

# 3

## Your First Visual C# Project



### Review and Preview

In the first two classes, you learned about **forms**, **controls**, **properties**, and **event methods**. In this class, you're going to put that knowledge to work in building your first simple Visual C# project. You'll learn the steps in building a project, how to put controls on a form, how to set properties for those controls, and how to write your own event methods using a little C#.

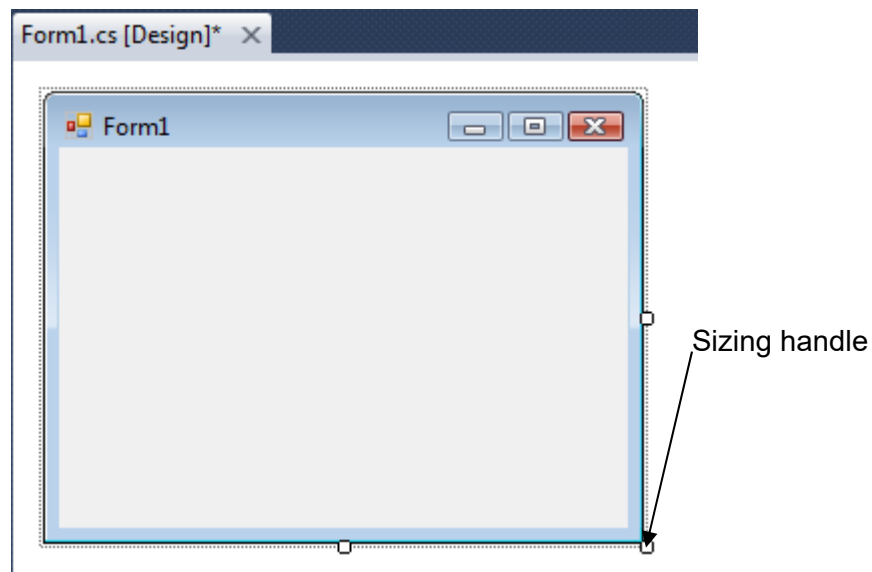
## Steps in Building a Visual C# Project

There are three primary steps in building a Visual C# Project:

1. Place (or draw) **controls** on the form.
2. Assign **properties** to the controls.
3. Write **event methods** for the controls.

Each of these steps is done with Visual C# in **design** mode.

Start Visual C# and start a new project (review the steps covered in Class 2, if necessary, naming it whatever you choose). Open the created form in the **Design** window. You should see something like this:



You can resize the form if you want. This is one of the 'Windows' techniques you should be familiar with. Notice the form has a 'sizing handle' in the lower right corner. If you move the cursor over this handles, a little 'double-arrow' will appear. At that point, you can click and drag the corner to its desired position. This allows you to increase the width and height of the form at the same time. If you hold the cursor over the right or lower edge (until the arrow appears), you can resize the width and height, respectively. Practice sizing the form.

## Placing Controls on the Form

The first step in building a Visual C# project is to place controls on the form in their desired positions. So, at this point, you must have decided what controls you will need to build your project. Many times, this is a time-consuming task in itself. And, I guarantee, you will change your mind many times. Right now, we'll just practice putting controls on the form.

Controls are selected from the Visual C# **Toolbox** window (**Windows Form** controls). Click a tool in the toolbox and hold the mouse button down. Drag the selected tool over to the form. When the cursor pointer is at the desired upper left corner, release the mouse button and the default size control will appear. This is the classic “drag and drop” operation. Once the control is on the form, you can still move or resize the control. To **move** a control, left-click the control to select it (crossed-arrows will appear). Drag it to the new location, then release the mouse button. To **resize** a control, left-click the control so that it is selected. If you move the cursor over one its four sizing handles, a little ‘double-arrow’ will appear. At that point, you can click and drag the corresponding edge or corner to its desired position.

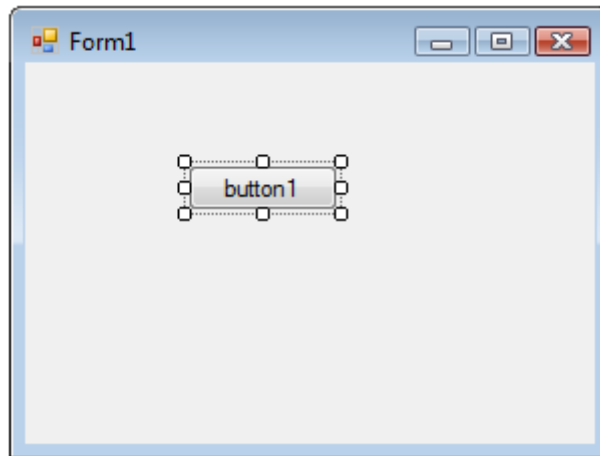
There are other ways to place a control on the form – you will learn them as you progress in your programming skills. One way is to simply double-click the control in the toolbox and it will appear in the upper left corner of the form. We prefer the drag and drop method since the control is placed where you want it.

## Example

Make sure Visual C# is still running and there is a form on the screen as well as the **Toolbox** (click **View** on the main menu, then **Toolbox** if it is not there). Go to the toolbox and find the **button** control. It looks like this:

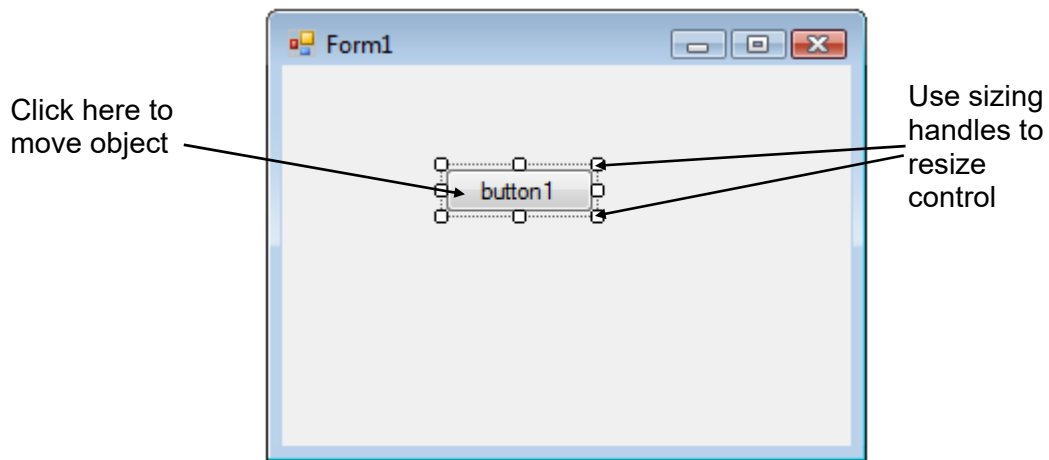


Drag and drop the button onto the form. Your form should look something like this:



Notice the sizing handles around the button. This indicates this is the **active** control. Click on the form and those handles disappear, indicating the form is now the active control. Click on the button again to make it active.

As mentioned, controls can always be moved and resized. To **move** a control you have drawn, click the object on the form (a cross with arrows will appear). Now, drag the control to the new location. Release the mouse button. To **resize** a control, click the control so that it is selected (active) and sizing handles appear. Use these handles to resize the object.



Move the button around and try resizing it. Make a real big button, a real short button, a real wide button, a real tall button. Try moving the button around on the form.

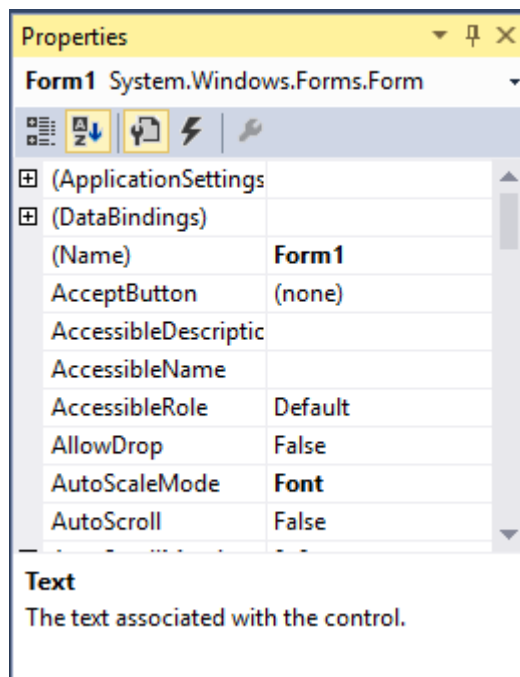
Drag and drop another button control on the form. Move and resize it. Click from button to button noticing the last clicked control has the sizing handles, making it the active control. Spend some time placing controls on the form. Use other controls like labels, text boxes, radio buttons, and check boxes. Move them around, resize them. Try to organize your controls in nicely lined-up groups. These are skills that will be needed in building Visual C# projects.

You also need to know how to remove controls from a form. It is an easy process. Click on the control you want to remove. It will become the active control. Press the **Del** (delete) key on your keyboard. The control will be removed. Before you delete a control, make sure you really want to delete it. Delete any controls you may have placed on the form.

## Setting Control Properties (Design Mode)

Once you have the desired controls on the form, you will want to assign properties to the controls. Recall properties specify how a control appears on the form. They establish such things as control size, color, what a control 'says', and position on the form. When you place a control on the form, it is given a set of default properties by Visual C#. In particular, its geometric properties (governing size and location) are set when you place and size the control on the form. But, many times, the default properties are not acceptable and you will want to change them. This is done using the **Properties Window**.

If Visual C# is not running on your computer, start it now. Start another new project. There should be a blank form in the design window. If it's not there, select the **View** menu and choose **Designer**. Find the **Properties Window** (press <F4> if it's not there):



Click the **Alphabetic** view (the button with A-Z on it) if **Categorized** properties are displayed. Also make sure the **Properties** button, next to the Alphabetic view button is depressed (always make sure this button is pressed when working with properties). Recall the box at the top of the properties window is the **control list**, telling us which controls are present on the form. Right now, the list only has one control, that being the form itself. Let's look at some of the form's properties.

First, how big is the form? All controls are rectangular in shape and two properties define the size of that rectangle. Scroll down the list of properties and find the **Size** property. You will see two numbers listed separated by commas. The first number is the **Width** of the form in pixels (a pixel is a single dot on the form). The second number is the **Height** of the form in pixels. Click on the little plus sign (+) in the box next to the Size property. The Width and Height properties will be displayed individually. Resize the form and notice the Height and Width properties change accordingly. You can also change the width and height of the form by typing in values for the desired property in the Properties window. Try it.

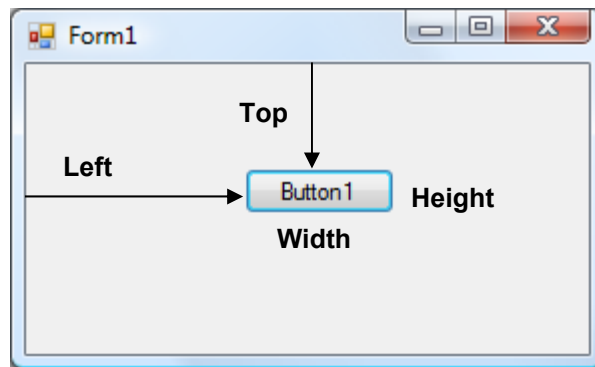
Scroll to the **BackColor** property. You probably guessed that this sets the background color of the form. The value listed for that property is probably **Control** (a light gray). To change the BackColor property, click on BackColor, then on the drop-down arrow that appears in the property side of the list. Choose one of the three 'tabs' that appear: **Custom**, **Web**, or **System**, then choose a color. My favorite is Custom. With this choice, a palette of colors will appear, you can choose a new color and notice the results.

Scroll to the **Text** property. This property establishes what is displayed in the form's title bar. Click on Text, then type in something on the right side of the property window and press **<Enter>**. Notice the new Text appears in the form title bar.



That's all there is to setting control properties. First, select the control of interest from the control list. Then, scroll down through properties and find the property you want to change. Click on that property. Properties may be changed by typing in a new value (like the Width and Height values and the Text property) or choosing from a list of predefined options (available as a drop-down list, like color values).

Let's look at some of the **button** properties. Add a button control to your form. Select the button in the control list of the properties window. Like the form, the button is also rectangular. Scroll down to the **Size** property and click on the little plus (+) sign to expand this property. The **Width** property gives its width in pixels and **Height** gives its height in pixels. Two other properties specify the location of the button on the form. Scroll down to the **Location** property and expand it. Values for **X** (the **Left** property) and **Y** (the **Top** property) are displayed. **Left** gives the horizontal position (in pixels) of the left side of the button relative to the left side of the form. Similarly, **Top** is the vertical position (in pixels) of the top side of the button relative to the top of the form (the top of the form being defined as the lower part of the title bar). For a single button, these properties are:



Another important property for a button is the **Text** property. The text appearing on the button is the Text. It should indicate what happens if you click that button. Change the **Text** property of your button. Put a couple more buttons on the form. Move and size them. Change their Text and BackColor properties, if you want.

We have seen that to change from one control to another in the properties window, we can click on the down arrow in the controls list and pick the desired control. A shortcut method for switching the listed properties to a desired control is to simply click on the control on the form, making it the **active** control. Click on one of the buttons. Notice the selected control in the properties window changes to that control. Click on another button - note the change. Click on the form. The selected control becomes the form. You will find this shortcut method of switching from one control to another very useful as you build your own Visual C# projects.

## Naming Controls

The most important property for any control is its **Name**. Because of its importance, we address it separately. When we name a control, we want to specify two pieces of information: the **type** of control and the **purpose** of the control. Such naming will make our programming tasks much easier.

In the Visual C# programming community, a rule has been developed for naming controls. The first three letters of the control name (called a **prefix**) specify the type of control. Some of these prefixes are (we will see more throughout the class):

<u>Control</u>	<u>Prefix</u>
Button	<b>btn</b>
Label	<b>lbl</b>
Text Box	<b>txt</b>
Check Box	<b>chk</b>
Radio Button	<b>rdo</b>

After the control name prefix, we choose a name (it usually starts with an upper case letter to show the prefix has ended) that indicates what the control does. The complete control name can have up to 40 characters. The name must start with a letter (this is taken care of by using prefixes) and can only contain letters (lower or upper case), numbers, and the underscore (\_) character. Even though you can have 40 character control names, keep the names as short as possible without letting them lose their meaning. This will save you lots of typing.

Let's look at some example control names to give you an idea of how to choose names. These are names used in the **Sample** project looked at in Class 1 and Class 2. Examples:

**btnBeep** - Button that causes a beep

**txtType**- Text box where information could be typed

**rdoBlue** - Radio button that changes background color to Blue

**chkTop** - Check box that displays or hides the toy top

**picTop** – Picture box that has the picture of a toy top

This should give you an idea of how to pick control names. We can't emphasize enough the importance of choosing proper names. It will make your work as a programmer much easier.

It is important to note that the Visual C# language is case sensitive. This means the names **picTop** and **PICTOP** are not treated the same. Make sure you assign unique names to each control. We suggest mixing upper and lower case letters in your control names for improved readability. Just be sure when you type in control names that you use the proper case.

## Setting Properties in Run Mode

To illustrate the importance of proper control names, let's look at a common task in Visual C#. We have seen one of the steps in developing a Visual C# project is to establish control properties in design mode. You can also establish or change properties while your project is in run mode. For example, in the **Sample** project, when you clicked on a radio button, the **BackColor** property of the form was changed. When you clicked on a toy name, that toy either appeared or disappeared. To change a property in run mode, we need to use a line of C# code (you're about to learn your first line of C#!). The format for this code is:

```
controlName.PropertyName = PropertyValue;
```

That is, we type the control's name, a dot (same as a period or decimal point), the name of the property we are changing (found in the properties window), an equal sign (called an assignment operator), and the new value. Such a format is referred to as **dot notation**. Make sure the line ends with a semi-colon (;) – almost every line of code in Visual C# will end with a semi-colon.

In **Sample**, the code used to display the toy top on the form is:

```
picTop.Visible = true;
```

The **Visible** property of a control can be **true** (control is displayed) or **false** (control is not displayed). Notice proper control naming makes this line of code very understandable, even if you don't know any C#. It says that the picture box displaying the top has been made visible.

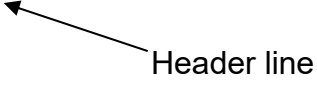
One exception to the rule we just used is when we set **Form** properties. To set a form property at run-time, you use the Visual C# keyword **this** to refer to the form. For example, in Sample, to set the background color of the form to blue, we use:

```
this.BackColor = Color.Blue;
```

## How Control Names are Used in Event Methods

Another place the importance of proper control naming becomes apparent is when we write event methods (discussed next). We have seen that event methods are viewed in the code window. The structure for event methods is:

```
private void controlName_EventName(object sender, EventArgs  
e)  
{  
    [C# code goes here]  
}
```



There's a lot to look at. The first, long line that takes up two lines here (due to margin constraints), is the **header** line. Then the method begins with a left curly brace (`{`) and ends with a right curly brace (`}`). You will see lots of braces in C#. The actual C# code goes between these two braces. Let's look at the header, ignoring the information in parentheses for now. Notice the control name is used as is the event name. Can you see that, with proper naming, we can easily identify each control's event method?

As an example, using **Sample** again, the **CheckedChanged** event method for the **rdoBlue** control is:

```
private void rdoBlue_CheckedChanged(object sender, EventArgs
e)
{
    // change form color to blue
    this.BackColor = Color.Blue;
}
```

We recognize this is the code that is executed when the user changes the **Checked** property (clicks on) of the **rdoBlue** radio button. Proper naming makes identifying and reading event methods very easy. Again, this will make your job as a programmer much easier. Now, let's write our first event method.



## Writing Event Methods

The third step in building a Visual C# application is to write event methods for the controls on the form. To write an event method, we use the code window. Review ways to display the code window in your project. This step is where we need to actually write C# code or do computer programming. You won't learn a lot of C# right now, but just learn the process of finding event methods and typing code.

Each control has many possible events associated with it. You don't write C# code for each event method - only the ones you want the computer to respond to. Once you decide an event is to be 'coded,' you decide what you want to happen in that event method and translate those desires into actual lines of C# code. As seen earlier, the format for each event method is:

```
private void controlName_EventName(object sender, EventArgs
e)
{
    [C# code goes here]
}
```

In the header line (remember it's one long line), the word '**private**' indicates this method is private to the form (only usable by the form - don't worry about what this means right now). The word **void** indicates nothing is being computed by the method. The words enclosed in parentheses tell us what information is provided to the event method. These values are known as the method **arguments** and we won't concern ourselves with them right now. The code goes between the curly braces following this header line.

Writing the C# code is the creative portion of developing a Visual C# application. And, it is also where you need to be very exact. Misspellings, missing punctuation, and missing operators will make your programs inoperable. You will

find that writing a computer program requires exactness. So, the process to write event methods is then:

- Decide which events you want to have some response to
- Decide what you want that response to be
- Translate that response into C# code
- Establish the event method in the code window
- Type in the C# code

And, it is a process best illustrated by example. This example project is saved as **FirstCode** in the course projects folder (**\BeginVCS\BVCS Projects**).

## Example

If Visual C# is not running on your computer, start it and begin a new project. Name it **FirstCode**.

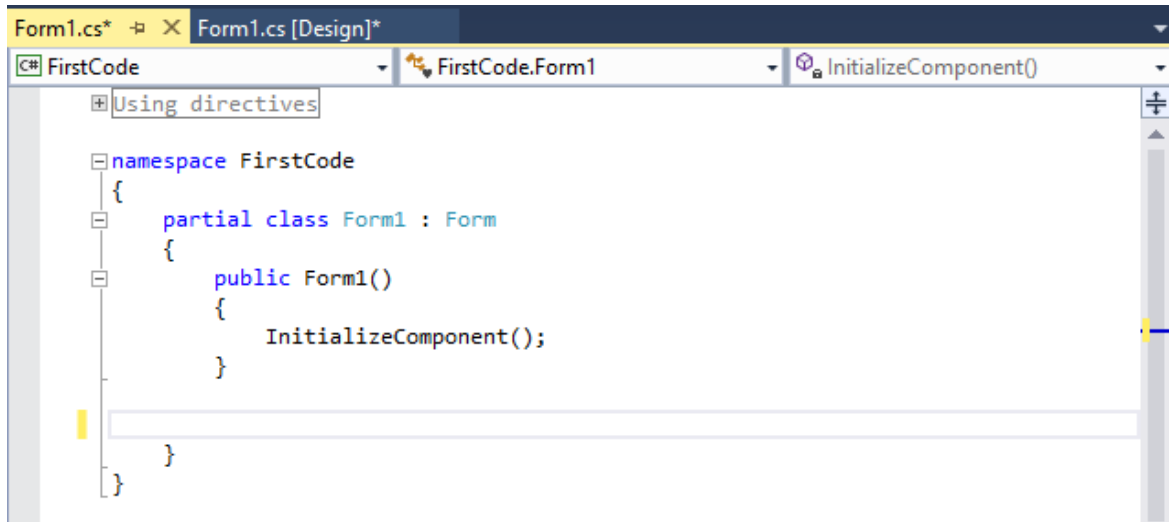
- Put a single button on the form.
- Set the **Text** property of the form to **My First Code**.
- Set the **Name** property of the button to **btnBeep**.
- Set the **Text** property of the button to **Beep!!**

At this point in the design process, your form should look something like this:



We want to write a single event method - the method that responds to the **Click** event of the button. When we click on that button, we want the computer to make a beep sound. Let's look at how to establish the event method.

Display the code window (pressing <F7> is one way; choose **View**, then **Code** in the menu is another):



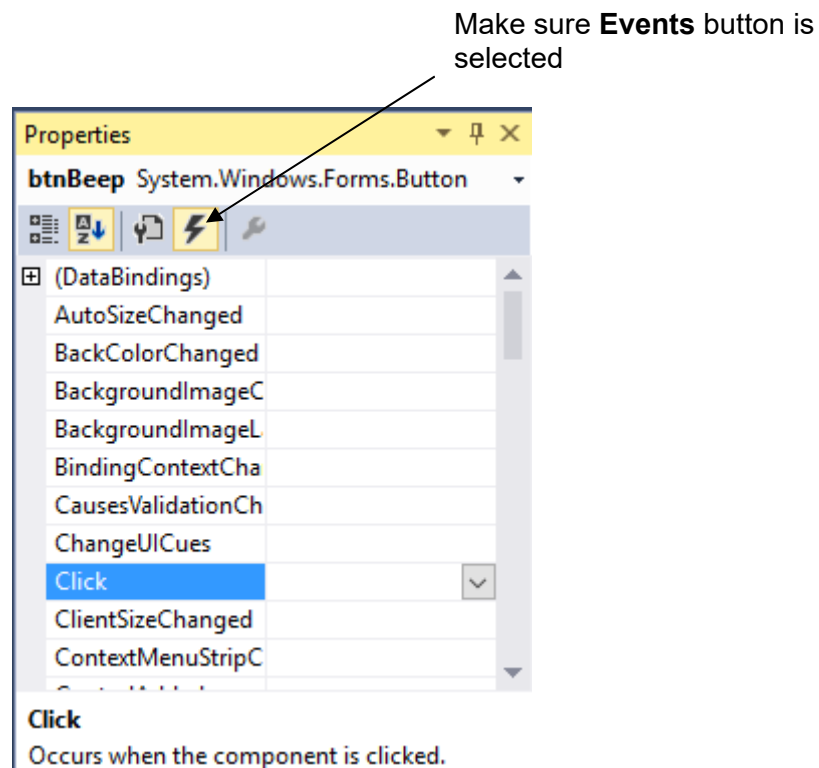
The header line (**namespace FirstCode**) starts the code. In Visual C#, everything that makes up your project is called a **namespace**. Your form is called a **class**. The line **public Form1()** begins the form **constructor**. The constructor consists of a single line saying **InitializeComponent()**;. This code accesses a routine written by the Visual C# environment to set up the form that you designed. Notice again the use of curly braces to start and end code segments. You don't have to worry much about any of this code – just don't change any of it. The only code we will change is associated with event methods we establish and write. Let's do that now for the button control.

Recall when we looked at the properties window, we mentioned that, in addition to establishing control properties, that window is also used to establish event methods. Be aware that since the properties window has these two purposes, you should always be aware whether the **Properties** or **Events** button is selected in the window's toolbar. The steps to establish a blank event method for a particular control are:

- View the application form in design mode.
- Make desired control active, so its name appears at top of properties window.
- Go to properties window – select **Events** button (looks like a lightning bolt).
- Find event of interest.
- Double-click the event name.

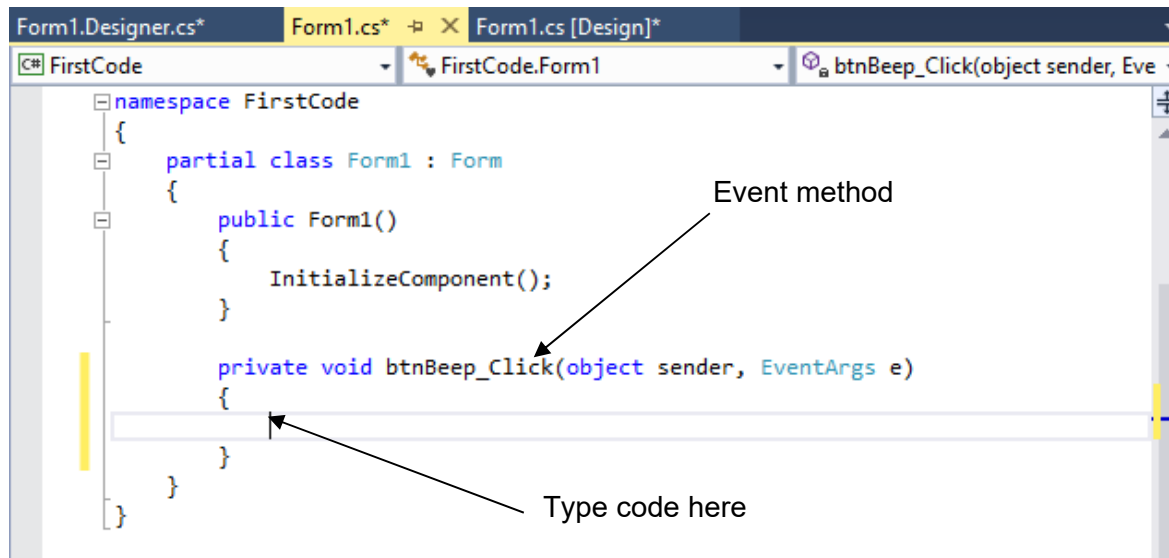
At this point, the code window will open displaying the newly formed event method. Let's follow these steps for our button control.

We want to write code for the button control **Click** event. Display the form design window. Select the button control (**btnBeep**) in the properties window drop-down box (or click the button control on the form) to make it active. Scroll down the events list and highlight the Click event:



Any name already assigned to the **Click** event method would be listed on the right side of the properties window. There should be no name in that area – a name will be automatically assigned. Double-click the word **Click** in the properties window.

The code window should open and appear as:

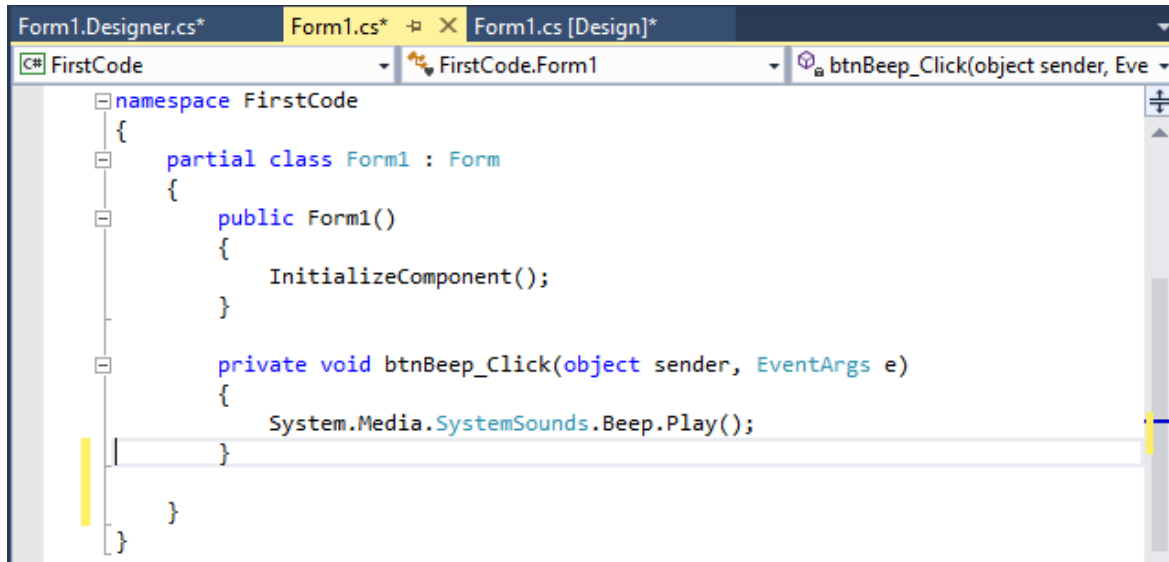


Notice the **Click** method for the **btnBeep** button is now displayed under the form constructor code. This is where all event methods will appear. If you return to the properties window, the method name **btnBeep\_Click** will appear next to the Click event. We type the code to make the computer beep between the two curly braces following the method header line.

The code window acts like a word processor. You can type text in the window and use many of the normal editing features like cut, paste, copy, find, and replace. As you become a more proficient programmer, you will become comfortable with using the code window. Click on the region between the two braces. Type the single line exactly as shown:

```
System.Media.SystemSounds.Beep.Play();
```

The code window should now look like this:



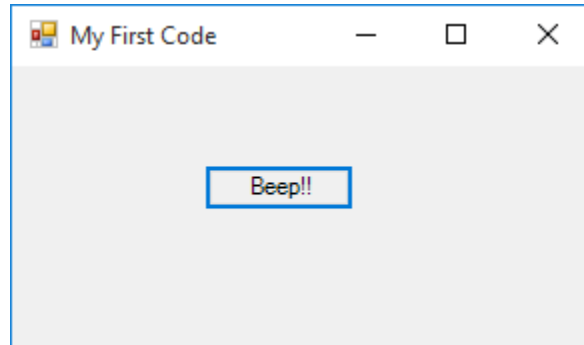
Notice after you typed the line, it was indented and parentheses were added at the end (indicating this is a built-in function). The Visual C# environment does this additional 'formatting.' The long line of code:

```
System.Media.SystemSounds.Beep.Play();
```

is a C# instruction that simply tells the computer to beep. You have now written your first line of C# code.



Your project is now ready to run. **Run** the project (click the **Start** button on the toolbar or press <F5>). The form will appear:



(If it doesn't, go back and make sure you've done all steps properly). Click the button. The computer should beep or some sound like a beep should be heard. You caused a **Click** event on the **btnBeep** control. The computer recognized this and went to the **btnBeep\_Click** event method. There it interpreted the line of code [Console.Beep();] and made the computer beep. Stop your project. Go back to the code window and find the **btnBeep\_Click** event. After the 'beep' line, add this line:

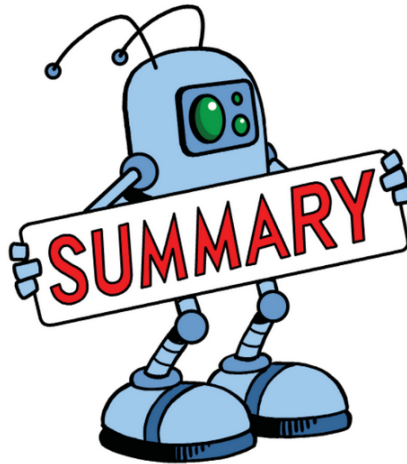
```
btnBeep.BackColor = Color.Blue;
```

Make sure you type it in exactly as shown, paying particular attention to letter case – code in computer programs must be exact. Run the project again. Click on the button. Explain what happens in relation to the control, the event method, and the C# code. Stop your project.

You may have noticed when you added this second line of code that as soon as you typed **btnBeep**, then a dot, a little window popped up with lots of choices for completing the line (**BackColor** was one of them). Similarly, once you typed **Color**, then a dot, a choice of colors (including **Blue**) popped up. This is the Visual C# **Intellisense** feature (you probably also noticed it when typing the code

to make the computer beep). It helps a lot when it comes to typing code.

Intellisense is a very useful part of Visual C#. You should become acquainted with its use and how to select suggested values. You usually just scroll down the list (you can type the first few letters of a choice for faster scrolling), pick the desired item and continue typing. The choice will be inserted in the proper location. We tell you about the Intellisense feature now so you won't be surprised when little boxes start popping up as you type code.



You have now finished your first complete Visual C# project. You followed the three steps of building an application:

1. Place controls on the form
2. Assign control properties
3. Write control event methods

You follow these same steps, whether building a very simple project like the one here or a very complicated project.

Now, knowing these steps, you're ready to start working your way through the Visual C# toolbox, learning what each control does. You can now begin learning elements of the C# language to help you write programs. And, you can begin learning new features of the Visual C# environment to aid you in project development. In each subsequent class, you will do just that: learn some new controls, learn some C#, and learn more about Visual C#.

