Silverlight 4

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

فهرست مطالب

فصل ۲۱ - بررسی کنترل DataForm

مقدمه

کنترل DataForm جزو مجموعه‌ی Silverlight Toolkit بوده و همانطور که از اسم آن نیز پیدا است، جهت مقاصد RAD (Rapid Application Development)، از تعاریف مدل‌های برنامه، فرم‌های ورود ، ویرایش و حذف اطلاعات را به صورت خودکار تولید می‌کند. در طی فصل جاری قصد داریم توانایی‌های این کنترل قدرتمند را که جزو ملزومات برنامه‌های بزرگ تجاری است، بررسی کنیم.

نمایش و ویرایش یک شیء از طریق کنترل DataForm

می‌خواهیم در طی یک مثال کاربردی، اطلاعات یک شیء را به صورت خودکار تبدیل به فرم ورود اطلاعات متناظر آن نمائیم. همچنین امکانات ذخیره سازی و لغو عملکرد کنترل DataForm را نیز بررسی کنیم. برای این منظور یک پروژه‌ی جدید Silverlight را آغاز نمائید (شکل ۱). پوشه‌های Models و ViewModels را به آن افزوده و سپس ارجاعاتی را به اسمبلی‌های کتابخانه‌ی MVVM Light toolkit اضافه نمائید. این فایل‌ها را در مسیر ذیل می‌توانید پیدا کنید:

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

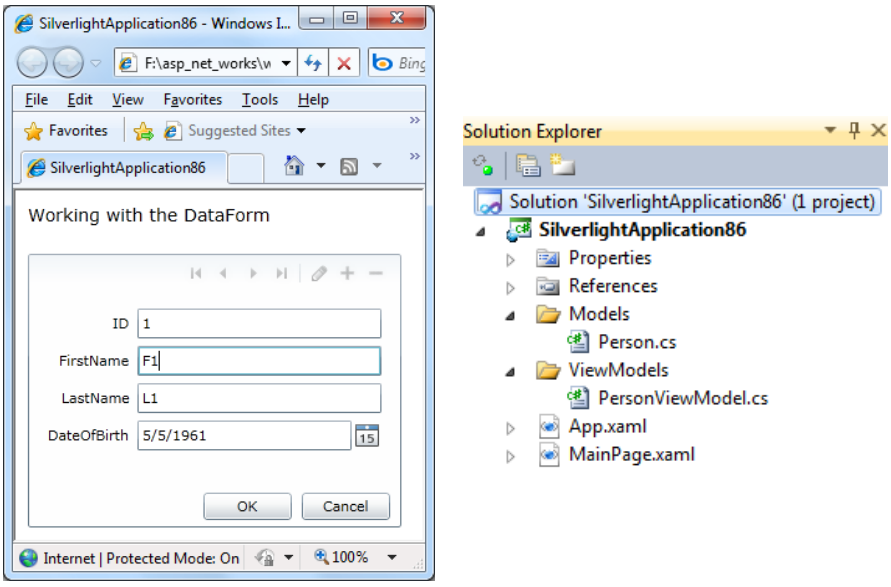
کدهای مدل برنامه را در ادامه ملاحظه خواهید نمود:

Person.cs

```
using System;
using System.ComponentModel;

namespace SilverlightApplication86.Models
{
    public class Person : IEditableObject
    {
        public int ID { get; set; }
        public string FirstName { get; set; }
        public string LastName { get; set; }
        public DateTime DateOfBirth { get; set; }
```

```
        #region IEditableObject Members

        private Person _tmpPerson;
        public void BeginEdit()
        {
            // save current values
            _tmpPerson = new Person
           {
               ID = this.ID,
               FirstName = this.FirstName,
               LastName = this.LastName,
               DateOfBirth = this.DateOfBirth
           };
        }

        public void CancelEdit()
        {
            // reset values
            this.ID = _tmpPerson.ID;
            this.FirstName = _tmpPerson.FirstName;
            this.LastName = _tmpPerson.LastName;
            this.DateOfBirth = _tmpPerson.DateOfBirth;
        }

        public void EndEdit()
        {
            //save to db ...
        }

        #endregion
    }
}
```

در مورد اینترفیس جدید IEditableObject در ادامه توضیح داده خواهد شد.



شکل ۱– نمایی از ساختار اولین برنامه‌ی کاربرد DataForm .

کدهای ViewModel برنامه به شرح زیر هستند:

PersonViewModel.cs

```
using System;
using System.Windows;
using System.Windows.Controls;
using GalaSoft.MvvmLight.Command;
using SilverlightApplication86.Models;

namespace SilverlightApplication86.ViewModels
{
    public class PersonViewModel
    {
        public Person Person { get; set; }
        public RelayCommand<DataFormEditEndedEventArgs>
            EditEndedCommand { set; get; }

        public PersonViewModel()
        {
            EditEndedCommand =
                new RelayCommand<DataFormEditEndedEventArgs>(
                    editEndedCommand);

            Person = new Person
            {
                ID = 1,
```

```
                FirstName = "F1",
            LastName = "L1",
            DateOfBirth = new DateTime(1961, 5, 5)
        };
    }

    private void editEndedCommand(DataFormEditEndedEventArgs obj)
    {
        //TODO: save data ...
        if (obj.EditAction == DataFormEditAction.Commit)
            MessageBox.Show(string.Format("{0} Saved!",
                Person.FirstName));
    }
}
}
```

و کدهای XAML مرتبط با View برنامه در ادامه ذکر شده است:

MainPage.xaml

```
<UserControl x:Class="SilverlightApplication86.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:toolkit
    ="http://schemas.microsoft.com/winfx/2006/xaml/presentation/toolkit"
    xmlns:vm="clr-namespace:SilverlightApplication86.ViewModels"
    xmlns:
    cmd="clr-namespace:GalaSoft.MvvmLight.Command;assembly=GalaSoft.MvvmLight.Extras.SL4"
    xmlns:
    i="clr-namespace:System.Windows.Interactivity;assembly=System.Windows.Interactivity">
    <UserControl.Resources>
        <vm:PersonViewModel x:Key="vmPersonViewModel" />
    </UserControl.Resources>
    <StackPanel DataContext=
                "{Binding Source={StaticResource vmPersonViewModel}}">
        <TextBlock Text="Working with the DataForm"
            Margin="10"
           FontSize="14" />
        <toolkit:DataForm x:Name="myDataForm"
            AutoEdit="False"
            CommandButtonsVisibility="All"
            Grid.Row="1"
            Width="300"
            Height="220"
            Margin="10"
            CurrentItem="{Binding Person}"
            HorizontalAlignment="Left"
```

```
            VerticalAlignment="Top">

            <i:Interaction.Triggers>
                <i:EventTrigger
                    EventName="EditEnded">
                    <cmd:EventToCommand
                        Command="{Binding EditEndedCommand,
                                          Mode=OneWay}"
                        PassEventArgsToCommand="True"/>
                </i:EventTrigger>
            </i:Interaction.Triggers>

        </toolkit:DataForm>
    </StackPanel>
</UserControl>
```

توضیحات:

برای اضافه کردن کنترل DataForm بهتر است آن‌را از جعبه ابزار VS.NET کشیده و بر روی فرم برنامه رها کنید. به این صورت افزودن ارجاعات لازم به اسمبلی‌های مورد نیاز آن و همچنین فضاهای نام متناظر آن‌ها به صورت خودکار به پروژه و View جاری انجام خواهد شد.

مدل برنامه بیانگر خواص مختلف شیء Person است. این کلاس، اینترفیس استاندارد IEditableObject را نیز پیاده سازی کرده است. پیاده سازی آن از این جهت حائز اهمیت می‌باشد که بدون آن، دکمه‌ی لغو عملیات در DataForm نمایش داده شده، فعال نخواهد بود. هر یک از متدهای BeginEdit ، CancelEdit و EndEdit شبیه به روال‌های رخدادگردان عمل کرده و با کلیک بر روی دکمه‌های ویرایش و یا لغو ویرایش فراخوانی می‌گردند. توسط این روال‌ها می‌توان اطلاعات وارد شده توسط کاربر را به صورت خودکار دریافت کرد، به حالت اول بازگرداند و یا در یک بانک اطلاعاتی ذخیره نمود.

کلاس ViewModel برنامه، یک وهله از شیء Person را در اختیار View قرار خواهد داد. همچنین کار تعریف روال رخداد گردان EditEnded را نیز بر عهده دارد. به همین جهت نیاز بود تا از کتابخانه‌ی MVVM Light toolkit استفاده می‌شد. در این حالت یا می‌توان عملیات ذخیره سازی اطلاعات را در روال‌های تعریف شده در مدل برنامه انجام داد و یا در ViewModel فوق.

در View برنامه ابتدا DataContext مربوط به StackPanel دربرگیرنده‌ی DataForm با اطلاعات کلاس ViewModel مقدار دهی خواهد شد. سپس یک DataForm بر روی فرم قرار گرفته است. این کنترل بر اساس اطلاعات موجودیت دریافتی از طریق خاصیت CurrentItem خود، یک فرم ورود اطلاعات شکیل را به صورت خودکار تولید خواهد نمود. کنترل DataForm فیلدهای عمومی کلاس Person را یافته و بر اساس نوع آن‌ها کنترل‌های درخوری را به فرم تشکیل شده خواهد افزود. برای مثال با توجه به اینکه فیلد DateOfBirth از نوع DateTime تعریف شده است، یک کنترل DatePicker را به صورت خودکار به فرم نهایی برنامه اضافه کرده است.

در کنترل DataForm ، تنظیم خاصیت CommandButtonsVisibility به All سبب شده است تا نوار ابزاری بالای فرم نمایش داده شود. با تنظیم خاصیت AutoEdit به False مطمئن خواهیم شد که در اولین بار نمایش فرم، در حالت ویرایش اطلاعات قرار نخواهیم گرفت. زمانیکه کاربر بر روی دکمه‌ی Edit کلیک نماید و فرم را ویرایش کند، با فشردن دکمه‌ی OK ، سبب بروز رخداد EditEnded می‌گردد. از این رخداد می‌توان در ViewModel برنامه جهت ذخیره سازی اطلاعات برنامه در یک بانک اطلاعاتی کمک جست. برای انتقال این رخداد به ViewModel ، از ویژگی EventToCommand کتابخانه‌ی MVVM Light toolkit استفاده گردید.

نکته‌ی جالب دیگر کنترل DataForm ، تعیین اعتبار خودکار ورودی کاربر است. برای مثال سعی کنید در فیلد ID یک رشته را وارد نمائید. بلافاصله بر اساس نوع عددی این فیلد، پیغام اعتبار سنجی مناسبی نمایش داده خواهد شد.

نمایش و ویرایش لیستی از اشیاء به کمک کنترل DataForm

در مثال قبل آموختیم که چگونه از یک موجودیت، فرم ورود اطلاعات متناظر آن‌را به صورت خودکار ایجاد نمائیم. اما در اغلب برنامه‌های کاربردی نیاز است تا با لیستی از اشیاء بتوان کار کرد. برای مثال بتوان اشیاء لیستی از کارکنان را ویرایش، حذف و یا اضافه نمود. کنترل DataForm جهت کار با این سناریو نیز آمادگی کامل را دارد.

برای بررسی یک مثال کاربردی در این زمینه، همان مثال قبل را تکمیل خواهیم کرد. مدل برنامه نیازی به تغییر ندارد، اما ViewModel آن جهت تعریف لیستی از اشخاص تغییر خواهد کرد:

PersonViewModel.cs

```
using System;
using System.Collections.ObjectModel;
using System.Windows;
using System.Windows.Controls;
using GalaSoft.MvvmLight.Command;
using SilverlightApplication86.Models;

namespace SilverlightApplication86.ViewModels
{
    public class PersonViewModel
    {
        public Person Person { get; set; }
        public RelayCommand<DataFormEditEndedEventArgs>
               EditEndedCommand { set; get; }

        public ObservableCollection<Person> Persons { get; set; }

        public PersonViewModel()
```

```
        {
            EditEndedCommand =
                new RelayCommand<DataFormEditEndedEventArgs>(
                    editEndedCommand);

            Person = new Person
            {
                ID = 1,
                FirstName = "F1",
                LastName = "L1",
                DateOfBirth = new DateTime(1961, 5, 5)
            };

            Persons = new ObservableCollection<Person>()
                        {
                          new Person
                          {
                            ID=1,
                            FirstName="F1",
                            LastName="L1",
                            DateOfBirth = new DateTime(1981, 5, 5)
                          },
                          new Person
                          {
                            ID=2,
                            FirstName="F1",
                            LastName="L2",
                            DateOfBirth = new DateTime(2009, 1, 1)
                          },
                          new Person
                          {
                            ID=3,
                            FirstName="F3",
                            LastName="L3",
                            DateOfBirth = new DateTime(1922, 6, 10)
                          }
                        };
        }

        private void editEndedCommand(DataFormEditEndedEventArgs obj)
        {
            //TODO: save data ...
            if (obj.EditAction == DataFormEditAction.Commit)
                MessageBox.Show(string.Format("{0} Saved!",
                    Person.FirstName));
        }
    }
}
```

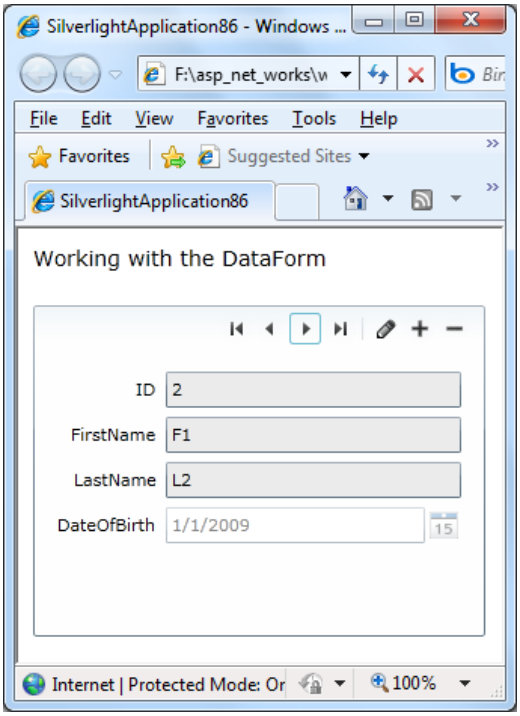
تفاوت این کلاس با نمونه ی قبلی آن، اضافه شدن یک شیء ObservableCollection از نوع کلاس Person است که با سه عضو اولیه مقدار دهی گردیده است.

در View برنامه تنها دو سطر باید تغییر کنند:

MainPage.xaml

```
        ...
    <toolkit:DataForm x:Name="myDataForm"
        ...
        CurrentItem="{Binding Person, Mode=TwoWay}"
        ItemsSource="{Binding Persons}"
        ...
```

با توجه به انقیاد دو طرفه تعریف شده در اینجا، هر آیتم جاری برنامه در شیء Person منعکس خواهد شد. لیست اشخاص تعریف شده نیز از طریق خاصیت ItemsSource در اختیار کنترل DataForm قرار می‌گیرد. اکنون اگر برنامه را اجرا نمائید شکل ۲ نمایان خواهد شد.



شکل ۲– نمایی از دومین مثال استفاده از کنترل DataForm .

در اینجا به سادگی می‌توان یک شیء را حذف کرد، ویرایش نمود و یا به لیست موجود افزود. علاوه بر آن امکان حرکت بین رکوردهای مختلف نیز موجود است.

اگر در کنترل DataForm خاصیت AutoCommit به True تنظیم شده باشد، با حرکت بین رکوردهای مختلف، اطلاعات تغییر یافته به صورت خودکار به اشیاء متناظر اعمال خواهند شد و در این حالت کلیک بر روی

دکمه‌ی OK الزامی نخواهد بود. اگر خاصیت AutoEdit به True تنظیم شود، با حرکت بین رکوردها، حالت آغازین نمایش هر رکورد، حالت ویرایش خواهد بود. اگر علاقمندید که متن برچسب دکمه‌های OK و Cancel را تغییر دهید، خواص CommitButtonContent و CancelButtonContent را مقدار دهی کنید. مقدار خاصیت CommandButtonsVisibility مشخص خواهد نمود که آیا دکمه‌های حذف یک شیء و یا افزودن یک رکورد جدید نمایش داده شوند یا خیر:

```
CommandButtonsVisibility="Navigation, Cancel, Commit, Add, Delete"
```

همانطور که ملاحظه می‌نمائید هر کدام از عناصر تشکیل دهنده‌ی نوار ابزار کنترل DataForm را می‌توان به این لیست افزود و یا حذف کرد.

علاوه بر آن یک سری رخداد نیز در اختیار برنامه نویس‌ها خواهد بود. رخداد AddingNewItem درست پیش از افزوده شدن یک شیء به لیست، بروز داده خواهد شد؛ رخداد DeletingItem پیش از حذف یک شیء و رخداد CurrentItemChanged پس از حذف یا افزودن و یا حرکت بین رکوردها فراخوانی می‌گردد. در روال رخداد گردان DeletingItem ، اگر خاصیت لغو آرگومان دریافتی از نوع CancelEventArgs را با true مقدار دهی کنیم (e.Cancel = true) ، می‌توان از حذف شدن رکورد جاری جلوگیری کرد و یا بهتر است پیغامی را به کاربر نمایش داده و در صورت عدم تائید او نسبت تنظیم e.Cancel = true اقدام نمود.

سفارشی سازی کنترل DataForm با استفاده از DataAnnotations

تبدیل خودکار یک شیء مدل برنامه به فرم ورود اطلاعات متناظر با آن بسیار جالب توجه و کاربردی است. اما در یک برنامه‌ی کاربردی نیاز خواهد بود تا بتوان برچسب‌های بامعناتری را ارائه داد یا بتوان یک فیلد را به صورت فقط خواندنی معرفی کرد و غیره. خوشبختانه می‌توان کنترل DataForm را جهت برآوردن این نیازها سفارشی سازی نمود. برای این منظور یکی از روش‌های موجود استفاده از فضای نام System.ComponentModel.DataAnnotations می‌باشد که در فصل روش‌های تعیین اعتبار ورودی کاربران با آن آشنا شدیم. بنابراین ارجاعی را به اسمبلی آن به پروژه‌ی مثال قبل اضافه نمائید. اکنون کلاس مدل برنامه را مطابق کدهای بعد تغییر دهید. تاثیر این تغییرات را در شکل ۳ می‌توانید ملاحظه نمائید.

Person.cs

```
using System;
using System.ComponentModel;
using System.ComponentModel.DataAnnotations;

namespace SilverlightApplication86.Models
{
    public class Person : IEditableObject
    {
```

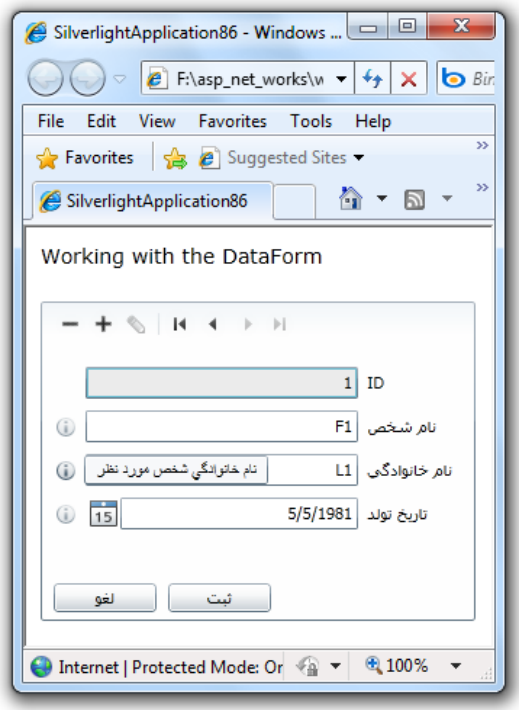
```
        [Editable(false)]
    [Display(Description = "شماره پرسنلی")]
    public int ID { get; set; }

    [Display(Name = "نام شخص", Description = "نام شخص مورد نظر")]
    public string FirstName { get; set; }

    [Display(Name = "نام خانوادگی",
        Description = "نام خانوادگی شخص مورد نظر")]
    public string LastName { get; set; }

    [Display(Name = "تاریخ تولد",
        Description = "تاریخ تولد شخص مورد نظر")]
    public DateTime DateOfBirth { get; set; }

    #region IEditableObject Members
    //same as before ...
    #endregion
  }
}
```

در اینجا به کمک DataAnnotations ، برچسب فیلدهای تعریف شده به همراه توضیحاتی در مورد آن‌ها ارائه شده است. اگر از DataAnnotations استفاده نشود و یا پارامتر Name آن ذکر نگردد، همان نام انگلیسی فیلد نمایش داده خواهد شد (همانند فیلد ID) . نحوه‌ی نمایش فیلد ID به صورت فقط خواندنی نیز ذکر گردیده است.

در اینجا View برنامه را نیز اندکی ویرایش کرده‌ایم تا برای حالت راست به چپ زبان فارسی مناسب باشد. کدهای XAML تغییر کرده را در ادامه ملاحظه خواهید نمود.

MainPage.xaml

```
        ...
    <toolkit:DataForm x:Name="myDataForm"
        FlowDirection="RightToLeft"
        FontFamily="Tahoma"
        CancelButtonContent="لغو"
        CommitButtonContent="ثبت"
        ....
```



شکل ۳– سفارشی سازی ظاهر یک DataForm به کمک DataAnnotations .

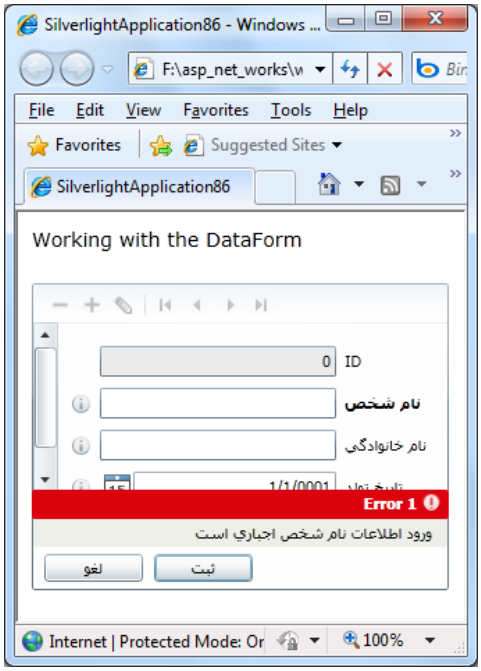
لیست کامل، مفصل و به روز تمامی DataAnnotations مهیا را در آدرس بعد می‌توانید ملاحظه نمائید:

http://bit.ly/9VoNLq

علاوه بر آن همانند مطالبی که در فصل روش‌های مختلف تعیین اعتبار اطلاعات در Silverlight ذکر گردید، از DataAnnotations جهت تعریف قیود فیلدهای مختلف نیز می‌توان استفاده کرد. در اینجا ذکر این ویژگی‌ها به تنهایی کافی است و سایر موارد توسط کنترل DataForm به صورت خودکار لحاظ خواهد شد. برای مثال فیلد نام شخص را در مدل برنامه به صورت زیر تغییر دهید. سپس برنامه را اجرا کرده و بر روی دکمه‌ی افزودن یک رکورد جدید کلیک کنید. پس از ظاهر شدن صفحه‌ی ورود اطلاعات، اگر بر روی دکمه‌ی ثبت کلیک نمائید، با پیغام شکل ۴ مواجه خواهید شد.

Person.cs

```
...
 [Display(Name = "نام شخص",
       Description = "نام شخص مورد نظر")]
 [StringLength(30, MinimumLength = 3,
       ErrorMessage = "نام شخص باید بین سه تا ۳۰ حرف وارد شود")]
 [Required(ErrorMessage = "ورود اطلاعات نام شخص اجباری است")]
 public string FirstName { get; set; }
...
```

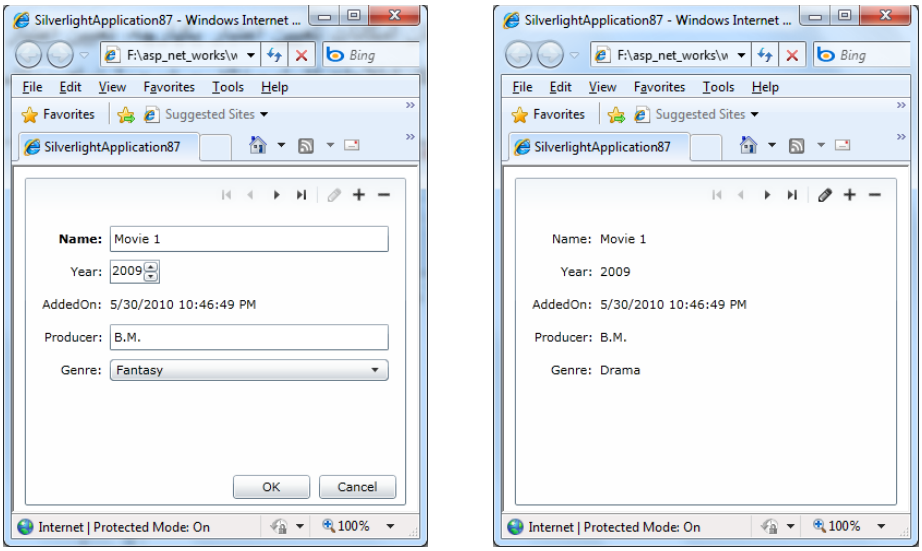


شکل ۴– استفاده از DataAnnotations جهت تعاریف اعتبارسنجی فیلدهای DataForm .

در این شکل علت ضخیم نمایش داده شدن برچسب فیلد نام شخص، استفاده از ویژگی [Required] برای خاصیت متناظر با آن است.

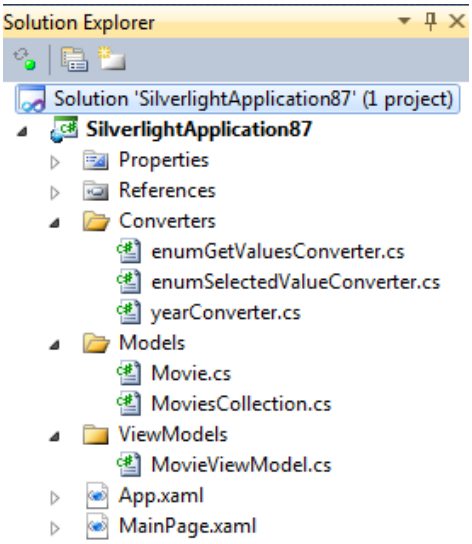
سفارشی سازی کنترل DataForm با استفاده از DataTemplates

تاکنون نحوه‌ی استفاده و سفارشی سازی مقدماتی کنترل DataForm را بررسی نمودیم. اما این‌ها نیز ممکن است کافی نباشند! برای مثال نیاز است در برنامه بجای کنترل‌های پیش فرض که بر اساس نوع داده‌های فیلدهای یک مدل تشکیل می‌شوند، از یک سری کنترل‌های دلخواه و سفارشی دیگر استفاده گردد. در این حالت می‌توان قالب حالات مختلف کنترل DataForm را ویرایش نموده و کنترل‌های مورد نظر خود را تعریف کنیم. بدیهی است در این حالت از امکانات تولید پویای فیلدهای فرم خبری نخواهد بود؛ اما همچنان امکانات تعیین اعتبار یکپارچه، تعیین اعتبار مدل حین ثبت نهایی، صفحه بندی خودکار اطلاعات اشیاء مختلف و غیره را خواهیم داشت.



شکل ۵– سفارشی سازی قالب‌های نمایش و ویرایش اطلاعات در DataForm .

پروژه‌ی Silverlight جدیدی را آغاز نمائید. در این مثال همانند شکل ۵ قصد داریم قالب‌های حالت نمایش فقط خواندنی و حالت ویرایش کنترل DataForm را کاملا سفارشی سازی نمائیم. ساختار فایل‌ها و پوشه‌های این پروژه را در شکل ۶ می‌توانید مشاهده نمائید. پس از آغاز پروژه، دو کنترل DataForm و NumericUpDown را از جعبه ابزار VS.NET کشیده و بر روی فرم رها کنید تا ارجاعات لازم به اسمبلی‌های آن‌ها و همچنین فضاهای نام متناظر آن‌ها به صورت خودکار افزوده شوند.



شکل ۶– نمایی از ساختار پروژه‌ی سفارشی سازی قالب‌های DataForm .

همانطور که در شکل نیز مشخص است، می‌خواهیم اعضای یک enum را در یک ComboBox نمایش دهیم. به همین منظور نیاز به تبدیل کننده‌ی ذیل خواهد بود. این تبدیل کننده، اعضای یک enum را یافته و تبدیل به آرایه‌ای از رشته‌ها می‌کند:

enumGetValuesConverter.cs

```
using System;
using System.Globalization;
using System.Linq;
using System.Reflection;
using System.Windows.Data;

namespace SilverlightApplication87.Converters
{
    public class EnumGetValuesConverter : IValueConverter
    {
        public object Convert(object value, Type targetType,
            object parameter, CultureInfo culture)
        {
            Type enumType = value.GetType();
            var infos = enumType.GetFields(
                BindingFlags.Public | BindingFlags.Static);

            return infos.Select(fi => fi.Name).ToArray();
        }

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

علاوه بر آن نیاز است زمانیکه یکی از آیتم‌های ComboBox انتخاب می‌شود بتوان مقدار رشته‌ای دریافتی را به معادل enum آن تبدیل کرده و در اختیار شیء جاری مقید شده به آن قرار داد. کلاس تبدیل کننده‌ی بعد به این جهت ایجاد شده است:

enumSelectedValueConverter.cs

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

namespace SilverlightApplication87.Converters
{
```

```
    public class EnumSelectedValueConverter : IValueConverter
    {
        public object Convert(object value,
            Type targetType, object parameter,CultureInfo culture)
        {
            return value.ToString();
        }

        public object ConvertBack(object value,
            Type targetType, object parameter, CultureInfo culture)
        {
            if (value == null) return null;

            if (!targetType.IsEnum || value.GetType() != typeof(String))
                throw new ArgumentException();

            return Enum.Parse(targetType, value.ToString(), true);
        }
    }
}
```

تبدیل کننده‌ی بعد برای استفاده در کنترل NumericUpDown و تبدیل مقادیر دریافتی از آن جهت شیء جاری ایجاد شده از مدل برنامه، تهیه شده است:

yearConverter.cs

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

namespace SilverlightApplication87.Converters
{
    public class YearConverter : IValueConverter
    {
        public object Convert(object value,
            Type targetType, object parameter, CultureInfo culture)
        {
            return value.ToString();
        }

        public object ConvertBack(object value,
            Type targetType, object parameter, CultureInfo culture)
        {
            return (int)(double)value;
        }
    }
}
```

کلاس ساده‌ی مدل برنامه را در ادامه ملاحظه خواهید نمود. این کلاس دو اینترفیس استاندارد IEditableObject و InotifyPropertyChanged را پیاده سازی می‌کند و در مورد جزئیات هر یک پیشتر توضیح داده شده است (برای استفاده از امکانات مقید سازی TwoWay و همچنین فعال شدن دکمه‌ی Cancel در DataForm).

Movie.cs

```
using System;
using System.ComponentModel;
using System.ComponentModel.DataAnnotations;

namespace SilverlightApplication87.Models
{
    public enum Genres
    {
        Comedy,
        Fantasy,
        Drama,
        Thriller
    }

    public class Movie : IEditableObject, INotifyPropertyChanged
    {
        int _movieID;
        public int MovieID
        {
            get { return _movieID; }
            set
            {
                if (_movieID == value) return;
                _movieID = value;
                raisePropertyChanged("MovieID");
            }
        }

        string _name;
        [StringLength(30, MinimumLength = 3,
              ErrorMessage = "نام فیلم باید بین سه تا ۳۰ حرف وارد شود")]
        [Required(ErrorMessage = "ورود اطلاعات نام فیلم اجباری است")]
        public string Name
        {
            get { return _name; }
            set
            {
                if (_name == value) return;
                _name = value;
                raisePropertyChanged("Name");
```

```
        }
    }

    int _year;
    public int Year
    {
        get { return _year; }
        set
        {
            if (_year == value) return;
            _year = value;
            raisePropertyChanged("Year");
        }
    }


    DateTime _addedOn;
    public DateTime AddedOn
    {
        get { return _addedOn; }
        set
        {
            if (_addedOn == value) return;
            _addedOn = value;
            raisePropertyChanged("AddedOn");
        }
    }

    string _producer;
    public string Producer
    {
        get { return _producer; }
        set
        {
            if (_producer == value) return;
            _producer = value;
            raisePropertyChanged("Producer");
        }
    }

    Genres _genre;
    public Genres Genre
    {
        get { return _genre; }
        set
        {
            if (_genre == value) return;
            _genre = value;
```

```
                raisePropertyChanged("Genre");
            }
        }

        #region INotifyPropertyChanged Members
        public event PropertyChangedEventHandler PropertyChanged;
        void raisePropertyChanged(string propertyName)
        {
            var handler = PropertyChanged;
            if (handler == null) return;
            handler(this, new PropertyChangedEventArgs(propertyName));
        }
        #endregion

        #region IEditableObject Members
        private Movie _tmpMovie;
        public void BeginEdit()
        {
            // save current values
            _tmpMovie = new Movie
                        {
                            MovieID = this.MovieID,
                            Name = this.Name,
                            Year = this.Year,
                            AddedOn = this.AddedOn,
                            Producer = this.Producer,
                            Genre = this.Genre
                        };
        }

        public void CancelEdit()
        {
            // reset values
            MovieID = _tmpMovie.MovieID;
            Name = _tmpMovie.Name;
            Year = _tmpMovie.Year;
            AddedOn = _tmpMovie.AddedOn;
            Producer = _tmpMovie.Producer;
            Genre = _tmpMovie.Genre;
        }

        public void EndEdit()
        {
            //TODO: save to db ...
        }
        #endregion
    }
}
```

مرسوم است جهت ارائه‌ی لیستی از مدل برنامه با کمک شیء ObservableCollection به نحو ذیل عمل نمود:

MoviesCollection.cs

```
using System.Collections.ObjectModel;

namespace SilverlightApplication87.Models
{
    public class MoviesCollection : ObservableCollection<Movie>
    {
    }
}
```

اکنون کدهای کلاس ViewModel برنامه به شرح ذیل خواهند بود. در اینجا لیستی از اشیاء مدل در اختیار View برنامه قرار خواهد گرفت:

MovieViewModel.cs

```
using System;
using SilverlightApplication87.Models;

namespace SilverlightApplication87.ViewModels
{
    public class MovieViewModel
    {
        public MoviesCollection Movies { set; get; }

        public MovieViewModel()
        {
            Movies = new MoviesCollection
                        {
                            new Movie
                                {
                                    MovieID = 1,    Name = "Movie 1",
                                    Year = 2009,    Producer = "B.M.",
                                    Genre = Genres.Drama,
                                    AddedOn = DateTime.Now
                                },
                            new Movie
                                {
                                    MovieID = 2,    Name = "Movie 2",
                                    Year = 2008,    Producer = "B.M.",
                                    Genre = Genres.Fantasy,
                                    AddedOn = DateTime.Now
                                }
                        };
        }
    }
}
```

کدهای XAML متناظر با ViewModel برنامه را در ادامه مشاهده خواهید نمود:

MainPage.xaml

```
<UserControl x:Class="SilverlightApplication87.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:toolkit=
    "http://schemas.microsoft.com/winfx/2006/xaml/presentation/toolkit"
    xmlns:vm="clr-namespace:SilverlightApplication87.ViewModels"
    xmlns:conv="clr-namespace:SilverlightApplication87.Converters">
    <UserControl.Resources>
        <vm:MovieViewModel x:Key="vmMovieViewModel" />
        <conv:YearConverter x:Key="yearConverter" />
        <conv:EnumGetValuesConverter x:Key="enumGetValuesConverter" />
        <conv:EnumSelectedValueConverter
                  x:Key="enumSelectedValueConverter" />
    </UserControl.Resources>
    <StackPanel
       DataContext=
         "{Binding Source={StaticResource vmMovieViewModel}}">
        <toolkit:DataForm
            Name="form1"
            AutoEdit="False"
            CommandButtonsVisibility="All"
            AutoGenerateFields="False"
            ItemsSource="{Binding Movies}"
            Width="350"
            Height="300"
            Margin="10"
            >
            <toolkit:DataForm.ReadOnlyTemplate>
                <DataTemplate>
                    <StackPanel Orientation="Vertical">
                        <toolkit:DataField  Label="Name:" >
                            <TextBlock Text="{Binding Name}" />
                        </toolkit:DataField>
                        <toolkit:DataField  Label="Year:" >
                            <TextBlock Text="{Binding Year}" />
                        </toolkit:DataField>
                        <toolkit:DataField  Label="AddedOn:" >
                            <TextBlock Text="{Binding AddedOn}" />
                        </toolkit:DataField>
                        <toolkit:DataField  Label="Producer:" >
                            <TextBlock Text="{Binding Producer}" />
                        </toolkit:DataField>
                        <toolkit:DataField  Label="Genre:" >
```

```
                        <TextBlock Text="{Binding Genre}" />
                    </toolkit:DataField>
                </StackPanel>
            </DataTemplate>
        </toolkit:DataForm.ReadOnlyTemplate>
        <toolkit:DataForm.EditTemplate>
            <DataTemplate>
                <StackPanel Orientation="Vertical">
                    <toolkit:DataField  Label="Name:" >
                     <TextBox  Text="{Binding Name, Mode=TwoWay}" />
                    </toolkit:DataField>
                    <toolkit:DataField  Label="Year:" >
                        <toolkit:NumericUpDown
                            Minimum="1900"
                            Maximum="2050"
                            HorizontalAlignment="Left"
                            Value="{Binding Year,
                                    Mode=TwoWay,
                                    Converter=
                                {StaticResource yearConverter}}" />
                    </toolkit:DataField>
                    <toolkit:DataField  Label="AddedOn:" >
                        <TextBlock  Text="{Binding AddedOn}" />
                    </toolkit:DataField>
                    <toolkit:DataField  Label="Producer:" >
                        <TextBox
                         Text="{Binding Producer, Mode=TwoWay}" />
                    </toolkit:DataField>
                    <toolkit:DataField  Label="Genre:" >
                        <ComboBox x:Name="cboGenres"
                        ItemsSource="{Binding Genre,
                        Converter=
                          {StaticResource enumGetValuesConverter}}"
                        SelectedItem="{Binding Genre, Mode=TwoWay,
                        Converter=
                       {StaticResource enumSelectedValueConverter}}" />
                    </toolkit:DataField>
                </StackPanel>
            </DataTemplate>
        </toolkit:DataForm.EditTemplate>
    </toolkit:DataForm>
</StackPanel>
</UserControl>
```

توضیحات:

در ابتدای این View ، تعاریف اشیاء تبدیل کننده و همچنین ViewModel برنامه ذکر شده‌اند و سپس وهله‌ای از کلاس ViewModel به DataContext مربوط به StackPanel انتساب داده شده است.

در این مثال خاصیت AutoGenerateFields کنترل DataForm به False تنظیم شده است؛ زیرا قصد داریم نحوه‌ی تولید فیلدهای آن‌را کاملا کنترل نموده و سفارشی سازی نمائیم. سپس با نحوه‌ی ایجاد و قالب سفارشی برای حالت نمایشی (ReadOnlyTemplate) و حالت ویرایش (EditTemplate) کنترل DataForm آشنا شدیم:

MainPage.xaml

```
...
            <toolkit:DataForm.ReadOnlyTemplate>
                <DataTemplate>
                ...
                </DataTemplate>
            </toolkit:DataForm.ReadOnlyTemplate>
            <toolkit:DataForm.EditTemplate>
                <DataTemplate>
                ...
                </DataTemplate>
            </toolkit:DataForm.EditTemplate>
...
```

در DataTemplate هر کدام از این قالب‌ها می‌توان از انواع و اقسام اشیاء طرح بندی رابط کاربر، استفاده کرده و طراحی دلخواهی را اعمال نمود. تنها یک نکته‌ی جدید در DataTemplate کنترل DataForm مطرح شده است و آن هم اشیاء toolkit:DataField می‌باشند. ذکر DataField الزامی نیست اما اگر مطرح شود به صورت خودکار برچسبی را در کنار شیء مورد نظر ما قرار می‌دهد.