Module 9

Designing the User Interface for a Graphical Application

Module Overview

- Using XAML to Design a User Interface
- Binding Controls to Data
- Styling a UI

Lesson 1: Using XAML to Design a User Interface

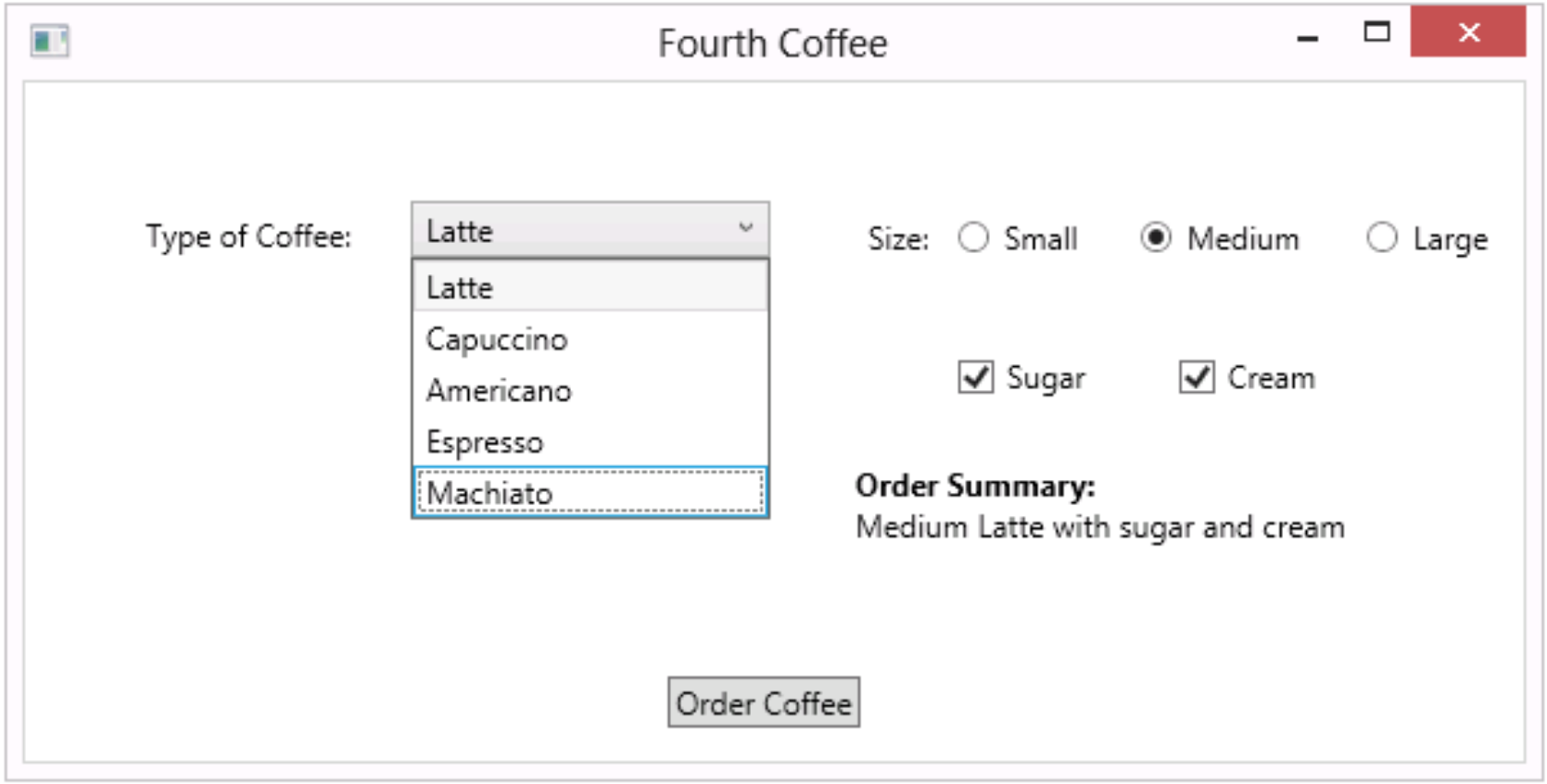
- Introducing XAML
- Common Controls
- Setting Control Properties
- Handling Events
- Using Layout Controls
- Demonstration: Using Design View to Create a XAML UI
- Creating User Controls

Introducing XAML

- Use XML elements to create controls
- Use attributes to set control properties
- Create hierarchies to represent parent controls and child controls

Common Controls

- Button
- Checkbox
- ComboBox
- Label
- ListBox
- RadioButton
- TabControl
- TextBlock
- TextBox



Setting Control Properties

- Use attribute syntax to define simple property values

```
<Button Content=“Click Me" Background="Yellow" />
```

- Use property element syntax to define complex property values

```
<Button Content=“Click Me"
  <Button.Background>
    <LinearGradientBrush StartPoint="0.5, 0.5"
                         EndPoint="1.5, 1.5">
      <GradientStop Color="AliceBlue" Offset="0" />
      <GradientStop Color="Aqua" Offset="0.5" />
    </LinearGradientBrush>
  </Button.Background>
</Button>
```

Handling Events

- Specify the event handler method in XAML

```
<Button x:Name="btnMakeCoffee"
        Content="Make Me a Coffee!"
        Click="btnMakeCoffee_Click" />
```

- Handle the event in the code-behind class

```
private void btnMakeCoffee_Click(object sender,
   RoutedEventArgs e)
{
   lblResult.Content = "Your coffee is on its way.";
}
```

- Events are bubbled to parent controls

Using Layout Controls

- Canvas
- DockPanel
- Grid
- StackPanel
- VirtualizingStackPanel
- WrapPanel

Demonstration: Using Design View to Create a XAML UI

In this demonstration, you will learn how to:

- Add controls to the design surface in Visual Studio
- Edit controls by using designer tools
- Edit controls by editing XAML directly
- Use Visual Studio tools to create event handlers

Creating User Controls

To create a user control:

- Define the control in XAML
- Expose properties and events in the code-behind class

To use a user control:

- Add an XML namespace prefix for the assembly and namespace
- Use the control like a standard XAML control

Lesson 2: Binding Controls to Data

- Intoduction to Data Binding
- Binding Controls to Data in XAML
- Binding Controls to Data in Code
- Binding Controls to Collections
- Creating Data Templates

Introduction to Data Binding

- Data binding has three components:
  - Binding source
  - Binding target
  - Binding object
- A data binding can be bidirectional or unidirectional:
  - TwoWay
  - OneWay
  - OneTime
  - OneWayToSource
  - Default

Binding Controls to Data in XAML

- Use a binding expression to identify the source object and the source property

```
<TextBlock
  Text="{Binding Source={StaticResource coffee1},
    Path=Bean}" />
```

- Specify the data context on a parent control

```
<StackPanel>
  <StackPanel.DataContext>
    <Binding Source="{StaticResource coffee1}" />
  </StackPanel.DataContext>
  <TextBlock Text="{Binding Path=Name}" />
  …
</StackPanel>
```

Binding Controls to Data in Code

- Create data binding entirely in code
- Create **Path** bindings in XAML and set the **DataContext** in code

```
<StackPanel x:Name="stackCoffee">
  <TextBlock Text="{Binding Path=Name}" />
  <TextBlock Text="{Binding Path=Bean}" />
  <TextBlock Text="{Binding Path=CountryOfOrigin}" />
  <TextBlock Text="{Binding Path=Strength}" />
</StackPanel>
```

```
stackCoffee.DataContext = coffee1;
```

Binding Controls to Collections

- Set the **ItemsSource** property to bind to an **IEnumerable** collection

```
lstCoffees.ItemsSource = coffees;
```

- Use the **DisplayMemberPath** property to specify the source field to display

```
<ListBox x:Name="lstCoffees"
         DisplayMemberPath="Name" />
```

Creating Data Templates

Specify how each item in a collection should be displayed

```
<DataTemplate>
  <Grid>
  …
    <TextBlock Text="{Binding Path=Name}" Grid.Row="0"
               FontSize="22" Background="Black"
               Foreground="White" />
    <TextBlock Text="{Binding Path=Bean}"
               Grid.Row="1" />
    <TextBlock Text="{Binding Path=CountryOfOrigin}"
               Grid.Row="2" />
    <TextBlock Text="{Binding Path=Strength}"
               Grid.Row="3" />
  </Grid>
</DataTemplate>
```

Lesson 3: Styling a UI

- Creating Reusable Resources in XAML
- Defining Styles as Resources
- Using Property Triggers
- Creating Dynamic Transformations
- Demonstration: Customizing Student Photographs and Styling the Application Lab

Creating Reusable Resources in XAML

itucation

- Define resources in a **Resources** collection
- Add an **x:Key** to uniquely identify the resource

```
<Window.Resources>
  <SolidColorBrush x:Key="MyBrush" Color="Coral" />
  …
</Window.Resources>
```

- Reference the resource in property values

```
<TextBlock Text="Foreground"
    Foreground="{StaticResource MyBrush}" />
```

- Use a resource dictionary to manage large collections of resources

Defining Styles as Resources

- Identify the target control type
- Provide an **x:Key** value if required
- Use **Setter** elements to specify property values

```
<Style TargetType="TextBlock" x:Key="BlockStyle1">
   <Setter Property="FontSize" Value="20" />
   <Setter Property="Background" Value="Black" />
   …
</Style>
```

- Reference the style as a static resource

```
<TextBlock Text="Drink More Coffee"
    Style="{StaticResource BlockStyle1}" />
```

Using Property Triggers

Use triggers to apply style properties based on conditions:

- Use the **Trigger** element to identify the condition
- Use **Setter** elements apply the conditional changes

```
<Style TargetType="Button">
  <Style.Triggers>
    <Trigger Property="IsMouseOver" Value="True">
      <Setter Property="FontWeight" Value="Bold" />
    </Trigger>
  </Style.Triggers>
</Style>
```

Creating Dynamic Transformations

- Use an **EventTrigger** to identify the event that starts the animation
- Use a **Storyboard** to identify the properties that should change
- Use a **DoubleAnimation** to define the changes

```
<EventTrigger RoutedEvent="Image.MouseDown">
  <BeginStoryboard>
    <Storyboard>
      <DoubleAnimation
          Storyboard.TargetProperty="Height"
          From="200" To="300" Duration="0:0:2" />
    </Storyboard>
  </BeginStoryboard>
</EventTrigger>
```

Demonstration: Customizing Student Photographs and Styling the Application Lab

In this demonstration, you will learn about the tasks that you will perform in the lab for this module.

Lab: Customizing Student Photographs and Styling the Application

- Exercise 1: Customizing the Appearance of Student Photographs
- Exercise 2: Styling the Logon View
- Exercise 3: Animating the StudentPhoto Control (If Time Permits)

Logon Information

- Virtual Machine: 20483B-SEA-DEV11, MSL-TMG1
- User Name: Student
- Password: Pa$$w0rd

Estimated Time: 90 minutes

Lab Scenario

- Now that you and The School of Fine Arts are happy with the basic functionality of the application, you need to improve the appearance of the interface to give the user a nicer experience through the use of animations and a consistent look and feel.
- You decide to create a **StudentPhoto** control that will enable you to display photographs of students in the student list and other views. You also decide to create a fluid method for a teacher to remove a student from their class. Finally, you want to update the look of the various views, keeping their look consistent across the application.

Module Review and Takeaways

- Review Question(s)