Graphical User Interfaces

Objectives: At the end of the chapter students should be able to:
1) Explain the concept of event driven programming.
2) Describe the purpose of and use of Forms,
3) Describe the purpose of and use of graphical components (controls),
4) Understand the use of events and event handling,
5) Understand the use of Properties for controls

Contents of This Chapter
7.1 Some History
7.2 Event Driven Programming
7.3 Windows Forms Applications
7.4 Graphical User Interface Components (Controls)
7.5 Beyond the Basics
In this course you will write programs that use the Console for input and output, and you will write programs that use a graphical user interface (GUI). GUI programs use a different style of programming called event driven programming. In this chapter you will learn how Graphical User Interfaces work, and how they are constructed.

7.1 Some History

Console applications require all user input to be in the form of text from the keyboard and output to the display screen is also all text based. For many applications this can be a very tedious and time consuming process, and it severely limits the kinds of applications that can be written. If all applications were text based, we would have no image processing, no drawing programs, and no games with visual content. Operating system developers realized this restriction very early and began to experiment with other ways for programs to communicate with their users. One of the early and most successful efforts was the Graphical User Interface work for a system being developed by Xerox in their Palo Alto Research Center (PARC). The original system was named “MACS” and the next generation was named “Alto.” At Xerox PARC they developed such amazing technologies as laser printing, SmallTalk - an Object-Oriented Programming language (OOP), Ethernet, Superpaint frame buffer and CRT displays, WYSIWIG (What You See Is What You Get) interfaces, mouse technology and use, relational databases, solid-state lasers, graphical user interfaces (GUI), VLSI circuit design, Worm disk drives, natural language processing, optical storage, fiber optics, collaborative workspaces and tools, encryption systems, Internet standards, and digital rights management, just to mention a few.

A very interesting outcome of this work was that a small startup company in the bay area heard of the great things being explored at Xerox PARC and made arrangements to visit the research facility. This small startup company was Apple Computer and the company executive was Steve Jobs. Jobs and a number of his development engineers visited Xerox PARC and within a year they released the Lisa and later the Mac computers. What was more amazing is that Xerox PARC just gave away most of their technology. Amazingly Xerox never really capitalized (other than laser printers and laser printing technologies) on most of the technologies they pioneered; however, other companies such as Oracle, Apple, 3Com, Apollo, Sun, Adobe, Microsoft and others did. Because of Xerox PARC we have the current graphical user interface, mouse, Ethernet networking, object-oriented programming, color laser printers, and many other technologies.

7.2 Event Driven Programming

The console programs that you have written so far have been procedural programs. A procedural program may include decisions, loops, branches and methods, but its distinguishing characteristic is that
you as the programmer specify exactly what happens from beginning to end. Graphical User Interface programs are much different. When you write a program that runs with a Graphical User Interface (GUI) you place various components, such as buttons and text fields, on the interface. When the user interacts with one of these components an event is generated. These events get stored on an event queue. A dispatcher takes events from the event queue and routes them to an appropriate event handler, written by you, the programmer. The event queue and the dispatcher are part of a GUI framework provided as part of the toolkit that you used to create your program. There is no concept of running a GUI program from start to finish. When a GUI program runs, there is simply a loop that runs in the background waiting for events to occur.

7.3 Windows Forms Applications

Microsoft provides us with a couple of different ways to create GUI programs. In this course we will write Windows Forms Applications because they are the easiest to learn to program. To create a new Windows Forms Application start Visual Studio and select File->New->Project as shown in Figure 7.1.

![Figure 7.1 Creating a New Windows Forms Application (GUI)]

In the New Project dialog, shown in Figure 7.2, click on Windows then select Windows Forms Application to create a GUI application. Insure that the Location and Name are what you want and click
OK to create the application. Here the name of the project is `GUI_Example_1` and the projects location is `G:\CS1410\Example Code`.

7.3.1 THE FORM

The most important element of a Graphical User Interface is the **Form**. The Form object represents a window. Figure 7.3 shows the basic Form created when you select a Windows Form Application. Note that the name on the tab says “Form1.cs[design]”. This simply means that you are looking at the page where you design the layout of the Form. The Form has a **Title Bar**(1) containing an **icon**(2) for the Form, and the **title**(3), in this case “Form1”. On the far right end of the title bars are three Buttons. These Buttons are the **Minimize**(4) button, the **Maximize/Restore**(5) button, and the **Close**(6) button.

The Form also has a **Frame**(7). The Frame is shown as a border that encloses the entire Form. The area inside the Title Bar and Frame of the Form is the called the **Client Area**(8). This is the working area where you will lay out the rest of the interface.
It is important to understand that all of the GUI components you will see in a Windows Forms Application are really objects that have Fields of data, Properties, Indexers, Methods and Events. So, everything you will learn about classes and the objects constructed from these classes is used to create a GUI component. The term GUI control is often used instead of GUI Component to describe these objects. The neat thing about a GUI control is that it has methods that know how to draw the visible graphic object on your computer display.

### 7.3.2 PROPERTIES OF A CONTROL

Every GUI control has a set of properties that describe the various aspects of that component. The properties for a particular control can be seen in its Properties Window. To see the properties of any GUI component, right click on the component and select “Properties” from the context menu, as shown in Figure 7.4.

You can also use the menu bar and select

```
View->Other Windows->Properties Window
```

to make the Properties Window visible.
Chapter 7: Graphical User Interfaces

Properties look like, act like, and work like publicly accessible variables; however, they are really methods. Using these methods, a programmer can set the value of a variable or retrieve the value of a variable. In Visual Studio you can inspect and change the values of Properties. In the Properties Window for the Form you can set such properties as the font, the background color, the size of the form and the text on the title bar. Figure 3.5 shows the Property Window for a Form control. The title that appears in the title bar of the Form is stored in the Text property of the form.

Visual Studio assigns each control a unique name, but you should change the name property to something more meaningful for your application. Figure 7.6 illustrates changing the name property.

One good approach to naming a control is to make the first three or four characters of the name stand for the type of control that it is referencing. The following is a list of such name prefixes:

- **Txt**  TextBox
- **Lbl**  Label
- **Btn**  Button
- **CBox**  ComboBox
- **RBtn**  RadioButton
- **Chb**  CheckBox
- **RTxt**  RichTextBox
- **MuT**  MenuStrip
Figure 7.5 Properties Window for a Form Control
Chapter 7: Graphical User Interfaces

Figure 7.6 Changing the Name Property

Figure 7.7 illustrates how the Text property is changed. Remember that the Text property contains the title text that will be displayed on the Title Bar.

Figure 7.7 Changing the Text Property (the text in the title bar)

7.3.3 CLIENT AREA
Most of our GUI design work will revolve around placing various controls on the Client area. Note that the Client Area uses X, Y and Z coordinates to place controls on its surface. The X coordinate is zero in the upper left corner of the Client Area and the positive direction is horizontal from left to right. The Y coordinate is zero from the upper left corner of the Client Area and the positive direction is vertically down the screen. The Z direction is zero on the Client Area and moves positive from the Client Area out of the screen towards the users, the Z coordinate is used to stack controls on the screen.
7.3.4 THE TOOLBOX

Controls are added to the client area by using the **ToolBox**. To see the ToolBox if it is not already visible, use the menu bar to select

**View->Other Windows ->ToolBox**

Figure 7.8 shows the ToolBox. The toolbox lists all of the different controls that can be added to your interface. In this figure the arrow points to the Button component in the ToolBox. If you double click on the Button icon a Button control will be displayed in the upper left corner of your Form’s Client Area. Using the mouse you can drag the button control to any location in the client area of the form. When selected, the button control will have resize handles around it. You can use these resize handles to change the size and shape of the button. The Button’s Name property will automatically be set to “button1” and its Text property will be set to “button 1”. You can and should change these to more meaningful names (see the style guide).

7.3.5 EVENT HANDLERS

We mentioned earlier, that unlike text based applications, GUI Applications are event driven. Events are mainly generated by two user inputs: a key press or a left mouse button click. In general, events are numerous and varied in a complex GUI, but for our purpose these two events will suffice to illustrate the use of event handling. When your GUI form is first loaded and started the operating system makes a method call to your application’s Main method, shown in the code listing on the following page. Note that this code is automatically generated by Visual Studio when you create a new Windows Form Application. You should never change this code.
namespace GUI_Example_1
{
    static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new FrmExample());
        }
    }
}

The last line in the Main() method creates our Form object and passes it to the Run() method. The Form constructor creates all of the controls that we have designed onto the form and initializes them. The Form is then displayed on the screen. The Run method now waits for events to be generated. When an event is generated, the program will try to find an event handler to execute. Once you have designed the visual interface for your program, your next job is to write the event handlers necessary to make your GUI components active.

To create an event handler, double click on the component that you want to generate an event handler for. This will generate the skeleton code for an event handler for that component. For example, if you had a button component named BtnTest on your form, and you double clicked on it, the following code would automatically be generated:

    private void BtnTest_Click(object sender, EventArgs e)
    {
    
    }

This code is really a method that will run when the user clicks on the button. When you run your application and do a Left-Mouse-Click on the “Test” button, a Left-Mouse-Click event is generated and a message is sent to the Windows OS to let it know that a user clicked the Left-Mouse-Button on the BtnTest button component. The Windows OS then looks at this message (event), and determines that it came from your application. It then sends this message to your application’s event queue. Your application, when it gets around to it, examines the event and calls the BtnTest_Click method passing it
two parameters. The parameter sender is the reference to the BtnTest object upcast to an object reference and the parameter e contains information about the mouse click event. Inside the body of the method you add whatever code you want to execute when the user clicks on that button. For example, to set the TxtName.Text property to the string literal George you would write

```csharp
private void BtnTest_Click(object sender, EventArgs e)
{
    TxtName.Text = "George";
}
```

Most GUI components are capable of generating many different events, and you have to be specific about which event you want to write an event handler for. Consider the example of a TextBox control namedTxtName. If you click on the TextBox and open the Properties Window you will notice a button at the top of the window that looks like a lightning bolt. This is the Shazam button. When you click on the Shazam button you will see the list of event handlers for the TextBox control. There are quite a few events as you can see in Figure 7.9. If you double click on the KeyPress event Visual Studio will generate an event handler as shown.

```csharp
private void TxtName_KeyPress(object sender, KeyPressEventArgs e)
{
    //add this code to the Event-Handler
    if(e.KeyChar == (char)Keys.Enter)
    {
        //add your code here
    }
}
```

### 7.4 Graphical User Interface Components (Controls)

In this section we will provide a brief overview of some of the most often used graphical user interface components. Keep in mind that there are many, many other GUI components available, and this is just an overview of a select few. You can find detailed documentation on all of the GUI controls on the MSDN web site.
Chapter 7: Graphical User Interfaces

7.4.1 FORM
We have already discussed the Form control. Every GUI Application consists of a Form and a number of associated controls on the Form’s Client area. The Form has a number of properties and events. The properties and events that we are initially interested in are

**Properties:**
- Name - The reference to the Form object
- Text - The text displayed on the Form’s Title Bar

**Events:**
- MouseClick
- KeyPress

There are two ways to exit your application and close the Form. You can either call the class’s Exit method or the Form’s Close method as shown in the code snippets below.

```csharp
Application.Exit();
Close(); or this.Close();
```

7.4.2 BUTTON
A Button control is used when you want to give the user an easy way to initiate some action. The Button properties and events that are of interest to us are:

**Properties:**
- Name - The reference to the Button object
- Text - The text displayed on the button
- BackColor = The button’s background color;
- ForeColor = The button’s foreground color

**Events:**
- MouseClicked

7.4.3 RADIOBUTTON
A RadioButton is designed to be used with two or more Radiobuttons in a group. When a RadioButton is clicked its Checked property is set to true. And the other Radiobuttons in the group will be unchecked and their Checked property set to false. This concept originated from buttons on old mechanical radios. When you pushed one button in it forced all the other buttons out, thus only one button at a time could be selected. The RadioButton properties and events of interest are:

**Properties:**
- Name - the reference to the RadioButton object
- Text - The text displayed next to the RadioButton
- Checked - True if the button is checked, otherwise "false"
Events:
  MouseClicked

7.4.4 TEXTBOX
TextBox controls are typically used to display text or to give the user a place to enter text from the keyboard. A TextBox stores the text it displays in its Text property. The following code snippets illustrate how to work with some of the properties of a TextBox.

    TxtName.Text = "Hello world";
    string stg = TxtName.Text;
   TxtName.BackColor = Color.Aqua;
   TxtName.ForeColor = Color.Red;
   TxtName.Enabled = false;
   TxtName.Visible = false;
   TxtName.ReadOnly = true;
   TxtName.TextAlign = HorizontalAlignment.Center;

Events:
  MouseClick
  KeyPress

We will often use TextBoxes in this class as a way of getting input from the user, and to display output for the user to see. Here is a small code example that gets an integer value from a TextBox, doubles the value, and then displays the result in a second TextBox.

    const int DOUBLE = 2;
    int num = int.Parse(inTxtBox.Text);
    int doubledNum = num * DOUBLE;
    string outStr = $"{doubledNum:D}", doubledNum);
    outTxtBox.Text = outStr;

7.4.5 RICHTEXTBOX
A RichTextBox control, is a combination of a multiline visual display panel and a backing array of strings. Whatever is placed in the display panel is stored in the backing array and whatever is stored in the backing array is displayed in the visible panel. The following code snippets illustrate how to work with some of the properties of a RichTextBox.

    RTxtDisplay.AppendText("Hello world!
");
    RTxtDisplay.Clear();
    int length = RTxtDisplay.Lines.Length;

7.4.6 LISTBOX
A ListBox control provides the user with a list of items that can be selected. You can define these items at design time, or they can be added at run time using the Add method. When the user selects an item an
event is generated. The following code illustrates how to work with some of the properties of a ListBox control.

```csharp
ListBox.Items.Clear();
ListBox.Items.Add("Line O");
string stg = (string)ListBox.Items[0];
int count = ListBox.Items.Count;

Events
    SelectedIndexChanged
    KeyPress
```

### 7.4.7 COMBOBOX

The ComboBox is a combination of a TextBox, Dropdown ListBox and a Button. It has some unique properties that make it a very useful control. The code snippets below show how to access some of its properties.

```csharp
ComboBox.Items.Clear();
ComboBox.SelectedIndex;
ComboBox.Items.Add("One");
string stg = (string)ComboBox.Items[0];

Events
    SelectedIndexChanged
    KeyPress
```

### 7.4.8 LABEL

Labels are mainly used to display information to the user and Label things like TextBoxes. However, they have a Text property that can be set at design time, but modified at runtime. You can get the Label’s Text property or set it. Labels can also respond to mouse clicks and other such events.

**Properties:**
- Name
- Text
- Font
- Size
- Shape
- Color

**Events**
- MouseClick
7.4.9 MESSAGE BOX

A MessageBox control displays a dialogue box, some text and optionally a button and an icon. They are useful for giving feedback or warnings to a user. You cannot add a MessageBox control at design time. They are created using the method `MessageBox.Show()`. There are several forms of this method. Two are shown below:

```csharp
MessageBox.Show( string ); // displays a message box with the given string

MessageBox.Show(string, string); // displays a message box with the given string and a caption
```
7.5 Beyond the Basics

Fig. 7.10 is an example of a more complex design of a GUI interface. It has a MenuStrip with two menu items Exit and File. The MenuStrip control provides the ability to have main menus such as Exit and File and submenus like Load and Save, as shown in Fig. 7.11 and 7.12. You can add these to the MenuStrip at design time. By double-clicking on the Load or Save submenus Visual Studio will generate an event handler for these submenus.

![GUI Example Form](image)

**Figure 7.10** A Complete GUI Example

![Menu Strip Submenus](image)

**Figure 7.11** The Exit Submenus

**Figure 7.12** The File Submenus
Also included on the Client design surface is a Label that contains the text *Name*, a read-only TextBox that contains the text *George Washington*, and a ComboBox that shows the current selection of *Butter*. The ComboBox is a combination of three controls: 1) a Button, a TextBox and a dropdown ListBox, as shown in Fig. 7.13. When you left-mouse-click on the small button with the down arrow on it the dropdown ListBox appears as shown in the Figure. When you left-mouse-click on the *Butter* list item it will generate a SelectedIndexChanged event. Each item in the ListBox is indexed in an array of strings from “0 Bread,” “1 Butter,” “2 Eggs,” and “3 Bacon.”

Finally, the client area contains a RichTextBox control with the text “Selected Index: 1” displayed, A ListBox control with the text “Line 0, Line 2” etc. displayed, and two RadioButtons – Salary Employee and Hourly Employee and three Buttons – “Test”, “Exit” and “Clear All”.

Detailed descriptions of all of the graphical user interface controls can be found on the MSDN website.