

# System.Threading.Timer Class

```
[ILAsm]
.class public sealed Timer extends System.MarshalByRefObject implements
System.IDisposable

[C#]
public sealed class Timer: MarshalByRefObject, IDisposable
```

## Assembly Info:

- *Name:* mscorlib
- *Public Key:* [00 00 00 00 00 00 00 00 04 00 00 00 00 00 00 00]
- *Version:* 2.0.x.x
- *Attributes:*
  - CLSCompliantAttribute(true)

## Implements:

- **System.IDisposable**

## Summary

Provides a mechanism for executing methods at specified intervals.

## Inherits From: System.MarshalByRefObject

**Library:** BCL

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

## Description

A `System.Threading.TimerCallback` delegate is used to specify the methods associated with a `Timer`. The methods do not execute in the thread that created the timer; they execute in a separate thread that is automatically allocated by the system. The timer delegate is specified when the timer is constructed, and cannot be changed.

When creating a timer, the application specifies an amount of time to wait before the first invocation of the delegate methods (due time), and an amount of time to wait between subsequent invocations (period). A timer invokes its methods once when its due time elapses, and invokes its methods once per period thereafter. These values can be changed, or the timer disabled using the `System.Threading.Timer.Change` method.

1 When a timer is no longer needed, use the `System.Threading.Timer.Dispose` method  
2 to free the resources held by the timer.

### 3 **Example**

4

5 The following example demonstrates the features of the `System.Threading.Timer`  
6 class.

7

8 [C#]

```
9 using System;
10 using System.Threading;
11
12 class TimerExampleState {
13     public int counter = 0;
14     public Timer tmr;
15 }
16
17 class App {
18     public static void Main() {
19         TimerExampleState s = new TimerExampleState();
20
21         // Create the delegate that invokes methods for the timer.
22         TimerCallback timerDelegate = new TimerCallback(CheckStatus);
23
24         // Create a timer that waits one second, then invokes every second.
25         Timer timer = new Timer(timerDelegate, s, 1000, 1000);
26
27         // Keep a handle to the timer, so it can be disposed.
28         s.tmr = timer;
29
30         // The main thread does nothing until the timer is disposed.
31         while (s.tmr != null)
32             Thread.Sleep(0);
33         Console.WriteLine("Timer example done.");
34     }
35     // The following method is called by the timer's delegate.
36
37     static void CheckStatus(Object state) {
38         TimerExampleState s = (TimerExampleState) state;
39         s.counter++;
40         Console.WriteLine("{0} Checking Status
41 {1}.", DateTime.Now.TimeOfDay, s.counter);
42         if (s.counter == 5) {
43             // Shorten the period. Wait 10 seconds to restart the timer.
44             (s.tmr).Change(10000, 100);
45             Console.WriteLine("changed...");
46         }
47         if (s.counter == 10) {
48             Console.WriteLine("disposing of timer...");
49             s.tmr.Dispose();
50             s.tmr = null;
51         }
52     }
}
```

```
1  }
2  An example of some output is
3
4  10:51:40.5809015 Checking Status 1.
5
6
7  10:51:41.5823515 Checking Status 2.
8
9
10 10:51:42.5838015 Checking Status 3.
11
12
13 10:51:43.5852515 Checking Status 4.
14
15
16 10:51:44.5867015 Checking Status 5.
17
18
19 changed...
20
21
22 10:51:54.5911870 Checking Status 6.
23
24
25 10:51:54.6913320 Checking Status 7.
26
27
28 10:51:54.7914770 Checking Status 8.
29
30
31 10:51:54.8916220 Checking Status 9.
32
33
34 10:51:54.9917670 Checking Status 10.
35
36
37 disposing of timer...
38
39
40 Timer example done.
41
42
43 The exact timings returned by this example will vary.
44
```

# Timer(System.Threading.TimerCallback, System.Object, System.Int32, System.Int32) Constructor

```
[ILAsm]
public rtspecialname specialname instance void .ctor(class
System.Threading.TimerCallback callback, object state, int32 dueTime,
int32 period)

[C#]
public Timer(TimerCallback callback, object state, int dueTime, int
period)
```

## Summary

Constructs and initializes a new instance of the `Timer` class.

## Parameters

Parameter	Description
<i>callback</i>	A <code>System.Threading.TimerCallback</code> delegate.
<i>state</i>	A <code>System.Object</code> containing application-specific information relevant to the methods invoked by <i>callback</i> , or <code>null</code> .
<i>dueTime</i>	A <code>System.Int32</code> containing the amount of time to delay before <i>callback</i> invokes its methods, in milliseconds. Specify <code>System.Threading.Timeout.Infinite</code> to prevent the timer from starting. Specify zero to start the timer immediately.
<i>period</i>	A <code>System.Int32</code> containing the time interval between invocations of the methods referenced by <i>callback</i> , in milliseconds. Specify <code>System.Threading.Timeout.Infinite</code> to disable periodic signaling.

## Description

*callback* invokes its methods once after *dueTime* elapses, and then invokes its methods each time the *period* time interval elapses.

If *dueTime* is zero, *callback* performs its first invocation immediately. If *dueTime* is `System.Threading.Timeout.Infinite`, *callback* does not invoke its methods; the timer is disabled, but can be re-enabled using the `System.Threading.Timer.Change` method.

If *period* is zero or `System.Threading.Timeout.Infinite` and *dueTime* is not `System.Threading.Timeout.Infinite`, *callback* invokes its methods exactly once; the periodic behavior of the timer is disabled, but can be re-enabled using the `System.Threading.Timer.Change` method.

## Exceptions

Exception	Condition
<b>System.ArgumentOutOfRangeException</b>	<i>dueTime</i> or <i>period</i> is negative and is not equal to <code>System.Threading.Timeout.Infinite</code> .
<b>System.ArgumentNullException</b>	<i>callback</i> is a null reference.

# Timer(System.Threading.TimerCallback, System.Object, System.TimeSpan, System.TimeSpan) Constructor

```
[ILAsm]
public rtspecialname specialname instance void .ctor(class
System.Threading.TimerCallback callback, object state, valuetype
System.TimeSpan dueTime, valuetype System.TimeSpan period)

[C#]
public Timer(TimerCallback callback, object state, TimeSpan dueTime,
TimeSpan period)
```

## Summary

Constructs and initializes a new instance of the `Timer` class.

## Parameters

Parameter	Description
<i>callback</i>	A <code>System.Threading.TimerCallback</code> delegate.
<i>state</i>	A <code>System.Object</code> containing application-specific information relevant to the methods invoked by <i>callback</i> , or <code>null</code> .
<i>dueTime</i>	A <code>System.TimeSpan</code> set to the amount of time to delay before <i>callback</i> invokes its methods. Set the value to <code>System.Threading.Timeout.Infinite</code> milliseconds to prevent the timer from starting. Specify zero to start the timer immediately.
<i>period</i>	A <code>System.TimeSpan</code> set to the time interval between invocations of the methods referenced by <i>callback</i> . Set the value to <code>System.Threading.Timeout.Infinite</code> milliseconds to disable periodic signaling.

## Description

The *callback* delegate invokes its methods once after *dueTime* elapses, and then invokes its methods each time the *period* time interval elapses.

If *dueTime*, in milliseconds, is zero, *callback* performs its first invocation immediately. If

*dueTime* is `System.Threading.Timeout.Infinite`, no method invocation occurs. The timer is disabled, but can be re-enabled using the `System.Threading.Timer.Change` method.

If *period* is zero or `System.Threading.Timeout.Infinite` milliseconds and *dueTime* is not `System.Threading.Timeout.Infinite`, *callback* invokes its methods exactly once. The periodic behavior of the timer is disabled, but can be re-enabled using the `System.Threading.Timer.Change` method.

## Exceptions

Exception	Condition
<b>System.ArgumentOutOfRangeException</b>	The number of milliseconds in the value of <i>dueTime</i> or <i>period</i> is negative and not equal to <code>System.Threading.Timeout.Infinite</code> , or is greater than <code>System.Int32.MaxValue</code> .
<b>System.ArgumentNullException</b>	<i>callback</i> is a null reference.

# Timer.Change(System.Int32, System.Int32)

## Method

```
[ILAsm]  
.method public hidebysig instance bool Change(int32 dueTime, int32 period)  
  
[C#]  
public bool Change(int dueTime, int period)
```

### Summary

Changes the start time and interval between method invocations for a timer.

### Parameters

Parameter	Description
<i>dueTime</i>	A <code>System.Int32</code> containing the amount of time to delay before the delegate specified at <code>System.Threading.Timer</code> construction time invokes its methods, in milliseconds. Specify <code>System.Threading.Timeout.Infinite</code> to prevent the timer from restarting. Specify zero to restart the timer immediately.
<i>period</i>	A <code>System.Int32</code> containing the time interval between invocations of the methods referenced by the delegate specified at <code>System.Threading.Timer</code> construction time, in milliseconds. Specify <code>System.Threading.Timeout.Infinite</code> to disable periodic signaling.

### Return Value

`true` if the current instance has not been disposed; otherwise, `false`.

### Description

The delegate specified at `System.Threading.Timer` construction time invokes its methods once after *dueTime* elapses, and then invokes its methods each time the *period* time interval elapses.

If *dueTime* is zero, the delegate specified at `System.Threading.Timer` construction time performs its next invocation immediately. If *dueTime* is `System.Threading.Timeout.Infinite`, no method invocation occurs. The timer is disabled, but can be re-enabled by calling this method and specifying a non-negative value for *dueTime*.



If *period* is zero or `System.Threading.Timeout.Infinite` and *dueTime* is not `System.Threading.Timeout.Infinite`, the delegate specified at `System.Threading.Timer` construction time invokes its methods exactly once. The periodic behavior of the timer is disabled, but can be re-enabled by calling this method and specifying a positive value for *period*.

## Exceptions

Exception	Condition
<b>System.ArgumentOutOfRangeException</b>	<i>dueTime</i> or <i>period</i> is negative and is not equal to <code>System.Threading.Timeout.Infinite</code> .

# Timer.Change(System.TimeSpan, System.TimeSpan) Method

```
[ILAsm]
.method public hidebysig instance bool Change(valuetype System.TimeSpan
dueTime, valuetype System.TimeSpan period)

[C#]
public bool Change(TimeSpan dueTime, TimeSpan period)
```

## Summary

Changes the start time and interval between method invocations for a timer.

## Parameters

Parameter	Description
<i>dueTime</i>	A <code>System.TimeSpan</code> set to the amount of time to delay before the delegate specified at <code>System.Threading.Timer</code> construction time invokes its methods. Specify <code>System.Threading.Timeout.Infinite</code> milliseconds to prevent the timer from restarting. Specify zero to restart the timer immediately.
<i>period</i>	A <code>System.TimeSpan</code> set to the time interval between invocations of the methods referenced by the delegate specified at <code>System.Threading.Timer</code> construction time. Specify <code>System.Threading.Timeout.Infinite</code> milliseconds to disable periodic signaling.

## Return Value

`true` if the current instance has not been disposed; otherwise, `false`.

## Description

The delegate specified at `System.Threading.Timer` construction time invokes its methods once after *dueTime* elapses, and then invokes its methods each time the *period* time interval elapses.

If *dueTime*, in milliseconds, is zero, the delegate specified at `System.Threading.Timer` construction time performs its next invocation immediately. If *dueTime* is `System.Threading.Timeout.Infinite` milliseconds, no method invocation occurs. The timer is disabled, but can be re-enabled by calling this method and specifying a non-

negative value for *dueTime*.

If *period* is zero or `System.Threading.Timeout.Infinite` milliseconds and *dueTime* is not `System.Threading.Timeout.Infinite` milliseconds, the delegate specified at `System.Threading.Timer` construction time invokes its methods exactly once. The periodic behavior of the timer is disabled, but can be re-enabled by calling this method and specifying a positive value for *period*.

## Exceptions

Exception	Condition
<b>System.ArgumentOutOfRangeException</b>	<i>dueTime</i> or <i>period</i> is negative and is not equal to <code>System.Threading.Timeout.Infinite</code> .

# 1 Timer.Dispose() Method

```
2 [ILAsm]  
3 .method public final hidebysig virtual void Dispose()  
  
4 [C#]  
5 public void Dispose()
```

## 6 Summary

7 Releases the resources held by the current instance.

## 8 Description

9 [Note: This method is implemented to support the System.IDisposable interface.]  
10  
11  
12

# Timer.Dispose(System.Threading.WaitHandle) Method

```
[ILAsm]  
.method public hidebysig instance bool Dispose(class  
System.Threading.WaitHandle notifyObject)  
  
[C#]  
public bool Dispose(WaitHandle notifyObject)
```

## Summary

Releases the resources held by the current instance.

## Parameters

Parameter	Description
<i>notifyObject</i>	Specifies a System.Threading.WaitHandle to be signaled when the timer has been disposed of.

## Return Value

true if the call succeeds; otherwise, false.

## Description

When this method completes, the System.Threading.WaitHandle specified by *notifyObject* is signaled.

This method calls System.GC.SuppressFinalize to prevent the garbage collector from finalizing the current instance.

## Exceptions

Exception	Condition
System.ArgumentNullException	<i>notifyObject</i> is null.

- 1
- 2
- 3

# 1 Timer.Finalize() Method

```
2 [ILAsm]  
3 .method family hidebysig virtual void Finalize()  
  
4 [C#]  
5 ~Timer()
```

## 6 Summary

7 Releases the resources held by the current instance.

## 8 Description

9 [Note: Application code does not call this method; it is automatically invoked by during  
10 garbage collection unless finalization by the garbage collector has been disabled. For  
11 more information, see `System.GC.SuppressFinalize`, and `System.Object.Finalize`.

12 This method overrides `System.Object.Finalize`.

13  
14  
15 ]