CHAPTER 2

The VBA Programming Language

To become proficient in developing ArcObjects macros and applications, you need to first become fluent in the VBA programming language. This chapter presents VBA’s basic programming elements, which include variables and control statements. If you have studied other programming languages, the VBA language will be familiar to you. Even if you are not a programmer, VBA is easy to learn.

Referencing Objects with Variables

Your application can guide ArcGIS to execute many types of tasks, such as loading a shape file or preparing a map layout. To accomplish these and other tasks, your VBA program needs information such as the shape file name or page size for layout. These types of information are stored in variables.

A variable is an area in memory reserved for storing a piece of information that can be accessed or changed by a program. Variables can store data types such as text strings and numbers. They can also reference objects. Objects are the things you work with when developing ArcGIS applications. For example, when an application turns on the visibility of a layer, it needs to reference the Layer object.

In ArcGIS, your application does not directly reference an object. Instead, it points to one of the object’s interfaces. An object’s interface organizes related properties and methods of an object. Throughout this book you will see many examples of how variables can point to interfaces to access objects in ArcObjects.
VBA requires that you declare variables in advance of their use. However, you can declare variables explicitly or implicitly. For implicit declaration, just use the variable. When VBA encounters the variable for the first time, it will declare it. In the case of explicit declaration, you declare the variable before using it. To declare a variable, you use one of the following keywords: Dim, Private, Public, or Static. For example, to declare the MyVar variable you use the following statement.

```
Dim MyVar
```

The Dim keyword is the most used method of declaration. It declares a variable to be used inside a procedure. Other keywords for declaration have special purposes. For example, declaring with the Public keyword makes the variable available to more than one procedure. The preceding example of a declaration creates a variable named MyVar as a variant. Variant variables can accept different types of data. You can explicitly indicate the data type a variable can accept, as in the following.

```
Dim MyVar As Long
```

In the preceding example MyVar can only accept numeric data types of long integers. The following are other common data types.

- **Boolean**: Variables of this type can be either True or False.
- **Date**: Date and time values are stored in these types of variables.
- **Double**: Variables of this type store numerical values with decimal point.
- **String**: This data type is for storing text.
- **Object**: Variables of this type point to objects.

You are strongly urged to always declare variables. You can force the explicit declaration by adding the following line to the beginning of your module.

```
Option Explicit
```
You can declare the variable anywhere in your macro, as long as it is before its first use. You can also declare multiple variables on a single Dim statement, separating each variable with a comma.

Using the Global Application Objects

VBA offers two objects that are always available to you while running ArcMap. The two objects can be accessed with the keywords Application and ThisDocument. The Application keyword references the ArcMap program, and ThisDocument points to the current ArcMap document. The following example shows a macro that displays the name of the first layer in a map. The macro produces an error if there are no layers.

Option Explicit
Sub MyMacro()
    Dim pMxDocument As IMxDocument
    Dim pMap As IMap
    Dim pLayer As ILayer
    Set pMxDocument = Application.Document
    Set pMap = pMxDocument.FocusMap
    Set pLayer = pMap.Layer(0)
    MsgBox pLayer.Name
End Sub

In the preceding example there are three variable declarations. Each variable is declared as an ArcObjects interface. In this manner they point to an object through the object’s interface. The Set pMxDocument = Application.Document document statement points the pMxDocument to the IMxDocument interface to access the ArcMap document. The Document property of the Application global object returns the pointer to the interface.

Scopes of the Variable

Scope of a variable refers to the area of VBA where the variable is available. A variable declared in a procedure is not available outside that procedure. Such a variable is often known as a local variable. A vari-
able can be declared outside the procedures of a module so that it is available to all procedures of that module.

You can declare local variables using the Dim or Static keyword inside a procedure. The macro in the last section has an example of declaring three local variables. The following example shows you how to declare variables outside the procedure.

```vba
Option Explicit
Public pMxDocument As IMxDocument
Private pMap As IMap
Sub MyMacro()
    Dim pLayer As ILayer
    Set pMxDocument = Application.Document
    Set pMap = pMxDocument.FocusMap
    Set pLayer = pMap.Layer(0)
    MsgBox pLayer.Name
End Sub
```

In the preceding example the pMxDocument variable is declared with the Public keyword. A variable with the Public scope is available to all procedures in all modules. The Private scope of the pMap variable makes it available to all procedures of the module making the declaration.

**Writing VBA Statements**

You develop VBA macros by writing VBA statements. Among statement types, assignment is the most common. The assignment statement consists of an equals sign with a variable on its left and an expression or object to its right. The result of the expression or the object on the right is then assigned to the variable on the left. The following are two assignment statements.

```vba
SName = “Alaska”
Set pMxDocument = Application.Document
```

You need to use the Set keyword when assigning an object. The following sections discuss other statement types.
Conditional Statement

Controlling the flow of logic in any programming language is a basic operation. Such controls range from executing a set of statements if a certain condition prevails to executing the same set more than once.

An If statement is used to conditionally execute a series of statements. The condition is a Boolean expression resulting in True or False. For instance, you may want to make a layer visible if its name is “STATE”, as shown in the following code segment.

```vba
If pLayer.Name = "STATE" Then
    pLayer.Visible = True
End If
```

The condition is `pLayer.Name = "STATE"`. If the layer name is “STATE”, the condition returns True; otherwise, it returns False. When the condition is True, the statements between the If and End If lines are executed. If statements can also be nested. A nested If structure is a conditional block inside another If block. An If structure starts with the following statement.

```vba
If condition Then
```

The structure can have the following optional statements.

```vba
ElseIf condition Then
or
Else
```

An If structure must end with the following statement.

```vba
End If
```

VBA executes the program lines following the If statement if the condition expression is true. Otherwise, the execution moves to the ElseIf or Else lines if provided.
Loop Structure

Loop structures execute a set of code lines more than once. For example, you may write one set of code to edit a given layer. Then you iterate through all layers with a loop structure, making the same edits. There are two types of loops. Fixed iteration loops repeat for a predefined number of times. Indefinite loops repeat until a condition stops the loop.

Fixed iteration loops are used when the numbers of iterations are known in advance. For example, you may want to loop through all layers of a map and make each visible. The For/Next loop structure is used for fixed iterations. The syntax follows.

For counter = start To end
    ... counter
Next counter

The following example iterates through the layers of the active map.

For LayerCount = 0 To pMxDocument.FocusMap.LayerCount - 1
    ... LayerCount
Next LayerCount

VBA starts the loop by assigning the start value to the counter. It executes the statements inside the loop structure. When VBA reaches the Next statement, it increments the counter and repeats the loop until the counter reaches the end value.

The Do/While loop structure is used for indefinite loops. Such loops are used when the number of iterations is not known in advance. The following code segment shows you an example of looping through features of a layer.

Set pEnumFeature = pMxDocument.FocusMap.FeatureSelection
pEnumFeature.Reset
Set pFeature = pEnumFeature.Next
Do While Not pFeature Is Nothing
    ... pFeature
    Set pFeature = pEnumFeature.Next
Loop
In the preceding example the loop repeats as long as the variable pFeature is not “Nothing.” Nothing is a special value for object variables without an object. When pEnumFeature.Next runs out of features, the value of pFeature is set to Nothing. You must be very careful with these types of loops. Unless you account for a way to end the loop, you could have an infinite loop that would run forever. In the preceding example, the Set pFeature = pEnumFeature.Next statement inside the loop structure will eventually cause the loop to end.

**Continuation Statement**

When you write VBA programs, each statement must be placed on one line. You can break a statement into multiple lines by adding the continuation character to the end of the incomplete lines. The continuation character is the underscore. The following code segment shows how a continuation character can break a statement into two lines.

```vba
Set pFeature = _
pEnumFeature.Next
```

Because of the page width limit in this book, many of the VBA statements are broken into multiple lines using the continuation character.

**Adding Comments**

Documenting your program can save time in the future when you need to maintain the application code. The VBA character for comments is a single quote (’). Whenever this character appears, everything to the end of the line is read as a comment. The exception is when a single quote appears inside a text string.

**Creating Procedures**

The VBA code you write must be placed inside procedures. A macro can consist of one or more procedures. There are two types of procedures: subs and functions. The primary difference between the two is that a function can return a value. A procedure starts with the keyword Sub or Function and ends with the End Sub or End Function statement.
VBA automatically adds the starting and ending statements when you use the Macro dialog or Insert menu option to add new procedures.

You can pass variables between procedures by using arguments. The following code segment shows an example of the Function statement.

```
Function CalculateSquareRoot(NumberArg As Double) As Double
...
End Function
```

VBA has many built-in functions you can use in your macro. The following are common built-in functions that manipulate text strings.

- `InStr (start, string1, string2, compare)`: Returns the position of the first instance of the string2 inside string1.
- `Len (string)`: Returns the length of the string.
- `Mid (string, start, length)`: Returns parts of the string that start at the start position for the given length.

The following code segment displays the text Maryland in a message box.

```
Public Sub MySub()
    Dim strName As String
    strName = "Maryland"
    MsgBox Mid(strName, 5, 4)
End Sub
```