

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 30 - Repainting Windows


In this lesson you'll learn to write programs that repaints their own windows.


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\Lesson30** 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 **Repaint.FRM** inside the **C:\VBMyProg\Lesson30** directory.

 Select **Save Project As** from the **File** menu of Visual Basic, and save the project as **Repaint.VBP** inside the **C:\VBMyProg\Lesson30** 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 Repaint Program

You'll design the window of the Repaint program so that it will look as shown in Figure 30.1.

 Set the properties of Form1 as follows:

Name: Form1

Caption: The Repaint Program

BackColor: White

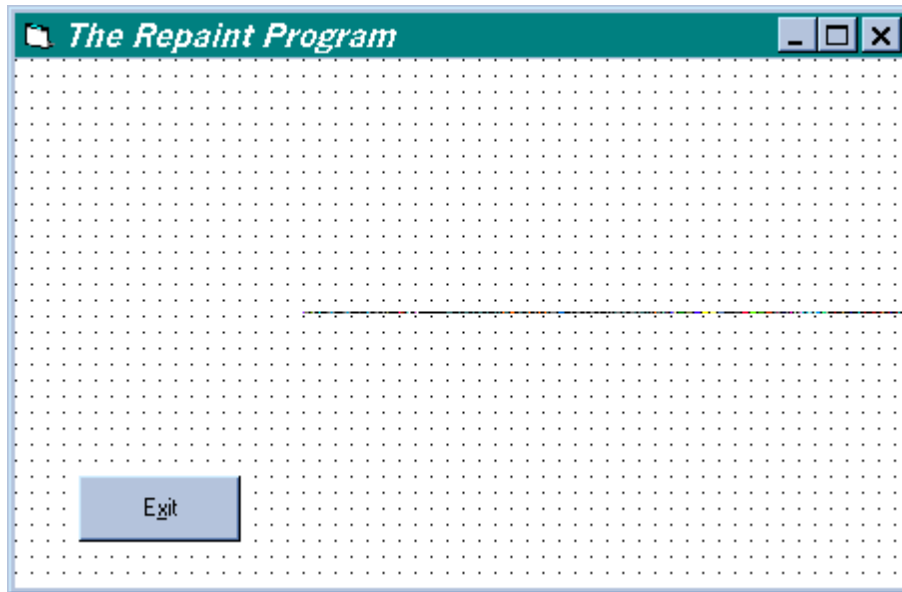



Figure 30.1. Form1 of the Repaint program

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

Attaching Code to the Click Event of the Exit Button

You'll now attach code to the **Click** event of the Exit button:

 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 Repaint program terminates itself.

Adding the DrawCircles Procedure

You'll now add a procedure called **DrawCircles** to Form1.

 Make sure the code window is the selected window, and then select **Procedure** from the **Insert** menu of Visual Basic.

Visual Basic responds by displaying the **Insert Procedure** dialog box.

 Set the **Insert Procedure** dialog box as follows:

Name: DrawCircles

Type: Sub

Scope: Public

All Local variables as Statics check box: Not checked

 Click the OK button of the **Insert Procedure** dialog box.

Visual Basic responds by inserting the **DrawCircles** procedure inside the general section of Form1.

 Type the following code inside the **DrawCircles** procedure:

```
Public Sub DrawCircles()
```

```
=====
```

```
Form1.Circle (500, 500), 300, RGB(255, 0, 0)
Form1.Circle (550, 550), 300, RGB(0, 255, 0)
Form1.Circle (600, 600), 300, RGB(0, 0, 255)
```

End Sub

The code inside the **DrawCircles** procedure draws 3 circles, a red circle, a green circle, and a blue circle.

To make the width of the pen that is used to draw the circles wider, do the following:

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

Now let's write code that executes the **DrawCircles** procedure:

 Type the following code inside the **Form_Click()** procedure:

```
Private Sub Form_Click()

    DrawCircles

End Sub
```

The **Form_Click()** procedure is automatically executed whenever the user clicks the form. Let's see this in action:

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

 Execute the Repaint program.

=====

 Click inside the window of the Repaint program.

The Repaint program responds by drawing the three circles as shown in Figure 30.2.

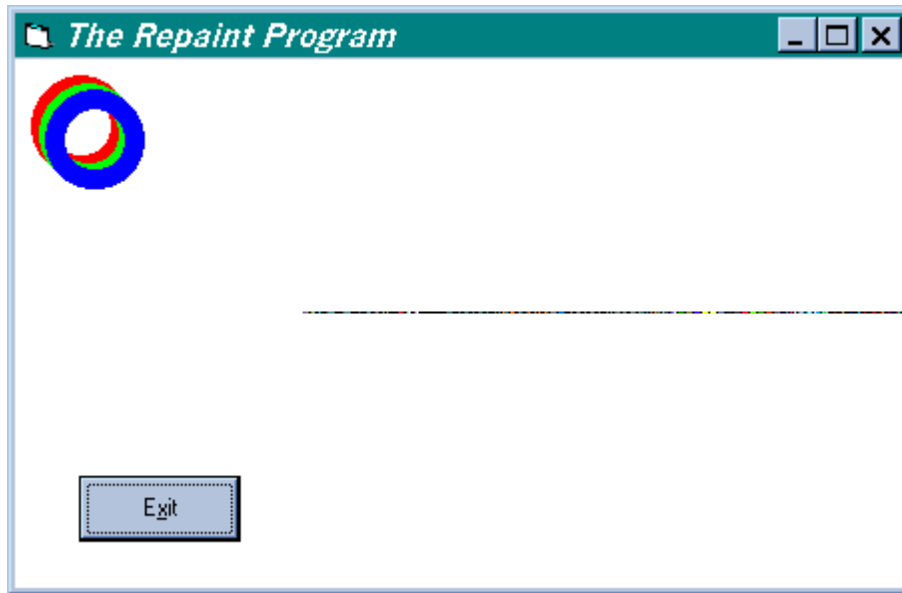




Figure 30.2. After clicking the window of the Repaint program, three circles are drawn.

Now perform the following experiment:

 Minimize the window of the Repaint program, then restore the size of the window.

As you can see, the window of the Repaint program does **not** have the three circles drawn in it. Later in this lesson you will learn how to refresh the window so that upon minimizing and then restoring the window, the three circles will be redrawn. For now, you must click inside the window to refresh the drawing of the three circles.

 Click the Exit button of the Repaint program to terminate the program.

Why Not Use Form_Load()?

In the preceding, you were instructed to attach code to the **Form_Click()** procedure. Whenever the user click the window of the program, the **Form_Click()** procedure is executed, and the three circles are drawn.

Suppose that you want the three circles to be drawn without the need to click the window of the program. From which procedure will you execute the **DrawCircles** procedure? You might think that the **Form_Load()** procedure is a good place to execute the **DrawCircles** procedure. However, if you'll execute the **DrawCircles** from within the **Form_Load()** procedure you will **not** see the three circles! Why? Because the **DrawCircles** procedure uses the following statements:

```
Form1.Circle (500, 500), 300, RGB(255, 0, 0)
Form1.Circle (550, 550), 300, RGB(0, 255, 0)
Form1.Circle (600, 600), 300, RGB(0, 0, 255)
```

That is, the **DrawCircles** procedure "works" on Form1. So Form1 must be ready already when you execute the **DrawCircles** procedure. Form1 is ready only after **Form_Load()** is executed. So if you try to execute the **DrawCircles** procedure from within **Form_Load()** you will **not** see the three circles drawn.

Now that you know from where **not** to execute the **DrawCircles** procedure, the question remains: From which procedure will you execute the **DrawCircles** procedure so that the user will not have to click the window for drawing the three circles? One possible answer is: from within the **Form_Resize()** procedure:



Delete the code inside the **Form_Click()** procedure. After deleting the code, the **Form_Click()** procedure should look as follows:

```
Private Sub Form_Click()

End Sub
```

=====

 Type the following code inside the **Form_Resize()** procedure:

```
Private Sub Form_Resize()
```

```
    DrawCircles
```

```
End Sub
```


The **Form_Resize()** procedure is executed automatically whenever Form1 is resized. Let's see this in action:

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

 Execute the Repaint program.

The window of the Repaint program appears as shown back in Figure 30.2. Yes, the three circles are drawn! When you start the Repaint program, the **Form_Load()** procedure is executed. Then Form1 is displayed. This means that the **Form_Resize()** procedure is executed, because **Form_Resize()** is executed whenever the user resizes the size of Form1, as well as when Form1 is displayed for the first time.


Now, while the Repaint program is still running, perform the following experiment:

 Minimize the window of the Repaint program, and then restore the window of the Repaint program.

As you can see, the three circles are drawn inside the window of the Repaint program. When you restore the size of the window, the **Form_Resize()** procedure is executed (because the window was resized from a minimized state to a normal size), and the code that you typed inside the **Form_Resize()** procedure draws the three circles all over again.

=====

Now perform the following experiment:

 Drag the caption of the window of the Repaint program to the left until a portion of the circles are outside the screen. Then drag the window of the Repaint program back to its original position.

As you can see in Figure 30.3, the portions of the circles that were outside the screen disappear!

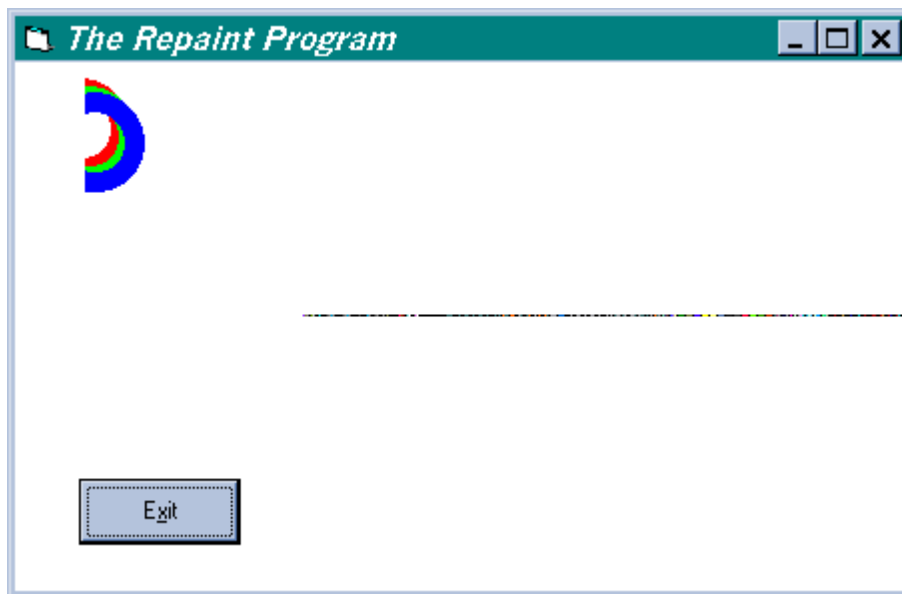



Figure 30.3. *The portions of the circles that were outside the screen disappear!*


The conclusion is that when there is a need to repaint the window of the program (as is the case when you drag the window of the program back after a portion of the window was outside the screen), the program will **not** repaint the window automatically. The same thing will happen if you cover the window of the Repaint program with a window of another application. When you then move the other window and expose the window of the Repaint program, you will **not** see the three circles, because the program does **not** repaint the window.

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

The Paint Method

The **Form_Paint()** procedure is the ultimate place in the program for repainting the window of the application.

The **Paint** event occurs whenever Windows find that it is necessary to repaint the window of the application. Windows will **not** repaint the window for you. Your code has to repaint the window. But at least, your program gets the **Paint** event that is an indication that the window of the application must be repainted. You'll now see this in action:

 Delete the code inside the **Form_Resize()** procedure. After deleting the code, the **Form_Resize()** procedure should look as follows:

```
Private Sub Form_Resize()
```

```
End Sub
```

The reason you are instructed to delete the code inside the **Form_Resize()** procedure, is because the **Form_Paint()** procedure could be the only place in the program to write code that repaints the window.

 Type the following code inside the **Form_Paint()** procedure:

```
Private Sub Form_Paint()
```


```
    DrawCircles
```

```
End Sub
```


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

 Execute the Repaint program.


As you can see, the program displays the three circles (because upon displaying the window of the program for the first time, the **Paint** event occurs).

 Cover the window of the Repaint program with a window of another program, and then expose the window of the Repaint program.


When you expose the window of the Repaint program, the three circles appear, because the **Paint** event occurs, and hence the code inside the **Form_Paint()** procedure is executed.

 Drag the window of the repaint program outside the screen. Then drag the window back to its original position.

Yes, again the **Paint** event occurred, and hence the three circles are redrawn.

 Minimize the window of the Repaint program, and then restore the size of the window.

Again, the **Paint** event occurs and hence the three circles are drawn.

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

What You Accomplished in This Lesson



You completed Lesson 30 of the Self Study Visual Basic tutorial. In this lesson you learned to take advantage of the **Paint** event to repaint the window of the application.

Frequently Asked Questions



Q1. In a previous lesson I learned about the **Refresh** method. For example in the following code, the **Refresh** method is executed inside the **For()** loop. Thus, the program will update the label in each execution of the **For()** loop:

```
For Counter = 1 To 1000

    lblCounter.Caption = Str(Counter)
    Form1.Refresh

Next
```

Will the three circles of the **Repaint** program be drawn if I execute the **Refresh** method?

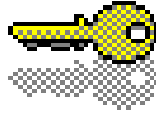
A1. No. The **Refresh** method causes the immediate refreshing of the controls inside Form1. Drawings that were drawn using drawing methods such as a **Line**, **Circle**, **Print** and so on will not be refreshed after executing the **Refresh** method.

Exam



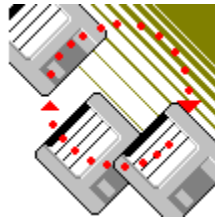
1. The **Form_Resize()** procedure is executed whenever the form is resized and when Form1 is _____
2. The **Form_Paint()** procedure is executed whenever the form has to be repainted, and Form1 is _____

Answers to Exam




1. The **Form_Resize()** procedure is executed whenever the form is resized and when Form1 is displayed for the first time.
2. The **Form_Paint()** procedure is executed whenever the form has to be repainted, and Form1 is displayed for the first time.

Project



As you saw in this lesson, the **Paint** event is useful for repainting the window of the application. You write the code of the repainting inside the **Form_Paint()** procedure. Sometimes, you will need to repaint the window in different ways (depending on the state of the program). This will be demonstrated next:

 Place a Check box control inside Form1 and then set its properties as follows:

Name: chkCircles

Value: 1-Checked

BackColor: White

Your Form1 should now look as shown in Figure 30.4.

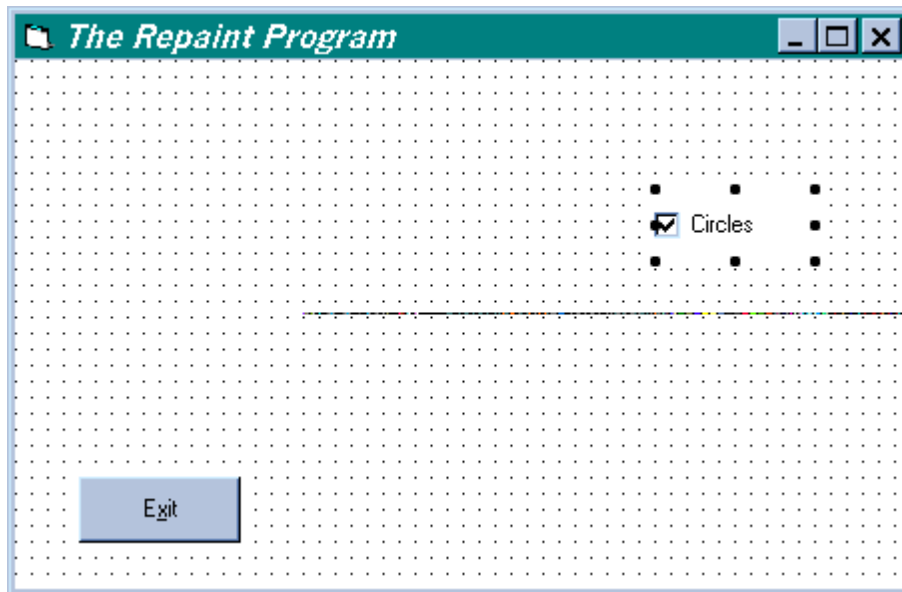



Figure 30.4. Form1 with the check box in it.

The repaint program should function as follows:

- 1 When there is a check mark inside the check box, the three circles should be displayed.
- 2 When there is **no** check mark inside the check box, the three circles should **not** be displayed.

 Type the following code inside the **chkCircles_Click()** procedure:

```
Private Sub chkCircles_Click()
```

```
If chkCircles.Value = 0 Then
```

=====

```
Form1.Cls
End If

If chkCircles.Value = 1 Then
    DrawCircles
End If

End Sub
```

The code that you typed executes two **If** statements. The first **If** statement is satisfied if there is **no** check mark inside the check box. In this case, the **Cls** method is executed (a method that clears the window):

```
Form1.Cls
```


If there is a check mark inside the check box, the second **If** statement is satisfied, and the **DrawCircles** procedure is executed to draw the three circles:

```
DrawCircles
```


Let's see your code in action:


 Execute the Repaint program.

The window of the Repaint program appears with the three circles in it. This is consistent with the status of the check box (when there is a check mark inside the check box, the circles should be displayed).

 Click the check box to remove the check mark from it.


The three circles disappear. This is consistent with the status of the check box (when there is no check mark inside the check box, the circles should not be displayed).

 Click the check box several more times and verify that the circles appear/disappear according to the status of the check box.


 Make sure that there is no check mark inside the check box, and then drag the window of the program outside the screen. Then drag the window back to its original position inside the screen.

When you restore the window to its original position inside the screen, the three circles appear regardless of the status of the check box!

So currently, there is no check mark inside the check box, and yet, the three circles appear.

 Click the Exit button to terminate the program.

Based on the preceding experiment, it is clear that you need to modify the code inside the **Form_Paint()** procedure.

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

```
Private Sub Form_Paint()
```

```
    '''DrawCircles  
    chkCircles_Click
```

```
End Sub
```

The code that you typed inside the **Form_Paint()** procedure comments out the statement that executes the **DrawCircles** procedure.


You added the following statement to the **Form_Paint()** procedure:

```
chkCircles_Click
```


=====

So whenever there is a need to repaint the window, the **chkCircles_Click()** procedure is executed. And as you know, the **chkCircles_Click()** procedure either clears the screen or draws the three circles


 Execute the Repaint program.

 Make sure that there is a check mark inside the check box, and then drag the window of the program outside the screen. Then drag the window back to its original position inside the screen.

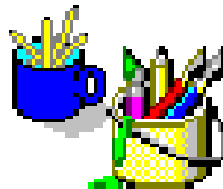
As you can see, the three circles appear (and this is ok, because there is a check mark inside the check box).

 Make sure that there is **no** check mark inside the check box, and then drag the window of the program outside the screen. Then drag the window back to its original position inside the screen.

As you can see, the three circles do **not** appear (and this is ok, because there is **no** check mark inside the check box).

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

Cosmetic Considerations



The code that you write inside the **Form_Paint()** procedure is executed automatically whenever there is a need to repaint the window. Depending on the user actions, the **Form_Paint()** procedure can be executed a lot of times. If the code that you write inside the **Form_Paint()** procedure takes a long time to

=====

execute, the user will notice a delay every time the window of the application has to be repainted. So be careful, and only type code that is necessary for the repainting inside the **Form_Paint()** procedure. In other words, try to write code that makes the process of repainting the window as fast as possible.

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