextBlock Text="LastName: " Margin="5" Grid.Row="1" />
        <TextBox Margin="5" Name="txtLastName"
                 Grid.Column="1" Grid.Row="1"
                 Text="{Binding EnteredData.LastName, Mode=TwoWay}"
                 />
        <Button Grid.Row="2" Content="Add" Name="btnAdd"
                Margin="5" Width="75" HorizontalAlignment="Left"
                Command="{Binding AddDataCommand}"
                CommandParameter="{Binding EnteredData}"
                />

        <ListBox Grid.Row="3" Grid.ColumnSpan="2"
                 Height="190" Margin="5"
                 ItemsSource="{Binding Persons}"
                 VerticalAlignment="Top" Width="Auto"
                 Name="lstData"
                 >
            <ListBox.ItemTemplate>
                <DataTemplate>
                   <StackPanel Orientation="Horizontal">
                     <TextBlock Text="{Binding FirstName}" Margin="5" />
                     <TextBlock Text=", "  />
                     <TextBlock Text="{Binding LastName}" Margin="5" />
                   </StackPanel>
                </DataTemplate>
            </ListBox.ItemTemplate>
        </ListBox>
    </Grid>
</UserControl>
```

چند نکته در این مثال قابل توجه می‌باشند :

۱. یک روش دیگر تعریف دسترسی به اطلاعات ViewModel برنامه، تعریف آن به صورت یک Static Resource است که در ابتدای فایل، به همراه افزودن فضای نام مرتبط آن به کدهای XAML برنامه قابل مشاهده است.

۲. سپس این Static Resource به DataContext مربوط به Grid اصلی صفحه نسبت داده شده است تا در اختیار کلیه‌ی اشیای فرزند آن قرار گیرد.

۳. انقیاد به شیء EnteredData از نوع دو طرفه تعریف شده است تا به محض ورود اطلاعات توسط کاربر، اطلاعات این شیء نیز به صورت خودکار به روز رسانی شده و در ViewModel برنامه قابل دسترسی باشد.
۴. در دکمه‌ی افزودن اطلاعات، دو ویژگی جدید Command و CommandParameter را می‌توان مشاهده کرد. Command به رخداد متناظر آن در ViewModelبرنامه مقید شده است و توسط CommandParameter، شیء EnteredData مهیا، به روال‌های مدیریت Command ارسال خواهد شد.
۵. در ListBox تعریف شده، نمونه‌ای از پیاده سازی DataTemplate را ملاحظه می‌نمائید.
۶. در فایل Code behind متناظر با این View ، هیچ نوع منطق تجاری پیاده شده‌ای را نخواهید یافت.

در طی فصل بعد با اصول کاری الگوی M-V-VM که در مثال‌های فصل جاری به کرات مورد استفاده قرار گرفت و پایه و اساس آن‌را مباحث Binding تشکیل می‌دهد، بیشتر آشنا خواهیم شد.

انقیاد کنترل‌ها به یکدیگر در Silverlight

علاوه بر امکان مقید سازی اطلاعات به کنترل‌ها و یا برعکس (بر اساس حالات Binding ذکر شده)، امکان مقید سازی کنترل‌ها به یکدیگر نیز در Silverlight وجود دارد. برای نمونه در مثال زیر، کنترل TextBlock به کنترل TextBox مقید شده است. به این صورت با ورود هر عبارتی در TextBox ، بلافاصله عبارت وارد شده در TextBlock نیز نمایش داده می‌شود:

XAML

```
<StackPanel>
  <TextBox Margin="10" Name="txtData" />
  <TextBlock Margin="10"
             Text="{Binding ElementName=txtData,
                            Path=Text}" />
</StackPanel>
```

در اینجا ElementName نام کنترلی است که قرار است اطلاعات از آن دریافت شود و Path ، نام خاصیت مطلوب آن کنترل را مشخص می‌سازد.
و یا برای نمونه در کدهای XAML بعد، تنها در صورتیکه CheckBox انتخاب شده باشد، TextBox مقید به آن فعال خواهد شد.

XAML

```
<StackPanel>
```

```
    <CheckBox Name="chkNeed" Content="Need Data?" />
    <TextBox IsEnabled="{Binding ElementName=chkNeed, Path=IsChecked}" />
</StackPanel>
```

انقیاد داده‌ها و مبحث تبدیل اطلاعات

گاهی از اوقات نیاز است تا اطلاعات برنامه را پیش از نمایش نهایی آن‌ها به قالب مناسبی تبدیل نمود. برای مثال اگر اطلاعات در حال انقیاد شامل مسیر یک سری فایل تصویری است، بهتر است این مسیرها را پیش از نمایش، تبدیل به اطلاعات تصویر اصلی نمود؛ یا برای مثال تاریخ ذخیره شده در یک بانک اطلاعاتی، میلادی است و شما قصد نمایش آن‌را به صورت تاریخ شمسی دارید. به همین جهت یکی دیگر از توانایی‌های Binding در Silverlight فراهم آوردن امکان تبدیل اطلاعات پیش از نمایش آن‌ها است. برای این منظور تنها کافی است کلاسی که قرار است منطق این تبدیلات را ارائه دهد، اینترفیس استاندارد IValueConverter را پیاده سازی نماید. تعریف عمومی متدهای این کلاس به شکل زیر است:

C#

```
using System;
using System.Windows.Data;

namespace SilverlightApplication47
{
    public class FarenheitToCelciusConverter : IValueConverter
    {
        public object Convert(object value, Type targetType,
               object parameter,
               System.Globalization.CultureInfo culture)
        {
            //...
        }

        public object ConvertBack(object value, Type targetType,
               object parameter,
               System.Globalization.CultureInfo culture)
        {
            //...
        }
    }
}
```

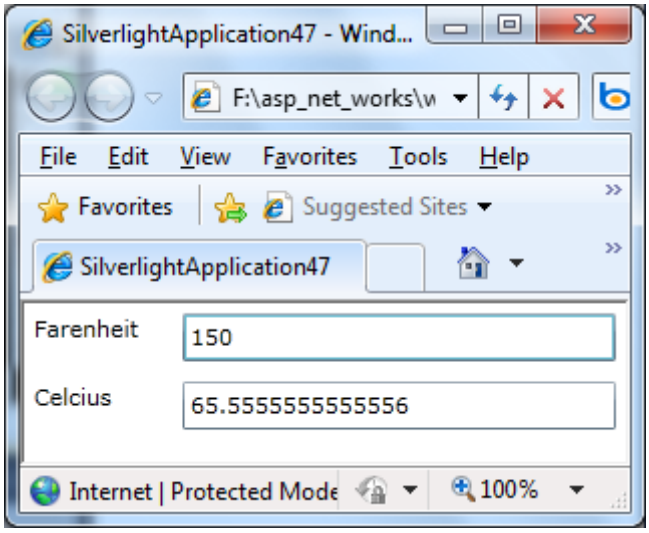
همانطور که ملاحظه می‌نمائید دو متد اصلی Convert و ConvertBack کار تبدیل اطلاعات را انجام خواهند داد. در متد Convert ، پارامتر value دریافت و سپس تبدیلات مناسب روی آن صورت گرفته و نهایتا بازگشت

داده خواهد شد. مقدار بازگشت داده شد، مقداری است که به کاربر نمایش داده می‌شود. سیلان اطلاعات در متد Convert از منبع به مقصد است و در متد ConvertBack برعکس آن می‌باشد (برای مثال کاربر اطلاعاتی را وارد نموده است. در حالت انقیاد دو طرفه نیاز است تا منبع داده، آن‌را با فرمت اصلی دریافت کرده و به روز شود). در اکثر موارد نیازی به پیاده سازی ConvertBack نبوده و تنها ارائه‌ی آن به صورت ذیل کفایت می‌کند. برای مثال تبدیل مسیر یک فایل تصویری به تصویر معادل آن مطلوب است اما عکس این عملیات عموما نیاز نخواهد بود:

C#

```
public object ConvertBack(object value, Type targetType,
       object parameter,
       System.Globalization.CultureInfo culture)
{
    throw new  NotImplementedException();
}
```

در ادامه در طی یک مثال کاربردی، به بررسی تبدیل اطلاعات در حین انقیاد آن‌ها خواهیم پرداخت. مثال بعد کار تبدیل واحدهای دما را انجام می‌دهد (شکل ۶).



شکل ۶– نمایی از برنامه‌ی انقیاد داده‌ها به همراه تبدیل اطلاعات

کدهای کلاس تبدیل کننده‌ی دما در این مثال با توجه به پیاده سازی اینترفیس IValueConverter به شرح بعد می‌باشند :

FarenheitToCelciusConverter.cs

```
using System;
using System.Windows.Data;
```

```
namespace SilverlightApplication47
{
    public class FarenheitToCelciusConverter : IValueConverter
    {
        public object Convert(object value, Type targetType,
               object parameter,
               System.Globalization.CultureInfo culture)
        {
            string sourceValue = value.ToString();
            double decimalValue;
            if (Double.TryParse(sourceValue, out decimalValue))
            {
                if (parameter.ToString() == "format1")
                {
                    //do something...
                }
                return (decimalValue - 32.0) * (5.0 / 9.0);
            }
            return value;
        }

        public object ConvertBack(object value, Type targetType,
               object parameter,
               System.Globalization.CultureInfo culture)
        {
            string sourceValue = value.ToString();
            double decimalValue;
            if (Double.TryParse(sourceValue, out decimalValue))
            {
                return (decimalValue * (9.0 / 5.0)) + 32.0;
            }
            return value;
        }
    }
}
```

و نحوه‌ی استفاده از این کلاس در کدهای XAML برنامه به شرح زیر هستند :

MainPage.xaml

```
<UserControl x:Class="SilverlightApplication47.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"
    xmlns:local="clr-namespace:SilverlightApplication47"
    mc:Ignorable="d" d:DesignHeight="300" d:DesignWidth="400">
    <UserControl.Resources>
        <local:FarenheitToCelciusConverter
```

```
                                x:Key="myTemperatureConverter" />
    </UserControl.Resources>
    <Grid x:Name="LayoutRoot" Background="White">
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto" />
            <RowDefinition Height="*" />
        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="99" />
            <ColumnDefinition Width="301*" />
        </Grid.ColumnDefinitions>
        <TextBlock Margin="5" Text="Farenheit"  />
        <TextBlock Grid.Row="1" Margin="5"
                 Text="Celcius" VerticalAlignment="Top" />
        <TextBox Grid.Column="1" Margin="5" Name="txtFarenheit"/>
        <TextBox Grid.Column="1" Grid.Row="1"
                 Margin="5" Name="txtCelcius"
                 VerticalAlignment="Top"
                 Text="{Binding Path=Text, ElementName=txtFarenheit,
                       Mode=TwoWay,
                       Converter =
                       {StaticResource myTemperatureConverter},
                       ConverterParameter = 'format1' }"
                 />
    </Grid>
</UserControl>
```

چند نکته‌ی زیر در حین استفاده از تبدیل کننده‌ی تعریف شده حائز اهمیت می‌باشند:

۱. ابتدا فضای نام دربرگیرنده‌ی کلاس تبدیل کننده به همراه نام مستعار آن به فایل XAML جاری اضافه می‌شوند.
۲. سپس کلاس تبدیل کننده به صورت یک منبع ایستا (Static Resource) تعریف خواهد شد.
۳. اکنون می‌توان جهت استفاده از آن در قسمت تعاریف Binding از خواص Converter و ConverterParameter استفاده کرد. کار خاصیت ConverterParameter ارسال یک پارامتر دلخواه به متدهای تبدیل کننده است. برای مثال مشخص سازی فرمتی خاص.

لازم به ذکر است که در Silverlight 4 جهت سهولت تعریف تبدیل کنندهایی که صرفا قالب خاصی را به اطلاعات ارائه شده می‌بخشند، خاصیت StringFormat نیز به امکانات Binding اضافه شده است. برای مثال فرمت دهی به تاریخ ارائه شده، یا اعداد یکی از گزارشات برنامه :

XAML

```
<TextBox Text="{Binding ReleaseDate, StringFormat='MMM dd, yyyy', Mode=TwoWay}" />
<TextBlock Text="{Binding Price, StringFormat='c'}" />
```