

TegoSoft Inc.

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

P.O.Box 389, Bellmore, NY 11710

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

 e-mail: tegosoft@msn.com

Lesson 22 - Drawing Circles With the Mouse


In this lesson you'll learn to write programs that let your user draw Circles with the mouse.


Creating the Directory of the Project and Saving the Files of the Project


As usual, you'll start by creating the directory of the project, and saving the files of the project.

 Create the **C:\VbMyProg\Lesson22** directory.


You'll save the files of this lesson to this directory.

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

 Make the window of Form1 the selected window, select **Save File As** from the **File** menu of Visual Basic, and save the file as **MyMouse.FRM** inside the **C:\VbMyProg\Lesson22** directory.

 Select **Save Project As** from the **File** menu of Visual Basic, and save the project as **MyMouse.VBP** inside the **C:\VBMyProg\Lesson22** directory.

Always Use Option Explicit

 Inside the general declarations section of Form1 type the following code:

```
' Force variable declarations
Option Explicit
```

The **Option Explicit** statement forces you to declare variables before using the variables.

Designing the Window of the MyMouse Program

You'll now design the window of the MyMouse program so that it will look as shown in Figure 22.1.

 Set the properties of Form1 as follows:

Name: Form1

Caption: The MyMouse Program

Icon: Set the Icon property to the file **Mouse04.ICO** that resides inside the **\Icons\Computer** directory of Visual Basic.

BackColor: White

Implementing the Exit Button

You'll now implement the Exit button:

 Place a CommandButton inside Form1, then set the properties of the CommandButton as follows:

Name: cmdExit

Caption: E&xit

Your Form1 should now look as shown in Figure 22.1

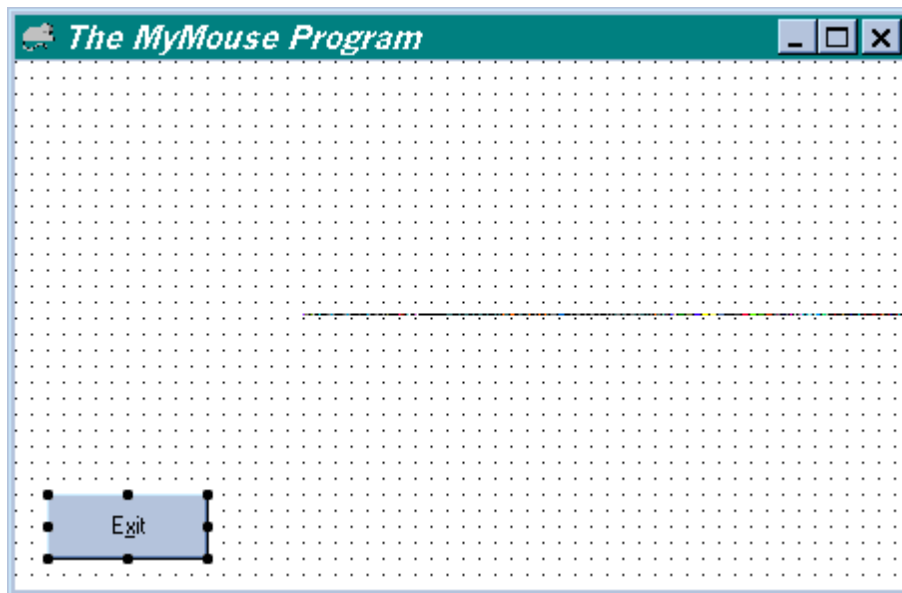



Figure 22.1. Form1 of the MyMouse program in design mode.

 Add the following code inside the **cmdExit_Click()** procedure:

```
Private Sub cmdExit_Click()
```


```
End
```

```
End Sub
```

So whenever the user clicks the Exit button, the MyMouse program terminates itself.

Attaching Code to the MouseDown Event of Form1

You'll now attach code to the **MouseDown** event of Form1. The **MouseDown** event occurs whenever the user presses any of the mouse buttons while the mouse cursor inside the window of the program.

 Double click Form1 to display the Code window, set the **Object** list box of the Code window to **Form**, and set the **Proc** list box of the Code window to **MouseDown**.

Visual Basic responds by displaying the **Form_MouseDown()** procedure ready to be edited by you as follows:

```
Private Sub Form_MouseDown(Button As Integer, _  
                            Shift As Integer, _  
                            X As Single, _  
                            Y As Single)
```

```
End Sub
```

(In the preceding code we spread the first line of the procedure over several lines because the first line is long and does not fit the width of the page. However, in the Code window of Visual Basic you'll see one long line)

 Type code inside the **Form_MouseDown()** procedure so that the procedure will look as follows:

```
Private Sub Form_MouseDown(Button As Integer, _  
                            Shift As Integer, _  
                            X As Single, _
```


=====

Y As Single)


Beep

End Sub


The **Form_MouseDown()** procedure is automatically executed whenever the user presses down any of the button mouse. So the PC will beep whenever the user will place the mouse cursor inside Form1 and will press down any of the mouse buttons. Let's see this in action:

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


 Execute the MyMouse program.

 Place the mouse cursor inside a free area inside the Window of the MyMouse program, and press down the **left** button of the mouse.

The MyMouse program responds by beeping.

 Place the mouse cursor inside a free area inside the Window of the MyMouse program, and press down the **right** button of the mouse.

The MyMouse program responds by beeping.

 Experiment with the MyMouse program, then click its **Exit** button to terminate the program.

Detecting Which Mouse Button is Down

Currently, no matter which button of the mouse you press down, the PC beeps. You'll now write code that causes the PC to beep only if the user presses the left button of the mouse.


=====


 Modify the code inside the **Form_MouseDown()** procedure so that it will look as follows:

```
Private Sub Form_MouseDown(Button As Integer, _  
                            Shift As Integer, _  
                            X As Single, Y As Single)  
  
    If Button = 1 Then  
        Beep  
    End If  
  
End Sub
```


Button is the first parameter of the **Form_MouseDown()** procedure. The program automatically updates the **Button** parameter with a value that represents the mouse button that is down.

When **Button** is equal to **1**, it means that the left button of the mouse is down. The **If** condition is therefore satisfied provided that the user presses the left button of the mouse down (while the mouse cursor is inside a free area of the window's program).


 Select **Save Project** from the **File** menu of Visual Basic to save your work, then execute the MyMouse program.

 Place the mouse cursor inside a free area inside the window of the MyMouse program, then press down the left button of the mouse.

As you can hear, the PC beeps (because the **If** condition inside the **Form_MouseDown()** procedure is satisfied).


 Place the mouse cursor inside a free area inside the window of the MyMouse program, then press down the right button of the mouse.

The PC does **not** beep (because the **If** condition inside the **Form_MouseDown()** is **not** satisfied).

 Click the **Exit** button of the MyMouse program to terminate the program.

Drawing a Circle

You'll now modify the code inside the **Form_MouseDown()** procedure so that every time the user presses down the left button of the mouse, a circle is drawn.

 Double click Form1 to display the Code window, set the **Object** list box to **Form**, and set the **Proc** list box to **MouseDown**. Then modify the code inside the **Form_MouseDown()** procedure so that it will look as follows:

```
Private Sub Form_MouseDown(Button As Integer, _  
                            Shift As Integer, _  
                            X As Single, _  
                            Y As Single)
```

```
If Button = 1 Then
```

```
    Form1.Circle (X, Y), 600
```

```
End If
```

```
End Sub
```

The code under the **If** statement draws a circle as follows:

```
Form1.Circle (X, Y), 600
```

=====

X and **Y** are parameters of the **Form_MouseDown()** procedure. **X** and **Y** are the coordinates of the mouse cursor at the time the user presses the button of the mouse.

The **Circle** method draws a circle. The **(X,Y)** parameter is the coordinates of the center of the circle.

The next parameter that you specified for the **Circle** method is **600**. This is the radius of the circle. So the circle has a radius of 600 units.


The ScaleMode Property of Form1

Take a look at the **ScaleMode** property of Form1. The default value that Visual Basic set for the **ScaleMode** property of Form1 is: **1- Twip**. So the circle will have a radius of **600** twips.


Let's see your code in action:


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

 Execute the MyMouse program.

 Place the mouse cursor inside a free area of the window of the program, and then press down the left button of the mouse.

The MyMouse program responds by drawing a circle.

 Draw additional circles. Every time you press down the left button of the mouse, a circle is drawn with the center of the circle at the point where the mouse cursor is located, and the radius of the circle is 600 twips (see Figure 22.2).

 Experiment with the MyMouse program, then click the Exit button to terminate the program.

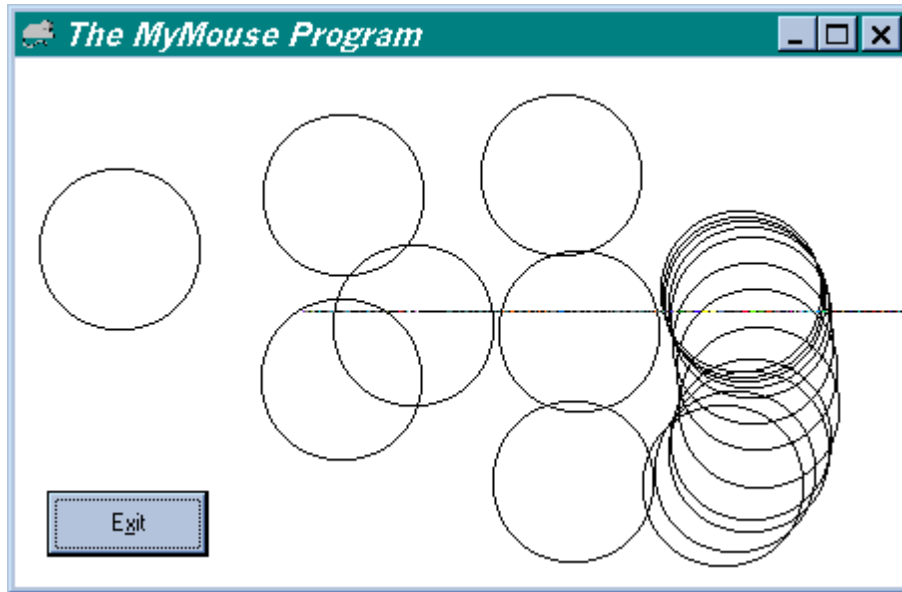


Figure 22.2. Drawing Circles with the MyMouse program.

What You Accomplished in This Lesson



You completed Lesson 22 of the Self Study Visual Basic tutorial. In this lesson you learned to write programs that let the user draw circles with the mouse.

Frequently Asked Questions



Q1. To detect whether the left button of the mouse is pressed down, the **Button** parameter of the **Form_MouseDown()** can be examined as follows:

```
Private Sub Form_MouseDown(Button As Integer, _
                           Shift As Integer, _
                           X As Single, Y As Single)

If Button = 1 Then
    ' Yes, left button of mouse was pressed.
    ...
    ...
    ...
End If

End Sub
```

What code can I write to detect if the right button of the mouse is pressed down?

A1. When **Button** is equal to **1**, it means that the left button of the mouse is down. When **Button** is equal to **2**, it means that the right button of the mouse is down. Thus, the following **If** condition detects if the right button of the mouse is down:

```
Private Sub Form_MouseDown(Button As Integer, _
                           Shift As Integer, _
                           X As Single, Y As Single)

If Button = 2 Then
    ' Yes, right button of mouse was pressed.
    ...
    ...
    ...
```

=====

End If

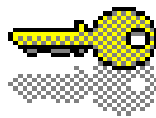
End Sub

Exam



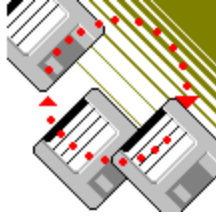
1. The **Button** parameter of the **Form_MouseDown()** procedure is equal to:
 - (a) Always 1
 - (b) Always 2
 - (c) Depends on what button of the mouse is pressed.
 - (d) The **Form_MouseDown()** procedure does not have a parameter called **Button**.

Answers to Exam




1. (c) The **Button** parameter of the **Form_MouseDown()** procedure is equal to: it depends which button of the mouse is pressed.

Project



Enhance the MyMouse program so that when the user presses the left button of the mouse, two circles are drawn. One with radius 600 twips, and the other circle with radius 300 twips.

 Modify the **Form_MouseDown()** procedure so that it will look as follows:

```
Private Sub Form_MouseDown(Button As Integer, _  
                           Shift As Integer, _  
                           X As Single, _  
                           Y As Single)
```

```
    If Button = 1 Then
```

```
        Form1.Circle (X, Y), 600
```

```
        Form1.Circle (X, Y), 300
```

```
    End If
```

```
End Sub
```

You added the following statement:


```
Form1.Circle (X, Y), 300
```


So now two circles will be drawn. Let's see this in action:

=====

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

 Execute the MyMouse program.

 Place the mouse cursor inside the window of the program and note that every time you press down the left button of the mouse, two circles are drawn (see Figure 22.3.)

 Experiment with the MyMouse program, then click the Exit button to terminate the program.

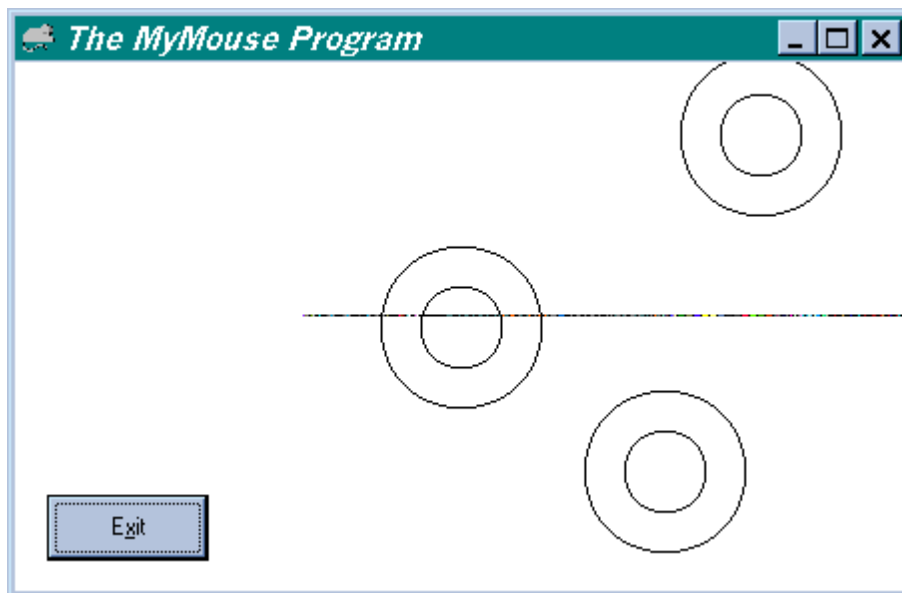
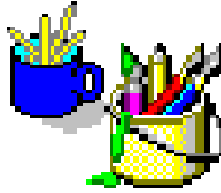



Figure 22.2. *The MyMouse program draws two circles every time the user presses the left button of the mouse.*

Cosmetic Considerations



In the Project section of this lesson you added code that draws a second circle. Currently, both circles are drawn as black circles.

Enhance the MyMouse program so that different colors are used when drawing the circles. Also, make the width of the pen that draws the circles wider.

 Modify the **Form_MouseDown()** procedure so that it will look as follows:

```
Private Sub Form_MouseDown(Button As Integer, _
                            Shift As Integer, _
                            X As Single, _
                            Y As Single)

If Button = 1 Then

    Form1.Circle (X, Y), 600, RGB(255, 0, 0)
    Form1.Circle (X, Y), 300, RGB(0, 255, 0)

End If

End Sub
```

You added the color parameter to the **Circle** method.


The large circle is drawn as a red circle:


```
Form1.Circle (X, Y), 600, RGB(255, 0, 0)
```




The small circle is drawn as a green circle:

```
Form1.Circle (X, Y), 300, RGB(0, 255, 0)
```

 Select **Save Project** from the **File** menu of Visual Basic to save your work. Execute the MyMouse program.

 Place the mouse cursor inside the window of the program and then press the left button of the mouse.

As shown in Figure 22.4, the circles are drawn as red and green circles.

 Experiment with the MyMouse program, then terminate the program.

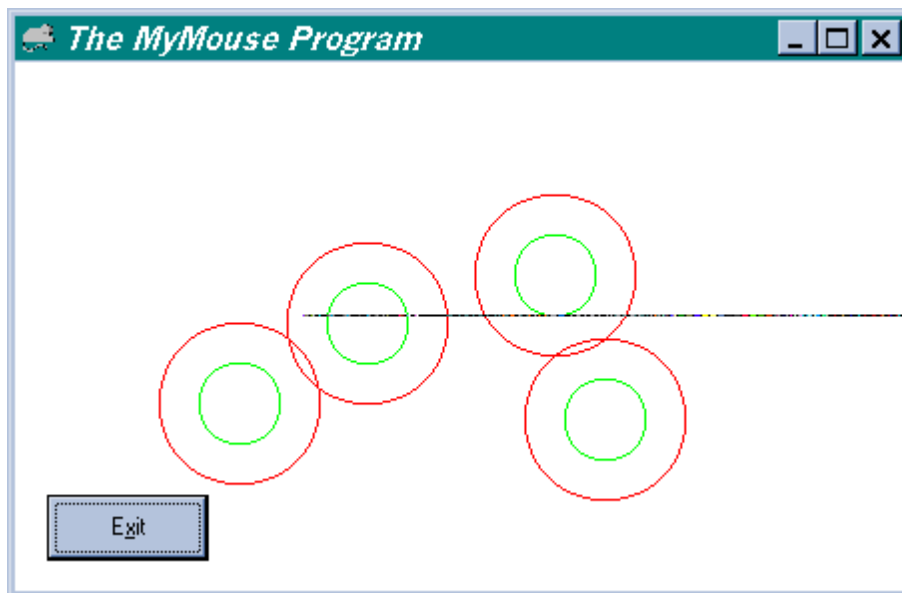




Figure 22.4. *Drawing red and green circles with the MyMouse program.*

To draw the circles with a wider pen, all that you have to do is set the **DrawWidth** property of Form1 to a larger number. The default value of the **DrawWidth** property of Form1 is 1.


=====

 Set the **DrawWidth** property of Form1 to **10**.

 Select **Save Project** from the **File** menu of Visual Basic to save your work. Execute the MyMouse program.

 Place the mouse cursor inside the window of the program and press the left button of the mouse.

As shown in Figure 22.5, the circles are drawn with a wider pen.

 Experiment with the MyMouse program, then terminate the program.

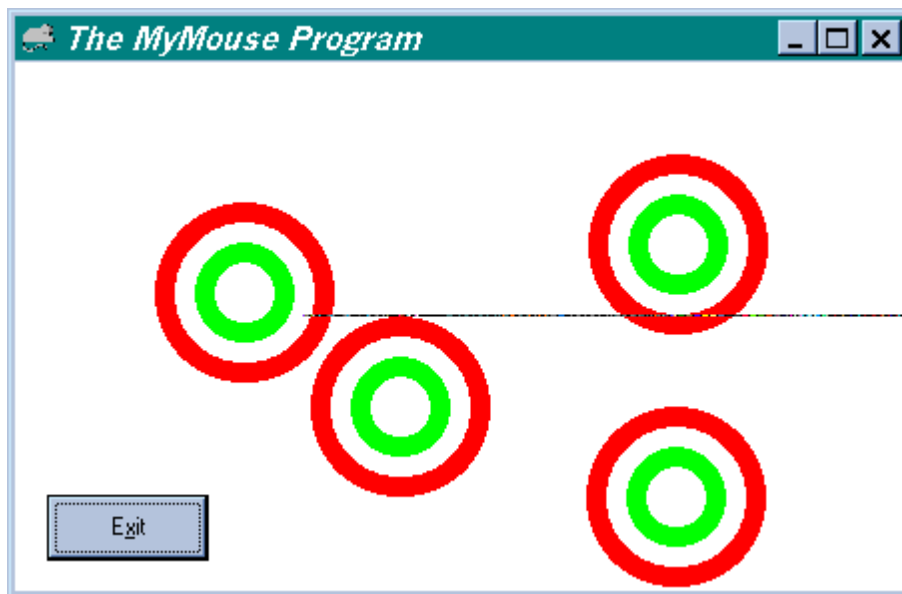


Figure 22.5. *The circles that the MyMouse program draws when the DrawWidth property of Form1 is set to 10.*

Why does the **DrawWidth** property of Form1 effect the width of the pen that is used to draw the circles? Because the **Circle** method "works" on the Form1 object. For example, the large circle is drawn as follows:

```
Form1.Circle (X, Y), 600, RGB(255, 0, 0)
```

Several properties of the object on which the **Circle** method "works" on have effects on the way the **Circle** method works. You already saw that the **DrawWidth** property of Form1 determines the pen width of the pen that will be used for drawing the circles.


Let's take a look at two additional properties that effect the way the **Circle** method works:


The default value that Visual Basic set for the **FillStyle** property of Form1 is: **1-Transparent**

 Set the **FillStyle** property of Form1 to: **0-Solid**


 Set the **FillColor** property of Form1 to the Blue color.

So now the circles will be drawn as filled circles. The color that will be used to fill the circles is the blue color. Let's see this in action:

 Select **Save Project** from the **File** menu of Visual Basic to save your work. Execute the MyMouse program.

 Place the mouse cursor inside the window of the program and press the left button of the mouse.

As shown in Figure 22.6, the circles are drawn as circles that are filled with solid blue paint.

 Experiment with the MyMouse program, then terminate the program.

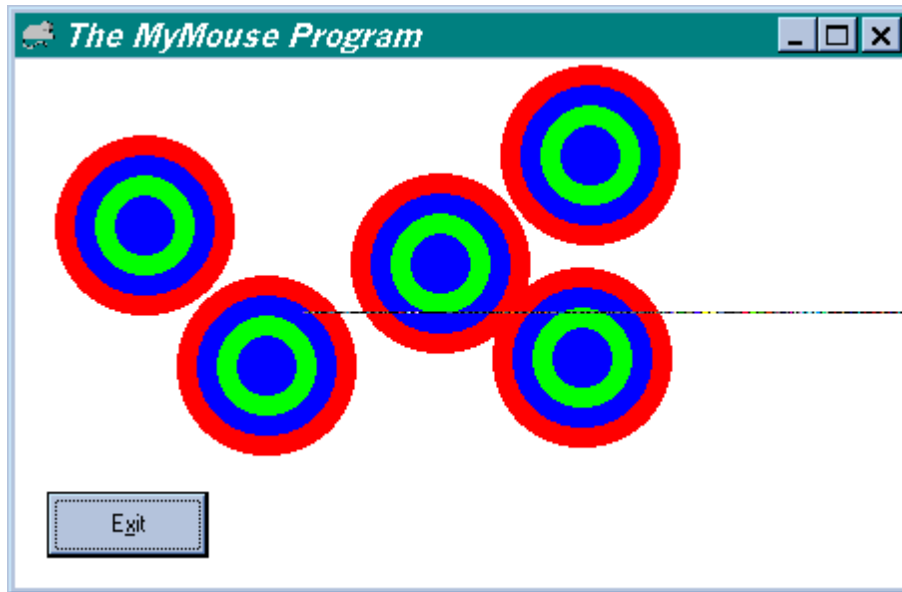


Figure 22.6. The circles are filled with solid blue paint.

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


USA

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 USA): _____
State (if inside USA): _____

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. 031796-1