

Factsheet



Andrew Hunter is often described as the Master of all that is deep and dark in the .NET Framework. He hand-crafted the engine for the new ANTS Memory Profiler 5 and, last year, he worked on the ANTS Performance Profiler 4 project.

He's a quiet kind of guy but when he opens his mouth, it's always to say something insightful and intelligent. We therefore thought we would get him to set you a little challenge with a .NET Memory Management Quiz. Here are his detailed answers to the quiz.

Detailed answers to Andrew's .NET Memory Management Quiz

1. .NET developers should explicitly call GC.Collect to improve their application's performance.

Answer: False

The GC.Collect method causes a full collection of all objects. This is expensive and takes a significant amount of time, as every live object must be visited and checked. The .NET garbage collector algorithm is finely tuned so that it does full collections only when it is worth the expense of doing so. You almost never need to call GC.Collect directly: the CLR is almost always much better at determining the best time to start removing objects, and it's not usually harmful to have a few unused objects left around. .NET will remove them before they become a problem.

If .NET does not appear to be reclaiming memory, the culprit is much more likely to be issues with fragmentation of the large object heap than the garbage collector failing to run. In server applications such as ASP.NET, .NET may also have higher limits configured for the heaps so it can perform garbage collections less frequently.

Forcing a full collection will force some short-lived objects into a higher generation, which can actually result in the program having an increased memory footprint until the standard behaviour removes these objects again. Additionally, a full collection requires the CLR to visit all objects allocated by the program, which can force the operating system to page heavily, resulting in seriously degraded performance.

2. A garbage collection removes all objects that are no longer referenced by the program.

Answer: False

The answer is 'false' but this is a bit of a trick question. A single garbage collection won't remove everything. Firstly, the .NET garbage collector is generational and most collections won't check old objects that have previously survived. Secondly, some objects have finalizers; when these objects become unreferenced they are moved to the finalizer queue the first time they are collected, and they need to be collected again before their memory is reclaimed.

redgate®

ingeniously simple tools

Factsheet

3. Which of these items is not a GC root?

- A static variable
- A local variable
- A class field
- A method parameter

Answer: a class field

Class fields will be garbage collected once all references to the object that contains them are eliminated. All the others are GC roots and any objects put in these locations will not be removed by the garbage collector.

Local variables are GC roots while a method is on the stack, and method parameters are just another form of local variable. Static variables are actually stored in an array that forms the 'real' root but, as far as the developer is concerned, they act exactly like roots.

4. The memory used by an object is released when Dispose() is called.

Answer: False

Calling Dispose() will not immediately release any managed objects referenced by the disposed object, so it will still be using memory. For most disposable objects, some memory will be released (whatever was used by the unmanaged resources the object was managing) but the only way to free all of the memory used by the object is to make sure that nothing references it any more, and wait for a garbage collection. If the Dispose() method does not call GC.SuppressFinalize, the memory might not be freed until at least two garbage collections have been performed.

5. Allocating memory on the managed heap in .NET applications is extremely fast.

Answer: True

Memory allocation is extremely quick because it involves only a simple pointer relocation to create space for the new object. The managed heap can be thought of as a block of contiguous memory. When you create a new object, the object's memory is allocated at the next available location on the managed heap.

6. Objects larger than 85KB are never moved by the garbage collector.

Answer: True

Objects larger than 85KB are allocated on the Large Object Heap. This is still garbage collected, so unreachable objects are freed, but it is never compacted. This can lead to fragmentation of the Large Object Heap, which can cause OutOfMemoryExceptions to be thrown because there is no single free block large enough to accommodate new allocations. .NET can allocate some smaller objects on the large object heap in rare circumstances if it needs to, but that 85KB is the limit at which objects will always end up there.

Factsheet

About Andrew Hunter's recent work and his recommendations

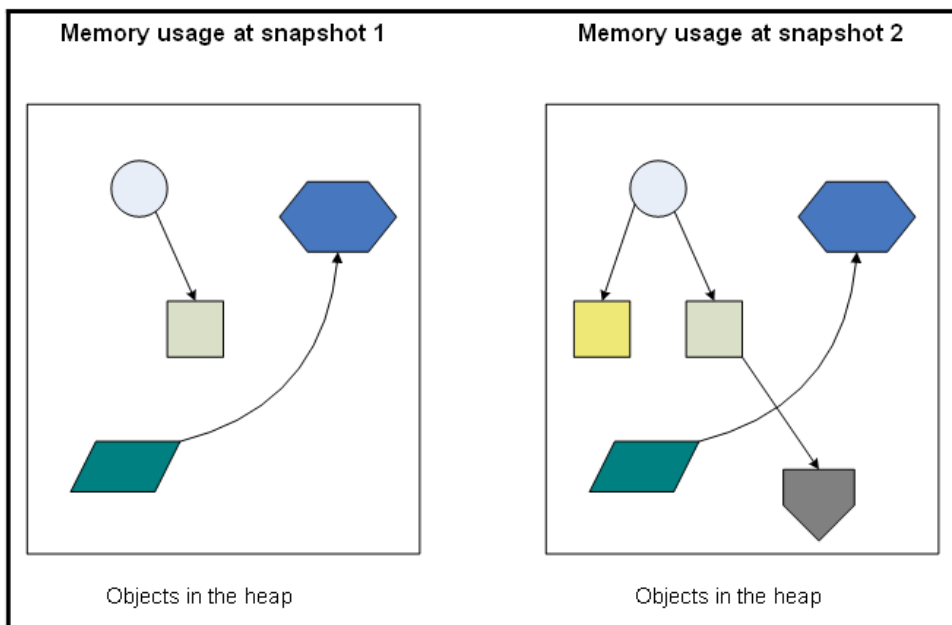
Andrew wrote the engine for the new ANTS Memory Profiler 5 from Red Gate, and he recommends you **use ANTS Memory Profiler™ if your application:**

- **Is displaying signs of high memory usage** – you suspect that your program is holding on to too much memory.
- **Needs regular restarts** because of degrading performance over time – application recovers on restart, but degrades again later on.
- **Needs a quality-control check** before a release or at an important milestone during a development cycle.

Automatic memory management in .NET makes development a lot easier; however, it's still easy to introduce memory leaks into your application. For example, in a complex application, it's easy to forget to unregister event handlers, and these are notorious for holding on to objects which you don't need to keep in memory any more. This typically leads to an increase in memory usage, which can lead to your application exhibiting poor performance, or even running out of memory and crashing, if it remains unchecked and unresolved. This is where a memory profiler becomes necessary.

How does ANTS Memory Profiler work?

With ANTS Memory Profiler, you can take as many snapshots as you want while your application is running, so you can compare application memory states.



You can download ANTS Memory Profiler from: http://www.red-gate.com/products/ants_memory_profiler/index.htm

Red Gate Software

Newnham House, Cambridge Business Park, Cambridge CB4 0WZ, UK
tel: +44 (0) 870 160 0037 toll-free: 1 866 RED GATE (733 4283)
email: sales@red-gate.com web: www.red-gate.com

redgate®

ingeniously simple tools