

TegoSoft Inc.

Copyright © 1994-1996 TegoSoft Inc. ® All Rights Reserved

Web Site: <http://www.tegosoft.com>

☒ e-mail: tegosoft@msn.com

Lesson 3 - Scrollbars

Common User Interface

With Visual Basic you write Windows programs. One of the main advantages of using Windows is that the *user interface* mechanism of Windows is very popular and well accepted by millions of people. For example, a typical Windows program includes CommandButtons, CheckBoxes, Scrollbars, OptionButtons, EditBoxes, and so on. These objects are the means by which your program communicates with the user of the program. So the *user interface mechanism* is the same for all Windows programs. After developing a Windows program with Visual Basic, you don't have to bother explaining to your user how to use the CommandButton for example. Why? Because the fact that your user uses Windows means that it is assumed that your user knows how to use the standard user interface mechanism of Windows.

In the previous lesson you learned how to incorporate a CommandButton into your Visual Basic programs. Next you'll learn how to incorporate other objects into your programs.

The MyScroll Program

The program that you'll now develop is called the **MyScroll** program. As implied by its name, the MyScroll program includes a scrollbar in it.

Here are the specifications of the MyScroll program:

- Upon starting the MyScroll program, the window shown in Figure 3.1 appears.

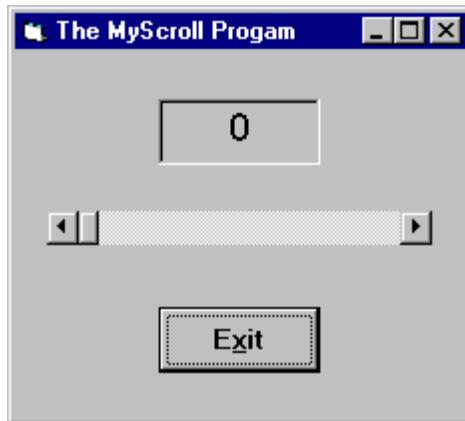


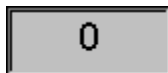
Figure 3.1. *The window of the MyScroll program.*

As shown in Figure 3.1, the window of the MyScroll program includes the following objects in it:

- An Exit button:



- A Label that displays the number **0** in it:



- A scrollbar:



- The user can now change the scrollbar position, and the label will display a number that corresponds to the new position of the scrollbar. For example, when the scrollbar position is in its extreme left position, the label displays the number 0. 0 represents the minimum value of the scrollbar position.

when the scrollbar position is at the extreme right position, the label displays the number 15. 15 represents the maximum value of the scrollbar position.


As the user changes the scrollbar position, the label should display the number that corresponds to the scrollbar position. The scrollbar can have the following positions:

0, 1, 2, 3, ..., 15

Now that you understand what the MyScroll program should do, let's implement it.


Create the Directory of the Project


You'll save the files of the MyScroll program into the C:\VBMyProg\Lesson03 directory.


 Use the Explorer or the File Manager to create the C:\VBMyProg\Lesson03 directory.

Start Visual Basic, and Save the Form and the Project Files


As usual, you start the project by saving Form1 and saving the project file.

 Start Visual Basic, and then select **New Project** from the **File** menu.

 Select **Project** from the **View** menu of Visual Basic to display the Project window. Click the **View Form** button of the Project window to make **Form1** the selected form.


 While Form1 is the selected window, select **Save File As** from the **File** menu of Visual Basic.

Visual Basic responds by displaying the **Save File As** dialog box.

 Save the Form file as **MyScroll.frm** inside the C:\VBMyProg\Lesson03 directory.


 Select **Save Project As** from the **File** menu of Visual Basic.

Visual Basic responds by displaying the **Save Project As** dialog box.

 Save the project file as **MyScroll.vbp** inside the C:\VBMyProg\Lesson03 directory.

Set the Properties of Form1

You'll now set the properties of Form1.

 Make the Form1 window the selected window, and then press the F4 key on your keyboard to display the Properties window of Form1.

 Set the properties of Form1 as follows:

Property	Setting
Name	frmMyScroll
Caption	The MyScroll Program
Height	3705
Width	3270
BackColor	Light gray



NOTE

In this tutorial you'll set the **Name** properties of forms with names that start with the characters: **frm**

For example, you changed the **Name** property of the form of the MyScroll program from **Form1** to **frmMyScroll**.

The form should now look as shown in Figure 3.2.

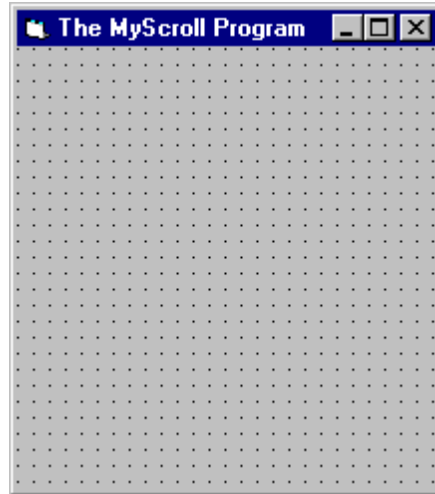


Figure 3.2. *The frmMyScroll form after setting its properties.*



NOTE

In the previous lesson you were not instructed to change the **Name** property of the form. Now you were instructed to change the **Name** property of Form1 from **Form1** to **frmMyScroll**. The sole reason for instructing you to do so, is to illustrate that the form has the **Name** property, and you can set its **Name** property to a name that is appropriate to the program that you are implementing. Typically, when a program has a single form in it (such as the MyScroll program), it is ok to leave the **Name** property of **Form1** as Form1. But sometimes, your programs will have multiple forms in them, and in this case, it is best to name the forms according to their roles in the program.


Save your work:




Select **Save Project** from the **File** menu of Visual Basic.

Placing the Exit button inside the Form

As shown in Figure 3.1, the window of the MyScroll program has an Exit button in it. You'll now place the Exit button inside the form.

 Display the Toolbox window by selecting **Toolbox** from the **View** menu, and then double click the CommandButton icon inside the Toolbox window.

Visual Basic responds by placing a CommandButton object inside the form.

 Make the CommandButton the selected object, press the F4 key on your keyboard to display the Properties window of the CommandButton, and then set the properties of the CommandButton as follows:

Property	Setting
Name	cmdExit
Caption	E&xit
Top	2640
Left	840
Height	495
Width	1215



NOTE

In Windows, it is customary to underline the **x** in **Exit**. This is the reason that you are instructed to set the **Caption** property of the **Exit** button to **E&xit**. So pressing Alt-X has the same effect as clicking the Exit button.

The form should now look as shown in Figure 3.3.

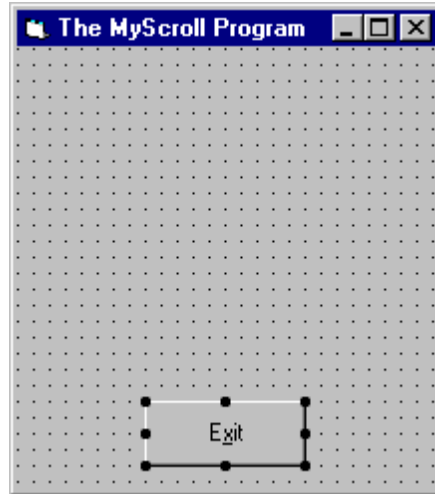




Figure 3.3. The Exit button inside the frmMyScroll form.


Setting the Font Property of the Exit Button

The font that is used for the caption of the **Exit** button is specified in the **Font** property of the **cmdExit** CommandButton. You'll now set a new value for the **Font** property of the Exit button


 Display the Properties window of the cmdExit button (i.e., make the cmdExit button the selected object, and then press the F4 key on your keyboard).

 Click inside the cell to the right of the **Font** property cell inside the Properties window of the **cmdExit** button.

Visual Basic responds by placing the three dots icon in the cell to the right of the **Font** property.

 Click the three dots icon that appear on the cell to the right of the **Font** property.

Visual Basic responds by displaying the **Font** dialog box. You use the **Font** dialog box to set the font of your choice for the caption of the cmdExit button.

 Set the values of the font inside the **Font** dialog box as follows (see Figure 3.4):

Font: MS Sans Serif

Font style: Bold

Size: 10

and then click the OK button of the **Font** dialog box.

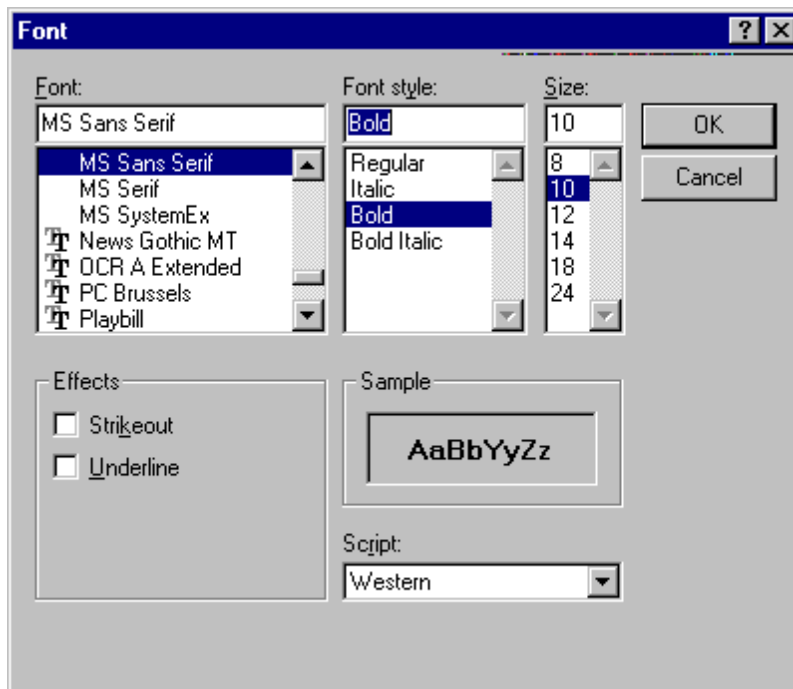


Figure 3.4. Setting the Font property of the Exit button

Your **frmMyScroll** form should now look as shown in Figure 3.5.

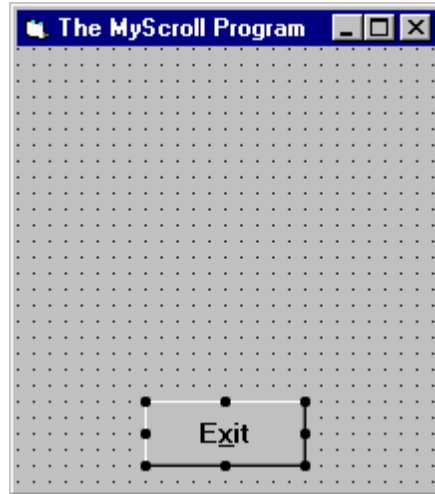


Figure 3.5. *The Exit button with its new font.*

Take a look at the caption of the Exit button in Figure 3.5, and compare it to the caption of the Exit button in Figure 3.3. As you can see, now the caption of the Exit button has the font that you assigned to the **Font** property of the cmdExit button.



NOTE

The sole purpose of setting a new value to the **Font** property to the Exit button is to illustrate the use of this property. (Typically, you can use the default font that Visual Basic set).




NOTE


It is very important to remember not to set a "fancy" font to the **Font** property of the objects in your programs. Why? Because the fonts that are shown in the **Font** dialog box (see Figure 3.4) are the fonts available on **your** PC. Your user however may not have the fonts that you have on your PC. So what will happen if you set the **Font** property to a font that your user does not have? When your user will execute the program, Windows will automatically substitute a different font. Windows will make its best efforts to substitute a font that most closely resembles the original font. Nevertheless, if you want to make sure that your user will see the forms the way you designed them, make sure not to use special fancy fonts that do not reside in most PCs. That is, assume that the user has only these fonts that came with Windows. If you installed additional fonts on your PC, do not use these fonts in your programs (unless you have a very good reason to do so).


Attaching Code to the Exit Button

You'll now attach code to the Exit button.


 Select **Project** from the **View** menu, make sure **frmMyScroll** is highlighted inside the Project window, and then click the **View Code** button inside the Project window.

Visual Basic responds by displaying the Code window.

 Set the **Object** list box of the Code window to **cmdExit** (because you are about to attach code to the **cmdExit** button).

 Set the **Proc** list box of the Code window to **Click** (because you want the code that you are about to type to be automatically executed when the user **Clicks** the cmdExit button).

Visual Basic responds by displaying the **cmdExit_Click()** procedure ready for you to type code in it.

 Type code inside the cmdExit_Click() procedure. After typing the code, the cmdExit_Click() procedure should look as follows:


```
Private Sub cmdExit_Click()
```

```
End
```


```
End Sub
```

The code that you typed is executed automatically whenever the user clicks the Exit button, and this code terminates the MyScroll program.


Save your work:

 Select **Save Project** from the **File** menu of Visual Basic.

Just to make sure that everything is working as expected, execute the MyScroll program:


 Select **Start** from the **Run** menu.

Visual Basic responds by executing the MyScroll program.

 Terminate the MyScroll program by clicking its Exit button (or by pressing Alt-X).

Placing the Scrollbar inside the Form

As shown in Figure 3.1, the window of the MyScroll program should have a Scrollbar in it. You'll now place a scrollbar object inside the **frmMyScroll** Form.

 Display the Toolbox window by selecting **Toolbox** from the **View** menu, and then double click the horizontal Scrollbar tool inside the Toolbox window. The horizontal Scrollbar tool is shown in Figure 3.6. In Figure 3.6 the icon of the horizontal Scrollbar tool is shown on the second column from the left and fourth row from the top. However, in your Toolbox window, the icon of the horizontal Scrollbar tool may be located in a different location than the location shown in Figure 3.6. So make sure that you are using the horizontal Scrollbar tool by placing (not clicking) the mouse cursor on the horizontal Scrollbar icon inside the toolbox window. The yellow rectangle that pops up should have the text **HScrollBar** in it.

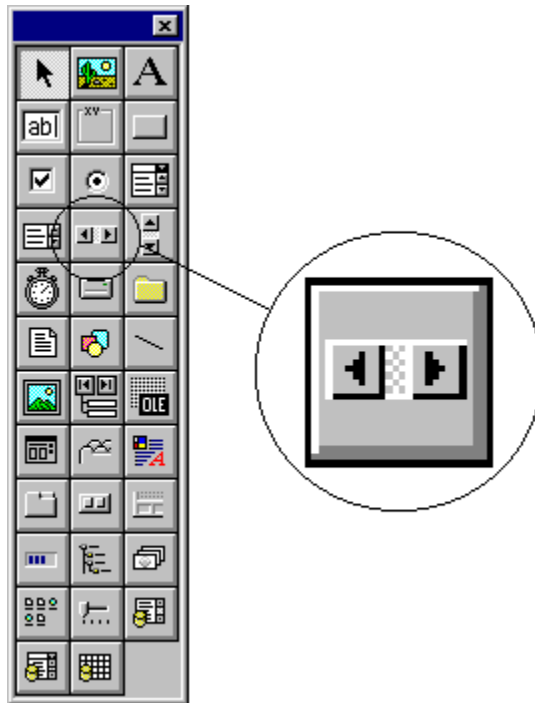




Figure 3.6. The icon of the Horizontal Scrollbar tool inside the Toolbox window.

 Set the properties of the horizontal scrollbar as follows:

Property	setting
Name	hsbSet
Min	0
Max	15
Value	0
Left	360
Top	1680
Width	2415
Height	255

Let's go over the properties that you set for the scrollbar:

-  You set the **Name** property of the scrollbar to **hsbSet**. This scrollbar is used for setting the text inside the label located on the top of the MyScroll

program's window (see Figure 3.1). So it is appropriate to have the name **hsbSet** for the **Name** property of this scrollbar. You preceded the name with the characters **hsb**, because in this tutorial, the characters **hsb** are used as the first three characters of the names of horizontal scroll bars.

- You set the **Min** property of the **hsbSet** scrollbar to **0**. This means that the extreme left side of the scrollbar represents the value **0**.

- You set the **Max** property of the **hsbSet** scrollbar to **15**. This means that the extreme right side of the scrollbar represents the value **15**.

- You set the **Value** property of the scrollbar to **0**. This means that the current position of the scrollbar is at **0**. So when you'll start the MyScroll program, the scrollbar position will be on the extreme left side. Had you set the **Value** property of the scrollbar to 15, the current position of the scrollbar would have been on the extreme right side. Had you set the **Value** property of the scrollbar to 1, the scrollbar position would be a little bit to the right from the 0 position, and so on.

- You set the **Left** property of the scrollbar to **360**. This means that the upper left corner of the scroll bar is located **360** units from the left edge of the frmMyScroll window.

- You set the **Top** property of the scrollbar to **1680**. This means that the upper left corner of the scroll bar is located **1680** units below the top edge of the frmMyScroll window.

- You set the **Width** property of the scrollbar to **2415**. This means that the width of the scrollbar is **2415** units.

- You set the **Height** property of the scrollbar to **255**. This means that the height of the scrollbar is **255** units.

Your frmMyScroll form should now look as shown in Figure 3.7.

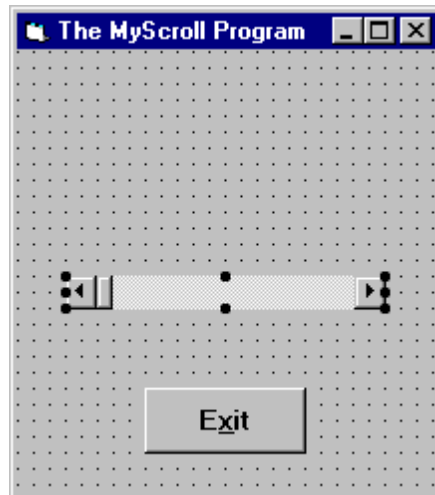




Figure 3.7. The frmMyScroll window with the scrollbar in it.

Save your work:

 Select **Save** Project from the **File** menu of Visual Basic.

Placing the Label inside the Form

As shown in Figure 3.1, the window of the MyScroll program should have a label object in it. You'll now place a **Label** object inside the **frmMyScroll** Form.

 Display the Toolbox window by selecting **Toolbox** from the **View** menu, and then double click the **Label** tool inside the Toolbox window. The **Label** tool is shown in Figure 3.8. In Figure 3.8 the icon of the Label tool is shown on the first column from the right and first row from the top. However, in your Toolbox window, the icon of the Label tool may be located in a different location than the location shown in Figure 3.8. So make sure that you are using the Label tool by placing (not clicking) the mouse cursor on the Label icon inside the toolbox window. The yellow rectangle that pops up should have the text **Label** in it.

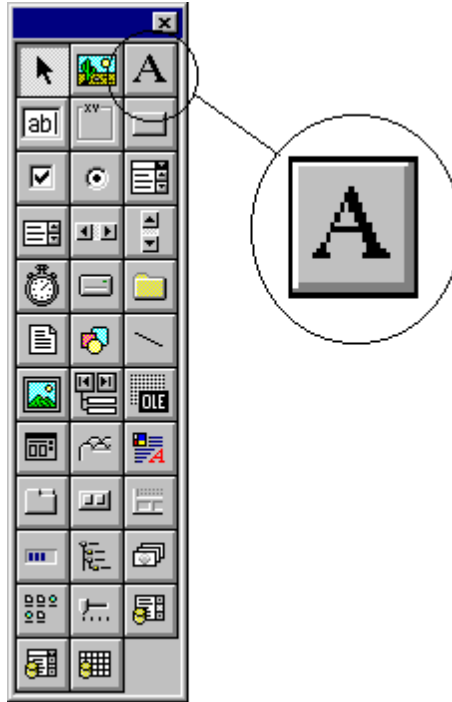




Figure 3.8. *The icon of the Label tool inside the Toolbox window.*

 Make sure that the Label that you placed inside the frmMyScroll program is the selected object, and then press the F4 key on your keyboard to display the Properties window of the Label.

 Set the properties of the Label as follows:

Property	setting
Name	lblResult
Caption	0
Left	960
Top	480
Width	1215
Height	495
Alignment	2 - Center
BorderStyle	1-Fixed Single
Font	Font: MS Sans Serif

	Font style: Bold Font size: 12
--	-----------------------------------

Let's go over the properties that you set for the **lblResult** Label:

- You set the **Name** property of the Label to **lblResult**. This label is used for displaying the result of setting the scrollbar (see Figure 3.1). So it is appropriate to have the name **lblResult** for the **Name** property of this label. You preceded the name with the characters **lbl**, because in this tutorial, the character **lbl** are used as the first three characters of the names of labels.
- You set the **Caption** property of the label to **0**. This means that the text that is displayed inside the label is **0**.
- You set the **Left** property of the label to **960**. This means that the upper left corner of the label is located **960** units from the left edge of the frmMyScroll window.
- You set the **Top** property of the label to **480**. This means that the upper left corner of the label is located **480** units below the top edge of the frmMyScroll window.
- You set the **Width** property of the label to **1215**. This means that the width of the label is **1215** units.
- You set the **Height** property of the label to **495**. This means that the height of the label is **495** units.
- You set the **Alignment** property of the label to **2-center**. This means that the text that is displayed inside the label is centered.
- You set the **BorderStyle** property of the label to **1-Fixed Single**. This means that the label is enclosed in a rectangle that has a style called **Fixed Center**.
- You set the **Font** property of the label to **MS Sans Serif, Bold, with size 12**. This means that the text that is displayed inside the label has this font setting.

Your frmMyScroll form should now look as shown in Figure 3.9.

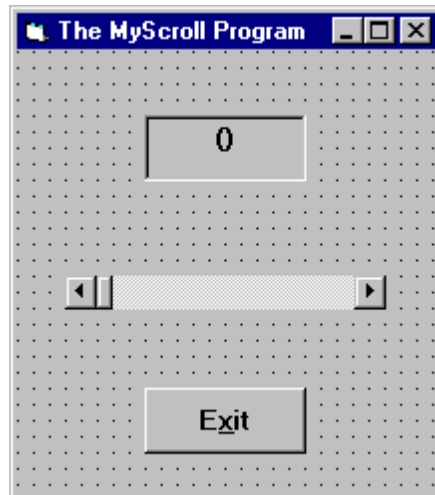



Figure 3.9. The frmMyScroll with the Label object in it.

Save your work:

 Select **Save** Project from the **File** menu of Visual Basic.


Executing MyScroll (Without Code Yet)

So far, you placed all the objects that the frmMyScroll form needs, and you attached code to the Exit button. As of yet, you did not attach code that accomplishes the task of updating the lblResult label with a number that represents the hsbSet scrollbar position.

Nevertheless, let's execute the MyScroll program at this stage of its development:

 Select **Start** from the **Run** menu of Visual Basic.

Visual Basic responds by executing the MyScroll program, and the window shown back in Figure 3.1 appears.

 Change the position of the scrollbar. That is, click the left and right arrow icons that appear on the left and right sides of the scrollbar, drag (slide) the square indicator of the scrollbar, and click in between the

square indicator of the scrollbar and the arrow icons of the scrollbar. All these operations cause changes in the current position of the scrollbar.

But as you can see, you can change the scrollbar position all you want, and the text inside the label remains 0. Why doesn't the label react to the change in position of the scrollbar? Because you did not yet attach any code that accomplishes this.



NOTE

The sole purpose of instructing you to execute the MyScroll program at this stage of its development is to show you that the scrollbar is functioning. That is, you can change the position of the square indicator of the scrollbar. But of course, the text inside the lblResult label is not being updated, because you did not yet type any code that updates the label with a number that represents the current position of the scrollbar.

Attaching Code to the Scrollbar

You'll now attach code to the scrollbar. What will this code do? This code will update the **lblResult** label with a number that represents the current position of the **hsbSet** scrollbar.

As always, the question is: where will you type the code? Here is how you locate the section of the Code window where your code should be typed:



Select **Project** from the **View** menu of Visual Basic to display the Project window.



Click the **View Code** button of the Project window.

Visual Basic responds by displaying the Code window.



Set the **Object** list box at the top left of the Code window to **hsbSet**, and make sure that the **Proc** list box at the top right of the Code window is set to **Change**.

In the preceding step you set the **Object** list box to **hsbSet** and the **Proc** list box to **Change**. You did this, because you want the code to

=====


automatically be executed whenever the user changes the hsbSet scrollbar position.

Visual Basic wrote the following code for you:

```
Private Sub hsbSet_Change()
```

```
End Sub
```

So Visual Basic prepared the **hsbSet_Change()** procedure for you. Whatever code you'll type inside the **hsbSet_Change()** procedure, this code will be automatically executed whenever the user **changes** the scrollbar position.

 Type code inside the **hsbSet_Change()** procedure. After typing the code, the **hsbSet_Change()** procedure should look as follows:

```
Private Sub hsbSet_Change()
```

```
    lblResult.Caption = Str(hsbSet.Value)
```

```
End Sub
```

The code that you typed sets the **Caption** property of the **lblResult** label to the **Value** property of the **hsbSet** scrollbar:

```
lblResult.Caption = Str(hsbSet.Value)
```

To understand the preceding code, look at the expression that appears to the left of the equal sign:

```
lblResult.Caption
```

The preceding expression means the **Caption** property of the **lblResult** label.

Yes, when you want to indicate a property of an object, you first type the **Name** of the object, then a period, and then the property.

As another example, to indicate the **Caption** property of the **cmdExit** CommandButton you'll write:

```
cmdExit.Caption
```

(However, in the MyScroll program there is no need to write code that sets a value to the **Caption** property of the **cmdExit** button).

In a similar manner, you indicate the **Value** property of the **hsbSet** scroll bar as follows:

```
hsbSet.Value
```

Again, in the preceding expression you first typed the **Name** of the object (**hsbSet**), then a period, and then the property name (**Value**). Recall that **Value** is the property of a scrollbar, and this property represents the current position of the scrollbar.

When the user executes the program, whenever the user changes the scrollbar position, the **Value** property is updated automatically. So if for example the user changes the scrollbar position and places the scrollbar at the **4** position, then two things occur automatically:

- The **Value** property of the **hsbSet** scrollbar is automatically set to **4**.
- The **hsbSet_Change()** procedure is executed.

The code that you typed sets the **Caption** property of the **lblResult** label to the **Value** property of the **hsbSet** scrollbar:

```
lblResult.Caption = Str(hsbSet.Value)
```

So putting it altogether, the label is updated with a number that represents the scrollbar position.



NOTE

You used the following statement:

```
lblResult.Caption = Str(hsbSet.Value)
```

That is, in the preceding statement you set the **Caption** property of the label (which is a string) to the **Value** property of the scrollbar (which is an integer number).

You perform the conversion from an integer to a string by using the **Str()** function.

If for example the Value property of the scroll bar is equal to the integer 1, then **Str(hsbSet.Value)** is equal to the string 1.

The Str() function "works" on whatever is supplied in its parenthesis. In the preceding statement, the **Value** property of the **hsbSet** scrollbar is typed inside the parenthesis of the **Str()** function. So the **Str()** function converts the number to a string, and the **Caption** property of the **lblResult** label is assigned with the resultant string.

In this case, the Str() function is referred to as a function that **returns a string**, because the Str() function converts a number to a **string**.

The expression that you typed inside the parenthesis of the Str() function is known as the **parameter** of the function. So the parameter of the Str() function is a number. The parameter of the Str() function is converted to a string.

When equating "things", make sure that the "things" that you are equating are of the same type. This is the reason you first converted the integer to a string, and then you set the Caption property of the Label to the resultant string.



NOTE

When you attached code to the scrollbar, you selected **hsbSet** inside the **Object** list box of the Code window, and you selected **Change** from the **Proc** list box of the Code window. As a result, Visual Basic displayed the `hsbSet_Change()` procedure ready for you to be edited.

In Visual Basic terminology, the **Change** is called an **Event**. So when the user changes the scrollbar position, the **Change** event occurs automatically, and as a result, the `hsbSet_Change()` procedure is automatically executed.

Similarly, when you attached the End statement to the cmdExit button, you selected **cmdExit** from the **Object** list box, and you selected **Click** from the **Proc** list box. As a result, Visual Basic displayed the `cmdExit_Click()` procedure. In Visual Basic terminology, the process goes as follows:

When the user clicks the cmdExit button, the **Click** event occurs, and as a result, the `cmdExit_Click()` procedure is executed automatically.

Executing the MyScroll Program

You completed the MyScroll program. So let's see it now in action



Select **Start** from the **Run** menu to execute the MyScroll program.

The window of the MyScroll program appears as shown back in Figure 3.1.



Change the scrollbar position, and notice that as you change the scrollbar position, the label is updated accordingly.



Experiment with the MyScroll program, and then click its Exit button to terminate the program.


Final Touch: Dragging the Scrollbar




Are you ready for a "final touch" for the MyScroll program? Here it is:

In the previous page you completed the implementation of the MyScroll program. As the user changes the scrollbar position, the label is updated with a number that represents the current position of the scrollbar.

To see one particular (annoying) limitation in the current state of the MyScroll program, follow the following steps:

 Select **Start** from the **Run** menu to execute the MyScroll program.


Visual Basic responds by executing the MyScroll program.

 Place the mouse cursor on the square indicator of the scrollbar, and drag it with the mouse.

The moment you release the mouse, the label is updated with a number that represents the new position of the scrollbar. But **during** the dragging, the label was **not** updated.

 Click the Exit button to terminate the MyScroll program.

You'll now enhance the MyScroll program so that when you drag the scrollbar, the label is updated **continuously**.

 Select **Project** from the **View** menu to display the Project window, make sure that **frmMyScroll** is highlighted inside the Project window, and then click the **View Form** button.

Visual Basic responds by displaying the frmMyScroll form.

 Double click the scrollbar.

Visual Basic responds by displaying the Code window, and in the **Object** list box of the Code window, the **hsbSet** scrollbar item is selected.



NOTE

Whenever you double click an object, the Code window appears with the **Object** list box set with the name of the object which you double clicked.

For example, in the preceding step you double clicked the **hsbSet** scrollbar. So as a result, the **Code** window appears, and the **Object** list box has **hsbSet** in it.

Had you double clicked the **cmdExit** button, the Code window would have appeared with the **cmdExit** in its **Object** list box.

Ok, you are currently viewing the Code window with **hsbSet** in its **Object** list box. Most probably, inside the **Proc** list box of the Code window that you are now viewing you see the **Change** event. At this point, you want to add code to the **Scroll** event of the scrollbar.



Set the **Proc** list box of the Code window to **Scroll** (and the **Object** list box should be set to **hsbSet**).

Visual Basic responds by displaying the `hsbSet_Scroll()` procedure as follows:

```
Private Sub hsbSet_Scroll()
```

```
End Sub
```

The **Scroll** event occurs automatically whenever the user **drags** the scrollbar.

What code do you want to write inside the `hsbSet_Scroll()` procedure? You want to write code that updates the **lblResult** label with the current position of the scrollbar.



Type code inside the `hsbSet_Scroll()` procedure. After typing the code, the `hsbSet_Scroll()` procedure should look as follows:

```
Private Sub hsbSet_Scroll()
```

=====

In this lesson you implemented a program the utilizes the popular scrollbar control.

In subsequent lessons of this tutorial, you'll continue to explore the fascinating world of Visual Basic.

Frequently Asked Questions



Q1: I used the Label control in this lesson. The Label control displays text. But the label control does not let the user type text inside the label during runtime. Is it possible to set the label so that the user will be able to type code inside the label?

A1: No. The purpose of the Label is to display text, and the user **cannot** type text inside the label. To be able to type text, you have to use the Edit box control (discussed in a later lesson of this tutorial). So remember, Label controls are used for displaying text, and the user cannot type text inside the label.

Q2: In this lesson I placed the scrollbar inside the form. And I was instructed to set the Height property of the scrollbar as shown in Figure 3.1. Can I make the scrollbar wider?

A2: Yes. You can set a new value for the Height property of the scrollbar (or you can use the mouse to drag the handles of the scrollbar). Figure 3.10 for example shows the scrollbar wider.

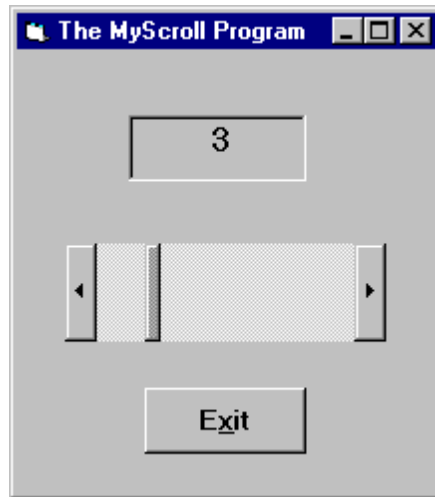


Figure 3.10. *Making the scrollbar wider (not recommended)*

However, it is not recommended to make the scrollbar wider. Why? Because people are used to see the scrollbar as shown in Figure 3.1. If you make your scrollbars wider, your users will either think that something is wrong with the program, or that you are too creative. But of course, if your application justifies the use of wide scrollbars, go for it.

Exam



1. The **Value** property of the scrollbar represents the:

- (a) No such property.
- (b) The current position of the scrollbar
- (c) The price of the control in US Dollars.

2. The **Change** event occurs whenever the scrollbar current position is changed.

- (a) True.
- (b) False.

3. The **Scroll** event occurs whenever the scrollbar current position is changed.

- (a) True.
- (b) False.

4. There is something wrong with the following statement. What 's wrong with it?

```
lblMyLabel.Caption = "This is my lucky number:" + 13
```

5. Is there anything wrong with the following statement?

```
lblMyLabel = "This is my lucky number:" + Str(13)
```

6. Is there anything wrong with the following statement?

```
lblMyLabel.Caption = This is my lucky number: + Str(13)
```

Answers to Exam



- 1. (b) The current position of the scrollbar.

2. (a) The **Change** event occurs whenever the scrollbar current position is changed.

3. (a) The **Scroll** event occurs whenever the scrollbar current position is changed.

4. Do not add a number to a string!

The correct way to write the statement is as follows:

```
lblMyLabel.Caption = "This is my lucky number:" + Str(13)
```

In the preceding statement, the number 13 was converted to a string, and then the result is added to another string. The sum of these two strings (which is also a string) is assigned to the **Caption** property of the lblMyLabel label.

5. Everything is ok.

At first glance, you may think that the statement should be written as follows:

```
lblMyLabel.Caption = "This is my lucky number:" + Str(13)
```

But as it turns out, every control has the so called "default property", or "the most important property". In the case of the **Label** control, the default property is the **Caption** property. So if you omit the name of the property which you are setting, Visual Basic understands it as if you are trying to set the **Caption** property of the Label.

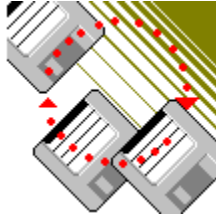
Try to avoid writing code that omits the name of the property, because sooner or later, you will not understand your own code.

6. The statement is invalid!

The string must be enclosed with double quotes as follows:

```
lblMyLabel.Caption = "This is my lucky number:" + Str(13)
```

Project



Enhance the MyScroll program so that it will do the following:

- As the user changes the scrollbar position, the background color of the program's window should change accordingly. Per your design, the scrollbar of the MyScroll program can be set to any integer value between 0 and 15. This means that the scrollbar can have 16 different numbers:

0, 1, 2, ..., 15

Thus, after implementing the enhancement, you'll see 16 different colors as you changes the scrollbar position from 0 to 15.

To be able to perform this enhancement, you have to know the following:


In Visual Basic, color can be represented by the **QBColor()** function.

- For example, the **black** color is specified as follows:
QColor(0)


- The **white** color is specified as follows:
QColor(15)


In other words, you specify an integer between 0 and 15 inside the parenthesis of the QBColor() function, where each integer represents a different color. You already know that 0 represents black, and 15 represents white. What are the colors that correspond to the other integers? After implementing the enhancement, you'll be able to answer this question.

- 👉 Select **Project** from the **View** menu of Visual Basic to display the Project window.

 Click the **View Code** button of the Project window.

Visual Basic responds by displaying the Code window.

 Set the **Object** list box at the top left of the Code window to **hsbSet**, and make sure that the **Proc** list box at the top right of the Code window is set to **Change**.

 Add code inside the `hsbSet_Change()` procedure. After adding the code, the `hsbSet_Change()` procedure should look as follows:

Private Sub hsbSet_Change()

```
lblResult.Caption = Str(hsbSet.Value)
```

```
frmMyScroll.BackColor = QBColor(hsbSet.Value)
```


End Sub


The code that you added to the `hsbSet_Change()` procedure sets the **BackColor** property of the `frmMyScroll` form as follows:

```
frmMyScroll.BackColor = QBColor(hsbSet.Value)
```


In the preceding code you supplied **hsbSet.Value** as the parameter of the **QBColor()** function. For example, if the current position of the scrollbar is on the extreme left (which represents the value **0**), then **hsbSet.Value** is equal to **0**, and therefore **0** is supplied as the parameter of the **QBColor()** function. As stated, **QBColor(0)** represents the **black** color. So putting it altogether, the **BackColor** property of the `frmMyScroll` form is set to **black** when the scrollbar is at the **0** position.


And when the scrollbar position is at another position, the **BackColor** property of the `frmMyScroll` form changes to the color that corresponds to the scrollbar position.

 Select **Project** from the **View** menu of Visual Basic to display the Project window.

 Click the **View Code** button of the Project window.

Visual Basic responds by displaying the Code window.

 Set the **Object** list box at the top left of the Code window to **hsbSet**, and make sure that the **Proc** list box at the top right of the Code window is set to **Scroll**.


 Add code inside the `hsbSet_Scroll()` procedure. After adding the code, the `hsbSet_Scroll()` procedure should look as follows:


Private Sub hsbSet_Scroll()


```
lblResult.Caption = Str(hsbSet.Value)  
frmMyScroll.BackColor = QBColor(hsbSet.Value)
```

End Sub

So now, as the user drags the square of the scrollbar, the background color of the window changes accordingly.

 Select **Save Project** from the **File** menu to save your work.

 Select **Start** from the **Run** menu, and verify that as you change the scrollbar position, the background color of the **frmMyScroll** form changes.

 Experiment with the MyScroll program, and then terminate the program.

How To Contact TegoSoft



You can contact TegoSoft Inc. by any one of the following methods:

- Use TegoSoft Internet Web site:
<http://www.tegosoft.com>
- Send TegoSoft an e-mail:
tegosoft@msn.com
- Send TegoSoft a letter:
TegoSoft Inc.
P.O.Box 389
Bellmore, NY 11710
U.S.A.

Technical Support



If you have a technical question, you can post the question to the TegoSoft Technical Support staff, and they will try to answer your question.

The **best** way to post a technical question is by sending TegoSoft an e-mail.

- The e-mail of TegoSoft is:
tegosoft@msn.com

When sending TegoSoft an e-mail with a technical question, please follow the following format:

Date: _____
Your name: _____
Company (if applicable): _____
Your phone number: _____
Your e-mail: _____
Country (if not U.S.A): _____
State (if inside U.S.A): _____

Operating System used: _____

Programming language and version : _____

My technical question is:

Copyright © and Notices

Copyright © 1994, 1995, 1996 by TegoSoft Inc. ® All Rights Reserved

TegoSoft Self Study Tutorials & Software

Copyright ©1994, 1995, 1996 by TegoSoft Inc. ® All Rights Reserved

Although precaution was taken in the preparation of this document, TegoSoft assumes no responsibility for errors or omissions. TegoSoft is not liable for damages resulting from the use of the information contained in this document. Use this document at your own risk. This document is copyright protected. This means that you should treat this document like any other copyright material. No part of this document should be copied in any way. You are not allowed to use this document or any part of this document for any purpose other than read it. You are not allowed to publish this document or any part of this document in any way. You are not allowed to sell this document or any part of this document, you are not allowed to incorporate this document or any part of this document in any book, magazine, Web Site, Electronic forums, disks, CDs, or any other media. The only thing that you are allowed to do with this document is read it for the sole purpose of studying the material that is presented in this document. If this document includes software, the software must be treated in the exact same way that this document is treated. You are not allowed to distribute the software in any way. The sole purpose of supplying the accompanying software is to enable you to experience with it. No part of this document should be modified. If accompanying software is included with this document, you are not allowed to modify the software.

Rev. 19960607-1