Visual Studio Magazine Online
Classic VB Corner

Easing the Data Cramp

64-bit versions of Windows present perils for Win32 apps, and opportunities too. For example, it's incredibly easy to double your address space.

March 22, 2010 · by Karl E. Peterson

Hi, my name is Karl, and I, uh, like Windows 7. Hard to believe I'm here saying that, but there you go. Given my initial net-positive reaction, I decided I might as well dive right into (off?) the deep end. So I added another 8GB of RAM, and a terabyte of mirrored disk for good measure, to my main system and loaded up the x64 version. It's a strange new world, to be sure.

A friend had told me that "even 32-bit apps work better" in 64-bit versions of Windows, so I decided to look into that. I vaguely recalled that there was a way you could increase the address space in an application to 3GB, and after just a little googling it started coming back. By using the /LARGEADDRESSAWARE switch while linking a program, you tell Windows that you don't plan on doing "anything stupid" with pointers, like signed comparisons or twiddling the highest bit.

Executables so linked will then be given a larger address space, either 3 or 4 gigabytes, depending on the operating system. You must edit boot.ini in Windows 2003 and earlier systems to enable this. Unfortunately, there is no native option in Classic VB to set this linker flag, so we must do it manually using the EditBin utility that comes with Visual Studio 6. By default, this would be installed here:

```
%programfiles%\Microsoft Visual Studio\VC98\Bin\editbin.exe
```

If that folder is on your system path, you can immediately expand the address space of your application (on operating systems that support it!) like this:

```
D:\Code\Samples\MemStatus>editbin /largeaddressaware memmax.exe
Microsoft (R) COFF Binary File Editor Version 6.00.8447
Copyright (C) Microsoft Corp 1992-1998. All rights reserved.

D:\Code\Samples\MemStatus>
```

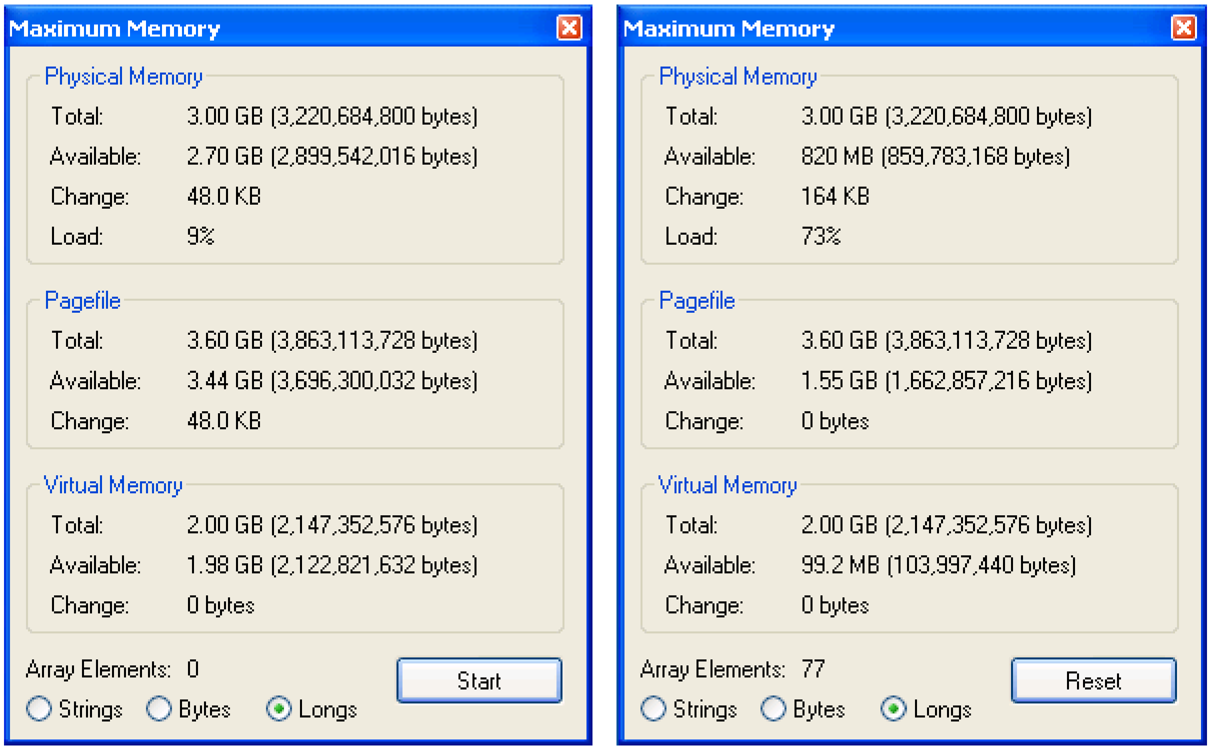
To confirm the results of throwing this switch, we can call the GlobalMemoryStatusEx API function. GlobalMemoryStatusEx returns a MEMORYSTATUSEX structure packed with useful information about the current memory situation of the machine you're running on and for your specific process:

```
Private Type MEMORYSTATUSEX
   dwLength As Long
   dwMemoryLoad As Long
   ullTotalPhys As Currency
   ullAvailPhys As Currency
   ullTotalPageFile As Currency
   ullAvailPageFile As Currency
   ullTotalVirtual As Currency
   ullAvailVirtual As Currency
   ullAvailExtendedVirtual As Currency
End Type
```

To call GlobalMemoryStatusEx, just create a new copy of the MEMORYSTATUSEX structure, and set its first member equal to its length. A return value other than zero indicates success:

```
   Dim ms As MEMORYSTATUSEX
   ms.dwLength = Len(ms)
   If GlobalMemoryStatusEx(ms) Then
      Debug.Print "Total Load: "; _
         Trim$(CStr(ms.dwMemoryLoad)); "%"
   End If
```

I've written up a drop-in ready class module that provides all the information offered by this API call, which you may want to download from my site. Also included is a little test application that refreshes the CMemStatus class every 100ms and displays the most current numbers on screen. Here's how it looks in XP:



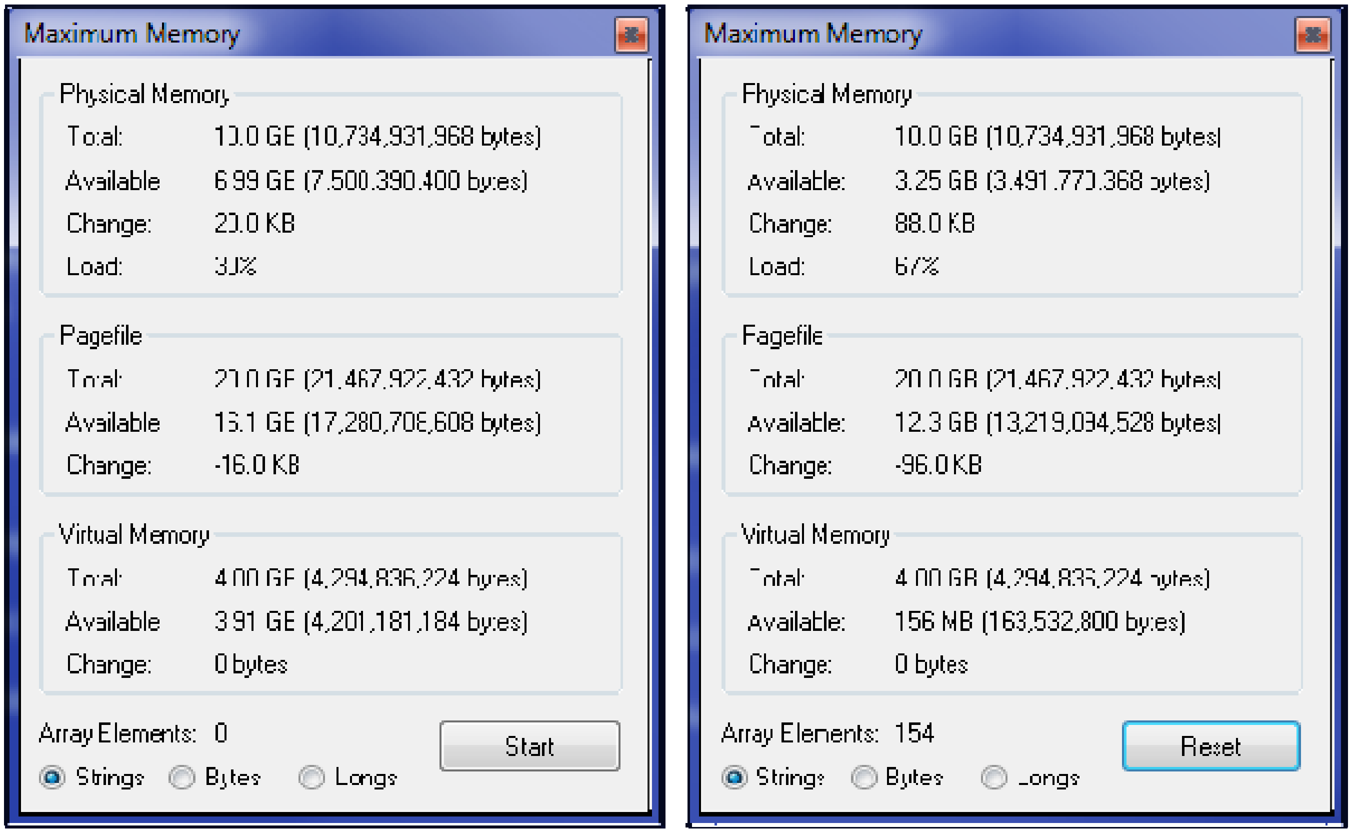
```
Maximum Memory

Physical Memory
 Total:       3.00 GB (3,220,684,800 bytes)
 Available:   2.70 GB (2,899,542,016 bytes)
 Change:      48.0 KB
 Load:        9%

Pagefile
 Total:       3.60 GB (3,863,113,728 bytes)
 Available:   3.44 GB (3,696,300,032 bytes)
 Change:      48.0 KB

Virtual Memory
 Total:       2.00 GB (2,147,352,576 bytes)
 Available:   1.98 GB (2,122,821,632 bytes)
 Change:      0 bytes

Array Elements:  0                    Start
Strings   Bytes   [✓ Longs]
```

```
Maximum Memory

Physical Memory
 Total:       3.00 GB (3,220,684,800 bytes)
 Available:   820 MB (859,783,168 bytes)
 Change:      164 KB
 Load:        73%

Pagefile
 Total:       3.60 GB (3,863,113,728 bytes)
 Available:   1.55 GB (1,662,857,216 bytes)
 Change:      0 bytes

Virtual Memory
 Total:       2.00 GB (2,147,352,576 bytes)
 Available:   99.2 MB (103,997,440 bytes)
 Change:      0 bytes

Array Elements:  77                   Reset
Strings   Bytes   [✓ Longs]
```

Figure 1.

The first picture shows what it the application displayed when it first started up. The second shows how it looks after I've allocated 77 arrays of Long integers, each of which consumed 25MB. You can see in the lower panel that my Virtual Memory has dropped from 1.98GB available to just 99.2MB. The top panel shows the total load on the system memory increasing from 9% to 73% over that same test run.

This little applet was edited to enable it to be large-address aware, though. Here's how it looks in Windows 7 x64:



Maximum Memory

Physical Memory
Total: 10.0 GB (10,734,931,968 bytes)
Available 6.99 GB (7,500,390,400 bytes)
Change: 20.0 KB
Load: 30%

Pagefile
Total: 20.0 GB (21,467,922,432 bytes)
Available 16.1 GB (17,280,708,608 bytes)
Change: -16.0 KB

Virtual Memory
Total: 4.00 GB (4,294,836,224 bytes)
Available 3.91 GB (4,201,181,184 bytes)
Change: 0 bytes

Array Elements: 0
[✓ Strings] Bytes Longs
Start

Maximum Memory

Physical Memory
Total: 10.0 GB (10,734,931,968 bytes)
Available: 3.25 GB (3,491,770,368 bytes)
Change: 88.0 KB
Load: 67%

Pagefile
Total: 20.0 GB (21,467,922,432 bytes)
Available: 12.3 GB (13,219,034,528 bytes)
Change: -96.0 KB

Virtual Memory
Total: 4.00 GB (4,294,836,224 bytes)
Available: 156 MB (163,532,800 bytes)
Change: 0 bytes

Array Elements: 154
[✓ Strings] Bytes Longs
Reset

Figure 2.

Notice that the system is now giving me 4GB of address space (Virtual Memory), of which 3.91GB are available! In this test, I allocated 154 String variables, each of which used up 25MB of space (13,107,200 characters).

At this point, all my tests to date would seem to indicate, you are free to use as much memory as the system is willing to give you. Of course, being as this isn't a native feature of the language, there's no way to say this will work out well in all situations.

You can use my CMemStatus class to guide your decisions about how much memory to allocate, taking care not to cause undo paging. *No one wants to endure an application that just grinds the disk to death because the coder couldn't come up with more elegant algorithms and data structures.* But no matter what, it won't hurt to rigorously apply tests of this nature:

```
   If Err.Number = 7 Then
      ' Out of memory – be graceful!
   End If
```

Let me know, either through my web site or leave a comment below, if you run into other limitations with 4-Gigabyte Tuning your apps.

About the Author

Karl E. Peterson wrote Q&A, Programming Techniques, and various other columns for VBPJ and VSM from 1995 onward, until Classic VB columns were dropped entirely in favor of other languages. Similarly, Karl was a Microsoft BASIC MVP from 1994 through 2005, until such community contributions were no longer deemed valuable. He is the author of VisualStudioMagazine.com's new Classic VB Corner column. You can contact him through his Web site if you'd like to suggest future topics for this column.