Garbage Collection in .NET - A deeper look for the beginners

By [**Gerald Leslie Jones**](http://www.codeproject.com/script/Membership/View.aspx?mid=661498), 3 Nov 2003

Top of Form

Introduction

Garbage collection is a process of releasing the memory used by the objects, which are no longer referenced. This is done in  different ways and different manners in various platforms and languages. We will see how garbage collection is being done in .NET.

Garbage Collection basis

* Almost every program uses resources such as database connection, file system objects etc. In order to make use of these things some resources should be available to us.
* First we allocate a block of memory in the managed memory by using the new keyword. (This will emit thenewobj instruction in the Microsoft intermediate language code generated from C#, VB.NET, Jscript.NET or any other .NET compliance language).
* Use the constructor of the class to set the initial state of the object.
* Use the resources by accessing the type’s members.
* At last CLEAR THE MEMORY.

When we look at these steps, it seems to be a very simple process to do with. But how many of us do that without forgetting to release the memory block. C++ programmers will agree with me. C++ has got a special member function called Destructor. It has the same name of the constructor or the class with the ‘~’ (tilde) symbol to start with. This is a special kind of function which will be called every time by the system when ever the system finds that object will not be used any more by the program. (When the scope and lifetime of the object goes off).

But how many times have programmers forgotten to release the memory. Or how many times the programmers try to access the memory which was cleaned.

These two are the serious bugs, which will lead us to memory leak and commonly occurring. In order to overcome these things the concept of automatic memory management has come. Automatic memory management or Automatic garbage collection is a process by which the system will automatically take care of the memory used by unwanted objects (we call them as garbage) to be released. Hurrah….. Thanks to Microsoft's Automatic Garbage collection mechanism.

Automatic Garbage Collection in .NET

When Microsoft planned to go for a new generation platform called .NET with the new generation language called C#, their first intention is to make a language which is developer friendly to learn and use it with having rich set of APIs to support end users as well. So they put a great thought in Garbage Collection and come out with this model of automatic garbage collection in .NET.

They implemented garbage collector as a separate thread. This thread will be running always at the back end. Some of us may think, running a separate thread will make extra overhead. Yes. It is right. That is why the garbage collector thread is given the lowest priority. But when system finds there is no space in the managed heap (managed heap is nothing but a bunch of memory allocated for the program at run time), then garbage collector thread will be given REALTIME priority (REALTIME priority is the highest priority in Windows) and collect all the un wanted objects.

How does Garbage collector locate Garbage

When an program is loaded in the memory there will be a bunch of memory allocated for that particular program alone and loaded with the memory. This bunch of memory is called Managed Heap in .NET world. This amount of memory will only be used when an object is to be loaded in to the memory for that particular program.

This memory is separated in to three parts.

* Generation Zero.
* Generation One and
* Generation Two.

|  |  |  |
| --- | --- | --- |
| Generation Zero | Generation One | Generation Two |

Figure 1.1 Managed Heap Structure.

Ideally Generation zero will be in smaller size, Generation one will be in medium size and Generation two will be larger.

When we try to create an object by using NEW keyword in the high level languages. It will simply emit newobj in to the MSIL file. (newobj is a Microsoft Intermediate Language instruction to create a new type). When newobjexecutes, the system will,

* Calculate the number of bytes required for the object or type to be loaded in to the managed heap.
* Add the bytes required for an object’s overhead. Each object has two overhead fields: a method table pointer and a SyncBlockIndex. On a 32-bit system, each of these fields requires 32 bits, adding 8 bytes to each object. On a 64-bit system, each is 64 bits, adding 16 bytes to each object.
* The CLR then checks that the bytes required to allocate the object are available in the reserved region (committing storage if necessary). IF the object fits, it is allocated at the address pointed to by NextObjPtr. The type’s constructor is called (passing NextObjPtr) for the this parameter), and the newobj MSIL instruction (or the new operator) returns the address of the object. Just before the address is returned,NextObjPtr is advanced past the object and indicates the address where the next object will be placed in the heap.
* These processes will happen at the Generation zero level.

|  |  |  |  |
| --- | --- | --- | --- |
| A | B | C |  |

Figure 1.2 Allocating objects in the Managed Heap

When Generation Zero is full and it does not have enough space to occupy other objects but still the program wants to allocate some more memory for some other objects, then the garbage collector will be given the REALTIME priority and will come in to picture.

Now the garbage collector will come and check all the objects in the Generation Zero level. If an object’s scope and lifetime goes off then the system will automatically mark it for garbage collection.

Note:

Here in the process the object is just marked and not collected. Garbage collector will only collect the object and free the memory.

Garbage collector will come and start examining all the objects in the level Generation Zero right from the beginning. If it finds any object marked for garbage collection, it will simply remove those objects from the memory.

Here comes the important part. Now let us refer the figure 1.2 above. There are three objects in the managed heap. If A and C are not marked but B has lost it scope and lifetime. So B should be marked for garbage collection. So object B will be collected and the managed heap will look like this.

|  |  |  |  |
| --- | --- | --- | --- |
| A |  | C |  |

Figure 1.3 Memory Structure after Sweep

But do remember that the system will come and allocate the new objects only at the last. It does not see in between. So it is the job of garbage collector to compact the memory structure after collecting the objects. It does that also. So the memory would be looking like as shown below now.

|  |  |  |
| --- | --- | --- |
| A | C |  |

Figure 1.4 Memory Structure after Compact

But garbage collector does not come to end after doing this. It will look which are all the objects survive after the sweep (collection). Those objects will be moved to Generation One and now the Generation Zero is empty for filling new objects.

If Generation One does not have space for objects from Generation Zero, then the process happened in Generation Zero will happen in Generation one as well. This is the same case with Generation Two also.

You may have a doubt, all the generations are filled with the referred objects and still system or our program wants to allocate some objects, then what will happen? If so, then the MemoryOutofRangeException will be thrown.

Bottom of Form