

Silverlight 4

فهرست مطالب

فصل ۱۲ – اعمال قالب‌های متفاوت به برنامه‌های Silverlight	۲۴۰
نحوه‌ی تعریف و استفاده از منابع (Resources)	۲۴۰
مدیریت مرکزی تنظیمات اشیاء با کمک Styles	۲۴۲
تعریف Styles به صورت آبشاری	۲۴۳
استفاده از فایل‌های Style	۲۴۴
تعریف ظاهری دگرگون شده برای کنترل‌های بصری برنامه	۲۴۸
معرفی کلاس VisualStateManager	۲۵۲
استفاده از قالب‌های آماده در Silverlight	۲۵۶

چاپ عمومی غیر رایگان این مطالب بدون مجوز کتبی از طرف نویسنده به هر نحوی غیرمجاز است.
انتشار این مطالب بر روی اینترنت و یا استفاده از آن به صورت مستقیم و یا غیر مستقیم در نشریات الکترونیکی با ذکر مأخذ بلا مانع است.

فصل ۱۲ – اعمال قالب‌های متفاوت به برنامه‌های Silverlight

نحوه‌ی تعریف و استفاده از منابع (Resources)

برنامه نویس‌ها با نحوه‌ی تعریف و استفاده‌ی مجدد از کدهای خود آشنا هستند. برای مثال تعریف متغیرها، توابع، کلاس‌ها یا کتابخانه‌های خارجی؛ اما این مفهوم در XAML چگونه پیاده سازی می‌شود؟ در کدهای XAML نیز می‌توان منابعی با قابلیت استفاده‌ی مجدد را تعریف نمود که به آن‌ها XAML Resources نیز گفته می‌شود. هر منبع تعریف شده در XAML از نوع ResourceDictionary می‌باشد؛ بنابراین قید ویژگی Key آن‌ها حین تعریف، الزامی است. هر FrameworkElement تعریف شده در Silverlight (مانند User control ، Page ، کلیه عناصر طرح بندی و غیره) دارای خاصیتی هستند به نام Resource که از نوع ResourceDictionary می‌باشد و امکان تعریف منابع مخصوص آن‌ها را به سادگی فراهم می‌سازند. برای توضیح کاربردی این موارد لطفاً به مثال بعد دقت بفرمائید :

CalendarDay.cs

```
namespace SilverlightApplication53.Model
{
    public class CalendarDay
    {
        public string DayName { get; set; }
        public bool IsWeekend { get; set; }
    }
}
```

CalendarDays.cs

```
using System.Collections.Generic;

namespace SilverlightApplication53.Model
{
    public class CalendarDays : List<CalendarDay>
    {
        public CalendarDays()
        {
            this.Add(new CalendarDay { DayName = "شنبه", IsWeekend = false });
            this.Add(new CalendarDay { DayName = "یک شنبه", IsWeekend = false });
        }
    }
}
```

```

        this.Add(new CalendarDay { DayName = "دو شنبه", IsWeekend = false });
this.Add(new CalendarDay { DayName = "سه شنبه", IsWeekend = false });
this.Add(new CalendarDay { DayName = "چهارشنبه", IsWeekend = false });
this.Add(new CalendarDay { DayName = "پنج شنبه", IsWeekend = true });
this.Add(new CalendarDay { DayName = "جمعه", IsWeekend = true });
    }
}
}

```

MainPage.xaml

```

<UserControl x:Class="SilverlightApplication53.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:sys="clr-namespace:System;assembly=mscorlib"
    xmlns:biz="clr-namespace:SilverlightApplication53.Model"
    mc:Ignorable="d" d:DesignHeight="300" d:DesignWidth="400">
    <UserControl.Resources>
        <SolidColorBrush Opacity='.5'
            Color='Brown'
            x:Key='SunBrush' />
        <Rectangle Fill='DarkOrange'
            Width='140'
            Height='40'
            x:Key='BurntRectangle' />
    </UserControl.Resources>
    <StackPanel x:Name="LayoutRoot" Background="White">
        <StackPanel.Resources>
            <sys:String x:Key='HelloString'>سلام</sys:String>
            <biz:CalendarDay DayName='Tuesday'
                IsWeekend='False'
                x:Key='day' />
            <biz:CalendarDays x:Key='Days' />
        </StackPanel.Resources>

        <TextBlock FontSize='18'
            Text='Using Resources'
            Foreground='{StaticResource SunBrush}' />
        <TextBlock Text='{StaticResource HelloString}' />
        <ListBox ItemsSource='{StaticResource Days}'
            DisplayMemberPath='DayName' />
    </StackPanel>
</UserControl>

```

در این مثال نحوه‌ی تعریف انواع و اقسام منابع را در XAML مشاهده می‌نمائید. با توجه به اینکه برای هر FrameworkElement می‌توان یک خاصیت Resource نیز تعریف نمود، در اینجا جهت User control و StackPanel تعریف شده، منابعی تعریف شده‌اند. این منابع می‌توانند از نوع‌های پیش فرض تعریف شده در Silverlight باشند مانند یک SolidColorBrush و یا از نوع یک کلاس سفارشی تعریف شده مانند روزهای هفته و کلاس‌های مرتبط با آن که کدهای آن پیشتر ارائه شدند. برای استفاده از این نوع منابع نیاز است تا فضاهای نام مرتبط با آن‌ها در ابتدای فایل XAML معرفی گردند.

پس از تعریف منابع، اکنون نوبت به استفاده‌ی از آن‌ها فرا می‌رسد. با روش تعریف آن‌ها به صورت مختصر در طی فصل‌های قبل نیز آشنا شده‌ایم:

```
{StaticResource resourceName}
```

برای مثال:

```
<TextBlock Text='{StaticResource HelloString}' />
```

TextBlock تعریف شده در اینجا جهت یافتن منبعی به نام HelloString، درخت اشیاء مجاور خود را تا سطح برنامه (فایل App.XAML) طی می‌کند (از پایین‌ترین سطح به بالا). این مثال حاوی یک نکته‌ی جدید Binding نیز می‌باشد:

```
<ListBox ItemsSource='{StaticResource Days}'
        DisplayMemberPath='DayName' />
```

برای این ListBox هیچ نوع DataTemplate ایی تعریف نکرده‌ایم. بنابراین همانند یک ListBox بسیار معمولی عمل کرده و با کمک ویژگی DisplayMemberPath آن، خاصیت مورد نظر نمایشی را از منبع ثابت Days دریافت نموده و نمایش می‌دهد.

مدیریت مرکزی تنظیمات اشیاء با کمک Styles

شبیه به امکان تعریف CSS برای فایل‌های HTML، امکان تعریف Styles در فایل‌های XAML به صورت Resource مهیا است. برای مثال یک سری TextBlock را در نظر بگیرید که ویژگی‌های قلم، اندازه‌ی قلم، رنگ و غیره‌ی آن‌ها یکسان است. تعریف خواص تکراری اشیاء، علاوه بر بالا بردن حجم کار انجام شده، امکان انجام تغییرات در آن‌ها را مشکل می‌سازد. برای مثال اگر قرار باشد رنگ آن‌ها را تغییر دهیم باید تمامی TextBlock‌ها را یافته و ویژگی ذکر شده‌ی آن‌ها را تغییر داد، که این مساله در یک برنامه‌ی بزرگ نه عاقلانه است و نه مقرون به صرفه.

لطفا در ادامه به مثالی در این زمینه دقت بفرمائید. این مثال نحوه‌ی تعریف یک Style را در قسمت Resources برنامه ارائه می‌دهد. توسط یک Style مشخص خواهد شد که نوعی که قرار است تنظیمات تعریف شده به آن

اعمال گردد چیست (TargetType). سپس به کمک یک سری Setter نام خاصیت‌ها و مقدار آن‌ها مشخص می‌گردد. در پایان جهت اعمال این Style جدید، تنها کافی است ویژگی Style آن کنترل مقدار دهی گردد:

MainPage.xaml

```
<UserControl x:Class="SilverlightApplication53.DefineStyles"
    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>
        <Style x:Key="btnStyle" TargetType="Button">
            <Setter Property="Background" Value="Brown" />
            <Setter Property="Margin" Value="20" />
        </Style>
    </UserControl.Resources>
    <StackPanel>
        <Button Content="Button1" Margin="5" />
        <Button Content="Button2" Margin="5"
            Style="{StaticResource btnStyle}"
        />
    </StackPanel>
</UserControl>
```

تعریف Styles به صورت آبشاری

با کمک ویژگی BasedOn می‌توان تعاریف یک Style را در Style دیگری به ارث برد. برای مثال دو نوع Style ویژه را برای نمایش متن طراحی کرده‌اید که جهت نمایش عنوان و متن اصلی به کار می‌روند و تنها در یک یا دو خاصیت اندازه‌ی قلم و رنگ آن با هم متفاوت هستند، اما سایر خواص یکسانی را باید ارائه دهند.

XAML

```
<!-- Base style for all controls -->
<Style x:Key="BaseStyle"
    TargetType="Control">
    <Setter Property="FontSize"
        Value="14" />
    <Setter Property="FontFamily"
        Value="Verdana" />
    <Setter Property="Foreground"
        Value="DarkBlue" />
</Style>
```

```

<!-- Base style for all Content controls -->
<Style x:Key="ContentControlBaseStyle"
TargetType="ContentControl"
BasedOn="{StaticResource BaseStyle}">
    <Setter Property="HorizontalContentAlignment"
Value="Center" />
    <Setter Property="VerticalContentAlignment"
Value="Center" />
</Style>

```

استفاده از فایل‌های Style

امکان قرار دادن تعاریف Styles در فایل‌های XAML به صورت جداگانه نیز وجود دارد که عموماً جهت ارائه‌ی قالب‌ها مورد استفاده قرار می‌گیرند. برای اعمال این نوع فایل‌ها می‌توان از فایل App.XAML به صورت زیر استفاده کرد:

۱. تعریف فایل منابع خارجی:

با استفاده از منوی پروژه، گزینه‌ی Add new Item و سپس انتخاب افزودن Silverlight Resource Dictionary :

NewTheme.XAML

```

<ResourceDictionary
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
    <!-- Resource dictionary entries should be defined here. -->
</ResourceDictionary>

```

۲. استفاده از فایل منابع خارجی و تعریف آن در سطح کل برنامه:

App.XAML

```

<Application.Resources>
    <ResourceDictionary Source="Themes/newTheme.xaml"/>
</Application.Resources>

```

برای مثال یک پروژه‌ی جدید را آغاز نموده و سپس پوشه‌ی جدید Themes را به آن اضافه نمایید. در این پوشه یک Silverlight Resource Dictionary جدید را به نام Blue.XAML با محتوای زیر ایجاد کنید

(همچنین فراموش نکنید که در خواص این فایل در VS.NET باید خاصیت Build action را به Content تغییر داد):

Blue.xaml

```
<ResourceDictionary
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
    <!--DEFINING A LIST OF COLORS FOR RE-USE-->
    <Color x:Key="TextBrush">#FF000000</Color>

    <Color x:Key="NormalBrushGradient1">#FFBAE4FF</Color>
    <Color x:Key="NormalBrushGradient2">#FF398FDF</Color>
    <Color x:Key="NormalBrushGradient3">#FF006DD4</Color>
    <Color x:Key="NormalBrushGradient4">#FF0A3E69</Color>

    <Color x:Key="NormalBorderBrushGradient1">#FFBBBBBB</Color>
    <Color x:Key="NormalBorderBrushGradient2">#FF737373</Color>
    <Color x:Key="NormalBorderBrushGradient3">#FF646464</Color>
    <Color x:Key="NormalBorderBrushGradient4">#FF000000</Color>

    <Color x:Key="ShadeBrushGradient1">#FF62676A</Color>
    <Color x:Key="ShadeBrushGradient2">#FFD1D4D6</Color>
    <Color x:Key="ShadeBrushGradient3">#FFFFFFFF</Color>

    <Color x:Key="SliderBorderGradient1">#FF3F3F3F</Color>
    <Color x:Key="SliderBorderGradient2">#FFADADAD</Color>

    <!--DEFINING A LIST OF BRUSHES FOR RE-USE-->
    <LinearGradientBrush x:Key="NormalBrush"
        EndPoint="0.5,1" StartPoint="0.5,0">
        <GradientStop Color="{StaticResource NormalBrushGradient1}"
            Offset="0" />
        <GradientStop Color="{StaticResource NormalBrushGradient2}"
            Offset="0.41800001263618469" />
        <GradientStop Color="{StaticResource NormalBrushGradient3}"
            Offset="0.418" />
        <GradientStop Color="{StaticResource NormalBrushGradient4}"
            Offset="1" />
    </LinearGradientBrush>

    <LinearGradientBrush x:Key="NormalBorderBrush" EndPoint="0.5,1"
        StartPoint="0.5,0">
        <GradientStop
            Color="{StaticResource NormalBorderBrushGradient1}" />
        <GradientStop
            Color="{StaticResource NormalBorderBrushGradient2}"
            Offset="0.38" />
        <GradientStop
```



```

        Color="{StaticResource NormalBorderBrushGradient3}"
        Offset="0.384" />
    <GradientStop
        Color="{StaticResource NormalBorderBrushGradient4}"
        Offset="1" />
</LinearGradientBrush>

<LinearGradientBrush
    x:Key="ShadeBrush" EndPoint="0.5,1" StartPoint="0.5,0">
    <GradientStop
        Color="{StaticResource ShadeBrushGradient2}" Offset="0" />
    <GradientStop
        Color="{StaticResource ShadeBrushGradient3}" Offset="0.1" />
    <GradientStop
        Color="{StaticResource ShadeBrushGradient3}" Offset="1" />
</LinearGradientBrush>

<!--Button-->
<Style TargetType="Button">
    <Setter Property="Background"
        Value="{StaticResource NormalBrush}"/>
    <Setter Property="Foreground" Value="#FF000000"/>
    <Setter Property="Padding" Value="3"/>
    <Setter Property="BorderThickness" Value="2"/>
    <Setter Property="BorderBrush"
        Value="{StaticResource NormalBorderBrush}" />
</Style>

<!--TextBox-->
<Style TargetType="TextBox">
    <Setter Property="BorderThickness" Value="1"/>
    <Setter Property="Background"
        Value="{StaticResource NormalBrushGradient1}"/>
    <Setter Property="Foreground" Value="#FF000000"/>
    <Setter Property="Padding" Value="2"/>
    <Setter Property="BorderBrush"
        Value="{StaticResource NormalBorderBrush}" />
</Style>

<!--Slider-->
<Style TargetType="Slider">
    <Setter Property="BorderThickness" Value="1"/>
    <Setter Property="Maximum" Value="10"/>
    <Setter Property="Minimum" Value="0"/>
    <Setter Property="Value" Value="0"/>
    <Setter Property="BorderBrush">
        <Setter.Value>
            <LinearGradientBrush EndPoint="0.5,1"

```

```

        StartPoint="0.5,0">
        <GradientStop Color="#FFA3AEB9" Offset="0"/>
        <GradientStop Color="#FF8399A9" Offset="0.375"/>
        <GradientStop Color="#FF718597" Offset="0.375"/>
        <GradientStop Color="#FF617584" Offset="1"/>
    </LinearGradientBrush>
</Setter.Value>
</Setter>
</Style>
</ResourceDictionary>

```

در این فایل منبع، نحوه‌ی تعریف قالب‌های جدید یک سری کنترل مانند Slider ، TextBox و Button را ملاحظه می‌نمائید. سپس نحوه‌ی اعمال این قالب جدید و معرفی آن به صورت یک ResourceDictionary به صفحه‌ی اصلی برنامه به صورت زیر خواهد بود:

MainPage.xaml

```

<UserControl x:Class="SilverlightApplication58.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>
        <ResourceDictionary Source="Themes/Blue.xaml" />
    </UserControl.Resources>
    <StackPanel x:Name="LayoutRoot" Width="200"
        HorizontalAlignment="Center" VerticalAlignment="Center">
        <TextBlock Text="This is our form." Margin="0,0,0,10"/>
        <Button Content="Save" Margin="0,0,0,10"/>
        <TextBox Text="Something to edit." Margin="0,0,0,10"/>
        <Slider/>
    </StackPanel>
</UserControl>

```

به این ترتیب دیگر نیازی نخواهد بود تا به صورت دستی خاصیت Style تک تک عناصر صفحه را مقدار دهی نمود و تعاریف قالب‌ها بر اساس محتوای فایل Resource Dictionary تعریف شده، دریافت می‌گردند. این امکان تعریف و اعمال مستقیم فایل‌های منابع جزو ویژگی‌های جدید Silverlight 4 بوده و در نگارش‌های قبلی، از Silverlight toolkit و ImplicitStyleManager آن کمک گرفته می‌شد که اکنون جزئی از Silverlight 4 است.

همچنین باید در نظر داشت که عموماً فایل‌های منبع همانند مثال قبل بسیار حجیم شده و مدیریت آن‌ها مشکل می‌شود. به همین جهت امکان تعریف محتوای آن‌ها در چندین فایل منبع مجزا و سپس یکی کردن آن‌ها به کمک MergedDictionaries نیز میسر است که روش آن‌را در مثال زیر ملاحظه می‌نمائید :

MainPage.xaml

```
...
<ResourceDictionary>
  <ResourceDictionary.MergedDictionaries>
    <ResourceDictionary Source="ResourceA.xaml"/>
    <ResourceDictionary Source="ResourceB.xaml"/>
  </ResourceDictionary.MergedDictionaries>
</ResourceDictionary>
...
```

تعریف ظاهری دگرگون شده برای کنترل‌های بصری برنامه

توسط Styles تنها می‌توان ویژگی‌های کنترل‌ها را به صورت یکپارچه مقدار دهی نمود. اما اگر به برنامه‌های چندرسانه‌ای تهیه شده با Silverlight دقت نمائید، اشیایی نامتعارف را مشاهده خواهید کرد؛ دکمه‌هایی گرد و یا کنترل‌هایی که از لحاظ ظاهری در مقایسه با نمونه‌ی اصلی کاملاً دگرگون شده‌اند. Silverlight برای سهولت بخشیدن به تغییر ظاهر پیش فرض کنترل‌های تعریف شده، Control Template را معرفی کرده است (به آن control skinning هم گفته می‌شود). به این صورت شما می‌توانید شکل ظاهری یک دکمه را کاملاً دگرگون نمائید، اما این دکمه هنوز همان شیء دکمه است با تمام خواص و رخدادهای منتسب به آن. لطفاً جهت توضیح کاربردی این مبحث به مثال بعد دقت بفرمائید.

App.xaml

```
<Application xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  x:Class="SilverlightApplication54.App"
  >
  <Application.Resources>
    <Style x:Key="RoundButton" TargetType="Button">
      <Setter Property="Template">
        <Setter.Value>
          <ControlTemplate TargetType="Button">
            <Grid>
              <Ellipse Width="200" Height="200">
                <Ellipse.Fill>
                  <RadialGradientBrush
```

```

        GradientOrigin=".2,.2">
        <GradientStop Offset=".2"
            Color="White" />
        <GradientStop Offset="1"
            Color="Blue" />
        </RadialGradientBrush>
    </Ellipse.Fill>
</Ellipse>
<TextBlock Text="Push me!" FontSize="28"
    HorizontalAlignment="Center"
    VerticalAlignment="Center"
/>
</Grid>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>
</Application.Resources>
</Application>

```

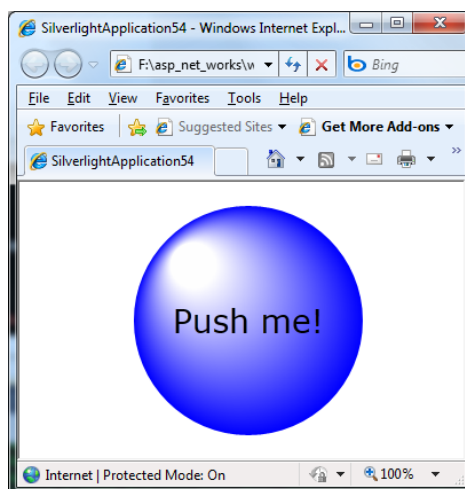
از آنجائیکه می‌خواهیم قالب جدید تعریف شده در تمامی فایل‌های برنامه در دسترس باشد، آن‌را در فایل App.XAML تعریف نموده‌ایم. به این شکل با تعریف یک ControlTemplate جدید، شکل دکمه‌ی تعریف شده را به صورت یک بیضی و متن داخل آن تغییر داده‌ایم. ابتدا یک Style جدید تعریف شده است. سپس خاصیت آن مساوی Template و مقدار آن مساوی ControlTemplate تعریف شده، قرار گرفته است. اکنون اگر از این قالب تعریف شده در صفحه‌ی اصلی برنامه استفاده نمائیم، روش کار و نتیجه‌ی نهایی به صورت زیر خواهد بود (شکل ۱):

MainPage.xaml

```

<UserControl x:Class="SilverlightApplication54.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">
        <Button Width="400" Height="200"
            Style="{StaticResource RoundButton}" />
    </Grid>
</UserControl>

```



شکل ۱- نمایش از ظاهر دگرگون شده‌ی یک دکمه با کمک ControlTemplates.

حاصل نهایی به دو علت زیر مطلوب نیست:

۱. اگر طول و عرض دکمه‌ی نهایی را تغییر دهید، تغییری را مشاهده نخواهید کرد زیرا طول و عرض و متن نهایی در Style مربوطه به صورت صریح قید شده‌اند.
۲. امکان تعریف محتوای جدید، داخل این دکمه‌ی سفارشی شده وجود ندارد. برای مثال دیگر نمی‌توان یک تصویر را در این بین قرار داد و دکمه‌ای ترکیبی را شاهد بود.

این مشکلات به کمک روش‌های زیر قابل حل می‌باشند:

۱. بجای مقدار دهی مستقیم و صریح خواص در فایل Style، آن‌ها را به صورت {TemplateBinding ControlProperty} تعریف کنید. برای مثال بجای ControlProperty، طول و عرض و غیره، قرار گرفته و مقدار نهایی را از دکمه‌ی تعریف شده دریافت خواهند نمود.
۲. بجای استفاده از TextBlock از کنترل ContentPresenter استفاده کنید. به این ترتیب امکان تعریف و ایجاد دکمه‌های ترکیبی همانند قبل میسر می‌شود.

به این ترتیب پس از اعمال نکات فوق خواهیم داشت:

App.xaml

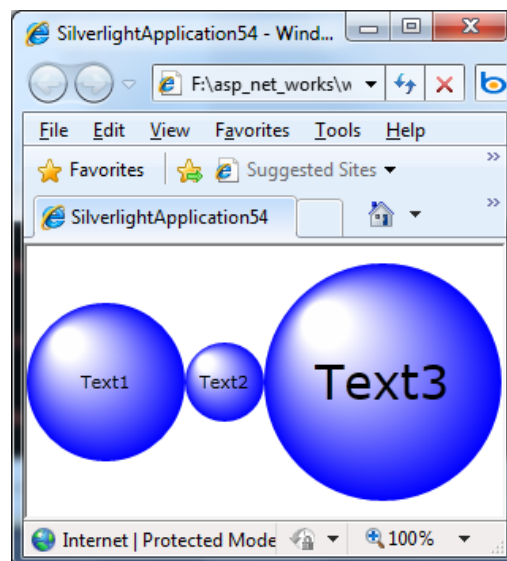
```
<Application xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    x:Class="SilverlightApplication54.App"
    >
    <Application.Resources>
        <Style x:Key="RoundButton" TargetType="Button">
            <Setter Property="Template">
                <Setter.Value>
                    <ControlTemplate TargetType="Button">
```

```

        <Grid>
        <Ellipse
            Width="{TemplateBinding Width}"
            Height="{TemplateBinding Height}">
            <Ellipse.Fill>
                <RadialGradientBrush
                    GradientOrigin=".2,.2">
                    <GradientStop Offset=".2"
                        Color="White" />
                    <GradientStop Offset="1"
                        Color="Blue" />
                </RadialGradientBrush>
            </Ellipse.Fill>
        </Ellipse>
        <ContentPresenter
            Content="{TemplateBinding Content}"
            HorizontalAlignment="Center"
            VerticalAlignment="Center"
            />
        </Grid>
    </ControlTemplate>
</Setter.Value>
</Setter>
</Style>
</Application.Resources>
</Application>

```

در ادامه جهت بررسی تغییرات صورت گرفته، سه دکمه را با اندازه‌های متفاوت بر روی فرم قرار خواهیم داد (شکل زیر):



شکل ۲- سه دکمه دایره‌ای با اندازه‌های متفاوت

```
<UserControl x:Class="SilverlightApplication54.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">
    <StackPanel Orientation="Horizontal">
        <Button Width="100" Height="100"
            Style="{StaticResource RoundButton}"
            Content="Text1"
        />
        <Button Width="50" Height="50"
            Style="{StaticResource RoundButton}"
            Content="Text2"
        />
        <Button Width="150" Height="150"
            Style="{StaticResource RoundButton}"
            Content="Text3"
            FontSize="30"
        />
    </StackPanel>
</UserControl>
```

تا اینجا توانستیم ظاهر پیش فرض یک دکمه را کاملاً دگرگون سازیم. اما حاصل نهایی باز هم مطلوب نیست! زیرا یک دکمه‌ی استاندارد در حالات متفاوتی مانند عبور اشاره‌گر Mouse از روی آن یا فشردن شدن و امثال آن، ظاهر اندکی متفاوت را به خود می‌گیرد و برای نمونه حس فشردن شدن دکمه را به کاربر القاء می‌کند. برای فائق آمدن بر این مشکلات و نقایص، شیء VisualState به Silverlight اضافه شده است. برای مثال توسط VisualState می‌توان یک پویانمایی (Animation) را حین عبور اشاره‌گر Mouse از روی دکمه ایجاد نمود.

معرفی کلاس VisualStateManager

حالات (States) جهت تعریف ساده‌تر ظاهر کنترل‌ها معرفی شده‌اند. بنابراین هر حالت نمایانگر ظاهری متفاوت خواهد بود. برای کاهش تعداد تعریف‌هایی که باید صورت گیرد، حالات به گروه‌های مختلفی تقسیم شده‌اند. برای درک بهتر این مبحث لطفاً به جدول زیر دقت بفرمائید. در این جدول گروه‌های مختلف حالات یک کنترل CheckBox به همراه ریز حالات تعریف شده در هر گروه ذکر گردیده است :

Group Name	States
CommonStates	Normal, MouseOver, Pressed, Disabled
CheckStates	Checked, Unchecked, Indeterminate
FocusStates	Focused, ContentFocused, Unfocused

و تمام این حالات در زمان تعریف شیء CheckBox توسط ویژگی TemplateVisualState معرفی شده‌اند :

C#

```
[TemplateVisualState(Name="ContentFocused", GroupName = "FocusStates")]
[TemplateVisualState(Name = "MouseOver", GroupName = "CommonStates")]
[TemplateVisualState(Name = "Focused", GroupName = "FocusStates")]
[TemplateVisualState(Name = "Checked", GroupName = "CheckStates")]
[TemplateVisualState(Name = "Unchecked", GroupName = "CheckStates")]
[TemplateVisualState(Name = "Indeterminate", GroupName = "CheckStates")]
[TemplateVisualState(Name = "Pressed", GroupName = "CommonStates")]
[TemplateVisualState(Name = "Disabled", GroupName = "CommonStates")]
[TemplateVisualState(Name = "Unfocused", GroupName = "FocusStates")]
[TemplateVisualState(Name = "Normal", GroupName = "CommonStates")]
public class CheckBox : ToggleButton ...
```

برای اینکه بتوان استفاده‌ی مفیدی را از حالات تعریف شده کرد نیاز به کلاس VisualStateManager می‌باشد. VisualStateManager کلاسی است که کار مدیریت انتقال بین حالات مختلف را برعهده دارد. هر حالت دارای یک نام و یک Storyboard جهت تعریف پویانمایی، می‌باشد که در حین نمایش حالت مورد نظر اجرا خواهد شد. ذکر خاصیت Transition، در اینجا ضروری نیست اما اگر تعریف گردد زمان انتقال بین حالات مختلف را می‌توان مقدار دهی نمود. حالت پیش فرض، انتقال آنی است از یک حالت به حالتی دیگر. نحوه‌ی تعریف استاندارد یک VisualStateManager به همراه گروه‌های آن و همچنین خاصیت Transition را در ادامه ملاحظه خواهید نمود :

XAML

```
<vsm:VisualStateManager.VisualStateGroups>
  <vsm:VisualStateGroup x:Name="CommonStates">
    <vsm:VisualState x:Name="MouseOver">
      <Storyboard>
        ...
      </Storyboard>
    </vsm:VisualState>
    <vsm:VisualState x:Name="Normal">
      <Storyboard>
        ...
      </Storyboard>
    </vsm:VisualState>
```



```

        <Storyboard>
            <ColorAnimation
                Storyboard.TargetName="tickBox"
                Storyboard.TargetProperty=
                    "(Rectangle.Fill).(SolidColorBrush.Color)"
                To="Brown" Duration="0:0:0.5" />
            </Storyboard>
        </VisualState>
        <VisualState x:Name="Disabled">
            <Storyboard>
                <ColorAnimation
                    Storyboard.TargetName="tickBox"
                    Storyboard.TargetProperty=
                        "(Rectangle.Fill).(SolidColorBrush.Color)"
                    To="Gray" Duration="0:0:0.5" />
                </Storyboard>
            </VisualState>
        <VisualState x:Name="Normal">
            <Storyboard>
                <ColorAnimation
                    Storyboard.TargetName="tickBox"
                    Storyboard.TargetProperty=
                        "(Rectangle.Fill).(SolidColorBrush.Color)"
                    Duration="0:0:0.5" />
                </Storyboard>
            </VisualState>
        </VisualStateGroup>
        <VisualStateManager.VisualStateGroups>
            <VisualStateGroup x:Name="CheckStates">
                <VisualState x:Name="Checked">
                    <Storyboard>
                        <DoubleAnimation
                            Storyboard.TargetName="tick"
                            Storyboard.TargetProperty="Opacity"
                            To="1" Duration="0" />
                        </Storyboard>
                    </VisualState>
                <VisualState x:Name="Unchecked" />
            </VisualStateGroup>
        </VisualStateManager.VisualStateGroups>

        <Rectangle
            x:Name="tickBox"
            Stroke="Blue" Width="12"
            Height="12" HorizontalAlignment="Left"
            RadiusX="2" RadiusY="2" Fill="Aqua" />

        <Path x:Name="tick" Opacity="0"
            Width="12" Height="12"
            HorizontalAlignment="Left"
            Stroke="Black" StrokeThickness="2"

```

```

Data="M3,6 L5,9 9,2" />

<ContentPresenter Grid.Column="1" />
</Grid>
</ControlTemplate>
</CheckBox.Template>
</CheckBox>
</Grid>
</UserControl>

```

در این مثال روش دیگر معرفی ControlTemplate را به ازای یک کنترل ملاحظه می‌نمائید (این روش در مورد تعریف Style تنها برای یک کنترل نیز کاربرد دارد). بدیهی است اگر نیاز به استفاده از این قالب جدید در کل برنامه وجود داشته باشد بهتر است یک Style جدید را در فایل App.XAML و قسمت Application.Resources آن معرفی نمود تا بتوان از قابلیت استفاده‌ی مجدد آن بهره برد. در اینجا به کمک یک مستطیل حاشیه‌ی سفارشی CheckBox جدید، ایجاد شده است و سپس بوسیله‌ی شیء Path، علامت تیک آن بازسازی گردیده است. سپس جهت بهبود حالات نمایشی آن، اکثر حالات مفید یک CheckBox بر اساس گروه‌های مختلف استاندارد آن، تعریف و مقدار دهی شده‌اند. در مورد پویا نمایی، Storyboard و جزئیات آن در طی فصلی مجزا به صورت مفصل بحث خواهد شد.

استفاده از قالب‌های آماده در Silverlight

امکانات قابل توجهی را در فصل جاری جهت تغییر ظاهر پیش فرض کنترل‌های استاندارد Silverlight ملاحظه نمودید. تاکنون بسیاری از طراحان حرفه‌ای نسبت به ایجاد قالب‌هایی بسیار زیبا برای برنامه‌های Silverlight اقدام کرده‌اند که تعدادی از آن‌ها را (شکل ۳) در آدرس زیر می‌توانید مشاهده و دریافت کنید (جزئی از مجموعه‌ی Silverlight toolkit):

<http://bit.ly/9V6lEP>

یک سری قالب جدید نیز با ارائه‌ی 4 Silverlight به این مجموعه اضافه شده است که از آدرس‌های زیر قابل دریافت است:

<http://tinyurl.com/2ep68po>

<http://go.microsoft.com/fwlink/?LinkId=192411>

<http://drop.io/sltemplates>

همچنین قالب‌های بیشتری از این دست را می‌توان در آدرس زیر، قسمت Themes دریافت نمود:

<http://gallery.expression.microsoft.com/en-us/>













روش‌های متفاوتی برای اعمال قالب‌های Silverlight toolkit به یک برنامه‌ی Silverlight وجود دارند که در ادامه آن‌ها را مرور خواهیم نمود:

۱. استفاده از جعبه ابزار VS.NET

پس از نصب Silverlight toolkit، قالب‌های قابل استفاده در جعبه ابزار VS.NET ظاهر خواهند شد (شکل ۴). برای نمونه BubbleCremeTheme را از جعبه ابزار VS.NET کشیده و بر روی فرم رها کنید. به این صورت ارجاعات لازم به اسمبلی حاوی این قالب و همچنین Silverlight toolkit اضافه شده، به علاوه فضای نام مربوطه را نیز به فایل XAML جاری اضافه می‌کند. اکنون تنها کافی است که ناحیه مورد نظر خود را جهت اعمال این قالب، با تگ‌های قالب اضافه شده محصور نمائیم که نمونه‌ای از کدهای XAML آن را در ادامه مشاهده خواهید نمود (شکل ۵).



شکل ۳- قالب‌های رایگان برنامه‌های Silverlight موجود در Silverlight toolkit

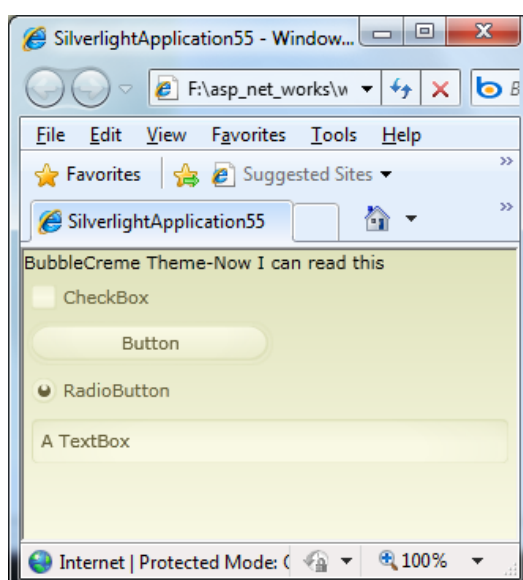
-  BubbleCremeTheme
-  BureauBlackTheme
-  BureauBlueTheme
-  ExpressionDarkTheme
-  ExpressionLightTheme
-  RainierOrangeTheme
-  RainierPurpleTheme
-  ShinyBlueTheme
-  ShinyRedTheme
-  SystemColorsTheme
-  TwilightBlueTheme
-  WhistlerBlueTheme

شکل ۴- قالب‌های نصب شده به همراه Silverlight toolkit

محصول سازی یک StackPanel با قالب BubbleCremeTheme :

MainPage.xaml

```
<UserControl x:Class="SilverlightApplication55.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">
    <Grid x:Name="LayoutRoot" Background="White">
        <toolkit:BubbleCremeTheme Name="bubbleCremeTheme1" >
            <StackPanel>
                <TextBlock Text="BubbleCreme Theme-Now I can read this" />
                <CheckBox Content="CheckBox" Margin="5" IsChecked="true" />
                <Button HorizontalAlignment="Left" Content="Button"
                    Width="150" Margin="5"/>
                <RadioButton Content="RadioButton" Margin="5" />
                <TextBox Text="A TextBox" Margin="5" />
            </StackPanel>
        </toolkit:BubbleCremeTheme>
    </Grid>
</UserControl>
```



شکل ۵- نمایش از برنامه در حالت اعمال قالب BubbleCreme .

سایر قالب‌های موجود نیز به همین ترتیب قابل استفاده می‌باشند. با کشیدن و رها کردن آن‌ها بر روی فرم، ارجاع لازم به اسمبلی مخصوص آن‌ها اضافه شده و سپس توسط فضای نام مرتبط با Silverlight toolkit قابل استفاده خواهند بود.

۲. تعریف قالب مورد نظر در قسمت **Application.Resources**

در این روش ابتدا ارجاعاتی را به اسمبلی‌های زیر باید اضافه نمائید :

```
System.Windows.Controls
System.Windows.Controls.Data
System.Windows.Controls.Data.DataForm.Toolkit
System.Windows.Controls.Data.Input
System.Windows.Controls.DataVisualization.Toolkit
System.Windows.Controls.Input
System.Windows.Controls.Input.Toolkit
System.Windows.Controls.Layout.Toolkit
System.Windows.Controls.Theming.Toolkit
System.Windows.Controls.Toolkit
```

اسمبلی‌های مرتبط با Silverlight toolkit را در محل نصب آن‌ها می‌توانید پیدا کنید. برای مثال :

C:\Program Files (x86)\Microsoft SDKs\Silverlight\v4.0\Toolkit\month\Bin

سپس یک پوشه جدید به نام Themes را به پروژه‌ی جاری اضافه کرده و فایل زیر را در آن کپی نمائید (یکی از چندین قالب موجود جهت آزمایش). در ادامه با مراجعه به خواص این فایل در VS.NET، Build Action آن‌را به Content تنظیم نمائید:

System.Windows.Controls.Theming.BureauBlue.xaml

این فایل را در مسیر زیر می‌توانید پیدا کنید :

C:\Program Files (x86)\Microsoft SDKs\Silverlight\v4.0\Toolkit\month\Themes\Xaml

در ادامه باید ارجاعی را نیز به فایل اسمبلی آن اضافه کرد:

System.Windows.Controls.Theming.BureauBlue

این اسمبلی در مسیر زیر قرار دارد :

C:\Program Files (x86)\Microsoft SDKs\Silverlight\v4.0\Toolkit\month\Themes\

اکنون کار مقدمات افزودن پیش نیازها به پایان می‌رسد و می‌توان فایل قالب جدید را به فایل App.XAML

برنامه معرفی کرد:

App.xaml

```
<Application xmlns=
"http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
x:Class="SilverlightApplication56.App">
<Application.Resources>
<ResourceDictionary
Source="Themes/System.Windows.Controls.Theming.BureauBlue.xaml"/>
</Application.Resources>
</Application>
```

به این صورت دیگر نیازی نخواهد بود تا تغییری در قسمت‌های مختلف برنامه داده شود، زیرا این قالب تعریف شده در فایل App.XAML به صورت یکنواخت به تمام قسمت‌های پروژه اعمال می‌گردد.

۳. اعمال قالب‌ها با کمک کدنویسی و در زمان اجرای برنامه

برای کد نویسی نیز ابتدا باید اعمال ذکر شده در روش دوم را مرحله به مرحله دنبال نمود. ابتدا فایل قالب خود را از مسیر Themes\Xaml واقع در محل نصب Silverlight toolkit به پوشه‌ی جدیدی در برنامه به نام Themes انتقال داده و در خواص این فایل، Build action را به Content تغییر دهید. اکنون یکبار پروژه را Build نمایید تا تعداد اسمبلی‌هایی که باید به پروژه اضافه شوند در خطاهای حاصل مشخص گردند که لیست آن‌ها در قسمت دوم، ذکر گردید. برای آزمودن امکان بارگذاری آن قالب خاص، یکبار روش دوم را بررسی کنید (افزودن تعاریف به فایل App.XAML). اگر خطایی وجود داشته باشد در حین اجرا و Debug برنامه در VS.NET مشخص می‌شود (عموما عدم افزودن ارجاعی به اسمبلی متناظر با هر قالب، سبب عدم بارگذاری برنامه خواهد شد). پس از اینکه مطمئن شدید، قالب یا قالب‌های مورد نظر بدون مشکل قابل استفاده هستند، تعریف منابع اضافه شده در فایل App.XAML را حذف کنید، تا بتوانیم در ادامه از روش کد نویسی و بارگذاری پویای قالب‌ها استفاده نماییم:

MainPage.xaml.cs

```
using System;
using System.Windows;

namespace SilverlightApplication57
{
    public partial class MainPage
    {
        public MainPage()
        {
            InitializeComponent();
            this.Loaded += pageLoaded;
        }

        void pageLoaded(object sender, RoutedEventArgs e)
        {
            var rd = new ResourceDictionary();
            rd.Source =
                new Uri("/Themes/System.Windows.Controls.Theming.RainierOrange.xaml",
                    UriKind.Relative);
            //Load resource dictionary
            //Clear previous styles if any...
            Application.Current.Resources.MergedDictionaries.Clear();
            //Add the loaded resource dictionary
            //to the application merged dictionaries
            Application.Current.Resources.MergedDictionaries.Add(rd);
        }
    }
}
```

```
}  
}  
}
```

جهت یادآوری از فصل‌های قبل لازم به ذکر است که استفاده از گزینه‌ی زیر در خواص این نوع پروژه‌ها جهت بارگذاری‌های بعدی سریع‌تر برنامه، به شدت توصیه می‌گردد:

Reduce XAP size by using application library caching