FEBRUARY 1985 EDITION OF ATUNC NEWSLETTER
VOL 2 # 2

Undocumented Limitations of Apple /// Business Basic

Part 1 Daryl Anderson D A DataSystems 12/09/84

String Array Limitations - VARIABLE & OUT OF MEMORY ERRORS

Known to apply to versions 1.0 and 1.1 and expected for 1.2 but possibly not the last off the assembly line 1.23 ?!
(1) Because of the way Business Basic stores string data there is an absolute limitation of 21841 total elements of a string array. This limit applies at DIM time (e.g. before any actual string data is stored). Thus DIM A$(21841) on my machine leaves 111514 bytes free and DIM A$(21842) causes an OUT OF MEMORY ERROR. Similarly DIM A$(99,217) is ok (100*218 elements) but DIM A$(100,217) is not.

Try this program on your machine...

```
10 CLEAR
20 INPUT "A$ dim value : ";ADIM
30 DIM A$(ADIM)
40 PRINT "FREE MEMORY = ";FRE
50 GOTO 10
```

(2) YOU SHOULD ALWAYS TRY TO DIMension STRING ARRAYS BEFORE ALL OTHERS. Depending entirely upon the specific dimensions of a numeric array DImmed prior to the string array, you may see the dreaded VARIABLE ERROR.

Try this one :

```
10 CLEAR
15 DIM X%(30000)
20 INPUT "A$ dim value : ";ADIM: rem try 1830 then 1950
30 DIM A$(ADIM)
40 PRINT "FREE MEMORY = ";FRE
45 A$(ADIM)="xxxxx"
50 GOTO 10
```

Note that the assignment of actual data in line 45 is necessary to trigger the error since the DIM only works on low-memory pointers but the assign forces attempted creation of an illegal "Back Pointer" - see below.

Explanation of this second error, and diagnostics to be applied if you must DIM a numeric array first, requires some understanding of how basic stores string arrays... to whit :

GARBAGE COLLECTION

While processing simple string assignments like A$=A$+B$ or while evaluating INPUT values or a variety of other 'stringy' data, Business Basic must create a number of temporary strings in storage. In the process of working with temporary and permanent strings Basic just uses up more and more free memory. When storage fills up with these temporary and permanent string data Business Basic (and most interpreter basics for that matter) must go back through memory and free up the space used by the temporary strings. Since they are intermixed with permanent string values this is a somewhat involved process - usually referred to as GARBAGE

COLLECTION.

IBM vs APPLE ///

If you sit down to an IBM PC running interpreted basic in a 'stringy' program, sooner or later you'll hit the Garbage Collection (hence GC); on this machine, and most Microsoft Basic machines this can really take some time - up to 2-5 minutes I've seen. Many users try the old "toggle the on-off switch" trick and lose everything. (Break, cntl-c, etc don't work during GC because Basic is buried deep within its own internals at this point).

On the Apple /// Business Basic a design decision appears to have been made to try to avoid this GC hangup.

STRING DESCRIPTORS

Strings in BusBas are implemented, to a point, in a manner similar to MS basics; the actual string contents are stored in high memory, filling down to a lower limit defined by simple variables and arrays which are less dynamic. An entry for each string is stored in low variable memory which looks a lot like simple variable storage but is actually a 'descriptor' containing IN THREE BYTES the length of the actual string data (on byte) and a number indicating the location of the real string data in high memory (two bytes).

BACK POINTERS

To alleviate the GC hangups, BusBas also stores a 3 byte BACK POINTER up in high memory with the actual string data which identifies the string type (e.g. temp, array) in one byte and the location of the FORWARD POINTER or STRING DESCRIPTOR in low memory in two bytes.

The location of the string descriptor is described as an offset from a base value which is the beginning of array storage (for array strings) or simple variable storage (for non arrays).

But since this value is contained in 2 bytes the maximum it can contain, and thus the maximum offset of the string descriptor from the base, is 64K or 65,536 bytes.

Now we can see why (1) and (2) occur.

Explanation of (1) above.

If the very first array declared (DIMmed) is A$ containing a long list of three byte slots for future descriptors, each another 3 bytes offset from that base, then eventually one of those slots will be more than 65535 bytes from the base. Checking 65536/3 = 21845. Well it turns out that each array has a bit or header info stored in low memory before the actual slots, for our A$() this is 7 bytes. Remembering that DIM(21841) gives 21842 elements we get 7 + 21842 * 3 = 65533, 2 bytes free and no room for another triplet.

Interesting note is that part of that header info is the variable name. Adding 2 bytes in DIM AAA$(21841) is OK but CLEAR then DIM AAAA$(21841) bombs.

Explanation of (2) above.

DIMming other non-string arrays before string arrays has the same effect of "using up" offset values from the base of 'start of array storage' which the Back Pointers must refer to.

Dimensioning X%(30000) takes up 60000+ bytes at the front of array storage and A$(0) will be at offset 60015, thus leaving room for only (65535-60015)/3 = approximately 1840 additional slots.

Note that the DIMming will not cause an error, since it only builds a long string of 3 byte descriptors. It is only when an attempt is made to allocate an actual string for an element outside the offset range (e.g. A$(1900) or A$(10000) here) that the error occurs since it is only then that the Interpreter tries to build a back pointer with an offset value greater than 64K.

Part 2 Daryl Anderson D A DataSystems 12/20/84

Further String Array Limitations - OUT OF MEMORY ERRORS

Known to apply to versions 1.0 and 1.1 and expected for 1.2 but possibly not the last off the assembly line 1.23 ?!

Have you ever gotten an OUT OF MEMORY ERROR, then done a PRINT FRE only to be told that you've got 108,427 bytes free ? If so read on.

(1) Because of the way business basic stores strings there are some absolute limits to the total amount of string data which can be associated with a given program. The exact limits are very dependent upon specifics of the application. Surpassing the limits will result in a VARIABLE ERROR.

Background :

Each simple string or element of a string array is associated with a 3 byte string 'descriptor' in low memory (details in Part 1). The significant portion of that descriptor is a 2 byte component which specifies the actual location of the string data itself - specifically it carries the offset of that data from the top of free memory. Since this is a 2 byte value it can only manage to legitimately describe a value of x'FFFF or 64K.

Thus the greatest offset of a piece of string data from top memory is 64K and the overall limit to string data total for a program is 64K (=65535 bytes).

However :
This does not mean you could store, for example, 512 strings of 128 bytes each or 65536 single character strings. This is because, as mentioned in UNDOC1.BAS, each collection of string data in high-memory carries with it the useful but infamous 'back pointer' with 3 bytes of data.

THUS the high-memory space (and thus the piece of that max 64K offset) used by each string is :

{string_len + 3} bytes.

SO... 65536/4 = 16384 single character strings max
65536/83 = 789 80 byte strings

and various mixes in between.

TYPICALLY you will run into this problem if, for example, you were writing a Word Processor in Basic and wanted to manage 1000 lines of max 80 characters in memory, about 20 pages. Figuring on a 256K machine the roughly 80K storage is available. Basic will calmly allow you to DIM LINES$(1000) then BOMB out at about line 700. Since you've inevitably got other strings, including quoted literals, in the program, the 789 noted above is not reached.

TRAPPING for this error is tricky since Basic's string temporaries themselves reside in this high memory area and thus the code that manages

the error might itself hang, internally. Performance will degrade significantly as you approach this limit as garbage collection rate increases.

NOTICE that single character strings and other small ones are extremely space inefficient - total space required by any string, including descriptor
is :

{string_len + 6} bytes

so 1000 single character strings take up 7000 bytes.

Part 3 Daryl Anderson D A DataSystems 12/20/84

Numeric array limitations - OUT OF MEMORY ERROR

Although not at all as esoteric as the prior two notes in this series, I thought it might be useful to briefly document another undocumented limitation of Business Basic.

SPECIFICALLY

No single numeric array may occupy more than 64K (=65536) bytes of memory. This occurs because the array header stores a 2 byte value containing the length of the space occupied by the array. As in the prior two notes, this 2 byte limitation works out to a64K cap on the number to be represented.

This element of the array header is necessary and quite useful since it allows the Basic interpreter's "find this array entry in memory" routine to quickly skip over the space occupied by a non-matching entry.

Since integers store in two bytes, reals in four and long integers in eight, the following approximate max DIMs apply:

```
DIM int%(32750)
DIM real(16375)
DIM lint&(8175)
```

These values are only approximate since the length info in the header included the length of the header itself which takes up more space for longer variable names and larger number of dimensions.

Try these...

```
DIM A%(32763)  ...ok  ...CLEAR
DIM AA%(32763) ...nope ...OUT OF MEMORY

DIM A%(32763)  ...ok   32764 total elements right
DIM A%(3,8190) ...nope but still 32764 elements = (3+1)*(8190+1)
```