VBstrAPI
String Object Library
for Visual Basic
CatStr String Buffer Objects
ArrayStr Huge String Objects

Visual Basic String Object Library

A string buffer and huge string array library.

I know, you're probably thinking "What?! Yet another string library? And I suppose I need another *left foot*!" Well, no. You probably don't need another left foot. At least I sincerely hope not. However, I have not tried to re-invent the square wheel. What I *have* done is to fill a particularly specialized need that arose from the smoky depths of the **Visual Basic Out of String Space** dungeon.

There are several realities we all, as VB programmers, have to face. The fact that VB provides a limited amount of string space is one. While developing several specialized applications to access a massively parallel database computer (the ATT3600/Teradata DBC1012), and to overcome the inherent limitations in MS Access®, I found myself bashing my head against the proverbial brick wall. Although I won't get into the issue of ODBC stack problems when synchronously executing multiple queries right now (although I *have* written a library that solves that problem), I will get into the issue of buffering medium to huge amounts of text data. Writing truly professional ODBC compliant applications requires more than Visual Basic's Snapshots, Dynasets, bound Data controls and limited string space. Fortunately many of these problems can be addressed using DLLs written specifically for Visual Basic.

<u>Problem #1: Visual Basic String Limitations</u>

A textbox is limited to ~64k of text. That can be too small as it is, but it's made even smaller by Visual Basic's imposition of the 32k barrier. Clever programmers know that you can buffer the text and 'switch pages' by allocating new textboxes or by re-using the visible one. We also know that filling a textbox a line-at-a-time (even when the textbox is hidden) takes a very much longer time than just assigning the whole thing at once. Buffering the text prior to filling the textbox is an obvious answer. However, buffering the text using normal VB means can have two side effects: the dreaded out-of-string-space demon, or the moderately slow string concatenation demon.

Out-of-string-space is caused when code is written that results in more than 64k of string space being used in the same form (as well as other situations.) When using Mid$, Left$, Right$ and the "&" concatenation (or pipe) character, string space is usually allocated in the same segment. That means that the sum of the characters in all arguments may not exceed 64k. However, an external library may assign as much space as it likes and handle strings more effectively. Still, a string is never larger than 64k. And, using some esoteric methods, you *can* overcome the 32k barrier. Not a very comfortable thought.

Concatenation of strings is another issue. VB is written in C/C++. C/C++ uses a number of string functions to handle the concatenation of strings. If, as a C/C++

programmer, you are presented with the problem of concatenating one string to another, you will likely use the **strcat()** function to do so. The **strlen()** function is often used to determine the length of the string. This function starts at the beginning of the string and counts characters until the terminating null (0) is located. *Viola*, the string length is found. Now if you need to concatenate strings to another string often, then each time the string's length is calculated to locate it's end. Now you can concatenate the other string. If you are lucky enough to know that you will be concatenating a particular string a number of times, then you can maintain a pointer to the end of the string and just add characters to the end. The **strecopy()** function can be used in some cases. VB does not give you this luxury.

So, what is my point, anyway? Well, as a program receives data from some source it will either use a method such as:

```
Form1!TextBox1.Visible = False
...
...
TextBuffer$ = ... some data source ...
Form1!TextBox1 = Form1.TextBox1 & TextBuffer$ & Chr(13) & Chr(10)
...
...
Form1!TextBox1.Visible = True
```

or:

```
TextBuffer$ = TextBuffer$ & ... some data source ... & Chr(13) & Chr(10)
... all data received ...
Form1!TextBox1 = TextBuffer$
```

The first method is tremendously slow; even on a Pentium. At least I think so. The second method is better, although after each "=" and "&" VB is going to do it's 'concatenation' thing. Faster, but still fairly slow. If you have ever tried reading a text file using **Open** and **Line Input** to fill a TextBox, then you know how slow *that* can be. Opening a file as BINARY and using **Get** to read the file is *light speed* in comparison. So would it not be nice if we could just blast the text into a buffer in the most efficient way possible, and then just copy the text in one flash into the TextBox? Well, yes ... it would.

VBstrAPI exports an object called CatStr (Concatenation String). This object can create a large number of 64k concatenation string buffers. Each buffer is designed to know where the end of itself is, and is lightning fast at concatenating strings. Since in many cases a CR/LF is required at the end of a line, the object is designed to handle that bit automatically as well. Since you may wish to buffer fixed-length fields of text, the object is designed to handle concatenating and returning fixed-length blocks too. The CatStr object is a 1 to 65535 character object. You can define it to whatever size within that range.

Since the CatStr Object Server within VBstrAPI can handle a large number of strings and can be used by any number of applications, CatStr objects are referenced by a 'handle'. I mention this now so I can give you an example of how CatStr can be used. (See VBstrAPI.HLP for a tutorial on object servers)

```
CHandle% = CreateNewCatString( 32768 ) ' string size + 1

rc% = CatStrAddLine( CHandle%, ... data source ... )

... repeat until all data received or rc% indicates that the string is full ...
```

```
Form1!TextBox1 = CatStrCopy( CHandle% )

DestroyCatString CHandle%
```

In this example, a CatStr Object is created, used and destroyed. **No VB string space or data space is used.** And, as a bonus, the string concatenation method used is *very* efficient. What is not so obvious is that the CatStrAddLine method added the CR/LF automatically to the line of text *and* returns a status code to let your program know if the Add was successful. If the data source is greater than 32k in length, my program can detect this and switch pages ... or something. Anything but fail or cause an out-of-string-space error.

If the data source were feeding you with fixed-length strings that you wanted to buffer, then the program would be only slightly more involved. For example:

```
Dim FieldBuffer As String * 40 ' say, a 40 character field

CHandle% = CreateNewCatString( 65536 ) ' maximum size
While ... there is data ...

        FieldBuffer = ... data source ...
        rc% = CatStrAdd( CHandle%, FieldBuffer )

Wend

CatStrResetCLP CHandle% ' this resets the Current Line/Field Pointer
                       ' so CatStr can begin at the beginning of the
                       ' buffer

Status% = 0
While Status% = 0

        ... some data sink ... = CatStrNext( CHandle%, 40, Status% )
        ' Status% will contain -1 if no more fields

Wend

DestroyCatString CHandle%
```

In this more detailed example, CatStr is handling fixed-length strings. Maybe you have noticed something in all of this? Yup, CatStr is treated like a serial file. You append strings to it using the CatStrAdd method (for fields) and the CatStrAddLine method (for text lines). You can also use CatStrAdd to add variable-length lines where you do not want or need CR/LFs. CatStrAdd will simply concatenate *unaltered* any string to the CatStr object's buffer. CatStrNext and CatStrNextLine are used to retrieve the next n-characters or the next logical line from the buffer. **(See VBstrAPI.HLP for a more detailed description of all 10 CatStr Object methods.)**

So, the CatStr Object is a serial (or concatenation) object much like a serial file. It has it's uses. I depend upon it, and so do my applications.

Problem #2: Buffering very large sets of strings

With all of it's advantages, CatStr is just not enough in some cases. In my work I often need to collect large datasets and either analyze, graph, pattern-match or copy them. I am sure all of us have used temporary disk files and databases to handle large amounts of data. This, however, is seldom the *fastest* method available. I have tried using humungous ram disks to solve some problems, and that often works for me. Not so, however, for my customers and work mates.

Not everyone has the luxury of lots of RAM, either; but if you do, then how to use it with VB, eh? As I mentioned earlier, I deal mostly with large datasets of fixed field and row (record) sizes. Buffering this data using Snapshots and DynaSets has proven, at best, inefficient and, at worst, impossible. So I created the ArrayStr Huge String Array Object. Although the name infers that this is an array much like any that VB creates, well, it is not.

The ArrayStr Object is designed to buffer as much data in memory as is possible. This buffer needs to be serial in nature, much like CatStr is, but also must be like any array: randomly accessible. So the ArrayStr Object has two personalities: Serial and Random access.

For example:

```
SHandle% = CreateNewStringArray( 10000, 1024 ) ' 10,000 rows of
                                              ' 1,023 characters

While .. there is data ..

        RowBuffer$ = ... some data source ...
        rc& = PutArrayNext( SHandle%, RowBuffer$ )
        DoEvents  ' allow windows and other programs to enjoy the weather

WEnd

'
' now, we can do whatever with the buffered rows
'

' serial example:

ArrayStrSetCLP SHandle%, 0          ' reset the Current Line Pointer

For ii& = 0 to 9999

        Print GetArrayNext( SHandle% )

Next

' Random access example:

Print GetArrayStr( SHandle%, 4321 ) ' the entry 4322 (zero-based array)

' insert a row
status% = InsertArrayStr( SHandle%, 5000, "Insert this in the middle." )

' delete a row
status% = DeleteArrayStr( SHandle%, 2145 )

DestroyStringArray SHandle%
```

In the above example, you can see that the Huge ArrayStr object can handle very large buffers **and** insert and delete rows *in memory*. I use this object often to buffer a large number of rows and pipe them 32k at a time to TextBoxes or Grids. With a little programming, you can give the impression to your application's user that almost instant access is available to anywhere in the table or file.

I also use CatStr objects to buffer a block of ArrayStr lines so the user is offered a number of rows to view while the ArrayStr object is still receiving rows.

(See VBstrAPI.HLP for more complete information on all 13 ArrayStr methods.)

Summary

The point of this document is to give you an idea of what VBstrAPI.DLL can do for you. Even if you decide that you don't need or want the library, perhaps the discussion has given you a few ideas. Programming is not a particularly easy job, and we need to help each other whenever we can.

Obviously, I hope you think the library is valuable. Please give it a try. The Shareware distribution archive and library can be evaluated for 30 days with no obligation. If you decide to buy, then you will receive a registered, royalty free version of the library. No special keywords for the registered version are necessary. By using a unique registration keyword, however, you will be able to self-register updates and maintenance releases until a major version change.

And for a **limited time**, the library is only **US$15.00**. Not bad, when you think of it. The library will be ready for distribution on or before February 20, 1995.

Other Visual Basic Libraries from the Author

VBossAPI.DLL

Have you ever needed to create a programming or script language to include in your application? Maybe you wanted to parse text files or program files. Perhaps you want to include numerical expressions and variables in a special calculator or application.

Well, VBossAPI.DLL provides the tools you need to develop text parsers and script languages. It's Token/Keyword parsing engine makes it easy to customize your script or parser for other languages: French, German, English, Spanish ... you name it.

It supports variables, data types, tokens and keywords, an expression evaluator and comes with an example language application that shows you what the parser is doing each step of the way.

Look for VBOS11.ZIP in the MS Basic forum on CompuServe. Use SWREG to register.

Only US$19.95

VBdbcAPI.DLL

ODBC compliant applications written in Visual Basic (and other languages too) that need to support multiple-parallel multi-tasking query execution have a problem: ODBC needs a lot of stack space. Also, many ODBC calls require the

Visual Basic programmer to jump through C/C++ hoops to decode returned data. And, there is no way to implement SQLBind() using Visual Basic without using a library.

Well, here it is. This library is not just a wrapper for ODBC calls. Who needs 'just another wrapper' any way? This library supplies it's own stack and variable memory (where applicable) for VB. An almost unlimited number of query activities can be running in parallel, thanks to VBdbcAPI.DLL's special stack handling features.

Now many ODBC calls will return native Visual Basic strings to simplify your code and improve debugging.

This library will not be released until March, 1995. As soon as the help file and documentation are complete the distribution package will be available on CompuServe first.

Expected price is: US$24.95