

1 System.GC Class

```
2 [ILAsm]  
3 .class public sealed GC extends System.Object  
4 [C#]  
5 public sealed class GC
```

6 Assembly Info:

- 7 • *Name*: mscorlib
- 8 • *Public Key*: [00 00 00 00 00 00 00 00 04 00 00 00 00 00 00]
- 9 • *Version*: 2.0.x.x
- 10 • *Attributes*:
 - 11 ○ CLSCompliantAttribute(true)

12 Summary

13 Provides a mechanism for programmatic control of the garbage collector.

14 Inherits From: System.Object

15

16 **Library:** BCL

17

18 **Thread Safety:** All public static members of this type are safe for multithreaded operations.
19 No instance members are guaranteed to be thread safe.

20

21 Description

22 [*Note*: The *garbagecollector* is responsible for tracking and reclaiming objects allocated
23 in managed memory. Periodically, the garbage collector performs a garbage collection to
24 reclaim memory allocated to objects for which there are no valid references. Garbage
25 collections happen automatically when a request for memory cannot be satisfied using
26 available free memory.

27

28 A garbage collection consists of the following steps:

- 29 1. The garbage collector searches for managed objects that are referenced in managed
30 code.
- 31 2. The garbage collector attempts to finalize unreferenced objects.
- 32 3. The garbage collector frees unreferenced objects and reclaims their memory.

33 During a collection, the garbage collector will not free an object if it finds one or more
34 references to the object in managed code. However, the garbage collector does not
35 recognize references to an object from unmanaged code, and can free objects that are
36 being used exclusively in unmanaged code unless explicitly prevented from doing so. The
37 `System.GC.KeepAlive` method provides a mechanism that prevents the garbage collector
38 from collecting objects that are still in use in unmanaged code.

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30

Implementations of the garbage collector should track the following information:

- Memory allocated to objects that are still in use
- Memory allocated to objects that are no longer in use
- Objects that require finalization prior to being freed

Other than managed memory allocations, implementations of the garbage collector should not maintain information about resources held by an object, such as file handles or database connections. When a type uses unmanaged resources that must be released before instances of the type are reclaimed, the type should implement a *finalizer*. In most cases, finalizers are implemented by overriding the `System.Object.Finalize` method; however, types written in C# or C++ implement destructors, which compilers turn into an override of `System.Object.Finalize`.

In most cases, if an object has a finalizer, the garbage collector calls it prior to freeing the object. However, the garbage collector is not required to call finalizers in all situations. Also, the garbage collector is not required to use a specific thread to finalize objects, or guarantee the order in which finalizers are called for objects that reference each other but are otherwise available for garbage collection.

In scenarios where resources must be released at a specific time, classes should implement the `System.IDisposable` interface, which contains a single method (`System.IDisposable.Dispose`) that is used to perform resource management and cleanup tasks. Classes that implement `System.IDisposable.Dispose` must specify, as part of their class contract, if and when class consumers call the method to clean up the object. The garbage collector does not, by default, call the `System.IDisposable.Dispose` method; however, implementations of the `System.IDisposable.Dispose` method can call methods in the `System.GC` class to customize the finalization behavior of the garbage collector.

]

1 GC.Collect() Method

```
2 [ILAsm]  
3 .method public hidebysig static void Collect()  
4 [C#]  
5 public static void Collect()
```

6 Summary

7 Requests that the garbage collector reclaim memory allocated to objects for which there
8 are no valid references.

9 Description

10 A call to this method is only a suggestion; such a call does not guarantee that any
11 inaccessible memory is, in fact, reclaimed.

12

1 GC.KeepAlive(System.Object) Method

```
2 [ILAsm]  
3 .method public hidebysig static void KeepAlive(object obj)  
4 [C#]  
5 public static void KeepAlive(object obj)
```

6 Summary

7 Creates a reference to the specified object.

8 Parameters

Parameter	Description
<i>obj</i>	A System.Object that is not to be reclaimed by the garbage collector.

9 10 Description

11 The purpose of the System.GC.KeepAlive method is to ensure the existence of a
12 reference to an object that is at risk of being reclaimed by the garbage collector
13 prematurely.

14
15 The System.GC.KeepAlive method performs no operations and does not produce any
16 side effects.

17
18 This method is required to be implemented in such a way as to prevent optimizing tools
19 from omitting the method call from the executable code.

20
21 During program execution, after the call to the System.GC.KeepAlive method is
22 executed, if there are no additional references to *obj* in managed code or data, *obj* is
23 eligible for garbage collection.

24

1 GC.ReRegisterForFinalize(System.Object)

2 Method

```
3 [ILAsm]  
4 .method public hidebysig static void ReRegisterForFinalize(object obj)  
5 [C#]  
6 public static void ReRegisterForFinalize(object obj)
```

7 Summary

8 Requests that the specified object be added to the list of objects that require
9 finalization.

10 Parameters

Parameter	Description
<i>obj</i>	The <code>System.Object</code> to add to the set of objects that require finalization.

11 Description

13 The `System.GC.ReRegisterForFinalize` method adds *obj* to the list of objects that
14 request finalization before the garbage collector frees the object. *obj* is required to be
15 the caller of this method.

16
17 Calling the `System.GC.ReRegisterForFinalize` method does not guarantee that the
18 garbage collector will call an object's finalizer. [*Note:* For more information, see
19 `System.Object.Finalize`.]

20
21
22
23 [*Note:* By default, all objects that implement finalizers are added to the list of objects
24 that require finalization; however, an object might have already been finalized, or might
25 have disabled finalization by calling the `System.GC.SuppressFinalize` method.]
26
27

28 Exceptions

Exception	Condition
<code>System.ArgumentNullException</code>	<i>obj</i> is a null reference.

29
30

1 GC.SuppressFinalize(System.Object) Method

```
2 [ILAsm]  
3 .method public hidebysig static void SuppressFinalize(object obj)  
4 [C#]  
5 public static void SuppressFinalize(object obj)
```

6 Summary

7 Instructs the garbage collector not to call the `System.Object.Finalize` method on the
8 specified object.

9 Parameters

Parameter	Description
<i>obj</i>	The <code>System.Object</code> whose <code>System.Object.Finalize</code> method will not be called.

10 11 Description

12 The method removes *obj* from the set of objects that require finalization. *obj* is required
13 to be the caller of this method.

14
15 [Note: Objects that implement the `System.IDisposable` interface should call this
16 method from the `System.IDisposable.Dispose` method to prevent the garbage
17 collector from calling `System.Object.Finalize` on an object that does not require it.]
18
19

20 Exceptions

Exception	Condition
<code>System.ArgumentNullException</code>	<i>obj</i> is a null reference.

21

22

1 GC.WaitForPendingFinalizers() Method

```
2 [ILAsm]  
3 .method public hidebysig static void WaitForPendingFinalizers()  
4 [C#]  
5 public static void WaitForPendingFinalizers()
```

6 Summary

7 Suspends the current thread until the set of objects waiting for finalization is empty.

8 Description

9 `System.GC.WaitForPendingFinalizers` blocks an application until all objects that are
10 awaiting finalization have been finalized.

11
12 When the garbage collector finds objects that can be reclaimed, it checks each object to
13 determine the object's finalization requirements. If an object implements a finalizer and
14 has not disabled finalization by calling `System.GC.SuppressFinalize`, the object is
15 passed to the thread that handles finalization. The
16 `System.GC.WaitForPendingFinalizers` method blocks until all finalizers have run to
17 completion.

18