

Re: CObArray with huge number of elements

Source: <http://www.tech-archive.net/Archive/VC/microsoft.public.vc.mfc/2005-07/msg00145.html>

- *From:* Joseph M. Newcomer <newcomer@xxxxxxxxxxxxx>
 - *Date:* Sat, 02 Jul 2005 11:06:37 -0400
-

I should have run `nGrowby == -1`. Here's the data:

```
c:>c:\tests\add\unoptimized\add -1
Start: size=0, GrowBy=-1
250000 elements, deltaT = 0.341 sec, rate = 733138/sec
done adding items: size=250000
```

```
c:>c:\tests\add\release\add -1
Start: size=0, GrowBy=-1
250000 elements, deltaT = 0.326 sec, rate = 766871/sec
done adding items: size=250000
```

```
c:>c:\tests\add\debug\add -1
Start: size=0, GrowBy=-1
250000 elements, deltaT = 9.495 sec, rate = 26330/sec
done adding items: size=250000
```

joe

On Sat, 02 Jul 2005 02:09:59 -0400, Joseph M. Newcomer <newcomer@xxxxxxxxxxxxx> wrote:

```
>Actually, not true. You can read the code, or, as I did, single-step through the code to
>see that it does the obvious thing. Add calls SetAtGrow. SetAtGrow examines the size that
>has been allocated, and if it is within the current bounds, simply initializes the element
>to 0 and returns, allowing the assignment to take place without any additional allocation.
>I found that the first time I did an Add() (having done SetSize(0, 100000), it allocated
>one item (having taken the path that the array was empty and its data pointer was NULL).
>The next time, it added 100000 elements, and the third time it just set the element.
>
>However, he has not stated if he is doing this in the debug version, and it is worth
>examining the output of this little console app which was created as "Uses MFC", because I
>was using CDWordArray. The argv[1] is the nGrowBy value for SetSize.
>
>int _tmain(int argc, TCHAR* argv[], TCHAR* envp[])
> {
> int nRetCode = 0;
>
> // initialize MFC and print and error on failure
> if (!AfxWinInit(::GetModuleHandle(NULL), NULL, ::GetCommandLine(), 0))
> {
```

Re: CObArray with huge number of elements

```
> // TODO: change error code to suit your needs
> cerr << _T("Fatal Error: MFC initialization failed") << endl;
> return 1;
> }
>
> CDWordArray dw;
> int GrowBy = 100000;
> if(argc > 1)
>   GrowBy = atoi(argv[1]);
> dw.SetSize(0, GrowBy);
> printf("Start: size=%d, GrowBy=%d\n", dw.GetSize(), GrowBy);
> LARGE_INTEGER start;
> QueryPerformanceCounter(&start);
> int i;
> for(i = 0; i < 250000; i++)
>   dw.Add(i);
> LARGE_INTEGER stop;
> QueryPerformanceCounter(&stop);
>
> LARGE_INTEGER deltaT;
> deltaT.QuadPart = stop.QuadPart - start.QuadPart;
>
> LARGE_INTEGER frequency;
> QueryPerformanceFrequency(&frequency);
>
> LARGE_INTEGER ms;
> ms.QuadPart = (deltaT.QuadPart * 1000) / frequency.QuadPart;
> double seconds = (double)ms.QuadPart / 1000.0;
> printf("%d elements, deltaT = %8.3f sec, rate = %1.0f/sec\n", i, seconds, (double)(i
>/ seconds);
>
> printf("done adding items: size=%d\n", dw.GetSize());
>
> return nRetCode;
> }
```

>Note the following:
>In the debug mode, the larger GrowBy is (the input parameter), the slower it runs. For a
>simple loop, I get 26K/sec for 10K, 17K/sec for 100K, and 2693/sec for 1M. But now look
>at the data in the release version: for 10K, I get nearl 6M/sec, for 100K I get 12.5M/sec,
>and for 1M I get 15.5M/sec.

>
>I hand-edited the commas into the numbers so they were easier to read.

>
>Engineering means getting data to validate your decisions. Collecting performance data on
>a debug version clearly has nothing to do with reality, and therefore can be dismissed are
>irrelevant to solving the problem.

>
>Now, just to show the overheads are in the debug library support, and not in the code
>itself, I created a new build configuration, "Unoptimized", in which I built a release
>version with all optimizations disabled. Check out the numbers at the end.

Re: CObArray with huge number of elements

>
>It is important when solving problems to solve the RIGHT problem, and performance is not
>even something to consider in the debug version. Sure, it slows down the development, but
>it cannot in any way be considered relevant to what release performance is going to be. To
>determine release performance, the only valid way is to measure the release version.
>
>By the way, it makes a cool Excel spreadsheet, particularly if the y-axis is plotted
>logarithmically.
>
>c:\>c:\tests\add\debug\add 10000
>Start: size=0, GrowBy=10000
>250000 elements, deltaT = 9.385 sec, rate = 26,638/sec
>done adding items: size=250000
>
>c:\>c:\tests\add\debug\add 100000
>Start: size=0, GrowBy=100000
>250000 elements, deltaT = 14.090 sec, rate = 17,743/sec
>done adding items: size=250000
>
>c:\>c:\tests\add\debug\add 1000000
>Start: size=0, GrowBy=1000000
>250000 elements, deltaT = 92.834 sec, rate = 2,693/sec
>done adding items: size=250000
>
>c:\>c:\tests\add\release\add 10000
>Start: size=0, GrowBy=10000
>250000 elements, deltaT = 0.043 sec, rate = 5,813,953/sec
>done adding items: size=250000
>
>c:\>c:\tests\add\release\add 100000
>Start: size=0, GrowBy=100000
>250000 elements, deltaT = 0.020 sec, rate = 12,500,000/sec
>done adding items: size=250000
>
>c:\>c:\tests\add\release\add 1000000
>Start: size=0, GrowBy=1000000
>250000 elements, deltaT = 0.016 sec, rate = 15,625,000/sec
>done adding items: size=250000
>
>c:\>c:\tests\add\unoptimized\add 10000
>Start: size=0, GrowBy=10000
>250000 elements, deltaT = 0.045 sec, rate = 5,555,556/sec
>done adding items: size=250000
>96% of the rate of the optimized version
>
>c:\>c:\tests\add\unoptimized\add 100000
>Start: size=0, GrowBy=100000
>250000 elements, deltaT = 0.021 sec, rate = 11,904,762/sec
>done adding items: size=250000
>95% of the rate of the optimized version
>

Re: CObArray with huge number of elements

```
>c:>c:\tests\add\unoptimized\add 1000000
>Start: size=0, GrowBy=1000000
>250000 elements, deltaT = 0.019 sec, rate = 13,157,895/sec
>done adding items: size=250000
>84% of the rate of the optimized version
> joe
>
>On Fri, 01 Jul 2005 18:28:22 -0500, "Scott McPhillips [MVP]" <org-dot-mvps-at-scottmcp>
>wrote:
>
>>phil wrote:
>>
>>> thanks for the quick reply
>>>
>>> I've tried the following
>>>
>>> As a simple test I tried this...
>>> CObject *this = new MyObject();
>>> CObArray things;
>>> things.SetSize(0,1000000);
>>> for (long y=0;y<1000000;y++)
>>> things.Add(object); // just add the same one repeatedly
>>>
>>>
>>> while this does not grind to a halt it is VERY slow – about 1000 element per
>>> second
>>>
>>>
>>> this
>>> CObject *this = new MyObject();
>>> CObArray things;
>>> things.SetSize(1000000); // set the size before hand
>>> for (long y=0;y<1000000;y++)
>>> things[y]=object;
>>>
>>> is too fast to notice anything has happened –
>>>
>>> Is there some bad overhead with CObArray::Add ?
>>>
>>> thanks for the reply and any other ideas will be much appreciated
>>
>>That "too fast" version is the right answer. After you set the size you
>>don't want to use 'Add' any more. 'Add' makes the array size larger.
>>This seems to be stated pretty clearly in the docs for CObArray::Add
>>that I am looking at. If you haven't found the docs yet try clicking on
>>CobArray in your code, then hitting the F1 key.
>
>Joseph M. Newcomer [MVP]
>email: newcomer@xxxxxxxxxxxxx
>Web: http://www.flounder.com
>MVP Tips: http://www.flounder.com/mvp\_tips.htm
```

Re: CObArray with huge number of elements

Re: CObArray with huge number of elements

Joseph M. Newcomer [MVP]

email: newcomer@xxxxxxxxxxxxxx

Web: <http://www.flounder.com>

MVP Tips: http://www.flounder.com/mvp_tips.htm

• **References:**

◆ ***CObArray with huge number of elements***

◇ *From:* phil

◆ ***Re: CObArray with huge number of elements***

◇ *From:* Joseph M . Newcomer

◆ ***Re: CObArray with huge number of elements***

◇ *From:* phil

◆ ***Re: CObArray with huge number of elements***

◇ *From:* Scott McPhillips [MVP]

◆ ***Re: CObArray with huge number of elements***

◇ *From:* Joseph M . Newcomer

• Prev by Date: ***Re: How to switch the document within CMultiDocTemplate instance?***

• Next by Date: ***Re: Replacement for GetDesktopWindow() ?***

• Previous by thread: ***Re: CObArray with huge number of elements***

• Next by thread: ***Re: CObArray with huge number of elements***

• Index(es):

◆ ***Date***

◆ ***Thread***