

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 36 - Message Boxes


In this lesson you'll learn to design programs that utilize the standard message box of Windows. In this lesson you'll also learn to write code that terminates programs properly.


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


As usual for any project that you design with Visual Basic, you'll start by creating the directory of the project, and saving the files of the project.

 Create the **C:\VBMyProg\Lesson36** 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 **MyMsg.FRM** inside the **C:\VBMyProg\Lesson36** directory.

 Select **Save Project As** from the **File** menu of Visual Basic, and save the project as **MyMsg.VBP** inside the **C:\VBMyProg\Lesson36** directory.

Always Use Option Explicit

 Display the general declarations section of Form1 (i.e., double click inside Form1 to display the Code window of Form1. Set the **Object** list box of the Code window to **(General)**, and set the **Proc** list box of the Code window to **(declarations)**). 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.

Implementing the Window of the MyMsg Program


You'll now implement the window of the MyMsg program so that it will look as shown in Figure 36.1.

 Set the properties of Form1 as follows:

Name: Form1

BackColor: White

Caption: The MyMsg Program


 Place a CommandButton inside Form1, and set its properties as follows:

Name: cmdExit

Caption: E&xit

Placing the Display Message Button

You'll now place the **Display Message** button inside Form1.

 Place a **CommandButton** inside Form1, and set its properties as follows:

Name: cmdDisplayMessage

Caption: &Display Message

Your Form1 should now look as shown in Figure 36.1

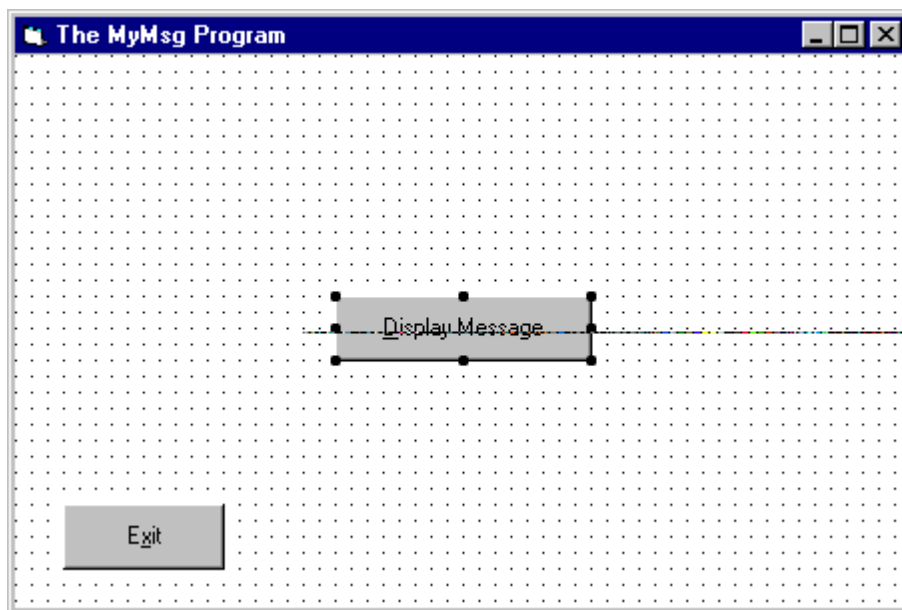


Figure 36.1. Form1 of the MyMsg program.

Attaching Code to the Click Event of the Exit Button

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

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

```
Private Sub cmdExit_Click()
```


```
End
```

```
End Sub
```

So whenever the user clicks the Exit button, the MyMsg program terminates.

Attaching Code to the Click Event of the Display Message Button

You'll now attach code to the **Click** Event of the **Display Message** button.

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

```
Private Sub cmdDisplayMessage_Click()
```

```
MsgBox "Do you like it?"
```

```
End Sub
```

The code that you typed is executed automatically whenever the user clicks the **Display Message** button. This code displays a message box as follows:

```
MsgBox "Do you like it?"
```

Let's see your code in action:

=====

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

 Execute the MyMsg program.

The window of the MyMsg program appears as shown in Figure 36.2.

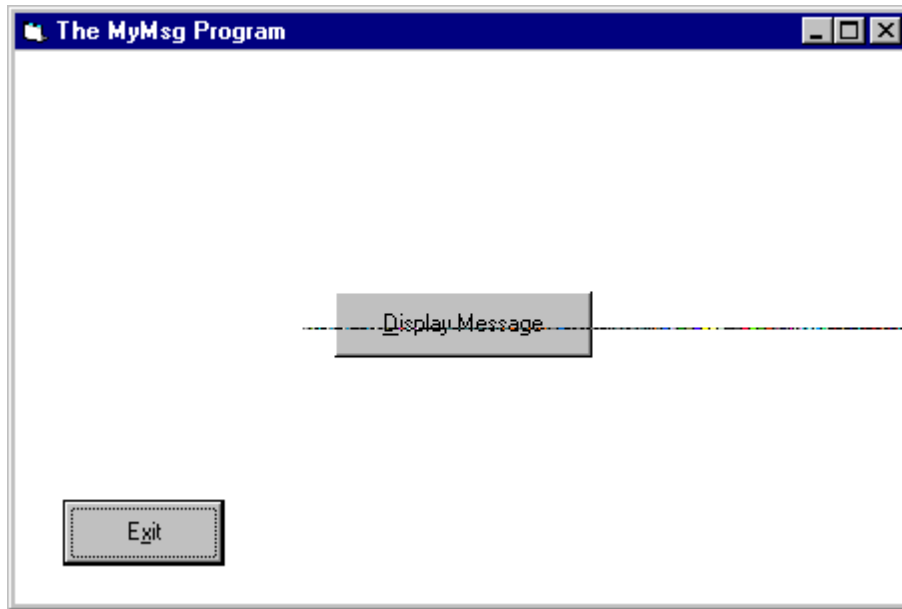



Figure 36.2. *The window of the MyMsg program.*

 Click the **Display Message** button.


The MyMsg program responds by displaying the message box shown in Figure 36.3.




Figure 36.3. *The message box that the MyMsg program displays after clicking the Display Message button.*

 While the message box is displayed, try to switch to the window of the MyMsg window.

As you can see, the program refuses to make the window of the MyMsg window the active window. This means that the message box shown in Figure 36.3 is a modal dialog box.

 Click the OK button of the dialog box shown in Figure 36.3.

The MyMsg program responds by closing the message box.

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

The Parameters of the MsgBox

As you saw in the preceding section, the message box (see Figure 36.2) has the title MyMsg. Also, the message box has the OK button in it.

It is possible to make the message box prettier and with additional buttons in it as will be demonstrated next:

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

```
Private Sub cmdDisplayMessage_Click()
```

```
Dim Answer
```

```
Dim Message
```

```
Dim Title
```

```
Dim ButtonsAndIcon
```

```
Message = "Do you like it?"
Title = "This is my title"
ButtonsAndIcon = _
    vbYesNoCancel + _
    vbDefaultButton2 + _
    vbQuestion
```

MsgBox Message, ButtonsAndIcon, Title

End Sub

The code that you typed declares the following local variables:

```
Dim Answer
Dim Message
Dim Title
Dim ButtonsAndIcon
```

The **Message** variable is set as follows:

```
Message = "Do you like it?"
```

The **Title** variable is set as follows:

```
Title = "This is my title"
```

The **ButtonsAndIcon** variable is set as follows:

```
ButtonsAndIcon = _
    vbYesNoCancel + _
    vbDefaultButton2 + _
    vbQuestion
```

=====

Finally, the **MsgBox** statement is executed as follows:

```
MsgBox Message, ButtonsAndIcon, Title
```

So you supplied **Message** as the first parameter of the **MsgBox**. This means that the body of the message box will display the message: **Do you like it?**


You supplied **ButtonsAndIcon** as the second parameter of the **MsgBox**, and **ButtonsAndIcon** is equal to the following expression:

```
vbYesNoCancel + vbDefaultButton2 + vbQuestion
```

The message box will have the following characteristics:

- The message box includes the **Yes** button, the **No** button and the **Cancel** button.
- Upon displaying the message box, the second button from the left (the **No** button) will be the button with the keyboard focus.
- The message box will have the Question mark icon in it.

Let's see your code in action:

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

 Click the **Display Message** button.

The MyMsg program responds by displaying the message box shown in Figure 36.4.

As you can see in Figure 36.4, now the title of the message box is **This is my title**, the message box has a question mark icon in it, and the message box has three buttons (**Yes**, **No**, and **Cancel**).

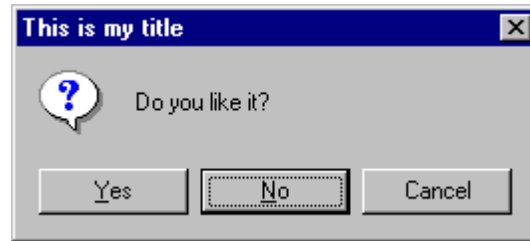



Figure 36.4. *The message box with an icon and buttons in it.*

 Click any of the buttons of the message box.

The MyMsg program responds by closing the message box.

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

Analyzing the User's Respond

The **MsgBox** is often used to display messages to the user. After the user reads the message, the user closes the message box by clicking the OK button of the message box.

You also use the **MsgBox** to display a question to the user, and the user responds by clicking one of the buttons that you placed inside the message box. Your program will then analyze the user response. This will be demonstrated next:

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

```
Private Sub cmdDisplayMessage_Click()
```

```
=====
```

```

Dim Answer
Dim Message
Dim Title
Dim ButtonsAndIcon

Message = "Do you like it?"
Title = "This is my title"
ButtonsAndIcon = _
    vbYesNoCancel + _
    vbDefaultButton2 + _
    vbQuestion

Answer = MsgBox(Message, ButtonsAndIcon, Title)

Select Case Answer
    Case vbYes
        Form1.Print "You answered: Yes"

    Case vbNo
        Form1.Print "You answered: No"

    Case vbCancel
        Form1.Print "You answered: Cancel"
End Select

End Sub

```

You modify the statement that display the message box as follows:

```
Answer = MsgBox(Message, ButtonsAndIcon, Title)
```

=====

So now **MsgBox** returns a value, and the returned value is assigned to the variable **Answer**. Note that because now **MsgBox** returns a value, you have to enclose the parameters of the **MsgBox** with parenthesis.

After the user clicks any of the buttons inside the message box, the program continues with the execution of the statements that follow the **MsgBox** statement. So after the user closes the message box, the following code is executed:

```
Select Case Answer
    Case vbYes
        Form1.Print "You answered: Yes"


    Case vbNo
        Form1.Print "You answered: No"

    Case vbCancel
        Form1.Print "You answered: Cancel"
End Select
```

The **Select Case** analyzes the value of **Answer**. If for example the user clicked the **Yes** button to close the message box, **MsgBox** returns the value **vbYes**, and **Answer** is therefore equal to **vbYes**. The **Case vbYes** is satisfied, and as a result, the **Print** method displays the message: **You answered: Yes**.

If the user clicks the **No** button of the message box, the **Case vbNo** is satisfied, and if the user clicks the **Cancel** button, the **Case vbCancel** is satisfied.

Let's see you code in action:


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

 Click the **Display Message** button.

The MyMsg program responds by displaying the message box shown back in Figure 36.4.

 Click any of the buttons of the message box.

The MyMsg program responds by closing the message box, and inside the window of the MyMsg program, you see a message that tells you which of the buttons inside the message box was clicked.

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

What You Accomplished in This Lesson



In this lesson you learned to implement programs that utilize the standard message box of Windows. This **MsgBox** is used as to display messages and questions to the user, and then the user can answer the questions by clicking the appropriate button inside the message box.

Frequently Asked Questions



Q1. Can I place only the **OK** button inside the message box?

A1. Yes, supply the following as the second parameter of the **MsgBox**:

```
ButtonAndIcon = vbOKOnly + vbQuestion
```

If you supply the preceding as the second parameter of **MsgBox()**, the message box will include an **OK** button and the question mark icon.

Q2. Can I place only the **OK** and **Cancel** buttons inside the message box?

A2. Yes, supply the following as the second parameter of the **MsgBox**:

```
ButtonAndIcon = vbOKCancel + vbQuestion
```

If you supply the preceding as the second parameter of **MsgBox()**, the message box will include an **OK** button, a **Cancel** button, and the question mark icon.

Q3. What other values can I supply to the second parameter of the **MsgBox()**?

A3. When you supply **vbAbortRetryIgnore** as the second parameter of the **MsgBox**, the message box will include the **Abort**, **Retry**, and **Ignore** buttons.

You already saw that you can use the **vbYesNoCancel** to place the **Yes**, **No**, and **Cancel** buttons inside the message box.

When you supply **vbYesNo** to the second parameter of the **MsgBox**, the message box will include the **Yes** and **No** buttons.

When you supply **vbRetryCancel** as the second parameter of **MsgBox**, the message box will include the **Retry** and **Cancel** buttons.

Q4. What other icons can I place inside the message box?

A4. You can display the following icons:

vbCritical (the Critical icon).

=====

vbQuestion (the question mark icon).
vbExclamation (the Exclamation icon).
vbInformation (the i icon).

So for example, to display the **Yes** button, **No** button and the Critical icon inside the message box, set the second parameter of the **MsgBox()** as follows:

```
vbYesNo + vbCritical
```

You cannot display more than one icon inside the message box.

Q5. What are these **vbYesNo**, **vbCritical**, and so on?

A5. **vbYesNo** is equal to **4**. **vbCritical** equals to **16**. So when you supply the second parameter of the **MsgBox** with the value **vbYesNo+vbCritical**, you are supplying **20** (4+16) as the second parameter of the **MsgBox()**. However, to make the program easier to read and understand, Visual Basic has these internal constants declared already for you.

For example, if you see the following statement:

```
Answer = MsgBox(Message, vbYesNo+vbCritical, Title)
```

then you can immediately tell that this message box has the **Yes** and **No** buttons in it, and that the Critical icon appears inside the message box.

The following statement:

```
Answer = MsgBox(Message, 20, Title)
```

produces the same exact message box. However, it is hard to understand from the preceding statement what buttons and icon will be displayed inside the message box.

=====

Q6. In the MyMsg program, I used the constant **vbDefaultButton2** to make the second button of the message box the button with the keyboard focus. Can I make the first button the button with the keyboard focus?

A6. Yes, to make the first button inside the message box the button with the keyboard focus, add the constant **vbDefaultButton1** to the second parameter of the **MsgBox()** function.

For example the statement

```
Answer = _  
    MsgBox(Message, vbYesNo+ vbDefaultButton1, Title)
```

displays the **Yes** button on the left side and the **No** button on the right side. Because you also added **vbDefaultButton1** to the second parameter, the first button from the left (the **Yes** button) will be the button with the default keyboard focus.

To make the second button from the left the default button, use **vbDefaultButton2** instead of **vbDefaultButton1**. If you have three buttons inside the message box and you want to make the third button from the left the default button with the keyboard focus, then use **vbDefaultButton3**.

Q7. I saw that **MsgBox()** can return the **vbOK** value for example. What other values can the **MsgBox()** return?

A8. The **MsgBox()** can return the following values:

vbOK if the user clicks the OK button.

vbCancel if the user clicks the Cancel button.

vbAbort if the user clicks the Abort button.

vbRetry if the user clicks the Retry button.

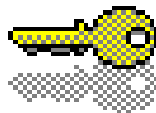
vbIgnore if the user clicks the Ignore button.
vbYes if the user clicks the Yes button.
vbNo if the user clicks the No button.

Exam



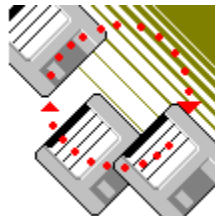
1. Write code that displays the Question mark and the Critical icons inside the message box.

Answers to Exam



1. This was a trick question. You can only place one icon inside the message box.

Project



Enhance MyMsg program so that it will do the following:

- When the user clicks the Exit button, a message box appears that asks the user if he/she really wants to terminate the program.
- If the user clicks the **Yes** button of the message box, the program terminates. If the user clicks the **No** button of the message box, the program does not terminate itself.

Here is how you implement the preceding mechanism:

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

```
Private Sub cmdExit_Click()  
  
    '''End  
    Dim Answer  
    Dim Message  
    Dim ButtonsAndIcon  
    Dim Title  
  
    Message = "Are you sure you want to quit?"  
    ButtonsAndIcon = vbYesNo + vbCritical  
    Title = "Exit"  
  
    Answer = MsgBox(Message, ButtonsAndIcon, Title)  
  
    Select Case Answer  
        Case vbYes  
            End  
  
        Case vbNo
```

```
        ' Do nothing
End Select
```

End Sub

You commented out the **End** statement:

```
'''End
```

You declared the following local variables:

```
Dim Answer
Dim Message
Dim ButtonsAndIcon
Dim Title
```

You set the following values for the local variables:

```
Message = "Are you sure you want to quit?"
ButtonsAndIcon = vbYesNo + vbCritical
Title = "Exit"
```

Then you display the message box as follows:

```
Answer = MsgBox(Message, ButtonsAndIcon, Title)
```

The preceding statement displays the message box with the **Yes** button and **No** button in it. Also, the **vbCritical** constant was added to the **ButtonsAndIcon** variable, so the message box will have the Critical icon in it.

The returned value of **MsgBox()** is assigned to the **Answer** variable. Next, a **Select Case** is executed to examine the value of **Answer**:


=====

```
Select Case Answer
    Case vbYes
        End

    Case vbNo
        ' Do nothing
End Select
```

If the user clicks the **Yes** button of the message box, **Answer** is equal to **vbYes**, and therefore the **Case vbYes** is satisfied. This means that the **End** statement is executed and this causes the program to terminate. If the user clicks the **No** button, **Answer** is equal to **vbNo**, and the **Case vbNo** is executed. You did not write any code under the **Case vbNo** (you wrote only a comment). So the program does not terminate itself when the user clicks the **No** button.

Let's see your code in action:

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

 Click the Exit button.

The MyMsg program responds by displaying the message box shown in Figure 36. 5.

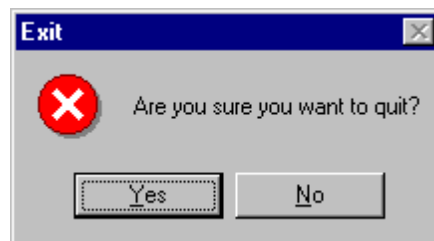




Figure 36.5. *The message box that the MyMsg program displays after the user clicks the Exit button.*

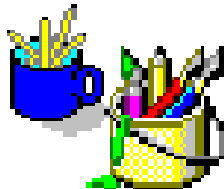
 Click the **No** button.

The MyMsg program does **not** terminate itself.

 Click the Exit button again, and then click the **Yes** button of the message box.

The MyMsg program responds by terminating itself.


Cosmetics and Other Considerations



In some programs, it is a good idea to incorporate a message box that asks the user to confirm that indeed the program should be terminated. However, the way you implemented the confirmation process in the Project section of this lesson has a little flaw in it.

To see the flaw, perform the following experiment:

 Execute the MyMsg program.

 Click the Close icon that appears on the title of the window of the MyMsg program.

The MyMsg program terminates itself. However, the message box that asks the user to confirm the termination does **not** appear (because the user did not click the Exit button).

Your program should be consistent. That is, if you decide that a confirmation message box will appear when the user wants to terminate the program, then the message box should appear no matter how the user decides to terminate the program (by using an Exit menu, an Exit button, the Close icon of the window, or any other way).

Here is how you implement code that will display the confirmation dialog box no matter how the user tries to terminate the program:

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


```
Private Sub cmdExit_Click()
```

```
Unload Form1
```

```
End Sub
```

The code that now appears inside the **cmdExit_Click()** procedure executes the **Unload** method to unload Form1. As a responds, the **Unload** event occurs, which means that the **Form_Unload()** procedure will automatically be executed.

So instead of having the **End** statement inside the **cmdExit_Click()** procedure, your code causes the program to execute the **Form_Unload()** procedure.

 Add code to the **Form_Unload()** procedure. After adding the code, the **Form_Unload()** procedure should look as follows:

```
Private Sub Form_Unload(Cancel As Integer)
```

```
Dim Answer
```

```
Dim Message
```

```
Dim ButtonsAndIcon
```

```
Dim Title
```

```
=====
```

```

Message = "Are you sure you want to quit?"
ButtonsAndIcon = vbYesNo + vbCritical
Title = "Exit"

Answer = MsgBox(Message, ButtonsAndIcon, Title)

Select Case Answer

    Case vbYes
        Cancel = False

    Case vbNo
        Cancel = True

End Select

End Sub

```

If you had not written any code inside the **Form_Unload()** procedure, after the user clicks the Exit button, the **Form_Unload()** procedure would execute, and because there is no code inside the **Form_Unload()** procedure, the program will terminate itself after executing the empty **Form_Unload()** procedure.

Also, when the user clicks the Close icon of the window of the program, the **Form_Unload()** procedure is automatically executed. So if you do not have any code inside the **Form_Unload()** procedure, the program would execute the empty **Form_Unload()** procedure, and then the program would terminate itself.

In other words, no matter how you terminated the program, when Form1 is about to unload itself, the **Form_Unload()** procedure is executed, giving you a last chance to prevent the unloading of the form.

Note that the **Form_Unload()** procedure has the **Cancel** parameter.

=====

```
Private Sub Form_Unload(Cancel As Integer)
...
...
...
End Sub
```

Usually, your code reads the values of the parameters of the procedures, and your code makes use of these parameters. In the case of the **Form_Unload()** procedure, your code can set the value of the **Cancel** parameter to **True** or **False**. If **Cancel** is equal to **True** when the program reaches the last line of the **Form_Unload()** procedure, then the form will not unload itself. In other words, you can write code inside the **Form_Unload()** procedure that sets the **Cancel** variable to **True**, and this will cancel the request to unload the form.

Yet, if you do not set the value of **Cancel** (or set the value of **Cancel** to **False**), then the form will unload itself.

Now let's take a look at the code that you typed inside the **Form_Unload()** procedure:

You declared local variables:

```
Dim Answer
Dim Message
Dim ButtonsAndIcon
Dim Title
```

You set the local variables and display a message box:

```
Message = "Are you sure you want to quit?"
ButtonsAndIcon = vbYesNo + vbCritical
Title = "Exit"
Answer = MsgBox(Message, ButtonsAndIcon, Title)
```

=====

The you execute a **Select Case**:

```
Select Case Answer
```

```
    Case vbYes  
        Cancel = False
```


```
    Case vbNo  
        Cancel = True
```


```
End Select
```

If the user clicked the **Yes** button of the message box (yes, the user wants to exit), then **Answer** is equal to **vbYes**, **Case vbYes** is satisfied, and **Cancel** is set to **False**. This means that when the program reaches the last line of the **Form_Unload()** procedure, the value of **Cancel** is equal to **False**, and the program will continue with the process of unloading the form. When Form1 is unloaded, the program is terminated (because the program has a single form in it, the Form1 form).

On the other hand, if the user clicked the **No** button of the message box, (the user does not want to exit), the value of **Answer** is **vbNo**. This means that the **Case vbNo** is satisfied, and the code under this case sets **Cancel** to **True**. So the program will not unload Form1, and therefore the program will not be terminated.


Let's see your code in action:

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

 Try to terminate the program (by clicking the Exit button or by clicking the Close icon that appears in the title of the window of the

MyMsg program). Either way, the **Form_Unload()** procedure is executed, which causes the program to display the message box.

 Click the **No** button of the message box, and verify that the program does not terminate itself.

 Experiment with the MyMsg program, then terminate the MyMsg 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


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

=====

TegoSoft Visual Basic 4 Self Study Tutorial - Lesson 36

Page 26 of 27

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