Top of Form



**Introduction to C# Programming: Lesson 12**

**Chapter 1**

**Introduction**

So far, this course has focused on some very standard programming structures. You learned how to create and use variables in the sequence, selection and repetition structures. You then learned how to extend this knowledge by working with arrays and classes. And while this knowledge will allow you to write very substantial, useful programs, these programs lack the pizzazz of current programs. The reason I say this is because all of the programs you have written so far are run at the command line and interact with the user one element at a time. This is how all programs used to run. However, programs that use a graphical user interface (GUI) are much easier for the user and are what users have come to expect.

You may have even experienced this fact during your coursework. I usually have a couple people admit that they have written a program that is very useful. They are so proud of their work that they will show the program to a friend. Since their programs do not have the glitter and shine of a GUI, the friend usually is impressed, but not stunned. Therefore, in your final lesson you will learn how to program with a GUI.

Those of you who have a copy of Microsoft's Visual C# .NET have an Integrated Development Environment (IDE) available to you. This IDE allows you to very easily create Windows objects by dragging and dropping. While this is good, it is still useful for you to learn how to do this without the IDE. This will give you a better appreciation of what the IDE is doing for you.  
  
  
  
  
**Chapter 2**

**Message Boxes**

I am sure that you have been using a program and a little dialog box has popped up giving you some instruction. It may have been a warning, or it may have asked you a question. These boxes are called *message boxes*, and they are very easy to include in your program by including the appropriate namespace. To do this, you just need to type the following statement at the top of your program:

using System.Windows.Forms;

The name of the class you will use to create a message box is called MessageBox. MessageBox has an overloaded static method named *Show()*. The following simple program can be used to display a message box:

using System;

using System.Windows.Forms;

public class MessageEx1

{

public static void Main()

{

MessageBox.Show("Hello world!");

}

}

If you run this code, you should see the following:

  
Fig. 12.1. Message box created from running MessageEx1.exe

Notice how Main only has one statement:

MessageBox.Show("Hello world!");

This code will open a standard-looking Windows dialog box containing a title bar, a **Close** button, a message, and an **OK** button. When the user clicks on either the **Close** button or the **OK** button, the message box closes. All of these capabilities have been taken care of for you in the MessageBox class. If you weren't convinced that object-oriented programming was easier a few lessons ago when you wrote your own class, you should now!

Now you may be thinking, "Big deal. It's just a message box. So what?" Well, the big deal is that we can do much more with our message boxes than just displaying a message. We can use these things to interact with our users in more useful ways. Remember earlier I said that the Show() method was overloaded? Well, that means that we can pass over more arguments to the method other than just the text we want displayed. For example, we can write code to make the message box have a title by passing two strings to the method:

MessageBox.Show("Hello world!",   
"Hello Program");

Now that you have learned some basics about the MessageBox class, let's explore some different ways to make our programs more interesting. Try typing the following program:

using System;

using System.Windows.Forms;

public class Count

{

public static void Main()

{

int count = 1;

MessageBox.Show("The value of count is: " ¬

+ count, "Count Program");

count = count + 1;

MessageBox.Show("The value of count is: " ¬

+ count, "Count Program");

}

}

This example should open a message box that displays some text, and then concatenated on the end of that text is the value of the count variable. The program then adds one to *count* and displays another message box.

As I said before, we can use the message box to interact with the user by asking a question. Of course, if they only have the option of the **OK** button, then the program won't be very useful. Instead, we can display different buttons by passing an argument that lets the message box know which buttons to display. For example:

MessageBox.Show("Hello!",

"Hello Program", MessageBoxButtons.YesNo);

This message box will display the text, a title, a **Yes** button, and a **No** button. Also, I didn't mention it earlier, but the Show() method is a value-returning method that returns a DialogResult object. So, if the user clicks the **Yes** button, the Show() method will return a DialogResult.Yes object that we can test. This may be easier to see with an example:

using System;

using System.Windows.Forms;

public class CountQuestion

{

public static void Main()

{

int count = 1;

String message = "The value of count is: ";

String question = "\nDo you want to add one?";

if (MessageBox.Show(message + count + question, ¬

"Count Program", MessageBoxButtons.YesNo) ¬

== DialogResult.Yes)

{

count = count + 1;

MessageBox.Show(message + count, "Count ¬

Program");

}

}

}

This program displays a message box that shows the value of *count* and then asks the user if they want to add one to that value. If the user clicks the **Yes** button, then one is added to *count*, and a new message box is displayed. However, if the user does not click on the **Yes** button, the program ends.

I want to point out in this example how I made the text inside the message move to the next line. I did this with the new line character ("\n"). In fact, your messages can have any number of lines. Of course, you probably won't want to have too many lines; otherwise, your users may not read all of the text.

I also want you to notice how I wrote my if statement onto multiple lines. Recall that white space does not matter to the compiler, and I am using this to my advantage. Now my code is easier to read.

As you may have guessed, there are many different sets of buttons that you can have displayed in your message box. The following table lists some of them.

|  |
| --- |
| **Table 12.1. MessageBoxButton values** |
| **MessageBoxButton Member** |
| AbortIgnoreRetry |
| OK |
| OKCancel |
| RetryCancel |
| YesNo |
| YesNoCancel |

Although there are many other options available with the Show() method, I want to finish this subject by talking about the icon that can be displayed. For example:

MessageBox.Show("Having fun?",

"My Program",

MessageBoxButtons.YesNo,

MessageBoxIcon.Question);

This line of code will display a question, show a title, show a **Yes** and a **No** button, and display the question mark icon. It is useful to provide an icon to your user when you display a message box, because it will give your user a better clue of what is happening. As you probably guessed, there are many different icons that you can have your message box display. The following table shows some of them:

|  |  |
| --- | --- |
| **Table 12.2. MessageBoxIcon values** | |
| **MessageBoxIcon Member** | **Image Displayed** |
| Error | https://api.ed2go.com/CourseBuilder/2.0/images/resources/prod/cpb-0/error.gif |
| Exclamation | https://api.ed2go.com/CourseBuilder/2.0/images/resources/prod/cpb-0/exclamation.gif |
| Information | https://api.ed2go.com/CourseBuilder/2.0/images/resources/prod/cpb-0/information.gif |
| Question | https://api.ed2go.com/CourseBuilder/2.0/images/resources/prod/cpb-0/question.gif |

**Chapter 3**

**Creating a Windows Form Object**

The previous lesson was just your first step into the GUI world. After all, there is only so much that can be done by asking yes or no questions. Look at the browser in which you are reading this text. It has lots of different buttons, menus and objects with which you can interact. Programmers usually refer to these Windows objects as *controls*. For the remainder of this lesson, you will learn about how to make your programs contain some of these controls.

The most basic control that is used in GUI applications is the form. Your browser window, or any other application window, is actually called a form. Let me show you what I mean with another simple example:

using System;

using System.Windows.Forms;

using System.Drawing;

public class MyForm

{

public static void Main()

{

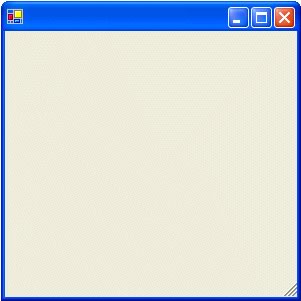
Form form1 = new Form();

form1.ShowDialog();

}

}

If you run this program, you will see something like the following:

  
Fig. 12.1. Simple Windows form

This first thing you should notice is the addition of the System.Drawing namespace. This namespace will provide some important classes for us later. Looking at the code, you can see that Form is a class that we can instantiate. The first line creates our form object named *form1*, and the second line calls on the ShowDialog() method on that object.

In addition to the ShowDialog() method, the Form class also has a number of instance variables. Programmers usually refer to control instance variables as *properties*. The following table lists a few common form properties:

|  |  |
| --- | --- |
| **Table 12.3. Common Form Properties** | |
| **Form Properties** | **Description** |
| BackColor | The background color  as a System.Drawing.Color |
| ForeColor | The foregrond color  as a System.Drawing.Color |
| Height | The height in pixels |
| Left | The distance in pixels from the left |
| Text | The text to be displayed |
| Top | The distance in pixels from the top |
| Width | The width in pixels |

Actually, most controls have these same properties. Try typing the following program in to see exactly what each of these properties does.

using System;

using System.Windows.Forms;

using System.Drawing;

public class MyForm

{

public static void Main()

{

Form form1 = new Form();

form1.Text = "My New Form";

form1.BackColor = Color.Blue;

form1.ForeColor = Color.White;

form1.Height = 100;

form1.Width = 500;

form1.Left = 200;

form1.Top = 400;

form1.ShowDialog();

}

}

Notice how you set the values of the control's properties exactly the same as you did when setting the values of public instance variables for your controls. You may want to try to change some of the values to get a better feel for what the properties do.

People began writing applications this way and soon found that it was easier to organize the code if instead they put the code that described their form into its own class and had their Main() method just run the application. The following example shows how this can be done to set a few form properties and display the form.

using System;

using System.Windows.Forms;

using System.Drawing;

public class MyForm2 : Form

{

public static void Main()

{

Application.Run(new MyForm2());

}

public MyForm2()

{

Text = "My New Form";

BackColor = Color.Blue;

ForeColor = Color.White;

}

}

Once again, you can see that the Main() method passes a new MyForm2 object to the Run() method. When the MyForm2 object is created, the constructor is called and some of the form's properties are set. Notice how there is no direct call to the ShowDialog() method. This is because the Run() method is taking care of that for us.

You may also be wondering what is going on with the line that says:

public class MyForm2 : Form

This line is telling the compiler that our new class, MyForm2, is inheriting from the Form class. Due to time constraints, this course will not go further into the topic of inheritance except to say that a derived class gets all of the methods and variables of a base class, and it can add methods and variables of its own. In our example here, MyForm2 is the derived class and Form is the base class.

The reason I show you this new way to write your application is because some of you are using an IDE, and others may eventually use one. When Microsoft's IDE creates GUI applications, this is how the code is written. Since I want you to be able to know what is happening in their code, I will write the remaining programs in this fashion.  
  
  
  
  
**Chapter 4**

**Adding More Controls to Your Form**

OK now, I realize that just creating a form isn't very exciting. So, let's put some other controls on our form. The first control I want to discuss is the Label control. When you want to display some text on your form, you can create a label.

**Labels**

A Label control has a number of properties associated with it. Actually, all of the common form properties that were listed earlier are contained in the Label control as well. That means that if you want text displayed inside your Label control, you will need to set its Text property. Type the following code in to see how to place a label on your form:

using System;

using System.Windows.Forms;

using System.Drawing;

public class MyForm2 : Form

{

private Label myLabel;

public static void Main()

{

Application.Run(new MyForm2());

}

public MyForm2()

{

Text = "My New Form";

BackColor = Color.Blue;

ForeColor = Color.White;

myLabel = new Label();

Controls.Add(myLabel);

myLabel.Text = "My first Label";

}

}

The first thing I want to point out to you is that the label is a private instance variable. This is done so that all of the methods have access to this control. Next, you can see that inside the MyForm2 constructor, in addition to the code to create the form, we have code to instantiate our myLabel Control. We next call on the Add() method to add this control to our form. Finally, we write code to change the value of the Text property. Since we did not set the value of the Top or Left properties, the default values are used, which are zero. Therefore, when you run this program, the label will show up in the upper left corner of the form.

**Buttons**

Although labels let us display text to our user, this doesn't make for very exciting programs. Let's make our program interactive by creating a button control to put on our form. You have already seen how to choose which buttons show up on a message box, but now we want to create a button that will show up on your form. In addition to the common properties, buttons have a few others and some interesting methods associated with them. One such method is the Click method. When we talk about GUI applications, we usually refer to their methods as *events*. Therefore, we will write our code to create and place the button controls on the form in our form's constructor. Then we will write our button's event methods below the constructor. Take the following program as an example. Since our programs are starting to get very long, I will put important new code in **bold** text.

using System;

using System.Windows.Forms;

using System.Drawing;

public class MyForm2 : Form

{

private Label myLabel;

**private Button myButton;**

public static void Main()

{

Application.Run(new MyForm2());

}

public MyForm2()

{

Text = "My New Form";

BackColor = Color.Blue;

ForeColor = Color.White;

myLabel = new Label();

Controls.Add(myLabel);

myLabel.Text = "Don't click the Button!";

myLabel.Top = 10;

myLabel.Left = 75;

myLabel.Width = 200;

**myButton = new Button();**

**Controls.Add(myButton);**

**myButton.Text = "Self Destruct";**

**myButton.Top = 50;**

**myButton.Left = 75;**

**myButton.Click += new System.EventHandler(myButton\_Click);**

}

**private void myButton\_Click(object sender, System.EventArgs e)**

**{**

**myLabel.Text = "Couldn't resist, could you!";**

**}**

}

Again, you can see that we are creating a new private instance variable for our button. Inside the constructor there is code that instantiates this class. I wrote code to set some of the properties of the controls that you should understand at this point. However, I want to bring your attention to the line:

myButton.Click += new System.EventHandler

(myButton\_Click);

This line is setting the Click property of the button using the += operator. This operator is called an *update operator*. It is saying to set the Click property equal to whatever it used to be, plus this new thing. That new thing is an *event handler*. As the name implies, event handlers handle events. So this statement is really just saying to make the Click property be whatever it used to be, plus add an event handler called *myButton\_Click*.

What exactly is the event handler myButton\_Click? Well, as you continue down the code, you will see that it is just a method. This method is written as:

private void myButton\_Click(object sender,

System.EventArgs e)

{

myLabel.Text = "Couldn't resist, could you!";

}

You can see that this method is written just like any other method, with a couple of exceptions. One difference is that we give it private access because we do not want anyone outside the class to have access to it. A second difference is the parameter list. These parameters are standard and must be included in the method header. The programs you write for this course will not use them, but as you write more advanced programs, these parameters come in handy. For now, just be sure to put them in your code. The final difference between this method and the other methods you have seen is how it is called. In fact, we do not write any code to call on this method. Instead, when the button is clicked, the event handler that we created will know to call this method.

Looking inside the method, you can see that all this method does is change the Text property of the Label control. Try typing this program in and run it. It might not be the most useful program in the world, but I think it might be impressive enough to show your friends. After all, this program is a true Windows application that looks fairly standard, and it allows for interaction with the user.

**Text Boxes**

Labels are useful because we can use them to display information to the user. However, there are many times when we want to allow the user to enter text into our program. This can be done with a TextBox control. Once again, a text box has all of those common properties that forms and labels have. Usually when your program first starts, you will not want any text to show in a text box because this control will be used to get information from the user. So, let's write a little program that will use two labels, a text box, and a button.

We will make our first label prompt the user for their name. We will allow them to enter their name in the text box and then give them a button to click when they are ready. Finally, the program will display a personalized message to the user in the second label.

using System;

using System.Windows.Forms;

using System.Drawing;

public class PersonalMessage : Form

{

private Label lblPrompt;

private Label lblOut;

private TextBox txtName;

private Button cmdOk;

public static void Main()

{

Application.Run(new PersonalMessage());

}

public PersonalMessage()

{

Text = "Interactive Hello";

lblPrompt = new Label();

Controls.Add(lblPrompt);

lblPrompt.Text = "Enter your name and click ¬

\'OK\'";

lblPrompt.Top = 10;

lblPrompt.Left = 75;

lblPrompt.Width = 200;

lblOut = new Label();

Controls.Add(lblOut);

lblOut.Text = "";

lblOut.Top = 200;

lblOut.Left = 75;

lblOut.Width = 200;

txtName = new TextBox();

Controls.Add(txtName);

txtName.Text = "";

txtName.Top = 50;

txtName.Left = 75;

txtName.Width = 100;

cmdOk = new Button();

Controls.Add(cmdOk);

cmdOk.Text = "Ok";

cmdOk.Top = 100;

cmdOk.Left = 75;

cmdOk.Click += new System.EventHandler ¬

(cmdOk\_Click);

}

private void cmdOk\_Click(object sender,

System.EventArgs e)

{

lblOut.Text = "Hello " + ¬

txtName.Text;

}

}

Now, I admit these programs are getting longer and longer. I usually find that the length of programs sometimes intimidates my students. I always reassure those people that while there may be a lot of instructions, the computer will just read them one at a time, and I encourage them to do the same. If you look at this code, you will see that there are only a couple new lines of code. Also, many of these lines are very similar to ones we have already seen. Notice that you create the text box control, instantiate it, and set its properties exactly the same as you did for buttons and labels. Really, the only new thing that is done in this example is that in addition to setting properties, we are getting the values stored there. Notice the only line of code in the OK Buttons Click Event:

lblOut.Text = "Hello " + txtName.Text;

This is just a simple assignment statement. Recall way back in the course when you first learned about the assignment statement, I said always translate this as "evaluate the expression on the right and put that value in the variable on the left." That means that this code translates to "concatenate the string 'Hello' and the value of txtName's Text property. Take the result and put it in lblOut's Text property." Easy enough, right?

Go ahead and type this program in and then run it. You will see that, in fact, when you type your first name into the text box and click the button, a personalized message will be shown on the form. But the most interesting part of this program is when you go back into the text box, type your first and last name, and click on the button. Notice that the contents of the output label change with the new information.

This is what makes people love GUI applications so much better. Remember when you first learned how to read input from the console at the beginning of the course? Since you did not know how to write a loop, your program needed to be run a second time if you wanted to type in your last name also. GUI applications do not work this way. Instead, the program is always running and available to run the code until the user chooses to close the application.

One final thing that I must mention about getting information out of the Text property of a text box control, the data type. Text that is contained in the Text property is always a string. Even if the user types in a number, that number will be stored as a string. That means that if you want to store the value in a text box's Text property in a number variable, you will need to convert it to a number using the Convert class. The following code will store the value in the txtAge text box in an int variable named *intAge*:

int intAge = Convert.ToInt32(txtAge.Text);

I show you this because there will be many times when you want to get a number out of a control, and you will need to remember to convert it to a number before doing any mathematical operations on it.  
  
  
  
  
**Chapter 5**

**Summary**

This lesson has given you the knowledge to make attractive, modern GUI applications. You first learned about how to create message boxes to interact with the user. This provided an excellent way to get simple answers from your user, but you were still very limited. The remainder of the lesson showed you how to write a full Windows application that offered more flexibility to interact with your user. You learned about the properties of forms, labels, buttons, and text boxes and the events that are associated with them.

As you continue in programming with C#, you will find that there are many other types of controls that are available to you for use on your forms, like list boxes, combo boxes, check boxes, radio buttons, and many other things that make applications easier for the user to use. Each of these controls has unique properties and events associated with it.

Unfortunately, there just isn't enough time in this course to go over every control that is available. The good news is that now you have a solid background in programming structures to write very useful programs and enough knowledge of controls to go out and learn about the ones not covered here. You can then write applications that are not only useful, but they can be professional looking and easy to use.

In closing, I hope that you enjoyed learning about C# with me in this course. I certainly enjoyed showing you the basic components of the language with simple programs. I also enjoyed having you build your way up to more complex and much more useful programs at the end. I hope you will continue learning more about this wonderful language!

**Final Steps**

|  |
| --- |
| Don't forget that there's a quiz for this last lesson and a final exam for the course. When you feel you're ready to take the final exam, click the **Completion** link at the top or bottom of any page in this classroom. After you've finished taking the final, please don't forget to evaluate the course and obtain your completion letter! |

**Other Online Courses**  
If you enjoyed this course, here are a few other ed2go online courses you may be interested in:

|  |
| --- |
| Intermediate C# Programming  Build upon your current knowledge of programming logic by writing Graphical User Interface applications in the C# programming language. Learn how to write applications with menus, toolbars, and files, and find out how to interact with databases and write SQL queries.  Introduction to Alice 2.0 Programming  Learn how to write three-dimensional animation and game programs while learning object-oriented programming principles at the same time. If you're a novice programmer who wants to master programming concepts like decisions, loops, methods, and arrays, this course is for you! While building fun interactive worlds (including a flight simulator), you'll also learn more advanced programming concepts like arrays and event-driven programming. When you're finished, you'll be able to apply these concepts to other programming languages, like Java, C++, C#, or Visual Basic.  Introduction to Visual Basic .NET  Visual Basic .NET is the most widely used programming language and development tool for creating Windows applications. The Visual Basic .NET programming language is very easy to learn, and the Visual Basic .NET development environment enables you to rapidly build Windows applications with little more than a few clicks of your mouse. By the end of this course, you will be writing your own sophisticated Windows applications. You'll also understand event-driven and object-oriented programming concepts that will help you master both Visual Basic .NET and other programming languages. |

**Supplementary Material**

|  |
| --- |
| [Color Members](http://msdn.microsoft.com/en-us/library/system.drawing.color.aspx)  http://msdn.microsoft.com/en-us/library/system.drawing.color.aspx |
| This page will show you all of the built-in colors that are available to you. In addition, there is a way to create your own color by stating its red, green and blue values. |

**FAQs**   
  
**Q:** Can I make my GUI Applications handle Exceptions the same way as my Console Applications did?  
  
**A:** Yes. Actually, everything that you learned in the course can be applied to GUI Applications. That means that you can make your programs handle Exceptions, use arrays and work with files. It just presents the information to the user in a different way.

**Assignment**   
  
  
For your final assignment, I want you to re-write the previous assignment as a GUI application. In case you have forgotten, here is the problem:

Write a program that will keep track of movies you have seen using files. In addition to storing the name of the movie (a string), also keep track of the movie's rating (int) as a number of stars between 1 and 5. To make your program more flexible, use a file to keep track of the data so that it can be added to in the future.

In the previous lesson you were asked to write a menu system on the Console Window. Now, you can instead create three buttons:

1. Display the movie data
2. Add movie data
3. Quit

When the user chooses to quit, copy the data from the arrays to the data file. To make your program more flexible, be sure to handle any exceptions that may be thrown from invalid input.

If you chose to write a Movie class in the previous lesson, then you should only need to build your interface and make the GUI talk to it.

I want to tell you about a couple of text box properties that will make this program a little easier to write:

1. Multiline. If you make the Height property bigger and correctly set the Multiline property, your text box can display more than one line of text. To set this property, just type:  
     
   <Control Name>.Multiline = true;
2. Scroll Bars. If your text box can show multiple lines, you will want a vertical scroll bar to make it easier for your user to view the text. To set this property, type:  
     
   <Control Name>.ScrollBars = System.Windows.Forms.ScrollBars.Vertical;

[Click here for solution: **MovieGUI.zip**](https://api.ed2go.com/CourseBuilder/2.0/images/resources/prod/cpb-0/MovieGUI.zip)

|  |
| --- |
| Copyright © 1997 - 2012 Education To Go. All rights reserved. The material on this site cannot be reproduced or redistributed unless you have obtained prior written permission from Education To Go. Education To Go and ed2go are registered trademarks of Education To Go, a part of Cengage Learning. |

Bottom of Form