

## Re: Memory limit reached with Windows Mobile

---

*Source:*

<http://www.tech-archive.net/Archive/PocketPC/microsoft.public.pocketpc.developer/2007-03/msg00022.html>

---

- *From:* "Patrick A." <[patrick.avenel@xxxxxxxxx](mailto:patrick.avenel@xxxxxxxxx)>
  - *Date:* 1 Mar 2007 10:19:19 -0800
- 

Chris,

So you're basically creating a copy of every class you may ever use at startup? That would explain the memory problem – you just can't do that. This is a memory-constrained device. You have to load stuff when it's needed and dump it when its not so the memory manager has an opportunity to actually "manage" memory.

I don't agree completely with the last part about "managing" memory.

There is supposed to be 25 Mb of available memory, not 12 Mb. Once I reached the 12 Mb limit, I cannot load any DLLs, yet if I "dumped" a few Mb of memory to get under 12 Mb, loaded my DLL, and loaded the "dumped" memory again, it would work. I think the memory manager could do something like that, and allocate the memory for the data smartly, ie not at the only place where DLLs can be loaded! Or at least it could move some data to another place if a DLL tries to load and it can't, although there is still memory available for data.

We have a big, maybe huge application (over 5 Mb of Native DLLs and 3.5 Mb of C# DLLs), which requires maybe 15 Mb of memory to run perfectly. That is not supposed to be a problem considering the available virtual memory (almost twice that value!). Yet it is, and it seems to be because the memory is ill-managed by Windows CE and the Compact Framework.

We're going to see what could be loaded/unloaded on demand, but as you stated, we need to create our modules on an as-needed basis, and that sometimes means a lot of them have to be loaded simultaneously, because they often interact...

On a side note, they are supposed to offer 2 Gb per process for Windows CE 6, but since we have 12 instead of 32 with Windows CE 4.2, I'm not sure what to expect...

Re: Memory limit reached with Windows Mobile

---

Patrick A.

On 27 fév, 18:19, "<ctacke/>" <ctacke[.]opennetcf[.]com> wrote:

So you're basically creating a copy of every class you may ever use at startup? That would explain the memory problem – you just can't do that. This is a memory–constrained device. You have to load stuff when it's needed and dump it when its not so the memory manager has an opportunity to actually "manage" memory.

I can only assume the decision to do what you've done is for performance, but (as you've found) it just isn't going to work. Use a progress bar or something to let the user know you're busy if you need to, but you need to create your modules on an as–needed basis.

---

Chris Tacke – Embedded MVP  
OpenNETCF Consulting  
Managed Code in the Embedded Worldwww.opennetcf.com

---

"Patrick A." <patrick.ave...@xxxxxxxx> wrote in message

[news:1172594881.575436.168920@xx](mailto:news:1172594881.575436.168920@xx)

Fragmentation may be to blame for the ability to create small objects on the heap but not map in a native DLL (depends on teh DLL size too). I think the key is going to be to reduce the GC Heap size on your app to prevent the situation from occurring in the first place. A 12MB heap seems pretty large. What kinds of objects are you creating and holding that would account for this size?

Actually, simply by loading our 18 Native DLLs, we reduce the Virtual Memory by 5.44 Mb.

At that point, we already have less than 7 Mb left before we won't be able to load any more Native DLLs.

We then initialize our 60 "modules" using Reflection each time, and we store the 60 classes obtained in a Hashtable.

Every module also loads the critical classes and resources it will need later, open the connections to its databases, loads and display its icon on our main page.

Launching the application takes approximately 30 seconds.

Each module or class taken individually doesn't seem to use much space

Re: Memory limit reached with Windows Mobile

(a few kb here, a few kb there), but once it's all done, we're down a few other megabytes.

When a few screens are loaded, a few cards read, a few tickets printed, etc. we reach the 12 Mb limit, and no new Native DLL can be loaded.

Independently from the size of my objects, the simple fact that there is still 15 Mb of Virtual Memory unused and it's already impossible to load new DLLs is still a mystery to me.

Why can I load my X DLLs then Y Mb of data, and it's not possible to load a single DLL after loading my Y Mb of data?

It's very frustrating.

—  
Patrick AVENEL

On 27 fév, 16:29, "<tacke/>" <tacke[ @]opennetcf[dot]com> wrote:

Fragmentation may be to blame for the ability to create small objects on the heap but not map in a native DLL (depends on the DLL size too). I think the key is going to be to reduce the GC Heap size on your app to prevent the situation from occurring in the first place. A 12MB heap seems pretty large. What kinds of objects are you creating and holding that would account for this size?

—  
Chris Tacke – Embedded MVP  
OpenNETCF Consulting  
Managed Code in the Embedded Worldwww.opennetcf.com  
—

"Patrick A." <patrick.ave...@xxxxxxxx> wrote in message

[news:1172586923.041400.307630@xx](mailto:news:1172586923.041400.307630@xx)  
Chris,

## Re: Memory limit reached with Windows Mobile

Ok, that's a start, but it's only telling you physical memory.  
Take a  
look  
at available virtual, I think it will be more useful here (it tells  
you  
virtual space available in your slot).

I just run my sample/test application checking AvailableVirtual  
instead.

Before loading my DLLs, it returns 28,180,480.  
I then load my DLLs until I get a MissingMethodException trying to  
load a DLL.  
It then returns 15,335,424.

It apparently used 12,85 Mb of Virtual Memory, and although there is  
still 15,35 Mb left, it's not possible to load anymore DLLs.  
Nevertheless, I can still bring down that AvailableVirtual down to  
under 1 Mb by loading DateTime arrays.

To complete the test, I take down the AvailableVirtual to 13 Mb left  
by loading DateTime arrays, and then try to load Native DLLs.  
I get a MissingMethodException on the very first try.

It would seem that both are allocated in the same virtual memory space  
then?  
Yet Native DLLs can only be loaded in the "first 12,85 Mb of that  
virtual memory"?

In our "real" application, it seems that at some point, we have used  
over 12,85 Mb of Virtual Memory, and that we can't load any Native  
DLLs after that.  
Simply by changing the order in which our application is loaded  
(Native DLLs first, then the rest of the application), we managed to  
fix 99% of the application.  
For some reason though, "something" needs to be loaded when an HTTPS  
connection is established, and we can't figure out what or where.

Re: Memory limit reached with Windows Mobile

Regards,

--  
Patrick

On 27 fév, 14:27, "<ctacke/>" <ctacke[@]opennetcf[dot]com> wrote:

I do feel there is a misunderstanding about our sample/test application, which is only meant to test the limits and stress the impact between loading Native DLLs and loading "C# stuff". Your help is greatly appreciated though!

<ctacke>  
No, I think we both understand fully what you're doing. What we're trying to get at is the "why" of what you're doing. After all running a test just for the sake of running it isn't useful. You need to know exactly what you're testing.  
</ctacke>

I don't know for sure where everything is loaded (shared, process space, ROM XIP space..) since the compact framework is doing that by himself.

<ctacke>  
And therein lies the problem. Once you can determine the load on the process slot and shared memory, we can figure out what it's running out of and then treat the "why".

## Re: Memory limit reached with Windows Mobile

</ctacke>

I use  
OpenNetCf.Win32.Core.GlobalMemoryStatus().AvailablePhysical  
to  
get an idea of the RAM our application used (by comparing  
the value  
just before and just after).

<ctacke>

Ok, that's a start, but it's only telling you physical memory.  
Take a  
look  
at available virtual, I think it will be more useful here (it tells  
you  
virtual space available in your slot).  
</ctacke>

When you load the interface  
classes, you're not walking  
each class  
and  
its  
methods and properties via  
reflection are you? If so  
that's going  
to  
take  
up some serious space.

That's interesting. We use Reflection to call a given method  
from a  
Factory class in our modules. This method returns a class,  
which  
implements a known interface, and we store this class in a  
Hashtable  
for future use. We then call the methods of this class through  
the  
Interface. Could that be eating up space? Is there a better  
pattern?

## Re: Memory limit reached with Windows Mobile

<ctacke>

Not necessarily anything better. Using reflection is fine – but when

you

load via reflection, a single copy of every item gets loaded into the

AppDomain heap. Inspecting an assembly by walking all of its classes and

methods (like if you were doing something like Reflector) will generate

a

lot of these objects. The AppDomain heap can never be shrunk, so if you

load a lot up, you've killed a lot of your process space until the app

exits.

</ctacke>

Entrek is great for native apps, but of no value for managed apps.

I'd

look at the CF 1.0 perf counters first, and then watch the app with

RPM

and see what's up. Do a web search for RPM or Remote Performance

Monitor

with the CF. It's likely to show you pretty quickly how things are

going

bad (unfortunately its granularity doesn't yet show the "where" so

that'll

take some detective work).

I tried to use the mscoree.stat file by adding the registry entry, but the file ended up empty..

Re: Memory limit reached with Windows Mobile

<ctacke>

It never gets filled until the app actually exits.

</ctacke>

Do you know any good performance monitor for the CF 1.0?

<ctacke>

Unfortunately no, there isn't one. Running against CF 2.0 and using RPM

is

your shortest route to really understanding what's going on here.

</ctacke>

I'll run our application on the CF 2.0 as soon as possible, but that probably won't be this week...

Thanks again,  
Regards,

—

Patrick AVENEL

On 27 fév, 09:07, "The PocketTV Team"

<supp...@xxxxxxxxxxxxxx> wrote:

i completely understand Patrick's points.

apparently ctacke does not understand what a test is and also that real-life applications sometimes need many dll's and lots of memory.

Re: Memory limit reached with Windows Mobile

Patrick, i hope you can find someone who understand what you are explaining (i understand all very well), and who knows the answer to your questions (that, i don't know, but i undersatnd what you are trying to test, and why).

ctacke: re-read Patrick's questions, maybe you will understand better. he explained all quite well, and i understand his point, that apparently you don't get.

"<ctacke/>" <ctacke[ @ ]opennetcf[dot]com> wrote in message

[news:ui%23lftdWHHA.1636@xxxxxxxxxxxxxxxxxxxxxxxxxxxxx](mailto:news:ui%23lftdWHHA.1636@xxxxxxxxxxxxxxxxxxxxxxxxxxxxx)

This application can either :  
1- Load the same DLL (copied with a different name) over and over.  
2- Fill an ArrayList with DateTime[1000] while checking free RAM on the device.

<ctacke>  
Checking RAM by what mechanism?  
<ctacke>

## Re: Memory limit reached with Windows Mobile

We noticed that :  
– after 60 DLLs loaded with  
(1), we get a  
MissingMethodException.

<ctacke>  
Sure. Loading native DLLs  
takes at least 64k each. 60  
DLLs would  
be  
~3.8MB of virtual space.  
How is this valid? Does  
your app actually  
need  
60+ native DLLs?  
</ctacke>

– after 25 Mb used with (2),  
we get an OOMException.

<ctacke>  
Again, exactly what is this  
testing? I'm not sure exactly  
how big a  
DateTime is (your number  
or 1k items indicates 25k

...

plus de détails »– Masquer le texte des messages précédents –

– Afficher le texte des messages précédents –