

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 24 - Arrays


In this lesson you'll learn to write programs that utilize arrays.


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\Lesson24** 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 **MyArray.FRM** inside the **C:\VBMyProg\Lesson24** directory.

 Select **Save Project As** from the **File** menu of Visual Basic, and save the project as **MyArray.VBP** inside the **C:\VBMyProg\Lesson24** 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 MyArray Program

You'll now design the window of the MyArray program so that it will look as shown in Figure 24.1.

 Set the properties of Form1 as follows:

Name: Form1

Caption: The MyArray Program

BackColor: White

ForeColor: Red

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

Implementing the Array Button

You'll now place the Array CommandButton inside Form1:

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

Name: cmdArray

Caption: &Array

Your Form1 should now look as shown in Figure 24.1

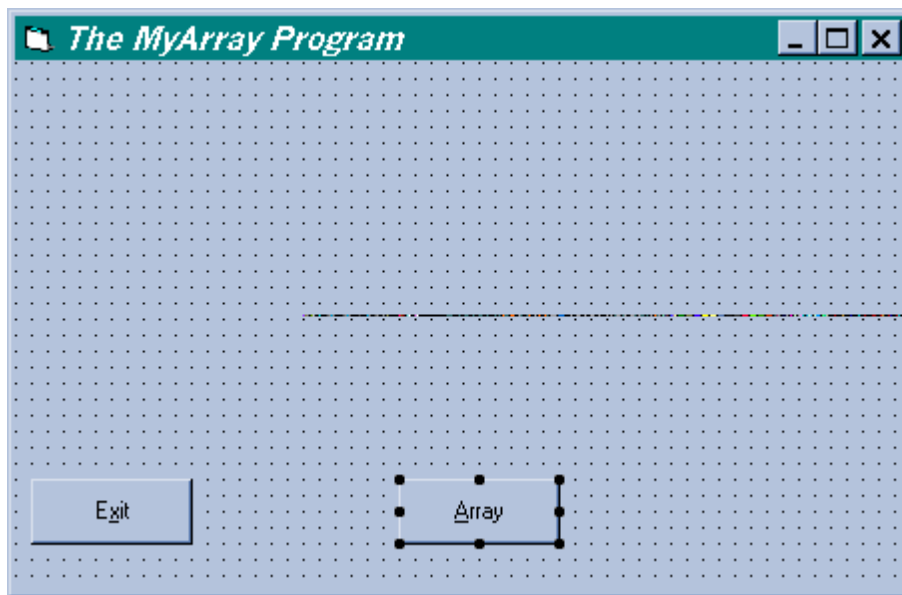



Figure 24.1. Form1 of the MyArray program in design mode.

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 MyArray program terminates itself.

Creating an Array of Variables

You'll now create an array of variables.

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

```
Private Sub cmdArray_Click()  
  
Dim MyArray(4)  
Dim Counter  
  
MyArray(0) = "Element 0"  
MyArray(1) = "Element 1"  
MyArray(2) = "Element 2"  
MyArray(3) = "Element 3"  
MyArray(4) = "Element 4"
```

=====

```
Counter = 0
Do While Counter < 5
    Print "MyArray(" + Str(Counter) + ")=" + MyArray(Counter)
    Counter = Counter + 1
Loop

End Sub
```

Let's go over the code that you typed:

You typed the following declaration:

```
Dim MyArray(4)
```

This means that an array was created. The array is composed of **5** elements.

The first element is **MyArray(0)**

The second element is **MyArray(1)**

The third element is **MyArray(2)**

The fourth element is **MyArray(3)**

The fifth element is **MyArray(4)**



NOTE

The default indexing mechanism of the **MyArray** array that you created is 0-based. This means that the first element of the array is **MyArray(0)** (not MyArray(1)).

And the last element of the array is **MyArray(4)**

Actually, when you declare the **MyArray** array as follows:

=====

```
Dim MyArray(4)
```

It is as if you typed the following:

```
Dim MyArray(0 To 4)
```

The preceding statement (where **0 To 4** is mentioned) is the full syntax. However, you can omit the **0 To** words as you did inside the **cmdArray_Click()** procedure,

The next statement that you typed inside the **cmdArray_Click()** procedure declares another local variable:

```
Dim Counter
```

Then you typed code that fills the five elements of the array:

```
MyArray(0) = "Element 0"  
MyArray(1) = "Element 1"  
MyArray(2) = "Element 2"  
MyArray(3) = "Element 3"  
MyArray(4) = "Element 4"
```

So the first element of the array is filled with the string **Element 0**, the second element of the array is filled with the string **Element 1**, and so on.

You initialize the **Counter** variable to **0**:

```
Counter = 0
```

Then you execute a **Do While** loop:

```
Do While Counter < 5
```

```
=====
```

```
Print "MyArray(" + Str(Counter) + ")=" + MyArray(Counter)
Counter = Counter + 1
```

Loop

The first line of the **Do While** loop specifies the condition: **Counter < 5** This means that the body of the **Do While** loop will be executed for as long as **Counter** is less than **5**.

The first statement inside the **Do While** loop prints a string:

```
Print "MyArray(" + Str(Counter) + ")=" + MyArray(Counter)
```

For example, when the **Do While** loop is executed for the first time, **Counter** is equal to **0**, Hence, the first statement inside the **Do While** loop prints the following:

```
MyArray(0)=MyArray(0)
```

Because **MyArray(0)** is equal to **Element 0**, the **Print** statement displays the string:

```
MyArray(0)=Element 0
```

The second statement inside the **Do While** loop increases **Counter** by **1**:

```
Counter = Counter + 1
```

So now **Counter** is equal to **1** ($0+1=1$).

The program then encounters the **Loop** statement. This means that the program returns to the first line of the **Do While** loop. The condition on the first line of the **Do While** loop is checked. Currently, **Counter** is equal to **1**. The **Do While** condition specifies the condition **Counter<5**. This means that the body of the **Do While** loop is executed again.

=====

Now the Print statement prints the following string:

```
MyArray(1)=MyArray(1)
```

Because **MyArray(1)** is equal to **Element 1**, the Print statement displays the string:

```
MyArray(1)=Element 1
```

This process continues until **Counter** is equal to **5**. When **Counter** is equal to **5**, the **Do While** condition is not satisfied any more, and the **Do While** loop is terminated.

Putting it all together, the **Do While** loop prints the values of the five elements of the **MyArray** array as follows:

```
MyArray(0)=Element 1  
MyArray(1)=Element 2  
MyArray(2)=Element 3  
MyArray(3)=Element 4  
MyArray(4)=Element 5
```

Let's see your code in action:


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

 Execute the MyArray program.

 Click the **Array** button.

=====

The MyArray program responds by displaying the elements of the array (see Figure 24.2).

 Click the **Exit** button to terminate the MyArray program.

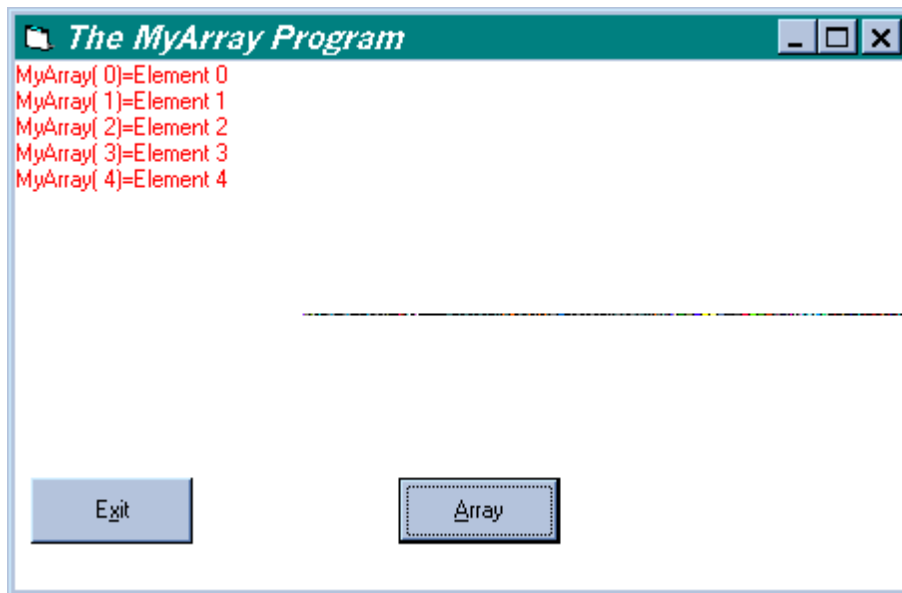


Figure 24.2. *Displaying the elements of the array.*

Re-dimensioning the Array

You declared the **MyArray** array as follows:

```
Dim MyArray(4)
```

The preceding statement means that there is a total of five elements in the array.

Sometimes, you will want to change the dimension of the array during the execution of the program. For example, you may want to start by allocating 5 element of the array, and then later to allocate only 3 elements to the array. It is possible to do this in Visual Basic as follows:

 Modify the code of the **cmdArray_Click()** procedure so that it will look as follows:

```
Private Sub cmdArray_Click()  
  
    'Dim MyArray(4)  
    Dim MyArray()  
  
    Dim Counter  
  
    ReDim MyArray(4)  
    MyArray(0) = "Element 0"  
    MyArray(1) = "Element 1"  
    MyArray(2) = "Element 2"  
    MyArray(3) = "Element 3"  
    MyArray(4) = "Element 4"  
  
    Counter = 0  
    Do While Counter < 5  
        Print "MyArray(" + Str(Counter) + ")=" + MyArray(Counter)  
        Counter = Counter + 1  
    Loop  
  
    ReDim MyArray(2)  
    Counter = 0  
    Do While Counter < 3  
        Print "MyArray(" + Str(Counter) + ")=" + MyArray(Counter)  
        Counter = Counter + 1  
    Loop  
  
End Sub
```

Let's go over the code that you typed:

You commented out the statement that declares five elements in the array:

```
'Dim MyArray(4)
```

Instead, you declare the array as follows:

```
Dim MyArray()
```

So at this point in the program, the array cannot be used, because the program does not know how many elements to allocate for the array. (You declared a dynamic array).

The local variable declaration remains as it was before:

```
Dim Counter
```

Next, you use the **ReDim** statement to allocate five elements to the array:

```
ReDim MyArray(4)
```



NOTE

The **ReDim** statement allocates elements in the array.

Example:

The following statement allocates five elements to the array:

```
ReDim MyArray(4)
```

Now that the array has five elements, you filled the elements with strings:

=====

```
MyArray(0) = "Element 0"  
MyArray(1) = "Element 1"  
MyArray(2) = "Element 2"  
MyArray(3) = "Element 3"  
MyArray(4) = "Element 4"
```

A **Do While** loop then displays the five elements of the array:

```
Counter = 0  
Do While Counter < 5  
    Print "MyArray(" + Str(Counter) + ")=" + MyArray(Counter)  
    Counter = Counter + 1  
Loop
```

And now to the interesting part:

You again use the **ReDim** statement:

```
ReDim MyArray(2)
```


So now the array has only **3** elements in it (**MyArray(0)**, **MyArray(1)** and **MyArray(2)**). If you try to access the **MyArray(3)** element for example, you'll get an error message, because the **MyArray** has only 3 elements in it.

Finally, you display the three elements of the **MyArray** array:

```
Counter = 0  
Do While Counter < 3  
    Print "MyArray(" + Str(Counter) + ")=" + MyArray(Counter)  
    Counter = Counter + 1  
Loop
```

Let's see your code in action:

=====

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

 Execute the MyArray program.

 Click the **Array** button.

The MyArray program responds by displaying the five elements of the array, and then the program displays the first three elements of the new array (the new array has only three elements).

The thing to now here is that once you re-dimensioned the array, the previous contents of the elements is deleted. This is the reason you see in Figure 24.3 null values for the first three elements of the new array.

In the next section you'll learn how to write code that preserves the contents of the elements after re-dimensioning the array.

 Click the Exit button to terminate the MyArray program.

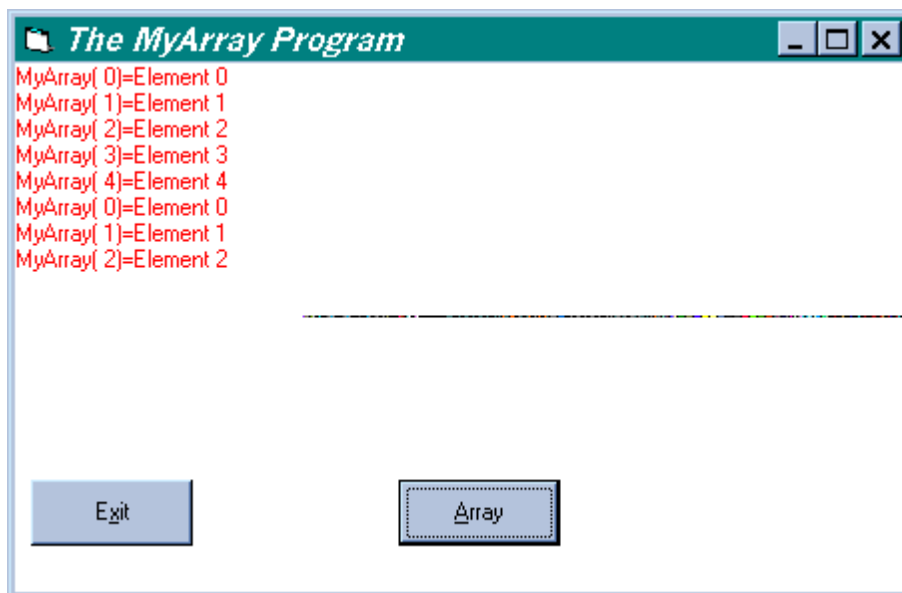



Figure 24.3. Displaying the elements of the MyArray array.

Preserving the Elements of the Array

In the previous section you saw that once you re-dimension the array, the contents of the elements is lost. The new elements of the new array are filled with null.

You'll now write code that preserves the contents of the elements of the new array:

 Modify the **ReDim** statement inside the **cmdArray_Click()** procedure so that it will look as follows:

```
Private Sub cmdArray_Click()  
...  
...  
...  
  
ReDim Preserve MyArray(2)  
...  
...  
...  
  
End Sub
```

In the preceding code, you replaced the statement:

```
ReDim MyArray(2)
```

with the statement:

```
ReDim Preserve MyArray(2)
```

=====

The **Preserve** keyword causes the new three elements to preserve their old values. So the new arrays has the following elements:

MyArray(0) = "Element 0"

MyArray(1) = "Element 1"

MyArray(2) = "Element 2"

Of course, the values of **MyArray(3)** and **MyArray(4)** are lost forever, because the new array does not have these elements any more.


Let's see your code in action:

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

 Execute the MyArray program.

 Click the **Array** button.

As shown in Figure 24.4., the MyArray program first display the original five elements the array, and then the three new elements of the new array. As you can see, the new three elements preserved their old values.

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



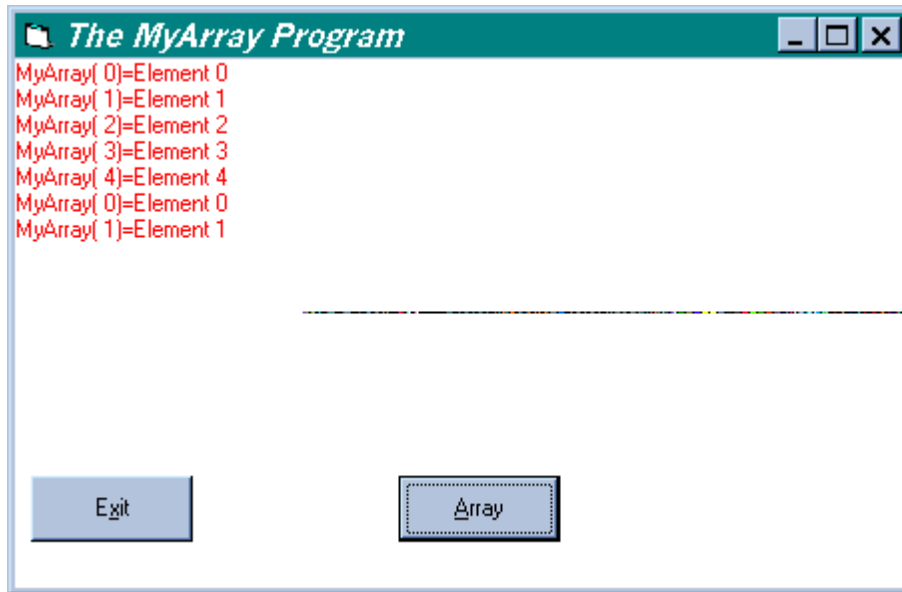


Figure 24.4. The new three elements preserved their old values.

What You Accomplished in This Lesson



You completed Lesson 24 of the Self Study Visual Basic tutorial. In this lesson you learned to write programs that utilize arrays. You learned about static arrays (an array which you have to allocate the number of elements in the **Dim** statement), and you learned about dynamic arrays (an array which you allocate elements to it with the **ReDim** statement). You also learned how to preserve the values of the elements of a dynamic array with the **Preserve** keyword.

Frequently Asked Questions



Q1. The MyArray program displays the elements of the **MyArray** array using a **Do While** loop. Can I use the **For** loop to display the elements of the array?

A1. Yes, in this case, you'll use the following code:

```
For Counter = 0 To 4
    Print "MyArray(" + Str(Counter) + ")=" + MyArray(Counter)
Next
```

Q2. I Set the **ForeColor** of Form1 to the red color. As a result, the **Print** method prints the strings using the red color. What is the correlation between the **ForeColor** property of Form1 and the **Print** method?

A2. You execute the **Print** method as follows:

```
Print "MyArray(" + Str(Counter) + ")=" + MyArray(Counter)
```

Actually, you have to specify the object in which the **Print** method will "work" on.

So the preceding **Print** statement should actually be typed as follows:

```
Form1.Print "MyArray(" + Str(Counter) + ")=" + MyArray(Counter)
```

When you omit the name of the object, the default object is used (which in this case is Form1).

Now back to the **ForeColor** property of Form1: The **Print** method uses the color specified as the **ForeColor** method of the object Form1. You set the **ForeColor** property of Form1 to red, hence the **Print** method uses red color.

Q3. I declared the **MyArray** array as follows:

```
Dim MyArray(4)
```

Can I declare the array in a more specific manner?

=====

Q3. Yes. For example, to declare an array of strings, use the following statement:

```
Dim MyArray(4) as String
```

To declare an array of integers, use the following statement:

```
Dim MyArray(4) as Integer
```

and so on.

Exam



1. How many elements are allocated for the following array?

```
Dim HerArray (10)
```

Answers to Exam



1. The statement

```
Dim HerArray (10)
```

allocates 11 elements for the array, because the default base of the indexing mechanism is **0**.

To change the index base mechanism to **1**, you have to type the following statement inside the general declarations section:

```
Option Base 1
```

In this case, the first element is **HerArray(1)**, the second element is **HerArray(2)**, the so on. The last element is **HerArray(10)**. So if you include the **Option Base 1** statement, then the statement

```
Dim HerArray (10)
```

means that there are **10** elements in the array.

If you do not add any **Option Base** statement inside the general declarations section, it is as if you included the following statement inside the general declarations section:

```
Option Base 0
```

Are you confused by this **Option Base** statement? If so, use the full syntax for declaring or re-dimensioning arrays.

To declare a static array, use the following:

```
Dim TheirArray (3 To 7)
```

In the preceding you declared an array where the first element is **TheirArray(3)**, the second element is **TheirArray(4)**, and so on. The last element is **TheirArray(7)**. This is the case regardless of the **Option Base**

=====

statement (because you use the full syntax where you specify the first and last indexes of the elements).

Like wise, you can declare an array as follows:

```
Dim MyArray (0 to 2)
```

Regardless of what is the situation with the **Option Base** statement, the first element of the array is **MyArray(0)**, the second element is **MyArray(1)**, and the last element is **MyArray(2)**.

The same applies for the **ReDim** statement, where you can use the full syntax statement.

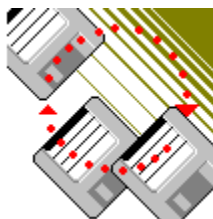
```
' Declare a dynamic array  
Dim MyDynamicArray()
```

Then you allocate elements to the dynamic array as follows:

```
ReDim Preserve MyDynamicArray(0 To 2)
```

The first element is **MyDynamicArray(0)**, the second element is **MyDynamicArray(1)**, and the last element is **MyDynamicArray(2)**.

Project




Currently, the MyArray program demonstrates a feature (the array feature) by displaying string with the **Print** method.

In your future projects, you will occasionally want to test features and concepts by implementing a small program that tests something, and the **Print** method is a convenient way to display the results of the test.

However, the **Print** method prints strings line after line starting at the top of the window. If you want to perform the test again, the **Print** method will print on the next line on the display (not starting from the top line of the window). You can see this yourself by clicking the **Array** button of the MyArray program.

After filling the display, you'll have so much printing inside the window, that you will not know what is going on. The solution is to clear the window before printing, so that the printing will always start at the top line of the window.

Here is how you clear the window from any previous printing:

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

```
Private Sub cmdArray_Click()  
...  
...  
...  
  
Form1.Cls  
  
Counter = 0  
Do While Counter < 5  
    Print "MyArray(" + Str(Counter) + ")=" + MyArray(Counter)  
    Counter = Counter + 1  
Loop
```

=====


...
...
...


End Sub


So before you start the printing, you execute the **Cls** method on Form1 as follows:

```
Form1.Cls
```

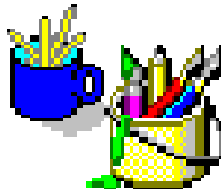
The **Cls** method clears any previous printing inside Form1.

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

 Click the **Array** button, and notice that when you click the **Array** button again, the window is cleared, and then the printing starts at the top of the window.

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

Cosmetic Considerations



The MyArray program demonstrates the use of a single dimension array.

In Visual Basic, you can declare your array as multidimensional arrays. Up to 60 dimensions are allowed.

=====

For example, the following declares an array called **OurArray**:

```
Dim OurArray (2, 3)
```

The preceding statement declares a **2** by **3** array. This array has two dimensions. All together, the array has **6** elements. Here are the 6 elements of the array:

```
(0,0) (0,1) (0,2)  
(1,0) (1,1) (1,2)
```

To set the **(0,1)** element for example to **I am the 0,1 element**, as follows:

```
OurArray(0,1,)= "I am the 0,1 element"
```

Although you can use array with additional dimensions, usually, the program becomes very hard to write, read, and understand if the program uses arrays with more than 2 dimensions.

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