Event Driven Programming

Elements of Event Driven Programming

Chalew Tesfaye
Department of Computer Science
Debre Berhan University
2017

Objectives

- At the end of the lesson students should be able to:
  - ✓ Understand events and delegates
  - ✓ Create an event handler for a control
  - ✓ Work with windows controls
  - ✓ Work with properties and events of controls
  - ✓ Design and build windows based application

Contents

Working with Controls

Building blocks of GUI based application

MenuStrip Control

- Displays application commands and options grouped by functionality.
- **Creating a MenuStrip**
  - ✓ We can create a MenuStrip control using a Forms designer at design-time or using the MenuStrip class in code at run-time or dynamically.
  - ✓ To create a MenuStrip control at design-time, you simply drag and drop a MenuStrip control from Toolbox to a Form in Visual Studio.
  - ✓ Once a MenuStrip is on the Form, you can add menu items and set its properties and events.

…

- Creating a MenuStrip control at **run-time** is merely a work of creating an instance of MenuStrip class, set its properties and adds MenuStrip class to the Form controls.
- First step to create a dynamic MenuStrip is to create an instance of MenuStrip class.

```
MenuStrip MainMenu = new MenuStrip();
```

- In the next step, you may set properties of a MenuStrip control.

```
 MainMenu.BackColor = Color.OrangeRed;
MainMenu.ForeColor = Color.Black;
MainMenu.Text = "File Menu";
MainMenu.Font = new Font ("Georgia", 16);
```

- the next step is to add the MenuStrip to a Form.

```
this.MainMenuStrip = MainMenu;
Controls.Add(MainMenu);
```

Setting MenuStrip Properties

- **Name**

  ```
  MainMenu.Name = "MainMenu";
  ```

- **Positioning a MenuStrip**
  - ✓ The Dock property is used to set the position of a MenuStrip. It is of type DockStyle that can have values Top, Bottom, Left, Right, and Fill. The following code snippet sets Location, Width, and Height properties of a MenuStrip control.

    ```
    MainMenu.Dock = DockStyle.Left;
    ```

- **Font**

  ```
  MainMenu.Font = new Font ("Georgia", 16);
  ```

- **Background and Foreground**

  ```
  MainMenu.BackColor = Color.OrangeRed;
  MainMenu.ForeColor = Color.Black;
  ```

- **MenuStrip Items**
  - ✓ A Menu control is nothing without menu items. The Items property is used to add and work with items in a MenuStrip.
  - ✓ We can add items to a MenuStrip at **design-time** from Properties Window by clicking on Items Collection

…

- A ToolStripMenuItem represents a menu items.

```
// Create a Menu Item
ToolStripMenuItem FileMenu = new ToolStripMenuItem ("File");
FileMenu.BackColor = Color.OrangeRed;
FileMenu.ForeColor = Color.Black;
FileMenu.Text = "File Menu";
FileMenu.Font = new Font ( "Georgia", 16);
FileMenu.TextAlign = ContentAlignment.BottomRight;
FileMenu.ToolTipText = "Click Me";
//add it to Forms Controls
MainMenu.Items.Add (FileMenu);
```

- **Adding Menu Item Click Event Handler**

```
FileMenu.Click += new System.EventHandler (this.FileMenuItemClick);
private void FileMenuItemClick (object sender, EventArgs e)
{
   MessageBox.Show("File menu item clicked");
}
```

StatusBar Control

- A StatusBar control is a combination of StatusBar panels where each panel can be used to display different information.
- For example, one panel can display current application status and other can display date and other information and so on. A typical StatusBar sits at the bottom of a form.
- **Creating a StatusBar**

```
protected StatusBar mainStatusBar = new StatusBar();
protected StatusBarPanel statusPanel = new StatusBarPanel();
protected StatusBarPanel datetimePanel = new StatusBarPanel();
```

…

```
private void CreateStatusBar()
{  // Set first panel properties and add to StatusBar
   statusPanel.BorderStyle = StatusBarPanelBorderStyle.Sunken;
   statusPanel.Text = "Application started. No action yet.";
   statusPanel.ToolTipText = "Last Activity";
   statusPanel.AutoSize = StatusBarPanelAutoSize.Spring;
   mainStatusBar.Panels.Add (statusPanel);
   // Set second panel properties and add to StatusBar
   datetimePanel.BorderStyle = StatusBarPanelBorderStyle.Raised;
   datetimePanel.ToolTipText = "DateTime: " + System.DateTime.Today.ToString();
   datetimePanel.Text = System.DateTime.Today.ToLongDateString();
   datetimePanel.AutoSize = StatusBarPanelAutoSize.Contents;
   mainStatusBar.Panels.Add (datetimePanel);
   mainStatusBar.ShowPanels = true;
   this.Controls.Add(mainStatusBar);   // Add StatusBar to Form controls
}
```

…

```
private void button1_Click (object sender, EventArgs e)
{
    statusPanel.Text = "Button is clicked.";
}
private void checkBox1_CheckedChanged (object sender, EventArgs e)
{
    statusPanel.Text = "CheckBox is checked.";
}
private void textBox1_TextChanged (object sender, EventArgs e)
{
    statusPanel.Text = "TextBox edited.";
}
```

Dialog Controls

Accessing OS Resource

ColorDialog Control

- A ColorDialog control is used to select a color from available colors and also define custom colors.
- **Creating a ColorDialog**
  - ✓ We can create a ColorDialog control using a Forms designer at design-time or using the ColorDialog class in code at run-time (also known as dynamically).
  - ✓ Unlike other Windows Forms controls, a ColorDialog does not have and not need visual properties like others.
  - ✓ The only purpose of ColorDialog to display available colors, create custom colors and select a color from these colors. Once a color is selected, we need that color in our code so we can apply it on other controls.
  - ✓ Again, you can create a ColorDialog at design-time but It is easier to create a ColorDialog at run-time.
- **Design-time**
  - ✓ To create a ColorDialog control at design-time, you simply drag and drop a ColorDialog control from Toolbox to a Form in Visual Studio
  - ✓ Adding a ColorDialog to a Form adds following two lines of code.

```
private System.Windows.Forms.ColorDialog colorDialog1;
this.colorDialog1 = new System.Windows.Forms.ColorDialog();
```

…

- **Run-time**
  - ✓ First step to create a dynamic ColorDialog is to create an instance of ColorDialog class.

    ```
    ColorDialog colorDlg = new ColorDialog();
    ```
  - ✓ ShowDialog method of ColorDialog displays the ColorDialog.

    ```
    colorDlg.ShowDialog();
    ```
  - ✓ Once the ShowDialog method is called, you can pick colors on the dialog.
- **Setting ColorDialog Properties**
  - ✓ **AllowFullOpen**
    - **AllowFullOpen** property makes sure that **Define Custom Color** option is enabled on a ColorDialog.
    - If you wish to disable this option, you can set **AllowFullOpen** property to false
    - The following code snippet sets the AllowFullOpen property to false.

      ```
      colorDlg.AllowFullOpen = false;
      ```

Color, AnyColor, and SolidColorOnly

- ✓ Color property is used to get and set the color selected by the user in a ColorDialog.
- ✓ AnyColor is used to get and set whether a ColorDialog displays all available colors in the set of basic colors.
- ✓ SolidColorOnly is used to get and set whether a ColorDialog restricts users to selecting solid colors only.
- ✓ The following code snippet sets these properties.

```
colorDlg.AnyColor = true;
colorDlg.SolidColorOnly = false;
colorDlg.Color = Color.Red;
```

…

- **Using ColorDialog in Applications**
- Now let's create an application that will use a ColorDialog to set colors of bunch of controls.
- The following code snippet is the code for Foreground Color and Background Color buttons click event handlers.

```
private void ForegroundButton_Click (object sender, EventArgs e)
{
    ColorDialog colorDlg = new ColorDialog();
    colorDlg.AllowFullOpen = false;
    colorDlg.AnyColor = true;
    colorDlg.SolidColorOnly = false;
    colorDlg.Color = Color.Red;
    if (colorDlg.ShowDialog() == DialogResult.OK)
    {
        textBox1.ForeColor = colorDlg.Color;
        listBox1.ForeColor = colorDlg.Color;
        button3.ForeColor = colorDlg.Color;
    }
}
```

…

```
private void BackgroundButton_Click (object sender, EventArgs e)
{
    ColorDialog colorDlg = new ColorDialog();
    if (colorDlg.ShowDialog() == DialogResult.OK)
    {
        textBox1.BackColor = colorDlg.Color;
        listBox1.BackColor = colorDlg.Color;
        button3.BackColor = colorDlg.Color;
    }
}
```

FontDialog Control

- **Creating a FontDialog**
  - ✓ We can create a FontDialog control using a Forms designer at design-time or using the FontDialog class in code at run-time (also known as dynamically).
  - ✓ Unlike other Windows Forms controls, a FontDialog does not have and not need visual properties like others.
  - ✓ You use a FontDialog to list all the fonts and select one of them and usually apply selected fonts on controls or some contents.
- **Design-time**
  - ✓ To create a FontDialog control at design-time, you simply drag and drop a FontDialog control from Toolbox to a Form in Visual Studio. Adding a FontDialog to a Form adds following two lines of code.

```
private System.Windows.Forms.FontDialog fontDialog1;
this.fontDialog1 = new System.Windows.Forms.FontDialog ();
```

…

- **Run-time**
  - ✓ Creating a FontDialog control at run-time is merely a work of creating an instance of FontDialog class, set its properties and add FontDialog class to the Form controls.
  - ✓ First step to create a dynamic FontDialog is to create an instance of FontDialog class.

    ```
    FontDialog fontDlg = new FontDialog();
    ```
  - ✓ ShowDialog method of FontDialog displays the FontDialog.

    ```
    fontDlg.ShowDialog();
    ```

…

FontDialog Properties

✓ Show Properties

- A FontDialog control can have a color drop down that allows users to select a color of the selected font.
- A FontDialog can also have Apply and Help buttons ShowColor, ShowApply, and ShowHelp properties can be set to true to show these options. The ShowEffects property represents whether the dialog box contains controls that allow the user to specify strikethrough, underline, and text color options.
- The following code snippet sets these properties to true.

```
fontDlg.ShowColor = true;
fontDlg.ShowApply = true;
fontDlg.ShowEffects = true;
fontDlg.ShowHelp = true;
```

✓ Maximum and Minimum Font Size

- We can restrict the maximum and minimum font size by setting MaxSize and MinSize properties. The following code snippet sets these properties to 40 and 22 respectively.

```
fontDlg.MaxSize = 40;
fontDlg.MinSize = 22;
```

…

- **Font and Color**
  - ✓ The Font and Color properties represent the selected font and color in a FontDialog. The following code snippet gets these properties and sets font and color of a TextBox and a Label controls.

```
if (fontDlg.ShowDialog() != DialogResult.Cancel)
{
    textBox1.Font = fontDlg.Font;
    label1.Font = fontDlg.Font;
    textBox1.BackColor = fontDlg.Color;
    label1.ForeColor = fontDlg.Color;
}
```

OpenFileDialog Control

- The OpenFileDialog object interacts with the Computer API (Application Programming Interface) to present available files to the user and retrieves the user file selection back to the program.
- **Opening and Reading a Text File using the OpenFileDialog form in C#.NET**
  - ✓ This object is part of the System.Windows.Forms library so no additional using reference will be needed.
  - ✓ This dialog can be customized to show the file type, beginning directory, and the title to be displayed on the dialog itself.
  - ✓ When you limit the file type to just the extension .txt as I did in the following sample code, only those particular file types will be visible to the user although they do have the option to select All files (*) as well.

…

```
private void btnOpenTextFile_Click (object sender, EventArgs e) {
        String input = string.Empty; //First, declare a variable to hold the user's file selection.
          ///Because the OpenFileDialog is an object, we create a new instance by declaring a
          //variable with the data type OpenFileDialog and setting it equal to the new instance.
        OpenFileDialog dialog = new OpenFileDialog();
          //Now we set the file type we want to be available to the user.  In this case, text files.
        dialog.Filter =   "txt files (*.txt)|*.txt|All files (*.*)|*.*";
          dialog.InitialDirectory = "C:"; // Next is the starting directory for the dialog
        dialog.Title = "Select a text file";  // and the title for the dialog box are set.
          //Once the dialog properties are set, it is ready to present to the user.
         if (dialog.ShowDialog() == DialogResult.OK)
             strFileName =  dialog.FileName;
         if (strFileName == String.Empty)
             return; //user didn't select a file to open
 }
```

If the user selected a file, then the DialogResult property value will be "OK" and we will also have the file name and path of that file.

SaveFileDialog Control

- A SaveFileDialog control is used to save a file using Windows Save File Dialog.
- **Creating a SaveFileDialog**
  - ✓ **Design-time**
    - To create a SaveFileDialog control at design-time, you simply drag and drop a SaveFileDialog control from Toolbox to a Form in Visual Studio.
    - Adding a SaveFileDialog to a Form adds following two lines of code.

```
private SaveFileDialog saveFileDialog1;
this.saveFileDialog1 = new SaveFileDialog();
```

...

- ✓ **Run-time**
  - > Creating a SaveFileDialog control at run-time is merely a work of creating an instance of SaveFileDialog class, set its properties and add SaveFileDialog class to the Form controls.
  - > First step to create a dynamic SaveFileDialog is to create an instance of SaveFileDialog class.

```
SaveFileDialog SaveFileDialog1 = new SaveFileDialog();
```

  - > ShowDialog method displays the SaveFileDialog.

```
SaveFileDialog1.ShowDialog();
```

…

- **Setting SaveFileDialog Properties**

```
private void SaveButton_Click (object sender, EventArgs e)
{
   SaveFileDialog saveFileDialog1 = new SaveFileDialog();
   saveFileDialog1.InitialDirectory = @"C:\";
   saveFileDialog1.Title = "Save text Files";
   saveFileDialog1.CheckFileExists = true;
   saveFileDialog1.CheckPathExists = true;
   saveFileDialog1.DefaultExt = "txt";
   saveFileDialog1.Filter = "Text files (*.txt)|*.txt|All files (*.*)|*.*";
   saveFileDialog1.FilterIndex = 2;
   saveFileDialog1.RestoreDirectory = true;
   if (saveFileDialog1.ShowDialog() == DialogResult.OK)
   {
     textBox1.Text = saveFileDialog1.FileName;
   }
}
```

PrintDialog Control

- A PrintDialog control is used to open the Windows Print Dialog and let user select the printer, set printer and paper properties and print a file.
- **Creating a PrintDialog**
  - ✓ We can create a PrintDialog at design-time as well as at run-time.
  - ✓ **Design-time**
    - To create a PrintDialog control at design-time, you simply drag and drop a PrintDialog control from Toolbox to a Form in Visual Studio
  - ✓ **Run-time**
    - Creating a PrintDialog control at run-time is simple. First step is to create an instance of PrintDialog class and then call the ShowDialog method.
    - The following code snippet creates a PrintDialog control.

```
PrintDialog PrintDialog1 = new PrintDialog();
PrintDialog1.ShowDialog();
```

…

- **Printing Documents**
  - ✓ PrintDocument object represents a document to be printed.
  - ✓ Once a PrintDocument is created, we can set the Document property of PrintDialog as this document. After that we can also set other properties.
  - ✓ The following code snippet creates a PrintDialog and sends some text to a printer.

```
private void PrintButton_Click (object sender, EventArgs e)
{
    PrintDialog printDlg = new PrintDialog();
    PrintDocument printDoc = new PrintDocument();
    printDoc.DocumentName = "Print Document";
    printDlg.Document = printDoc;
    printDlg.AllowSelection = true;
    printDlg.AllowSomePages = true;
    //Call ShowDialog
    if (printDlg.ShowDialog() == DialogResult.OK)
        printDoc.Print();
}
```

More information on

- John Sharp. Microsoft Visual C# 2013 Step by Step, 2015 Microsoft Press USA
- Joel Murach, Anne Boehm. Murach C# 2012, Mike Murach & Associates Inc USA, 2013
- Microsoft Corporation. Microsoft Visual Studio 2012 Product Guide, 2012

QUESTIONS

?