CHƯƠNG II:
XAML - Extensible Application Markup Language

1 Tổng quan về XAML

1.1 XAML là gì

XAML được viết tắt bởi cụm từ: Extensible Application Markup Language là một ngôn ngữ dạng khai báo. Bạn có thể tạo ra các phần tử đồ họa(UI) với những khai báo thông qua thẻ trong XAML. Sau đó bạn có thể dùng file mã lệnh tách biệt của nó(code-behind) để trả về những sự kiện và điều khiển những đối tượng mà bạn đã định nghĩa trong XAML. Nó là một ngôn ngữ mô tả dựa trên XML là rất trực quan để xây dựng giao diện từ những bước phác thảo cho tới sản xuất sản phẩm, đặc biệt hữu ích cho đối tượng có kinh nghiệm thiết kế website và kỹ thuật.

1.2 Khai báo đối tượng

Có hai cách để khai báo đối tượng trong XAML:

1.2.1 Khai bao trực tiếp:

Sử dụng thẻ đóng mở để khai báo một đối tượng giống như là một phần tử XML. Bạn cũng có thể sử dụng cú pháp này để khai báo đối tượng gốc (root object) hoặc để xét các giá trị các thuộc tính.

1.2.2 Khai báo gián tiếp:

Sử dụng giá trị trực tiếp để khai báo một đối tượng. Bạn có thể sử dụng cú pháp này để thiết lập giá trị của thuộc tính.Thông thường, điều này có nghĩa là chỉ những thuộc tính mà được hỗ trợ bởi đối tượng mới có thể sử dụng được phương pháp này.

1.3 Thiết lập đặc tính cho đối tượng

Có những cách sau để khai báo đặc tính cho đối tượng:

1.3.1 Sử dụng cú pháp theo thuộc tính

Dưới đây là ví dụ xét giá trị cho các thuộc tính: Width, Height, Fill của đối tượng Rectangle:
<Rectangle Width="100" Height="100" Fill="Blue" />

1.3.2 Sử dụng cú pháp theo đặc tính của thành phần(element):

Dưới đây là ví dụ xét đặc tính **Fill** theo cách này cho đối tượng Rectangle:

```
<Rectangle Width="100" Height="100">
    <Rectangle.Fill>
        <SolidColorBrush Color="Blue"/>
    </Rectangle.Fill>
</Rectangle>
```

1.3.3 Sử dụng cú pháp theo nội dung

Dưới đây là ví dụ xét đặc tính **Text** cho đối tượng TextBlock(Giống như đối tượng Label trong Winform, Webform):

```
<TextBlock>
      Hello!
</TextBlock>
```

1.3.4 Sử dụng theo một tập hợp

Đây là một trường hợp khá thú vị trong XAML, bởi có những cách khác nhau để thể hiện tập hợp này. Hơn nữa nó có thể xuất hiện ở phần đầu tiên của XAML cho phép bạn xét những đặc tính chỉ đọc (read-only) của đối tượng.
Dưới đây là ví dụ xét đặc tính theo những cách khác nhau sử dụng theo kiểu tập hợp.

Cách 1:

```
       <LinearGradientBrush>
            <LinearGradientBrush.GradientStops>
                <!-- Ở đây thẻ GradientStopCollection được chỉ rõ. -->
                <GradientStopCollection>
                    <GradientStop Offset="0.0" Color="Red" />
                    <GradientStop Offset="1.0" Color="Blue" />
                </GradientStopCollection>
            </LinearGradientBrush.GradientStops>
        </LinearGradientBrush>
```

Cách 2:

```
<LinearGradientBrush>
     <LinearGradientBrush.GradientStops>
 <! –Không cần khai báo GradientStopCollection, bộ phân tích sẽ hiểu và tạo ra nó -
->
      <GradientStop Offset="0.0" Color="Red" />
      <GradientStop Offset="1.0" Color="Blue" />
     </LinearGradientBrush.GradientStops>
</LinearGradientBrush>
```

Cách 3:
Ngoài ra, có những đặc tính mà chúng có những tập hợp đặc tính nhưng chúng được xác định như là đặc tính nội dung của lớp. Trong trường hợp này ta xét đến đặc tính GradientStops được xử dụng ở trên. Bạn có thể loại bỏ đặc tính này và sẽ có kết quả như sau:

```
<LinearGradientBrush>
    <GradientStop Offset="0.0" Color="Red" />
    <GradientStop Offset="1.0" Color="Blue" />
</LinearGradientBrush>
```

1.4 Root elements và namespace trong XAML

Một file XAML chỉ được có duy nhật một Root element và phải thỏa mãn cả hai tiêu chí sau: well-formed XML(có mở và đóng thẻ) và valid XML(tuân thủ theo Document Type Definition(DTD)). Ví dụ dưới đây cho Root element điển hình của XAML cho Silverlight với Root element là thành phần UserControl.

```
<UserControl x:Class="MySilverlight.Page"
  xmlns="http://schemas.microsoft.com/client/2007"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
>
    <Grid>
    </Grid>
</UserControl>
```

1.5 Sự kiện

XAML là một ngôn ngữ khai báo cho đối tượng và những đặc tính của chúng, nhưng nó cũng bao gồm những cú pháp cho sự kết hợp sự kiện cho đối tượng trong những thẻ đánh dấu. Bạn chỉ rõ tên của sự kiện như là một thuộc tính tên trên đối tượng mà sự kiện được nghe. Về giá trị của thuộc tính, bạn chỉ rõ tên của hàm nghe sự kiện mà bạn định nghĩa ở phần code-behind hoặc ở phần javascript.
Việc có khai bao hay không đặc tính **x:Class** ở thẻ root trong XAML có ảnh hưởng đến việc xử lý sự kiện. Nếu bạn khai báo x:Class tức là việc xử lý sự kiện của bạn sẽ được thực hiện trong code-behind, trường hợp này thường xuất hiện trong kiểu lập trình Managed API (chứa trong silverlight 2.0). Còn ngược lại thì việc xử lý sự kiện của bạn được thực hiện ngay trong thẻ Javascript chứa trong HTML, trường hợp này thường xuất hiện trong kiểu lập trình JavaScript API (chứa trong Silverlight 1.0).

- Ví dụ dưới đấy chỉ rõ cho bạn thấy cách tạo một sự kiện trong trường hợp kiểu lập trình Manged API

```
<UserControl x:Class="Chapter2.Page"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Width="400" Height="300">
    <Grid x:Name="LayoutRoot" Background="White">
      <Button Click="Button_Click" Content="Click me"></Button>
    </Grid>
</UserControl>
```

Doạn mã trên, trong thẻ root <UserControl> chúng ta đã khai báo đặc tính `x:Class="Chapter2.Page"`, điều này có nghĩa trong chương trình của chúng ta có một file chứa class `Chapter2.Page`. Class này sẽ đảm nhiệm việc xử lý các sự kiện đã khai báo ở file XAML (`Page.xaml`). Trong thẻ `<Grid>` chúng ta tạo thêm một nút <Button> và khai báo trong nút đó một sự kiện `Click="Button_Click"`.Sự kiện này sẽ được xử lý ở trong code-behind như sau

```
namespace Chapter2
{
    public partial class Page : UserControl
    {
        public Page()
        {
            InitializeComponent();
        }

        private void Button_Click(object sender, RoutedEventArgs e)
        {
           //Xu ly su kien Button click o day
            MessageBox.Show("Xin chao!");
        }
    }
}
```

- Còn ví dụ dưới đây sẽ chỉ rõ cho bạn thấy cách tạo một sự kiện trong trường hợp kiểu lập trình JavaScript API

```
<UserControl
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Width="400" Height="300">
    <Grid x:Name="LayoutRoot" Background="White">
        <Button x:Name="button1" Click="Button_Click" Content="Button 1"></Button>
    </Grid>
</UserControl>
```

Doạn mã trên, trong thẻ root <UserControl> chúng ta không khai báo đặc tính x:Class. Việc xử lý sự kiện sẽ được thực hiện ngay trong trang HTML có chứa chương trình silverlight này.

```
<script type="text/javascript">
    function Button_Click() {
        alert("Xin chao!");
    }
</script>
```

2 Các namescope trong XAML

Trong Silverlight, một XAML namescope lưu trữ quan hệ giữa những tên từ khóa của XAML đã được định nghĩa của đối tượng và những khởi tạo tương đương của nó. Điều này cũng tương tự như ý nghĩa của "namescope" trong các ngôn ngữ lập trình và công nghệ khác. XAML namescope được tạo ra trong quá trình dịch mã XAML và trong quá trình tạo hình đối tượng. Những tên mà được tạo trong namescope sau đó được sử dụng ở code-behind thao tác lúc chạy chương trình để truy cập tới đối tượng được tạo bởi quá trình dịch file XAML. Để biết thêm chi tiết các bạn có thể tham khảo tại địa chỉ: http://msdn.microsoft.com/en-us/library/cc189026(VS.95).aspx

3 Sử dụng XAMLReader.Load

Sử dụng JavaScript API cho Silverlight, để tạo đối tượng trong lúc thực thi ứng dụng bắt buộc phải sử dụng qua phương thức CreateFromXaml. Sử dụng manged API những class hoặc cấu trúc (structure) cho phép có thêm các thành phần bên trong thực sự được khởi tạo nếu nó có trong cây thành phần của Silverlight. Trong hầu hết các trường hợp mà bạn muốn khởi tạo đối tượng trong lúc thực thi, bạn có thể đơn giản triệu gọi constructor của class liên quan tới nó.

Tuy vậy, Bạn vẫn có thể tạo đối tượng thông qua đầu vào là XAML thông qua managed API, qua việc sử dụng phương thức **XamlReader.Load**.

Phương thức **Load** trong managed API là tương ứng với **CreateFromXaml** trong JavaScript API, cũng giống như CreateFromXaml, đầu vào cho phương thức **Load** là những chuỗi và đầu ra là những đối tượng mà có thể thêm vào mạng đối dạng cây của Silverlight.

XamlReader đơn giản được thiết kể để đọc xml "XmlReader" có trong Silverlight cũng như trong các công nghệ khác của **Microsoft** . XamlReader là lớp được xây dựng theo kiểu static với những phương thức tạo đối tượng; nó tạo đối tượng song song với việc xử lý XAML để sinh ra trong lúc thực thi(run-time) những cây đối tượng từ XAML trong Silverlight

Các bạn cần lưu ý những điểm sau khi sử dụng phương thức Load:

- Nội dung chuỗi XAML phải định nghĩa một phần tử gốc (root element)
- Nội dung chuỗi XAML phải là well-formed XML, và valid XML

4 XAML và các Custom class

XAML hỗ trợ khả năng có thể định nghĩa tùy chỉnh lớp(custom class) hoặc cấu trúc(structure) trong bất kỳ ngôn ngữ runtime nào (CLR), và sau đó truy cập vào class thông qua thẻ đánh dấu của XAML(XAML markup), bao gồm cách sử dụng hỗn hợp của những thẻ Silverlight đã định nghĩa trong XAML và những thẻ XAML mà tham chiếu tới custom class tương ứng của nó (Mỗi custom class chứa tương ứng 2 file .cs và .xaml, ví dụ: myclass.xaml và myclass.xaml.cs)

4.1 Custom class trong ứng dụng hoặc Assemblies

Custom class dùng trong XAML có thể định nghĩa theo cách riêng biệt:

- Trong code-behind được đóng gói trong ứng dụng Silverlight
- Như là một class được định nghĩa trong một assembly tách biệt, như là một thư viện thực thi hoặc DLL

Mỗi cách trên đều có những ưu và nhược điểm như sau:

- Ưu điểm của việc tạo class và đóng gói riêng biệt là khả năng có thể chia sẻ và dùng được trong nhiều ứng dụng Silverlight khác. Đồng thời là khả năng quản lý phiên bản của control dễ dàng hơn và nó làm cho nó có khả năng tạo ra class mà bạn dự dịnh sử dụng như là một root element trong trang XAML.
- Ưu điểm của việc tạo custom class trong ứng dụng là về mặt kỹ thuật tương đối đơn giản và giảm thiểu kích cỡ và kiểm tra khi bạn gặp vấn đề trong dự án Silverlight dựa trên code-behind. Tuy nhiên có điểm hạn chế là bạn không thể dùng custom class như là một root element. Bạn phải tham chiếu custom class của bạn qua một assembly khác hoặc là giới hạn subclass sử dụng User Control hỗ trợ code-behind trong dự án Silverlight của bạn.
- Dù là định nghĩa trong cùng một assembly hay assembly khác nhau, custom class phải được ánh xạ qua CLR namespace và XML namespace để được tham chiếu trong XAML

4.2 Ràng buộc để Custom Class trở thành thành phần đối tượng trong XAML

Để được tạo đối tượng như là một thành phần đối tượng của XAML, custom class của bạn phải đáp ứng các yếu tố sau đây:

- Custom class phải để public và hỗ trợ khởi tạo mặc định không có tham số truyền vào (default constructure ‘parameterless’).
- Custom class không phải là class lồng(class lồng và dấu chấm ‘.’ ở cú pháp của nó ảnh hưởng tới những đặc điểm của Silverlight như là các property kèm theo.

Ngoài ra để đối tượng của bạn như là một thành phần đối tượng của XAML, bạn có thể cho phép sử dụng những property cho các public property của Custom class của bạn điều này làm cho Custom class của bạn như là một kiểu property. Điều này bởi vì bây giờ đối tượng có thể được khởi tạo giống như một thành phần đối tượng và có thể xét thuộc tính cho nó như là một property

4.3 Yêu cầu đối với sự kiện trong XAML đối với Custom class

Để sử dụng cú pháp theo kiểu thuộc tính để tương tác với sự kiện trong XAML, sự kiện phải được public trong class mà hỗ trợ constructure mặc định, hoặc sự kiện phải được định nghĩa trong class trừu tượng và sau đó sự kiện có thể truy cập được qua những class kế thừa.

5 XAML và Type Converter

6 Layout

Khi bạn xây dựng ứng dụng **Silverlight**, một trong những điều bạn cần làm đầu tiên đó là việc sẽ bố trí giao diện đồ họa của bạn như thế nào. **Silverlight** cung cấp cho bạn 3 kiểu bố trí khác nhau, đó là: **Canvas**, **StackPanel** và **Grid**.

6.1 Canvas

Định nghĩa một khu vực mà trong đó bạn có thể chỉ ra vị trí của từng đối tượng thành phần bằng cách sử dụng các tọa độ tham chiếu. Bạn có thể sử dụng lồng các Canvas với nhau và những thành phần bên trong của Canvas phải là một UIElement. Trong nhiều trường hợp thì Canvas chỉ đóng vai trò như một đối tượng để chứa đựng những đối tượng khác và không có một thuộc tính hiển thị nào. Một Canvas sẽ không được hiển thị nếu nó có một trong những thuộc tính sau đây:

- Thuộc tính Height xét giá trị 0
- Thuộc tính Width xét giá trị 0
- Background bằng null hoặc là Nothing ở VS Basic
- Opacity xét giá trị 0
- Visibility bằng Visibility.Collapsed
- Một trong những Canvas ở mức độ cao hơn (parrent) của nó không được hiển thị.

Ví dụ dưới đây chỉ ra là hình chữ nhật được cách trái 30pixel và cách trên 30pixel

```
<Canvas Width="640" Height="480" Background="White">
    <Rectangle Canvas.Left="30" Canvas.Top="30" Fill="red" Width="200" Height="200"
/>
</Canvas>
```

6.2 StackPanel

Sắp xếp những thành phần bên trong nó thành một dòng và có thể hiển thị theo hai kiểu ngang hoặc là dọc. Giá trị mặc định được gán cho thuộc tính **Orientation** là chiều dọc (**Vertical**) và giá mặc định được xét cho hai thuộc tính **HorizontalAlignment** và **VerticalAlignment** là **Stretch**

Ví dụ dưới đây hướng dẫn cách tạo những những đối tượng trong **StackPanel**

```
<StackPanel Margin="20">
    <Rectangle Fill="Red" Width="50" Height="50" Margin="5" />
    <Rectangle Fill="Blue" Width="50" Height="50" Margin="5" />
    <Rectangle Fill="Green" Width="50" Height="50" Margin="5" />
    <Rectangle Fill="Purple" Width="50" Height="50" Margin="5" />
</StackPanel>
```

6.3 Grid

Định nghĩa dạng lưới phức tạp bao gồm những dòng và cột. Mặc định Grid chứa một cột và một dòng. Để định nghĩa nhiều cột hoặc dòng chúng ta dùng **ColumnDefinitions** và **RowDefinitions.** Mỗi **ColumnDefinition** và **RowDefinition** trong **ColumnDefinitions** và **RowDefinitions** xác định một dòng hoặc cột. **ColumnDefinition** và **RowDefinition** cũng định nghĩa kích thước của mỗi cột và dòng sử dụng đối tượng **GridLength**

Ví dụ: Dưới đây là ví dụ dùng **Grid** để lên một thiết kế giao diện cơ bản.

```
<Grid x:Name="LayoutRoot" Background="#DCDCDC" Width="400" Height="300"
ShowGridLines="True">
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="250" />
        <ColumnDefinition Width="150" />
    </Grid.ColumnDefinitions>
    <Grid.RowDefinitions>
        <RowDefinition Height="Auto" />
        <RowDefinition Height="2*" />
        <RowDefinition Height="*"/>
    </Grid.RowDefinitions>
    <TextBlock Grid.Row="0" Grid.Column="0" Grid.ColumnSpan="2" Margin="10"
FontWeight="Bold" Text="Contoso Corporation" HorizontalAlignment="Center"
VerticalAlignment="Center" />
    <Grid x:Name="FormLayoutGrid" Grid.Row="1" Grid.Column="0"
ShowGridLines="True">
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="Auto" />
            <ColumnDefinition />
        </Grid.ColumnDefinitions>
        <Grid.RowDefinitions>
            <RowDefinition Height="*" />
            <RowDefinition Height="*" />
            <RowDefinition Height="*" />
        </Grid.RowDefinitions>
        <TextBlock Grid.Row="0" Grid.Column="0" Text="First Name" Margin="10"
HorizontalAlignment="Left" VerticalAlignment="Center" />
        <TextBox Grid.Row="0" Grid.Column="1" Margin="10" />
```

```
        <TextBlock Grid.Row="1" Grid.Column="0" Text="Last Name" Margin="10"
HorizontalAlignment="Left" VerticalAlignment="Center" />
        <TextBox Grid.Row="1" Grid.Column="1" Margin="10" />
        <TextBlock Grid.Row="2" Grid.Column="0" Text="Address" Margin="10"
HorizontalAlignment="Left" VerticalAlignment="Center" />
        <TextBox Grid.Row="2" Grid.Column="1" Margin="10" />

    </Grid>
</Grid>
```

Thêm vào một vài đối tượng qua code-behind C#

```
public Page()
 {
    InitializeComponent();
    LayoutDesign();
 }

 private void LayoutDesign()
 {
    //Tạo  Stackpanel cho ListBox Control
    StackPanel DeptStackPanel = new StackPanel();
    DeptStackPanel.Margin = new Thickness(10);

    LayoutRoot.Children.Add(DeptStackPanel);
    Grid.SetColumn(DeptStackPanel, 1);
    Grid.SetRow(DeptStackPanel, 1);

    TextBlock DeptListHeading = new TextBlock();
    DeptListHeading.Text = "Department";

    ListBox DeptList = new ListBox();
    DeptList.Items.Add("Finance");
    DeptList.Items.Add("Marketing");
    DeptList.Items.Add("Human Resources");
    DeptList.Items.Add("Payroll");

    DeptStackPanel.Children.Add(DeptListHeading);
    DeptStackPanel.Children.Add(DeptList);

    //Tạo StackPanel cho các nút
    StackPanel ButtonsStackPanel = new StackPanel();
    ButtonsStackPanel.Margin = new Thickness(10);
    ButtonsStackPanel.Orientation = Orientation.Horizontal;
    ButtonsStackPanel.HorizontalAlignment = HorizontalAlignment.Center;
```

```
        LayoutRoot.Children.Add(ButtonsStackPanel);
        Grid.SetColumn(ButtonsStackPanel, 0);
        Grid.SetRow(ButtonsStackPanel, 2);
        Grid.SetColumnSpan(ButtonsStackPanel, 2);

        Button BackButton = new Button();
        BackButton.Content = "Back";
        BackButton.Height = 30;
        BackButton.Width = 100;

        Button CancelButton = new Button();
        CancelButton.Content = "Cancel";
        CancelButton.Height = 30;
        CancelButton.Width = 100;

        Button NextButton = new Button();
        NextButton.Content = "Next";
        NextButton.Height = 30;
        NextButton.Width = 100;

        ButtonsStackPanel.Children.Add(BackButton);
        ButtonsStackPanel.Children.Add(CancelButton);
        ButtonsStackPanel.Children.Add(NextButton);

        BackButton.Margin = new Thickness(10);
        CancelButton.Margin = new Thickness(10);
        NextButton.Margin = new Thickness(10);
    }
}
```