CHAPTER 2

GENERAL CONVERSION GUIDES AND AVENUE WRAPS

This chapter presents a set of general syntax guidelines for converting ArcView® 3.x Avenue code to "VB code", and addresses (a) numeric variables and arithmetic operations, (b) string variables and manipulation thereof, (c) transcendental and other intrinsic functions, (d) query of variables and "If" statements, (e) lists, arrays and collections, (f) data type declaration, definition and conversion, (g) iterative operations such as "Do", "For" and "While" loops, (h) miscellaneous general types of operations, such as getting the current date and time, system alert sound, and summary of declaration of variables, and (i) the use of certain Avenue Wraps of general nature, a list of which is presented below and overleaf.

In the section describing the use of the Avenue Wraps, the user will find the Avenue Wrap's corresponding Avenue request, the description of the input and output (returned) variables, and variable declaration. As a reminder, keep in mind that the variables within the argument list should be declared in the module that first initializes or defines these variables.

It was stated in Chapter 1, and it is worth repeating here, that Avenue requests could be concatenated, by separating each request with a period (.). Avenue Wraps, however, cannot be concatenated. As an example, in Avenue the two requests, GetFTab and FindField could be concatenated in a single statement:

```
aField = aTheme.GetFTab.FindField(aFieldName)
```

However, with Avenue Wraps, each request must appear as a separate statement:

```
Call avGetFTab(pmxDoc,aTheme,aFTab,pFeatClass,pLayer)
aField = aFTab.FindField(aFieldName)
```

The Avenue Wraps of this chapter are listed below in alphabetical order with a short description and the chapter - page number where a full description may be found.

2.1 Numbers, Arithmetic Operations and Error Trapping

2.1.1 Variable Types and Declarations

In Avenue there is no distinction between types of numeric variables. In VB programming, however, there is a distinction between whole numbers (numbers with no fractional part - no decimal point) and floating point numbers (numbers with fractional part - decimal point), and each one of them is further classified based upon the precision of the associated value. Thus, in VB we have variables that may contain:

- **Integer** numbers that are stored as 16-bit (2-byte) numbers ranging in value from -32,768 to 32,767. Such variables should be declared as:

  ```
  Dim theNumber As Integer
  ```

  Note that in some other programming languages, these variables are referred to as Short integers.
- **Long** integer numbers that are stored as signed 32-bit (4-byte) numbers ranging in value from -2,147,483,648 to 2,147,483,647. Such variables should be declared as:

  ```
  Dim theNumber As Long
  ```
- **Single** precision floating point numbers that are stored as IEEE 32-bit (4-byte), and ranging in value from -3.402823E38 to -1.401298E-45 for negative values and from 1.401298E-45 to 3.402823E38 for positive values. Such variables should be declared as:

  ```
  Dim theNumber As Single
  ```
- **Double** precision floating point numbers that are stored as IEEE 64-bit (8-byte), and ranging in value from -1.79769313486231E308 to -4.94065645841247E-324 for negative values and from 4.94065645841247E-324 to 1.79769313486232E308 for positive values.. Such variables should be declared as:

  ```
  Dim theNumber As Double
  ```

When declaring several variables, more than one variable declaration may appear on the same line, for example:

```
Dim theNumb1 As Double, theNumb2 As Double
Dim theNumb3 As Long, theNumb4 As Integer
```

In view of the above, it is important to distinguish between the four number types when programming in VB. The difference between the short and long integers, and single and double precision is the precision of the numbers and memory requirements. As a general rule, the Avenue Wraps use **long** integers for all counters and loop indices, and short **integers** for all others. As for

Note: Declare FTab and VTab field index variables as Long, for example:
Dim col As Long

NUMBERS, ARITHMETIC OPERATIONS AND ERROR TRAPPING

floating numbers, Avenue wraps utilizes **double** precision variables for all variables associated with geometric related operations. All others are dependent upon their specific application and need.

2.1.2 Arithmetic Operations

There is no difference between Avenue and VB with regards to the most common arithmetic notation (symbology of operations) or sequence of operations, and use of parentheses. Hence, any such code may be ported directly from Avenue to VB. However, there are some subtle differences of which the novice VB programmer should be cognizant. These differences pertain to operators, and to transcendental and other intrinsic functions. The available operators in VB compared to those of Avenue are presented below:

Operation	In Avenue	In VB	Comments
Exponentiation	^	^	
Multiplication	*	*	
Division	/	/	Returns a floating number
Division	Not available	\	Returns an integer number
Addition	+	+	
Subtraction	-	-	

2.1.3 Intrinsic Functions

Whereas in Avenue a function was a request invoked by its name being preceded by its argument list and a period (.), in VB the function is invoked by its name followed by its argument list enclosed in parentheses. Furthermore, there is a difference in some of the function names, and whereas a function may be available in Avenue, it is not so in VB, and vise versa. For examples refer to Table 2-1.

TABLE 2-1 INTRINSIC FUNCTIONS

Function	In Avenue	In VB	Function	In Avenue	In VB
Absolute value	A = B.**Abs**	A = **Abs**(B)	Arccosine	ANG = A.**ACos**	Not available
Arcsine	ANG = A.**ASin**	Not available	Arctangent	ANG = A.**ATan**	ANG = **Atn**(ANG)
Cosine	A = ANG.**Cos**	A = **Cos**(ANG)	e^B	Not available	A = **Exp**(B)
Natural Log	A = **Ln**(b)	A = **Log**(B)	Log of a base	A = **Log**(10)	Not available
Modulo	A = B.**Mod**(C)	Not available	Random number	Not available	A = **Rnd**(B)
Signum	Not available	A = **Sgn**(B)	Sine	A = ANG.**Sin**	A = **Sin**(ANG)
Square root	C = (A^2+B^2).**Sqrt**	C = **Sqr**(A^2+B^2)	Tangent	A = ANG.**Tan**	A = **Tan**(ANG)

NOTES: 1. The angles in all trigonometric functions are in radians
2. For the Arcsine and Arctangent functions, see also the icasinan and icatan functions in Chapter 12

TABLE 2-2
ROUNDING AND TRUNCATION OF NUMBERS

Given B =	if A is to be	In Avenue Use	In VB Use	
10.2	10	A = B.**Floor**	B = **Int**(A)	or
		A = B.**Round**	B = **Fix**(A)	or
		A = B.**Truncate**		
10.5	10	A = B.**Floor**	B = **Int**(A)	or
		A = B.**Round**	B = **Fix**(A)	or
		A = B.**Truncate**		
10.8	10	A = B.**Floor**	B = **Int**(A)	or
		A = B.**Truncate**	B = **Fix**(A)	
10.2	11	A = B.**Ceiling**	B = **Int**(A) + 1	or
			B = **Fix**(A) + 1	
10.5	11	A = B.**Ceiling**	B = **Int**(A) + 1	or
			B = **Fix**(A) + 1	
10.8	11	A = B.**Ceiling**	B = **Int**(A) + 1	or
		A = B.**Round**	B = **Fix**(A) + 1	
-10.2	-10	A = **Ceiling**(B)	B = **Fix**(A)	or
		A = B.**Round**		or
		A = B.**Truncate**		
-10.5	-10	A = **Ceiling**(B)	B = **Fix**(A)	or
		A = B.**Round**		or
		A = B.**Truncate**		
-10.8	-10	A = **Ceiling**(B)	B = **Fix**(A)	or
		A = B.**Truncate**		
-10.2	-11	A = B.**Floor**	B = **Int**(A)	
-10.5	-11	A = B.**Floor**	B = **Int**(A)	
-10.8	-11	A = B.**Floor**	B = **Int**(A)	or
		A = B.**Round**		

NUMBERS, ARITHMETIC OPERATIONS AND ERROR TRAPPING

When copying Avenue source code and pasting it on a VB procedure, if there are two plus signs (++) in a statement line, and there are no other conversion errors in that statement, or they have been corrected, one of the two plus signs will disappear. Hence, first take care of the two pluses.

2.1.4 Rounding and Truncation of Numbers

Table 2-2 identifies the use, and hence the comparison between the various functions that are available in Avenue and VB.

2.1.5 String Messages

There are several Avenue Wraps contained in Chapter 6 that enable the programmer to display various types of message boxes, or to display messages in the status bar, all of which require the input of a message box title or heading, and/or an instruction. These may be specified as direct text in the Avenue Wrap subroutine or function, or in the form of a variable. In either case, there are four conversion issues that should be kept in mind. Note that these issues represent generic string manipulation rules and they are not specific to the message boxes.

- **Concatenation**: Two strings may be concatenated to form one by use of the plus (+) sign. This is the same in both Avenue and VB. However, if a space is required between the two strings, in Avenue the programmer could introduce two consecutive plus signs (++) to denote an extra space. This is not possible in VB. If an additional space is desired it must be so introduced between double quotes if it is to separate two numeric variables, or be incorporated at the end of the preceding string, or at the start of the subsequent string.

- **Number Conversion** In Avenue, a number was converted to a string with the request AsString. In VB, such conversion is typically made with the function CStr.

- **New Message Line** In Avenue, a new line was introduced in a message string by introducing the characters +NL+ between two strings. In VB, this is done by introducing the function Chr(13) within two plus signs.

- **Program Continuation Lines** In Avenue, the program was able to break the code and continue it in the next line. This is not so in VB. To continue a statement onto the next line, a space and an underscore (_) must appear at the end of the line to be continued.

As an example of the above consider the Avenue code below and its conversion to VB. Note: (a) the conversion of ++ to + and the introduction of the space character(s) in the hard-coded strings, (b) the substitution of

CStr for .AsString, (c) the substitution of Chr(13) for NL, and (d) the introduction of " _"to continue the statement on another line.

With Avenue

```
MsgBox.Warning("The lengths"++D1.AsString++
"and"++D2.AsString+NL+"are invalid", aTitle)
```

With Avenue Wraps

```
Call avMsgBoxWarning("The lengths " + CStr(D1) + _
" and " + CStr(D2) + Chr(13) + "are invalid", aTitle)
```

2.1.6 Error Trapping

A good feature to take advantage of when developing in either **Visual Basic** or **Visual Basic for Applications** is the ability to trap errors. Error trapping provides the developer a means to avoid application runtime errors, which typically results in the application to cease to operate properly. By avoiding application runtime errors, should an error be encountered, the application can still be used to perform other functions, rather than simply "dying". An example of how error trapping can be implemented is shown below:

```
'
Public Sub ShowErrorTrapping()
'
Dim pMxApp As IMxApplication
Dim pmxDoc As IMxDocument
Dim pActiveView As IActiveView
Dim pMap As IMap
'
'  ---This statement informs the application where to
'  ---branch when an error is detected in the procedure
On Error GoTo Errorhandler
'
'  ---Get the active view
Call avGetActiveDoc(pMxApp, pmxDoc, pActiveView, pMap)
'
'  ---do something else
    .
    .
    .
'
'  ---At this point, our work is done
Exit Sub
'
'  ---Handle any errors detected in the procedure
Errorhandler:
'
'  ---Display detected error number and a description
MsgBox "Error " & Err.Number & " - " & Err.Description & _
       Chr(13) & "Subroutine: ShowErrorTrapping"
'
End Sub
```

MANIPULATION OF STRING VARIABLES

TABLE 2-3
STRING MANIPULATION FUNCTIONS

	In Avenue	In VB
Concatenate two strings	aString1**+** aString2	aString1**+** aString2
Concatenate two strings separated by a space	String1**++** aString2	aString1**+** " " **+** aString2
Capitalize the first letter of a word in a string when words are separated with input character	String1.**BasicProper** (chr)	Not available
Remove from the start and end of a string the specified characters	String1.**BasicTrim** (L, R)	**avBasicTrim**(String1,L,R)
Extract the word at the specified position (0 to N-1), where N is the number of words (space delimited)	String1.**Extract** (Position)	Not available
Returns the position of the first occurrence of String2 in String1	String1.**IndexOf** (String2)	**InStr**(1,String1,String2,1)
Change all string characters to lower case	String1.**LCase**	**LCase**(String1)
Return the specified left most characters	String1.**Left** (nChr)	**Left**(String1, nChr)
Extract a string, starting at a specified offset, the specified number of characters	String1.**Middle** (Off, nChr)	**Mid**(String1, Off, nChr)
Capitalize the first letter of a word in a string when words are separated with blank space	String1.**Proper**	Not available
Place a string within a pair of quotes	String1.**Quote**	Not available
Return the specified left most characters	String1.**Right** (nChr)	**Right** (String1,nChr)
Introduce at the positions shown in the cntrList, the str characters	String1.**Split**(cntrList,str)	Not available
Replace in a string all occurrences of a1 with a2	String1.**Substitute** (a1,a2)	**Replace**(String1, a1, a2)
Replace in a string all occurrences of a1 with a2	String1.**Translate** (a1,a2)	Not available
Remove leading & trailing spaces	String1.**Trim**	**Trim**(String1)
Change all string characters to upper case	String1.**UCase**	**UCase**(String1)
Remove the pair of quotes from a string within a pair of quotes	String1.**Unquote**	Not available

2.2 Manipulation of String Variables

2.2.1 String Manipulation Requests and Functions

There are several text string manipulation requests in Avenue, most all of which have to be converted to VB code. The sole exception is the concatenation of two strings with a plus sign (+) to create a single new string. Three of the string manipulation requests have been addressed in the preceding section. Shown in Table 2-3 are the various Avenue string manipulation requests and their counterparts, if any in VB. In addition to the requests of Table 2-3, the following are considered as requests of rather common use:

- To determine the number of characters in a string:

  The **Avenue** request is:
  ```
  nChars = theString.Count
  ```
  The **VB** function is:
  ```
  nChars = Len(theString)
  ```
  with the variables declared as:
  ```
  Dim nChars As Long
  Dim theString As String
  ```

TABLE 2-4
BOOLEAN QUERYING OF VARIABLES AND IF STATEMENTS

The concatenation of more than one if condition in an "if" statement is the same in both Avenue and VB

In Avenue	In VB
◗ Querying whether a string variable is a number	

`If(theString.IsNumber)Then`
`  ...do something`
`End`
	`If(IsNumeric(theString))Then`
	`  ...do something`
	`End If`

◗ Querying whether a string variable is not a number	

`If(theString.IsNumber.Not)Then`
`  ...do something`
`End`
	`If(Not IsNumeric(theString))Then`
	`  ...do something`
	`End If`

◗ Querying whether a string variable has not been defined	

`If(theString.IsNull)Then`
`  ...do something`
`End`
	`If(IsNull(theString))Then`
	`  ...do something`
	`End If`

◗ Querying whether a string variable has been defined	

`If(theString.IsNull.Not)Then`
`  ...do something`
`End`
	`If(Not IsNull(theString))Then`
	`  ...do something`
	`End if`

◗ Querying whether a string variable has not been defined	
`If(theString = Nil)Then`	`If(IsNull(theString) Then`

MANIPULATION OF STRING VARIABLES

- To check if a string is within another string:
  The **Avenue** request is: `i = String.Contains(aString)`
  The **VB** function is: `i = InStr(1,String,aString,1)`
  with the variables declared as:
  ```
  Dim i As Long
  Dim String, aString As String
  ```
  where: i = 0 : denotes aString was not found
  i > 0 : position of first occurrence of aString in String with values beginning at 1
- To find position of first occurrence of aString in String:
  The **Avenue** request is: `i = String.IndexOf(aString)`
  The **VB** function is: `i = InStr(1,String,aString,1)`

2.2.2 Querying Variables and If Statements

At times it is necessary to query a variable in order to determine the type of the variable, and/or to change a variable from one type to another. For example, one may wish to change a number into a string so that it may be incorporated in a message box, or convert a number, which is in the form of a string into a number, so as to perform arithmetic operations. The latter often occurs when all data (text and numbers) of an application are read into the program as strings. Regarding the conversion of numbers, stored as strings, to variables of number type, attention has to be paid as to the type of number, integer, long, single or double. In Avenue there is no distinction between these types, they are all converted in the same manner. This is not so in VB.

Since in Avenue the "If", "For" and "While" statements all terminate with an "End" statement, it may be a good idea if the first thing to be done when converting to VB is to convert all "End" statements of an "If" statement to "End If".

Generally, querying of variables is done with the "If...Then...Elseif...Else...End" statement in Avenue or with the "If...Then...Elseif...Else...End If" statement in VB, with the Elseif and Else parts being optional in either Avenue or VB. Note that the operation of the If statement is the same in both Avenue and VB. The only difference between them being the ending statement.

In Avenue, an "If" query terminates with the word "End", while in VB it terminates with the words "End If". A compilation error will be displayed if an "If" statement does not terminate with "End If".

When querying a variable, although most of the times a positive (true) response is expected, at times a negative response is desired (false). Also, the variable theString must have been declared as a string or variant.

The most common commands regarding queries of variables and "If" statements are contained in Table 2-4, including positive and negative tests.

2.2.3 Converting and Initializing Variables

At times it is necessary to convert a variable from one type to another for most of the same reasons stated in the preceding section on querying variables.

- Setting a string variable to be undefined. This is usually performed at the beginning of a program to initialize a variable.

In Avenue

```
theString = Nil
```

In VB

```
theString = Null
```

Alternatively to the example above, in both Avenue and VB, a variable may be initialized to some number of string, which is bound to indicate an initialization such as:

A = 999999.0	which in VB will display as 999999#
B = 0.0	which in VB will display as 0#
I = -1	which in VB will also display as -1
theString = ""	which in VB will also display as ""

- Converting to a number a string which has been proven to be a number. Note the distinction in VB between the four aforementioned types of numbers.

In Avenue

```
theNumber = theString.AsNumber
```

In VB

`theNumber = CInt(theString)` to change the string into an integer in the range of -32,768 to 32,767; fractions are rounded to the nearest integer.

`theNumber = CLng(theString)` to change the string into a long integer in the range of -2,147,483,648 to 2,147,483,647; fractions are rounded to the nearest integer.

`theNumber = CSng(theString)` to change the string into a single precision floating number in the range of -3.402823E38 to -1.401298E-45 for negative values; 1.401298E-45 to 3.402823E38 for positive values.

`theNumber = CDBL(theString)` to change the string into a double precision floating number in the range of -1.79769313486231E308 to -4.94065645841247E-324 for negative values; 4.94065645841247E-324 to 1.79769313486232E308 for positive values,

`theVariant = CVar(aVariable)` to change a string, integer, long, single or double number into a variant

LISTS, ARRAYS AND COLLECTIONS

TABLE 2-5
LISTS AND COLLECTIONS

In Avenue	In VB	
To **create** or initialize a collection		
`aList = List.Make`	`Call CreateList(aList)`	
To **append** an item to a collection (see below for inserting an item)		
`aList.Add(aVal)`	`aList.Add(aVal)`	(#)
	`aList.Add aObj`	(##)
To **get** (extract) an item from a collection		
`aVal = aList.Get(j)`	`aVal = aList.Item(j)`	
To **remove** an item from a collection		
`aList.Remove(j)`	`aList.Remove(j)`	
To **insert** an item at the beginning of a collection		
`aList.Insert(aVal)`	`aList.Add (aVal), before:=1`	(#)
	`aList.Add aObj, before:=1`	(##)
To **shuffle** an item within a collection - j in this case denotes the position (subscript) after which the item aVal is to be inserted		
`aList.Insert(aVal)`	`aList.Add(aVal), before:=j`	(#)
`aList.Shuffle(aVal,j)`	`aList.Add aVal, before:=j`	(##)
To **replace** an item within a collection		
`aList.Set(j,aVal)`	`aList.Add(aVal), after:=j`	(#)
	`aList.Remove(j)`	
	`aList.Add aVal, after:=j`	(##)
	`aList.Remove(j)`	
To **count** the number of items in a collection		
`nItems = aList.Count`	`nItems = aList.Count`	
To **clear** a collection		
`aList.Empty`	`Call CreateList(aList)`	
To **clone** a collection		
`aList1 = aList2.Clone`	`Set aList1 = aList2`	

Notes:

- For the Avenue Wrap CreateList refer to the end of this chapter and Appendix D
- (#) Use this for non-objects (strings, numbers, etc.)
- (##) Use this for objects (Collections and ArcObjects)
- j is measured from base 0 — j is measured from base 1, so that, all Avenue index values used in collections will need to be incremented by one

2.3 Lists, Arrays and Collections

2.3.1 Definitions

In Avenue, a grouping of items such as variables, themes, tables, views and others could constitute a list. In VB, lists are referred to as collections. In addition to collections, the user is able to utilize arrays, much the same way as one would in Fortran or C. In VB, arrays are declared based upon the type of data they are to contain, while collections are declared as themselves. Thus, the corresponding Dim statements for the following samples would be:

`Dim iAry(5) As Integer`	iAry is a one dimensional array.
`Dim jAry(2,30) As Long`	jAry is a two dimensional array.
`Dim kAry(2,4,6) As Single`	kAry is a three dimensional array.
`Dim mAry() As Double`	mAry is a dynamic array.
`Dim aCol As Collection`	aCol is declared to be a non-initialized collection (Nothing).
`Dim aCol As New Collection`	aCol is declared to be, and initiated as a zero-length or empty collection.

Note that collections are one dimensional only. Reference is made to the Avenue Wrap CreateList of this chapter which may be used to initialize and empty a collection. This Avenue Wrap is not applicable to arrays.

2.3.2 Working with Arrays

Working with arrays in VB is quite similar to working with arrays in Fortran or C. One may assign values to array cells, or extract values from such cells by referring to the array and the desired cell index. The programmer should note that the default base index of an array in VB is zero (0) and not one (1). However, it can be changed to one (1), if so desired, by introducing in the declaration section of the module the statement:

```
Option base 1
```

Another way to control the issue of array subscripts is to specify the low and upper bounds of the subscripts. For example, the declaration:

```
Dim iArray(15) As Integer
```

denotes a one dimensional array with 15 cells between 0 and 14, or between 1 and 15 if Option base 1 had been specified, while the declaration:

```
Dim iArray(3 To 7) As Integer
```

denotes a one dimensional array with 5 cells between 3 and 7. In the latter case there are no 0, 1 and 2 cells.

Note that you cannot use the "Add" and "Count" commands with an array. Some of the operations associated with arrays include:

- To extract a value from a cell use an equation:

```
aVal = theArray(j)
```

where j must have been previously declared to be an integer or a long variable within the range limits of the Dim statement that declared theArray.

- To assign a value to a cell use an equation:

```
theArray(j) = aVal
```

where j must have been previously declared to be an integer or a long variable within the range limits of the Dim statement that declared theArray.

2.3.3 Working with Collections

Working with collections is not quite the same as working with arrays. The differences are:

- A collection may contain variables of different type.

- The base index of a collection is one (1) and not zero (0), the opposite of arrays.

The most common commands regarding Avenue lists and VB collections, and the differences between them are contained in Table 2-5.

2.3.4 Sorting of Collections

In Avenue the request "Sort" is used to sort a list. In VB, the Avenue Wrap "SortTwoLists" may be used to sort one or two collections, but not arrays. When sorting two collections, SortTwoLists treats the two collections as a two dimensional array to be sorted under one sort key, that being the first collection. The use of this Avenue Wrap is presented later on in this chapter, while the source code of "SortTwoLists" may be found in Appendix D.

2.3.5 Copying of Collections

There are two Avenue Wraps that do not have Avenue counterparts, and which allow the programmer to copy one collection into another. The CopyList enables the programmer to copy a non-object collection into another non-object collection, while CopyList2 enables the programmer to copy an object collection into another object collection. The use of these two Avenue Wraps is presented later on in this chapter, while their source listing may be found in Appendix D.

ITERATIVE OPERATIONS

2.4 Iterative Operations

2.4.1 The Iterative Statements

In Avenue there are only two iterative operation statements, the "For Each ... End" statement and the "While ... End" statement. In VB there are three, the "Do", the "For" and the "While ... Wend", of which:

- the "Do" has four variations, the "Do While ... Loop", the "Do Until ... Loop", the "Do ... Loop While", and the "Do ... Loop Until", and

- the "For" has two variations, the "For ... Next", and the "For Each ... Next".

2.4.2 Converting the Avenue "For Each ... End" Statement

In Avenue, this statement is comprised of the following lines:

```
For Each Rec in theList
   ... do something with Rec
End
```

where Rec is an object in theList and theList is a list.

To convert this statement, the programmer must be cognizant of what theList is comprised. If theList contains:

- Objects, then the programmer should use the following

```
For Each Rec in theList
   ... do something with Rec
Next Rec
```

  where Rec is an object in theList and declared accordingly, and theList is a collection of objects. In this example, theList contains objects of the same type.

- Variables, then the programmer should use the following

```
For iRec = 1 To theList.Count
   Rec = theList.Item(iRec)
   ... do something with Rec
Next iRec
```

  where iRec should be declared as an integer or long, theList may be a collection or array, and Rec as a variant. Note that in the above example, iRec is both a counter and an index to theList.

Alternatively, the user may elect to compute the index to theList for which something is to be done.

```
K = 5
For I = iLow To iHigh
    K = K + 1
   ... do something with theList(K)
Next I
```

In using the above variables I, K, iLow and iHigh, the programmer should keep in mind that in a collection the base reference to an Avenue list is zero (0), while the base reference to a VB collection is one (1).

2.4.3 Converting the Avenue "While ... Wend" Statement

In Avenue, this statement is comprised of the following lines:

```
While Expression
   ... do something as long the Expression is true
End
```

In VB, the programmer may use any one of the four "Do" iterative statement variations depending on how the programmer wishes to set the conditional expression to be evaluated. For example, consider the following:

```
DoOver = True
Do While DoOver
   ... do something
   If (something) Then
      ... do some other things
   Else
      DoOver = False
   End If
Loop
```

Regarding the four variations of the "Do" statement, the programmer should note that:

- The "While" condition performs the operations between "Do" and "Loop" for as long as the conditional expression (DoOver in the above example) is true, while the "Until" condition performs said operations until said condition is met.

ITERATIVE OPERATIONS

- By placing the conditional test at the top with the "Do" loop, the subsequent statements are executed up to the "Loop" statement only if the condition is true. Thus the possibility exists that said subsequent statements may never be executed. By placing the conditional test at the bottom with the "Loop" said subsequent statements will be executed at least once.

In addition to one of the above four variations of the "Do" statement, the programmer may elect to use the "While ... Wend" statement, which represents a more direct one to one conversion between Avenue and VB, and has only one difference, the substitution of "Wend" for "End" in the ending statement of the iterative operation. While this may at first seem to be preferential, it does not provide as good of a structured approach as the "Do" statement, particularly if an early exit of the iterative process is desired.

2.4.4 Early Exit of an Iterative Statement

At times it becomes desirable to exit an iterative process earlier than provided by the conditions of the iterative processes. In Avenue, the programmer could exit an iterative process earlier than dictated by the conditions of a "For" or "While" statement by introducing the "Break" statement. In VB, the user has the following options:

- In any of the iterative processes, the user may terminate a subroutine or function without completing the entire iteration process by introducing the "Exit Sub", or "Exit Function" statement respectively.

- In the two variations of the "For" statement, the programmer may terminate the iterative process, and proceed to continue with the next statement after the "Next" statement by introducing the "Exit For" statement line.

- In the four variations of the "Do" statement, the programmer may terminate the iterative process, and proceed to continue with the next statement after the "Loop" statement by introducing the "Exit Do" statement line.

- The only way to prematurely exit a "While ... Wend" iterative process is with a "GoTo" statement (see the following section about *Advancing to the Next Iteration*).

2.4.5 Advancing to the Next Iteration

At times it its desirable to skip to the next iteration from somewhere within the code of the iteration process. In Avenue, this can be accomplished with the "Continue" statement. Such a statement and function is not available in VB. One way to get around this problem is to restructure the code of the iteration routine perhaps with properly constructed "If" statements. Another way is with the use of the "GoTo" statement. As an example consider the following:

```
DoOver = True
K = 1
While DoOver
   Do something that involves modification of K
   If (K > 0) Then
      ... do something else with K
   Elseif (K = 0) Then
      Exit Sub     ' If K=0 exit the subroutine
   Elseif (K < 0) Then
      GoTo Line 1  ' If K<0 skip remaining steps,
   End If          ' but do not exit subroutine
      ... continue doing something
Line 1             ' Come here when K<0
   Wend
```

Note that a "GoTo" statement can be used in other instances, and more than once, in which case, different line numbers or text should be used. It is recommended that the use of this statement be a last resort case, because it does not create a well structured code, and can become confusing during the debugging stage.

MISCELLANEOUS OPERATIONS

2.5 Miscellaneous Operations

2.5.1 Current Time and Date

At times it is desirable to retrieve from the computing system the time and date that a program is being executed. This may be done as follows:

In Avenue

```
D  = Date.Now
d1 = D.SetFormat("d MMMM yyyy hhh m s").AsString
d2 = D.SetFormat("d MMMM yyyy").AsString
d3 = D.SetFormat("hhh m s").AsString
```

The second line (d1) above will get the date and time, the third line (d2) will get the date only, and fourth line (d3) will get the time only. The string appearing in the SetFormat statement may vary from what is shown above to meet a specific user format for the date and/or time.

In VB

```
Dim aDate1, aDate2, aDate3
aDate1 = Date
aDate2 = Now
aDate3 = FormatDateTime(aDate1,K)
```

`MsgBox aDate1`	prints	5/14/2002	
`MsgBox aDate2`	prints	5/14/2002 9:28:11 AM	
`MsgBox aDate3`	prints	5/14/2002 9:28:11 AM	if K = 0
		Tuesday, May 14, 2002	if K = 1
		5/14/2002	if K = 2
		9:28:11 AM	if K = 3
		9:28	if K = 4

2.5.2 System Beep

Usually when an error occurs during the execution of a program, or if an erroneous data is key entered in a form, it is a good idea for the program to issue a warning sound or beep. This is done as follows:

In Avenue

```
System.Beep
```

In VB

```
Beep
```

2.5.3 Variable Declarations

Although some of the following may have been addressed elsewhere in this book, it is felt that it is worth repeating. Before proceeding any further, it is necessary to distinguish between the

TABLE 2-6
VB DECLARATION OF COMMON OBJECTS AND VARIABLES

Object/variable	Declaration Statement
Document	Dim pDoc As IMxDocument
Map	Dim pMap As IMap
Layer (theme)	Dim pLayer As ILayer
Table (FTab)	Dim aFTab As IFields
Collection	Dim aList As New Collection
Selection	Dim aSel As ISelectionSet
Point	Dim pPoint As IPoint
Lines	Dim pLine As IPolyline
Polygon	Dim pPolygon As IPolygon
Variant	Dim aNumber As Variant
Integer	Dim aNumber As Integer
Single precision	Dim aNumber As Single
Double precision	Dim aNumber As Double
String	Dim aString As String

words "declare" and "define", and derivatives thereof. Each variable and object used in a program must first be **declared** as to its type (variant, integer, string, etc.). This is done with the Dim statement. Table 2-6 contains a summary of how various type of variables and objects should be declared. The list of declarations in this table is not by any means the complete list of declarations. Only the ones that are considered as the most common are presented therein.

In Avenue, all variables used in a script have to be **defined** or **initialized** prior to their use. That is, one could not say

```
A = B + 5.9
```

Unless B had been previously been assigned a value. Likewise, the statement below would be invalid

```
theFTab = theTheme.GetFTab
```

unless theTheme had previously been defined as a theme. However, there are variables and objects that for some reason need to be defined as null objects or empty variables. This implies that it will be desirable to also know

Note: Declare FTab and VTab tables as IFields, for example:
Dim theFTab As IFields
Dim theVTab As IFields

TABLE 2-7
LIST OF NULL DEFINITION IN VB

Object or variable	To define an object or variable as null or empty	To query whether an object or variable is null or empty
All objects	Set anObject = Nothing	If (anObject Is Nothing) Then
Variants	aVariant = Null	If (IsNull(aVariant)) Then
Strings	aString = Null	If (IsNull(aString)) Then
All numbers	not applicable	If (aNumber = 0) Then when a number has not been initialized

MISCELLANEOUS OPERATIONS

whether an object or a variable has been defined or not. In Avenue, the key word for such querying is "Nil". In VB, the corresponding word is "Null" for non-objects, and "Nothing" for objects, see Table 2-7 for a summary.

In VB we have the word "Empty" and the function "IsEmpty", which are associated with variables only, and not with objects. Thus, if we wish, for some reason or another to not define or initialize a variable, we can write:

```
B = Empty
... do something and then later on ask
If (Not IsEmpty(B)) Then
   A = B + 5.9
End If
```

This has no counterpart in Avenue.

2.5.4 Script Execution

In Avenue, the programmer was able to execute another script by using the av.Run statement. With VB code, the programmer executes another script by calling a subroutine or a function, depending upon how the other script has been implemented. Functions return one and only one value, while subroutines can return many, or none, values. For example:

In Avenue

```
myList = List.Make
myList.Add( TRUE )
returnValue = av.run( "script2", myList )
```

In VB with script2 implemented as a Subroutine

```
Dim myList As New Collection
Dim returnValue As Variant
Call CreateList(myList)
myList.Add (TRUE)
Call script2 (myList, returnValue)
```

In VB with script2 implemented as a Function

```
Dim myList As New Collection
Dim returnValue As Variant
Call CreateList(myList)
myList.Add (TRUE)
returnValue = script2 (myList)
```

Note, all references to the SELF statement must be replaced by putting the variables created with the SELF statement in the argument list of the subroutine or function.

2.6 General Avenue Wraps

GENERAL AVENUE WRAPS

2.6.1 Function avBasicTrim

This function enables the programmer to remove from a given string the specified leading and/or trailing characters.

The corresponding Avenue request is:

```
newString = theString.BasicTrim(LeadChar, TrailChar)
```

The call to this Avenue Wrap is:

```
newString = avBasicTrim(theString, LeadChar, TrailChar)
```

avBasicTrim

GIVEN: theString = the given string to be trimmed
LeadChar = the characters to be removed at the start of the given string
TrailChar = the characters to be removed at the end of the given string

RETURN: newString = the resultant string

The given and returned variables should be declared where first called as:

```
Dim theString As String, LeadChar As String, TrailChar As String
Dim newString As String
```

2.6.2 Function avClone

This function enables the programmer to make a new object by copying an existing object.

The corresponding Avenue request is:

```
theNewObject = theObject.Clone
```

The call to this Avenue Wrap is:

```
Set theNewObject = avClone(theObject)
```

avClone

GIVEN: theObject = object which is to be copied

RETURN: theNewObject = copy of the object

The given and returned variables should be declared where first called as:

```
Dim theObject As IUnknown
Dim theNewObject As IClone
```

GENERAL AVENUE WRAPS

avExecute

The Public variable, ugWinStyle or the Avenue Wraps Property WinStyle can be used to control the window style that is used by the avExecute subroutine. The default window style value is 1 and denotes that the Window is to have focus and be restored to its original size and position. Possible values include: 0, 1, 2, 3, 4 and 6.

2.6.3 Subroutine avExecute

This subroutine enables the programmer to execute a system level command. In using this subroutine, note that once the command has been issued, the statements that follow the call to avExecute will be immediately executed, there is no waiting for the system command to finish its processing. In order to pause ArcMap until said command is completed, one possibility is to perform a loop checking for the existence of a file, which could be created when said command has finished processing (see example below and avExecute2).

The corresponding Avenue request is:

System.Execute (aCommand)

The call to this Avenue Wrap is:

Call **avExecute**(aCommand)

GIVEN: aCommand = the command to be executed

RETURN: nothing

The given and returned variables should be declared where first called as:

Dim aCommand As String

The code below is an example of how to invoke a program from within a VBA module. The program "Adjust" reads a file called "inFile" and will create a "dummy" file called "outFile" when its processing is complete.

```
Public Sub Test
Dim aCmnd, inFile, outFile As String
   .....
   Perform some operations to create "inFile"
   .....
   aCmnd = "c:\Dir1\SubDir3\Adjust.exe " + inFile
   Call avExecute(aCmnd)
   Do While (True)
      If (avFileExists(outFile)) Then
         Exit Do
      End If
   Loop
   .....
End Sub
```

GENERAL AVENUE WRAPS

2.6.4 Subroutine avExecute2

This subroutine is similar to avExecute with the exception that the programmer supplies the name of a file which avExecute2 waits for to exist prior to terminating. Any statements following the call to avExecute2 will not be executed until the specified file exists. This subroutine provides better performance than avExecute on computers operating Windows 2000. Note that the ArcMap document file should have a name other the default of "Untitled..." assigned to it, if not, this subroutine will not function properly.

The corresponding Avenue request is:

```
System.Execute (aCommand)
```

The call to this Avenue Wrap is:

```
Call avExecute2(aCommand, aFileName)
```

avExecute2

GIVEN: aCommand = the command to be executed
aFileName = name of the file whose existance signals the end of processing

RETURN: nothing

The given and returned variables should be declared where first called as:

```
Dim aCommand As String, aFileName As String
```

2.6.5 Function avGetEnvVar

This function enables the programmer to get the full path for an environment variable. Below are examples of what is returned for what is given:

Given	Return
ARCHOME	C:\ARCGIS\ARCEXE81
TMP	C:\WINDOWS\TEMP
ABC	yields an empty string (""), assuming the ABC does not exist

The corresponding Avenue request is:

```
theEnvVar = System.GetEnvVar (aPath)
```

The call to this Avenue Wrap is:

```
theEnvVar = avGetEnvVar(aPath)
```

avGetEnvVar

GIVEN: aPath = name of the environment variable to be processed

RETURN: theEnvVar = full path name associated with the variable

GENERAL AVENUE WRAPS

The given and returned variables should be declared where first called as:
Dim aPath, theEnvVar As String

2.6.6 Subroutine avRemoveDupStrings

avRemoveDupStrings

This function enables the programmer to remove duplicate strings or numbers from a list (collection). In addition, the programmer can specify whether the strings in the list are to be treated as case sensitive or case insensitive. That is, are upper and lower case characters to be treated the same. If they are *not* to be treated the same, this is referred to as being *case sensitive*.

The corresponding Avenue request is:
aList.RemoveDuplicates

The call to this Avenue Wrap is:
Call **avRemoveDupStrings**(aList, caseFlag)

GIVEN: aList = list of strings or numbers to be modified
caseFlag = flag denoting the case sensitivity of the list
True = case sensitive, False = insensitive

RETURN: nothing

The given and returned variables should be declared where first called as:
Dim aList As New Collection, caseFlag As Boolean

2.6.7 Subroutine CopyList

CopyList

This subroutine enables the programmer to copy a collection into another collection, and then initialize (clear) the original collection (the collection that was copied). Note that this subroutine operates only on non-object collections, collections containing variants, numbers and strings. To copy an object collection into another object collection the programmer must use the CopyList2 Avenue Wrap, which is presented later on.

The corresponding Avenue request is:
There is no corresponding Avenue request.

The call to this Avenue Wrap is:
Call **CopyList**(origList, newList)

GIVEN: origList = list to be copied and then cleared

RETURN: newList = copy of the original list

The given and returned variables should be declared where first called as:
Dim origList As New Collection, newList As New Collection

GENERAL AVENUE WRAPS

2.6.8 Subroutine CopyList2

This subroutine enables the programmer to copy a collection into another collection, and then initialize or clear the original collection (the one that was copied). Note that these collections contain objects, not variables such as strings, numbers and so forth. To copy a non-object collection into another non-object collection use the CopyList Avenue Wrap described above.

The corresponding Avenue request is:
There is no corresponding Avenue request.

The call to this Avenue Wrap is:
Call **CopyList2**(origList, newList)

CopyList2

GIVEN: origList = Object list to be copied and then cleared

RETURN: newList = Object copy of the original list

The given and returned variables should be declared where first called as:
Dim origList As New Collection, newList As New Collection

2.6.9 Subroutine CopyList3

This subroutine enables the programmer to copy a collection into another collection, leaving the original collection (the collection that was copied) unaltered. Note that this subroutine will process non-objects and objects, offering a more generic version of CopyList and CopyList2 with the exception that the original collection is not cleared.

The corresponding Avenue request is:
There is no corresponding Avenue request.

The call to this Avenue Wrap is:
Call **CopyList3**(origList, newList)

CopyList3

GIVEN: origList = list to be copied

RETURN: newList = copy of the original list

The given and returned variables should be declared where first called as:
Dim origList As New Collection, newList As New Collection

GENERAL AVENUE WRAPS

CreateList

2.6.10 Subroutine CreateList

This subroutine enables the programmer to create a collection which contains either objects or variables, and initialize it to be an empty collection.

The corresponding Avenue request is:

```
newList = List.Make
```

The call to this Avenue Wrap is:

```
Call CreateList(newList)
```

GIVEN: nothing

RETURN: newList = the new empty collection

The given and returned variables should be declared where first called as:

```
Dim newList As New Collection
```

Dformat

2.6.11 Function Dformat

This function creates a string representation of a number according to a Fortran Fa.b format. In using this function, note the following:

- The given number (theNumber) can be declared as *Decimal*, *Double*, *Integer*, *Long*, *Short* or *Single*.
- If the number of digits to the right of the decimal point (DigitsRight) is zero (0), the output string will contain only the whole number part of a rounded floating point number (no decimal point will be included).
- If the number of digits to the right of the decimal point (DigitsRight) is one (1) or higher, the output string will contain a decimal point and that many zeros to the right of the decimal point.
- If the given number (theNumber) exceeds the number of characters, as specified by TotalDigits, the output string (theString) is expanded to accommodate the number, automatically.

The corresponding Avenue request is:

There is no corresponding Avenue request.

The call to this Avenue Wrap is:

```
theString = Dformat(theNumber, TotalDigits, DigitsRight)
```

GIVEN: theNumber = the number to be formatted
TotalDigits = the total number of characters, including leading spaces, decimal point and decimal digits, in the string to be passed back

GENERAL AVENUE WRAPS

DigitsRight = digits to the right of the decimal point

RETURN: theString = string representing the formatted number

The given and returned variables should be declared where first called as:

```
Dim theNumber As XX    (see commentary above regarding declaration)
Dim TotalDigits As Integer, DigitsRight As Integer
Dim theString As String
```

2.6.12 Subroutine SortTwoArrays

This subroutine enables the programmer to sort one or two different one-dimensional arrays. When sorting two arrays, the sorting of the second array corresponds to the sort of the first array. This subroutine is similar to SortTwoLists with the exception that this procedure sorts arrays not lists (collections). As such, this procedure operates much faster than SortTwoLists when dealing with a large number of elements.

In using this subroutine, note the following:

- The order of the arrays passed in are changed by this procedure.
- If only one array is to be sorted, the second array, Array2 can be passed in as Null.
- If Null is specified for aMssg, a progress bar will not be displayed during the sorting process.
- The arrays passed in can contain string and numeric data. They can not contain any objects.

The corresponding Avenue request is:

There is no corresponding Avenue request.

The call to this Avenue Wrap is:

```
Call SortTwoArrays(Array1, Array2, aMssg, anOrder)
```

SortTwoArrays

GIVEN:
- Array1 = first array of items to be sorted
- Array2 = second array of items to be sorted, if only one array to be sorted specify as NULL
- aMssg = progress bar message, if no message is desired specify as NULL
- anOrder = the sort order as a Boolean:
  True = ascending, and
  False = Descending

GENERAL AVENUE WRAPS

RETURN: nothing

The given and returned variables should be declared where first called as:
Dim Array1(), Array2(), aMssg As Variant, anOrder As Boolean

2.6.13 Subroutine SortTwoLists

This subroutine enables the programmer to sort one or two different lists (collections, not arrays). When sorting two lists, the sorting of the second list corresponds to the sort of the first list. That is, SortTwoLists treats the two lists (List1 and List2) as a two dimensional array (List1 constituting the first column and List2 the second) to be sorted under one sort key, that of List1.

In using this subroutine, note the following:

- The order of the lists passed in are changed by this script to reflect the effects of the sort (List1 and List2 are modified by this subroutine).
- If only one list is to be sorted, the second collection List2 can be an empty list, or passed in as Nothing.
- If Null is specified for aMssg, a progress bar will not be displayed during the sorting process.

There is no corresponding Avenue request for sorting two lists. The corresponding Avenue request to sort one list is:

aList.Sort(anOrder)

SortTwoLists

The call to this Avenue Wrap is:

Call **SortTwoLists**(List1, List2, aMssg, anOrder)

GIVEN:	List1	= first list of items to be sorted
	List2	= second list of items to be sorted, if only one list to be sorted specify as NOTHING
	aMssg	= progress bar message, if no message is desired specify as NULL
	anOrder	= the sort order as a Boolean: True = ascending, and False = Descending

RETURN: nothing

The given and returned variables should be declared where first called as:
Dim List1 As New Collection, List2 As New Collection
Dim aMssg As Variant, anOrder As Boolean