Silverlight 4

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

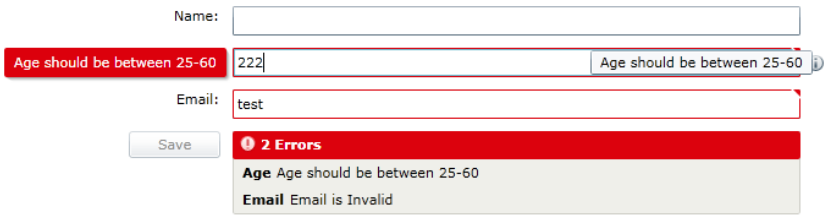
فهرست مطالب

فصل ۱۶ – روش‌های تعیین اعتبار ورودی کاربر در Silverlight

مقدمه

هیچگاه نباید به ورودی‌های کاربران اطمینان کرد؛ زیرا منشاء اصلی بسیاری از ناهماهنگی‌ها در یک سیستم و یا حملات صورت گرفته به آن‌ها، ورودی‌های تعیین اعتبار نشده‌ی کاربران است. در این فصل نگاهی خواهیم داشت به نحوه‌ی تعیین اعتبار ورودی کاربر در Silverlight که جزو مباحث تکمیلی و پیشرفته‌ی Binding نیز محسوب می‌شود و همچنین جزو الزامات تمامی برنامه‌های کاربردی و تجاری است.

تعیین اعتبار به کمک بررسی مستقیم اطلاعات و استثناءهای حاصل

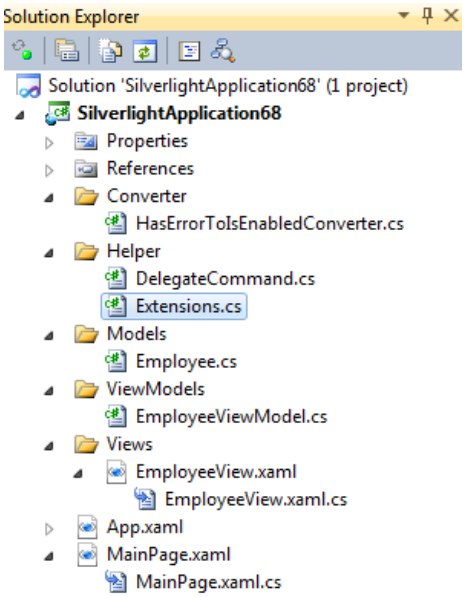


شکل ۱– نمایی از اولین برنامه‌ی تعیین اعتبار ورودی کاربر در Silverlight

در طی یک مثال کاربردی و کامل پیاده سازی شده توسط الگوی MVVM ، قصد داریم با سیستم اعتبار سنجی Silverlight که قابلیت یکپارچگی با استثناهای صادره در حین عملیات Binding را دارد، آشنا شویم. در این مثال (شکل ۱)، سه خاصیت نام، سن و آدرس ایمیل یک کارمند دریافت می‌گردد و مقادیر دریافتی قبل از هر عملیات دیگری تعیین اعتبار شده و اگر مطابق منطق مورد نظر ما نبودند، یک استثناء صادر خواهد شد. پیغام‌های صادره در این استثناءها به صورت خودکار در رابط کاربری برنامه همانند شکل ۱ به کاربر نمایش داده می‌شوند.

پیش نیاز استفاده از کنترل‌های پروژه‌ی جاری، افزودن ارجاعی به اسمبلی System.Windows.Controls.Data.Input می‌باشد. به این ترتیب کنترل‌های ValidationSummary و

DescriptionViewer ، قابل استفاده در پروژه خواهند بود که نمایی از آن‌ها را در شکل ۱ می‌توان مشاهده نمود. توسط کنترل ValidationSummary خلاصه‌ای در پایین صفحه جهت برشمردن کلیه تذکرات لازم جهت تعیین اعتبار مقادیر ورودی و توسط کنترل DescriptionViewer ، امکان نمایش یک Tooltip کوچک و زیبا در کنار TextBox مرتبط با دریافت مقدار سن، فراهم شده است.



شکل ۲– ساختار پوشه‌ها و فایل‌های اولین پروژه‌ی تعیین اعتبار ورودی کاربر در Silverlight

کلاس Extensions جهت سهولت کار با INPC ها ایجاد شده است (INPC مخفف INotifyPropertyChanged است). در تمام مثال‌های فصل‌های قبل، زمانیکه نیاز به فراخوانی متد raisePropertyChanged وجود داشت، می‌بایستی یک رشته را مساوی نام خاصیت مورد استفاده در آن وارد می‌کردیم. اما با کمک کلاس Extensions ذیل و با استفاده از متد Raise آن دیگر نیازی نیست تا نام خاصیت مورد نظر را به صورت یک رشته که امکان اشتباه تایپی در آن وجود دارد و تحت کنترل کامپایلر نیست، وارد نمائیم. به این صورت با کمک lambda expressions می‌توان ارجاع تحت نظر کامپایلری را از یک وهله‌ی شیء معرفی نمود:

Extensions.cs

```
using System;
using System.ComponentModel;
using System.Linq.Expressions;
using System.Reflection;

namespace SilverlightApplication68.Helper
{
    public static class Extensions
    {
```

```
/// <summary>
/// saves us from typos when using INPC.
/// </summary>
public static void Raise(
    this PropertyChangedEventHandler eventHandler,
    Expression<Func<object>> expression)
{
    if (null == eventHandler) return;

    PropertyInfo propertyInfo;
    ConstantExpression constantExpression =
          expression.GetPropertyInfo(out propertyInfo);

    foreach (var del in eventHandler.GetInvocationList())
    {
        del.DynamicInvoke(
            new[]
                {
                 constantExpression.Value,
                 new PropertyChangedEventArgs(propertyInfo.Name)
                });
    }
}

public static ConstantExpression GetPropertyInfo
    (this Expression<Func<object>> expression,
    out PropertyInfo propertyInfo)
{
    var lambda = expression as LambdaExpression;

    MemberExpression memberExpression;
    if (lambda.Body is UnaryExpression)
    {
      var unaryExpression = lambda.Body as UnaryExpression;
      memberExpression =
            unaryExpression.Operand as MemberExpression;
    }
    else
    {
        memberExpression = lambda.Body as MemberExpression;
    }

    if (memberExpression == null)
        throw new NullReferenceException("memberExpression");


    var constantExpression =
     memberExpression.Expression as ConstantExpression;
```

```
            if (constantExpression == null)
                throw new NullReferenceException("constantExpression");

            propertyInfo = memberExpression.Member as PropertyInfo;
            return constantExpression;
        }
    }
}
```

اکنون کاربردی از کلاس فوق، در کلاس Model برنامه که کدهای آن‌را در ادامه ملاحظه می‌کنید، ذکر گردیده است (متدهای PropertyChanged.Raise):

Employee.cs

```
using System;
using System.ComponentModel;
using SilverlightApplication68.Helper;

namespace SilverlightApplication68.Models
{
    public class Employee : INotifyPropertyChanged

    {
        private string _name = string.Empty;
        public string Name
        {
            get { return _name; }
            set
            {
                if (string.IsNullOrEmpty(value))
                {
                    throw new ArgumentException("Name is required");
                }

                if (_name != value)
                {
                    _name = value;
                    PropertyChanged.Raise(() => Name);
                }
            }
        }

        private int _age;
        public int Age
        {
            get { return _age; }
            set
            {
                if (value > 60 || value<25)
```

```
                {
                   throw new
                    ArgumentException("Age should be between 25-60");
                }

                if (_age != value)
                {
                    _age = value;
                    PropertyChanged.Raise(() => Age);
                }
            }
        }

        private string _email = string.Empty;
        public string Email
        {
            get { return _email; }
            set
            {
                if (string.IsNullOrEmpty(value))
                {
                    throw new ArgumentException("Email is required");
                }

                if (!value.Contains("@") && !value.Contains("."))
                {
                    throw new ArgumentException("Email is Invalid");
                }

                if (_email != value)
                {
                    _email = value;
                    PropertyChanged.Raise(() => Email);
                }
            }
        }

        public event PropertyChangedEventHandler PropertyChanged;
    }
}
```

در کلاس Model برنامه، سه خاصیت که بیانگر نام، سن و آدرس ایمیل یک کارمند هستند با پیاده سازی اینترفیس INotifyPropertyChanged، ارائه شده‌اند. در این کلاس دقیقا در لحظه‌ی دریافت مقادیر از کاربر، عملیات اعتبار سنجی آن‌ها صورت گرفته و اگر شرایط مورد نظر تامین نشوند، یک استثناء صادر می‌گردد. این استثناء در سیستم اعتبارسنجی Silverlight منتشر شده و سپس به کاربر ارائه می‌گردد.

برای مثال در View برنامه جهت استفاده از اطلاعات این استثناءها، ویژگی‌های ValidatesOnExceptions و NotifyOnValidationError سیستم Binding به True تنظیم شده‌اند (کدهای کامل XAML مرتبط با View را در ادامه ملاحظه خواهید نمود).

```
<TextBox Grid.Column="1" Text="{
        Binding EmployeeInfo.Age,
        Mode=TwoWay,
        ValidatesOnExceptions=True,
        NotifyOnValidationError=True}"
        Grid.Row="1" Margin="5"  Width="auto" />
```

با کمک خاصیت NotifyOnValidationError از سیستم تعیین اعتبار Silverlight درخواست می‌نمائیم که لطفا خطاهای حاصل از تعیین اعتبار را تحت نظر قرار دهید. علاه بر آن ویژگی BindingValidationError را نیز می‌توان در گرید اصلی View برنامه نیز تعریف نمود. کار این ویژگی، تعریف روال رخداد گردان دریافت خطاهای ناشی از تعیین اعتبار است؛ که عموما اینکار انجام نشده و به سیستم اعتبار سنجی Silverlight واگذار می‌شود. در صورت نیاز، امضای این روال رخدادگردان که در Code behind فایل View برنامه قرار خواهد گرفت و آنچنان با الگوی MVVM سازگار نیست به شرح زیر است:

EmployeeViewModel.cs

```
private void OnValidationError(object sender,
            ValidationErrorEventArgs e)
  {
    switch (e.Action)
    {
      case ValidationErrorEventAction.Added:
        Exception ex = e.Error.Exception;
        break;
      case ValidationErrorEventAction.Removed:
        break;
      default:
        break;
    }
  }
```

اکنون سؤال اینجا است که چه زمانی این خطاها دریافت و تحت کنترل قرار می‌گیرند؟ به همین جهت خاصیت ValidatesOnExceptions نیز به True تنظیم شده است. به این ترتیب به Silverlight خواهیم گفت در زمان دریافت اطلاعات از طریق سیستم Binding (با توجه به Mode=TwoWay)، لطفا هر گونه استثناء صادر شده را به عنوان خطای اعتبار سنجی در نظر گرفته و سبب از کار افتادن برنامه نگردید. علت وجود ArgumentException های تعریف شده در کلاس مدل برنامه نیز همین نکته است.

کنترل ValidationSummary زمانیکه بر روی یک فرم قرار می‌گیرد به کلیه رخدادهای BindingValidationError اشیاء صفحه‌ی جاری گوش فرا داده و خلاصه‌ی خطاهای حاصل را لیست می‌کند.

در ادامه کدهای ViewModel پروژه را ملاحظه می‌فرمائید:

EmployeeViewModel.cs

```
using System.Windows;
using System.Windows.Input;
using SilverlightApplication68.Helper;
using SilverlightApplication68.Models;

namespace SilverlightApplication68.ViewModels
{
    public class EmployeeViewModel
    {
        public ICommand SaveCommand { set; get; }
        public Employee EmployeeInfo { set; get; }

        public EmployeeViewModel()
        {
            EmployeeInfo = new Employee();
            SaveCommand = new DelegateCommand<Employee>(
                saveCommand, canSaveCommand);
        }

        private bool canSaveCommand(Employee arg)
        {
            //TODO: validating model

            return arg != null;
        }

        private void saveCommand(Employee obj)
        {
           //In a real applicaiton implement saving the
           //EmployeeInfo to the database
           //In M-V-VM applications the ViewModel does not raise
           //UI Message Boxes.
           //Instead, a Logger Service is used.
            MessageBox.Show(string.Format("{0} Saved.",
                    EmployeeInfo.Name));
        }
    }
}
```

کلاس ViewModel فوق بسیار ساده بوده و کار آن در اختیار قرار دادن اطلاعات شیء مدل برنامه به View و همچنین مدیریت روال رخدادگردان کلیک بر روی دکمه‌ی ذخیره سازی برنامه است. در این کلاس نیز از کلاس DelegateCommand معروفی که در طی فصول قبلی کتاب توسعه یافتند، استفاده گردیده است و از تکرار مجدد کدهای آن صرفنظر می‌شود.

دو نکته را در این ViewModel باید در نظر داشت:

- بهتر است تعیین اعتبار کلی مدل برنامه در متد canSaveCommand انجام شود. به این صورت تا زمانیکه این مدل، شرایط اعتبار سنجی‌های تعریف شده را تامین نکند، دکمه‌ی Save مرتبط فعال نخواهد شد.
- در متد saveCommand صرفا جهت نمایش اطلاعات وارد شده، از یک MessageBox استفاده گردیده است و باید در نظر داشت که ViewModel برنامه تا حد امکان نباید ارجاع مستقیمی را به عناصر بصری داشته باشد زیرا امکان انجام آزمون‌های واحد آن‌را با مشکل مواجه خواهند کرد.

در ادامه یک تبدیل کننده سفارشی را ایجاد نموده‌ایم. قصد داریم اگر کنترل ValidationSummary دارای تعدادی خطا (خلاصه‌ی خطاهای تعیین اعتبار برنامه) بود، دکمه‌ی ذخیره سازی اطلاعات غیرفعال گردد و برعکس. خاصیت HasErrors این کنترل دقیقا برای همین منظور تدارک دیده شده است اما برای کار ما مناسب نیست. زیرا نقیض آن به معنای عدم وجود خطا است و در این حالت است که قصد داریم کنترل دکمه‌ی ذخیره سازی اطلاعات فعال گردد یا برعکس. به همین جهت نیاز به کلاس تبدیل کننده‌ی سفارشی بعد می‌باشد:

HasErrorToIsEnabledConverter.cs

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

namespace SilverlightApplication68.Converter
{
    public class HasErrorToIsEnabledConverter : IValueConverter
    {
        public object Convert(object value, Type targetType,
            object parameter, CultureInfo culture)
        {
            return (bool)value ? false : true;
        }

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

کدهای XAML متناظر با View اصلی برنامه را در ادامه مشاهده می‌نمائید:

EmployeeView.xaml

```
<UserControl x:Class="SilverlightApplication68.Views.EmployeeView"
    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:SilverlightApplication68.ViewModels"
xmlns:Controls=
"clr-namespace:System.Windows.Controls;assembly=System.Windows.Controls.Data.Input"
xmlns:local="clr-namespace:SilverlightApplication68.Converter"
mc:Ignorable="d" d:DesignHeight="300" d:DesignWidth="400">
<UserControl.Resources>
    <vm:EmployeeViewModel x:Key="viewModel" />
    <local:HasErrorToIsEnabledConverter
             x:Key="HasErrorsToIsEnabledConverter" />
</UserControl.Resources>
<Grid x:Name="LayoutRoot" Background="White"
      DataContext="{Binding Source={StaticResource viewModel}}"
      >
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="88*" />
        <ColumnDefinition Width="184*" />
        <ColumnDefinition Width="28*" />
    </Grid.ColumnDefinitions>
    <Grid.RowDefinitions>
        <RowDefinition Height="Auto" />
        <RowDefinition Height="auto" />
        <RowDefinition Height="auto" />
        <RowDefinition Height="*" />
    </Grid.RowDefinitions>
    <TextBlock Text="Name:"  HorizontalAlignment="Right"
           Margin="5" Grid.Column="0" Grid.Row="0"  />
    <TextBlock Text="Age:" HorizontalAlignment="Right"
           Grid.Row="1" Margin="5" Grid.Column="0"  />
    <TextBlock Text="Email:" HorizontalAlignment="Right"
           Grid.Row="2" Margin="5" Grid.Column="0"  />
    <TextBox Grid.Column="1" Text="{
            Binding EmployeeInfo.Name,
            Mode=TwoWay,
            ValidatesOnExceptions=True,
            NotifyOnValidationError=True}"
            Margin="5"  Width="auto" />
    <TextBox Grid.Column="1" Text="{
            Binding EmployeeInfo.Age,
            Mode=TwoWay,
            ValidatesOnExceptions=True,
            NotifyOnValidationError=True}"
            Grid.Row="1" Margin="5"  Width="auto" />
    <Controls:DescriptionViewer
        Grid.Row="1" Grid.Column="2"
```

```
                Description="Age should be between 25-60" />
        <TextBox Grid.Column="1" Text="{
                Binding EmployeeInfo.Email,
                Mode=TwoWay,
                ValidatesOnExceptions=True,
                NotifyOnValidationError=True}"
                Grid.Row="2" Margin="5"  Width="auto" />
        <Button Content="Save" Grid.Row="3" HorizontalAlignment="Right"
                Command="{Binding SaveCommand}"
                CommandParameter="{Binding EmployeeInfo}"
                Margin="5" VerticalAlignment="Top" Width="75"
                IsEnabled="{Binding
                    ElementName=ValidationSummary1,Path=HasErrors,
           Converter={StaticResource HasErrorsToIsEnabledConverter}}"
                />
        <Controls:ValidationSummary
            Margin="5"
            x:Name="ValidationSummary1"
            Grid.Column="1"
            Grid.Row="3"
            VerticalAlignment="Top" />
    </Grid>
</UserControl>
```

در این View ابتدا سه فضای نام مورد استفاده تعریف شده‌اند:

۱. vm : جهت معرفی فضای نام ViewModels برنامه.

۲. Controls :

برای معرفی فضای نام System.Windows.Controls.Data.Input که پیشتر در مورد آن توضیح داده شد.

۳. local : برای معرفی فضای نام تبدیل کننده‌ی سفارشی تعریف شده است.

سپس در قسمت منابع ثابت این User control ، منابع تبدیل کننده و ViewModel برنامه تعریف شده‌اند. منبع ViewModel به DataContext گرید اصلی صفحه منتسب شده و سپس از این طریق کلیه کنترل‌های موجود در صفحه، منبع اصلی Binding خود را خواهند یافت.

همانطور که پیشتر نیز توضیح داده شد، هر سه کنترل TextBox قرار گرفته بر روی صفحه، دارای ویژگی‌های Binding متناظر با سیستم اعتبار سنجی بوده و به این ترتیب نسبت به خطاهای حاصل به صورت خودکار عکس العمل نشان خواهند داد.

نحوه‌ی انقیاد خاصیت فعال بودن کنترل دکمه به خاصیت HasErrors کنترل ValidationSummary نیز جالب توجه است و تبدیل کننده‌ی تعریف شده، کار ترجمه‌ی صحیح مقادیر HasErrors را عهده دار خواهد بود.

یکی دیگر از خاصیت‌هایی که جزو خواص تعاریف Binding به شمار می‌رود و در سیستم تعیین اعتبار نیز نقش ساز است، UpdateSourceTrigger نام دارد که دو مقدار Default و Explicit را در Silverlight می‌پذیرد. این خاصیت را در تعاریف فوق ملاحظه نمی‌نمائید زیرا مقدار Default آن مورد استفاده قرار گرفته

است. در این حالت پس از رخدادن LostFocus هر کدام از TextBox های قرار گرفته بر روی فرم، کار به روز رسانی اطلاعات متناظر آن‌ها و همچنین تعیین اعتبار اطلاعات دریافتی، انجام می‌شود (این رخداد برای اکثر کنترل‌های Silverlight مساوی PropertyChanged می‌باشد، اما برای TextBox ها LostFocus درنظر گرفته شده است). اگر مقدار آن مساوی Explicit قرار گیرد به این معنا است که کار اعتبار سنجی را به صورت دستی مدیریت خواهید نمود.

و نهایتا از این View در صفحه‌ی اصلی برنامه استفاده گردیده است:

MainPage.xaml

```
<UserControl x:Class="SilverlightApplication68.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:SilverlightApplication68.Views"
    mc:Ignorable="d" d:DesignHeight="300" d:DesignWidth="400">
    <Grid x:Name="LayoutRoot" Background="White">
        <local:EmployeeView />
    </Grid>
</UserControl>
```

تعیین اعتبار به کمک ویژگی‌های داده‌ها

در این قسمت قصد داریم از توانایی‌های فضای نام System.ComponentModel.DataAnnotations استفاده نمائیم. بنابراین لازم است در ابتدا ارجاعی را به این اسمبلی اضافه کنیم.

تاریخچه‌ی فضای نام System.ComponentModel.DataAnnotations به دات نت فریم ورک ۳ و نیم، سرویس پک یک بر می‌گردد؛ زمانیکه تیم ASP.NET ، پروژه‌ی ASP.NET Dynamic Data را معرفی کرد. در این پروژه با اعمال یک سری ویژگی‌ها (attributes) مانند [Required] و [Range] به خواص یک مدل، کار اعتبار سنجی خودکار آن‌ها در WebForms صورت خواهد گرفت. پس از آن از این امکانات در سایر فناوری‌های مرتبط نیز استفاده گردید. این ویژگی‌ها در جدول بعد لیست شده‌اند:

جدول ۱- معرفی Validation attributes

ویژگی (Attribute)	توضیحات	مثال
Required	ضروری بودن یک خاصیت را مشخص می‌نماید (null و یا خالی نباید باشد).	[Required]
StringLength	حداقل و حداکثر طول مجاز یک رشته را می‌توان توسط آن معرفی کرد.	[StringLength(10)]
Range	توسط آن می‌توان بازه‌ای از اعداد مجاز و یا تاریخ را مشخص کرد.	[Range(1, 500)]
RegularExpression	امکان تعریف عبارات باقاعده را جهت اعتبار سنجی خاصیت مورد نظر فراهم می‌آورد.	[RegularExpression("[1-9][0-9]{3}")]
CustomValidation	متدی سفارشی را جهت تعیین اعتبار مشخص می‌سازد.	CustomValidation(typeof(MyValidatorClass) ,"ValidateDateMethod")]

از کلاس ValidationHelper که کدهای آن‌را در ادامه ملاحظه خواهید نمود جهت سهولت کار با DataAnnotations استفاده می‌شود. از متد GetErrors آن می‌توان جهت تعیین اعتبار کلی یک شیء پیش از ذخیره سازی اطلاعات آن در بانک اطلاعاتی برنامه استفاده کرد. اگر تعداد خطاهایی که بر می‌گرداند مساوی صفر باشد به این معنا است که کاربر اطلاعات صحیحی را مطابق قیود تعریف شده‌ی ما وارد کرده است یا برعکس. از متد ValidateProperty این کلاس برای تعیین اعتبار ورودی کاربر دقیقا در لحظه‌ای که اطلاعات را وارد می‌کند می‌توان کمک جست. این متد نیز از lambda expressions جهت معرفی و یافتن نام خاصیت تحت نظر استفاده می‌کند و از رشته‌های خام در اینجا استفاده نشده است. این کلاس از متدهای کلاس Extensions که در مثال قبل معرفی شدند، استفاده می‌نماید.

ValidationHelper.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Linq.Expressions;
using System.Reflection;

namespace SilverlightApplication68.Helper
{
    public class ValidationHelper
    {
        public static IList<ValidationResult>
                    GetErrors(object instance)
        {
            var res = new List<ValidationResult>();
```

```
            Validator.TryValidateObject(instance,
                new ValidationContext(instance, null, null),
                res, true);
            return res;
        }

        public static void ValidateProperty(object value,
            Expression<Func<object>> expression)
        {
            PropertyInfo propertyInfo;
            var constantExpression =
               expression.GetPropertyInfo(out propertyInfo);
            Validator.ValidateProperty(value,
                new ValidationContext(
                     constantExpression.Value, null, null)
                {
                    MemberName = propertyInfo.Name
                });
        }
    }
}
```

بر این اساس، همان کلاس کارمند مثال قبل را به صورت ذیل بازنویسی خواهیم کرد :

Employee.cs

```
using System.ComponentModel;
using System.ComponentModel.DataAnnotations;
using SilverlightApplication68.Helper;

namespace SilverlightApplication68.Models
{
    public class Employee : INotifyPropertyChanged
    {
        private string _name = string.Empty;
        [Required(ErrorMessage = "{0} is required.")]
        [StringLength(15, MinimumLength = 6,
            ErrorMessage = "{0} len must be between {1} and {2}")]
        public string Name
        {
            get { return _name; }
            set
            {
                if (_name != value)
                {
                  ValidationHelper.ValidateProperty(value, () => Name);
                  _name = value;
                  PropertyChanged.Raise(() => Name);
                }
            }
```

```
        }

        private int _age;
        [Required(ErrorMessage = "{0} is required.")]
        [Range(25, 60,
            ErrorMessage = "{0} must be set between {1} and {2}")]
        public int Age
        {
            get { return _age; }
            set
            {
                if (_age != value)
                {
                  ValidationHelper.ValidateProperty(value, () => Age);
                  _age = value;
                  PropertyChanged.Raise(() => Age);
                }
            }
        }

        private string _email = string.Empty;
        [Required(ErrorMessage = "{0} is required.")]
        [RegularExpression(
            @"\w+([-+.']\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*",
            ErrorMessage = "Please enter a valid email.")]
        public string Email
        {
            get { return _email; }
            set
            {
                if (_email != value)
                {
                  ValidationHelper.ValidateProperty(value, () => Email);
                  _email = value;
                  PropertyChanged.Raise(() => Email);
                }
            }
        }

        #region INotifyPropertyChanged Members
        public event PropertyChangedEventHandler PropertyChanged;
        #endregion
    }
}
```

همچنین اکنون با کمک متد GetErrors کلاس ValidationHelper که پیشتر معرفی گردید، می‌توان کلیه خواص مدل برنامه را قبل از هرگونه استفاده‌ای از آن به صورت کلی اعتبار سنجی نمود:

EmployeeViewModel.cs

```
...
private void saveCommand(Employee obj)
{
   //validating model
   var errorInfos = ValidationHelper.GetErrors(obj);
   if(errorInfos.Any()) // using System.Linq;
   {
      MessageBox.Show("Model is not valid");
      return;
   }

   MessageBox.Show(string.Format("{0} Saved.", obj.Name));
}
...
```

تعریف ویژگی‌های سفارشی

ویژگی‌های قابل اعمال به داده‌ها با استفاده از ویژگی CustomValidation به سادگی قابل توسعه هستند. در ادامه یک کلاس که متد سفارشی تعیین اعتبار واحدها را ارائه می‌دهد ملاحظه می‌نمائید. خروجی این متد باید از نوع bool بوده و تنها یک پارامتر را از نوع خاصیتی که قرار است توسط آن اعتبار سنجی گردد، دریافت نماید.

MyValidations.cs

```
namespace SilverlightApplication68.CustomValidations
{
    public class MyValidations
    {
        public static bool UnitsValidation(int units)
        {
            return units < 100;
        }
    }
}
```

اکنون نحوه‌ی اعمال این ویژگی سفارشی تعریف شده به صورت زیر می‌تواند باشد:

```
[CustomValidation(typeof(MyValidations), "UnitsValidation")]
public int UnitsOnOrder;
```

پیاده سازی اینترفیس IDataErrorInfo جهت تعیین اعتبار اشیاء

جهت تکمیل امکانات تعیین اعتبار ورودی کاربران در Silverlight 4 ، اینترفیس IDataErrorInfo نیز به این مجموعه اضافه گردیده است. این اینترفیس برای برنامه نویسان WPF و یا WinForms مطلب جدیدی نیست و به درخواست آن‌ها اضافه گردیده است. امضای آن‌را در ادامه مشاهده می‌نمائید:

C#

```
public interface IDataErrorInfo
{
  string Error { get; }
  string this[string columnName] { get; }
}
```

با پیاده سازی خاصیت Error آن، می‌توان منطق سفارشی تعیین اعتبار ویژه‌ای را بر اساس کلاس‌ها و اشیاء دیگر برنامه ارائه داد (خاصیت Error برای Silverlight این به این معنا است: به من بگو کلا چه مشکلی در تعیین اعتبار شیء جای وجود دارد؟). مهم‌ترین قسمت آن ، خاصیت this می‌باشد که یک Indexer ارائه دهنده‌ی نام خواص تعیین اعتبار شونده‌ی شیء جاری است. برای این اساس می‌توان به ازای هر خاصیت، منطق ویژه‌ی اعتبار سنجی خاصی را ارائه داد (این Indexer برای Silverlight به این معنا است: به من بگو این خاصیت نامبرده شده از لحاظ تعیین اعتبار چه مشکلی دارد؟). در اینجا بدون ایجاد استثناءها می‌توان به تعیین اعتبار مقادیر خاصیت‌های یک شیء پرداخت. همچنین دیگر نیازی به بررسی مقادیر در محل تنظیم کننده‌ی set خواص نیز نخواهد بود.

پس از پیاده سازی اینترفیس IDataErrorInfo ، اکنون نوبت به شناساندن آن به سیستم Binding فرا می‌رسد. برای این منظور تنها کافی است خواص ValidatesOnDataErrors و NotifyOnValidationError به true تنظیم شوند. برای مثال :

XAML

```
<TextBox Text="{Binding Name, Mode=TwoWay,
                        ValidatesOnDataErrors=True,
                        NotifyOnValidationError=True}" />
```

برای توضیح کاربردی مباحث توضیح داده شده، لطفا به پیاده سازی دیگری از کلاس کارمند فصل جاری دقت بفرمائید:

Employee.cs

```
using System.ComponentModel;
using SilverlightApplication68.Helper;
namespace SilverlightApplication68.Models
{
    public class Employee : INotifyPropertyChanged, IDataErrorInfo
    {
        private string _name = string.Empty;
        public string Name
        {
            get { return _name; }
            set
            {
                if (_name != value)
                {
                    _name = value;
                    PropertyChanged.Raise(() => Name);
                }
            }
        }

        private int _age;
        public int Age
        {
            get { return _age; }
            set
            {
                if (_age != value)
                {
                    _age = value;
                    PropertyChanged.Raise(() => Age);
                }
            }
        }


        private string _email = string.Empty;
        public string Email
        {
            get { return _email; }
            set
            {
                if (_email != value)
                {
                    _email = value;
```

```
                    PropertyChanged.Raise(() => Email);
            }
        }
    }

    public event PropertyChangedEventHandler PropertyChanged;


    #region IDataErrorInfo Members
    public string Error
    {
        get { return null; }
    }

    public string this[string columnName]
    {
        get
        {
            switch (columnName)
            {
                case "Name":
                    if (string.IsNullOrEmpty(Name))
                        return "Name is required";
                    break;
                case "Age":
                    if (Age > 60 || Age < 25)
                        return "Age should be between 25-60";
                    break;
                case "Email":
                    if (string.IsNullOrEmpty(Email))
                    {
                        return "Email is required";
                    }

                    if (!Email.Contains("@") &&
                                !Email.Contains("."))
                    {
                        return "Email is Invalid";
                    }
                    break;
                default:
                    break;
            }
            return string.Empty;
        }
    }
    #endregion
  }
}
```

همانطور که در کدهای قبل نیز مشخص است، به کمک Indexer اینترفیس IDataErrorInfo ، امکان دریافت نام خواص مدل وجود دارد. سپس مقدارهای منتسب به هر کدام از خواص شیء جاری بررسی شده و در صورت وجود خطایی در منطق تعیین اعتبار آن‌ها، تنها کافی است رشته‌ای حاوی متن خطای مورد نظر ارائه گردد. این رشته به صورت خودکار به کاربر نمایش داده خواهد شد.

آشنایی با نحوه‌ی تعیین اعتبار غیر همزمان (Asynchronous) اشیاء

اینترفیس دیگری به Silverlight 4 اضافه شده است به نام INotifyDataErrorInfo که هدف آن سازگاری بیشتر با رفتار پیش فرض غیرهمزمان Silverlight و میسر ساختن تعیین اعتبار غیرهمزمان (asynchronous validation) می‌باشد. امضای این اینترفیس به شرح زیر است:

C#

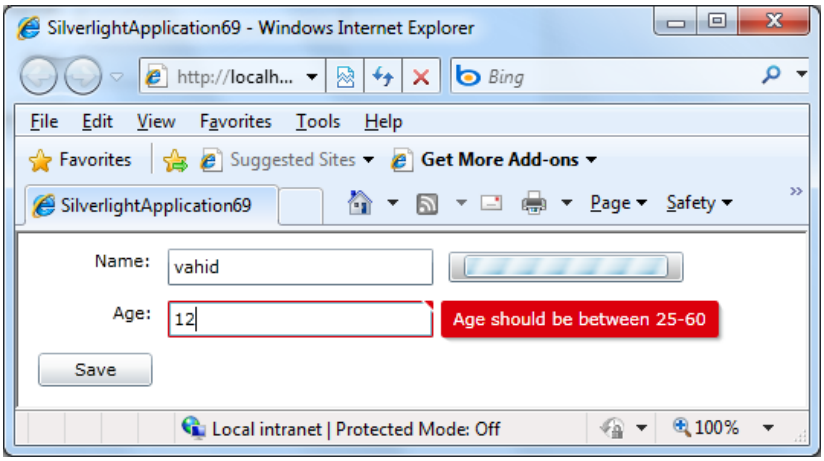
```
public interface INotifyDataErrorInfo
{
  bool HasErrors { get; }
  event EventHandler<DataErrorsChangedEventArgs> ErrorsChanged;
  IEnumerable GetErrors(string propertyName);
}
```

پس از پیاده سازی آن، با کمک خاصیت HasErrors می‌توان وضعیت اعتبار کل یک شیء را مشخص ساخت؛ موردی که در IDataErrorInfo پیش بینی نشده است. توسط متد GetErrors ، اطلاعات بیشتری را در مقایسه با یک رشته ساده بازگشتی در Indexer مرتبط با IDataErrorInfo می‌توان ارائه داد. همچنین در این روش یک خاصیت می‌تواند بیش از یک خطای اعتبار سنجی را نیز ارائه دهد.

رخداد ErrorsChanged در اینجا جهت کار با اعمال طولانی که نیاز به تعیین اعتبار دارند، بسیار مناسب است؛ برای مثال مراجعه به یک WCF Service ، دریافت اطلاعاتی از بانک اطلاعاتی و سپس تعیین اعتبار مقادیر دریافتی از کاربر بر این اساس. در اینجا اگر پس از پایان عملیات و بررسی منطق اعتبار سنجی ، مشکلی وجود داشته باشد، می‌توان به کمک رخداد ErrorsChanged ، رابط کاربر را جهت ارائه‌ی پیغام‌های لازم به کاربر مطلع ساخت. در این حالت حتما نیاز است تا در خواص Binding تعریف شده در رابط کاربر برنامه، خاصیت ValidatesOnNotifyDataErrors به true تنظیم گردد تا رابط کاربر به پیغام‌های صادره از طرف رخداد ErrorsChanged گوش فرا دهد.

بررسی یک مثال کاربردی

جهت درک بهتر کاربرد اینترفیس INotifyDataErrorInfo، لطفا به مثال بعد دقت بفرمائید. در این مثال قصد داریم در حین ثبت نام یک کاربر درسایت، نام کاربری او را با اطلاعات دریافتی از یک WCF Service مقایسه نموده و در صورت وجود نام کاربری مشابه، پیغام خطایی را به او نمایش دهیم (شکل ۳).

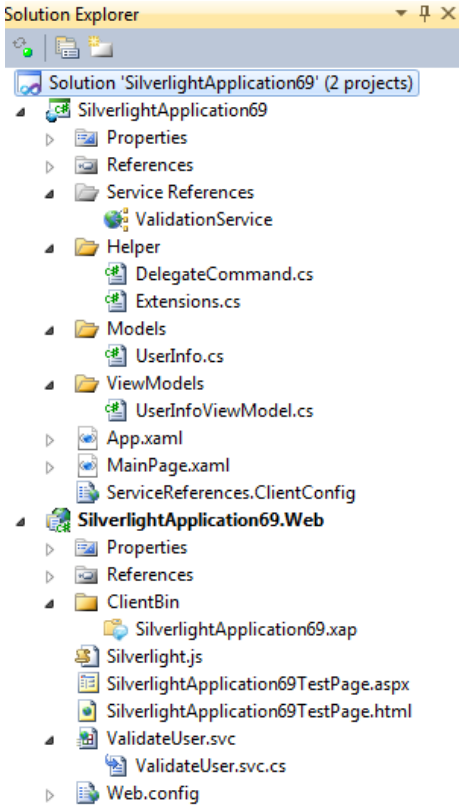


شکل ۳– نمایی از مثال تعیین اعتبار غیرهمزمان

برای این منظور یک پروژه‌ی جدید Silverlight را به همراه ASP.NET Web site آن آغاز نمائید. نمایی از ساختار فایل‌های و پوشه‌های این پروژه را در شکل ۴ ملاحظه می‌نمائید.

در این مثال از دو کلاس DelegateCommand و Extensions که در مثال‌های قبل کتاب جاری توسعه یافتند استفاده خواهد شد و از ذکر مجدد کدهای آن‌ها خودداری می‌گردد. همچنین از ذکر مباحث امنیتی مرتبط با WCF Service نیز صرفنظر می‌شود. بدیهی است موارد امنیتی یاد شده در فصل امنیت در Silverlight را باید به کلیه برنامه‌ها و پروژه‌های کاربردی خود پیش از ارائه‌ی نهایی آن‌ها اعمال نمائید.

پس از ایجاد ساختار اولیه پروژه، از منوی پروژه، گزینه‌ی Add new item ، یک Silverlight-enabled WCF Service را به نام ValidateUser.svc مطابق شکل ۴ به پروژه‌ی ASP.NET برنامه اضافه کنید. کدهای این WCF Service در ادامه ارائه شده‌اند:



شکل ۴– ساختار فایل‌های و پوشه‌های پروژه تعیین اعتبار غیر همزمان

ValidateUser.svc.cs

```
using System.ServiceModel;
using System.ServiceModel.Activation;
using System.Threading;
namespace SilverlightApplication69.Web
{
    [ServiceContract(Namespace = "")]
    [AspNetCompatibilityRequirements(RequirementsMode =
        AspNetCompatibilityRequirementsMode.Allowed)]
    public class ValidateUser
    {
        [OperationContract]
        public bool IsValid(string userName)
        {
            //TODO: رعایت مسایل امنیتی ذکر شده در فصل امنیت
            //TODO: منطق دریافت اطلاعات از بانک اطلاعاتی را در اینجا پیاده سازی نمائید
            Thread.Sleep(2000); //شبیه سازی عملیاتی طولانی
            return userName.ToLower().Trim() != "vahid";
        }
    }
}
```

توسط این WCF Service نام کاربری دریافت شده با کلمه‌ی vahid مقایسه شده و در صورت تساوی، false بازگشت داده خواهد شد.

در ادامه به پروژه‌ی Silverlight مراجعه نموده و از منوی پروژه، گزینه‌ی Add service reference را انتخاب کنید. در صفحه‌ی باز شده روی دکمه‌ی Discover کلیک نمائید. این دکمه قابلیت تشخیص سرویس‌های تعریف شده در سطح پروژه‌ی جاری را دارد. پس از انتخاب سرویس مورد نظر، نام ValidationService را وارد نموده و بر روی دکمه‌ی OK کلیک نمائید.

کدهای کلاس Model برنامه را که عملیات اعتبار سنجی نیز در آن قرار گرفته است، در ادامه ملاحظه خواهید نمود:

UserInfo.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Linq;
using SilverlightApplication69.Helper;

namespace SilverlightApplication69.Models
{
    public class UserInfo : INotifyPropertyChanged, INotifyDataErrorInfo
    {
        private string _name = string.Empty;
        public string Name
        {
            get { return _name; }
            set
            {
                if (_name != value)
                {
                    _name = value;
                    validateNameProperty();
                    PropertyChanged.Raise(() => Name);
                }
            }
        }

        private void validateNameProperty()
        {
            IsBusy = true;
            clearValidationErrors("Name");

            var srv = new ValidationService.ValidateUserClient();
            srv.IsValidCompleted += (o, h) =>
            {
```

```
            if (h.Error != null)
                return;

            string errMsg = string.Format(
              "Account with the ID {0} already exists", Name);
            if (!h.Result)
            {
                addValidationError("Name",errMsg);
            }
            else
            {
                removeValidationError("Name", errMsg);
            }

            IsBusy = false;
        };
        srv.IsValidAsync(Name);
    }

    private int _age;
    public int Age
    {
        get { return _age; }
        set
        {
            if (_age != value)
            {
                _age = value;
                validateAgeProperty();
                PropertyChanged.Raise(() => Age);
            }
        }
    }

    private void validateAgeProperty()
    {
        const string errMsg = "Age should be between 25-60";
        if (Age > 60 || Age < 25)
        {
             addValidationError("Age", errMsg);
        }
        else
        {
            removeValidationError("Age", errMsg);
        }
    }

    private bool _isBusy;
```

```
       public bool IsBusy
{
    get { return _isBusy; }
    set
    {
        if (_isBusy != value)
        {
            _isBusy = value;
            PropertyChanged.Raise(() => IsBusy);
        }
    }
}

public event PropertyChangedEventHandler PropertyChanged;


#region INotifyDataErrorInfo Members

public event EventHandler<DataErrorsChangedEventArgs>
             ErrorsChanged;

public System.Collections.IEnumerable GetErrors(
        string propertyName)
{
    if (_errorList != null &&
               _errorList.ContainsKey(propertyName))
        return _errorList[propertyName];

    return null;
}

public bool HasErrors
{
    get
    {
        if (_errorList == null)
            return false;

        return _errorList.Any();
    }
}

Dictionary<string, List<object>> _errorList;
private void addValidationError(string propName, object error)
{
    if (_errorList == null)
        _errorList = new Dictionary<string, List<object>>();
    if (!_errorList.ContainsKey(propName))
        _errorList[propName] = new List<object>();
    if (!_errorList[propName].Contains(error))
```

```
            {
                _errorList[propName].Add(error);
                OnErrorsChanged(propName);
            }
        }

        private void removeValidationError(string propName,
                        object error)
        {
            if (_errorList == null)
                _errorList = new Dictionary<string, List<object>>();
            if (!_errorList.ContainsKey(propName))
                _errorList[propName] = new List<object>();
            if (_errorList[propName].Contains(error))
            {
                _errorList[propName].Remove(error);
                OnErrorsChanged(propName);
            }
        }

        private void clearValidationErrors(string propName)
        {
            if (_errorList == null)
                return;
            if (!_errorList.ContainsKey(propName))
                return;
            _errorList[propName] = new List<object>();
            OnErrorsChanged(propName);
        }

        protected void OnErrorsChanged(string propName)
        {
            if (ErrorsChanged != null)
                ErrorsChanged(this,
                   new DataErrorsChangedEventArgs(propName));
        }

        #endregion
    }
}
```

مدل ساده‌ی برنامه از دو خاصیت نام و سن شخص ثبت نام کننده تشکیل می‌گردد. هدف از ارائه‌ی خاصیت IsBusy ، نمایش کنترل BusyIndicator در طی ۲ ثانیه تاخیری است که در سمت WCF Service ایجاد نمودیم. کلاس مدل برنامه دو اینترفیس INotifyPropertyChanged و INotifyDataErrorInfo را پیاده سازی نموده است.

هنگام استفاده از اینترفیس INotifyDataErrorInfo مرسوم است که به ازای هر خاصیت تعریف شده یک متد تعیین اعتبار نیز در ذیل آن تعریف گردد که در کدهای فوق مشخص هستند.

توسط متد validateNameProperty، کار تعیین اعتبار غیرهمزمان خاصیت Name انجام می‌شود (که جزئیات اعمال آن همانند مطالب عنوان شده در طی فصل آشنایی با بکارگیری وب سرویس‌ها است و نکته‌ی جدیدی ندارد). از آنجائیکه این عملیات طولانی است نیاز است تا در ابتدای کار کلیه خطاهای مرتبط با این خاصیت از سیستم حذف شده و سپس در پایان عملیات، پیغام‌های مناسبی در صورت لزوم به سیستم اضافه گردند. این پاک سازی اولیه در حین اعتبار سنجی خاصیت سن صورت نگرفته است زیرا کدهای آن وابسته به یک WCF Service خارجی نبوده و آنی است.

یک سری متد کمکی نیز به این کلاس جهت افزودن حذف خطاها اضافه شده است. همچنین نحوه‌ی پیاده سازی متد GetErrors و خاصیت HasErrors نیز ذکر گردیده است.

باتوجه به تکراری بودن اینگونه عملیات، مرسوم است که یک کلاس کمکی را مثلا به نام EntityBase ، با پیاده سازی اینترفیس‌های یاد شده ، تشکیل داده و سپس کلاس مدل برنامه را از آن به ارث می‌برند.

کدهای ViewModel برنامه را در ادامه ملاحظه می‌نمائید:

UserInfoViewModel.cs

```
using System.Windows;
using System.Windows.Input;
using SilverlightApplication69.Helper;
using SilverlightApplication69.Models;

namespace SilverlightApplication69.ViewModels
{
    public class UserInfoViewModel
    {
        public ICommand SaveUserInfo { set; get; }
        public UserInfo UserInfo { set; get; }

        public UserInfoViewModel()
        {
            UserInfo = new UserInfo();
            SaveUserInfo = new DelegateCommand<UserInfo>(
                saveUserInfo, canSaveUserInfo);
        }

        private bool canSaveUserInfo(UserInfo arg)
        {
            return true;
        }

        private void saveUserInfo(UserInfo obj)
```

```
        {
            if (obj.HasErrors)
            {
                MessageBox.Show("Model is not valid");
                return;
            }
            MessageBox.Show(string.Format("{0} Saved!", obj.Name));
        }
    }
}
```

این کلاس کار مدیریت روال رخدادگردان ذخیره سازی اطلاعات کاربر را برعهده دارد. همچنین یک خاصیت عمومی اطلاعات کاربر را نیز در اختیار View برنامه یا همان صفحه‌ای اصلی برنامه قرار می‌دهد. نکته‌ی مهم آن بررسی خاصیت HasErrors شیء مدل، پیش از اقدام به هرگونه عملیاتی با محتوای آن می‌باشد.

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

MainPage.xaml

```
<UserControl x:Class="SilverlightApplication69.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:SilverlightApplication69.ViewModels"
    mc:Ignorable="d" d:DesignHeight="300" d:DesignWidth="400"
    xmlns:toolkit=
     "http://schemas.microsoft.com/winfx/2006/xaml/presentation/toolkit">
    <UserControl.Resources>
        <vm:UserInfoViewModel x:Key="vmUserInfoViewModel" />
    </UserControl.Resources>
        <Grid x:Name="LayoutRoot" Background="White" Margin="5"
              DataContext="{Binding
                 Source={StaticResource vmUserInfoViewModel}}"
              >
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="88" />
            <ColumnDefinition Width="184" />
            <ColumnDefinition Width="*" />
        </Grid.ColumnDefinitions>
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto" />
            <RowDefinition Height="auto" />
            <RowDefinition Height="*" />
        </Grid.RowDefinitions>
```

```
            <TextBlock Text="Name:"  HorizontalAlignment="Right"
            Margin="5" Grid.Column="0" Grid.Row="0"  />
        <TextBlock Text="Age:" HorizontalAlignment="Right"
            Grid.Row="1" Margin="5" Grid.Column="0"  />
        <TextBox Grid.Column="1" Text="{
                Binding UserInfo.Name,
                Mode=TwoWay,
                ValidatesOnDataErrors=True,
                NotifyOnValidationError=True}"
                 Margin="5"  Width="auto" />
        <TextBox Grid.Column="1" Text="{
                Binding UserInfo.Age,
                Mode=TwoWay,
                ValidatesOnDataErrors=True,
                NotifyOnValidationError=True}"
                 Grid.Row="1" Margin="5"  Width="auto" />
        <Button Content="Save" Grid.Row="2" HorizontalAlignment="Right"
                Command="{Binding SaveUserInfo}"
                CommandParameter="{Binding UserInfo}"
                Margin="5" VerticalAlignment="Top" Width="75"
                />
        <toolkit:BusyIndicator
            Grid.Column="2"
            Grid.Row="0"
            Margin="5"
            Height="20"
            IsBusy="{Binding UserInfo.IsBusy}"
            HorizontalAlignment="Left"
            BusyContent="Validating..." />
    </Grid>
</UserControl>
```