Understanding Windows Forms

Windows Forms, also known as WinForms, is one of the longest lived and oldest techniques for building Desktop applications. Windows Forms work as containers to host controls that allow you to present an application. Even though WPF has been available since 2007, WinForms are still used heavily in Enterprise application development and remains a strong competitor for WPF. If your application doesn't need heavy graphics, multimedia and/or a rich eye candy UI then Windows Forms is the best technology to build a Desktop client application.

User Interface Design Principles

The best mechanism for interacting with any application is often a user interface. Therefore, it becomes important to have an efficient design that is easy to use. When designing the user interface, your primary consideration should be the people who will use the application. They are your target audience, and knowing your target audience makes it easier for you to design a user interface that helps users learn and use your application. A poorly designed user interface, on the other hand, can lead to frustration and inefficiency if it causes the target audience to avoid or even discard your application.

Forms are the primary element of a Microsoft Windows application. As such, they provide the foundation for each level of user interaction. Various controls, menus, and so on, can be added to forms to supply specific functionality. In addition to being functional, your user interface should be attractive and inviting to the user.

Best Practices for User Interface Design

The user interface provides a mechanism for users to interact with your application. Therefore, an efficient design that is easy to use is of paramount importance. The following are some guidelines for designing user-friendly, elegant, and simple user interfaces.

Simplicity

Simplicity is an important aspect of a user interface. A visually "busy" or overly complex user interface makes it harder and more time-consuming to learn the application. A user interface should allow a user to quickly complete all interactions required by the program, but it should expose only the functionality needed at each stage of the application. When designing your user interface, you should keep program flow and execution in mind so that users of your application will find it easy to use. Controls that display related data should be grouped together on the form. ListBox, ComboBox, and CheckBox controls can be used to display data and allow users to choose from preset options.

The use of a tab order (the order by which users can cycle through controls on a form by pressing the Tab key) allows users to rapidly navigate fields.
Trying to reproduce a real-world object is a common mistake when designing user interfaces. For instance, if you want to create a form that takes the place of a paper form, it is natural to attempt to reproduce the paper form in the application. This approach might be appropriate for some applications, but for others, it might limit the application and provide no real user benefit, because reproducing a paper form can limit the functionality of your application. When designing an application, think about your unique situation and try to use the computer’s capabilities to enhance the user experience for your target audience.

Default values are another way to simplify your user interface. For example, if you expect 90 percent of the users of an application to select Washington in a State field, make Washington the default choice for that field.

Information from your target audience is paramount when designing a user interface. The best information to use when designing a user interface is input from the target audience. Tailor your interface to make frequent tasks easy to perform.

**Position of Controls**

The location of controls on your user interface should reflect their relative importance and frequency of use. For example, if you have a form for both the input of required information and of optional information then the controls for the required information are more important and should receive greater prominence. In Western cultures, user interfaces are typically designed to be read from left to right and from top to bottom. The most important or frequently used controls are most easily accessed at the top of a form. Controls that will be used after a user completes an action on a form, such as a Submit button, should follow the logical flow of information and be placed at the bottom of the form.

It is also necessary to consider the relatedness of information. Related information should be displayed in controls that are grouped together. For example, if you have a form that displays information about a customer, a purchase order, or an employee then you can group each set of controls on a Tab control that allows a user to easily move back and forth between displays.

Aesthetics is also an important consideration in the placement of controls. You should try to avoid forms that display more information than can be understood at a glance. Whenever possible, controls should be adequately spaced to create visual appeal and ease of accessibility.

**Consistency**

Your user interface should exhibit a consistent design across each form in your application. An inconsistent design can make your application seem disorganized or chaotic, hindering adoption by your target audience. Don’t ask users to adapt to new visual elements as they navigate from form to form.

Consistency is created through the use of colors, fonts, size, and types of controls employed throughout the application. Before any actual application development takes place, you should decide on a visual scheme that will remain consistent throughout the application.

**Aesthetics**
Whenever possible, a user interface should be inviting and pleasant. Although clarity and simplicity should not be sacrificed for the sake of attractiveness, you should endeavor to create an application that will not dissuade users from using it.

**Color**

Judicious use of color helps make your user interface attractive to the target audience and inviting to use. It is easy to overuse color, however. Loud, vibrant colors might appeal to some users, but others might have a negative reaction. When designing a background color scheme for your application, the safest course is to use muted colors with broad appeal.

Always research any special meanings associated with color that might affect user response to your application. If you are designing an application for a company, you might consider using the company's corporate color scheme in your application. When designing for international audiences, be aware that certain colors might have cultural significance. Maintain consistency, and do not overdo the color.

Always think about how color might affect usability. For example, gray text on a white background can be difficult to read and thus impairs usability. Also, be aware of usability issues related to color blindness. Some people, for example, are unable to distinguish between red and green. Therefore, red text on a green background is invisible to such users. Do not rely on color alone to convey information. Contrast can also attract attention to important elements of your application.

**Fonts**

Usability should determine the fonts you choose for your application. For usability, avoid fonts that are difficult to read or highly embellished. Stick to simple, easy-to-read fonts such as Palatino or Times New Roman. Also, as with other design elements, fonts should be applied consistently throughout the application. Use cursive or decorative fonts only for visual effects, such as on a title page if appropriate, and never to convey important information.

**Images and Icons**

Pictures and icons add visual interest to your application, but careful design is essential to their use. Images that appear "busy" or that distract the user will hinder use of your application. Icons can convey information, but again, careful consideration of end-user response is required before deciding on their use. For example, you might consider using a red octagon similar to a U.S. stop sign to indicate that users might not want to proceed beyond that point in the application. Whenever possible, icons should be kept to simple shapes that are easily rendered in a 16-by-16-pixel square.

**Working with Windows Forms**

To work with Windows Forms, you need to create a Windows Forms Application project using Visual Studio 2010. To do so, click "Start" -> "All Programs" -> "Visual Studio 2010" and from the list shown choose Microsoft Visual Studio 2010. This will open the Visual Studio Start Page. Click "File" -> "New" -> "Project...". Now you will see the New Project dialog box from which you can choose the Windows Forms Application template, as shown in Figure 1-1.
By default, the project is named "WindowsFormsApplication1" (the next would be "WindowsFormsApplication2", and so on). You can enter another name for your project in the Name text box when you choose the project template, or you can rename your project later. Once you have chosen the Windows Forms Application template and desired name and location, click "OK". This will open the Visual Studio Integrated Development Environment (IDE), called such because it has all the development-related tools, windows, dialog boxes, options, and so forth, embedded (or integrated) inside one common window, that makes the development process easier.

In the IDE, you will see that a Windows Forms form named "Form1.cs" has been added to the project, and on the right side you can also see the Solution Explorer window. You also need to understand one more window called the Properties window. If the Properties window is not available below the Solution Explorer window, you can open it by clicking "View” -> “Properties Window” or pressing F4. Now the development environment will look like Figure 1-2.
Because this is a Windows Forms Application project, you will be working with controls or tools that allow you to provide functionality in the form of a GUI. You can pick the controls from the Toolbox, shown on the left side of the Windows Forms form in the development environment. If you hover your mouse pointer on the Toolbox tab then the Toolbox window will open; expand the All Windows Forms tool set, as shown in Figure 1-3. You can pick controls from there and drop them onto the surface of the Windows Forms form.
Figure 1-3. IDE with the Toolbox

Understanding the Design and Code Views

You mainly deal with two views in the Visual Studio IDE, the Design View and the Code View. When you open the Visual Studio IDE, by default it displays the Design View, as shown in Figure 1-3. The Design View allows you to drag controls and drop them onto the form. You can use the Properties window to set the properties of objects and forms or other files shown in the Solution Explorer. The Solution Explorer also allows you to rename the project, forms, or even other files included in the project. You can rename these objects by selecting them, right-clicking, and selecting "Rename" from the context menu.

Basically, the Design View gives you a visual way to work with the controls, objects, project files, and so forth. You’ll want to use the other view available in the Visual Studio IDE, the Code View, when you are working with code to implement the functionality behind the visual controls sitting on the surface of your Windows Forms forms.

To switch from the Design View to the Code View, click "View" -> "Code" or right-click the Windows Forms form in the Design View and select "View Code". Either method will open the Code View for you, as shown in Figure 1-4.
The Code View displays all the code functionality. In Figure 1-4, note the Form1.cs tab (in which you see the Code View) is beside the Form1.cs (Design) tab, that is actually the Design mode of the Windows form Form1; these tabs allow you to switch among all the GUI elements of the Design View and the related code in the Code View that helps you to provide the functionality. Interestingly, if you tried accessing the Toolbox while in the Code View, you will see that there are no controls in the Toolbox. But when you switch back to the Design View, you’ll find the Toolbox is fully loaded with the controls.

To switch back to the Design View, right-click the form in the Code View and select View Designer; you will see that now you are back to the Design View and can continue working with the visual elements, or controls.

You can also use Solution Explorer to switch among the Design and Code Views by selecting your desired Windows Forms form (in case you have multiple Windows Forms forms open), right-clicking, and choosing either View Code or View Designer. This will open either the Code or Design View of the selected Windows Forms form.

**Sorting Properties in the Properties Window**

Each object such as a form control has many properties you may need to set while working with an application. To help you navigate the many properties listed in the Properties window, you can sort them either categorically or alphabetically. Let’s look at each of these sorting options.
**Categorized View**

The Categorized View organizes properties in the form of sets of properties, and each set has a name to describe that collection of properties; for example, there are categories named Appearance, Behavior, Data, Design, Focus, and so on. You can switch to the Categorized View by clicking the icon on the very left of the toolbar shown at the top of the Properties window.

In Figure 1-5, which shows the Categorized View, under the Appearance category, you will see all properties listed that define the look and feel of the object (in this case, a form). Note that the other categories are also shown in Figure 1-5.

**Note:** I have intentionally kept the other categories in the collapsed mode in Figure 1-5, just to show you all the categories. When you switch to the Categorized View, you will see that all the categories are expanded by default.

![Categorized View of properties](image)

**Alphabetical View**

The Alphabetical View organizes properties in ascending order by name from a to z. You can switch to the Alphabetical View by clicking the icon second from the left on the toolbar shown in the top of the Properties window.

Figure 1-6 shows this view; all the properties listed are organized alphabetically. Working with the Alphabetical View, rather than the Categorized View, makes life much easier. For example, say you are seeking the Font property. In the Categorized View, you need to understand under which category this
property is located to find it. However, if you have properties organized in the Alphabetical View, you can easily locate this property because it begins with the letter F, so you know whether you need to go back or forward to find this property for your control.

![Alphabetical View of properties](image)

**Figure 1-6. Alphabetical View of properties**

**Setting the Properties of Solutions, Projects, and Windows Forms forms**

Before you begin putting controls onto the Windows Forms form, you need to learn how to modify some property values of the solution, project, and form you created earlier (shown previously in Figure 1-2).

Select the “WindowsFormsApplication1” solution, go to the Properties window, and set its Name property value to “WindowsFormApp”.

**Note**: In some cases you may not be able to see the solution (.sln) file in Visual Studio. To have a solution file listed, such as “Solution <ProjectName>(1project)”, as shown in Figure 1-7, you need to click “Tools” -> “Options”, go to the Project and Solutions tab, choose General, check the “Always show solution” option, and click “OK”.

Select the “WindowsFormsApplication1” project in Solution Explorer, go to the Properties window, and modify the Project File property value, that defines the file name of the project, to appear as “WinApp.csproj”.

Select the “WindowsFormsApplication1” project in Solution Explorer, go to the Properties window, and change the name of the Windows Forms form: select “Form1.cs” in the Solution Explorer, in the Properties window modify the File Name property from “Form1.cs” to “WinApp.cs” then click “Yes” in the dialog box that appears.
Now click "Form1", located in the Solution Explorer window. Once Form1 is selected, you will see that the list of properties has changed in the Properties window. Select the Text property and modify its value from "Form1" to "Windows Application". The Text property defines the name shown on the title bar of the form.

**Working with Controls**

Now that you have your Windows Forms application in place, you can start working with the controls.

The basic element of any Windows application is the control, that plays a key role by providing the visual meaning of the code functionality embedded in an application.

The most commonly used controls are Label, Button, TextBox, RadioButton, CheckBox, ListBox, ComboBox, MenuStrip, and ContextMenuStrip. Applications cannot exist without these controls, so you'll see how you can incorporate some of them into your application.

**Try It: Working with the Label, TextBox, and Button Controls**

In this exercise, you'll create a Windows Forms application with three labels, two text boxes, and a button. The application will accept your name as input and then flash a "Welcome" message in the form of a dialog box.

1. Go to the project named WinApp located under the solution, that you created earlier (refer to Figure 1-7). Ensure that you are in the Design View.
2. Drag a Label control onto the form, and position it at the top middle of the form. Select this label, navigate to the Properties window, and set the following properties:
   - Set the Name property to lblWelcome.
   - Set the Text property to Welcome.
   - Select the Font property, click the ellipsis button, and specify the size of the Label control as 16 points in the Size drop-down list.
   - Set the TextAlign property to TopCenter.

   **Tip:** You can also double-click any control in the Toolbox to add it to the form. The difference between dragging a control and double-clicking is that while dragging, you can position the control as you desire on the form. But if you just double-click a control then it will be added to the top-left corner; therefore, if you prefer it in a different location then you still need to drag it to there.

3. Drag two more Label controls onto the form, and put them below the "Welcome" text, a little toward the left of the form. Select the first label, navigate to the Properties window, and set the Name property to lblFirstName and the Text property to First Name.
4. Now select the second label, navigate to the Properties window, and set its Name property to lblLastName and its Text property to Last Name.
5. Drag two TextBox controls onto the form, and put the TextBox named textBox1 in front of the First Name label and the TextBox named textBox2 in front of the Last Name label.
6. Select textBox1, go to the Properties window, and set its Name property to txtFname. Select textBox2, and in the Properties window set its Name property to txtLname.

7. Drag a Button control onto the form, and place it below the Label and TextBox controls. Select the Button control, go to the Properties window, change the Name property to btnSubmit, and then set its Text property to Submit.

8. Now you have your GUI design of the application ready; it should resemble the form shown in Figure 1-8.

![Image of GUI design of the Windows Application form]

Figure 1-8. GUI design of the Windows Application form

It’s time to add functionality and switch to the Code View. You will obtain the First Name and Last Name values supplied by the user and flash a message on a click of the Submit button, which means you need to put all the functionality behind the Submit button's click event, that will eventually read the values from the TextBox controls. To do this, continue with the following procedure.

Double-click the Submit button. This will take you to the Code View, and you will see that the btnSubmitClick event template has been added to the code. Now you will add the code to show a dialog box, with a greeting and welcome message for the first name and last name that were entered. To do so, you will use the MessageBox class; this class provides a Show() function to display a dialog box with the provided information. Now let’s add the following code inside this btnSubmitClick event to do the desired functionality of a dialog, with a message, a caption in the dialog box’s title bar, an OK button, a Cancel button, and an information icon displayed:

```csharp
MessageBox.Show("Hello" + ' ' + txtFname.Text + ' ' + txtLname.Text + ' ' + "Welcome to the Windows Application", "Welcome", MessageBoxButtons.OKCancel, MessageBoxIcon.Information);
```

Now your Code View will show the button’s click event code, as shown in Figure 1-9.
Now click "Build" -> "Build Solution", and ensure that you see the following message in the Output window:

======== Build: 1 succeeded or up-to-date, 0 failed, 0 skipped ========

Now it’s time to run and test the application. To do so, press Ctrl+F5. Visual Studio 2010 will load the application.

Enter values in the First Name and Last Name text boxes, and then click the Submit button; you will see a message similar to the one shown in Figure 1-10.

How It Works

Visual Studio comes with many features to help developers while writing code. One of these features is that you can just double-click the GUI element for which you want to add the code, and you will be taken to the code associated with the GUI element in the Code View. For example, when you double-click the Submit button in the Design View, you are taken to the Code View, and the btnSubmitClick event template is automatically generated.
To provide the functionality for this control, you add the following code:

```csharp
setMessageBoxShow("Hello" + ' ' + txtFname.Text + ' ' + txtLname.Text + ' ' + "Welcome to the Windows Application", "Welcome", MessageBoxButtons.OKCancel, MessageBoxIcon.Information);
```

"setMessageBoxShow()" is a .NET Windows Forms method that pops up a message box based on the arguments provided. To display a "Welcome" message with the first name and last name specified by the user in the message box, you apply a string concatenation approach when writing the code.

In the code segment, you hard-code the message “Hello Welcome to the Windows Application,” but with the first name and last name of the user appearing after the word Hello and concatenated with the rest of the message, "Welcome to the Windows Application."

For readability, you also add single space characters (') concatenated by instances of the + operator between the words and values you are reading from the Text property of txtFnam and txtLname. If you do not include the single space character (') during string concatenation, the words will be run into each other, and the message displayed in the message box will be difficult to read.

The second parameter you are provided with is Caption, which is the title of the dialog. We hard-coded this as "Welcome," and then we chose our button set through "setMessageBoxButtons.OKCancel". The final argument we passed is "setMessageBoxIcon" and used an information type icon.

**Note:** "setMessageBoxShow()" is a very powerful and handy function; you may want to play a little more with various choices that IntelliSense shows for the "setMessageBoxButtons" and "setMessageBoxIcon" type arguments.

**Setting Dock and Anchor Properties**

Prior to Visual Studio 2005, resizing Windows forms would require you to reposition and/or resize controls on those forms. For instance, if you had some controls on the left side of a form and you tried to resize the form by stretching it toward the right side or bringing it back toward the left, the controls wouldn’t readjust themselves according to the width of the resized form. Developers were bound to write code to shift controls accordingly to account for the user resizing the form. This technique was very code heavy and not so easy to implement.

With Visual Studio 2005 and onwards came two new properties, Anchor and Dock, that are easy to set at design time. The same Dock and Anchor properties are available with Visual Studio 2010, and they solve the problem with the behavior of controls that users face while resizing forms.

**Dock Property**

The Dock property allows you to attach a control to one of the edges of its parent. The term parent applies to Windows Forms forms, because Windows Forms forms contain the controls that you drag and drop on them. By default, the Dock property of any control is set to None.

For example, a control docked to the top edge of a form will always be connected to the top edge of the form, and it will automatically resize in the left and right directions when its parent is resized.

The Dock property for a control can be set by using the provided graphical interface in the Properties
window, as shown in Figure 1-11.

![Figure 1-11. Setting the Dock property](image1.png)

**Anchor Property**

When a user resizes a form, the controls maintain a constant distance from the edges of their parent form using the Anchor property. The default value for the Anchor property for any control is set to Top, Left, that means that this control will maintain a constant distance from the top and left edges of the form. The Anchor property can be set by using the provided graphical interface in the Properties window, as shown in Figure 1-12.

![Figure 1-12. Setting the Anchor property](image2.png)

Because of the default setting of the Anchor property to Top, Left, if you try to resize a form by stretching it toward the right side, you will see that its controls are still positioned on the left rather than shifting to the center of the form to adjust to the size of the form after resizing is done.

If opposite edges, for example, Left and Right, are both set in the Anchor property, the control will stretch when the form is resized. However, if neither of the opposite edges is set in the Anchor property, the control will float when the parent is resized.

**Try It: Working with the Dock and Anchor Properties**
In this exercise, you will use the existing Windows Forms application named WinApp, that you created previously in the chapter. You will see how to modify this application in such a way that when you resize the form, its controls behave accordingly and keep the application presentable for the user.

1. Go to Solution Explorer and open the WinApp project. Open the WinApp form in the Design View.
2. Select the form by clicking its title bar; you will see handles around the form's border, that allow you to resize the form's height and width.
3. Place the cursor on the handle of the right border, and when the mouse pointer becomes double-headed, click and stretch the form toward the right side. You will see that the form's width increases, but the controls are still attached to the left corner of the form.
4. Similarly, grab the handle located on the bottom of the form and try to increase the height of the form. You will notice that the controls are still attached to the top side of the form.
5. Have a look at Figure 1-13, that shows a resized (height and width) form and the position of the controls. The controls appear in the top-left corner because their Dock property values are None and their Anchor property values are Top, Left.

![Figure 1-13. Resized form and position of controls](image)

6. Now you will try to set the Dock and Anchor properties for the controls and then retest the application.
7. Select the Label control named lbl Welcome, and set the Text value to Welcome. Go to the Properties window. Select the AutoSize property, and set its value to False (default value is True).
8. Resize the width of the Label control to the width of the form, and adjust the Label control to the top border of the form. Set this control's TextAlign property to Top, Center.
9. Set the Dock property for the Label control from None to Top, that means you want the label to always be affixed with the top border of the form.
10. Now select all the remaining controls (two Labels, two TextBoxes, and one Button) either by scrolling over all of them while holding down the left mouse button or by selecting each with a click while pressing down either the Shift or Ctrl key.
11. Once you have selected all the controls, go to the Properties window. You will see listed all the properties common to the controls you have selected on the form.
12. Select the Anchor property; modify its value from the default Top, Left to Top, Left, and Right. This will allow you to adjust the controls accordingly as soon as you resize the form. The controls will also grow in size accordingly to adjust to the width of the form, as you can see in Figure 1-14.
14. The effect of the Anchor property setting Top, Left, Right on a resized form

**Note:** The Anchor property has very interesting behaviors; you can try setting this property in various combinations and see the effects when you resize your form.

13. Return the form to its previous size so you can see the effects of setting another Anchor property.
14. Select all the controls again as you did in step 8. Set the Anchor property to Top only and try resizing the form now. You will notice that the controls are floating in the middle of the form when you resize it, as you can see in Figure 1-15.

15. Save the changes in your project by clicking “File” | “Save All”.

**How It Works**

When you resize the form, it will behave according to the settings of the Dock and Anchor properties.

In the first instance, you set the Dock property of the Label control to Top, that allows this Label control to be affixed to the top border of the form and span the entire width of the form. Setting the Anchor property of the remaining controls to Top, Left, and Right shifts the controls in such a manner that they will maintain a constant distance from the left and right borders of the form.
Adding a New Form to the Project

Any real-world or enterprise application will obviously need multiple Windows Forms forms to perform business functionality. By default, every project opens with only one Windows Forms form, but you are free to add more.

Try It: Adding a New Form to the Windows Project

In this exercise, you will add another Windows Forms form to your project. You will also work with ListBox, ComboBox, RadioButton, and CheckBox controls. In this new form, you will add data from two different text boxes to a ListBox and a ComboBox, respectively.

1. Navigate to Solution Explorer and select the WinApp project, right-click, and click "Add a Windows Form". This will add a new Windows Forms form to your project.
2. In the Add New Item dialog box displayed, change the form’s name from "Form1.cs" to "UserInfo.cs". Click "Add". The new form with the name "UserInfo" will be added to your project.
3. Ensure that the newly added form "UserInfo" is open in the Design View. Select the UserInfo form by clicking the form’s titlebar, navigate to the Properties window, and set the Size property’s Width to 455 and Height to 251.
4. Drag a Label control onto the form; select this control, navigate to the Properties window, and set the following properties:
   - Set the Name property to lblCountry.
   - Set the AutoSize property to false.
   - Set the Location property’s X to 12 and Y to 26.
   - Set the Size property’s Width to 71 and Height to 13.
   - Set the Text property to Enter Country.
5. Drag a TextBox control in front of the lblCountry label. Select this control, navigate to the Properties window, and set the following properties:
   - Set the Name property to txtCountry.
   - Set the Location property’s X to 97 and Y to 19.
   - Set the Size property’s Width to 129 and Height to 20.
6. Drag another Label below lblCountry, select this control, navigate to the Properties window, and set the following properties:
   - Set the Name property to lblState.
   - Set the AutoSize property to false.
   - Set the Location property’s X to 12 and Y to 65.
   - Set the Size property’s Width to 60 and Height to 13.
   - Set the Text property to Enter State.
7. Drag a TextBox control in front of the lblState label. Select this control, navigate to the Properties window, and set the following properties:
   - Set the Name property to txtState.
   - Set the Location property’s X to 97 and Y to 58.
   - Set the Size property’s Width to 129 and Height to 20.
8. Drag a ListBox control onto the UserInfoInfo form on the right side of the TextBox control you have added. Select this control, navigate to the Properties window, and set the following properties:
   - Set the Name property to lstCountry.
   - Set the Location property's X to 280 and Y to 12.
   - Set the Size property's Width to 129 and Height to 82.

9. Drag a ComboBox below the ListBox that you just added. Select this control, navigate to the Properties window, and set the following properties:
   - Set the Name property to cboState.
   - Set the Location property's X to 280 and Y to 117.
   - Set the Size property's Width to 129 and Height to 21.

10. Drag two Checkboxes below the Label controls, and name them chkPostalMail and chkEmail; set their Text property to Postal Mail and E-Mail, respectively.

11. Drag two RadioButtons below the TextBox controls, and name rdbMale and rdbFemale; set their Text property to Male and Female, respectively.

12. Drag a Button control to the UserInfo form to the left side below the CheckBox controls; select this control, navigate to the Properties window, and set the following properties:
   - Set the Name property to btnAdd.
   - Set the Location property's X to 12 and Y to 165.
   - Set the Size property's Width to 75 and Height to 23.
   - Set the Text property to Add.

13. Drag a Button control next to the Add button; select this control, navigate to the Properties window, and set the following properties:
   - Set the Name property to btnRemoveCountry.
   - Set the Location property's X to 105 and Y to 165.
   - Set the Size property's Width to 95 and Height to 23.
   - Set the Text property to Remove Country.

14. Drag a Button control next to the Remove Country button; select this control, navigate to the Properties window, and set the following properties:
   - Set the Name property to btnRemoveState.
   - Set the Location property's X to 220 and Y to 165.
   - Set the Size property's Width to 86 and Height to 23.
   - Set the Text property to Remove State.

15. Drag a Button control next to the Remove State button; select this control, navigate to the Properties window, and set the following properties:
   - Set the Name property to btnShowDetails.
   - Set the Location property's X to 327 and Y to 165.
   - Set the Size property's Width to 100 and Height to 23.
   - Set the Text property to Show Details.

Now you are done with the design part of the UserInfo form; while dragging and dropping the controls, you should place the controls to create a visually appealing layout, as shown in Figure 1-16.
You want the user to enter a name into the TextBoxes and click the "Add" button, after which the country will be added to the ListBox and the state will be added to the ComboBox. Accordingly, the "Remove" button will remove the country or state, and "Show Details" will show the selection made with a CheckBox and RadioButton. To do so, you need to write the code functionality behind the click event of all these buttons one by one.

16. Double-click the Add button and write the following code inside the btnAdd_Click event, that will read the country and state name entered into the textboxes and add them to the ListBox and ComboBox.

```csharp
lstCountry.Items.Add(txtCountry.Text);
txtCountry.Clear();
cboState.Items.Add(txtState.Text);
txtState.Clear();
```

17. Double-click the "Remove Country" button, and write the following code inside the "btnRemoveCountry_Click" event, that will remove the selected country from the ListBox named lstCountry:

```csharp
lstCountry.Items.Remove(lstCountry.SelectedItem);
```

18. Double-click the "Remove State" button, and write the following code inside the "btnRemoveState_Click" event, that will remove the selected state from the ComboBox named cboState.

```csharp
cboState.Items.Remove(cboState.SelectedItem);
```

19. Double-click the "Show Details" button, and write the following code inside the "btnShowDetails_Click" event, that will show the options selected via a CheckBox and RadioButton.

```csharp
if (chkEmail.Checked == true || chkPostalMail.Checked == true && rdbMale.Checked == true)
```
{   MessageBox.Show("Hello Mr, you will be contacted by either USPS or email","Information",MessageBoxButtons.OKCancel,MessageBoxIcon.Information); }
else if (chkEmail.Checked == true || chkPostalMail.Checked == true && rdbFemale.Checked == true) {
    MessageBox.Show("Hello Mam, you will be contacted by either USPS or email","Information",MessageBoxButtons.OKCancel,
    MessageBoxIcon.Information);
}

20. Go to the Build menu and select "Build Solution". You should see a message indicating a successful build.

Keep your current project open, because you'll need it immediately for the next exercise. (Don't worry, we'll explain how this and the next exercise works afterward.)

**Try It: Setting the Start-up Form**

Setting the start-up form in a Visual C# project is a little tricky, so I wanted to break it into its own exercise. To set a start-up form, you need to follow these steps:

1. In the project you modified in the previous exercise, navigate to Solution Explorer, open the Program.cs file, and look for the following code line:

   ```csharp
   Application.Run(new WinApp());
   ```

   This code line ensures the WinApp form will be the first form to run all the time because this is the first form called Form1, that was already added and you renamed it when you began this project; in order to set the UserInfo form as the start-up form, you need to modify this statement a little, as follows:

   ```csharp
   Application.Run(new UserInfo());
   ```

   Build the solution, and run and test the application by pressing Ctrl+F5. The UserInfo application form will be loaded. Enter a country and state name in the appropriate text boxes and click the Add button; you will see that the names you entered have been added to the ListBox and ComboBox

   elect both the CheckBoxes and one of the RadioButtons and click the ShowDetails button; you will see that a message box is displayed, as shown in Figure 1-18.
Figure 1-18. Using CheckBox and RadioButton with UserInfo Windows Forms application

**How It Works**

Let's have a look at the "Adding a New Form to the Windows Project" task first and understand the code, button by button and line by line.

First you have an Add button, that adds the country and state to the ListBox and ComboBox. The ListBox and ComboBox controls have a collection named Items, and this collection can contain a list of items, that is why you use it here. Next you call up the Add method of the Items collection, and finally you pass the value entered in the TextBox to the ListBox’s or ComboBox’s Items collection’s Add method, as shown here:

```csharp
lstCountry.Items.Add(txtCountry.Text);
```

Also, once an item is added, for a better user experience, it is advisable to clear the text box so a user can type in a new value with ease.

```csharp
txtCountry.Clear();
```

You repeat the same for the ComboBox:

```csharp
cboState.Items.Add(txtState.Text);
txtState.Clear();
```

For the "Remove Country" and "Remove State" buttons, you follow a similar approach as the Items collection, but this time instead of "Add()" you call the "Remove()" method, and as you know, the prerequisites for an item removal is that an item must be selected first either in the ListBox or the ComboBox. Hence, the code seeks a SelectedItem to be passed to the Remove() method.
Now for the "Show Details" button, you have used some conditional logic to produce a different message based on your selection, specifically with the Male and Female radio buttons.

The CheckBox and RadioButton controls offer a property named Checked, that can be either true or false, that is, checked or not checked. You build a condition around these and then show a message box.

```csharp
if (chkEmail.Checked == true || chkPostalMail.Checked == true && rdbMale.Checked == true)
{
    MessageBox.Show("Hello Mr, you will be contacted by either USPS or email", "Information", MessageBoxButtons.OKCancel, MessageBoxIcon.Information);
}
else
if (chkEmail.Checked == true || chkPostalMail.Checked == true && rdbFemale.Checked == true)
{
    MessageBox.Show("Hello Mam, you will be contacted by either USPS or email", "Information", MessageBoxButtons.OKCancel, MessageBoxIcon.Information);
}
```

In the "Setting the Startup Form" task, you create an instance of the AddName form in the Program.cs file, as shown in the following code:

```csharp
Application.Run(new UserInfo());
```