Calling shapelib from VB6

Invoking shapelib functions from Visual Basic 6 is straightforward; however, to be successful you must use indirection techniques (pointers) that may be unfamiliar to many VB programmers. Shapelib is a C API and, as such, makes liberal use of pointers in its data structures, function arguments and function return values. In those instances where pointers refer to native VB data types (Bytes, Longs, Doubles, among others), passing a pointer in C is equivalent in VB to passing ByRef, and no further work on your part is required. Passing a char* pointer in C is equivalent in VB to passing a string ByVal. In VB, a string passed ByVal is equivalent to passing a pointer to the first character of the string. (A ByRef-passed string in VB is actually a pointer to a pointer).

Shapelib contains a SHPObject structure (Type, in VB) that must be handled carefully. The C definition of SHPObject is:

```
typedef struct
{
    int     nSHPType;
    int     nShapeId;   /* -1 is unknown/unassigned */
    int     nParts;
    int     *panPartStart;
    int     *panPartType;
    int     nVertices;
    double  *padfX;
    double  *padfY;
    double  *padfZ;
    double  *padfM;
    double  dfXMin;
    double  dfYMin;
    double  dfZMin;
    double  dfMMin;
    double  dfXMax;
    double  dfYMax;
    double  dfZMax;
    double  dfMMax;
} SHPObject;
```

*padfX, *padfY, *padfZ, and *padfM are pointers to arrays of doubles of a length defined by nVertices. The actual arrays are not stored in this struct, only pointers to some other locations in memory where the actual arrays live. To define this struct in VB we use:

```
Public Type SHPObject
    nSHPType As Long
    nShapeId  As Long  '/* -1 is unknown/unassigned */
    nParts As Long
    panPartStart As Long
    panPartType As Long
    nVertices As Long
    padfX As Long  ' pointer to array of doubles
    padfY As Long
    padfZ As Long
    padfM As Long
    dfXMin As Double
    dfYMin As Double
    dfZMin As Double
    dfMMin As Double
```

```
    dfXMax As Double
    dfYMax As Double
    dfZMax As Double
    dfMMax As Double
End Type
```

In this VB declaration, *padfX is translated as a Long value, not an array and certainly not an array of Doubles. It will be up to us to interpret padfX, Y, Z, and M as pointers.

To get a pointer to an array of Doubles we use the following technique:

```
Dim dblArray(10) as Double
Dim pdblArray as Long
pdblArray = VarPtr(dblArray(0))
```

VarPtr(dblArray(0)) returns a pointer to the first element of dblArray, which is exactly what shapelib expects. (On my machine I get a value for pdblArray of 1797056: the memory address for this first array element).

VarPtr's inverse function is to recover our array from its pointer. To do this we use the Win32 API function RtlMoveMemory, commonly aliased as CopyMemory or MoveMemory.

```
Declare Sub MoveMemory Lib "kernel32" Alias "RtlMoveMemory" _
  (pDest As Any, pSource As Any, ByVal dwLength As Long)
Call MoveMemory(ByRef dblArray(0), ByVal pdblArray, _
  Len(dblArray(0)) * (UBound(dblArray) + 1)
```

Arguments for MoveMemory are a pointer to a destination memory location, a pointer to a source memory location, and the number of bytes to copy. In our example, the destination is the memory location used by the first element of dblArray (the rest of the array follows, uninterrupted). DblArray(0) is simply a single value of type Double, and passing it ByRef is equivalent, in VB, to passing its pointer. That is exactly what we want. The second argument is the memory pointer, which we have stored in pdblArray. We must pass this ByVal in order to pass its literal value (which we have decided to interpret as a pointer). Failing to pass pdblArray ByVal (i.e., if we pass it ByRef) will cause our program to blow up since we would actually be passing a pointer to our pointer. That is, we would be passing the address for a block of memory exactly 4 bytes wide (the size of a VB Long) containing the literal value of pdblArray. If we try to copy our whole array beginning at this memory location, we will overrun the 4 bytes allocated for pdblArray and overwrite memory used for who-knows what else. Our program will almost certainly crash.

The number of bytes we copy is calculated as the size of an array element multiplied by the number of elements in the array, calculated here as Len(dblArray(0)) * (UBound(dblArray) + 1). Pay careful attention to the number of bytes you instruct MoveMemory to copy, since you are copying directly to program memory and mistakes will have dire consequences. (well, only if you consider your program crashing dire. These are the kinds of activities engaged in by C/C++ programmers that your mother warned you about).

These are the basic techniques for using shapelib from VB. The accompanying source code contains a declaration module with all the shapelib 1.2-10 functions, structures, constants, and enumerations defined in Visual Basic. A test program is included which creates a simple shapefile and manipulates its data. The code is heavily commented with many Debug.Print statements. It is intended to be stepped through in debug mode to observe how the shapelib functions should be called.

Please send any comments and error reports to dgancarz@cfl.rr.com.

7 December 2003