


Static Constructors (C# Programming Guide)

07/20/2015 • 4 minutes to read •  +7

In this article

[Remarks](#)

[Example](#)

[C# language specification](#)

[See also](#)

A static constructor is used to initialize any [static](#) data, or to perform a particular action that needs to be performed once only. It is called automatically before the first instance is created or any static members are referenced.

C#

 Copy

```
class SimpleClass
{
    // Static variable that must be initialized at run time.
    static readonly long baseline;

    // Static constructor is called at most one time, before any
    // instance constructor is invoked or member is accessed.
    static SimpleClass()
    {
        baseline = DateTime.Now.Ticks;
    }
}
```

Remarks

Static constructors have the following properties:

- A static constructor does not take access modifiers or have parameters.
- A class or struct can only have one static constructor.
- Static constructors cannot be inherited or overloaded.
- A static constructor cannot be called directly and is only meant to be called by the common language runtime (CLR). It is invoked automatically.
- The user has no control on when the static constructor is executed in the program.
- A static constructor is called automatically to initialize the [class](#) before the first instance is created or any static members are referenced. A static constructor will run before an instance constructor. A type's static constructor is called when a static method assigned to an event or a delegate is invoked and not when it is assigned. If static field variable initializers are present in the class of the static constructor, they will be executed in the textual order in which they appear in the class declaration immediately prior to the execution of the static constructor.
- If you don't provide a static constructor to initialize static fields, all static fields are initialized to their default value as listed in [Default values of C# types](#).
- If a static constructor throws an exception, the runtime will not invoke it a second time, and the type will remain uninitialized for the lifetime of the application domain in which your program is running. Most commonly, a [TypeInitializationException](#) exception is thrown when a static constructor is unable to instantiate a type or for an unhandled exception occurring within a static constructor. For implicit static constructors that are not explicitly defined in source code, troubleshooting may require inspection of the intermediate language (IL) code.
- The presence of a static constructor prevents the addition of the [BeforeFieldInit](#) type attribute. This limits runtime optimization.
- A field declared as `static readonly` may only be assigned as part of its declaration or in a static constructor. When an explicit static constructor is not required, initialize static fields at declaration, rather than through a static constructor for

better runtime optimization.

ⓘ Note

Though not directly accessible, the presence of an explicit static constructor should be documented to assist with troubleshooting initialization exceptions.

Usage

- A typical use of static constructors is when the class is using a log file and the constructor is used to write entries to this file.
- Static constructors are also useful when creating wrapper classes for unmanaged code, when the constructor can call the `LoadLibrary` method.
- Static constructors are also a convenient place to enforce run-time checks on the type parameter that cannot be checked at compile time via constraints (Type parameter constraints).

Example

In this example, class `Bus` has a static constructor. When the first instance of `Bus` is created (`bus1`), the static constructor is invoked to initialize the class. The sample output verifies that the static constructor runs only one time, even though two instances of `Bus` are created, and that it runs before the instance constructor runs.

C#

 Copy

```
public class Bus
{
    // Static variable used by all Bus instances.
    // Represents the time the first bus of the day starts its route.
```

```
protected static readonly DateTime globalStartTime;

// Property for the number of each bus.
protected int RouteNumber { get; set; }

// Static constructor to initialize the static variable.
// It is invoked before the first instance constructor is run.
static Bus()
{
    globalStartTime = DateTime.Now;

    // The following statement produces the first line of output,
    // and the line occurs only once.
    Console.WriteLine("Static constructor sets global start time to {0}",
        globalStartTime.ToLongTimeString());
}

// Instance constructor.
public Bus(int routeNum)
{
    RouteNumber = routeNum;
    Console.WriteLine("Bus #{0} is created.", RouteNumber);
}

// Instance method.
public void Drive()
{
    TimeSpan elapsedTime = DateTime.Now - globalStartTime;

    // For demonstration purposes we treat milliseconds as minutes to simulate
    // actual bus times. Do not do this in your actual bus schedule program!
    Console.WriteLine("{0} is starting its route {1:N2} minutes after global start time {2}.",
        this.RouteNumber,
        elapsedTime.Milliseconds,
        globalStartTime.ToShortTimeString());
}
}
```

```
class TestBus
{
    static void Main()
    {
        // The creation of this instance activates the static constructor.
        Bus bus1 = new Bus(71);

        // Create a second bus.
        Bus bus2 = new Bus(72);

        // Send bus1 on its way.
        bus1.Drive();

        // Wait for bus2 to warm up.
        System.Threading.Thread.Sleep(25);

        // Send bus2 on its way.
        bus2.Drive();

        // Keep the console window open in debug mode.
        System.Console.WriteLine("Press any key to exit.");
        System.Console.ReadKey();
    }
}

/* Sample output:
    Static constructor sets global start time to 3:57:08 PM.
    Bus #71 is created.
    Bus #72 is created.
    71 is starting its route 6.00 minutes after global start time 3:57 PM.
    72 is starting its route 31.00 minutes after global start time 3:57 PM.
*/
```

C# language specification

For more information, see the [Static constructors](#) section of the [C# language specification](#).

See also

- [C# Programming Guide](#)
- [Classes and Structs](#)
- [Constructors](#)
- [Static Classes and Static Class Members](#)
- [Finalizers](#)
- [Constructor Design Guidelines](#)
- [Security Warning - CA2121: Static constructors should be private](#)

Is this page helpful?

 Yes  No
