Silverlight 4

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

فهرست مطالب

فصل ۱۱- آشنایی با MVVM Light Toolkit

سایر کتابخانه‌ها و Framework های موجود MVVM

در آدرس زیر لیست نسبتا به روزی از انواع و اقسام کتابخانه‌ها و Framework های تهیه شده برای MVVM را می‌توانید مشاهده نمائید:

http://www.japf.fr/silverlight/mvvm/index.html

در این بین کتابخانه‌ی MVVM Light Toolkit از لحاظ امکانات، محبوبیت و همچنین مستندات در صدر قرار دارد و از همه مهمتر تنها به WPF محدود نبوده و پشتیبانی از Silverlight و Windows Phone 7 را نیز لحاظ نموده است (شکل زیر) و با دو نسخه‌ی ۲۰۰۸ و ۲۰۱۰ ویژوال استودیو سازگاری دارد. Blendability نیز یکی از اهداف و نکات مثبت آن است.

Name		Popularity	Silverlight	Hosting	License	Documentation	Features
MVVM Light Toolkit	L	★★★★★	☑	CodePlex	MIT license	★★★★★	★★★★☆

شکل ۱– وضعیت کتابخانه‌ی MVVM Light Toolkit در بین سایر کتابخانه‌های موجود

کتابخانه‌ی MVVM Light Toolkit سورس باز بوده و از آدرس ذیل قابل دریافت است:

http://mvvmlight.codeplex.com/

نصب قالب‌های MVVM Light Toolkit مخصوص VS.Net 2008

ابتدا محل نصب قالب‌های VS.Net را باید یافت. این محل در VS.Net در آدرس زیر قابل مشاهده است:

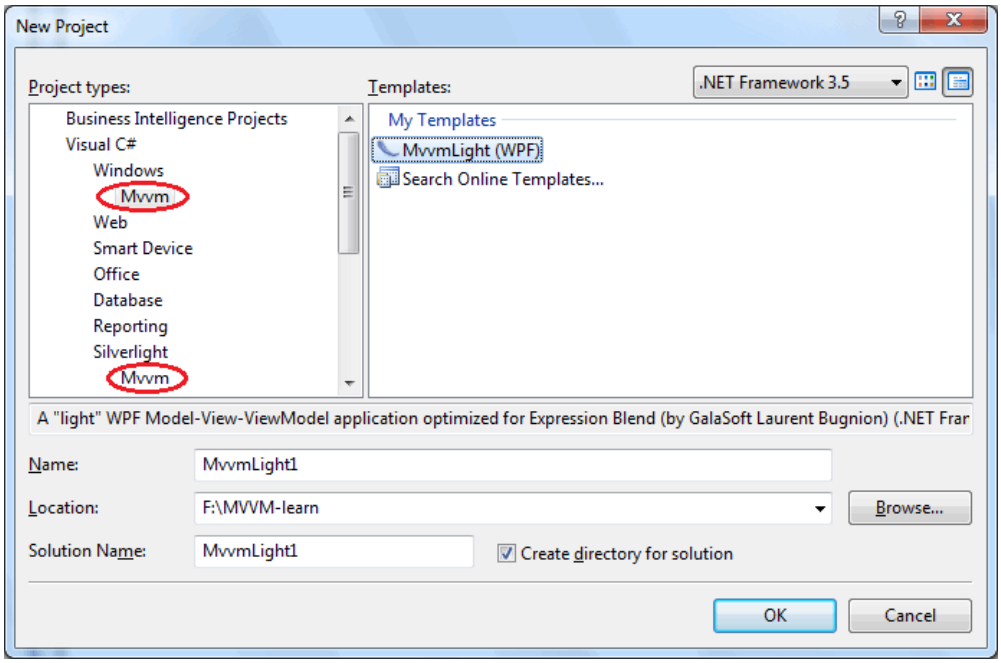
Tools Menu → Options → Projects and Solutions

برای مثال مسیر ریشه اصلی آن همانند آدرس ذیل باید باشد:

C:\Users\Vahid\Documents\Visual Studio 2008\Templates

در ادامه فایل GalaSoft.MvvmLight.**Templates**.Vn.**VS08**.zip را گشوده (یکی از فایل‌های دریافت شده مجموعه‌ی MVVM Light Toolkit است) و در مسیر ریشه یاد شده کپی نمائید.

اکنون اگر به صفحه‌ی ایجاد یک پروژه جدید در VS.Net 2008 مراجعه کنیم، قالب‌های اضافه شده قابل مشاهده هستند (شکل ۲).



شکل ۲– قالب‌های MVVM Light Toolkit مخصوص VS.Net 2008 .

نصب قالب‌های MVVM Light Toolkit مخصوص VS.Net 2010

برنامه VS.Net 2010 از پروژه‌های WPF3 ، WPF4 ، SL3 ، SL4 و Windows Phone 7 پشتیبانی می‌کند.

در اینجا نیز ابتدا باید مسیر پوشه‌ی ریشه‌ی قالب‌های پروژه‌های VS.Net 2010 را یافت. این محل در VS.Net در آدرس زیر قابل مشاهده است:

Tools Menu → Options → Projects and Solutions

برای مثال مسیر ریشه اصلی آن همانند آدرس ذیل باید باشد:

C:\Users\Vahid\Documents\Visual Studio 2010\Templates

در ادامه فایل GalaSoft.MvvmLight.**Templates**.Vn.**VS10**.zip را گشوده و در مسیر فوق کپی نمائید.

اکنون اگر به صفحه‌ی ایجاد یک پروژه جدید در VS.Net 2010 مراجعه کنیم، قالب‌های اضافه شده قابل مشاهده هستند (شکل ۳).

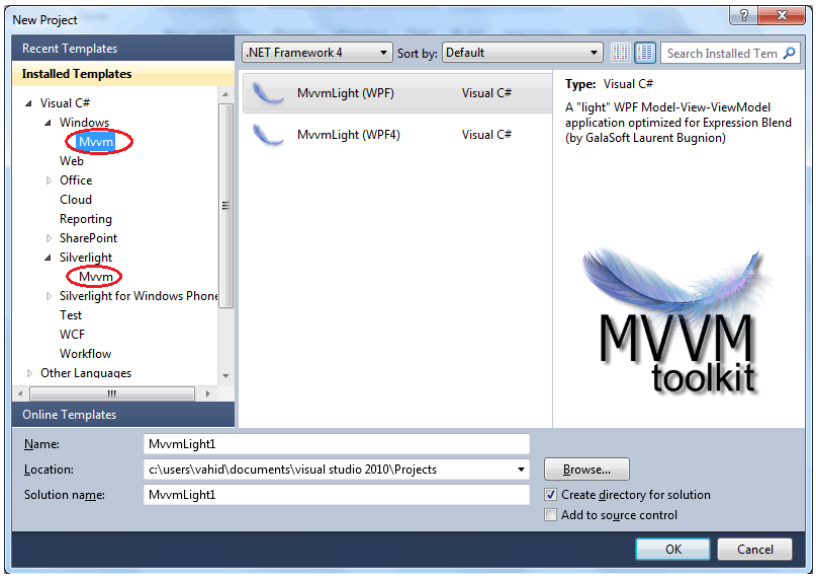
نصب Code Snippets مجموعه MVVM Light Toolkit در VS.Net 2008/2010

در هر دو نگارش ۲۰۰۸ و ۲۰۱۰ ویژوال استودیو دات نت، مسیر پوشه‌ی Code Snippets به صورت زیر مشخص می‌شود:

Tools Menu → Code Snippets Manager → My Code Snippets

پس از مراجعه به مسیر Tools Menu → Code Snippets Manager ، زبان انتخابی را بر روی C# یا VB قرار دهید تا قسمت My Code Snippets مشخص شود. سپس از قسمت location ، مسیر آن‌را کپی نمائید. برای مثال مسیر آن باید شبیه به آدرس زیر باشد:

C:\Users\Vahid\Documents\Visual Studio 2010\Code Snippets\Visual C#\My Code Snippets



شکل ۳– قالب‌های MVVM Light Toolkit برای VS.Net 2010 .

در ادامه فایل GalaSoft.MvvmLight.**Snippets**.Vn.zip را گشوده و در مسیر فوق کپی نمائید. اگر هر دو نگارش ۲۰۰۸ و ۲۰۱۰ را بر روی کامپیوتر خود دارید، برای هر دو نگارش این عملیات را تکرار نمائید.

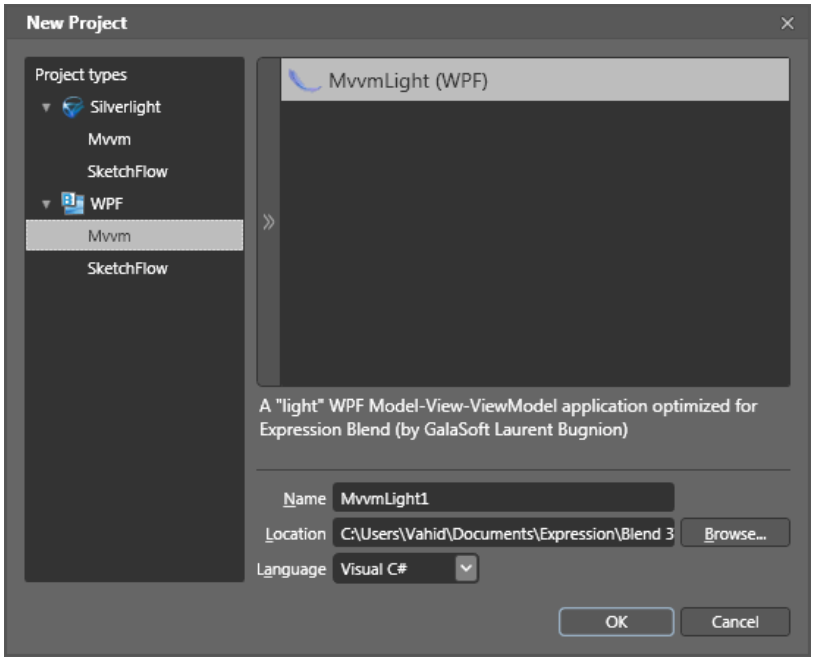
نصب فایل‌های باینری کتابخانه‌ی MVVM Light Toolkit

فایل GalaSoft.MvvmLight.**Binaries**.Vn.zip را گشوده و در درایو C کپی نمائید. پس از اینکار مسیر ذیل به صورت خودکار تشکیل خواهد شد:

C:\Program Files\Laurent Bugnion (GalaSoft)\Mvvm Light Toolkit\Binaries

نصب قالب‌های MVVM Light Toolkit مخصوص Expression Blend

نصب قالب‌های MVVM Light Toolkit مخصوص Expression Blend ساده است. فایل‌های GalaSoft.MvvmLight.Templates.Vn.**Blend3**.zip و GalaSoft.MvvmLight.Templates.Vn.**Blend4**.zip را یافته، گشوده و در پوشه‌ی My Documents کپی نمائید تا ساختار مرتبط به آن‌ها به صورت خودکار تشکیل شود. اکنون اگر یک پروژه جدید را در Expression Blend آغاز نمائیم (شکل ۴)، این قالب‌ها باید مشخص باشند.



شکل ۴- قالب‌های MVVM Light Toolkit مخصوص Expression Blend .

بررسی صحت نصب کتابخانه‌ی MVVM Light Toolkit

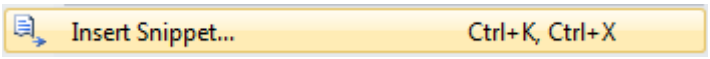
برای بررسی صحت نصب، یک پروژه جدید را با استفاده از قالب‌هایی که در مسیرهای یاد شده کپی کردیم، آغاز کرده و سپس به قسمت references آن مراجعه کنید. مطمئن شوید که اسمبلی‌های زیر بدون هیچگونه علامت خطایی به پروژه الحاق شده‌اند:

- GalaSoft.MvvmLight
- GalaSoft.MvvmLight.Extras
- System.Windows.Interactivity

تا اینجا کار نصب مقدماتی MVVM Light Toolkit و نحوه‌ی بررسی صحت نصب موارد عنوان شده به پایان می‌رسد. در ادامه چند مثال را در مورد نحوه‌ی استفاده از کلاس‌های کمکی آن بررسی خواهیم کرد.

استفاده از Code Snippets نصب شده

برای استفاده از Code snippets نصب شده، بر روی یک مکان خالی در کدهای صفحه کلیک راست کرده و از منوی ظاهر شده گزینه‌ی Insert snippets را انتخاب کنید (شکل زیر). Code snippets مرتبط با این جعبه ابزار MVVM در پوشه‌ی My code snippets قرار گرفته است. برای مثال به این صورت نحوه‌ی تعریف یک خاصیت کمک گیرنده از اینترفیس INotifyPropertyChanged که در اینجا به شکل مخفف INPC عنوان شده، بسیار ساده خواهد شد.



شکل ۵– نحوه‌ی مراجعه به پوشه‌ی Code Snippets نصب شده

مثال اول – بررسی RelayCommand

استفاده از رخدادها سبب تنیدگی رابط کاربر و کدهای برنامه خواهد شد، به همین جهت کنترل‌های بصری برنامه در WPF می‌توانند با ارائه‌ی خواص Command و bind آن به خواصی از نوع ICommand در ViewModel ، جداسازی لایه‌ها از یکدیگر را سبب شوند (به Commands ، weak events نیز گفته می‌شود). MVVM Light Toolkit با ارائه‌ی کلاس‌های RelayCommand و RelayCommand<T> کار با Commands را ساده‌تر کرده است. در مثال یک فصل جاری قصد داریم از این دو کلاس استفاده نمائیم.

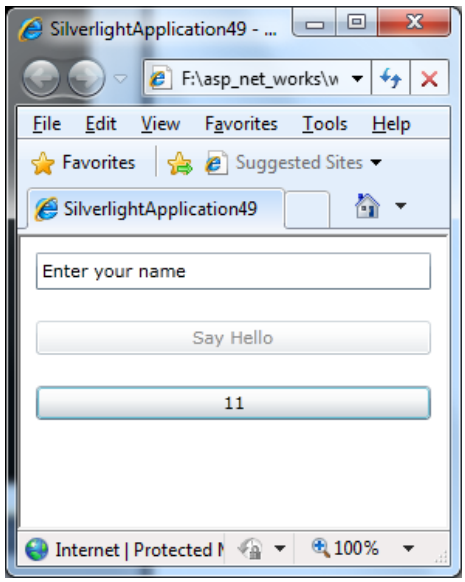
ساختار پوشه‌ها و نمایی از برنامه را در شکل‌های ۶ و ۷ مشاهده می‌نمائید. با کلیک بر روی دکمه‌ی Say Hello ، متن موجود در TextBox نمایش داده شده و با هر بار کلیک بر روی دکمه‌ی زیرین، برچسب دکمه یک واحد افزایش خواهد یافت. اگر این عدد فرد باشد، دکمه‌ی Say Hello غیرفعال می‌شود.

باید دقت داشت که هرچند این کتابخانه در حال حاضر تنها C# را پشتیبانی می‌کند ولی این موضوع اصلا اهمیتی نداشته و می‌توان ابتدا ساختار پوشه‌ها را به صورت دستی ایجاد نمود و سپس از مسیر زیر ارجاعات لازم به فایل‌های مرتبط را به پروژه جاری افزود.

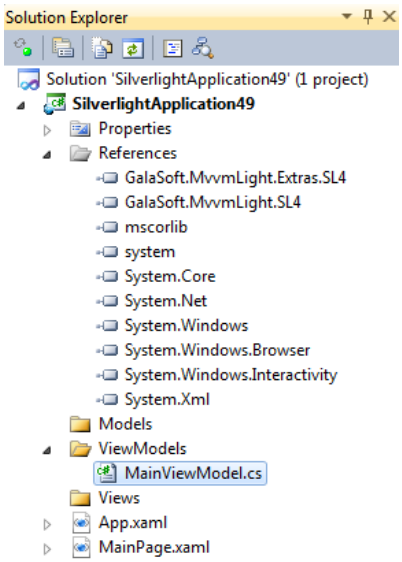
C:\Program Files\Laurent Bugnion (GalaSoft)\Mvvm Light Toolkit\Binaries\Silverlight4

در ادامه از منوی File گزینه‌ی Export template ، نسبت به ساخت یک قالب سفارشی برای استفاده‌های بعدی خود اقدام نمائید. پس از آن همانطور که در قسمت‌های قبل نیز ذکر شد، فایل تولیدی را در مسیر زیر کپی نمائید تا در حین ایجاد یک پروژه‌ی جدید در VS.NET 2010 به سادگی در دسترس و قابل استفاده باشد:

C:\Users\Vahid\Documents\Visual Studio 2010\Templates\ProjectTemplates\Silverlight\Mvvm



شکل ۶– نمایی از برنامه در حال اجرا



شکل ۷– ساختار پوشه‌های برنامه

View برنامه اول

در view برنامه ابتدا ارجاعی به فضای نام ViewModel برنامه اضافه شده و سپس DataContext آن به این ViewModel تنظیم گردیده است.

MainPage.xaml

```
<UserControl x:Class="SilverlightApplication49.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:vm="clr-namespace:SilverlightApplication49.ViewModels"
    mc:Ignorable="d" d:DesignHeight="300" d:DesignWidth="400">

    <UserControl.Resources>
        <vm:MainViewModel  x:Key="MainViewModel" />
    </UserControl.Resources>
```

```
    <UserControl.DataContext>
        <Binding Source="{StaticResource MainViewModel}" />
    </UserControl.DataContext>

    <StackPanel x:Name="LayoutRoot"
                Background="White">

        <TextBox Text="Enter your name"
                 x:Name="MyTextBox"
                 Margin="10" />

        <Button Content="Say Hello"
          Command="{Binding SayHelloCommand}"
          CommandParameter="{Binding ElementName=MyTextBox, Path=Text}"
                Margin="10" />

        <Button Content="{Binding Counter}"
                Command="{Binding IncreaseCounterCommand}"
                Margin="10" />
    </StackPanel>
</UserControl>
```

همانطور که در کدهای Xaml این View مشخص است، دکمه‌ی اول به SayHelloCommand در ViewModel مقید (bind) شده است و مقدار عبارت وارد شده در MyTextBox را به عنوان پارامتر ارسال خواهد نمود.

دکمه‌ی دوم به IncreaseCounterCommand مقید شده و برچسب آن از خاصیت Counter تامین می‌گردد.

برنامه اول ViewModel

در ViewModel برنامه، ابتدا دو فضای نام این Toolkit اضافه شده‌اند و سپس خواصی که در View مورد استفاده قرار می‌گیرند، به صورت Public معرفی گردیده‌اند.

MainViewModel.cs

```
using System.ComponentModel;
using System.Windows;
using GalaSoft.MvvmLight.Command;


namespace SilverlightApplication49.ViewModels
```

```
    {
public class MainViewModel : INotifyPropertyChanged
{
    #region Counter
    public const string CounterPropertyName = "Counter";

    private int _counter;

    public int Counter
    {
        get
        {
            return _counter;
        }

        set
        {
            if (_counter == value)
            {
                return;
            }

            _counter = value;
            RaisePropertyChanged(CounterPropertyName);
            SayHelloCommand.RaiseCanExecuteChanged();
        }
    }
    #endregion

    public RelayCommand<string> SayHelloCommand
    {
        get;
        private set;
    }

    public MainViewModel()
    {
        SayHelloCommand = new RelayCommand<string>(
            m => MessageBox.Show("Hello, " + m),
            m => _counter % 2 == 0);

        #region Hidden

        IncreaseCounterCommand = new RelayCommand(() =>
        {
            Counter++;
        });

        #endregion
    }
```

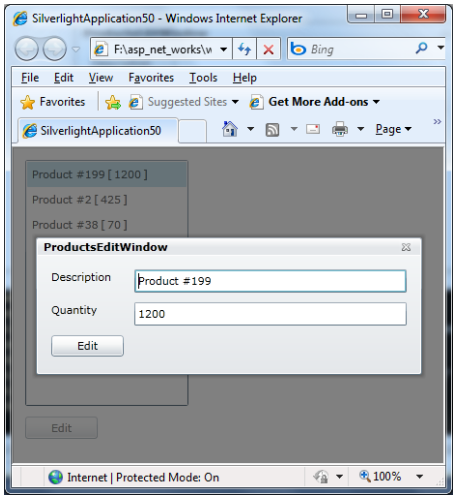
```
        #region INPC
        public event PropertyChangedEventHandler PropertyChanged;

        public void RaisePropertyChanged(string propertyName)
        {
            if (PropertyChanged != null)
            {
                PropertyChanged(this,
                     new PropertyChangedEventArgs(propertyName));
            }
        }
        #endregion

        #region Hidden

        public RelayCommand IncreaseCounterCommand
        {
            get;
            private set;
        }

        #endregion

    }
}
```

در این ViewModel از آنجائیکه تغییر در مقدار Counter باید در View منعکس گردد، اینترفیس INotifyPropertyChanged پیاده سازی گردیده است. سپس دو نوع Command را می‌توان مشاهده کرد (یک نوع ساده و یک نوع Generic آن‌را که CommandParameter دریافتی از View را پردازش می‌کند). آرگومان‌های سازنده‌ی RelayCommand ، به ترتیب execute و canExecute می‌باشند که در اینجا با کمک lambda expressions معرفی شده‌اند؛ بدیهی است در صورت تمایل می‌توان آن‌ها را با متدهای معادلی جایگزین کرد.

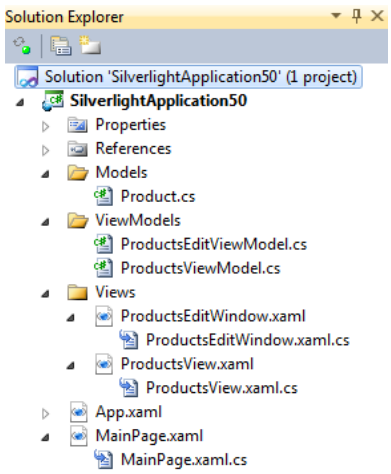
اگر نیاز به محاسبه‌ی مجدد شرط قسمت canExecute وجود داشت باید با کمک متد RaiseCanExecuteChanged همانند SayHelloCommand.RaiseCanExecuteChanged معرفی شده، عمل نمود (فراخوانی این مورد در برنامه‌های سیلورلایت ضروری است، زیرا برخلاف WPF ، از CommandManager محروم است و حالات رخدادها را باید به این صورت مدیریت نمود. در WPF ، این محاسبه‌ی مجدد به صورت خودکار صورت می‌گیرد و نیازی به فراخوانی RaiseCanExecuteChanged نیست).

مثال دوم – بررسی Messenger

گاهی از اوقات نیاز می‌شود تا View ها یا ViewModels برنامه با یکدیگر در ارتباط باشند اما نمی‌خواهیم آن‌ها را به یکدیگر گره زده و مشکلات نگهداری و تهیه‌ی آزمون‌های خودکار را پدید آوریم. برای نمونه در مثال دوم فصل جاری (شکل‌های ۸ و ۹)، قصد داریم لیستی از محصولات را نمایش داده و پس از انتخاب یکی از آن‌ها توسط کاربر و کلیک بر روی دکمه‌ی ویرایش، صفحه‌ی جدید ویرایش محصول نمایان گردد. نمایش یک صفحه جدید از داخل یک ViewModel برخلاف اصول M-V-VM است که می‌گوید ViewModel نباید هیچ ارجاعی از View ها یا کنترل‌های بصری را در خود نگه داری کند. برای این منظور MVVM Light Toolkit کلاس Messenger را تدارک دیده است. در اینجا یک کلاس، پیغامی را به همراه شیءایی ارسال کرده و کلاس دیگر می‌تواند به این ارسال گوش فراداده و عکس العمل نشان دهد (این کلاس درحقیقت الگوی Mediator بین دو شیء را بدون داشتن ارجاعی از هر کدام پیاده سازی می‌کند).



شکل ۸– نمایی از برنامه دوم فصل در حال اجرا



شکل ۹– ساختار پوشه‌ها و فایل‌های برنامه دوم فصل

مدل برنامه دوم

مدل برنامه‌ی دوم فصل، کلاس محصولات می‌باشد که از دو خاصیت عمومی توضیح و تعداد آن محصول تشکیل شده است و کدهای آن‌را در ادامه ملاحظه می‌نمائید (Product.cs):

Product.cs

```
using System.ComponentModel;

namespace SilverlightApplication50.Models
{
    public class Product : INotifyPropertyChanged
```

```
    {
        private int _quantity;
        public int Quantity
        {
            set
            {
                _quantity = value;
                if (PropertyChanged == null) return;
                onPropertyChanged("Quantity");
            }
            get { return _quantity; }
        }

        private string _description;
        public string Description
        {
            set
            {
                _description = value;
                if (PropertyChanged == null) return;
                onPropertyChanged("Description");
            }
            get { return _description; }
        }

        #region INotifyPropertyChanged Members

        public event PropertyChangedEventHandler PropertyChanged;
        private void onPropertyChanged(string propertyName)
        {
            if (PropertyChanged == null) return;
            PropertyChanged(this,
               new PropertyChangedEventArgs(propertyName));
        }
        #endregion
    }
}
```

ViewModel ها و View های متناظر برنامه دوم

برنامه دوم از دو View به همراه دو ViewModel تشکیل شده است. کدهای کلاس ProductsViewModel آن را در ادامه مشاهده خواهید کرد.

از طریق خاصیت عمومی Items یک سری محصول در لیست نمایشی برنامه ظاهر خواهند شد. EditCommand جهت عکس العمل نشان دادن به کلیک بر روی دکمه‌ی Edit در View متناظر این ViewModel ایجاد شده است.

ProductsViewModel.cs

```
using System.Collections.ObjectModel;
using GalaSoft.MvvmLight.Command;
using GalaSoft.MvvmLight.Messaging;
using SilverlightApplication50.Models;

namespace SilverlightApplication50.ViewModels
{
    public class ProductsViewModel
    {
        public ProductsViewModel()
        {
            //Add some dummy products
            Items = new ObservableCollection<Product>
              {
                new Product { Description="Product #1", Quantity=12},
                new Product { Description="Product #2", Quantity=42},
                new Product { Description="Product #3", Quantity=7}
              };

            //Respond to the events
            EditCommand = new RelayCommand<Product>(
                product => //selected item
                {
                    if (product == null) return;
                    Messenger.Default.Send(product, "edit product");
                });
        }

        public ObservableCollection<Product> Items { get; private set; }

        public RelayCommand<Product> EditCommand
        {
            get;
            private set;
        }
    }
}
```

نکته‌ی جدید این ViewModel سطر مربوط به Messenger.Default.Send است که توسط آن یک شیء انتخابی با یک کلید منحصربفرد (آرگومان دوم آن) ارسال خواهد شد. هر گوش فرا دهنده‌ای که مشترک دریافت

پیغام‌های این کلید منحصربفرد شده باشد، پیغام‌های آن‌را دریافت خواهد کرد و سایر گوش فرا دهنده‌ها با کلیدهای دیگری غیر از آن، این پیغام را دریافت نمی‌نمایند.

نکته‌ی دیگری که در اینجا سبب کاهش کد نویسی ما گردیده است، RelayCommand از نوع Product است. به این صورت CommandParameter تعریف شده در View کار ارسال این شیء انتخابی را عهده دار خواهد شد و به این صورت نیازی به تعریف یک خاصیت عمومی مجزا جهت دریافت آیتم انتخاب شده، نیست (همانند مثال فصل دوم).

کدهای View (ProductsView.xaml) متناظر با این ViewModel به شرح زیر می‌باشد:

ProductsView.xaml

```
<UserControl x:Class="SilverlightApplication50.Views.ProductsView"
    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:vm="clr-namespace:SilverlightApplication50.ViewModels"
    mc:Ignorable="d" d:DesignHeight="300" d:DesignWidth="400">

    <UserControl.Resources>
        <vm:ProductsViewModel x:Key="viewModel" />
    </UserControl.Resources>
    <Grid DataContext="{Binding Source={StaticResource viewModel}}">
        <ListBox Height="254"
                 HorizontalAlignment="Left" Margin="12,12,0,0"
                 x:Name="lstItems"
                 ItemsSource="{Binding Items}"
                 VerticalAlignment="Top" Width="169" >
            <ListBox.ItemTemplate>
                <DataTemplate>
                    <StackPanel Orientation="Horizontal">
                        <TextBlock Margin="2"
                                   Text="{Binding Description}" />
                        <TextBlock Margin="0 2"
                                   Text="[" />
                        <TextBlock Margin="2"
                                   Text="{Binding Quantity}" />
                        <TextBlock Margin="0 2"
                                   Text="]" />
                    </StackPanel>
                </DataTemplate>
            </ListBox.ItemTemplate>
        </ListBox>
        <Button Content="Edit" Height="23"
                HorizontalAlignment="Left" Margin="12,277,0,0"
```

```
                    Command="{Binding EditCommand}"
                CommandParameter=
                "{Binding ElementName=lstItems, Path=SelectedItem}"
                VerticalAlignment="Top" Width="75" />
    </Grid>
</UserControl>
```

ProductsView اولین View مجموعه جاری است که دارای Code behind می‌باشد و این مورد جزو موارد مجاز کد نویسی در Code behind یک View به شمار می‌رود:

ProductsView

```
using GalaSoft.MvvmLight.Messaging;
using SilverlightApplication50.Models;
namespace SilverlightApplication50.Views
{
    public partial class ProductsView
    {
        public ProductsView()
        {
            Messenger.Default.Register<Product>(this,
                "edit product",
                x =>
                {
                    var selectedItem = x;
                    if (selectedItem == null) return;
                    var win1 = new ProductsEditWindow();
                    Messenger.Default.Send(selectedItem, "doEdit");
                    win1.Show();
                }
                );
            InitializeComponent();
        }
    }
}
```

تنها کدی که در اینجا تعریف شده است، Messenger.Default.Register می‌باشد که به پیغام‌های رسیده edit product گوش فرا می‌دهد. اگر پیغامی با این شرط دریافت شد، پنجره جدید ویرایش اطلاعات را ایجاد کرده، آیتم انتخابی را به ViewModel آن ارسال می‌کند (Messenger.Default.Send با نام doEdit) و سپس پنجره را نمایش خواهد داد. به این صورت تبادل اطلاعاتی را بدون گره خوردگی لایه‌های برنامه به یکدیگر، خواهیم داشت.
از این ViewModel در MainWindow برنامه به صورت بعد استفاده می‌شود:

MainPage.xaml

```
<UserControl x:Class="SilverlightApplication50.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:vm="clr-namespace:SilverlightApplication50.Views"
    mc:Ignorable="d" d:DesignHeight="300" d:DesignWidth="400">
    <Grid x:Name="LayoutRoot" Background="White">
        <vm:ProductsView />
    </Grid>
</UserControl>
```

ViewModel صفحه ویرایش اطلاعات (ProductsEditViewModel.cs) به شرح زیر است:

ProductsEditViewModel.cs

```
using System.ComponentModel;
using GalaSoft.MvvmLight.Command;
using GalaSoft.MvvmLight.Messaging;
using SilverlightApplication50.Models;

namespace SilverlightApplication50.ViewModels
{
    public class ProductsEditViewModel : INotifyPropertyChanged
    {
        public ProductsEditViewModel()
        {
            EditCommand = new RelayCommand(
                () =>
                    Messenger.Default.Send("CloseEdit")
                    );

            Messenger.Default.Register<Product>(this,
               "doEdit",
                x =>
                {
                    SelectedProduct = x;
                });
        }

        private Product _selectedProduct;
        public Product SelectedProduct
        {
            set
            {
                _selectedProduct = value;
                if (PropertyChanged == null) return;
                onPropertyChanged("SelectedProduct");
```

```
            }
        get { return _selectedProduct; }
    }

    public RelayCommand EditCommand
    {
        get;
        private set;
    }

    #region INotifyPropertyChanged Members

    public event PropertyChangedEventHandler PropertyChanged;
    private void onPropertyChanged(string propertyName)
    {
        if (PropertyChanged == null) return;
        PropertyChanged(this,
              new PropertyChangedEventArgs(propertyName));
    }
    #endregion
  }
}
```

این ViewModel یک خاصیت عمومی به نام SelectedProduct را جهت bind به کنترل‌های TextBox و EditCommand را جهت بستن این پنجره ویرایش در اختیار View متناظر خود قرار می‌دهد.

همانطور که ملاحظه می‌کنید این ViewModel در سازنده‌ی خود با کمک کلاس Messenger به پیغام‌های doEdit برنامه گوش فرا می‌دهد و در صورت دریافت پیغامی، آن‌ها را به SelectedProduct کلاس انتساب خواهد داد. EditCommand برنامه جهت ارسال پیغام بسته شدن پنجره تعریف شده است.

کدهای View (ProductsEditWindow.xaml) متناظر با این کلاس به شرح زیر می‌باشد:

ProductsEditWindow.xaml

```
<controls:ChildWindow
x:Class="SilverlightApplication50.Views.ProductsEditWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:controls="clr-namespace:System.Windows.Controls;assembly=System.Windows.Controls"
xmlns:vm="clr-namespace:SilverlightApplication50.ViewModels"
           Width="400" Height="142"
           Title="ProductsEditWindow">
    <controls:ChildWindow.Resources>
        <vm:ProductsEditViewModel x:Key="viewModel" />
    </controls:ChildWindow.Resources>

    <Grid x:Name="LayoutRoot" Margin="2"
          DataContext="{Binding Source={StaticResource viewModel}}">
```

```
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="86" />
            <ColumnDefinition Width="*" />
        </Grid.ColumnDefinitions>
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="Auto" />
            <RowDefinition Height="Auto" />
        </Grid.RowDefinitions>

        <TextBlock Text="Description"  HorizontalAlignment="Left"
                   Margin="5" VerticalAlignment="Top" />
        <TextBlock Text="Quantity" Grid.Row="1"
                   HorizontalAlignment="Left"
                   Margin="5" VerticalAlignment="Top" />
        <TextBox Grid.Column="1"
            Text="{Binding SelectedProduct.Description, Mode=TwoWay}"
            HorizontalAlignment="Stretch" Margin="5"
            VerticalAlignment="Top" Width="auto" />
        <TextBox Grid.Column="1"
           Text="{Binding SelectedProduct.Quantity, Mode=TwoWay}"
           Grid.Row="1"  HorizontalAlignment="Stretch" Margin="5"
           VerticalAlignment="Top" Width="auto" />

        <Button Content="Edit"
              Grid.Row="2" Command="{Binding EditCommand}"
              HorizontalAlignment="Left" Margin="5"
              VerticalAlignment="Top" Width="75" />
    </Grid>
</controls:ChildWindow>
```

این View نیز دارای کدهای مختصری است که تنها کار بستن پنجره را عهده دار هستند و به پیغام‌های CloseEdit برنامه گوش فرا می‌دهند:

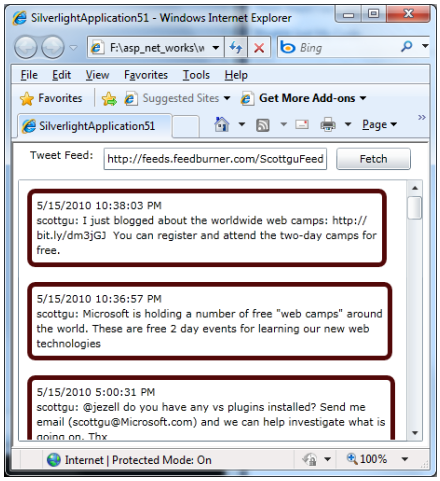
ProductsEditWindow.xaml.cs

```
using GalaSoft.MvvmLight.Messaging;

namespace SilverlightApplication50.Views
{
    public partial class ProductsEditWindow
    {
        public ProductsEditWindow()
        {
            Messenger.Default.Register<string>(
                "CloseEdit",
                x => this.Close());
```

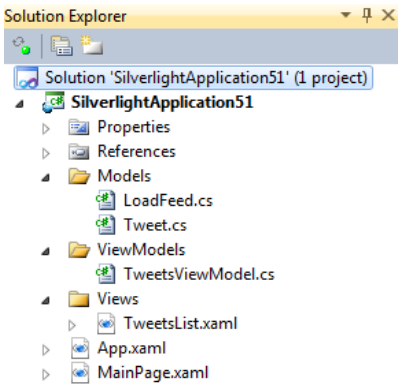
```
            InitializeComponent();
        }
    }
}
```

مثال سوم – بررسی Blendability

در این مثال (شکل‌های ۱۰ و ۱۱) قصد داریم اطلاعات Tweet های Scottgu (یکی از مدیران ارشد تیم .NET Framework) را از یک Feed تهیه شده نمایش دهیم. مشکلی که این نوع مسایل دارند، وابستگی به یک سرویس خارجی است که ممکن است در حین طراحی در دسترس نباشد یا کار کردن با آن سبب کندی طراحی گردد. به همین جهت نیاز است تا بتوان برای حالت طراحی (چه در طراح ویژوال استودیو و یا در Expression Blend) یک سری اطلاعات آزمایشی را تولید کرد. در MVVM Light Toolkit با کمک خاصیت IsInDesignModeStatic می‌توان تشخیص داد که آیا برنامه در حال اجرا است و یا در حال طراحی و سپس بر این اساس می‌توان برای ViewModel های خود داده زمان طراحی را ایجاد نمود.



شکل ۱۰– نمایی از برنامه سوم در حال اجرا



شکل ۱۱– ساختار پوشه‌ها و فایل‌های برنامه سوم

مدل برنامه سوم

ساختار یک Tweet را با کمک کلاس بعد می‌توان نمایش داد:

Tweet.cs

```
using System;
```

```
namespace SilverlightApplication51.Models
{
    public class Tweet
    {
        public DateTime CreatedAt { get; set; }
        public string Text { get; set; }
    }
}
```

سپس نیاز است تا بتوان اطلاعات RSS Feed مربوطه را دریافت و لیستی از Tweets را تهیه نمود. در Silverlight امکان خواندن اطلاعات Feed های استاندارد به صورت توکار وجود دارد که نحوه‌ی استفاده از آن را در کلاس بعد مشاهده می‌نمائید (ارجاعی به اسمبلی استاندارد System.ServiceModel.Syndication نیاز خواهد بود):

LoadFeed.cs

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Net;
using System.ServiceModel.Syndication;
using System.Xml;
//add a ref. to System.ServiceModel.Syndication

namespace SilverlightApplication51.Models
{
    public class LoadFeed
    {

        public IList<Tweet> TweetList { set; get; }
        public event EventHandler LoadComplete;

        public void LoadTweets(string url)
        {
            var wc = new WebClient();
            wc.OpenReadCompleted += wc_OpenReadCompleted;
            var feedUri = new Uri(url, UriKind.Absolute);
            wc.OpenReadAsync(feedUri);
        }

        private void wc_OpenReadCompleted(object sender,
                    OpenReadCompletedEventArgs e)
        {
            if (e.Error != null)
            {
                //Error in Reading Feed. Try Again later!!
                return;
```

```
            }

        using (Stream s = e.Result)
        {
            SyndicationFeed feed;
            TweetList = new List<Tweet>();

            using (XmlReader reader = XmlReader.Create(s))
            {
                feed = SyndicationFeed.Load(reader);
                foreach (SyndicationItem item in feed.Items)
                {
                    TweetList.Add(
                        new Tweet
                            {
                            CreatedAt = item.PublishDate.DateTime,
                            Text = item.Summary.Text
                            }
                        );
                }
            }
        }

        LoadComplete.Invoke(sender,e);
    }
  }
}
```

برنامه سوم ViewModel

جهت نمایش لیستی از آیتم‌های Tweet دریافت شده، خاصیت عمومی TweetItems در اختیار View برنامه قرار خواهد گرفت. همچنین یک RelayCommand از نوع string نیز تعریف شده است که کار واکنش نشان دادن به کلیک بر روی دکمه‌ی Fetch ، دریافت آدرس وارد شده در TextBox مرتبط و سپس اجرای عملیات دریافت اطلاعات Feed را برعهده دارد.

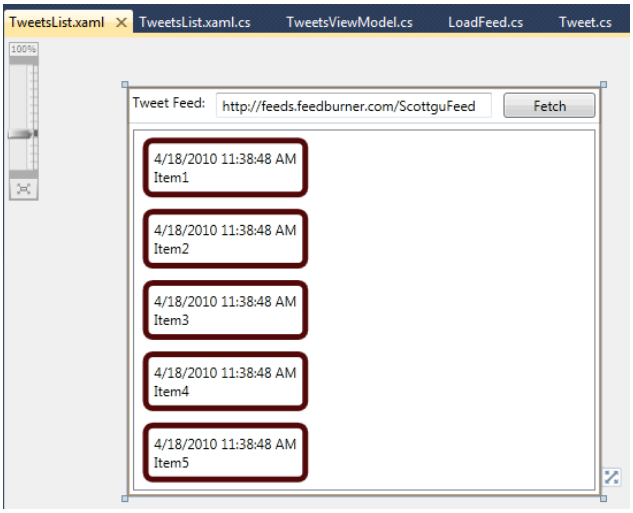
TweetsViewModel.cs

```
using System;
using System.Collections.ObjectModel;
using System.ComponentModel;
using GalaSoft.MvvmLight;
using GalaSoft.MvvmLight.Command;
using SilverlightApplication51.Models;

namespace SilverlightApplication51.ViewModels
{
```

```
public class TweetsViewModel : INotifyPropertyChanged
{
    private readonly ObservableCollection<Tweet> _tweetItems =
        new ObservableCollection<Tweet>();

    public ObservableCollection<Tweet> TweetItems
    {
        get
        {
            return _tweetItems;
        }
    }

    public TweetsViewModel()
    {
        DoFetch = new RelayCommand<string>(loadTweetItems);

        if (ViewModelBase.IsInDesignModeStatic)
        {
          _tweetItems.Add(new Tweet {
                Text = "Item1", CreatedAt = DateTime.Now });
            _tweetItems.Add(new Tweet {
                 Text = "Item2", CreatedAt = DateTime.Now });
            _tweetItems.Add(new Tweet {
                 Text = "Item3", CreatedAt = DateTime.Now });
            _tweetItems.Add(new Tweet {
                 Text = "Item4", CreatedAt = DateTime.Now });
            _tweetItems.Add(new Tweet {
                 Text = "Item5", CreatedAt = DateTime.Now });
        }
    }

    public RelayCommand<string> DoFetch { get; private set; }

    private LoadFeed _loadFeed;
    void loadTweetItems(string url)
    {
        if (string.IsNullOrEmpty(url)) return;

        BusyIndicatorIsBusy = true;

        _loadFeed = new LoadFeed();
        _loadFeed.LoadComplete += tweetItems_LoadComplete;
        _loadFeed.LoadTweets(url);
    }

    void tweetItems_LoadComplete(object sender, EventArgs e)
    {
```

```
            if (_loadFeed.TweetList == null) return;
            _tweetItems.Clear();
            foreach (var tweetItem in _loadFeed.TweetList)
            {
                _tweetItems.Add(tweetItem);
            }

            BusyIndicatorIsBusy = false;
        }

        private bool _busyIndicatorIsBusy;
        public bool BusyIndicatorIsBusy
        {
           set
            {
                _busyIndicatorIsBusy = value;
                if (PropertyChanged == null) return;
                onPropertyChanged("BusyIndicatorIsBusy");
            }
            get { return _busyIndicatorIsBusy; }
        }

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

تنها نکته‌ی جدید این ViewModel استفاده از خاصیت ViewModelBase.IsInDesignModeStatic می‌باشد. با کمک این خاصیت، برای مشاهده بهتر View در حالت طراحی، یک سری اطلاعات پیش فرض کمکی تهیه می‌شود. به این صورت کار کردن با این View در حالت طراحی که اطلاعات اصلی خود را باید از یک سرویس خارجی دریافت کند بسیار ساده خواهد شد (شکل بعد).



شکل ۱۲– نمایی از حالت طراحی View برنامه سوم به همراه نمایش داده‌های آزمایشی حالت طراحی.

View برنامه سوم

کدهای View متناظر با ViewModel برنامه به شرح زیر می‌باشند:

TweetsList.xaml

```
<UserControl x:Class="SilverlightApplication51.Views.TweetsList"
    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:vm="clr-namespace:SilverlightApplication51.ViewModels"
    mc:Ignorable="d" d:DesignHeight="300" d:DesignWidth="400"
    xmlns:toolkit="http://schemas.microsoft.com/winfx/2006/xaml/presentation/toolkit">
    <UserControl.Resources>
        <vm:TweetsViewModel x:Key="viewModel" />
    </UserControl.Resources>
    <Grid DataContext="{Binding Source={StaticResource viewModel}}"
          MinHeight="100" MinWidth="350">
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto" />
            <RowDefinition Height="*" />
        </Grid.RowDefinitions>
        <toolkit:BusyIndicator
             VerticalAlignment="Center" HorizontalAlignment="Center"
            IsBusy="{Binding BusyIndicatorIsBusy}" DisplayAfter="0">
            <StackPanel Grid.Row="0" Orientation="Horizontal">
                <TextBlock Margin="5" Text="Tweet Feed:"/>
                <TextBox Margin="5" Name="txtUrl" Width="240"
                     Text="http://feeds.feedburner.com/ScottguFeed"/>
```

```
                    <Button Margin="5" Content="Fetch" Width="80"
             Command="{Binding DoFetch}"
             CommandParameter="{Binding ElementName=txtUrl, Path=Text}"
                    />
            </StackPanel>
        </toolkit:BusyIndicator>

        <ListBox Margin="5" ItemsSource="{Binding TweetItems}"
                 ScrollViewer.HorizontalScrollBarVisibility="Disabled"
                 Grid.Row="1" >
            <ListBox.ItemTemplate>
                <DataTemplate>
                    <Border Margin="5" CornerRadius="6"
                          BorderThickness="5" BorderBrush="#FF520808" >
                        <StackPanel Margin="5">
                           <TextBlock  Text="{Binding CreatedAt}"  />
                           <TextBlock TextWrapping="Wrap"
                                      Text="{Binding Text}" />
                        </StackPanel>
                    </Border>
                </DataTemplate>
            </ListBox.ItemTemplate>
        </ListBox>
    </Grid>
</UserControl>
```

همانطور که ملاحظه می‌نمائید، متن وارد شده در txtUrl از طریق CommandParameter مقید شده به خاصیت Text این کنترل بصری (txtUrl)، به RelayCommand تعریف شده (DoFetch) منتقل خواهد شد.

در این مثال از کنترل BusyIndicator نیز جهت نمایش لطفا منتظر بمانید در حین دریافت اطلاعات Feed نیز استفاده شده است. این کنترل جزو مجموعه‌ی Silverlight toolkit است. کار نمایش و یا مخفی شدن آن به کمک خاصیت IsBusy آن مدیریت می‌شود که به خاصیت متناظری در ViewModel برنامه Bind شده است (خاصیت BusyIndicatorIsBusy). اگر مجموعه‌ای از کنترل‌های برنامه به عنوان محتوای کنترل BusyIndicator معرفی شوند، این مجموعه تا پایان کار و تنظیم خاصیت IsBusy به True ، غیرفعال خواهند بود. برای مثال در اینجا یک StackPanel به عنوان محتوای آن معرفی گردیده است. بهتر است برای افزوده شدن خودکار ارجاعات لازم جهت نمایش BusyIndicator، این کنترل را از جعبه ابزار VS.NET کشیده و بر روی فرم برنامه رها کنیم. در پایان برای استفاده از این View در صفحه‌ی اصلی برنامه خواهیم داشت :

MainPage.xaml

```
<UserControl x:Class="SilverlightApplication51.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:vm="clr-namespace:SilverlightApplication51.Views"
    mc:Ignorable="d" d:DesignHeight="300" d:DesignWidth="500">
    <Grid x:Name="LayoutRoot" Background="White">
        <vm:TweetsList />
    </Grid>
</UserControl>
```

مثال چهارم – بررسی EventToCommand

گاهی از اوقات در برنامه‌های WPF و یا سیلورلایت نیاز می‌شود تا اطلاعات EventArgs یک رخداد را نیز به ICommand ارسال کرد. برای مثال تعریف یک RelayCommand از نوع DragEventArgs جهت مدیریت بهتر رخدادهای drag & drop در ViewModel . این قابلیت تحت عنوان EventToCommand به MVVM Light Toolkit اضافه شده است که از کتابخانه‌های Expression blend و توانمندی‌های آن به ارث رسیده است (در اصل از کتابخانه‌ی System.Windows.Interactivity.dll آن استفاده می‌کند).

EventToCommand رفتاری است که می‌توان آن‌را به هر نوع FrameworkElement اضافه کرد (یک مستطیل، یک تصویر، هر نوع کنترلی و کلا هر نوع شیءایی که بتوان آن‌را به رابط کاربر برنامه افزود).

باید درنظر داشت که مطابق اصول MVVM ، ViewModel برنامه تا حد امکان نباید اطلاعاتی از رابط کاربر داشته باشد؛ زیرا این امر امکان تهیه‌ی آزمون‌های واحد آن‌را مشکل می‌سازد. بنابراین حین استفاده از قابلیت EventToCommand بهتر است به این نکته نیز توجه داشت.

برای استفاده از این قابلیت ابتدا فضاهای نام زیر باید به Xaml ما اضافه شوند:

XAML

```
xmlns:i=
"clr-namespace:System.Windows.Interactivity;assembly=System.Windows.Interactivity"
xmlns:cmd=
"clr-namespace:GalaSoft.MvvmLight.Command;assembly=GalaSoft.MvvmLight.Extras"
```

سپس نمونه‌ای از تعریف آن در یک View به صورت زیر می‌تواند باشد:

XAML

```
<Rectangle Fill="White"
           Stroke="Black"
           Width="200" Height="100">
    <i:Interaction.Triggers>
        <i:EventTrigger EventName="MouseEnter">
            <cmd:EventToCommand Command="{Binding MyCommand,
                                        Mode=OneWay}"
               CommandParameter="{Binding Text,
```

```
                                    ElementName=MyTextBox,
                                Mode=OneWay}"
                PassEventArgsToCommand="True"
                MustToggleIsEnabledValue="True" />
        </i:EventTrigger>
    </i:Interaction.Triggers>
</Rectangle>
```

و در ادامه برای گوش فرا دادن به رخدادهای آن در یک ViewModel به صورت زیر می‌توان عمل کرد:

C#

```
public  RelayCommand<MouseEventArgs>  MyCommand { get; private set; }
public MainViewModel()
{
    MyCommand = new RelayCommand<MouseEventArgs>(e =>{
        // e is of type MouseEventArgs
    });
}
```

معرفی مثال چهارم

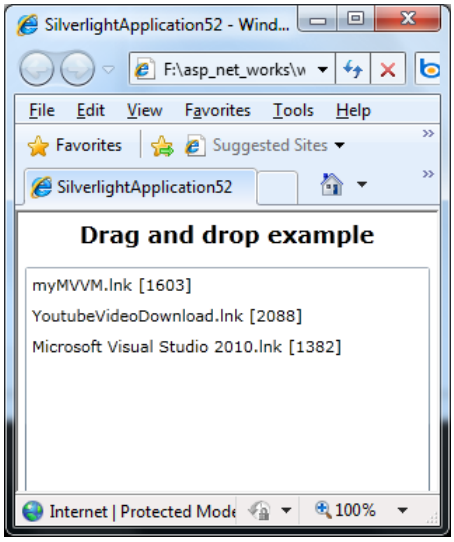
در این برنامه قصد داریم نام و مسیر پوشه‌ی فایل‌هایی را که از طریق drag & drop بر روی یک listBox کشیده و رها می‌شوند، نمایش دهیم (شکل‌های ۱۳ و ۱۴). پشتیبانی از Drag & drop ، یکی از ویژگی‌های جدید Silverlight 4 است.

مدل مثال چهارم

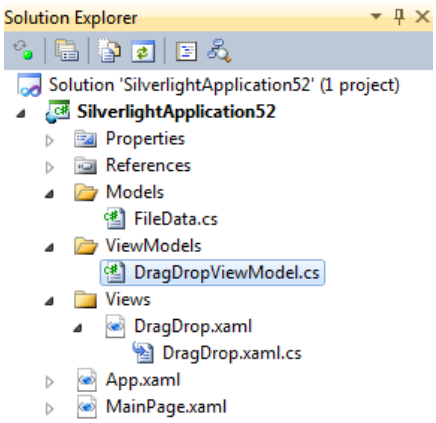
این مدل ساده بیانگر ساختار اطلاعات نمایشی فایل‌های کشیده و رها شده بر روی listBox می‌باشد:

FileData.cs

```
namespace SilverlightApplication52.Models
{
    public class FileData
    {
        public string Name { set; get; }
        public long Length { set; get; }
    }
}
```



شکل ۱۳– نمایی از برنامه چهارم در حال اجرا



شکل ۱۴– ساختار پوشه‌ها و فایل‌های مثال چهارم

View مثال چهار

در این View یک listBox را بر روی User control ایی قرار داده و سپس خاصیت AllowDrop آن‌را به true تنظیم کرده‌ایم.

DragDrop.xaml

```
<UserControl x:Class="SilverlightApplication52.Views.DragDrop"
    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:i=
"clr-namespace:System.Windows.Interactivity;assembly=System.Windows.Interactivity"
    xmlns:cmd=
"clr-namespace:GalaSoft.MvvmLight.Command;assembly=GalaSoft.MvvmLight.Extras.SL4"
    mc:Ignorable="d" d:DesignHeight="300" d:DesignWidth="400">
    <StackPanel>
        <TextBlock
            Text="Drag and drop example"
            FontWeight="Bold"
            FontSize="16"
            Margin="5"
            HorizontalAlignment="Center" />
        <ListBox AllowDrop="True" MinHeight="300"
                 ItemsSource="{Binding FilesList}" Margin="5">
```

```
            <i:Interaction.Triggers>
                <i:EventTrigger
                    EventName="Drop">
                    <cmd:EventToCommand
                        Command="{Binding DropCommand,
                                          Mode=OneWay}"
                        PassEventArgsToCommand="True"
                        MustToggleIsEnabledValue="True" />
                </i:EventTrigger>
            </i:Interaction.Triggers>
            <ListBox.ItemTemplate>
                <DataTemplate>
                    <StackPanel Orientation="Horizontal">
                        <TextBlock Text="{Binding Name}" />
                        <TextBlock Text=" [" />
                        <TextBlock Text="{Binding Length}" />
                        <TextBlock Text="]" />
                    </StackPanel>
                </DataTemplate>
            </ListBox.ItemTemplate>
        </ListBox>
    </StackPanel>
</UserControl>
```

در مورد تعریف فضاهای نام مرتبط با Interaction.Triggers جهت تعریف EventToCommand پیشتر توضیح داده شد و اینجا مثالی دیگر از این دست را جهت رخداد Drop ملاحظه می‌نمائید.

این View اطلاعات binding خود را از طریق کدهای زیر دریافت می‌کند (این هم یک روش تعریف DataContext است):

DragDrop.xaml.cs

```
using SilverlightApplication52.ViewModels;

namespace SilverlightApplication52.Views
{
    public partial class DragDrop
    {
        public DragDrop()
        {
            InitializeComponent();
            this.DataContext = new DragDropViewModel();
        }
    }
}
```

مثال چهارم ViewModel

کدهای ViewModel برنامه جهت گوش فرا دادن به رخداد Drop به شرح زیر هستند:

DragDropViewModel.cs

```
using System.Collections.ObjectModel;
using System.IO;
using System.Linq;
using System.Windows;
using System.Windows.Shapes;
using GalaSoft.MvvmLight.Command;
using SilverlightApplication52.Models;
namespace SilverlightApplication52.ViewModels
{
    public class DragDropViewModel
    {
        public ObservableCollection<FileData> FilesList { get; private set; }
        public RelayCommand<DragEventArgs> DropCommand  { get; private set; }
        public DragDropViewModel()
        {
            DropCommand = new RelayCommand<DragEventArgs>(dropped);
            FilesList = new ObservableCollection<FileData>();
        }

        private void dropped(DragEventArgs e)
        {
            if (e.Data == null) return;
            var files = e.Data.GetData(DataFormats.FileDrop) as FileInfo[];
            if (files == null) return;

            foreach (var file in files)
            {
                var fileData = new FileData
                {
                    Name = file.Name,
                    Length = file.Length
                };

                var any = FilesList.FirstOrDefault(
                    x => x.Length == fileData.Length
                        && x.Name == fileData.Name);
                if (any == null) FilesList.Add(fileData);
            }
        }
    }
}
```

در اینجا یک RelayCommand از نوع DragEventArgs تعریف شده و متد آن، اطلاعات DragEventArgs را از View برنامه جهت دریافت اطلاعات فایل‌های کشیده و رها شده دریافت می‌کند. به این صورت می‌توان خاصیت عمومی FilesList را مقدار دهی نمود و آن‌را جهت عملیات Binding و نمایش اطلاعات فایل‌ها در اختیار View گذاشت.