Silverlight

José Antônio da Cunha
IFRN

E-mail:jose.cunha@ifrn.edu.br

XAML

XAML – Extensible Application Markup Language e pronuncia-se zammel é uma linguagem de marcação usada para instanciar objetos .NET. (Windows Presentation Foundation – WPF).

Conceitualmente, XAML desempenha um papel muito semelhante ao HTML, e está ainda mais perto de seus primos mais rigorosos, XHTML.

Para manipular elementos XHTML, você pode usar o Javascript no lado do cliente. Para manipular elementos XAML, você escreve código C # do lado do cliente.

XAML

XAML básico

O padrão XAML é bastante simples:

•Todos os elementos em um documento XAML mapea uma instância de uma classe Silverligh. O nome do elemento coincide com o nome da classe de precisão. Por exemplo, o elemento <**Button**> instrui Silverlight para criar um objeto **Button**.
•Como acontece com qualquer documento XML, você pode aninhar um elemento dentro de outro.
•Você pode definir as propriedades de cada classe através de atributos. No entanto, em algumas situações um atributo não é suficientemente para realizar tal configuração.Nestes casos, você vai usa tags com uma sintaxe especial.

XAML

Dê uma olhada neste documento XAML, que representa uma página em branco (como as criadas pelo Visual Studio):

```
<UserControl x:Class="SilverlightCH12.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">

   </Grid>
</UserControl>
```

Este documento inclui apenas dois elementos - o elemento de mais alto nível o UserControl, que envolve todo o conteúdo da página Silverlight, e o Grid, no qual você pode colocar todos os seus elementos.

XAML

XAML Namespaces

Quando você usa um elemento como **<UserControl>** em um arquivo XAML, o Silverlight reconhece que você deseja criar uma instância da classe UserControl. No entanto, não significa necessariamente saber qual biblioteca de classe vá usar. Obviamente, você precisa encontrar uma maneira de indicar ao Silverlight que namespace usar.

```
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>
```

O atributo xmlns é um atributo especializados no mundo do XML e é reservado para a declaração de namespaces.

XAML

Core Silverlight Namespaces

Os dois primeiros namespaces são os mais importantes. Você vai precisar deles para acessar partes assenciais do runtimes do Silverlight:

•http://schemas.microsoft.com/winfx/2006/xaml/presentation é o núcleo do silverlight. Ele engloba todas as classes Silverlight essenciais, incluídos o UserControl e Grid. Este namespace é declarado sem um prefixo de namespace, por isso torna-se o namespace DEFAULT para todo o documento.
•xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml. Ele inclui várias características de utilidade XAML que permitem influenciar a forma como o documento é interpretada.

XAML

As informações de namespace permite que o analisador de XAML encontre a classe correta. Por exemplo, quando ele encontra os elementos UserControl e Grid, ele procura-os no namespace padrão. Em seguida, ele procura o namespaces Silverlight correspondente, até encontrar as classes correspondentes **System.Windows.UserControl e System.Windows.Controls.Grid**.

XAML

Design Namespace

Junto com estes namespaces temos mais dois namespaces especializados:

- http://schemas.openxmlformats.org/markup-compatibility/2006 Você pode usá-lo para dizer ao analisador de XAML, que informações devem processar e as informações a serem ignoradas.
- http://schemas.microsoft.com/expression/blend/2008 é o namespace reservado especificamente para características de design suportadas pelo Expression Blend e Visual Studio 2010.

XAML

Custom Namespace

Em muitas situações, você irá desejar ter acesso aos seus próprios namespaces. O exemplo mais comum é se você deseja usar controle silverlight que você criou. Neste caso, você precisa definir o novo XML namespace prefix e mapeá-lo para seu assembly. Aqui temos a sintaxe necessária:

```
<UserControl x:Class="SilverlightCH12.MainPage"
   xmlns:w="clr-namespace:Widgets;assembly=WidgetsLibrary"
...>
```

XAML

A declaração de namespace XML define três conjuntos de informações:

•**O prefixo XML namespace**: você usará o prefixo para referenciar o namespace nas páginas XAML. No exemplo, o prefixo utilizado foi “w”.
•**O .NET namespace**: neste caso, as classes estão localizadas no Widgets namespaces.
•**O assembly**: neste caso, as classes são partes do assembly WidgetsLibrary.

Se você deseja usar um controle que está localizado na aplicação corrente, então, você pode omitir o assembly, como você pode vê a seguir:

```
xmlns:w=“clr-namespace:Widgets“
```

XAML

Uma vez que você tenha mapeado o seu Namespace .NET para um namespace XML, você pode usá-lo em qualquer lugar em seu documento XAML. Por exemplo, se o namespace Widgets contém um controle denominado HotButton, você poderia criar um exemplo como este:

```
<w:HotButton Text="Click Me!" Click="DoSomething"></w:HotButton>
```

XAML

The Code-Behind Class

Normalmente, cada arquivo XAML terá um correspondente code-behind do lado do cliente com o código C#. O Visual Studio cria uma classe code-behind para o arquivo nomeado MainPage.xaml MainPage.xaml.cs.

XAML

Aqui está o que você vê na MainPage.xaml.cs arquivo:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Animation;
using System.Windows.Shapes;

namespace SilverlightCH12
{
    public partial class MainPage : UserControl
    {
        public MainPage()
        {
            InitializeComponent();
        }
    }
}
```

XAML

Nomeando Elementos (Naming Elements)

Há mais um detalhe a considerar. Em sua classe code-behind, você muitas vezes, quer manipular os elementos programaticamente. Por exemplo, você pode querer ler ou alterar as propriedades ou ligar e desligar os manipuladores de eventos em tempo real. Para tornar isso possível, o controle deve incluir um atributo de nome XAML. No exemplo anterior, o controle Grid já inclui o atributo **Name**, então você pode manipulá-lo em seu arquivo code-behind.

```
<Grid x:Name="LayoutRoot“ >
 </Grid>
```

O atributo Name informa ao analisador XAML para adicionar um campo como este para a parte gerada automaticamente da classe Principal:

```
private System.Windows.Controls.Grid LayoutRoot;
```

Agora você pode interagir com o grid em seu código da classe página usando o nome LayoutRoot.

XAML

Propriedades e Eventos em XAML (Properties and Events in XAML)

A Figura 2-1 a seguir, mostra um exemplo com vários elementos:

```
<UserControl x:Class="SilverlightCH12.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="grid1">
      <Grid.Background>

      </Grid.Background>
      <Grid.RowDefinitions>

      </Grid.RowDefinitions>
      <TextBox x:Name="txtQuestion"></TextBox>
      <Button x:Name="cmdAswerer"></Button>
      <TextBox x:Name="txtAswerer"></TextBox>
   </Grid>
</UserControl>
```

XAML

Propriedades simples (Simple Properties)

Como você já viu, os atributos de um elemento XML define as propriedades do objeto Silverlight correspondente. Por exemplo, vamos configurar o alinhamento, a margem e a fonte das caixas de texto no exemplo.

```
<TextBox x:Name="txtQuestion" VerticalAlignment="Stretch"
                HorizontalAlignment="Stretch"
               FontFamily="Verdana" FontSize="24" Foreground="Green" ></TextBox>
```

```
<TextBox x:Name="txtAswerer" VerticalAlignment="Stretch"
                HorizontalAlignment="Stretch"
               FontFamily="Verdana" FontSize="24" Foreground="Green"></TextBox>
```

XAML

Propriedades complexas (Complex Properties)

XAML oferece uma outra opção: a sintaxe da propriedade elemento. Com a sintaxe da propriedade elemento, você adiciona um elemento filho com um nome na forma Parent.Propertyname. Por exemplo, o Grid tem uma propriedade Background, que permite que você pinte a área por trás dos elementos. Entretanto, Se você precisar de algo mais avançada do que um preenchimento de cor sólida - você vai precisar adicionar uma etiqueta filho chamada Grid.Background, conforme mostrado aqui:

```
<Grid x:Name="grid1" Background="White">

</Grid>
```

```
<Grid.Background>

</Grid.Background>
```

XAML

Propriedades complexas (Complex Properties)

Usando a regra do XAML, você pode criar o LinearGradientBrush objeto usando um elemento com o nome LinearGradientBrush:

```
<Grid.Background>
        <LinearGradientBrush ></LinearGradientBrush>
 </Grid.Background>
```

XAML

Propriedades complexas (Complex Properties)

No entanto, não basta simplesmente criar o LinearGradientBrush - você também precisará especificar as cores do Gradiente. Você pode fazer isso preenchendo a propriedade LinearGradientBrush.GradientStops com uma coleção de objetos GradientStop. No entanto, a propriedade GradientStops é demasiado complexa para ser definido com um valor de atributo sozinho. Em vez disso, você precisará contar com a sintaxe da propriedade elemento:

```
<Grid.Background>
        <LinearGradientBrush >
          <LinearGradientBrush.GradientStops>

          </LinearGradientBrush.GradientStops>
        </LinearGradientBrush>
</Grid.Background>
```

XAML

Propriedades complexas (Complex Properties)

Finalmente, você pode preencher a coleção GradientStops com uma série de objetos GradientStop. Cada objeto tem um GradientStop Offset e propriedade Color. você pode fornecer estes dois valores usando a sintaxe de atributo de propriedade comum:

```
<Grid.Background>
        <LinearGradientBrush >
          <LinearGradientBrush.GradientStops>
            <GradientStop Offset="0.00" Color="Yellow" />
            <GradientStop Offset="0.50" Color="White" />
            <GradientStop Offset="1.00" Color="Purple" />
          </LinearGradientBrush.GradientStops>
        </LinearGradientBrush>
</Grid.Background>
```

XAML

Fazendo tudo no código:

```
LinearGradientBrush brush = new LinearGradientBrush();
```

```
GradientStop gradientStop1 = new GradientStop();
gradientStop1.Offset = 0;
gradientStop1.Color = Colors.Yellow;
Brush.GradientStops.Add(gradientStop1);
```

```
GradientStop gradientStop2 = new GradientStop();
gradientStop2.Offset = 0.5;
gradientStop2.Color = Colors.White;
Brush.GradientStops.Add(gradientStop2);
```

```
GradientStop gradientStop3 = new GradientStop();
gradientStop3.Offset = 1;
gradientStop3.Color = Colors.Purple;
Brush.GradientStops.Add(gradientStop3);
```

```
grid1.Backgroung = brush;
```

XAML

Falta definir as linhas do Grid:

```
<Grid.RowDefinitions>
        <RowDefinition Height="*"/>
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="*"/>
</Grid.RowDefinitions>
```

XAML

Propriedades anexadas (Attached Properties)

Propriedades anexadas sempre usa um nome de duas partes da seguinte forma: DefiningType.PropertyName. Essa sintaxe de nomeação de duas partes permite que o analisador de XAML distinguir entre uma propriedade normal e uma propriedade anexada.

No nosso exemplo, as propriedades anexadas permitir que os elementos individuais sejam colocados em linhas separadas no grid.

```
<TextBox ... Grid.Row = "0"></TextBox>
```

```
<Button ... Grid.Row = "1"></Button>
```

```
<TextBox ... Grid.Row = "2"></TextBox>
```

XAML

Como fica nosso exemplo:

```
<Grid x:Name="grid1">
      <Grid.RowDefinitions>
         <RowDefinition Height="*"/>
         <RowDefinition Height="Auto"/>
         <RowDefinition Height="*"/>
</Grid.RowDefinitions>
```

```
<TextBox x:Name="txtQuestion" VerticalAlignment="Stretch“
              HorizontalAlignment="Stretch"  Margin="10,10,13,10“  FontFamily="Verdana"
              FontSize="24" Foreground="Green" Grid.Row="0"></TextBox>
<Button x:Name="cmdAswerer" VerticalAlignment="Top" HorizontalAlignment="Left"
            Margin="10,0,0,20" Width="127" Height="23" Click="cmdAswerer_Click"    Grid.Row="1"
Content="Ask the Eight Ball"></Button>
<TextBox x:Name="txtAswerer" VerticalAlignment="Stretch" HorizontalAlignment="Stretch"
             Margin="10,10,13,10" TextWrapping="Wrap" IsReadOnly="True"
              Text="[Answer will appear here.]“ FontFamily="Verdana" FontSize="24"
              Foreground="Green" Grid.Row="2" ></TextBox>
```

XAML

```
<Grid.Background>
        <LinearGradientBrush >
           <LinearGradientBrush.GradientStops>
              <GradientStop Offset="0.00" Color="Yellow" />
              <GradientStop Offset="0.50" Color="White" />
              <GradientStop Offset="1.00" Color="Purple" />
           </LinearGradientBrush.GradientStops>
        </LinearGradientBrush>
</Grid.Background>
```

XAML

Eventos (Events)

Até agora, todos os atributos que você viu, foi o mapa de propriedades. No entanto, atributos também podem ser utilizados para ligar manipuladores de eventos. A sintaxe para isso é eventName = "EventHandlerMethodname". Por exemplo, o controle Button prevê um evento de clique. Você pode anexar um manipulador de eventos como este:

```
<Button ... Click="cmdAnswer_Click">
```

Isso pressupõe que há um método com o cmdAnswer_Click nome na classe code-behind. O manipulador de eventos deve ter a assinatura correta. Aqui está o método que faz o truque:

```
private void cmdAswerer_Click(object sender, RoutedEventArgs e)
        {
            txtAswerer.Text = "Qualquer coisa.";
        }
```

XAML

XAML Recursos (XAML Resources)

Silverlight inclui um sistema de recursos que integra com XAML.Usando os recursos, você pode:

•**Criar objetos não visuais**: Isto é útil se outros elementos usar esses objetos. Por exemplo, você pode criar um objeto de dados como um recurso e, em seguida, usar a ligação de dados para exibir suas informações em vários elementos.

•**Reutilização de objetos**: Depois de definir um recurso, vários elementos podem recorrer a ela. Por exemplo, você pode definir um brush com várias cores.

•**Centralizar informações**: Às vezes, é mais fácil extrair informações freqüentemente alterados em um lugar em vez de dispersá-la através de um arquivo de marcação complexa, onde é mais difícil de rastrear as mudanças.

XAML

Coleção de Recursos

Cada elemento inclui uma property Resources, que armazena uma coleção de recursos. Embora, cada elemento tenha uma property Resources, a forma mais comum de definir recursos é a nível de página. Porque, se você definir uma recurso a nível de página todos os elementos podem usá-lo. Por exemplo, no exemplo anterior, colocamos o brush no Grid, mas poderíamos tê-lo colocado no UserControl, da seguinte forma:

```
<UserControl.Resources>
  <LinearGradientBrush x:key="BackgroundBrush">
     <LinearGradientBrush.GradientStops>
        <GradientStop Offset="0.00" Color="Yellow" />
        <GradientStop Offset="0.50" Color="White" />
        <GradientStop Offset="1.00" Color="Purple" />
     </LinearGradientBrush.GradientStops>
  </LinearGradientBrush>
  ...
</UserControl.Resources>
```

XAML

Para usar um recurso em seu XAML, você necessita de uma forma de referi-se a ele. Isso é feito usando uma extensão de marcação - um tipo especializado de sintaxe que define uma propriedade em uma maneira fora do padrão. Extensão de marcação estender a linguagem XAML e podem ser reconhecidos por suas chaves: Para usar um recurso, você pode usar uma extensão chamada StaticResource:

```
<Grid x:Name="grid1" Background="{StaticResource BackgroungBrush}">
```

XAML

A hierarquia de recursos

Cada elemento tem sua coleção de recursos próprios, e o Silverlight executa uma pesquisa recursiva a sua árvore de elemento para localizar o recurso que você deseja. Por exemplo, imagine que você tenha a seguinte marcação.

XAML

```
<UserControl x:Class="SilverlightApplication2ch2.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">
      <StackPanel>
         <StackPanel.Resources>
            <LinearGradientBrush x:Key="ButtonFace">
               <GradientStop Offset="0.00" Color="Yellow"/>
               <GradientStop Offset="0.50" Color="White" />
               <GradientStop Offset="1.00" Color="Purple" />
            </LinearGradientBrush>
         </StackPanel.Resources>
         <Button Content="Click Me First" Margin="5" Background="{StaticResource
ButtonFace}"></Button>
         <Button Content="Click Me Next" Margin="5" Background="{StaticResource
ButtonFace}"></Button>
      </StackPanel>
   </Grid>
</UserControl>
```

XAML

No exemplo anterior, os botões configuraram seus backgrounds para o mesmo recurso. Quando o Silverlight encontrar esta solicitação(do recurso), então faz uma busca do recurso no botão, no StackPanel, se não encontrar, ele continua a procura pelo Grid e finalmente no UserControl.

Você ainda pode colocar o recurso na aplicação, usando a seguinte sintaxe:

```
<Application.Resources>
  ...
</Application.Resources>
```

A vantagem de colocar recursos na aplicação é que eles são completamente removidos da marcação em sua página, e eles podem ser reaproveitados através de um aplicativo inteiro. Neste exemplo a escolha, é interessante, se você planeja usar o brush em mais de uma página.

XAML

Acessando Recursos no Código

Normalmente, você define e utiliza os recursos através de marcação. No entanto, se surgir a necessidade, você pode trabalhar com os recursos através do código. Por exemplo, se você armazenar um LinearGradientBrush na seção <UserControl.Resources> com a chave nome ButtonFace, você pode usar código como este:

```
LinearGradientBrush brush = (LinearGradientBrush)this.Resources["ButtonFace"];

// Trocar a ordem da cor
Color color = brush.GradientStops[0].Color;
brush.GradientStops[0].Color = brush.GradientStops[2].Color;
brush.GradientStops[2].Color = color;
```

XAML

Organizando Recursos com Resource Dictionaries

Se você deseja compartilhar recursos entre múltiplos projetos, você pode criar um Resources dictionay. Um dicionário de recursos é apenas um documento XAML, que nada mais faz do que armazenar um conjunto de recursos.

```
<ResourceDictionary
   xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
   <LinearGradientBrush x:Key="ButtonFace">
      <GradientStop Offset="0.00" Color="Yellow" />
      <GradientStop Offset="0.50" Color="White" />
      <GradientStop Offset="1.00" Color="Purple" />
   </LinearGradientBrush>
</ResourceDictionary>
```

XAML

Ligação entre Elementos (Element-to-Element Binding)

Na seção anterior, você viu como usar a extensão de marcação StaticResource, que dá recursos adicionáis XAML. Outra extensão de marcação é a expressão de vinculação (Binding), que estabelece uma relação entre as informações de objeto de origem para um controle de destino.

XAML

One-Way Binding

Para entender como você pode vincular um elemento para outro elemento, considere as janelas simples mostrado na Figura 2-4. Ele contém dois controles: um controle Slider e um TextBlock com uma única linha de texto. Se você puxar o marcador deslizante do controle para a direita, o tamanho da letra do texto é aumentado imediatamente. Se você puxar para a esquerda, o tamanho da fonte é reduzida.

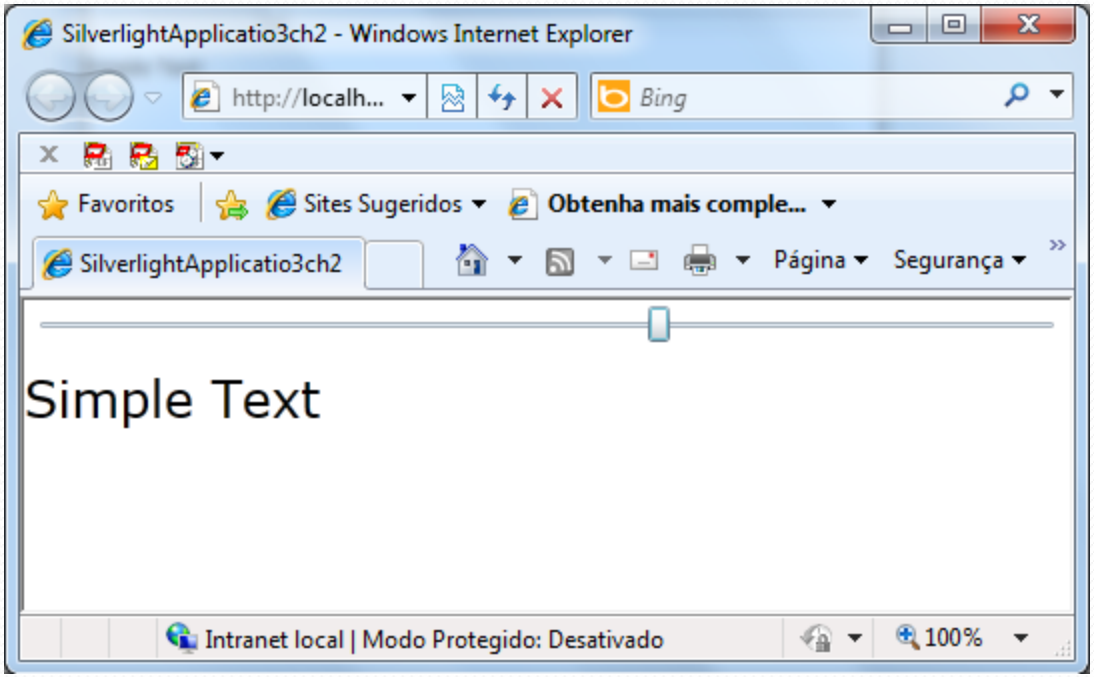

Figura 2-4 Ligando controle através de Data Binding

XAML

Veja o código XAML do exemplo anterior

```
<UserControl x:Class="SilverlightApplicatio3ch2.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">
      <StackPanel>
      <Slider x:Name="sliderFontSize" Margin="3" Minimum="1" Maximum="40"
Value="10"></Slider>
      <TextBlock Margin="0,8,20,12" Text="Simple Text" x:Name="lblSampleText"
            FontSize="{Binding ElementName=sliderFontSize, Path=Value}"></TextBlock>
      </StackPanel>
   </Grid>
</UserControl>
```

XAML

Two-Way Binding

Curiosamente há um caminho, para forçar os valores de fluxo em ambos os sentidos: desde a origem para o destino e do destino para a origem. O truque é definir a propriedade Mode do Binding.