

Markup Extension ها برای مواردی استفاده می‌شوند که قرار است مقداری غیر از یک مقدار ثابت و یک نوع قابل شناسایی در XAML برای یک value تنظیم شود. تمام مواردی در XAML که درون { } قرار می‌گیرند همان Markup Extension ها هستند. مانند Binding و یا StaticResources. علاوه بر Markup Extension های از پیش تعریف شده در XAML، می‌توان Markup Extension های شخصی را نیز تولید کرد. در واقع به زبان ساده‌تر Markup Extension برای تولید ساده‌ی داده‌های پیچیده در XAML استفاده می‌شوند.

لازم به ذکر است که Markup Extension ها می‌توانند به دو صورت Attribute Usage، درون { } :

```
"{Binding path=something,Mode=TwoWay}"
```

و یا Property Element Usage (همانند سایر Element های WPF) درون <> استفاده شوند:

```
<Binding Path="Something" Mode="TwoWay"></Binding>
```

برای تعریف یک Markup Extension، یک کلاس ایجاد می‌کنیم که از Markup Extensions ارث بری می‌کند. این کلاس یک Abstract Method به نام ProvideValue دارد که باید پیاده سازی شود. این متد مقدار خصوصیتی که Markup Extensions را فراخوانی کرده به صورت یک Object بر می‌گرداند که یکبار در زمان Load برای خصوصیت مربوطه اش تنظیم می‌شود.

```
public abstract Object ProvideValue(IServiceProvider serviceProvider)
```

همانطور که ملاحظه می‌کنید ProvideValue یک پارامتر IServiceProvider دارد که از طریق آن می‌توان به [IProvideValueTarget](#) دسترسی داشت. از این Interface برای گرفتن اطلاعات کنترل (TargetObject) و خصوصیتی ([TargetProperty](#)) که فراخوانی را انجام داده در صورت لزوم استفاده می‌شود.

```
var target = serviceProvider.GetService(typeof(IProvideValueTarget))as IProvideValueTarget;
var host = target.TargetObject as FrameworkElement;
```

Markup Extension ها می‌توانند پارامترهای ورودی داشته باشند:

```
public class ValueExtension : MarkupExtension
{
    public ValueExtension () { }
    public ValueExtension (object value1)
    {
        Value1 = value1;
    }
    public object Value1 { get; set; }
    public override object ProvideValue(IServiceProvider serviceProvider)
    {
        return Value1;
    }
}
```

و برای استفاده در فایل Xaml:

```
<TextBox Text="{app:ValueExtension test}" ></TextBox>
```

و یا می‌توان خصوصیت هایی ایجاد کرد و از آنها برای ارسال مقادیر به آن استفاده کرد:

```
<TextBox Text="{app:ValueExtension Value1=test}" ></TextBox>
```

تا اینجا موارد کلی برای تعریف و استفاده از Markup Extensions گفته شد. در ادامه یک مثال کاربردی می‌آوریم. برای مثال در نظر بگیرید که نیاز دارید DataType مربوط به یک DataTemplate را برابر یک کلاس Generic قرار بدهید:

```
public class EntityBase
{
    public int Id{get;set}
}

public class MyGenericClass<TType> where TType : EntityBase
{
    public int Id{get;set}
    public string Test{ get;set; }
```

In XAML:

```
<DataTemplate DataType="{app:GenericType ?}">
```

برای انجام این کار یک Markup Extensions به صورت زیر ایجاد می‌کنیم که Type را به عنوان ورودی گرفته و یک نمونه از کلاس Generic ایجاد می‌کند:

```
public class GenericType : MarkupExtension
{
    private readonly Type _of;
    public GenericType(Type of)
    {
        _of = of;
    }
    public override object ProvideValue(IServiceProvider serviceProvider)
    {
        return typeof(MyGenericClass<>)MakeGenericType(_of);
    }
}
```

و برای استفاده از آن یک نمونه از MarkupExtension ایجاد شده ایجاد کرده و نوع Generic را برای آن ارسال می‌کنیم:

```
<DataTemplate DataType="{app:GenericType app:EntityBase}">
```

این یک مثال ساده از استفاده از Markup Extensions است. هنگام کار با WPF می‌توان استفاده‌های زیادی از این مفهوم داشت، برای مثال زمانی که نیاز است ItemsSource یک Combobox از Description‌های یک Enum پر شود می‌توان به راحتی با نوشتن یک Markup Extensions ساده این عمل و کارهای مشابه زیادی را انجام داد.

نظرات خوانندگان

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۵/۰۸ ۲۳:۵۰

یک مثال جالب در این مورد

[DelayBinding: a custom WPF Binding](#)

اغلب در حین Bind کردن Property ها در XAML به مشکل Bind نشدن بر می‌خوریم. من معمولا از روش زیر استفاده می‌کنم:

```
public class DatabindingDebugConverter : IValueConverter
{
    #region IValueConverter Members

    public object Convert(object value, Type targetType, object parameter,
System.Globalization.CultureInfo culture)
    {
        Debugger.Break();
        return value;
    }

    public object ConvertBack(object value, Type targetType, object parameter,
System.Globalization.CultureInfo culture)
    {
        Debugger.Break();
        return value;
    }

    #endregion IValueConverter Members
}
```

و در XAML :

```
<DataTemplate.Resources>
    <debug:DatabindingDebugConverter x:Key="databindingDebugConverter"/>
</DataTemplate.Resources>
<DataGrid ItemsSource="{Binding myViewModel,Converter={StaticResource databindingDebugConverter}}"/>
```

و حال دو حالت می‌تواند اتفاق بیفتد :

1 - Break Point Hit نمی‌شود:

در این حالت مقدار myViewModel خالی (null) است و یا اصلا myViewModel در DataContext مربوط به DataGrid وجود ندارد در این صورت همچنین در پنجره Out Put Visual Studio:

```
System.Windows.Data Error: 35 : BindingExpression path error: 'X' property not found ...
```

و با search متن "System.Windows.Data Error: 35 : BindingExpression path error" در Out Put میتوان متوجه آن شد.

2 - Break Point Hit می‌شود:

در این حالت باید value را Watch کنیم (Shift+F9) تا ببینیم علت Bind نشدن چیست؟ شاید (در این مورد خاص) نوع myViewModel از IEnumerable نباشد ...

در حین بررسی و Debug ، شاید گاهی مسئله لاینحل به نظر برسد ، ولی به نظر من معمولا با کم و زیاد کردن آدرس (Binding Path) به یکی از دو حالت بالا خواهیم رسید ،

مثلا زمانی که Path به صورت myViewModel.MyProperty.MyInnerPtoperty است ، باید Path را با حالات زیر توسط Converter مذکور تست کنیم:

```
Binding"{Path=myViewModel.MyProperty.MyInnerPtoperty ,Converter="{StaticResource debugger}}"}"
Binding"{Path=myViewModel.MyProperty,Converter="{StaticResource debugger}}"}"
Binding"{Path=myViewModel,Converter="{StaticResource debugger}}"}"
Binding"{Path=.,Converter="{StaticResource debugger}}"}"
```

امیدوارم از Binding تان لذت ببرید.

شاید تا به حال در یک برنامه سازمانی نیاز به Bind کردن یک Enum به کنترل‌های XAML به چشمتان خورده باشد ، روشی که من برای این کار استفاده می‌کنم توسط یک [Markup Extension](#) به صورت زیر است :

```
public class ByteEnumerationExtention : MarkupExtension
{
    public ByteEnumerationExtention(Type enumType)
    {
        this.enumType = enumType;
    }

    private Type enumType;

    public Type EnumType
    {
        get { return enumType; }
        private set
        {
            enumType = value;
        }
    }

    public override object ProvideValue(IServiceProvider serviceProvider)
    {
        return (from x in Enum.GetValues(EnumType).OfType<Enum>()
                select new EnumerationMember
                {
                    Value = GetValue(x),
                    Description = GetDescription(x)
                }).ToArray();
    }

    private byte? GetValue(object enumValue)
    {
        return Convert.ToByte(enumValue.GetType().GetField("value__").GetValue(enumValue));
    }

    public object GetObjectValue(object enumValue)
    {
        return enumValue.GetType().GetField("value__").GetValue(enumValue);
    }

    public string GetDescription(object enumValue)
    {
        var descAttrib = EnumType.GetField(enumValue.ToString())
            .GetCustomAttributes(typeof(DescriptionAttribute), false)
            .FirstOrDefault() as DescriptionAttribute;
        return descAttrib != null ? descAttrib.Description : enumValue.ToString();
    }
}

public class EnumerationMember
{
    public string Description { get; set; }

    public byte? Value { get; set; }
}
```

: XAML

```
<ComboBox ItemsSource="{Binding Source={ Extensions:ByteEnumerationExtention {x:Type type:MyEnum} }}"
    DisplayMemberPath="Description"
    SelectedValuePath="Value" SelectedValue="{Binding SelectedItemInViewModel}"/>
```

: Enum

```
public enum MyEnum : short
{
    [Description("گزینه 1")]
    Item1 = 1,

    [Description("گزینه 2")]
    Item2 = 2,

    [Description("گزینه 3")]
    Item3 = 3,

    [Description("گزینه 4")]
    Item4 = 4,

    [Description("گزینه 5")]
    Item5 = 5,

    .
    .
    .
}
```

: ViewModel در SelectedItem

```
short? selectedItemInViewModel;
public short? SelectedItemInViewModel
{
    get
    {
        return selectedItemInViewModel;
    }
    set
    {
        selectedItemInViewModel = value;
        RaisePropertyChanged("SelectedItemInViewModel");
        //do calculations if needed
    }
}
```

[MarkupExtension](#) ها قبلا در اینجا توضیح داده شده اند. یکی از MarkupExtension های از پیش تعریف شده [x:Static](#) است که برای مقداردهی یک خصوصیت در XAML با یک مقدار استاتیک استفاده می‌شود. اگر بخواهید از یک ثابت (constant)، یک خصوصیت استاتیک (static property)، یا یک مقدار از یک enumeration، برای مقداردهی یک خصوصیت در XAML استفاده کنید باید از این MarkupExtension استفاده کنید.

برای مثال برای یک استفاده از یک خصوصیت استاتیک به صورت زیر عمل می‌کنیم:

```
namespace Test
{
    public class Constants
    {
        public static readonly string ConstantString = "Test";
    }
}
```

توجه داشته باشید که برای استفاده از این ثابت باید ابتدا فضای نام مربوط به آن را تعریف کنید.

```
xmlns:test="clr-namespace:ItemTest "
<Label Content="{x:Static test:Constants.ConstantString}" />
```

و یا برای مقدار دهی از طریق یک Enumeration

```
namespace Test
{
    public enum VisibilityEnum
    {
        Collapse,
        Hidden,
        Visible
    };
}
```

و در فایل XAML:

```
xmlns:test="clr-namespace:Test"
<Label Content="{x:Static test:VisibilityEnum.Collapse}" />
```

برای استفاده از یک ثابت نیز به همین صورت عمل می‌کنیم.

کاربران بیشتر برنامه های فارسی تمایل دارند که توسط کلید Enter درون فرم ها حرکت کنند. در برنامه های WPF و مخصوصا زمانی که شما از الگوی MVVM استفاده می کنید، انجام این کار اگر از روش های مناسب استفاده نکنید تا حدودی سخت می شود. برای حرکت روی TextBox ها و کنترل های مشابه می توانید این کار را به راحتی با Register کردن رویداد مربوط به آن نوع کنترل ها توسط [EventManager](#) یک بار در ابتدای برنامه انجام دهید.

```
public partial class App : Application
{
    EventManager.RegisterClassHandler(typeof(TextBox), TextBox.KeyDownEvent, new
    KeyEventHandler(TextBox_KeyDown));
    ...
}
private void TextBox_KeyDown(object sender, KeyEventArgs e)
{
    if (e.Key != Key.Enter)
        return;
    var focusedElement = Keyboard.FocusedElement as TextBox;
    focusedElement.MoveFocus(new TraversalRequest(FocusNavigationDirection .Next));
}
```

اما همانطور که در عنوان مطلب آورده شده است در این مطلب تصمیم دارم حرکت روی سلول های دیتا گرید توسط کلید Enter را شرح بدهم.

برای این کار نیز یک راه حل ساده وجود دارد و آن شبیه سازی فراخوانی کلید Tab هنگام فشردن کلید Enter است. چون همانطور که می دانید کلید Tab به صورت پیش فرض حرکت روی سلول ها را انجام می دهد. برای انجام آن کافی ست دیتا گرید خود را سفارشی کرده و در متد OnPreviewKeyDown عملیات زیر را انجام دهید:

```
public class CommonDataGrid : DataGrid
{
    protected override void OnPreviewKeyDown(KeyEventArgs e)
    {
        base.OnPreviewKeyDown(e);
        if (e.Key != Key.Enter || Keyboard.PrimaryDevice.ActiveSource == null) return;
        this.CommitEdit();
        var args = new KeyEventArgs
            (Keyboard.PrimaryDevice,
             Keyboard.PrimaryDevice.ActiveSource, 0, Key.Tab) { RoutedEvent = Keyboard.KeyDownEvent };
        InputManager.Current.ProcessInput(args);
    }
}
```

من خودم به شخصه هنوز تا به حال با [WPF](#) کار نکرده‌ام؛ اما قصد دارم از امروز در هر فرصتی که پیش می‌آید به یادگیری این فناوری پر سر و صدا بپردازم. از آنجا که مجموعه‌ی مرتب و به ترتیبی مثل [MVC](#) و [EF](#) در این زمینه در سایت موجود نبود، تصمیم گرفتم که خودم استارت این کار را بزنم که باعث میشه هم خودم بهتر یاد بگیرم و هم این سری برای افراد تازه کار موجود باشه.

آشنایی اولیه

WPF مخفف عبارات [Windows Presentation Foundation](#) است که ویکی پدیا این گونه ترجمه می‌کند: [بنیاد نمایش ویندوزی](#). در برنامه نویسی «ویندوز فرم» ما تمرکز دقیقی بر ساخت رابط کاربری برنامه به خصوص در رزولوشن‌های مختلف نداریم و در بسیاری از اوقات کد با رابط کاربری به شدت وابسته میشد که با ارائه WPF از نسخه‌ی سوم دات نت فریم ورک به بعد، این مشکل حل شد و همچنین عملیات [refactoring](#) را بسیار ساده‌تر کرد. در حالت ویندوز فرم به خاطر وابستگی شدید کد و UI، عملیات بهینه سازی کد اصلا موفق نبود.

WPF از ترکیب عناصر دو بعدی و سه بعدی، اسناد، موارد چند رسانه‌ای و رابط کاربری تشکیل شده‌است و موتور رندر آن بر اساس اطلاعات برداری از کارت گرافیک جهت نمایش ظاهر برنامه کمک می‌گیرد که باعث تهیه برنامه‌ای با رابط کاربری سریعت‌ر، مقیاس پذیرتر و بدون وابستگی به رزولوشن می‌شود.

جداسازی رفتارها و ظاهر برنامه

همانطور که گفتیم بخش رابط کاربری دیگر مستقل از کد برنامه شده است و ظاهر برنامه توسط زبان نشانه گذاری XAML ایجاد می‌شود و بخش کد هم با یکی از زبان‌های موجود در مجموعه دات نت نوشته خواهد شد. نهایتاً این دو بخش توسط رویدادها، فرامین و [DataBinding](#) با یکدیگر متصل می‌شوند. از مزایای جدا بودن این ویژگی:

عدم وابستگی این دو بخش

طراح و کدنویس می‌توانند هر کدام به طور جداگانه کار کنند.

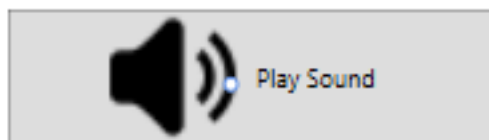
ابزارهای طراحی می‌توانند به طور جداگانه‌ای بر روی اسناد XML کار کنند بدون اینکه نیاز به درگیری با کدنویسی داشته باشند.

یکی از برنامه هایی که به طراحی رابط کاربری با پشتیبانی از XAML می‌پردازد برنامه [Microsoft Expression Blend](#) از مجموعه [Blend](#) است

Rich Composition

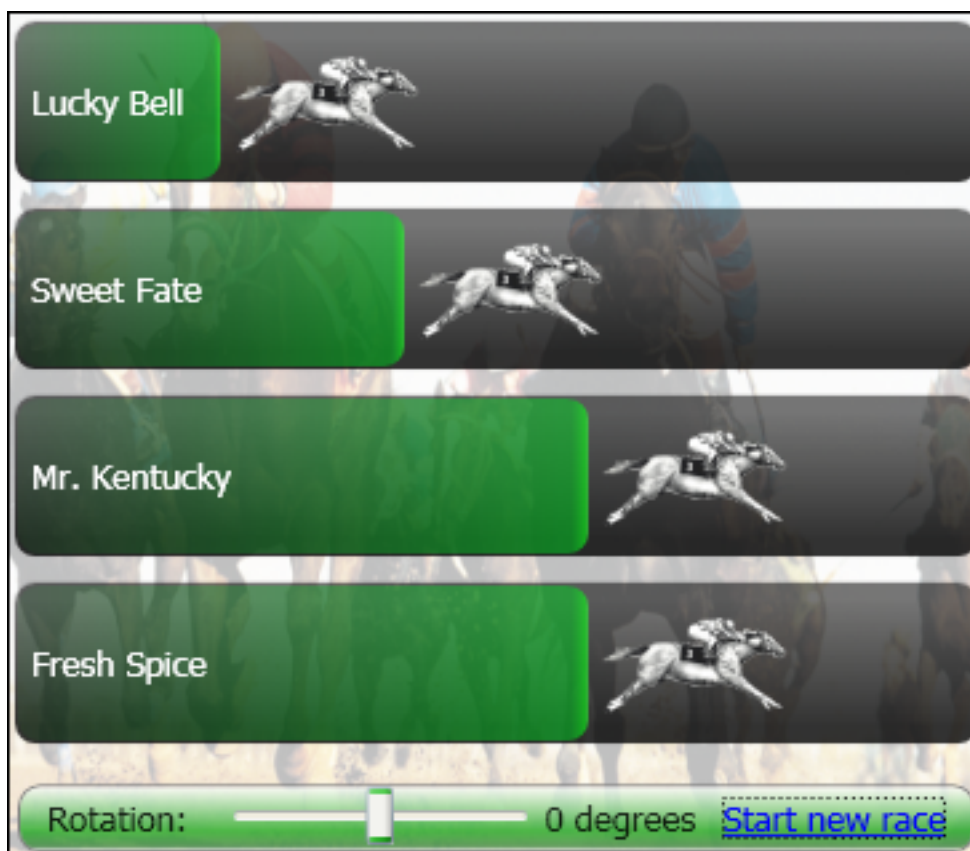
یکی از ویژگی‌های XAML، ساخت اشیاء ترکیبی هست که به راحتی با ترکیب تگ‌ها با یکدیگر و قرار دادن هر شیء داخل یک شیء دیگر می‌توان به یک شیء جدید دست یافت؛ مثل قرار دادن مجموعه ویدیوها در یک لیست. شیء زیر از ترکیب سه شیء تصویر و متن و دکمه ایجاد شده است:

```
<Button Margin="148,123,126,130">
    <StackPanel Orientation="Horizontal">
        <Image Source="speaker.png" Stretch="Uniform"/>
        <TextBlock Text="Play Sound" VerticalAlignment="Center" Margin="10" />
    </StackPanel>
</Button>
```



Highly Customizable

با استفاده از مفهوم Style همانند آنچه که در Html و CSS دارید می‌توانید اشیاء خود را خصوصی سازی کنید و ظاهر آن شیء را به طور کل تغییر دهید.



Resolution Independence

عدم وابستگی به رزولوشن یا وضوح تصویر دارد و به جای واحد پیکسل، از یک واحد منطقی که یک نود و ششم اینچ است، بهره می‌برد. از آنجا که این سیستم بر اساس وکتور ایجاد شده است، مقیاس پذیری آن در تغییر اندازه یا وضوح تصویر به شدت بالا رفته است.



به زودی در قسمت اول این سری کار را با XAML آغاز خواهیم کرد.

نظرات خوانندگان

نویسنده: حسین جهان
تاریخ: ۱۹:۵۴ ۱۳۹۴/۰۲/۰۶

با تشکر. امیدوارم ادامه داشته باشه چون من هم به تازگی کار با WPF رو شروع کردم.

نویسنده: حسن محسنی
تاریخ: ۱۴:۱۰ ۱۳۹۴/۰۲/۰۷

یه سری سیلورلایت چند سال قبل در این سایت منتشر شد:

<https://github.com/DotNETTips/Exports/tree/master/Others/Silverlight>

البته این سری برای سیلورلایت 4 هست و سیلورلایت هم به نظر توسط مایکروسافت کنار گذاشته شده اما اصول XAML اون منهای مباحث خاص سیلورلایت، می‌تونه پایه WPF و حتی مباحث برنامه‌های ویندوزهای 8 و 10 باشه.