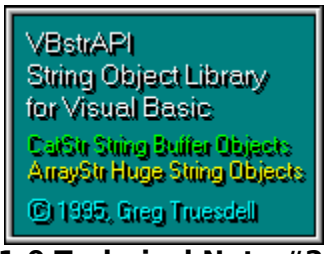

VBstrAPI.DLL v1.0 Technical Note #2
March 22, 1995

Visual Basic Strings and VBstrAPI

A recent trouble report was recently sent in by Jim Moran at Honeywell. It demonstrates Visual Basic string allocation limitations and VBstrAPI limitations. The trouble report is included here:

> Whenever I create an array (using CreateNewStringArray) with an element over 32K, then use GetArrayStr to get a string I have stored in this element, VB gives me a GPF fault at 003A:9002.
>
> Below is a code fragment that reproduces the problem, put in a Form_Load event:
>
> ```
> -------------- Code starts here --------------
>  'Create an array element over 32K
>  'Less than 64K, so should be OK
>  h = CreateNewStringArray(1, 33001)
>  MsgBox "Array created successfully"
>
>  'Make a string to put into the array element
>  tmp$ = String$(33000, "A")
>  p = PutArrayStr(h, 0, tmp$)
>  MsgBox "PutArrayStr successful"
>
>  'Put array element into string variable, crashes here
>  tmp2$ = GetArrayStr(h, 0)
>  'Never makes it to this message box
>  MsgBox "GetArrayStr successful"
> --------------- Code ends here ---------------
> ```
>
> According to your help file, it should be possible to have array elements as large as 64K.

The following information arose from my investigation into his trouble report.:

I hope I can explain the situation clearly. The problem as described is partly Visual Basic and partly VBstrAPI generated. VBstrAPI can, and indeed does, handle strings up to 65534 characters in length (including the null-terminator). The code provided, although ultimately resulting in a GPF in VB.EXE, is a classic demonstration of Visual Basic's out-of-string-space problem. (In fact, the very reason for needing a library like VBstrAPI.) It has also, unfortunately, demonstrated a problem I'm not certain of how to overcome; namely: how to detect when VB can not handle a string returned by a DLL. Let me begin by breaking the problem into sections:

VB String Space Usage

The following code will cause an Out of String Space error:

```
1: Dim Test1$, Test2$
2: Test1$ = String(33000,"A")
3: Test2$ = String(33000,"B")
```

The important thing to note is that the third line causes the error. This is because VB can not handle more than about 65,500 characters in the string space provided for each procedure. It is impossible to set a string to 65,535 characters. It is also impossible to have the sum of all common strings declared in a procedure greater that 65,500. VB just can **not** do it. Hence, the reasoning behind VBstrAPI.

The following code **should** also cause an Out of String Space error:

```
1: Dim SHandle%, Test1$, Test2$
2: SHandle% = CreateNewStringArray( 200, 65530 )
3: Test1$ = String(33000,"A")
4: rc = PutArrayStr( SHandle%, 0, Test1$ )
5: Test2$ = GetArrayStr( SHandle%, 0 )
```

The error should occur in the last line. The reason, of course, is because the sum of the string space used by `Test1` and `Test2` is greater that VB can handle. As you have noticed, it does not cause an Out of String Space error; instead it causes a GPF error. I will touch on the reason for this in the next section.

The above code could be rewritten to demonstrate how it can be made to work within VB's string space restrictions:

```
1: Dim SHandle%, Test1$
2: SHandle% = CreateNewStringArray( 200, 65001 )
3: Test1$ = String(65000,"A")
4: rc = PutArrayStr( SHandle%, 0, Test1$ )
5: Test1$ = ""
6: Test1$ = GetArrayStr( SHandle%, 0 )
```

This code will not cause an Out of String Space error for two reasons: Firstly, only one string variable is in use and Secondly, the string variable is erased (set to "") prior to being used to contain the text returned by GetArrayStr().

VBstrAPI is not able to overcome Visual Basic's inherent limitations concerning the amount of string space available to common local variables. It's strength is in providing string space outside of VB's reach, thus allowing large variables to interact with each other through appropriate programming steps.

Why does VBstrAPI cause a GPF Error when VB is out of local string space?

Well, the answer is that VBstrAPI doesn't know that VB is out of string space. So, when GetArrayStr is called to return a string, VBstrAPI passes the string buffer to Visual Basic's temporary string intact. No attempt is made to truncate the string

based on the space left. The Visual Basic API does not provide a way to do determine this, at least I have not run across a way. It would appear that the Visual Basic API `VBCreateTempHLStr()` function does not qualify the length of the string passed to it and, therefor, attempts to overwrite it's own string space. This, most certainly, would result in a GPF error.

So, what am I going to do about it?

This is something I will have to research, but I am not optimistic. Certainly the library's documentation will have to be updated to present this warning and work-around. Fortunately, as you have brought this to my attention, it has moved me to re-check more code associated with string declarations. I did find an unrelated bug that I will fix and release in the next revision.

The next release of VBstrAPI (rel 1.32) will include improved error checking and runtime error generation. The appropriate maximum size of an ArrayStr object element is 65534 bytes (2 bytes overhead.) VBstrAPI will now generate Visual Basic runtime errors if invalid limits or handles are passed. Unfortunately, this will not significantly help avoid the problem described in this technote.

So, I'm not sure if this is a real bug or not. VBstrAPI certainly does override some of the internal error code within Visual Basic. This is a good thing, usually, but seems to have an unfortunate, and possibly unavoidable, side effect. As a result of my investigation into the problem report, I'll have to live with this situation as an 'unresolved' (and frustrating) anomoly for now. I will contact Microsoft developer support for more information.

Summary

There are a number of work-arounds for Visual Basic's ~64k local string space problem. VBstrAPI was designed to help VB Programmers resolve some of those problems, as well as provide large, global string buffer objects. The string objects provide a method for declaring, using and destroying a large amount of string space within a procedure, among other things.

Care should be taken when handling large strings with local variables in a procedure. The rule is: the sum of all local string usage + string variable overheads can not exceed ~64K. Period.

If anyone would like to comment on the information included here, please contact me at:

Greg Truesdell
CompuServe ID: **74131, 2175** or via Internet at **74131.2175@compuserve.com**