GC.SuppressFinalize(Object) Method

Namespace: System

Assembly: System.Runtime.dll

In this article

Requests that the common language runtime not call the finalizer for the specified object.

C# Copy

```
public static void SuppressFinalize (object obj);
```

Parameters

`obj` Object

The object whose finalizer must not be executed.

Exceptions

ArgumentNullException

`obj` is `null`.

Examples

The following example demonstrates how to use the SuppressFinalize method in a resource class to prevent a redundant garbage collection from being called. The example uses the dispose pattern to free both managed resources (that is, objects that implement IDisposable) and unmanaged resources.

C# Copy

```
using System;
using System.ComponentModel;
using System.Runtime.InteropServices;

public class ConsoleMonitor : IDisposable
{
   const int STD_INPUT_HANDLE = -10;
   const int STD_OUTPUT_HANDLE = -11;
   const int STD_ERROR_HANDLE = -12;

   [DllImport("kernel32.dll", SetLastError = true)]
   static extern IntPtr GetStdHandle(int nStdHandle);

   [DllImport("kernel32.dll", SetLastError = true)]
   static extern bool WriteConsole(IntPtr hConsoleOutput, string
lpBuffer,
          uint nNumberOfCharsToWrite, out uint lpNumberOfCharsWritten,
          IntPtr lpReserved);

   [DllImport("kernel32.dll", SetLastError = true)]
   static extern bool CloseHandle(IntPtr handle);

   private bool disposed = false;
   private IntPtr handle;
   private Component component;

   public ConsoleMonitor()
   {
      handle = GetStdHandle(STD_OUTPUT_HANDLE);
      if (handle == IntPtr.Zero)
         throw new InvalidOperationException("A console handle is not
available.");

      component = new Component();

      string output = "The ConsoleMonitor class constructor.\n";
      uint written = 0;
      WriteConsole(handle, output, (uint) output.Length, out written,
IntPtr.Zero);
   }

   // The destructor calls Object.Finalize.
   ~ConsoleMonitor()
   {
      if (handle != IntPtr.Zero) {
         string output = "The ConsoleMonitor finalizer.\n";
         uint written = 0;
         WriteConsole(handle, output, (uint) output.Length, out
written, IntPtr.Zero);
      }
```

```
      else {
         Console.Error.WriteLine("Object finalization.");
      }
      // Call Dispose with disposing = false.
      Dispose(false);
   }

   public void Write()
   {
      string output = "The Write method.\n";
      uint written = 0;
      WriteConsole(handle, output, (uint) output.Length, out written,
IntPtr.Zero);
   }

   public void Dispose()
   {
      string output = "The Dispose method.\n";
      uint written = 0;
      WriteConsole(handle, output, (uint) output.Length, out written,
IntPtr.Zero);

      Dispose(true);
      GC.SuppressFinalize(this);
   }

   private void Dispose(bool disposing)
   {
      string output = String.Format("The Dispose({0}) method.\n",
disposing);
      uint written = 0;
      WriteConsole(handle, output, (uint) output.Length, out written,
IntPtr.Zero);

      // Execute if resources have not already been disposed.
      if (! disposed) {
         // If the call is from Dispose, free managed resources.
         if (disposing) {
            Console.Error.WriteLine("Disposing of managed resources.");
            if (component != null)
               component.Dispose();
         }
         // Free unmanaged resources.
         output = "Disposing of unmanaged resources.";
         WriteConsole(handle, output, (uint) output.Length, out
written, IntPtr.Zero);

         if (handle != IntPtr.Zero) {
            if (! CloseHandle(handle))
               Console.Error.WriteLine("Handle cannot be closed.");
         }
```

```
        }
        disposed = true;
    }
}

public class Example
{
    public static void Main()
    {
        Console.WriteLine("ConsoleMonitor instance....");
        ConsoleMonitor monitor = new ConsoleMonitor();
        monitor.Write();
        monitor.Dispose();
    }
}
// If the monitor.Dispose method is not called, the example displays
the following output:
//       ConsoleMonitor instance....
//       The ConsoleMonitor class constructor.
//       The Write method.
//       The ConsoleMonitor finalizer.
//       The Dispose(False) method.
//       Disposing of unmanaged resources.
//
// If the monitor.Dispose method is called, the example displays the
following output:
//       ConsoleMonitor instance....
//       The ConsoleMonitor class constructor.
//       The Write method.
//       The Dispose method.
//       The Dispose(True) method.
//       Disposing of managed resources.
//       Disposing of unmanaged resources.
```

Remarks

This method sets a bit in the object header of `obj`, which the runtime checks when calling finalizers. A finalizer, which is represented by the Object.Finalize method, is used to release unmanaged resources before an object is garbage-collected. If `obj` does not have a finalizer or the GC has already signaled the finalizer thread to run the finalizer, the call to the SuppressFinalize method has no effect.

Objects that implement the IDisposable interface can call this method from the object's IDisposable.Dispose implementation to prevent the garbage collector from calling Object.Finalize on an object that does not require it. Typically, this is done to prevent the

finalizer from releasing unmanaged resources that have already been freed by the IDisposable.Dispose implementation.

Applies to

.NET

5.0 RC1

.NET Core

3.1, 3.0, 2.2, 2.1, 2.0, 1.1, 1.0

.NET Framework

4.8, 4.7.2, 4.7.1, 4.7, 4.6.2, 4.6.1, 4.6, 4.5.2, 4.5.1, 4.5, 4.0, 3.5, 3.0, 2.0, 1.1

.NET Standard

2.1, 2.0, 1.6, 1.5, 1.4, 1.3, 1.2, 1.1, 1.0

UWP

10.0

Xamarin.Android

7.1

Xamarin.iOS

10.8

Xamarin.Mac

3.0

See also

- ReRegisterForFinalize(Object)
- Finalize()
- Dispose Pattern
