

# Re: High Memory Consumption of Classes and Arrays

---

*Source:*

<http://www.tech-archive.net/Archive/DotNet/microsoft.public.dotnet.framework.performance/2005-05/msg00006.htm>

---

- *From:* "Frank Hileman" <[frankhil@xx](mailto:frankhil@xx)>
  - *Date:* Sun, 1 May 2005 05:13:06 -0700
- 

Rüdiger Klaehn is correct; the most efficient way to use memory in .NET is to use arrays of structs. If you don't need polymorphism on your "objects" this works very well: there are 0 bytes of overhead per struct in an array. Only the array itself has overhead.

If you have too many arrays you might have to think of pooling mechanisms such as the struct linked list. Linked lists do have overhead though, even if each node is a struct in an array. The least amount of overhead is to store the representation of each "list" in your object as an index and a count into a common contiguous array of structs. This makes insertion wasteful (all higher indices invalidated), so you might only use this for a read-only list. There are hybrid list/array structures like a deque that have some of the savings of an array and flexibility of a list.

It could be that your data structures are too fine-grained and you need a whole different algorithm.

ArrayList is a bit wasteful if you examine it in Reflector. You should never put a struct into an ArrayList either; that boxes it and takes just as much memory as a reference type (class).

Regards,  
Frank Hileman

check out VG.net: <http://www.vgdotnet.com>  
Animated vector graphics system  
Integrated Visual Studio .NET graphics editor

"Christian Rattat" <[groupanswers@xxxxxxxxxxx](mailto:groupanswers@xxxxxxxxxxx)> wrote in message  
[news:ewpApJjTFHA.548@xxxxxxxxxxxxxxxxxxxxxxxx](mailto:news:ewpApJjTFHA.548@xxxxxxxxxxxxxxxxxxxxxxxx)  
>>(note that all numbers from here assume you are using a >32bit  
> processor, if  
>>you are using a 64 bit proc and runtime then references >and probably  
> the  
>>object header will have to increase).  
>

## Re: High Memory Consumption of Classes and Arrays

> I have a 32 bit architecture.  
>  
>>I'm pretty sure it should be 8(which is the size of the >class header,  
> 12 if  
>>you consider the reference as well). How are you >determining the size  
> of a  
>>given object?  
>  
> I use the GC GetTotalMemory in a loop where I allocate my objects. To  
> prevent side-effects I do this very often and you can see that after  
> each object created the free memory decreased by (in my example) 52  
> bytes. If I delete the objects the other way round I can measure that 52  
> bytes are freed again. I don't say this is the size of the object, but  
> this is the size the runtime allocates for each object.  
>  
>>Then you were mistaken. First off, why would you assume >that two  
> collections  
>>are only the size of their reference? While the data >contained in  
> \*your\*  
>>object will only be 4 bytes, the ArrayList class's fields >take up...16  
>>bytes. These bytes are taken up by  
>  
> I have never assumed that a collection is of the ref size only, haven't  
> I? As the collection that I use (ArrayList) does nothing else than  
> manging an array. I would have assumed that it just has the size of the  
> objects stored in an array plus maybe 4-8 bytes for management  
> information.  
>  
>  
> ..  
>  
>>And assuming that the way C++ does things is the way >everyone does  
> this will  
>>be your undoing. Learn the framework and its >idiosynchrosies instead  
> of  
>>trying to use it like another language.  
>  
> First off, I'm an 12 years experienced software developer I pretty well  
> know the dotnet platform but also c++, java, perl and other programming  
> languages. A have participated in very huge software projects with  
> extreme high performance requirements (Sun Enterprise 10000 clusters  
> with 32 processors to manage several millions of customer request per  
> day). So I think I don't need to discuss on that level...  
>  
> ..  
>  
>>Because there is no way to accuratly determine the size >of a managed  
> type in  
>>managed memory. The runtime is pretty much free to layout >a class in  
> any  
>>order it wants(unless you tell it otherwise, and even >then I don't

## Re: High Memory Consumption of Classes and Arrays

> think tis  
>>required to for reference types) or even theoretically >adjust the  
> space  
>>taken up by a variable if it can statically determine it >safe,  
> therefore the  
>>size of an object can vary between one runtime and >another or  
> potentially  
>>even on different runs(or maybe even the same instance, >if a garbage  
> collect  
>>happens between the two sizeof calls).  
>  
> If you read the text above, you have given the answer, that there is  
> obviously a mechanism that calculates the size. The clr must know at a  
> specific point how much memory is required to place an object into  
> memory. So, how could any memory be allocated if there is no information  
> about the size of that object? Impossible, right? So finally, why isn't  
> that information available to the user?  
> My answer: poor design  
>  
>  
>  
>>Why would you say a sealed class wouldn't need >polymorphic stuff? Can  
> a  
>>sealed class not override ToString or other virtual >methods it  
> inherited?  
>>Does being sealed mean the class cannot be refered to by >a variable  
> with a  
>>base type? This statement makes no sense at all. If >sealed classes  
> don't  
>>need a vtable pointer then what would the following do?(Mind you,  
> string is  
>>sealed)  
>  
> Well, this is true. Must have been my anger that had blocked my brain  
> ..  
>  
> ..  
>  
>>Without knowing your architecture, I certainly can't >offer much  
> advice. Have  
>>you tried using structures instead of classes?  
>  
> Sure. Structures are of same size as classes. I also tried to  
> explicetely layout the class and structures, pack it (using different  
> sizes 1,2,4) and so on.  
>  
>  
> Finally, none of my problems has been solved. Fact is:  
> a) instances of  
>  
> public class MyClass

## Re: High Memory Consumption of Classes and Arrays

> {  
> }  
>  
> which have no fields, methods (only the default ctor)  
>  
> consume 24 bytes. You can simply check it.  
>  
> b) The code  
>  
> object[] o = new object[0];  
>  
> consumes 16 bytes. Check it, too.  
>  
> c) Due to a) and b) any class that manages a list-like structure will at  
> least consume 40 bytes of memory.  
>  
>  
> As my structure requires to have 2 of this lists each element of this  
> structure will at least consume  
>  
>  $2 \times 40 \text{ bytes} + 24 \text{ bytes for the containing class itself} = 104 \text{ bytes.}$   
>  
> Now 104 bytes for each node means for a million nodes 104000000 bytes  
> and still none of my data is included.  
> In other words this structure will need around onehundred megabyte  
> memory just to manage the structure. By using the ArrayList class memory  
> consumption increases more again.  
>  
> This really sucks. I think I'm very deep into the dotnet paradigm and  
> know the clr/cls and most of the concepts of dotnet in-deep as well.  
>  
> So, what do you want to tell me? Do you have any ideas how I will be  
> able to significantly decrease the size of the objects instead? From my  
> current understanding, there is no way. So what's left to say: The  
> concepts of the dotnet platform regarding object memory storage seem to  
> be poor designed. There is obviously no way to manage large collections  
> of small objects in memory in any application based on the dotnet  
> platform. In my case this means concretely that managing a million  
> objects with actually 24 million bytes of data consume around 480  
> megabytes of memory.  
>  
> If not this is poor design then what would you call poor design?  
>  
>  
>  
>  
>  
>  
>  
>  
>  
>  
>

