Creating Visual Basic VBXs:
The Visual Basic String

by Fred C. Hill

The fourth in a series of articles on the secrets of writing VBX controls in Borland Pascal.

Up to this point I've given you everything you need to write your own VBX control using Borland Pascal. In this issue we'll clean up a few loose ends and prepare to write an actual VBX which its foundations in the real world. The examples I reference here will be the same as those in the Control Development Guide for the CDK provided by Microsoft.

Most of the problems with writing a custom control is overcoming the differences between the language of the VBX developer and the language of the VBX user. Of course, for our purposes here the user will be programming in Visual Basic and you will be using Borland Pascal. It is entirely possible however that if you adhere to the VB 1.0 VBX standard that the VBX can be used in Visual C++ or Borland C++. I've even heard that version 1.0 VBX's will be supported in the Delphi 95 environment.

The majority of this installment will focus on the various methods of string handling between Visual Basic and Pascal. String handling requires special attention because, not only are they dynamically allocated variable length buffers, they are also handled differently in the Basic, the C, and the Pascal languages.

You will have to deal with null-terminated strings to deal will parameters passed to API's and other routines written in C, as well as Basic language strings which use a string descriptor, maintained internally by Visual Basic. Each of these will require conversion to and from the Pascal string formats.

Visual Basic is an event driven environment and as such makes no claim on a specific area of memory. Each of the string buffers is allocated dynamically and can be moved around in memory at any time. Because of this the VB string is referenced by a handle rather than a pointer. Unlike a pointer, a handle remains valid even as the data's address changes.

Visual Basic allocates the following two kinds of strings:

Handle	Description
HSZ	Handle to a null-terminated string.
HLSTR	Handle to a Basic Language string.

Table 1

The HSZ type string is a standard null terminated string normally found in C. We won't be specifically addressing this string type in this article but will use and explain it in future projects. Of course to use this data type you must de-reference the handle to get the address of the string data.

The HLSTR type string is exactly like the strings declared and used in Visual Basic. The routines provided in the Control Development Kit (CDK) and detailed in table 2 and 3 are used to create, use, and destroy the strings. The routines are used to copy, increase, and decrease the length. In the process of working with the strings they may be moved in memory to accommodate these changes or other events. They are not terminated with nulls and nulls are allowed in the body of the string. The length is maintained in the string descriptor which is managed by the Visual Basic string space.

Any string type can be used within the VBX but when passing string parameters to a Visual Basic Event procedure you

must use the HLSTR type.

The following tables detail the routines exposed by the VBAPI_.TPW unit provided on the enclosed diskette.

Function	Description
VBCreateHsz	Allocates a null-terminated string and returns the strings handle
VBDerefHsz	Returns a pointer to the address of the string.
VBDestroyHsz	De-allocates the string space and invalidates the handle.

Table 2

Function	Description
VBCreateHlstr	Allocates a Basic language string and returns the strings handle.
VBDerefHlstr	Returns a pointer to the address of the string.
VBDestroyHlstr	Deallocates the string space and invalidates the handle.
VBGetHlstrLen	Returns the length of the string.

Table 3

In addition to the functions defined in Tables 2 & 3 there are also functions for declaring a temporary Basic Language string. A temporary string is automatically deleted the first time they are used by Visual Basic. Visual Basic allocates space for up to 20 temporary strings. (See the discussion on temporary strings later in this articale).

Creating and using a string

lpStr := VBCreateHlStr(pb, cbLen);

Creates a Basic language string from the data pointed to by *pb*. Once created the string is managed as part of the Visual Basic string space.

pb is a par pointer to the buffer containing the string.

cbLen is the number of bytes in the string. This could be zero, in which case the string returned is zero length. pb will not be used if cbLen is zero.

lpStr is defined in the VBAPI_ and is the handle for the Basic-language string.

This string can be placed in a parameter string for an Event or passed as a string property. When you are finished with the string you must eventually free it using **VBDestroyHlstr**.

Temporary Strings

Creates a temporary Basic language string from the data pointed to by pb. Once created the string is managed as part of the Visual Basic string space. Visual Basic allows up to 20 temporary strings at one time and are automatically deleted when they are used by Visual Basic. These temporary strings are generally used when returning a string to Visual Basic as a function return value. Do not attempt to delete a temporary string by calling **VBDestroyHlstr**, instead use a VBAPI_ function that deletes temporary strings such as **VBGetHlstr**.

lpStr := VBCreateTempHlStr(pb, cbLen);

pb is a par pointer to the buffer containing the string.

cbLen is the number of bytes in the string. This could be zero, in which case the string returned is zero length. pb will not be used if cbLen is zero.

lpStr is defined in the VBAPI_ and is the handle for the Basic-language string.

This string can be placed in a parameter string for an Event or passed as a string property.

Strings that move

What! Why would the strings I allocate and deallocate move? Even seasoned developers (myself included) have had trouble with the concept and problems with Windows in general and event programming specifically. It takes a long time to lose the "DOS" mentality and finally feel secure in an environment where Events are taking place which are no in your control.

Anytime you reference a Windows API function there is a better than average chance that the string you were pointing to a mere micro-second ago is now somewhere else in memory. The address you so laboriously dereferenced must be dereferenced again. This is particularly true with the Basic language strings as you'll see in the following sections.

lpStr := VBDerefHlstr(params.ClickString);

This routine returns a far pointer to the string data This pointer becomes invalid as soon as the string is moved in memory, and remember, any call to a Windows API can have that effect. Even though you can dereference the handle you cannot directly change the data length. This is done through the **VBSetHlstr** function. The deref function is primarily useful in examining the contents of the buffer.

Getting String Data

uShort := VBGetHlstr(hlstr, pb, cbLen);

Copies the data from the Visual Basic string, pointed at by the handle (*hlstr*) and places it in the buffer pointed to by *pb*.

hlstr	Handle to the Basic Language string
pb	Far pointer to the desination buffer. It must be large enough to hold the cbLen number of bytes.
cbLen	Maximum number of bytes to copy.

uShort will contain the actual number of characters copied into *pb*

example:
cBuffer := ^Buffer;
cbCount :=
VBGetHlstr(hlstr,cBuffer,sizeOf(Buffer) -1)

Setting String Data

err := VBSetHlstr(phlstr, pb, cbLen);

This function assigns a new string value (*pb*) to an existing Basic language string (*hlstr*). The string can be smaller, larger or equal in size to the existing string and will be managed by the Visual Basic string space. The string will almost always be moved in memory to accomodate the changes in length.

phlstr A far pointer to the string handle.
pb Far pointer to the string data which will replace the existing Basic language string.
cbLen Length of the new string.

example:

VBSetHlstr(phlstr, NULL, 0); will set the string to an empty string
VBSetHlstr(phlstr, 'New String', 10)

Replace an existing string

VBSetHlstr(phlstr, hlstr2, -1)
Copies the existing string into a second existing string. (*hlstr2*)

Cleaning up your environment

When you are done with the strings your VBX allocated, you should always clean up after yourself. Visual Basic doesn't do the clean up so anything you allocate will be left behind. Visual Basic developers have become very adept at tracking down those VBX's which are grabbibg and holding memory and they arn't quiet about denouncing them to the world at large.

VBDestroyHlstr(hlstr);

Just pass the handle and the memory used by the Basic language string is freed. The handle is now invalid and cannot be used to reference anything. Whatever you do, don't use this routine to destroy temporary strings. VBGetHlStr will free temporary strings.

Null terminated strings

As I'm sure you'll notice, I didn't cover the Null terminated strings (HSZ) at all. Actually the calls are very much like the HLStr calls and can be used interchangeably with the exception of the Event calls. Only an HlStr string can be passed as a parameter. Examine the VBAPI_.PAS file and compare it to the CDK reference manual and you'll begin to see how easy it is to use Visual Basic strings in your VBX. If you look at the example in the last issue (The Pascal Magazine, Issue 3) you'll see numerous places where the null terminated strings were created referenced and destroyed.

This is the final installment in the initial VBX training series. In the next issue we will begin creating a VBX for our own use. This VBX will continually monitor the Windows resources, the available memory and the available disk space. Each of these items will be optional. If a resource is selected and a change is detected the VBX will fire a custom event to alert the Visual Basic program when these resources change.

If there are additional tasks you'd like to see done in Borland Pascal, please drop a note to the editor of this magazine or to me on Compuserve and I'll try and include it in this or future VBX's.

Reference: Professional Features Book 1, Microsoft Visual Basic 3.0 Custom Control Guide

Fred Hill is president of Micro System Solutions. His company produces DOS and Windows applications in Borland Pascal 7.0 and Visual Basic 3.0. He may be reached on Compuserve 76060,102