Small Strings

This module is part of the original MMBasic library. It is reproduced here with kind permission of Hugh Buckle and Geoff Graham. Be aware it may reference functionality which has changed or is deprecated in the latest versions of MMBasic.

This code is intended for early versions of MMBasic where strings were fixed at 255 characters in length (but after PEEK and POKE were implemented). This is no longer the case but this code is included here partly to maintain coherence with the old MMBasic library but also because it demonstrates slicing strings and holding them in elements of an array - this could easily be adapted to hold strings larger than 255 characters (which is a limit in V5.2). It also demonstrates the use of PEEK & POKE to manipulate variables directly… a lot to study here.

Small Strings An MMBasic program by TassyJim - Oct 2012

Save space by using short strings. MMBasic strings are a fixed 255 bytes long. If you have a lot of short strings, you can save a lot of valuable memory by using short strings. The code creates an array of any (< 255) length strings. It would be a bit slow for intensive string manipulation but it can save heaps of space.

Strings use 256 bytes. The first byte is the length of the string and the remaining 255 bytes are where the actual string is stored. I used the same method so if you need strings 32 bytes long, you will have to specify 33 as the size. In MMBasic you can specify the starting array number as zero or one. I have stayed with one because that is the way I have always done it. I use the 'zero' location to store the new array size to save on two variables.

An example that might be relevant to you.

I have a list of 652 Aussie towns. The longest name in the list is “Kingston South East” which is 19 characters long. An array of 652 X 20 bytes is needed. 652*20/256 = 51. My method creates an array with 51 elements which uses about 13K - Full length strings would need 163K of memory. I also have a list of 65K place names but that's too much for the Maximite to chew on!

I deliberately kept to single dimension arrays but it is easy to use the short array as 2 dimensions. Most times a single dim array for the string and another standard numeric array for the rest of the data. In the Towns example this numeric array will store the latitude and longitude of the towns.

It is thanks to Geoff implementing PEEK and POKE and giving us the location of the variable table that I was able to do this.

The code makes use of PEEK and POKE so it does have the potential to cause havoc.

function makeSmall(aa,bb)

First we work out the number of normal string elements needed to store our small array. We then create the holding array and find its memory address. The first 2 memory locations are used to store the new array dimensions.

function PutSmall(b, a$)

We pass the array element and the string for storing. If the string is too long, it is truncated without any error message. The function returns -1 if an error occurs or the length of the string is incorrect.

function GetSmall$(b, a$)

Pass the array element and an optional error string. Return the string or the error string if array is out of bounds

The code has been tested on various hardware but not as part of a big program. Jim

SSTRING.BAS

```
'small strings by TassyJim
x=makeSmall( 39,40) ' 38+1 bytes long, 40 elements
print x;" string elements used"
memory
for n = 1 to 41
test$="Test string number "+str$(n)
x=PutSmall(n,test$)
print test$;" ";x
next n
print
print "Now we retrieve the strings"
for n = 1 to 41
test$=GetSmall$(n,"woops!")
print n;"  ";test$
next n
memory
end

function makeSmall( aa,bb) 'aa= string length, bb= array elements
local ascii$,g,n,c
makeSmall=-1
dim smallstring$(int(aa*bb/256)+1) ' allocate the required memory as a
normal array
for g=0 to 32 'look through a maximum of 32 variables
ascii$=""
for n = 0 to 32
c=peek(VARTBL,n+g*56)
if c=0 then exit for
ascii$=ascii$+chr$(c) ' retrieve variable name (end is chr$(0))
next n
SSl=peek(VARTBL,52+g*56)+peek(VARTBL,53+g*56)*256 ' string storage low word
SSh=peek(VARTBL,54+g*56)+peek(VARTBL,55+g*56)*256 ' string storage high word
if ascii$="SMALLSTRING$(" then
poke SSh, SSl, aa 'store the max string length
```

```
poke SSh, SSl+1, bb ' store the max number of elements
makeSmall=int(aa*bb/256)+1
exit for
endif
next g
end function 'returns -1 for error or number of normal string array elements
used

function PutSmall(b, a$) ' element number, string to store
local n,aa,bb
aa=peek(SSh, SSl) ' retrieve the max string length
bb=peek(SSh, SSl+1) ' retrieve the max number of elements
n=-1
b=int(b)
if b>0 and b<=bb then
    if len(a$)>=aa then
        a$=left$(a$,aa-1) ' trim the string to max length
    endif
    poke SSh, SSl+b*aa, len(a$) ' store the string length
    for n = 1 to len(a$)
        c=asc(mid$(a$,n,1))
        poke SSh, SSl+b*aa+n, c ' store the string
    next n
endif
PutSmall=n ' returns length of string or -1 if subscript out of range
end function

function GetSmall$(b, a$) ' element number, error message if out of bounds
local n,aa,bb,s
aa=peek(SSh, SSl) ' retrieve the max string length
bb=peek(SSh, SSl+1) ' retrieve the max number of elements
b=int(b)
if b>0 and b<=bb then
a$=""
s = peek(SSh,SSl+aa*b) ' size of element
for n = 1 to s
k=peek(SSh,SSl+aa*b+n) ' build the string
a$=a$+chr$(k)
next n
endif
GetSmall$=a$ 'returns string or error message
end function
```