Beginning Visual Basic

4. Project Design, Forms, Command Buttons

Review and Preview

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

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

Do you have some ideas of projects you would like to build using Visual Basic? 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 Basic environment, some new controls, and elements of the BASIC language. In this class, you will learn about project design, the form and command button controls, and build a complete project.

Project Design

You are about to start developing projects using Visual Basic. 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 Basic.

Make the BASIC code in your event procedures 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 BASIC. Make sure there are no errors in your project. This may seem like an obvious statement, but many programs are not error-free. Windows 95 has several hundred 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

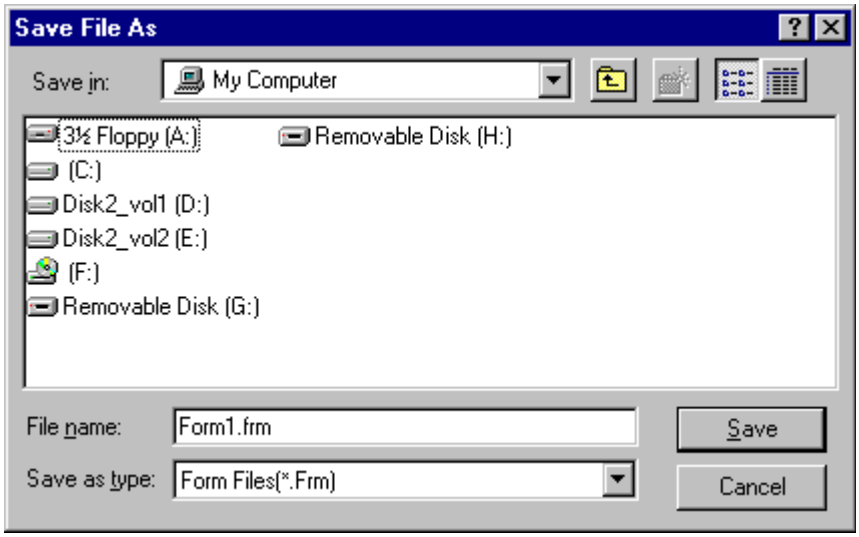
process. We will always try to explain why we do what we do in building a project. And, we will always try to list all the considerations we make.

Saving a Visual Basic Project

In Class 1, you learned how to open, run, and close a previously-saved Visual Basic project, but we never talked about how a project is saved for future use. Now that you are starting to build your own projects, you need to see how to save them. It's really quite easy. We will use the Visual Basic main window toolbar. Look for a button that looks like a small floppy disk. (With writeable CD ROM's coming out, how much longer do you think people will know what a floppy disk looks like? - the new Apple iMac doesn't even have a floppy disk drive!) This is the **Save Project** button:

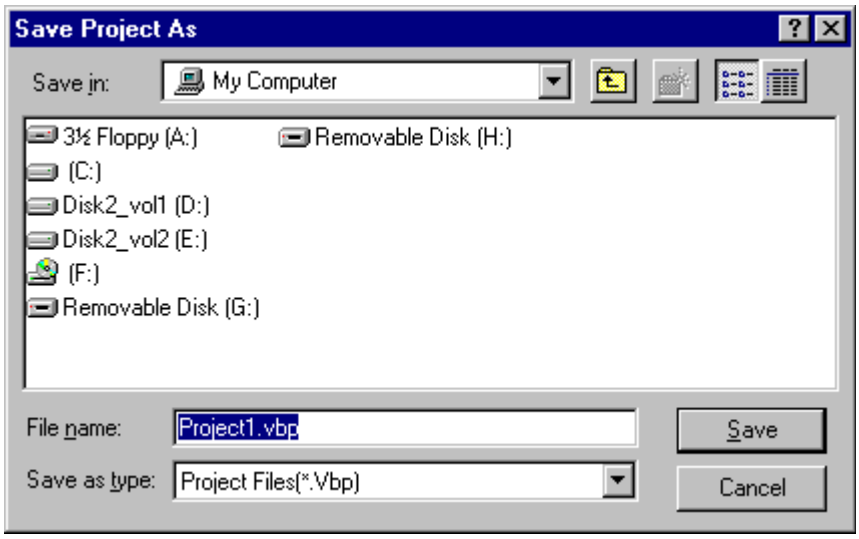
Clicking this button will have different results, depending on when you click it.

If you are working on a new project that has never been saved, the following window will appear:



This window asks you where you want to save your form and what you want to name it (recall this is a file with a **frm** extension). Move to the desired folder, assign a name to your form file (something that is meaningful) and click **Save**.

After saving your form, another window will appear:



This window asks where you want to save your project file and what you want to name it (a **vbp** extension). Again, pick a folder (usually the same folder your form file is saved in) and project name (again, make it meaningful) and click **Save**. At this point, your project is saved in two files: the form file and the project file.

If you are working on a project that has been saved previously and you click on the **Save Project** button, Visual Basic automatically saves both the form file and project file, using the same names, without asking you any questions. It is suggested you occasionally save your project as your work on it. And, always save your project before running it or before leaving Visual Basic. When you want to open a saved project, just click on the **Open Project** button on the Windows toolbar and select the project file, then click **Open**. The project file will open and the associated form will appear.

On-Line Help

Many times, while working in the Visual Basic 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 BASIC means. A great way to get help when you're stuck is to ask someone who knows the answer. Others 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 Basic books out there! Or, another great way to get help is to use the Visual Basic **On-Line Help** system.

Most Windows applications, including Visual Basic, have help files available for your use. To access the Visual Basic 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 Basic 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 Basic 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!

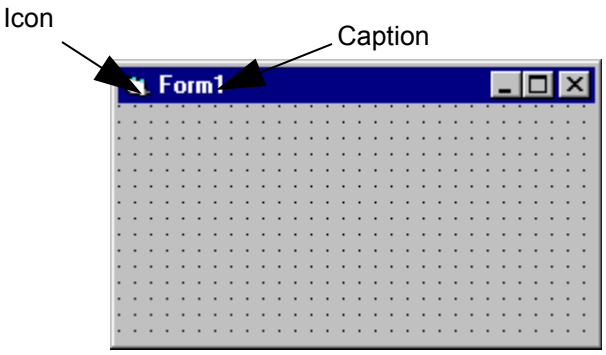
A really great feature about the Visual Basic on-line help system is that it is 'context sensitive.' What does this mean? Well, let's try it. Start Visual Basic and start a new project. Go to the properties window. Scroll down the window displaying the form properties and click on the word **BackColor**.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

That's enough new material about the Visual Basic environment. Now, let's look, in detail, at two important controls: the form itself and the command button. Then we'll start our study of the BASIC 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 Basic 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. Three letter prefix for form names is **frm**.
Caption	Text that appears in the title bar of form.
Icon	Reference to icon that appears in title bar of form (we'll look at creating icons in Class 7).
Left	Distance from left side of computer screen to left side of form.
Top	Distance from top side of computer screen to top side of form.
Width	Width of the form in twips.

Height	Height of form in twips.
BackColor	Background color of form.
BorderStyle	Form can either be sizable (can resize using the mouse) or fixed size.

Example

To gain familiarity with these properties, start Visual Basic and start a new project with just a form. Set the Top, Left, Height and Width property values and see their effect on form position and size. Resize and move the form and notice how those values are changed in the properties window. Set the Caption property. Pick a new background color using the selection techniques discussed in Class 3. To see the effect of the BorderStyle property, set a value (either **1-Fixed Single** or **2-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 primarily acts as a 'container' for other controls, but it does support events. That is, it can respond to some user interactions. We will only be concerned with two form events in this course:

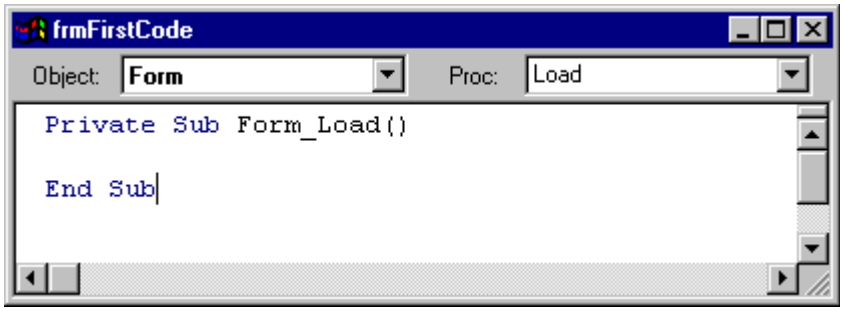
Event	**Description**
Click	Event executed when user clicks on the form with the mouse.
Load	Event executed when the form first loads into the computer's memory. This is a good place to set initial values for various properties and other project values.

One word about form naming. Recall we saw in a past class that control names are used in event procedures. This is not true for forms. All form event procedures have the format:

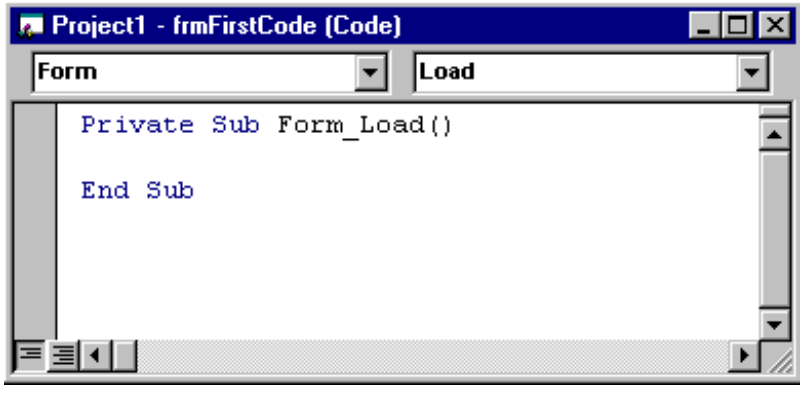
Form_EventName

That is, no matter what **Name** property you assign to the form, event procedures are listed under the word **Form**. So, when looking for form event procedures in the code window, scroll down the **Object List** until you find **Form**. Try this with the example just used to play with properties. Note if we assign the name **frmFirstCode** to the form, the code window will be::

VB4:



VB5, VB6:



Note that the word **Form** appears in the object list, not **frmFirstCode**. We always need to be aware of this peculiarity when working with form event procedures. All other controls will appear in the object list by their assigned Name property.

Command Button Control

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

In Toolbox:

On Form (default properties):

Properties

A few useful properties for the command button are:

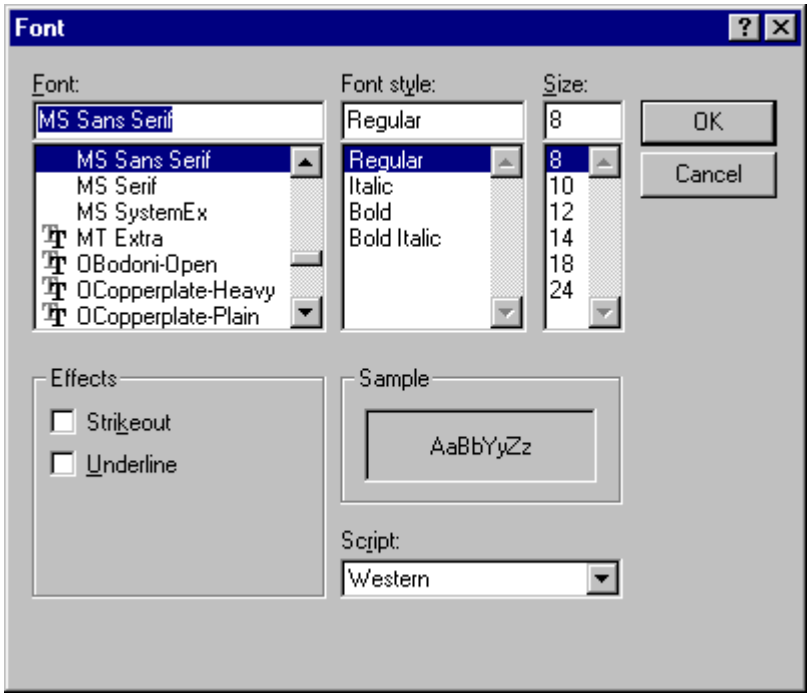
Property	Description
Name	Name used to identify command button. Three letter prefix for command button names is **cmd**.
Caption	Text that appears on the command button.
Font	Sets style, size, and type of caption text.
Left	Distance from left side of form to left side of command button.
Top	Distance from top side of form to top side of command button.
Width	Width of the command button in twips.
Height	Height of command button in twips.
Enabled	Determines whether command button can respond to user events (in run mode).
Visible	Determines whether the command button appears on the form (in run mode).

Example

Start Visual Basic and start a new project. Put a command button on the form. Move the button around and notice the changes in Top and Left properties. Resize the button and notice how Width and Height change. Set the Caption property.

Many controls, in addition to the command button, have a Font property, so let's take a little time to look at how to change it. Font establishes what the Caption 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:

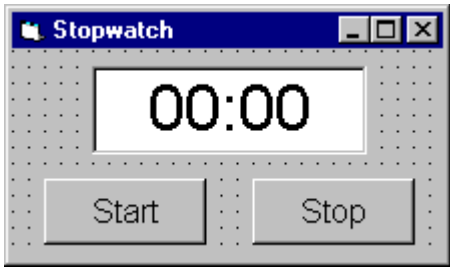
Click this button and a **Font Window** will appear:



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 command button Caption property.

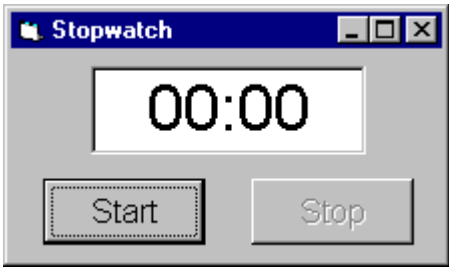
Two other properties listed for the command button are Enabled and Visible. Each of these properties can either be **True** (On) or **False** (Off). Most other controls also have these properties. Why do you need these?

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 Basic is in run mode. When a command 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 procedures for, to be invisible? The answer is similar to that for the Enabled property. Many times in a project, you will find times when 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 image 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 command 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 command button event of interest, but it is a very important one:

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

Every command button will have an event procedure corresponding to the Click event.

BASIC - The First Lesson

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

Event Procedure Structure

You know, by now, that event procedures are viewed in the Visual Basic code window. Each event procedure has the same general structure. First, there is a **header** line of the form:

```
Private Sub ControlName_EventName()
```

This tells us we are working with a **Private** (only accessible from our form), **Sub**routine (another name for a event procedure) that is executed when the event **EventName** occurs for the control **ControlName**. Makes sense, doesn't it?

The event procedure code begins following the header line. The event procedure 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. It will process lines until it reaches the event procedures **footer** line:

```
End Sub
```

The event procedure code is written in the BASIC language. BASIC is a set of keywords and symbols that are used to make the computer do things. There is a lot of content in BASIC 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! You must especially be exact when typing in event procedures. Good typing skills are a necessity in the computer age. As you learn Visual Basic 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 Basic applications.

Assignment Statement

The simplest, and most used, statement in BASIC 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 BASIC 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 BASIC.

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 Basic automatically supplies the proper type. If we want to change a property in an event procedure using the BASIC 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 **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 named frmExample to 4,000 twips, we would write in BASIC:

```
frmExample.Width = 4000
```

This says we replace the current Width of the form with the new value of 4000. Notice you write 4,000 as 4000 in BASIC - we can't use commas in large numbers.

A second property type is the **long integer** type. And, a long integer is just like its name says. An integer type property can have values up to 32,767.

Sometimes, we need bigger numbers than this, hence the long integer type. A long integer can have a value up to 2,147,483,647. That's pretty big, but do you realize Bill Gates couldn't even write down his net worth using a long integer? Maybe he needs someone at Microsoft to invent a **very long integer** type! The most common properties that uses long integers are colors, like the BackColor and ForeColor properties you will see for some controls. Remember, in a past class, we saw that the property value for gray is written as &H8000000F& - that's a shorthand notation (called a hexadecimal number) for a long integer. When assigning color properties, we must use long integers.

Fortunately, Visual Basic gives us lots of easy ways to refer to long integer numbers for colors, so it makes working with long integers easy. One way to use colors is with **symbolic constants**. Symbolic constants are used many places in Visual Basic - you'll see lots of them as you work through the course. All symbolic constants start with the two letters **vb** (Visual Basic). Some symbolic constants for colors are:

vbBlack - Black	**vbRed** - Red
vbGreen - Green	**vbYellow** - Yellow
vbBlue - Blue	**vbMagenta** - Magenta (purple)
vbCyan - Cyan (sky blue)	**vbWhite** - White

Each of these constants 'stores' the corresponding long integer value for the color it represents. To change our example form's BackColor property to blue, you would use this assignment statement:

```
frmExample.BackColor = vbBlue
```

This says the BackColor of the form is replaced by long integer value represented by the symbolic constant named vbBlue.

Another property type is the **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 command 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 our example form disappear (not a very good thing to do!), we would use the assignment statement:

```
frmExample.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:

```
frmExample.Visible = True
```

The last property type we need to look at 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 Caption property is a string type property. We will do lots of work with strings in Visual Basic, 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 example form a caption, we would use:

```
frmExample.Caption = "This is a caption in quotes"
```

This assignment statement says the Caption 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 BASIC 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, an apostrophe (‘). This symbol is to the left of the <Enter> key on most keyboards, not the key next to the 1 key. Anything written after the comment symbol will be ignored by the computer. You can have a comment take up a complete line of BASIC code, like this:

```
'Change form to blue
frmExample.BackColor = vbBlue
```

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

```
frmExample.BackColor = vbBlue '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.

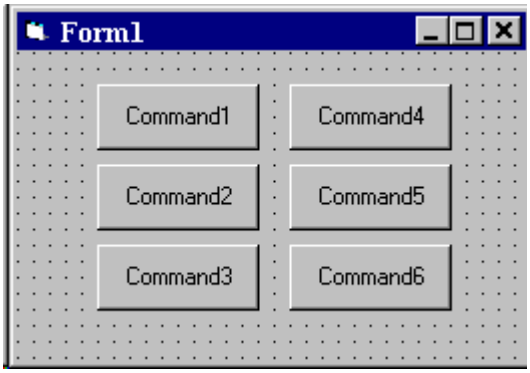
Project - Form Fun

Project Design

In this project, we will have a little fun with form properties using command 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.

Place Controls on Form

Start a new project in Visual Basic. Size the form so six command buttons will fit on the form. Place six command buttons on the form. Move the buttons around until the form looks something like this:



One warning. 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 Basic, that you can copy and paste controls. You might be tempted to do that here - why create six command buttons when you could just create one command button, then copy and paste it five

times? Yes, you could do that, but **don't**! If you do, it will look like you have command buttons on the form and you do, kind of. Copying controls gives you a different type of control - one you study in more advanced Visual Basic classes. So, in this class, we will always create single copies of every control we need. Later, as you become a better programmer, you might want to look into what is happening when you copy and paste controls.

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 Basic when the control is placed on the form).

Form1 Form:

Property Name	**Property Value**
Name	frmFormFun
Caption	Form Fun

Command1 Command Button:

Property Name	**Property Value**
Name	cmdShrink
Caption	Shrink Form

Command2 Command Button:

Property Name	**Property Value**
Name	cmdGrow
Caption	Grow Form

Command3 Command Button:

Property Name	Property Value
Name	cmdHide
Caption	Hide Buttons

Command4 Command Button:

Property Name	Property Value
Name	cmdRed
Caption	Red Form

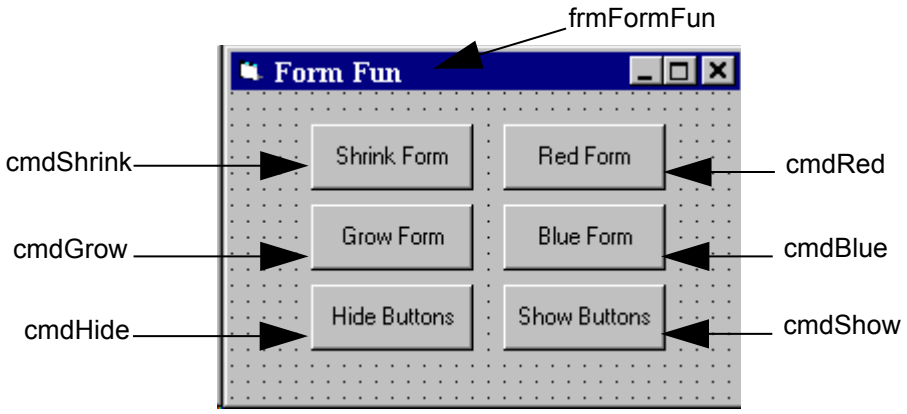
Command5 Command Button:

Property Name	Property Value
Name	cmdBlue
Caption	Blue Form

Command6 Command Button:

Property Name	Property Value
Name	cmdShow
Caption	Show Buttons
Visible	False

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



What we have are six command 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** command 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 Procedures

We have six command buttons on our form. We need to write code for the **Click** event procedure for each of these buttons. We'll also want to write a **Click** event procedure 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.

For each event procedure, you use the code window. Select the control in the object list and the event in the procedures list. Then click in the region between the header line and footer 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 **cmdShrink_Click** event procedure. In this procedure, we decrease the form height by 100 twips and decrease the form width by 100 twips:

```
Private Sub cmdShrink_Click()
'Shrink the form
'Decrease the form height by 100 twips
frmFormFun.Height = frmFormFun.Height - 100
'Decrease the form width by 100 twips
frmFormFun.Width = frmFormFun.Width - 100
End Sub
```

Before looking at the other event procedures, let's look a bit closer at this one since it uses a few ideas we haven't clearly discussed. This is the event procedure 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:

```
frmFormFun.Height = frmFormFun.Height - 100
```

Recall how the assignment operator (=) works. The right side is evaluated first. So, 100 is subtracted (using the - sign) from the current form height. That value is assigned to the left side of the expression, frmFormFun.Height. The result is the form Height property is replaced by the Height property minus 100 twips. After this line of code, the Height property has decreased by 100 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 frmFormFun.Height is, the right side will always be 100 smaller than the left side. But, even though this is not an equality, you will often hear programmers read this statement as "frmFormFun.Height equals frmFormFun.Height minus 100," 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 procedures. The **cmdGrow_Click** procedure increases form height by 100 twips and increases form width by 100 twips:

```
Private Sub cmdGrow_Click()
'Grow the form
'Increase the form height by 100 twips
frmFormFun.Height = frmFormFun.Height + 100
'Increase the form width by 100 twips
frmFormFun.Width = frmFormFun.Width + 100
End Sub
```

The **cmdRed_Click** event procedure changes the form background color to red:

```
Private Sub cmdRed_Click()
'Make form red
frmFormFun.BackColor = vbRed
End Sub
```

while the **cmdBlue_Click** event procedure changes the form background color to blue:

```
Private Sub cmdBlue_Click()
'Make form blue
frmFormFun.BackColor = vbBlue
End Sub
```

The **cmdHide_Click** event procedure is used to hide (set the **Visible** property to **False**) all command buttons except **cmdShow**, which is made **Visible**:

```
Private Sub cmdHide_Click()
'Hide all buttons but cmdShow
cmdGrow.Visible = False
cmdShrink.Visible = False
cmdHide.Visible = False
cmdRed.Visible = False
cmdBlue.Visible = False
'Show cmdShow button
cmdShow.Visible = True
End Sub
```

and the **cmdShow_Click** event procedure reverses these effects:

```
Private Sub cmdShow_Click()
'Show all buttons but cmdShow
cmdGrow.Visible = True
cmdShrink.Visible = True
cmdHide.Visible = True
cmdRed.Visible = True
cmdBlue.Visible = True
'Hide cmdShow button
cmdShow.Visible = False
End Sub
```

Lastly, the **Form_Click** event procedure is also used to 'grow' the form, so it has the same code as **cmdGrow_Click**:

```
Private Sub Form_Click()
'Grow the form
'Increase the form height by 100 twips
frmFormFun.Height = frmFormFun.Height + 100
'Increase the form width by 100 twips
frmFormFun.Width = frmFormFun.Width + 100
End Sub
```

(Make sure you have the correct procedure here. When you choose the **Form** control, the procedure displayed will be **Load**. Choose the **Click** event using the procedures list.) Review the earlier-discussed techniques for saving a new project. Save your project.

You should easily be able to see what's going on in each of these procedures. Pay special attention to how the Visible property was used in the cmdHide and cmdShow button click events. Notice too that many event procedures are very similar in their coding. For example, the **Form_Click** event is identical to the **cmdGrow_Click** event. This is often the case in Visual Basic projects. Unlike control placement, 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 Basic main menu, then **Copy**. Move the cursor to where you want to paste. You can even move to other event procedures. 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.

VB5 and **VB6** offer another way to reduce your typing load and the number of mistakes you might make. If you click **Tools**, then **Options** on the Visual Basic main menu, then select the **Editor** tab, there is an option called **Auto List Members**. If this option is selected, while you are writing BASIC 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. Then, you can select the desired completion, rather than type it. If you use **VB5** or **VB6**, you might want to try the Auto List Members option. Use on-line help to find out more about its use.

Run the Project

Go ahead! Run your project - click the **Start** button on the Visual Basic 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.

Try all the command 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 procedure works properly. This is another step in proper project design - thoroughly testing your project. Make sure every event works as intended. Stop your project (click the Visual Basic toolbar Stop button). Save your project if you changed anything.

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). Add more possible colors to the form using the other symbolic constants we defined. Change the **Hide Buttons** button so that it just sets the command buttons' Enabled property to False, not the Visible property. Similarly, modify the **Show Buttons** button.

Summary

Congratulations! You have now completed a fairly detailed (at least there's more than one control) Visual Basic project. You learned about project design, saving projects, details of the form and command 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 procedures. We will continue to use these steps in future classes to build other projects using new controls and more of the BASIC language.