Exercise 6 Strings, Variants and SafeArrays

This exercise demonstrates how to pass ***SafeArrays***, ***BSTRs*** and ***Structures*** between a Visual Basic client and a COM server written in C++. As you will soon discover, passing these parameters is a lot more complicated than you could reasonably expect. Nevertheless, these are just the kind of parameters COM servers work with in practice, so it is important to understand the issues involved.

Visual Basic can only work with automation compatible types and unfortunately, this rules out most of the common C++ types. In particular, we can't pass C++ character arrays (***CSTR***) to Visual Basic, nor can we pass any other type of array to Visual Basic. Even worse, some versions of Visual Basic won't accept user-defined types (structures) as parameters. Therefore, instead of using the familiar C++ types we have to use alternatives.

- Visual Basic uses a ***String*** type rather than a ***CSTR***. You can pass Visual Basic strings by using the ***BSTR*** data type.

- Visual Basic uses its own type of array. You can pass Visual Basic arrays by using the ***SAFEARRAY*** data type.

- Visual Basic does allow you to define structures, but you can't always pass structures to COM servers directly. However, Visual Basic will allow you to wrap up a structure as a ***VARIANT,*** which can be passed to COM servers. Unfortunately, converting a structure to a ***VARIANT*** is complicated.

Step 1 - Getting Started

Fire up VC6 and create a new ATL COM Wizard project in the directory:

COM Programming\Exercises\Strings, Variants and SafeArrays

Choose a DLL project with a name of ***DataTypes***.

Step 2 - Creating the COM Object

In class view, right click on the mouse and select the ***New Class*** option. Make sure that an ATL Class is selected. Choose a short name of ***Parameters*** on the names tab. ***Select 3 interfaces*** (uncheck the aggregatable option) and call them ***IBSTR, IVariant*** and ***ISafeArray***. Build the COM object.

Step 3 - Understanding the Visual Basic Types

Before we implement the above methods, we need to discuss how C++ and Visual Basic treat the various data types. Each of the data types are discussed below:

BSTR

These are the easiest to understand. The Visual Basic ***String*** type is mapped to a ***BSTR*** in IDL and C++. ***BSTR***s differ from normal ***CSTR***s because they contain a count of the number of characters in the string in addition to the null terminated character array. Before you can use a ***BSTR*** you must allocate storage for this array using the API function ***SysAllocString***. When you have finished with the ***BSTR*** you must de-allocate the array with ***SysFreeString***.

Arrays

You can use arrays in Visual Basic using a statement such as:

```
Dim array() As Long
```

However before an array can be passed to a COM server it must be converted into a ***SafeArray***. The ***SafeArray*** is actually a separate structure to the array itself. The ***SafeArray*** contains information about the array including the dimensions of the array, its upper and lower bounds and a pointer to the memory allocated to the array. Visual Basic will create this structure for you automatically, but in C++ you must create it using an API call.

For example, to create a single dimensional array of ***long***s in C++ write:

```
SAFEARRAY** ppArray
*ppArray = SafeArrayCreateVector(VT_I4, 0, 5);
```

and to access the data that the ***SAFEARRAY*** points at use:

```
long* data = 0;
SafeArrayAccessData(*ppArray, reinterpret_cast<void**>(&data));
```

When you have finished with the array it is important to release it:

```
SafeArrayUnaccessData (*ppArray)
```

Variants

Variants are special Visual Basic variables that are used to store any of the Visual Basic standard types (e.g. ***integer, long, single, string***). Furthermore, a variant can change its type at run time. For example, a variant can be used to hold a ***string*** at one moment and then later in the program hold an ***integer***. In C++ we have to use a ***union*** to achieve the same effect.

To simplify using variants in C++, Microsoft have defined a structure called a ***VARIANT,*** which contains a union describing all the Visual Basic standard types.

Types
Visual Basic allows you to define structures in much the same way as C++ except it calls them ***Type***s. For example, to define a structure describing a Rectangle we can write:

```
Private Type Rectangle
        left As Long
        top As Long
        height As Long
        width As Long
End Type
```

And to create a variable r of this type:

```
Dim r As Rectangle
```

Step 4 - Defining the Methods for the Interfaces

Now we can set up our COM object. We will define a separate interface for each type of parameter.

IBSTR
The ***BSTR***s will be created in a Visual Basic test harness and then passed to our COM server by reference. The ***IBSTR*** interface will have 2 methods:

```
ToUpper([in, out] BSTR* pString)
ToLower([in,out] BSTR* pString)
```

Add these methods from class view in the normal fashion. Implement the methods in the COM server.

IVariant
This interface will be used to pass structures to the COM server. Recall that Visual Basic does not allow structures to be passed directly. Instead we must wrap the structure up in a variant and pass the variant to the COM server.

Unfortunately, you can't just assign a user defined type to a variant. However, you can assign a user defined type to an array and then assign the array to a variant. Sound complicated? Wait until you try unpacking the structure on the COM server. You have to reverse the whole procedure in raw C++ code!

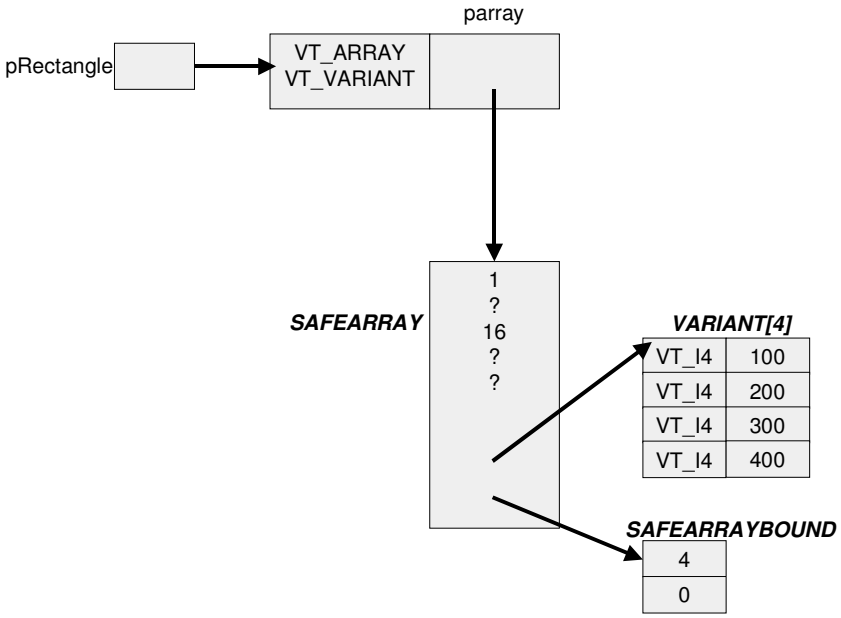
Here's the code to set up the variant in Visual Basic:

```
Dim packedStructure As Variant
Dim array(3) As Variant
array(0) = r.left
array(1) = r.top
array(2) = r.width
array(3) = r.height
packedStructure = array
```

The ***IVariant*** interface will have one method to test this out:

```
ExpandRectangle([in,out] VARIANT* pRectangle)
```

Add this method from class view in the normal fashion. The variant is passed by reference so that it can be updated in the COM server. Implement the method in the COM server to expand the height and width of the rectangle by 20 pixels and adjust the left and top positions to keep the centre of the rectangle in the same place.



ISafeArray

This interface will be used to test out passing arrays to our COM server. To keep things simple, we will confine ourselves to one-dimensional arrays, but multi-dimensional arrays are treated in the same way.

The ***ISafeArray*** interface will have 3 methods:

```
CreateArray([out] SAFEARRAY(long)* SafeArray);
DoubleArray([in,out] SAFEARRAY(long)* SafeArray);
SumTheArray([in] SAFEARRAY(long)* SafeArray, [out,retval] long* pTotal);
```

If you try to add these methods using the wizard, the wizard complains about the brackets after the ***SAFEARRAY***. This means you have to add these statements by hand to the IDL file. Recall that in C++ you can't use this notation and you should use an extra ***** in place of the ***(long)***:

```
CreateArray(SAFEARRAY ** SafeArray);
DoubleArray(SAFEARRAY ** SafeArray);
SumTheArray(SAFEARRAY ** SafeArray, long* pTotal);
```

Implement these methods as follows:

CreateArray:	create an array of 5 ***long***s in the COM server and then return it to Visual Basic.
DoubleArray	an array of 5 ***long***s is passed by reference - you should double each element of the array.
SumTheArray	an array of 5 ***long***s is passed by value – you should calculate the sum of each element and return the sum to Visual Basic.

Step 5 - Test in Visual Basic

Study the Visual Basic test harness in the example solution to see how to test your COM server. You can create your own test harness if you wish, but we suggest you copy the test harness from the example solution to save time.

As you can see, passing non-trivial parameter types between C++ and Visual Basic is very complicated. It is also very slow. Try modifying the test harness make multiple method calls and print out the timing statistics.