4

Project Design, Forms, Buttons

Review and Preview

You have now learned the parts of a Visual C# project and the three steps involved in building a project:

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

Do you have some ideas of projects you would like to build using Visual C#? If so, great. Beginning with this class, you will start to develop your own programming skills. In each class to come, you will learn some new features of the Visual C# environment, some new controls, and elements of the C# language. In this class, you will learn about project design, the form and button controls, and build a complete project.

Project Design

You are about to start developing projects using Visual C#. We will give you projects to build and maybe you will have ideas for your own projects. Either way, it's fun and exciting to see ideas end up as computer programs. But before starting a project, it's a good idea to spend a little time thinking about what you are trying to do. This idea of proper **project design** will save you lots of time and result in a far better project.

Proper project design is not really difficult. The main idea is to create a project that is easy to use, easy to understand, and free of errors. That makes sense, doesn't it? Spend some time thinking about everything you want your project to do. What information does the program need? What information does the computer determine? Decide what controls you need to use to provide these sets of information. Design a nice user interface (interface concerns placement of controls on the form). Consider appearance and ease of use. Make the interface consistent with other Windows applications, if possible. Familiarity is good in Windows based projects, like those developed using Visual C#.

Make the C# code in your event methods readable and easy to understand. This will make the job of making later changes (and you will make changes) much easier. Follow accepted programming rules - you will learn these rules as you learn more about C#. Make sure there are no errors in your project. This may seem like an obvious statement, but many programs are not error-free. The Windows operating system has many errors floating around!

The importance of these few statements about project design might not make a lot of sense right now, but they will. The simple idea is to make a useful, clearly written, error-free project that is easy to use and easy to change. Planning carefully and planning ahead helps you achieve this goal. For each project built in this course, we will attempt to give you some insight into the project design process. We will always try to explain why we do what we do in building a project. And, we will always try to list all the considerations we make.

Saving a Visual C# Project

When a project is created in Visual C#, it is automatically saved in the location you specify. If you are making lots of changes, you might occasionally like to save your work prior to running the project. Do this by clicking the **Save All** button in the Visual C# toolbar. Look for a button that looks like several floppy disks. (How much longer do you think people will know what a floppy disk looks like? – most new machines don't even have a floppy disk drive!)

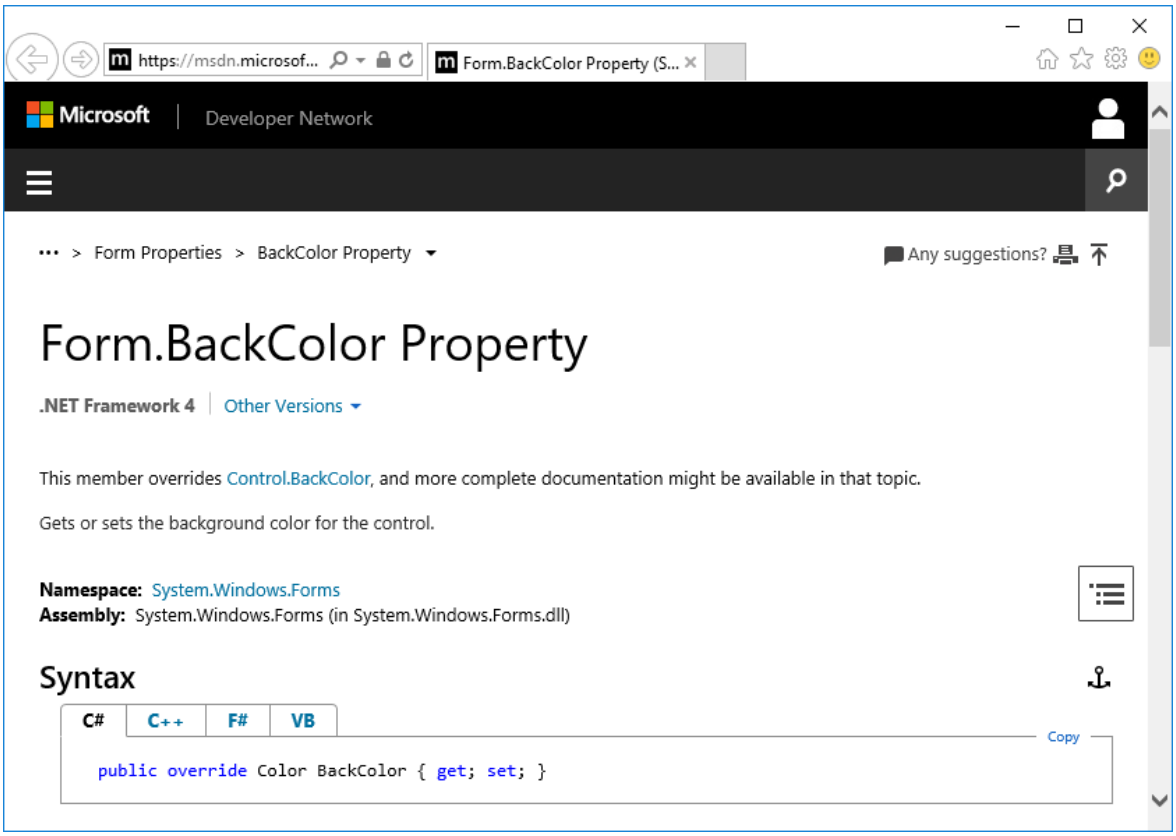
Always make sure to save your project before running it or before leaving Visual C#.

On-Line Help

Many times, while working in the Visual C# environment, you will have a question about something. You may wonder what a particular control does, what a particular property is for, what events a control has, or what a particular term in C# means. A great way to get help when you're stuck is to ask someone who knows the answer. People are usually happy to help you - they like the idea of helping you learn. You could also try to find the answer in a book and there are lots of Visual C# books out there! Or, another great way to get help is to use the Visual C# **On-Line Help** system.

Most Windows applications, including Visual C#, have help files available for your use. To access the Visual C# help system, click the **Help** item in the main menu, then **Contents**. At that point, you can search for the topic you need help on or scroll through all the topics. The Visual C# help system is just like all other Windows help systems. If you've ever used any on-line help system, using the system in Visual C# should be easy. If you've never used an on-line help system, ask someone for help. They're pretty easy to use. Or, click on **Start** on your Windows task bar, then choose **Help**. You can use that on-line help system to learn about how to use an on-line help system!

A great feature about the Visual C# on-line help system is that it is 'context sensitive.' What does this mean? Well, let's try it. Start Visual C# and start a new project. Go to the properties window. Scroll down the window displaying the form properties and click on the word **BackColor**. The word is highlighted. Press the **<F1>** key. A screen of information about the **Form.BackColor** property appears:

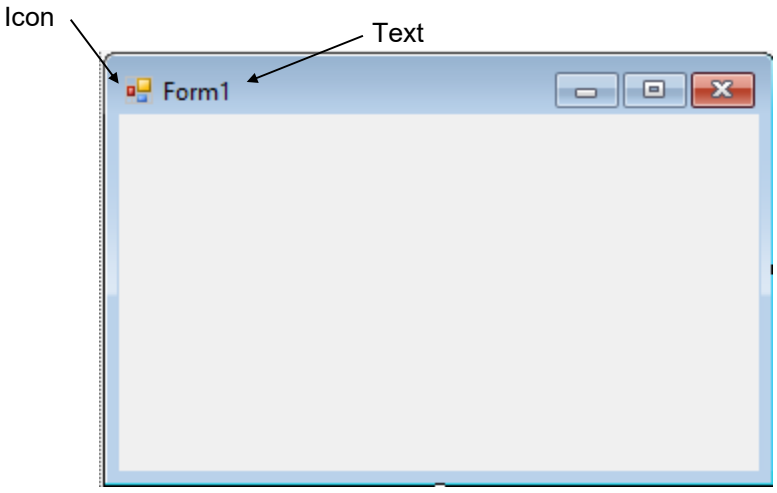


The help system has intelligence. It knows that since you highlighted the word BackColor, then pressed **<F1>** (**<F1>** has always been the key to press when you need help), you are asking for help about BackColor. Anytime you press **<F1>** while working in Visual C#, the program will look at where you are working and try to determine, based on context, what you are asking for help about. It looks at things like highlighted words in the properties window or position of the cursor in the code window.

As you work with Visual C#, you will find you will use ‘context-sensitive’ help a lot. Many times, you can get quick answers to questions you might have. Get used to relying on the Visual C# on-line help system for assistance. That’s enough new material about the Visual C# environment. Now, let’s look, in detail, at two important controls: the form itself and the button. Then we’ll start our study of the C# language and build a complete project.

The Form Control

We have seen that the **form** is the central control in the development of a Visual C# project. Without a form, there can be no project! Let's look at some important properties and events for the form control. The form appears when you begin a new project.



Properties

Like all controls, the form has many (over 40) properties. Fortunately, we only have to know about some of them. The properties we will be concerned with are:

Property	Description
Name	Name used to identify form. In this course, we will always use the default **Form1** for the name.
Text	Text that appears in the title bar of form.
BackColor	Background color of form.
Icon	Reference to icon that appears in title bar of form (we'll look at creating icons in Class 7).

Width	Width of the form in pixels (expand **Size** property)
Height	Height of form in pixels (expand **Size** property)
FormBorderStyle	Form can either be sizable (can resize using the mouse) or fixed size.
StartPosition	Determines location of form on computer screen when application begins (we usually use a value of **CenterScreen**).

The form is primarily a 'container' for other controls. Being a container means many controls (the button control, studied next, is an exception) placed on the form will share the **BackColor** property. To change this behavior, select the desired control (after it is placed on the form) and change the color.

Example

To gain familiarity with these properties, start Visual C# and start a new project with just a form. Set the Height and Width property values (listed under **Size** in the properties window) and see their effect on form size. Resize the form and notice how those values are changed in the properties window. Set the Text property. Pick a new background color using the selection techniques discussed in Class 3. Try centering the form by changing the StartPosition property. To see the effect of the BorderStyle property, set a value (either **Fixed Single** or **Sizable;** these are the only values we'll use in this course) and run the project. Yes, you can run a project with just a form as a control! Try resizing the form in each case. Note the difference. Stop this example project.

Events

The form does support events. That is, it can respond to some user interactions. We will only be concerned with three form events in this course:

Event	Description
Click	Event executed when user clicks on the form with the mouse.
Load	Event executed when the form first loads into the computer's memory. This is a good place to set initial values for various properties and other project values.
FormClosing	Event called when the project is ending. This is a good place to 'clean up' your project.

Recall, to create any corresponding event method, make the form the active control, choose Events in the properties window, then double-click the name of the event. The event method will appear in the code window. To view an 'already-created' method in the code window, use the Methods List drop-down box at the top of the code window.

Typical Use of Form Control

For each control in this, and following chapters, we will provide information for how that control is typically used. The usual design steps for a **Form** control are:

- Set the **Text** property to a meaningful title.
- Set the **StartPosition** property (in this course, this property will almost always be set to **CenterScreen**)
- Set the **FormBorderStyle** to some value. In this course, we will mostly use **FixedSingle** forms.
- Write any needed initialization code in the form's **Load** event.
- Write any needed finalization code in the form's **FormClosing** event.

Button Control

The **button** is one of the more widely used Visual C# controls. Buttons are used to start, pause, or end particular processes. The button is selected from the toolbox. It appears as:

In Toolbox:

On Form (default properties):

Properties

A few useful properties for the button are:

Property	Description
Name	Name used to identify button. Three letter prefix for button names is **btn**.
Text	Text (caption) that appears on the button.
TextAlign	How the caption text is aligned on the button.
Font	Sets style, size, and type of caption text.
BackColor	Background color of button.
ForeColor	Color of text on button.
Left	Distance from left side of form to left side of button (referred to by **X** in properties window, expand **Location** property).
Top	Distance from top side of form to top side of button (referred to by **Y** in properties window, expand **Location** property).

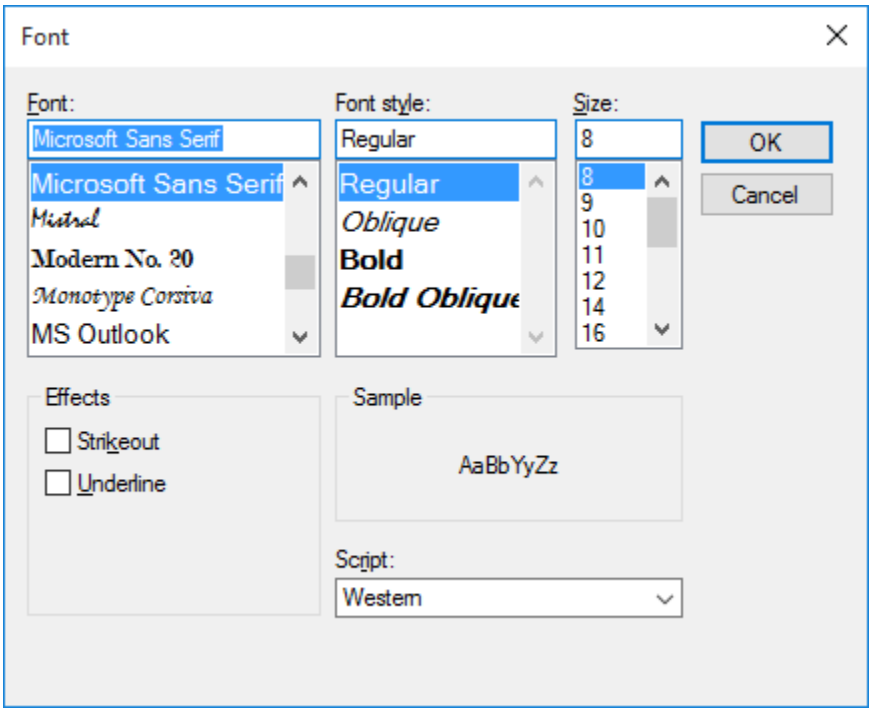
Width	Width of the button in pixels (expand **Size** property).
Height	Height of button in pixels (expand **Size** property).
Enabled	Determines whether button can respond to user events (in run mode).
Visible	Determines whether the button appears on the form (in run mode).

Example

Start Visual C# and start a new project. Put a button on the form. Move the button around and notice the changes in X and Y properties (listed under **Location** in the properties window). Resize the button and notice how Width and Height change. Set the Text property. Change BackColor and ForeColor properties.

Many controls, in addition to the button, have a Font property, so let's take a little time to look at how to change it. Font establishes what the Text looks like. When you click on Font in the properties window, a button with something called an **ellipsis** will appear on the right side of the window:

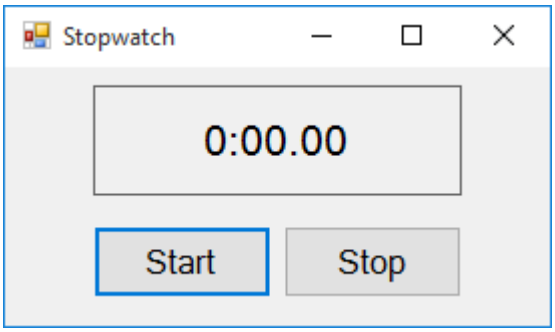
Click this button and a **Font Window** will appear:



With this window, you can choose three primary pieces of information: **Font**, **Font Style**, and **Size**. You can also have an underlined font. This window lists information about all fonts stored on your computer. To set the Font property, make your choices in this window and click **OK**. Try different fonts, font styles, and font size for the button Text property.

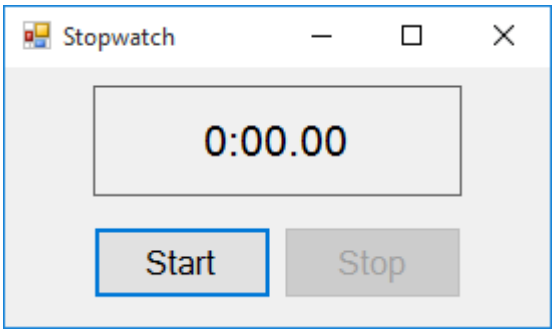
Two other properties listed for the button are Enabled and Visible. Each of these properties can either be **True** (On) or **False** (Off). Most other controls also have these properties. Why do you need these?

If a control's Enabled property is False, the user is unable to access that control. Say you had a stopwatch project with a Start and Stop button:



You want the user to click Start, then Stop, to find the elapsed time. You wouldn't want the user to be able to click the Stop button before clicking the Start button. So, initially, you would have the Start button's Enabled property set to True and the Stop button's Enabled property set to False. This way, the user can only click Start. Once the user clicked Start, you would swap property values. That is, make the Start button's Enabled property False and the Stop button's Enabled property True. That way, the user could now only click Stop.

The effects of a False Enabled property are only evident when Visual C# is in run mode. When a button is not Enabled (Enabled is False), it will appear 'hazy' and the user won't be able to click it. When Stop is not Enabled on the stopwatch, it looks like this:



So, use the Enabled property when you want a control on the form to be temporarily disabled. This is a decision made in the project design process we discussed earlier.

The Visible property is a bit more drastic. When a control's Visible property is set to False (its default value is True), the control won't even be on the form! Now, why would we want a control we just placed on the form, set properties for, and wrote event methods for, to be invisible? The answer is similar to that for the Enabled property. Many times in a project, you will find you want a control to temporarily go away. Remember the **Sample** project in Class 1 where check boxes controlled whether toys were displayed or not. The display of the toys was controlled via the picture box control's Visible property. Or, in the little stopwatch example, instead of setting a button's Enabled property to False to make it 'unclickable,' we could just set the Visible property to False so it doesn't appear on the form at all. Either way, you would obtain the desired result. This is another project design decision. One more thing - like the Enabled property, the effects of Visible being False are only evident in run mode. This makes sense. It would be hard to design a project with invisible controls!

Now, play with the Enabled and Visible properties of the button in the example you have been working with. Once you set either property, run the project to see the results. Note with Enabled set to False, you can't click the button. Note with Visible set to False, the button isn't there. When done, stop the example project.

Events

There is only one button event of interest, but it is a very important one:

Event	Description
Click	Event executed when user clicks on the button with the mouse.

Every button will have an event method corresponding to the Click event.

Typical Use of Button Control

The usual design steps for a button control are:

- Set the **Name** and **Text** property.
- Write code in the button's **Click** event.
- You may also want to change the **Font**, **Backcolor** and **Forecolor** properties.

C# - The First Lesson

At long last, we are ready to get into the heart of a Visual C# project - the C# language. You have seen that, in a Visual C# project, event methods are used to connect control events to actual actions taken by the computer. These event methods are written using C#. So, you need to know C# to know Visual C#. In each subsequent class in this course, you will learn something new about the C# language.

Event Method Structure

You know, by now, that event methods are created using the properties window and viewed in the Visual C# code window. Each event method has the same general structure. First, there is a **header** line of the form:

```
private void controlName_EventName(object sender, EventArgs
e)
```

This tells us we are working with a **private** (only accessible from our form) method, returning no information (word **void**), that is executed when the event **EventName** occurs for the control **controlName**. Makes sense, doesn't it? Again, for now we will ignore the information contained in the parentheses.

The event method code is enclosed in a matched set of curly braces that follow the header line. The event method code is simply a set of line-by-line instructions to the computer, telling it what to do. The computer will process the first line, then the second, then all subsequent lines.

Some C# Programming Rules

The event method code is written in the C# language. C# is a set of keywords and symbols that are used to make the computer do things. There is a lot of content in C# and we'll try to look at much of it in this course. Just one warning at this point. We've said it before, but it's worth saying again. Computer programming requires exactness - it does not allow errors! The Visual C# environment can point out some errors to you, but not all. You must especially be exact when typing in event methods. Good typing skills are a necessity in the computer age. As you learn Visual C# programming, you might like also to improve your typing skills using some of the software that's available for that purpose. The better your typing skills, the fewer mistakes you will make in building your Visual C# applications.

Here are some rules to follow as you type in your C# code:

- C# code requires perfection. All keywords must be spelled correctly. If you type **BckColor** instead of **BackColor**, a human may know what you mean, but a computer won't.
- C# is case-sensitive, meaning upper and lower case letters are considered to be different characters. When typing code, make sure you use upper and lower case letters properly. In C#, the words **Blue** and **blue** are completely different.
- Curly **braces** are used for grouping. They mark the beginning and end of programming sections. Make sure your C# code has an equal number of left and right braces. We call the section of code between matching braces a **block**.

- It is good coding practice to **indent** code within a block. This makes code easier to follow. Notice in examples we've seen, each block is indented 4 spaces. The Visual C# editor automatically indents code in blocks for you.
- Every C# statement will end with a semicolon. A **statement** is a program expression that generates some action (for example, the statement used to make the computer beep in the previous class). Note that not all C# expressions are statements (for example, the line defining an event method has no semicolon).

We'll learn a lot more C# programming rules as we progress.

Assignment Statement

The simplest, and most used, statement in C# is the **assignment** statement. It has this form:

```
leftSide = rightSide;
```

The symbol **=** is called the **assignment operator**. You may recognize this symbol as the equal sign you use in arithmetic, but it's not called an equal sign in computer programming. Why is that?

In an assignment statement, we say whatever is on the left side of the assignment statement is replaced by whatever is on the right side. The left side of the assignment statement can only be a single term, like a control property. The right side can be just about any legal C# expression. It might have some math that needs to be done or something else that needs to be evaluated. If there are such evaluations, they are completed before the assignment. We are talking in very general terms right now and we have to. The idea of an assignment statement will become very obvious as you learn just a little more C#.

Property Types

Recall a property describes something about a control: size, color, appearance. Each **property** has a specific **type** depending on the kind of information it represents. When we use the properties window to set a value in design mode, Visual C# automatically supplies the proper type. If we want to change a property in an event method using the C# assignment statement, we must know the property type so we can assign a properly typed value to it. Remember we use something called 'dot notation' to change properties in run mode:

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

controlName is the Name property assigned to the control, PropertyName is the property name, and PropertyValue is the new value we are assigning to PropertyName. We will be concerned with four property types.

The first property type is the **int** (stands for integer) type. These are properties that are represented by whole, non-decimal, numbers. Properties like the **Top**, **Left**, **Height**, and **Width** properties are integer type. So, if we assign a value to an integer type property, we will use integer numbers. As an example, to change the width property of a form to 1,100 pixels, we would write in C#:

```
this.Width = 1100;
```

Recall the keyword **this** is used to refer to the form. This says we replace the current Width of the form with the new value of 1100. Notice you write 1,100 as 1100 in C# - we can't use commas in large numbers.

A second property type involves **colors**. We need this to set properties like BackColor. Fortunately, Visual C# has a set of built-in colors to choose from. To set a control color (described by **ColorPropertyName**), we type:

```
controlName.ColorPropertyName = Color.ColorName;
```

As soon as we type the word **Color** and a dot on the right side of the assignment statement, a entire list of color names to choose from magically appears. To change the form background color to blue, use:

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

Another property type is the **bool** (stands for Boolean) type. It takes its name from a famous mathematician (Boole). It can have two values: **true** or **false**. We saw that the Enabled and Visible properties for the button have Boolean values. So, when working with Boolean type properties, we must insure we only assign a value of true or a value of false. To make a form disappear (not a very good thing to do!), we would use the assignment statement:

```
this.Visible = false;
```

This says the current Visible property of the form is replaced by the Boolean value false. We could make it come back with:

```
this.Visible = true;
```

There is one possible point of confusion when working with Boolean values. When setting a Boolean value using the properties window, the two choices are **True** or **False**. When setting Boolean values using C# code, the two choices are

true or **false**. That is, in one case, upper case words are used and in the other, lower case words are used. Be aware of this when writing code. A common error is to use upper case values, rather than the proper lower case values.

The last property type we need to look at here is the **string** type. Properties of this type are simply what the definition says - strings of characters. A string can be a name, a string of numbers, a sentence, a paragraph, any characters at all. And, many times, a string will contain no characters at all (an empty string). The Text property is a string type property. We will do lots of work with strings in Visual C#, so it's something you should become familiar with. When assigning string type properties, the only trick is to make sure the string is enclosed in quotes (**"**). You may tend to forget this since string type property values are not enclosed in quotes in the properties window. To give our a form a caption in the title bar, we would use:

```
this.Text = "This is a caption in quotes";
```

This assignment statement says the Text property of the form is replaced by (or changed to) the string value on the right side of the statement. You should now have some idea of how assignment statements work.

Comments

When we talked about project design, it was mentioned that you should follow proper programming rules when writing your C# code. One such rule is to properly comment your code. You can place non-executable statements (ignored by the computer) in your code that explain what you are doing. These **comments** can be an aid in understanding your code. They also make future changes to your code much easier.

To place a comment in your code, use the comment symbol, two slashes (//). Anything written after the comment symbol will be ignored by the computer. You can have a comment take up a complete line of C# code, like this:

```
// Change form to blue
this.BackColor = Color.Blue;
```

Or, you can place the comment on the same line as the assignment statement:

```
this.BackColor = Color.Blue; // Makes form blue
```

You, as the programmer, should decide how much you want to comment your code. We will try in the projects provided in this course to provide adequate comments. Now, on to the first such project.

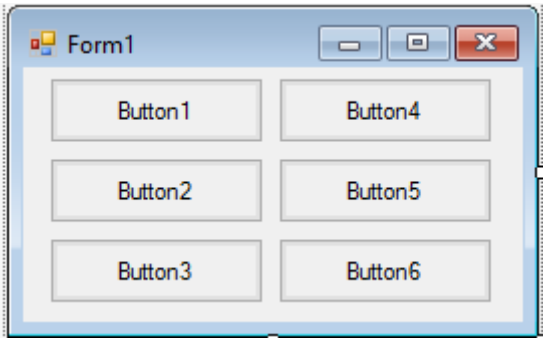
Project - Form Fun

Project Design

In this project, we will have a little fun with form properties using buttons. We will have a button that makes the form grow, one that makes the form shrink, and two buttons that change the form color. We'll even have a couple of buttons that make the other buttons disappear and reappear. This project is saved as **FormFun** in the course projects folder (**\BeginVCS\BVCS Projects**).

Place Controls on Form

Start a new project in Visual C#. Size the form so six buttons will fit on the form. Place six buttons on the form. Resize and move the buttons around until the form looks something like this:



If you've used Windows applications for a while, you have probably used the edit feature known as **Copy** and **Paste**. That is, you can copy something you want to duplicate, move to the place you want your copy and then paste it. This is something done all the time in word processing. You may have discovered, in playing around with Visual C#, that you can copy and paste controls. Try it here with the button controls and in other projects if you like. It works pretty nicely.

Set Control Properties

Set the control properties using the properties window. Remember that to change the selected control in the properties window, you can either use the controls list at the top of the window or just click on the desired control. For project control properties, we will always list controls by their default names (those assigned by Visual C# when the control is placed on the form).

Form1 Form:

Property Name	Property Value
StartPosition	CenterScreen
Text	Form Fun

button1 Button:

Property Name	Property Value
Name	btnShrink
Text	Shrink Form

button2 Button:

Property Name	Property Value
Name	btnGrow
Text	Grow Form

button3 Button:

Property Name	Property Value
Name	btnHide
Text	Hide Buttons

button4 Button:

Property Name	Property Value
Name	btnRed
Text	Red Form

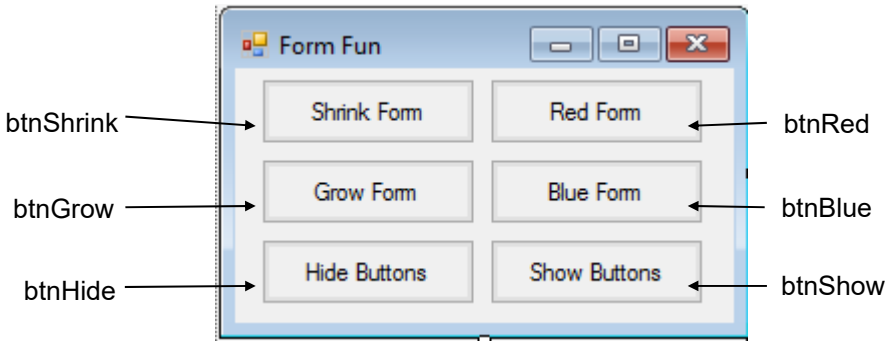
button5 Button:

Property Name	Property Value
Name	btnBlue
Text	Blue Form

button6 Button:

Property Name	Property Value
Name	btnShow
Text	Show Buttons
Visible	False

You can change other properties if you want - maybe change the Font property of the buttons. When you're done setting properties, your form should resemble this:



What we have are six buttons, two to change the size of the form, two to change form color, one to make buttons go away, and one to make buttons reappear. Notice the **Show Buttons** button has a Visible property of False. We don't want it on the form at first, since the buttons will already be there. When we make the buttons go away (by changing their Visible property) by clicking the **Hide Buttons** control, we will make the **Show Buttons** button appear. Makes sense, doesn't it? But, why is the **Show Buttons** button there if its Visible property is False? Remember a False Visible property will only be seen in run mode.

Write Event Methods

We have six buttons on our form. We need to write code for the **Click** event method for each of these buttons. We'll also want to write a **Click** event method for the form - we'll explain why. We have a button on the form that makes the form shrink. What if we shrink it so much, we can't click on the button to make it grow again? We can avoid that by allowing a click on the form to also grow the form. This 'thinking ahead' is one of the project design concepts we talked about.

Each event method is created using the properties window. To create the method, select the desired control on the form and go to the properties window. Click the **Events** button in the toolbar, scroll down to the desired event (**Click** in each case here) and double-click the event name. The code window will open with the method displayed. Then click in the region between the curly braces following the header line and start typing code. It's that easy. But, again, make sure you type in everything just as written in these notes. You must be exact!

First, let's type the **btnShrink_Click** event method. In this event method, we decrease the form height by 10 pixels and decrease the form width by 10 pixels:

```
private void btnShrink_Click(object sender, EventArgs e)
{
    // Shrink the form
    // Decrease the form height by 10 pixels
    this.Height = this.Height - 10;
    // Decrease the form width by 10 pixels
    this.Width = this.Width - 10;
}
```

Before looking at the other event methods, let's look a bit closer at this one since it uses a few ideas we haven't clearly discussed. This is the event method executed when you click on the button marked **Shrink Form**. You should easily recognize

the comment statements. The non-comment statements change the form height and width. Look at the statement to change the height:

```
this.Height = this.Height – 10;
```

Recall how the assignment operator (=) works. The right side is evaluated first. So, 10 is subtracted (using the - sign) from the current form height. That value is assigned to the left side of the expression, this.Height. The result is the form Height property is replaced by the Height property minus 10 pixels. After this line of code, the Height property has decreased by 10 and the form will appear smaller on the screen.

This expression also shows why we call the assignment operator (=) just that and not an equal sign. Anyone can see the left side of this expression cannot possibly be equal to the right side of this expression. No matter what this.Height is, the right side will always be 10 smaller than the left side. But, even though this is not an equality, you will often hear programmers read this statement as “this.Height equals this.Height minus 10,” knowing it’s not true! Remember how assignment statements work as you begin writing your own programs.

Now, let’s look at the other event methods. The **btnGrow_Click** event method increases form height by 10 pixels and increases form width by 10 pixels:

```
private void btnGrow_Click(object sender, EventArgs e)
{
    // Grow the form
    // Increase the form height by 10 pixels
    this.Height = this.Height + 10;
    // Increase the form width by 10 pixels
    this.Width = this.Width + 10;
}
```

The **btnRed_Click** event method changes the form background color to red:

```
private void btnRed_Click(object sender, EventArgs e)
{
    // Make form red
    this.BackColor = Color.Red;
}
```

while the **btnBlue_Click** event method changes the form background color to blue:

```
private void btnBlue_Click(object sender, EventArgs e)
{
    // Make form blue
    this.BackColor = Color.Blue;
}
```

The **btnHide_Click** event method is used to hide (set the **Visible** property to **false**) all buttons except **btnShow**, which is made **Visible** (note the use of lower case **true** and **false** when writing code):

```
private void btnHide_Click(object sender, EventArgs e)
{
    // Hide all buttons but btnShow
    btnGrow.Visible = false;
    btnShrink.Visible = false;
    btnHide.Visible = false;
    btnRed.Visible = false;
    btnBlue.Visible = false;
    // Show btnShow button
    btnShow.Visible = true;
}
```

and the **btnShow_Click** event method reverses these effects:

```
private void btnShow_Click(object sender, EventArgs e)
{
    // Show all buttons but btnShow
    btnGrow.Visible = true;
    btnShrink.Visible = true;
    btnHide.Visible = true;
    btnRed.Visible = true;
    btnBlue.Visible = true;
    // Hide btnShow button
    btnShow.Visible = false;
}
```

Lastly, the **Form1_Click** event method is also used to 'grow' the form, so it has the same code as **btnGrow_Click**:

```
private void Form1_Click(object sender, EventArgs e)
{
    // Grow the form
    // Increase the form height by 10 pixels
    this.Height = this.Height + 10;
    // Increase the form width by 10 pixels
    this.Width = this.Width + 10;
}
```

Save your project by clicking the **Save All** button (the multiple floppy disks button) in the toolbar.

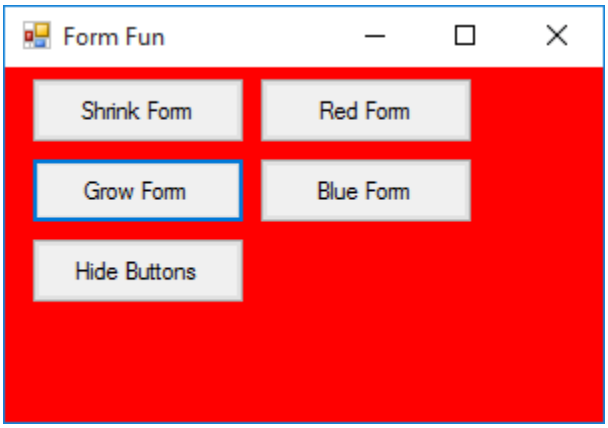
You should easily be able to see what's going on in each of these methods. Pay special attention to how the Visible property was used in the **btnHide** and **btnShow** button click events. Notice too that many event methods are very similar in their coding. For example, the **Form1_Click** event is identical to the **btnGrow_Click** event. This is often the case in Visual C# projects. We encourage the use of editor features like Copy and Paste when writing code. To copy something, highlight the desired text using the mouse - the same way you do

in a word processor. Then, select **Edit** in the Visual C# main menu, then **Copy**. Move the cursor to where you want to paste. You can even move to other event methods. Select **Edit**, then **Paste**. Voila! The copy appears. The pasted text might need a little editing, but you will find that copy and paste will save you lots of time when writing code. And, this is something you'll want to do since you probably have noticed there's quite a bit of typing in programming, even for simple project such as this. Also useful are **Find** and **Replace** editor features. Use them when you can.

The **Intellisense** feature of Visual C# is another way to reduce your typing load and the number of mistakes you might make. While you are writing C# in the code window, at certain points little boxes will pop up that display information that would logically complete the statement you are working on. This way, you can select the desired completion, rather than type it.

Run the Project

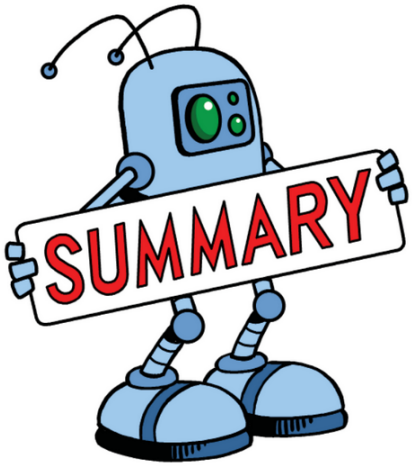
Go ahead! Run your project - click the **Start** button on the Visual C# toolbar. If it doesn't run properly, the only suggestion at this point is to stop the project, recheck your typing, and try again. We'll learn 'debugging' techniques in the next class. Here's a run I made where I grew the form and made it red:



Try all the buttons. Grow the form, shrink the form, change form color, hide the buttons, make the buttons reappear. Make sure you try every button and make sure each works the way you want. Make sure clicking the form yields the desired result. This might seem like an obvious thing to do but, for large projects, sometimes certain events you have coded are never executed and you have no way of knowing if that particular event method works properly. This is another step in proper project design - thoroughly testing your project. Make sure every event works as intended. When done trying out this project, stop it (click the Visual C# toolbar **Stop** button).

Other Things to Try

For each project in this course, we will offer suggestions for changes you can make and try. Modify the **Shrink Form** and **Grow Form** buttons to make them also move the form around the screen (use the Left and Top properties). Change the form color using other color values. Change the **Hide Buttons** button so that it just sets the buttons' Enabled property to false, not the Visible property. Similarly, modify the **Show Buttons** button.



Congratulations! You have now completed a fairly detailed (at least there's more than one control) Visual C# project. You learned about project design, saving projects, details of the form and button controls, and how to build a complete project. You should now be comfortable with the three steps of building a project: placing controls, setting properties, and writing event methods. We will continue to use these steps in future classes to build other projects using new controls and more of the C# language.