Visual Web Development

Terry Marris January 2009

14 Procedures

In chapter 13 we looked at functions. Now we examine procedures, call-by-value and call-by-reference.

14.1 Procedures

Procedures are similar to functions. They usually perform small, well-defined tasks. You write them once but they may be used many times. They may have zero, one or many parameters. They work with argument values passed to them. The essential difference is that a function is used for the value it returns and a procedure is used for its effect. For example, two procedures used in chapter 4 - Number Input Output display and hide an error label and an acknowledge button

```
Sub ShowError()
    lblError.Visible = True
    btnAck.Visible = True
End Sub

Sub HideError()
    lblError.Visible = False
    btnAck.Visible = False
End Sub
```

A procedure begins with Sub and ends with End Sub.

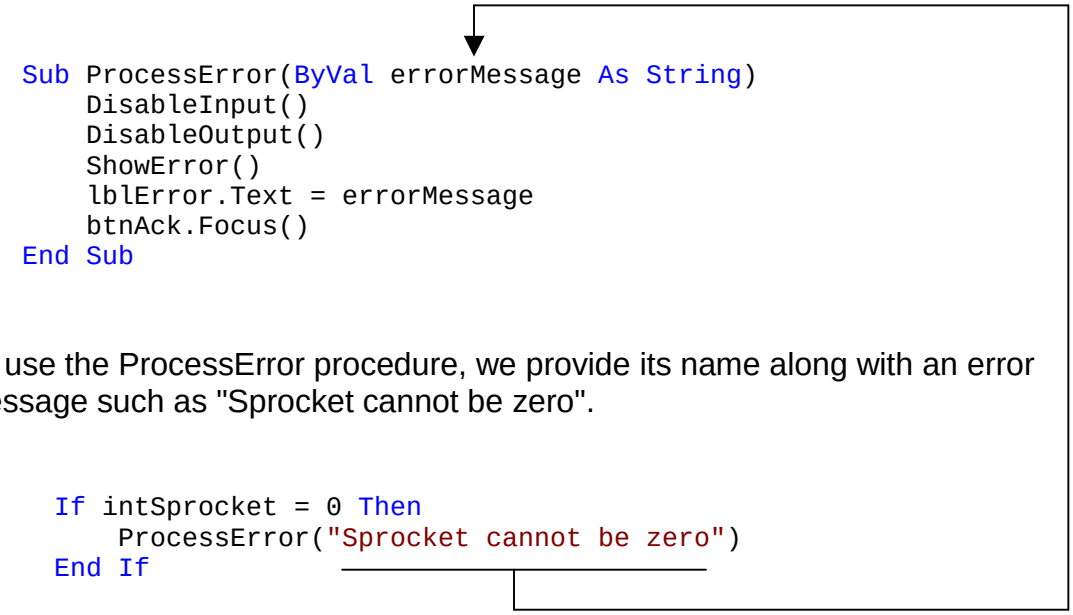
A procedure's name, e.g. ShowError, describes its purpose.

14.2 Call-by-Value

To call a procedure, to say "Hey, procedure, do your job!" we provide the procedure name along with any argument values it requires. Again, from Chapter 4 Number Input Output:

The ProcessError procedure receives an error message to be displayed; this message is identified by ByVal errorMessage As String.

ByVal errorMessage As String is known as a parameter.



```
Sub ProcessError(ByVal errorMessage As String)
    DisableInput()
    DisableOutput()
    ShowError()
    lblError.Text = errorMessage
    btnAck.Focus()
End Sub
```

To use the ProcessError procedure, we provide its name along with an error message such as "Sprocket cannot be zero".

```
If intSprocket = 0 Then
    ProcessError("Sprocket cannot be zero")
End If
```

The error message, "Sprocket cannot be zero" is passed to errorMessage.

"Sprocket cannot be zero" is known as an argument. Arguments are passed to parameters for processing.

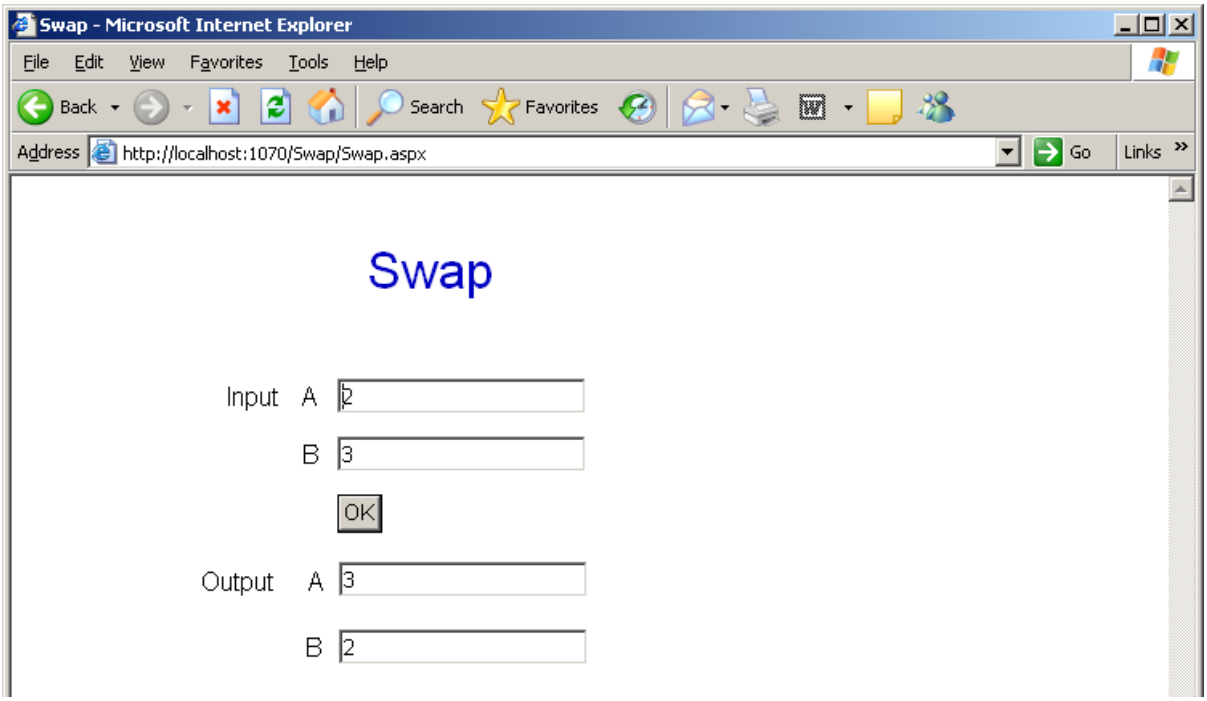
Here, the procedure call is *ProcessError("Sprocket cannot be zero")* and the argument *Sprocket cannot be zero* is passed by value to the procedure's parameter *ByVal errorMessage As String*.

The direction in which the argument is passed is strictly one way, from call to procedure.

Notice that, in the parameter list, the parameter is introduced with ByVal.

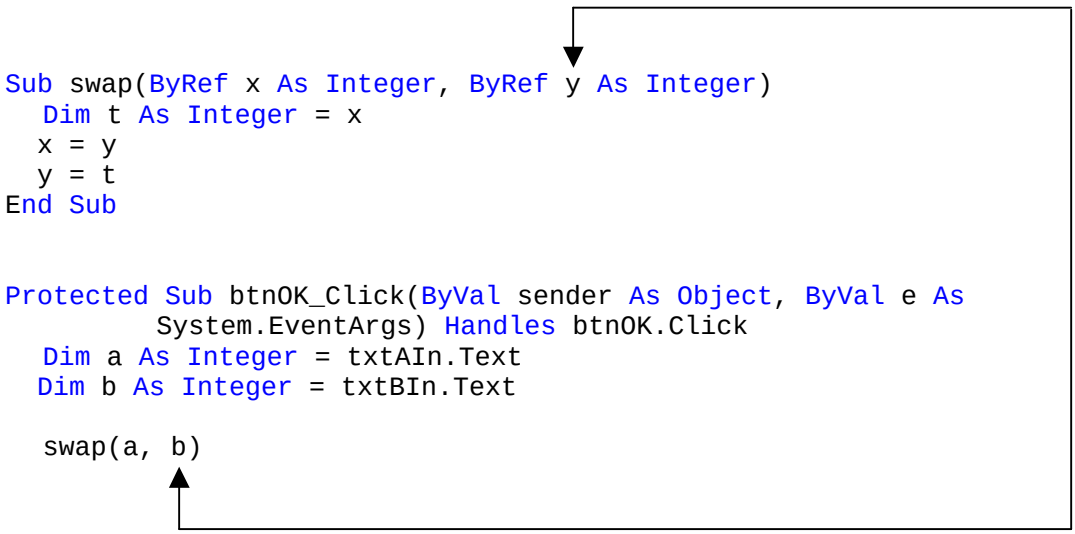
14.3 Call-by-Reference

With call-by-reference values are passed in both directions between arguments and parameters.



The user enters a = 2 and B = 3. A procedure is used to swap the values in these two variables. The result, A = 3 and B = 2, is output.

The swap() procedure, and its call, are shown below.



```
Sub swap(ByRef x As Integer, ByRef y As Integer)
   Dim t As Integer = x
   x = y
   y = t
End Sub


Protected Sub btnOK_Click(ByVal sender As Object, ByVal e As
          System.EventArgs) Handles btnOK.Click
   Dim a As Integer = txtAIn.Text
   Dim b As Integer = txtBIn.Text

   swap(a, b)


   txtAOut.Text = a
   txtBOut.Text = b
End Sub
```

Values for a and b are input. They are then passed to the swap() procedure. The swap procedure receives the values a and b and stores them in x and y respectively. Via an auxiliary variable t, the values of x and y are exchanged. The exchanged values are then sent back to a and b.

The order of items in the argument list (a, b) match the order of items in the parameter list (x, y).

The argument names may be the same as the parameter names without conflict because parameters act like local variables - they apply only to the procedure (or function) that declares them.

With call-by-reference, the arguments must be variables because they have to store the values returned via the parameters.

Notice that, in the parameter list, variables are introduced with ByRef.

14.4 Exercises

1. Try out the program, shown above, that demonstrates call-by-reference.

14.5 Conclusion

We have looked at procedures, call-by-value and call-by reference. Next, we look at array lists.