Passing an Array to and from a Visual Basic 6 Function or Subprocedure

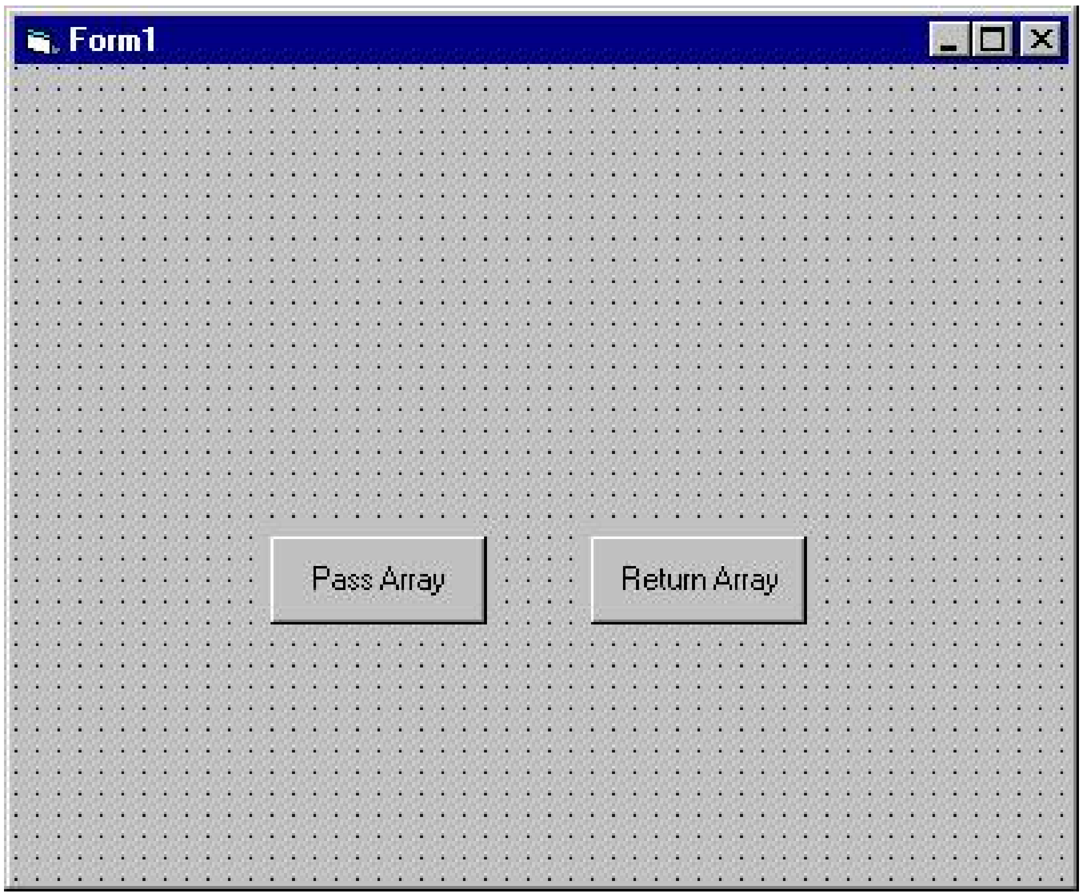
In last month's article, I discussed how to use a For…Each statement to 'loop' through the elements of an Array. In this article, I'll continue my discussion of Arrays by showing you how to pass an Array, as an argument to a procedure or a function, and also how to pass back a return value that is itself an Array.

But why?

That's the question a lot of my students ask---why would you want to pass an argument that is an Array?

The answer to this question presupposes a level of programming sophistication that is not always present in the beginner's mind---quite simply, beginners don't realize the many benefits of Array processing to begin with, so it's only natural that they wouldn't conceive of passing an entire array to a procedure or function for processing---but it's something that's done all the time. In fact, if for no other reason, you should get comfortable with the notion since API function and procedure calls frequently require that you pass an Array of some kind to them, and may also have, as a return value, an Array.

Let's first see how we can create a procedure that accepts, as an argument, an Array. Let's start by creating a form that has two command buttons, which I'll name cmdPassArray and cmdAcceptArray…



We'll place code in the cmdPassArray command button to pass an array to a procedure we'll write. We'll then place code in the cmdReturnArray command button to call a function, and to receive as a return value from that function, an array.

Write the code to pass the Array

Let's start out by writing the code for cmdPassArray first. Here's the code…

```
Private Sub cmdPassArray_Click()

Dim x(3) As Integer        'Declare a Static Integer Array of 4 elements

x(0) = 10
x(1) = 20
x(2) = 30
x(3) = 40

Call AcceptArray(x)        'Call the procedure and pass the Array

End Sub
```

Most of this code is not earthshaking. This line of code initializes a four element Integer

Array called x…

```
Dim x(3) As Integer          'Declare a Static Integer Array of 4 elements
```

The next four lines of code assign values to each of the four elements of the array…

```
x(0) = 10
x(1) = 20
x(2) = 30
x(3) = 40
```

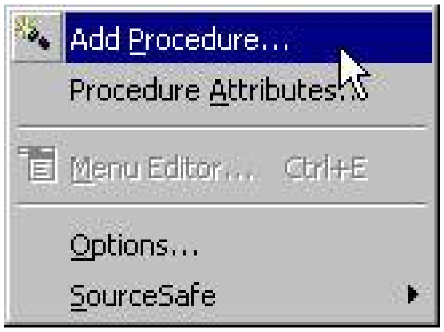
And this line of code calls the procedure 'AcceptArray' and passes it a single argument---the array 'x'….

```
Call AcceptArray(x)          'Call the procedure and pass the Array
```

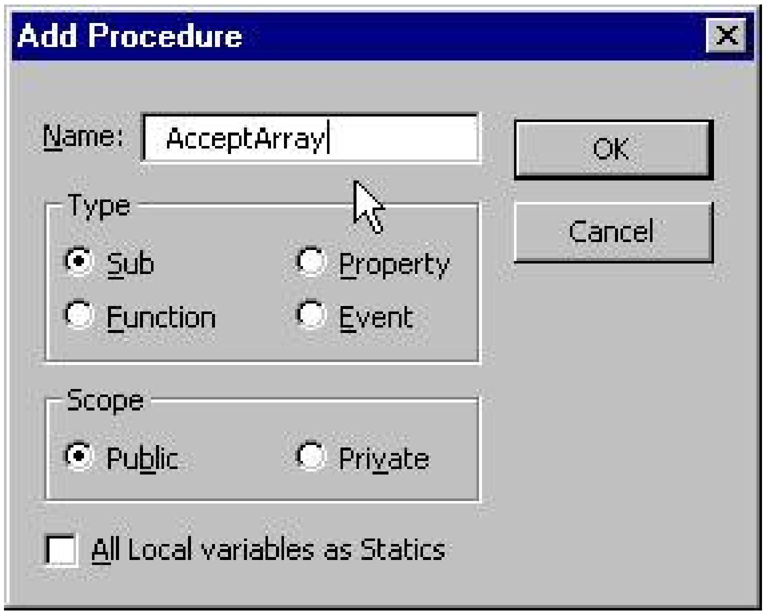
Notice how it appears that we are passing the value of the variable x---but since x is actually an Array, this syntax will pass each element of the Array to the procedure 'AcceptArray'.

Write the SubProcedure to accept the Array as an argument

Let's write the AcceptArray procedure now, and we'll see how we handle accepting the Array as an argument. To create a procedure in Visual Basic, select Tools-Add Procedure from the Visual Basic Menu Bar…



When the Add Procedure Dialog Box appears, we can then complete it with the name of our Procedure---AcceptArray---and also specify its Type as 'Sub--meaning that it is a SubProcedure that does not return a value…



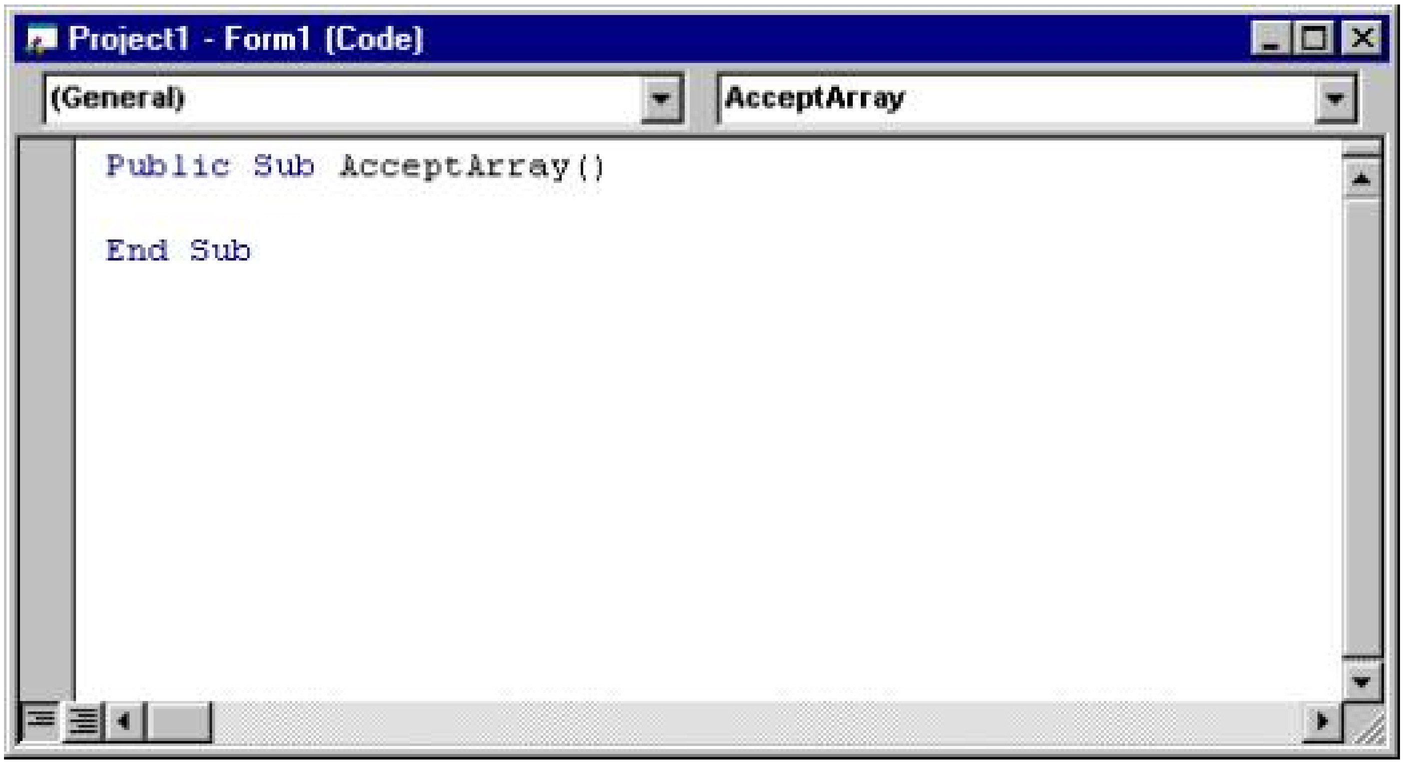
Name: AcceptArray

Type: [✓ Sub] Property Function Event

Scope: [✓ Public] Private

All Local variables as Statics

If we now click on the OK button, Visual Basic will create the SubProcedure 'stub' for the AcceptArray SubProcedure…



```
Public Sub AcceptArray()

End Sub
```

Private Sub AcceptArray(intArray() As Integer)

Dim obj As Variant

```
For Each obj In intArray
   Form1.Print obj
Next
```

```
End Sub
```

This code should look familiar to you--it's similar to the code from last month's article where I described how to use the For…Each statement to loop through the elements of an Array. The key is the Procedure header in which we 'tell' Visual Basic to accept an Array through the use of the empty set of parentheses following the argument intArray…

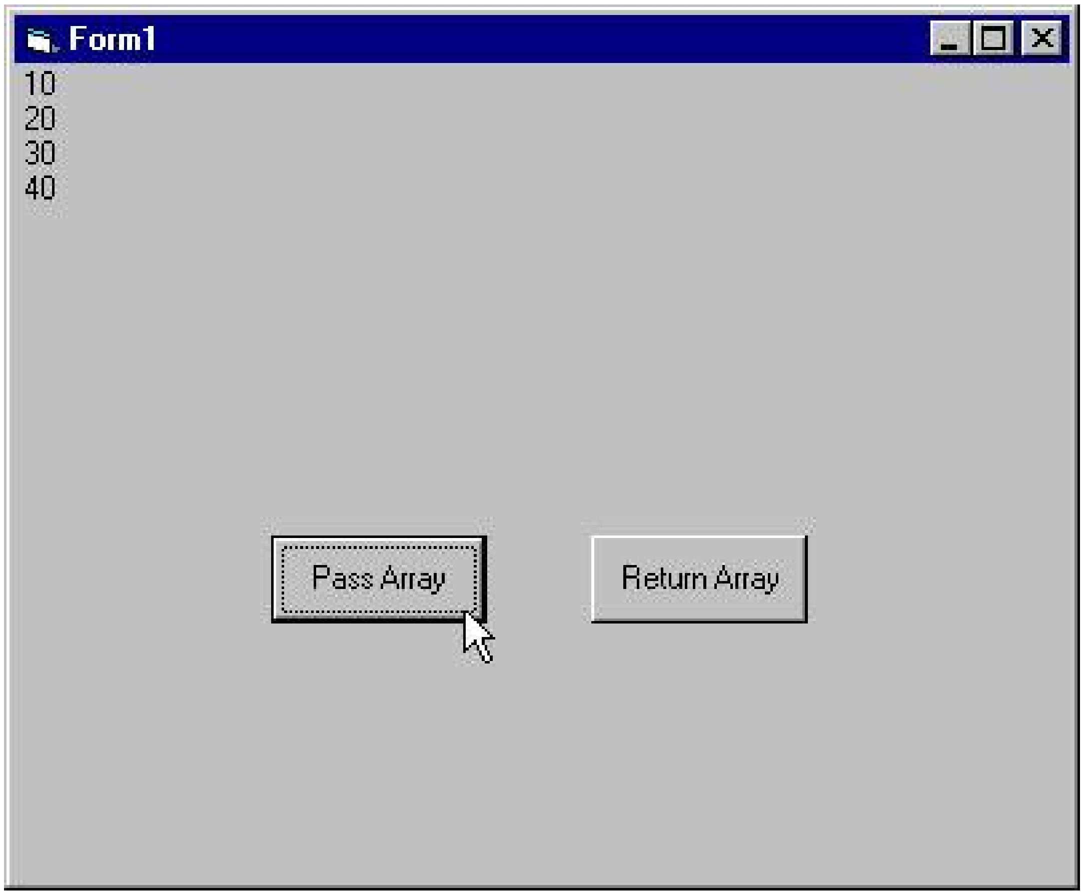
```
Private Sub AcceptArray(intArray() As Integer)
```

After that, it's a matter of using the For…Each statement to 'loop' through each element of the passed array to print the value of the array on the form…

```
Dim obj As Variant

For Each obj In intArray
   Form1.Print obj
Next
```

If we now run the program, you'll see the values of the Array printed on the form…



```
10
20
30
40
```

Before I move on, I'd like to point out by default, the Array has been passed to procedure AcceptArray By Value---that means that only a pointer to the Array was passed to the procedure, not a copy of the actual Array itself. For clarity, you really should specify the procedure header in this way…

```
Private Sub AcceptArray(ByRef intArray() As Integer)
```

For that reason, you need to bear in mind that if you modify any of the Array elements within AcceptArray, the value of the Array back in cmdPassArray will be altered also. For instance, this code in AcceptArray …

```
Private Sub AcceptArray(ByRef intArray() As Integer)

Dim obj As Variant

For Each obj In intArray
   Form1.Print obj
Next

intArray(0) = 11

End Sub
```

would result in the value of x(0) in the cmdPassArray procedure being set to 11 also. Be careful!

By the way, Visual Basic will not allow you to pass the value of an Array to a procedure By Value.)

Write the Function to pass an Array back to the calling procedure

If you understand how to pass an Array to a called procedure, then writing the Function to return an array back to a calling program isn't too difficult. Here's the code for the function…

```
Private Function ReturnArray() As Variant

Dim x(3) As Integer        'Declare a Static Integer Array of 4 elements

x(0) = 1
x(1) = 2
x(2) = 3
x(3) = 4

ReturnArray = x            'Pass the array back as a return value
```

```
End Function
```

As was the case with the procedure to accept an Array as an argument, the header is key here--notice the return value (that's the part that follows the word 'As')---it's a Variant

```
Private Function ReturnArray() As Variant
```

which it is required to be in order to pass an Array back to the procedure that calls the function. The rest of the code is pretty straightforward. This line of code initializes a four element Integer Array called x…

```
Dim x(3) As Integer         'Declare a Static Integer Array of 4 elements
```

The next four lines of code assign values to each of the four elements of the array…

```
x(0) = 1
x(1) = 2
x(2) = 3
x(3) = 4
```

In case you are not familiar with the syntax for returning a value from a function, you assign a value to the name of the function itself---since the name of the is ReturnArray, this line of code assigns the array 'x' to the name of the function 'ReturnArray' which results in the array 'x' being returned to the calling procedure…

```
ReturnArray = x            'Pass the array back as a return value
```

Write the code to receive the Array

Now it's time to write the code to call the function, and receive its return value. We'll place that code in the click event procedure of cmdReturnArray. Here’s the code…

```
Private Sub cmdReturnArray_Click()

Dim retval As Variant
Dim obj

retval = ReturnArray    'Assign the return value to a Variant Variable

For Each obj In retval
   Form1.Print obj
Next
```

```
End Sub
```

This line of code declares a variable to 'hold' the return value from the function ReturnArray that we will call…

```
Dim retval As Variant
```

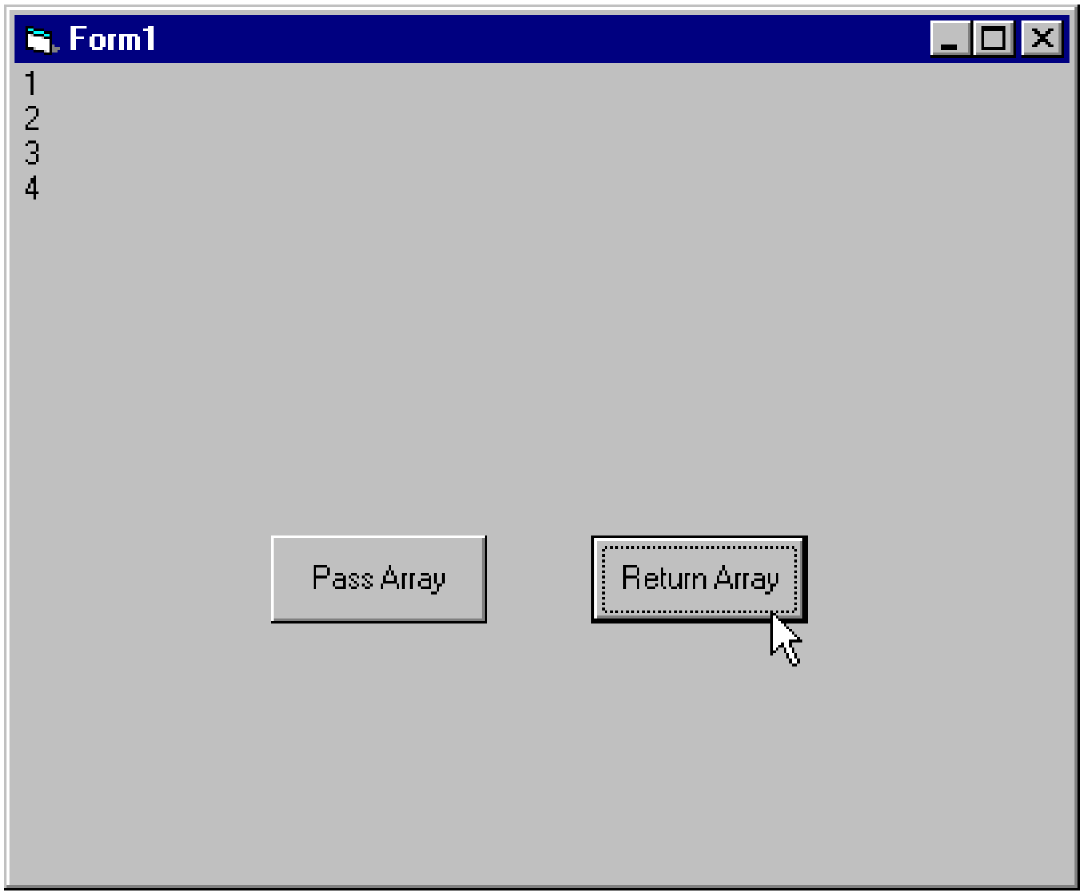
This next line of code is the call to the function. Since a function returns a value, we must either assign that return value to a variable, use it in an expression, or discard it. Here's we'll assign it to the variable retval…

```
retval = ReturnArray    'Assign the return value to a Variant Variable
```

It's interesting that the variable retval now has the characteristics of an Array. If we wanted to, we could determine its Upper and Lower bounds, and work with each of its elements. Of course, we know that there's an easy way to 'loop' through each of the elements of the array and that's to use the For…Each statement…

```
For Each obj In retval
   Form1.Print obj
Next
```

Now if we run this program, and click on the cmdReturnArray command button, the value of the Array in the ReturnArray function will be displayed on the form…



```
1
2
3
4
```

Summary

I hope this article has opened your eyes to the possibility of working with arrays.