Silverlight 4

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

فهرست مطالب

فصل ۳ – معرفی XAML

XAML چیست؟

XAML (/ˈzæməl/ تلفظ می‌شود) یا eXtensible Application Markup Language پایه و اساس طراحی رابط کاربر را در Silverlight تشکیل می‌دهد. XAML زبانی است مبتنی بر XML که توسط مایکروسافت طراحی شده است و تحت مجوز Open Specification Promise در دسترس می‌باشد. اجزای تشکیل دهنده‌ی XAML ، دقیقا با اشیاء CLR متناظر هستند و ویژگی‌های هر کدام از اجزای آن، قابل نگاشت به خواص و رخدادهای اشیاء CLR می‌باشند. به همین دلیل خروجی یک XAML را به صورت کامل می‌توان توسط کدهای زبان‌های مبتنی بر .NET. مانند C# و یا VB.NET نیز ایجاد کرد؛ هر چند کدهای XAML بسیار خواناتر بوده و قابلیت نگهداری بیشتری دارند و همچنین ویرایشگرهای پیشرفته‌ای مانند Microsoft Expression Blend نیز برای تولید آن‌ها مهیا است. XAML منحصر به Silverlight نبوده و در WPF و همچنین Windows workflow foundation نیز کاربرد دارد. قالب XPS نیز بر این اساس طراحی و تطبیق داده شده است.

هنگامیکه یک برنامه‌ی Silverlight را Compile می‌کنیم، فایل‌های XAML آن به فایل‌هایی با پسوند BAML. تبدیل شده (Binary XAML) و در قسمت resources و منابع اسمبلی برنامه درج می‌شوند. به همین جهت هنگام ارائه‌ی برنامه‌های Silverlight دیگر نیازی به ارائه‌ی فایل‌های XAML طراحی شده وجود ندارد.

مدل کد نویسی XAML شبیه به ASP.NET است و در آن امکان تهیه‌ی فایل‌های Code behind و همچنین code inline model نیز فراهم است (code inline model تنها در WPF پشتیبانی می‌شود و هنوز در Silverlight گنجانده نشده است).

مزایای استفاده از XAML

- کدهای XAML عموما کوتاه‌تر از نمونه‌های C# و یا VB.NET هستند و قابلیت خوانایی بالایی دارند
- امکان جدا سازی ظاهر برنامه از کدهای آن
- امکان تهیه‌ی رابط گرافیکی پویا با XAML ساده‌تر است

- امکان جدا سازی نقش طراح رابط گرافیکی و برنامه نویس
- فراهم بودن ویرایشگرهای قوی همانند MS Expression Blend و غیره

همانطور که عنوان شد کدهای XAML را به صورت کدهای زبان‌های .NET نیز می‌توان نمایش داد (هر المان XAML قابل نگاشت به یک کلاس می‌باشد). برای مثال کدهای XAML زیر را در نظر بگیرید که در یک StackPanel ، یک برچسب و یک دکمه قرار گرفته‌اند.

XAML

```
<StackPanel>
    <TextBlock Margin="20">Welcome to the World of XAML</TextBlock>
    <Button Margin="10" HorizontalAlignment="Right">OK</Button>
</StackPanel>
```

معادل این کد در C# به شکل ذیل است که خوانایی و قابلیت نگهداری آن در مقایسه با کدهای XAML فوق بسیار کمتر است:

C#

```
// Create the StackPanel
StackPanel stackPanel = new StackPanel();
this.Content = stackPanel;

// Create the TextBlock
TextBlock textBlock = new TextBlock();
textBlock.Margin = new Thickness(10);
textBlock.Text = "Welcome to the World of XAML";
stackPanel.Children.Add(textBlock);

// Create the Button
Button button = new Button();
button.Margin= new Thickness(20);
button.Content = "OK";
stackPanel.Children.Add(button);
```

برای اینکه بتوان یک چنین نگاشتی را فراهم آورد، تمامی کلاس‌های المان‌های Silverlight و یا WPF دارای سازنده‌های بدون پارامتر (parameter less constructors) هستند. زیرا زمانیکه المان <Button /> در کدهای XAML رابط کاربر قرار می‌گیرد، بلافاصله وهله‌ای از کلاس Button نیز تشکیل خواهد شد و برای اینکار نیاز است تا کلاس Button دارای سازنده‌ای بدون پارامتر باشد.

از مقایسه‌ی دو کد C# و XAML مشخص می‌گردد که مقادیر ویژگی‌های المان‌های ذکر شده در XAML معادل مقادیر خاصیت‌های اشیاء ، در کدهای C# هستند. بدیهی است این ویژگی‌ها جهت معرفی رخدادها نیز می‌توانند کاربرد داشته باشند.

ابزارهای توسعه‌ی XAML

- **ویرایشگر Visual Studio**

با پشتیبانی از intellisense و ویرایشگر code behind . در نسخه‌ی VS.NET 2010 بهبودهای قابل توجهی داشته است و بسیاری از قابلیت‌های MS Expression Blend را به ارث برده است.

- **Microsoft Expression Blend**

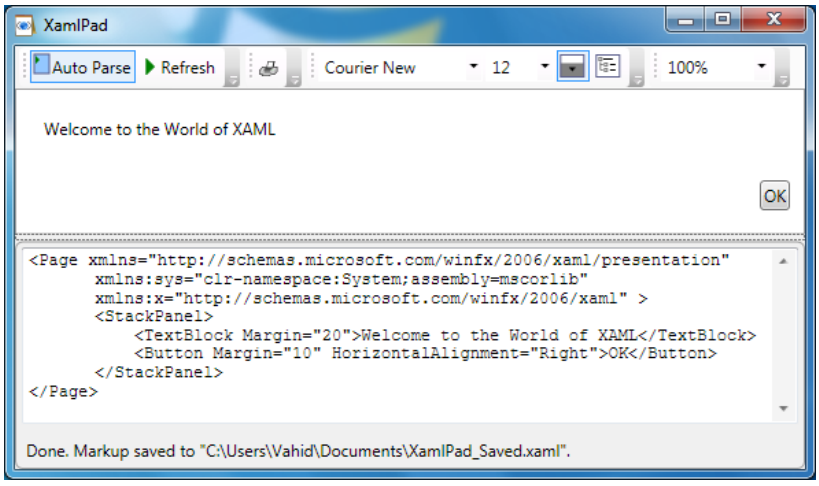
یکی از بهترین‌ها و تخصصی‌ترین ویرایشگر XAML است که نگارش تجاری آن‌را از آدرس زیر می‌توان تهیه نمود:

http://www.microsoft.com/expression/products/Blend_Overview.aspx
http://tinyurl.com/29bcpoh

- **XamlPad**

اگر نگارش ۶ و یا بالاتر SDK ویندوز بر روی کامپیوتر شما نصب باشد، این برنامه‌ی ساده را که جهت آزمایش سریع کدهای XAML مناسب است، در آدرس ذیل می‌توانید پیدا نمائید (شکل ۱):

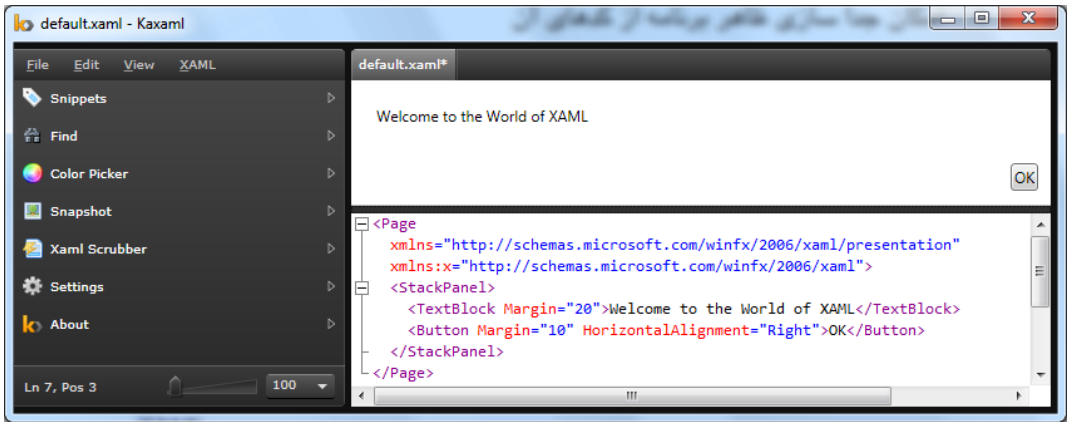
C:\Program Files\Microsoft SDKs\Windows\v6.0A\Bin



شکل ۱– نمایی از برنامه‌ی XamlPad

- **Kaxaml**

Kaxaml نیز یک ویرایشگر کم حجم و رایگان XAML محسوب می‌شود که از آدرس بعد قابل دریافت است (شکل ۲):



شکل ۲– نمایی از برنامه‌ی Kaxaml .

اصول کاری XAML

یک سری بایدها و نبایدها حین کار با XAML وجود دارند که در ادامه آن‌ها را مرور خواهیم نمود:

- فایل‌های XAML از تمامی اصول کاری فایل‌های XML نیز پیروی می‌کنند؛ خصوصا خوش فرم بودن آن‌ها و تطابق تگ‌ها با یکدیگر و الزامی بودن بسته شدن تمامی تگ‌ها.
- همانند فایل‌های XML ، فایل‌های XAML نیز باید به یک المان ریشه محدود باشند که در Silverlight عموما یک User control این المان ریشه را تشکیل می‌دهد.
- جهت مشخص سازی مقادیر ویژگی‌ها (attributes) باید از "" و یا '' استفاده شود.
- نباید از حروف غیرمجاز در XML بدون ملاحظات لازم در اینجا نیز استفاده گردد. برای مثال استفاده از حروف & و یا < هنگام ارائه‌ی مقادیر ویژگی‌های المان‌ها مجاز نیست و باید اصطلاحا escape شوند (به روش زیر):

حرف غیر مجاز	تبدیل شود به	
<	→	<
>	→	>
&	→	&
"	→	"
'	→	'

لازم به ذکر است که این تبدیلات تنها هنگام مقدار دهی ویژگی‌ها در XAML باید صورت گیرد. اگر مقادیر مورد نظر از طریق کدهای برنامه تنظیم می‌شوند، نیازی به انجام اینکار نیست. به علاوه استفاده از {} نیز به markup extension تفسیر خواهد شد (برای مثال جهت تعاریف عملیات Binding). در این حالت برای نمایش محتوای بین {----} باید یک {} را در ابتدای عبارت قرار داد:

<textblock text="{}this is {my} name"/>

- تمامی تگ‌های XAML همانند تگ‌های XML ، حساس به کوچکی و بزرگی حروف می‌باشند.

ویژگی‌های (attributes) عناصر XAML

حداقل چهار نوع ویژگی را می‌توان جهت عناصر XAML تعریف نمود که در ادامه به معرفی آنها خواهیم پرداخت. برای این منظور یک برنامه‌ی Silverlight جدید را در VS.NET آغاز نمائید (در اینجا نیازی به ایجاد پروژه‌ی میزبان از نوع ASP.NET website نیست). سپس کدهای MainPage.xaml آن‌را مطابق کدهای XAML ایی که در ادامه معرفی می‌گردند تغییر دهید.

در این مثال ۴ نوع ویژگی مشاهده می‌شود:

- Property attribute : متناظر است با خواص اشیاء. برای مثال در کدهای XAML این قسمت، ویژگی Fill ، یکی از خاصیت‌های شیء Rectangle است.
- Event attribute : متناظر است با رخدادهای اشیاء. برای نمونه رویداد MouseEnter با نام متد مدیریت کننده‌ی آن رخداد در Code behind صفحه مقدار دهی شده است.
- Directive attribute : معادلی در کدهای برنامه ندارد. برای مثال x:Name ذکر شده در کدهای XAML ذیل.
- Attached attribute : توسط ویژگی‌های الحاقی می‌توان خاصیت‌های یک کلاس را در کلاسی دیگر مقدار دهی نمود. برای مثال در اینجا خاصیت Left کلاس Canvas مقدار دهی شده است.

MainPage.xaml

```
<UserControl x:Class="SilverlightApplication3.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc=
        "http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d"
    d:DesignHeight="300" d:DesignWidth="400">
    <Grid x:Name="LayoutRoot" Background="White">
        <Rectangle Fill='Orchid'
                MouseEnter='Rectangle_MouseEnter'
                x:Name='smallRect'
                Canvas.Left='20'  />
    </Grid>
</UserControl>
```

فایل Code behind متناظر با این فایل XAML به شکل زیر می‌تواند باشد :

MainPage.xaml.cs

```
using System.Windows.Input;
namespace SilverlightApplication3
{
    public partial class MainPage
    {
        public MainPage()
        {
            InitializeComponent();
        }
        private void Rectangle_MouseEnter(object sender,
             MouseEventArgs e)
        {  }
    }
}
```

اگر علاقمند باشید که معادل C# کدهای XAML مثال این قسمت را ملاحظه نمائید، این کدها به شرح ذیل می‌توانند باشند:

C#

```
using System.Windows.Controls;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Shapes;

namespace SilverlightApplication3
{
    public partial class MainPage
    {
        public MainPage()
        {
            InitializeComponent();

            // equivalent code for four attribute types
            var rect = new Rectangle
                        {
                            Width = 40,
                            Fill = new SolidColorBrush(Colors.Orange)
                        };

            // read value from property
            double w = rect.Width;
```

```
            // assign delegate to event
            rect.MouseLeave += Rectangle_MouseEnter;

            // directive attributes are only needed in XAML

            // assign value to attached property
            Canvas.SetLeft(rect, 50);
            // read value from attached property
            double l = Canvas.GetLeft(rect);

            LayoutRoot.Children.Add(rect);
        }

      private void Rectangle_MouseEnter(object sender, MouseEventArgs e)
      { }
    }
}
```

فضاهای نام در XAML

برای استفاده از کلاس‌ها و اشیاء دات نت در XAML نیاز است تا فضاهای نام آن‌ها را به XAML مورد استفاده معرفی نمود. در اینجا فضاهای نام، به کمک ویژگی XMLNS معرفی می‌گردند. اگر به مثال‌های قبل دقت کرده باشید یک سری فضای نام پیش فرض در ابتدای فایل‌های XAML موجود است. به این صورت XAML Parser می‌تواند تشخیص دهد که اطلاعات اشیاء مختلف را باید از کجا دریافت نموده و سپس نسبت به ساخت وهله‌ای از آن‌ها اقدام نماید. برای مثال فضای نام زیر:

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

حاوی اطلاعات فضاهای نام System.Windows، System.Windows.Controls، System.Windows.Data، System.Media.Animation و بسیاری موارد دیگر می‌باشد و یا فضای نام زیر که با یک نام مستعار x (Alias) معرفی شده است :

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

جهت تعاریف ویژگی‌هایی مانند x:Null، x:Name، x:Key، x:Class، x:Uid، x:FieldModifier تعریف شده در فضای نام System.Windows.Markup کاربرد دارد.

حتی می‌توان اشیایی از نوع اسمبلی‌های سفارشی تهیه شده توسط خود یا سایر برنامه‌نویس‌ها را نیز به این صورت معرفی نمود. برای مثال فرض کنید که شیءایی را به نام StarShape، برنامه نویسی کرده‌اید و این شیء در فضای نام MyNameSpace قرار دارد. برای استفاده از آن در یک فایل XAML باید فضای نام آن‌را به شکل ذیل معرفی نمود (این نوع کلاس‌ها تنها پس از Compile ، در XAML قابل استفاده خواهند بود):

xmlns:**ctrl**='clr-namespace:MyNameSpace;assembly=MyNameSpace'

و سپس نحوه‌ی استفاده از آن کنترل جدید به صورت زیر می‌باشد:

```
<ctrl:StarShape />
```

اگر از افزونه‌ی ReSharper که از آدرس زیر قابل دریافت است استفاده نمائید، می‌توان کار تشخیص خودکار و الحاق اینگونه فضاهای نام را به سادگی توسط آن انجام داد:

http://www.jetbrains.com/resharper/

لازم به ذکر است که در VS.Net 2010 ، دو فضای نام جدید ذیل قابل ملاحظه هستند:

```
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
mc:Ignorable="d"
```

توسط این دو فضای نام، اطلاعاتی که XAML Parser می‌تواند از آن‌ها صرفنظر نماید معرفی می‌شوند. برای مثال طول و عرض صفحه‌ی طراحی را می‌توان به این وسیله معرفی نمود:

```
d:DesignHeight="300" d:DesignWidth="400"
```

علت این امر هم آن است که مشخص سازی دقیق طول و عرض یک User control عموما توصیه نمی‌شود زیرا به این صورت دیگر فرم جاری برنامه، تمامی طول و عرض مرورگر را پوشش نخواهد داد. اما نیاز است تا یک حداقلی از طول و عرض را جهت طراحی صفحه‌ی Use control بتوان معرفی نمود. برای این منظور DesignHeight و DesignWidth فوق، تعریف شده‌اند که تنها در حین طراحی در VS.NET 2010 و یا MS Expression blend معتبر هستند اما در زمان اجرا ندید گرفته می‌شود.

تبدیل کننده‌های انواع در Silverlight

در کدهای XAML مثال قبل، مقدار Orchid به ویژگی Fill نسبت داده شد. با توجه به اینکه ویژگی‌های تعریف شده‌ی یک عنصر در XAML دقیقا با خواص شیء مورد نظر تطابق دارند و مقادیر معرفی شده در XAML همگی از نوع رشته‌ای می‌باشند، تبدیل مقدار رشته ای Orchid به رنگ متناظر شیء Brush آن به صورت خودکار توسط کلاس BrushConverter تعریف شده در فضای نام System.Windows.Media انجام خواهد شد و شاهد بروز خطایی در برنامه نخواهیم بود. نمونه‌ای دیگر از این دست، تبدیل خودکار مقدار رشته‌ای نسبت داده شده به Canvas.Left تعریف شده در XAML به معادل عددی آن است.

امکان تعریف تبدیل کننده‌های انواع سفارشی نیز وجود دارد. برای این منظور لطفا به مثال ساده‌ی بعد دقت بفرمائید.

یک برنامه‌ی جدید Silverlight را در VS.NET آغاز نمائید (در اینجا نیازی به ایجاد پروژه‌ی میزبان از نوع ASP.NET website نیست). در ادامه یک User control جدید را به برنامه اضافه کنید (منوی پروژه، گزینه‌ی Add new item و سپس انتخاب Silverlight user control). نام آن‌را WeatherControl وارد نمائید. قصد

داریم توسط این کنترل، متن ساده‌ای را در صفحه‌ی اصلی برنامه نمایش دهیم؛ همچنین رنگ این کنترل سفارشی را نیز دریافت کرده و پردازش نمائیم. کدهای XAML این User control به شرح زیر هستند:

WeatherControl.xaml

```
<UserControl x:Class="SilverlightApplication4.WeatherControl"
    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">
    <StackPanel  x:Name="LayoutRoot" Background="White">
        <TextBlock FontSize='16'
                              Text='Weather Control' />
    </StackPanel>
</UserControl>
```

یک برچسب ساده قرار گرفته درون یک StackPanel توسط این User control نمایش داده می‌شود. اکنون قصد داریم خاصیت WeatherBackground را به کدهای آن اضافه نمائیم تا بتوان رنگ پس زمینه‌ی StackPanel را تنظیم نمود. اگر رنگ‌های انتساب داده شده به این خاصیت، جزو رنگ‌های استاندارد .NET باشند، عملیات تبدیل خودکار آن به Brush معادل انجام خواهد شد؛ اما در اینجا قصد داریم برای مثال رنگی به نام Sunny را در XAML وارد نمائیم. چون تبدیل کننده‌ی خودکاری برای این نام ویژه وجود ندارد، امکان توسعه‌ی تبدیل کننده‌های Silverlight مهیا است. برای این منظور کلاس جدید WeatherTypeConverter را به برنامه اضافه کنید. کدهای این کلاس به شرح بعد هستند:

WeatherTypeConverter.cs

```
using System;
using System.ComponentModel;
using System.Globalization;
using System.Windows.Media;

namespace SilverlightApplication4
{
    public class WeatherTypeConverter : TypeConverter
    {
        public override bool CanConvertFrom(
          ITypeDescriptorContext context,
                                    Type sourceType)
        {
            if (sourceType == typeof(string))
            {
```

```
                return true;
            }
            return base.CanConvertFrom(context, sourceType);
        }

        public override object ConvertFrom(
             ITypeDescriptorContext context,
                              CultureInfo culture, object value)
        {
            var text = value as string;

            switch (text)
            {
                case "Sunny":
                    return new SolidColorBrush(Colors.Yellow);
                case "Foggy":
                    return new SolidColorBrush(Colors.Gray);
                case "Rainy":
                    return new SolidColorBrush(Colors.Blue);
                default:
                    return new SolidColorBrush(Colors.White);
            }
        }
    }
}
```

نحوه‌ی ایجاد یک کلاس تبدیل کننده‌ی نوعی، با کمک پیاده سازی کلاس استاندارد TypeConverter می‌باشد. توسط متد CanConvertFrom مشخص می‌شود که آیا مقدار انتساب داده شده از نوعی است که می‌توان آن را تبدیل نمود یا خیر. سپس کار تبدیل نهایی در متد ConvertFrom انجام می‌شود.

در ادامه نحوه‌ی بکارگیری این تبدیل کننده‌ی سفارشی در کلاس کدهای WeatherControl.xaml به صورت زیر خواهد بود که در آن خاصیت عمومی WeatherBackground توسط ویژگی TypeConverter از نوع کلاس WeatherTypeConverter مزین شده است:

WeatherControl.xaml.cs

```
using System.ComponentModel;
using System.Windows.Media;

namespace SilverlightApplication4
{
    public partial class WeatherControl
    {
        public WeatherControl()
        {
            InitializeComponent();
        }
```

```
        [TypeConverter(typeof(WeatherTypeConverter))]
        public Brush WeatherBackground
        {
            get
            {
                return LayoutRoot.Background;
            }
            set
            {
                LayoutRoot.Background = value;
            }
        }
    }
}
```

اکنون می‌توان برای معرفی این User control جدید در صفحه‌ی اصلی برنامه، مطابق کدهای XAML ذیل عمل کرد:

MainPage.xaml

```
<UserControl x:Class="SilverlightApplication4.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:SilverlightApplication4"
    mc:Ignorable="d" d:DesignHeight="300" d:DesignWidth="400">
    <Grid x:Name="LayoutRoot" Background="White">
      <local:WeatherControl WeatherBackground='Sunny' />
    </Grid>
</UserControl>
```

در اینجا ابتدا فضای نام xmlns:local="clr-namespace:SilverlightApplication4" جهت معرفی محل قرارگیری کدهای WeatherControl به صفحه اضافه شده است و سپس این User control درون Grid صفحه قرار گرفته است. ویژگی WeatherBackground کنترل با کلمه‌ی Sunny مقدار دهی گشته است و کار تبدیل آن به رنگی معادل، توسط کلاس WeatherTypeConverter انجام خواهد شد.

تعریف مقادیری از خواص که خود یک شیء هستند

نحوه‌ی تعریف ویژگی‌ها و مقدار دهی آن‌ها در XAML بسیار ساده و کارآمد هستند اما گاهی از اوقات نیاز است تا یک شیء را به یک ویژگی نسبت داد. برای مثال قصد داریم یک تصویر را در کنار برچسب یک دکمه نمایش دهیم. تصویر، خود یک شیء است و هر چند انتساب آن به صورت یک رشته به یک ویژگی با بکارگیری کدهای .NET معادل آن میسر است اما روشی مرسوم و پسندیده نیست و احتمال خطا در آن زیاد است. خوشبختانه XAML برای مواجه شدن با این نوع مسایل، از شیوه‌ای به نام property-element syntax جهت معرفی زیر شیء های یک شیء، استفاده می‌کند (به شکل Parent.PropertyName). برای مثال:

XAML

```
<Button>
  <Button.Content>
     <Image Source="Images/OK.png" Width="50" Height="50" />
  </Button.Content>
</Button>
```

در اینجا یک تصویر به عنوان محتوای دکمه‌ی تعریف شده در XAML معرفی گشته است. به این صورت امکان ایجاد کنترل‌های ترکیبی در Silverlight به سادگی میسر می‌شود.

امکان تعریف ویژگی‌های متداول اشیاء نیز به این روش (property-element syntax) وجود دارد. برای نمونه دو دکمه‌ی معرفی شده‌ی بعد، معادل هستند:

XAML

```
<Button Margin="5">Click me!</Button>
```

XAML

```
<Button>
  <Button.Margin>5</Button.Margin>
  Click me!
</Button>
```

تعریف مقادیر ویژگی‌ها به شکلی پویا

مفاهیمی در XAML وجود دارند به نام Markup extensions که در حقیقت امکان تعریف مقادیر ویژگی‌ها را به شکلی پویا میسر می‌سازند. به این صورت مقادیر مورد استفاده در XAML به صورت پویا در حین اجرای برنامه دریافت خواهند شد. Markup extensions مهیا در Silverlight به صورت خلاصه به شرح زیر هستند (مبحث Binding یکی از مهمترین مباحث Silverlight و WPF بوده و در طی فصول بعدی بیشتر بحث خواهد شد):

- Binding
  دریافت اطلاعات از یک منبع داده و انتساب آن به یک خاصیت
- StaticResource
  استفاده از منابعی که به صورت منبع در قسمت ResourceDictionary تعریف می‌شوند
- TemplateBinding
  دریافت مقادیر از یک control template و انتساب آن به یک خاصیت
- x:Null
  انتساب null به یک خاصیت

برخلاف WPF ، Markup extensions در Silverlight قابل توسعه نیستند و امکان تهیه‌ی نمونه‌های سفارشی از این دست وجود ندارد.

برای توضیح مقدماتی این مباحث، لطفا به مثال بعدی دقت بفرمائید. ابتدا یک پروژه‌ی جدید Silverlight را آغاز نموده و سپس کدهای XAML صفحه‌ی اصلی آن‌را به شکل زیر تغییر دهید:

MainPage.xaml

```
<UserControl x:Class="SilverlightApplication5.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">

    <UserControl.Resources>
        <LinearGradientBrush x:Key='seaBrush'>
            <LinearGradientBrush.GradientStops>
                <GradientStop Offset="0" Color="Yellow" />
                <GradientStop Offset="0.5" Color="Orange" />
                <GradientStop Offset="0.8" Color="LightCoral" />
            </LinearGradientBrush.GradientStops>
        </LinearGradientBrush>
```

```
    </UserControl.Resources>

  <StackPanel x:Name="LayoutRoot">
      <!-- Binding MarkupExtension-->
      <TextBox Text='Hello'
             x:Name='textHello'
             Background='Goldenrod' />
     <TextBox Text='Goodbye'
  Background='{Binding ElementName=textHello, Path=Background}' />

      <!-- RelativeSource MarkupExtension-->
      <TextBox Text='Red'
 Background='{Binding Text,RelativeSource={RelativeSource Self }}' />

      <!-- StaticResource MarkupExtension -->
      <TextBox Text='StaticResource'
             Background='{StaticResource seaBrush}' />

      <!-- Null MarkupExtension -->
      <TextBox Text='Null MarkupExtension' Background='{x:Null}' />
  </StackPanel>
</UserControl>
```

توضیحات:

در قسمت UserControl.Resources ، منابع مورد استفاده User control تعریف می‌شوند برای مثال در اینجا یک LinearGradientBrush تعریف شده است. برای استفاده از آن حتما نیاز است تا ویژگی x:Key آن مقدار دهی گردد. در ادامه نحوه‌ی استفاده از این منبع را ملاحظه می‌فرمائید:

```
      <!-- StaticResource MarkupExtension -->
      <TextBox Text='StaticResource'
             Background='{StaticResource seaBrush}' />
```

برای معرفی Binding MarkupExtension ، دو جعبه‌ی متنی بر روی صفحه قرار گرفته‌اند. خاصیت Background جعبه متنی دوم به خاصیت Background جعبه متنی اول مقید (Bind) شده است:

```
      <!-- Binding MarkupExtension-->
      <TextBox Text='Hello'
             x:Name='textHello'
             Background='Goldenrod' />
     <TextBox Text='Goodbye'
  Background='{Binding ElementName=textHello, Path=Background}' />
```

در اینجا ElementName نام کنترلی را مشخص می‌کند که قرار است اطلاعات از آن دریافت شود و Path نیز خاصیتی از آن کنترل است که از اطلاعات آن جهت انقیاد، استفاده خواهیم کرد.

در ادامه یک کنترل جعبه‌ی متنی دیگر بر روی فرم قرار گرفته است که به خودش مقید شده است (توسط RelativeSource Self). این جعبه‌ی متنی، مقدار Text دریافت شده خود را به صورتی پویا به ویژگی Background خود انتساب می‌دهد:

```
        <!-- RelativeSource MarkupExtension-->
        <TextBox Text='Red'
  Background='{Binding Text,RelativeSource={RelativeSource Self }}' />
```

نمونه‌ای نیز در مورد نحوه‌ی انتساب null به یک کنترل، ذکر گردیده است:

```
        <!-- Null MarkupExtension -->
        <TextBox Text='Null MarkupExtension' Background='{x:Null}' />
```

تعریف ویژگی‌های سفارشی عناصر XAML

در مثال معرفی TypeConverter که پیشتر ارائه گردید، خاصیت WeatherBackground به صورتی متداول و همانند سایر خواص و کدهای مجموعه‌ی .NET Framework معرفی گردید. روش صحیح معرفی اینگونه خواص که در برنامه قرار است به شکل یک ویژگی XAML استفاده گردند باید به کمک مفهومی به نام Dependency property انجام شود. به این صورت امکان استفاده از بسیاری از ویژگی‌های پیشرفته‌ی Silverlight و WPF همانند پویانمایی (Animation)، Styles و Templates امکان پذیر خواهند شد و می‌توان از مزایای سیستم Binding بهره‌مند گردید. همچنین اینگونه خواص توسط موتور Silverlight از لحاظ مصرف حافظه نیز بهینه سازی می‌شوند. در حقیقت Dependency property سرویسی است جهت بسط توانایی‌ها و عملکرد خواص ساده و متداول مجموعه‌ی .NET.

کلیه ویژگی‌های عناصر استاندارد Silverlight نیز به کمک همین مفهوم پیاده سازی شده‌اند و در اصل یک Dependency property می‌باشند. برای توضیحات بیشتر لطفا به مثال ذیل دقت بفرمائید:

MainPage.xaml

```
<UserControl x:Class="SilverlightApplication6.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc=
      "http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d"
    d:DesignHeight="300" d:DesignWidth="400">
    <Grid x:Name="LayoutRoot" Background="White">
        <TextBlock Name="txtDp" Text="DP System ..." />
    </Grid>
</UserControl>
```

در اینجا یک برچسب ساده بر روی صفحه قرار گرفته است و در ادامه قصد داریم تا اندازه‌ی قلم این برچسب را در کدهای صفحه تغییر دهیم :

MainPage.xaml.cs

```
namespace SilverlightApplication6
{
    public partial class MainPage
    {
        public MainPage()
        {
            InitializeComponent();

            txtDp.FontSize = 20; //valid
            var size = txtDp.FontSize;//valid

            //using DP system
            txtDp.SetValue(FontSizeProperty,20);
            size = (double) txtDp.GetValue(FontSizeProperty);
        }
    }
}
```

همانطور که ملاحظه می‌کنید، به دو روش می‌توان اندازه‌ی قلم برچسب را تغییر داد:

- به کمک شیوه‌ی متداول مقدار دهی به خواص اشیاء
- با استفاده از سیستم Dependency properties و استفاده از متدهای SetValue و GetValue جهت تنظیم و یا دریافت مقادیر یک خاصیت. FontSizeProperty یکی از Dependency properties استاندارد Silverlight است.

نکته‌ی دیگری که در سیستم Dependency properties باید درنظر داشت، مباحث ارث بری خواص است. برای مثال در کد ذیل، اندازه‌ی قلم User control تنظیم شده است که بر روی اندازه‌ی قلم TextBlock تعریف شده در آن نیز تاثیر گذار است:

MainPage.xaml

```
<UserControl x:Class="SilverlightApplication6.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"
    FontSize="20">
    <Grid x:Name="LayoutRoot" Background="White">
        <TextBlock Text="DP System ..." />
    </Grid>
</UserControl>
```

تقدم و تاخر این ارث بری و تاثیر گذاری به ترتیب ذیل می‌باشد (از بیشتر به کمتر):

۱. پویانمایی (Animation)
۲. مقادیر تنظیم شده در کدهای برنامه
۳. Styles
۴. ارث بری در سیستم Dependency properties
۵. مقادیر پیش فرض

در ادامه قصد داریم نسبت به تعریف یک Dependency property سفارشی اقدام نمائیم. این مورد از این جهت حائز اهمیت است که اگر خواص عمومی یک User control را به صورت خواص معمولی NET. معرفی نمائیم، از قابلیت‌های Styles ، پویانمایی و بسیاری دیگر از قابلیت‌های پیشرفته‌ی Silverlight محروم خواهیم شد.

یک پروژه‌ی Silverlight جدید را آغاز نمائید. سپس یک User control جدید را به نام Shape5 با استفاده از منوی پروژه، گزینه‌ی Add new item و انتخاب Silverlight User control ایجاد نمائید. کدهای XAML این User control به شرح زیر هستند:

Shape5.xaml

```
<UserControl x:Class="SilverlightApplication7.Shape5"
    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="200" d:DesignWidth="200">
    <Grid x:Name="LayoutRoot" Background="White">
        <Path x:Name="star5"
                    Stroke="Orange"
       Data="F1 M 145.637,174.227L 127.619,110.39L 180.809,70.7577L
             114.528,68.1664L 93.2725,5.33333L 70.3262,67.569L 4,
             68.3681L 56.0988,109.423L 36.3629,172.75L 91.508,
            135.888L 145.637,174.227 Z " />
    </Grid>
</UserControl>
```

توسط این User control یک ستاره رسم می‌شود. اکنون قصد داریم یک خاصیت FillColor را برای این شکل ترسیم شده، تهیه نمائیم به نحوی که از طریق صفحه‌ی اصلی دربرگیرنده‌ی آن قابل تنظیم باشد:

Shape5.xaml.cs

```
using System.Windows;
using System.Windows.Media;

namespace SilverlightApplication7
{
    public partial class Shape5
    {
        public Shape5()
        {
            InitializeComponent();
        }

        public Brush FillColor
        {
            get { return (Brush)GetValue(FillColorProperty); }
            set { SetValue(FillColorProperty, value); }
        }

        public static readonly DependencyProperty FillColorProperty =
            DependencyProperty.Register("FillColor",
              typeof(Brush), typeof(Shape5),
               new PropertyMetadata(new SolidColorBrush(Colors.Brown),
               colorChangedCallBack));

        private static void colorChangedCallBack(DependencyObject d,
                                DependencyPropertyChangedEventArgs e)
        {
            var shape5 = d as Shape5;
            if (shape5 != null)
                shape5.onFillColorPropertyChanged(e);
        }

        private void onFillColorPropertyChanged(
                    DependencyPropertyChangedEventArgs e)
        {
            var color = e.NewValue as SolidColorBrush;
            if (color != null)
                star5.Fill = new SolidColorBrush(color.Color);
        }
    }
}
```

این مثال نحوه‌ی تعریف یک Dependency property (که همیشه به همین شکل و قالب استاندارد می‌باشد) را نمایش می‌دهد. همچنین نحوه‌ی عکس العمل نشان دادن به تغییر رنگ و تغییر مقدار آن‌را نیز توسط متد callback تهیه شده می‌توان ملاحظه کرد.

یک Dependency property جدید باید در مجموعه‌ی موجود آن‌ها ثبت شود. نحوه‌ی انجام این عملیات به صورت زیر است:

```
public static readonly DependencyProperty FillColorProperty =
    DependencyProperty.Register("FillColor",
      typeof(Brush), typeof(Shape5),
       new PropertyMetadata(new SolidColorBrush(Colors.Brown),
       colorChangedCallBack));
```

به این صورت خاصیت FillColor ، از نوع Brush در کلاس پایه Shape5 به همراه یک MetaData (مقدار اولیه‌ای که در خواص این user control در صفحه‌ی properties شیء ظاهر خواهد شد) و یک متد callback تعریف می‌گردد. این متد callback به ازای هر بار تغییر در مقدار FillColor ، فراخوانی خواهد شد.

نکته‌ای را که باید در اینجا به آن دقت داشت، static بودن این خاصیت و همچنین متد callback آن است.

نحوه‌ی استفاده از این User control در صفحه‌ی اصلی برنامه، مطابق کدهای XAML زیر است (ابتدا فضای نام مربوطه اضافه شده و سپس در یک گرید قرار گرفته است):

MainPage.xaml

```
<UserControl x:Class="SilverlightApplication7.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:SilverlightApplication7"
    mc:Ignorable="d"  d:DesignHeight="300" d:DesignWidth="400">
    <Grid x:Name="LayoutRoot" Background="White">
        <local:Shape5  FillColor="Yellow" />
    </Grid>
</UserControl>
```

همانطور که ملاحظه می‌کنید تهیه‌ی Dependency properties عملیاتی است تکراری و وقت گیر. به همین جهت یک سری Code snippets مخصوص اینکار برای VS.NET تهیه شده است که از پروژه‌ی سورس باز ذیل قابل دریافت است:

http://silverlightcontrib.codeplex.com

پس از دریافت آخرین نگارش آن، به پوشه‌ی Helpers\VS Snippets این مجموعه مراجعه نموده و دوبار بر روی فایل Silverlight Snippets.vsi کلیک نمائید تا عملیات نصب آن آغاز شود (شکل ۳). در صفحه‌ی دوم نصب آن پوشه‌های My Code Snippets را انتخاب نموده و عملیات نصب را به پایان برسانید.

اکنون برای مثال اگر در کدهای یک صفحه تایپ کنید sldp و سپس دکمه‌ی Tab را بفشارید، بلافاصله بدنه‌ی خام یک Dependency property برای شما ایجاد خواهد شد.

این مجموعه شامل موارد ذیل است:

- sldp

جهت ایجاد یک dependency property به همراه getter/setter .

- sldpc

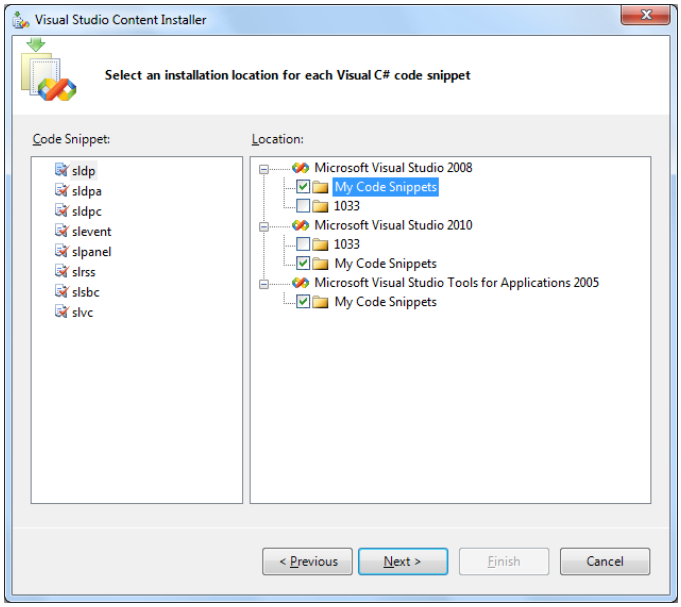
جهت ایجاد یک dependency property به همراه getter/setter و callback مرتبط

- sldpa

برای ایجاد یک dependency property به همراه متدهای Get و Set و callback مرتبط

جهت مشاهده‌ی سایر موارد پشتیبانی شده لطفا به آدرس ذیل مراجعه کنید:

http://bit.ly/bpW0Ao



شکل ۳– نصب Code snippets مخصوص VS.NET

معرفی مقدماتی Attached properties

در طی فصول آتی زمانیکه مباحث طرح بندی و layout مطرح می‌شوند با خواص الحاقی یا Attached properties سروکار خواهیم داشت. به همین جهت در فصل آشنایی با XAML لازم است مروری مقدماتی بر این مورد انجام شود.

گاهی از اوقات در حین طراحی رابط کاربر برنامه نیاز خواهد بود تا بین عناصر مختلف آن ارتباط برقرار شود. برای مثال یک دکمه را در ستون دوم و سطر اول یک Grid قرار دهیم. در سیستم قدیمی WinForms ، این مشکل با تعیین مکان مطلق یک کنترل بر روی فرم برطرف می‌گردد که سبب درهم تنیدگی کنترل‌ها و فرم جاری می‌گردد. در سیستم‌های جدید طراحی رابط کاربر WPF و Silverlight ، از Attached properties برای حل این مشکل کمک خواهیم گرفت. یک Attached property نیز در حقیقت یک نوع Dependency property است که به صورت عمومی در اختیار سایر عناصر یک صفحه‌ی XAML خواهد بود.

این نوع خواص توسط متد DependencyProperty.RegisterAttached به سیستم Dependency property جاری برنامه معرفی می‌شوند و از این لحظه به بعد در انحصار شیء معرفی کننده‌ی آن نبوده و توسط سایر اشیاء نیز قابل استفاده و مقدار دهی می‌باشند. نحوه‌ی دسترسی به این خواص به شکل Type.AttachedProperty می‌باشد. مهم‌ترین استفاده‌ی Attached properties در Silverlight تعیین مکان اشیاء فرزند یک شیء می‌باشند. برای مثال تعیین موقعیت یک برچسب در سلولی خاص از گرید؛ یا تعیین موقعیت یک شیء در Canvas و غیره. برای نمونه به دو شیء زیر دقت بفرمائید:

XAML

```
<TextBlock Canvas.Top='75' />
<CheckBox Canvas.Left='5'
          AutomationProperties.AcceleratorKey='A'
          Grid.Row='2'
          InputMethod.IsInputMethodEnabled='True'
          ScrollViewer.VerticalScrollBarVisibility='Hidden' />
```

تمامی خواصی که در این دو TextBlock و CheckBox تنظیم شده‌اند از نوع Attached properties می‌باشند زیرا به صورت مستقیم جزو خواص اصلی این اشیاء نبوده و از طریق سیستم Dependency properties ، به مجموعه به شکلی الحاقی معرفی شده‌اند. بنابراین سایر اشیاء می‌توانند از طریق قالب Type.AttachedProperty به آن‌ها دسترسی داشته باشند.