Static Constructors (C# Programming Guide)

Article • 08/02/2024

A static constructor is used to initialize any static data, or to perform a particular action that needs to be performed only once. It's called automatically before the first instance is created or any static members are referenced. A static constructor is called at most once.

C#

```
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;
    }
}
```

There are several actions that are part of static initialization. Those actions take place in the following order:

1. *Static fields are set to 0*. The runtime typically does this initialization.
2. *Static field initializers run*. The static field initializers in the most derived type run.
3. *Base type static field initializers run*. Static field initializers starting with the direct base through each base type to System.Object.
4. *Any static constructor runs*. Any static constructors, from the ultimate base class of Object.Object through each base class through the type run. The order of static constructor execution isn't specified. However, all static constructors in the hierarchy run before any instances are created.

Important

There is one important exception to the rule that a static constructor runs before any instance is created. If a static field initializer creates an instance of the type, that initializer runs (including any call to an instance constructor) before the static

constructor runs. This is most common in the *singleton pattern* as shown in the following example:

C#

```
public class Singleton
{
    // Static field initializer calls instance constructor.
    private static Singleton instance = new Singleton();

    private Singleton()
    {
        Console.WriteLine("Executes before static constructor.");
    }

    static Singleton()
    {
        Console.WriteLine("Executes after instance constructor.");
    }

    public static Singleton Instance => instance;
}
```

A module initializer can be an alternative to a static constructor. For more information, see the specification for module initializers.

Remarks

Static constructors have the following properties:

- A static constructor doesn't take access modifiers or have parameters.
- A class or struct can only have one static constructor.
- Static constructors can't be inherited or overloaded.
- A static constructor can't be called directly and is only meant to be called by the common language runtime (CLR). It's invoked automatically.
- The user has no control on when the static constructor is executed in the program.
- A static constructor is called automatically. It initializes the class before the first instance is created or any static members declared in that class (not its base classes) are referenced. A static constructor runs before an instance constructor. If static field variable initializers are present in the class of the static constructor, they run in the textual order in which they appear in the class declaration. The initializers run immediately before 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 doesn't invoke it a second time, and the type remains uninitialized for the lifetime of the application domain. 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 static constructors that aren't explicitly defined in source code, troubleshooting might 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` can only be assigned as part of its declaration or in a static constructor. When an explicit static constructor isn't required, initialize static fields at declaration rather than through a static constructor for better runtime optimization.
- The runtime calls a static constructor no more than once in a single application domain. That call is made in a locked region based on the specific type of the class. No extra locking mechanisms are needed in the body of a static constructor. To avoid the risk of deadlocks, don't block the current thread in static constructors and initializers. For example, don't wait on tasks, threads, wait handles or events, don't acquire locks, and don't execute blocking parallel operations such as parallel loops, `Parallel.Invoke` and Parallel LINQ queries.

> **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 can't be checked at compile time via 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#

```
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 simu-
late
        // actual bus times. Do not do this in your actual bus schedule pro-
gram!
        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.
        Console.WriteLine("Press any key to exit.");
        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

- The C# type system
- Constructors

- Static Classes and Static Class Members
- Finalizers
- Constructor Design Guidelines
- Security Warning - CA2121: Static constructors should be private
- Module initializers