

1 System.Threading.Timer Class

```
2 [ILAsm]  
3 .class public sealed Timer extends System.MarshalByRefObject implements  
4 System.IDisposable  
  
5 [C#]  
6 public sealed class Timer: MarshalByRefObject, IDisposable
```

7 Assembly Info:

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

13 Implements:

- 14 • **System.IDisposable**

15 Summary

16 Provides a mechanism for executing methods at specified intervals.

17 Inherits From: System.MarshalByRefObject

18
19 **Library:** BCL

20
21 **Thread Safety:** All public static members of this type are safe for multithreaded operations.
22 No instance members are guaranteed to be thread safe.

24 Description

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

29
30 When creating a timer, the application specifies an amount of time to wait before the
31 first invocation of the delegate methods (due time), and an amount of time to wait
32 between subsequent invocations (period). A timer invokes its methods once when its
33 due time elapses, and invokes its methods once per period thereafter. These values can
34 be changed, or the timer disabled using the `System.Threading.Timer.Change` method.

35
36 When a timer is no longer needed, use the `System.Threading.Timer.Dispose` method
37 to free the resources held by the timer.

38 Example

1 The following example demonstrates the features of the System.Threading.Timer
2 class.

3
4 [C#]

```
5 using System;
6 using System.Threading;
7
8 class TimerExampleState {
9     public int counter = 0;
10    public Timer tmr;
11 }
12
13 class App {
14     public static void Main() {
15         TimerExampleState s = new TimerExampleState();
16
17         // Create the delegate that invokes methods for the timer.
18         TimerCallback timerDelegate = new TimerCallback(CheckStatus);
19
20         // Create a timer that waits one second, then invokes every second.
21         Timer timer = new Timer(timerDelegate, s, 1000, 1000);
22
23         // Keep a handle to the timer, so it can be disposed.
24         s.tmr = timer;
25
26         // The main thread does nothing until the timer is disposed.
27         while (s.tmr != null)
28             Thread.Sleep(0);
29         Console.WriteLine("Timer example done.");
30     }
31     // The following method is called by the timer's delegate.
32
33     static void CheckStatus(Object state) {
34         TimerExampleState s = (TimerExampleState) state;
35         s.counter++;
36         Console.WriteLine("{0} Checking Status
37 {1}.", DateTime.Now.TimeOfDay, s.counter);
38         if (s.counter == 5) {
39             // Shorten the period. Wait 10 seconds to restart the timer.
40             (s.tmr).Change(10000, 100);
41             Console.WriteLine("changed...");
42         }
43         if (s.counter == 10) {
44             Console.WriteLine("disposing of timer...");
45             s.tmr.Dispose();
46             s.tmr = null;
47         }
48     }
49 }
```

50 An example of some output is

```
51
52 10:51:40.5809015 Checking Status 1.
53
54
55 10:51:41.5823515 Checking Status 2.
```

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

1 Timer(System.Threading.TimerCallback, 2 System.Object, System.Int32, System.Int32) 3 Constructor

```
4 [ILAsm]  
5 public rtspecialname specialname instance void .ctor(class  
6 System.Threading.TimerCallback callback, object state, int32 dueTime,  
7 int32 period)  
8  
9 [C#]  
10 public Timer(TimerCallback callback, object state, int dueTime, int  
period)
```

11 Summary

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

13 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.

14 15 Description

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

18
19 If *dueTime* is zero, *callback* performs its first invocation immediately. If *dueTime* is
20 `System.Threading.Timeout.Infinite`, *callback* does not invoke its methods; the timer
21 is disabled, but can be re-enabled using the `System.Threading.Timer.Change` method.

22
23 If *period* is zero or `System.Threading.Timeout.Infinite` and *dueTime* is not
24 `System.Threading.Timeout.Infinite`, *callback* invokes its methods exactly once; the

1 periodic behavior of the timer is disabled, but can be re-enabled using the
2 `System.Threading.Timer.Change` method.

3 Exceptions

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

4

5

1 Timer(System.Threading.TimerCallback, 2 System.Object, System.TimeSpan, 3 System.TimeSpan) Constructor

```
4 [ILAsm]  
5 public rtspecialname specialname instance void .ctor(class  
6 System.Threading.TimerCallback callback, object state, valuetype  
7 System.TimeSpan dueTime, valuetype System.TimeSpan period)  
  
8 [C#]  
9 public Timer(TimerCallback callback, object state, TimeSpan dueTime,  
10 TimeSpan period)
```

11 Summary

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

13 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.

14 15 Description

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

18
19 If *dueTime*, in milliseconds, is zero, *callback* performs its first invocation immediately. If
20 *dueTime* is `System.Threading.Timeout.Infinite`, no method invocation occurs. The
21 timer is disabled, but can be re-enabled using the `System.Threading.Timer.Change`
22 method.

1
2 If *period* is zero or `System.Threading.Timeout.Infinite` milliseconds and *dueTime* is
3 not `System.Threading.Timeout.Infinite`, *callback* invokes its methods exactly once.
4 The periodic behavior of the timer is disabled, but can be re-enabled using the
5 `System.Threading.Timer.Change` method.

6 **Exceptions**

Exception	Condition
System.ArgumentOutOfRangeException	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> .
System.ArgumentNullException	<i>callback</i> is a null reference.

7

8

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

2 Method

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

7 Summary

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

9 Parameters

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

10

11 Return Value

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

13 Description

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

17

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

23

24 If *period* is zero or System.Threading.Timeout.Infinite and *dueTime* is not
25 System.Threading.Timeout.Infinite, the delegate specified at
26 System.Threading.Timer construction time invokes its methods exactly once. The

1 periodic behavior of the timer is disabled, but can be re-enabled by calling this method
2 and specifying a positive value for *period*.

3 **Exceptions**

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

4

5

1 Timer.Change(System.TimeSpan, 2 System.TimeSpan) Method

```
3 [ILAsm]  
4 .method public hidebysig instance bool Change(valuetype System.TimeSpan  
5 dueTime, valuetype System.TimeSpan period)  
  
6 [C#]  
7 public bool Change(TimeSpan dueTime, TimeSpan period)
```

8 Summary

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

10 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.

11

12 Return Value

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

14 Description

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

18

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

24

25 If *period* is zero or `System.Threading.Timeout.Infinite` milliseconds and *dueTime* is
26 not `System.Threading.Timeout.Infinite` milliseconds, the delegate specified at
27 `System.Threading.Timer` construction time invokes its methods exactly once. The

1 periodic behavior of the timer is disabled, but can be re-enabled by calling this method
2 and specifying a positive value for *period*.

3 **Exceptions**

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

4

5

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

1
2 **Timer.Dispose(System.Threading.WaitHandle**
3 **) Method**

```
4 [ILAsm]  
5 .method public hidebysig instance bool Dispose(class  
6 System.Threading.WaitHandle notifyObject)  
7  
8 [C#]  
9 public bool Dispose(WaitHandle notifyObject)
```

9 **Summary**

10 Releases the resources held by the current instance.

11 **Parameters**

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

12
13 **Return Value**

14 `true` if the call succeeds; otherwise, `false`.

15 **Description**

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

18
19 This method calls `System.GC.SuppressFinalize` to prevent the garbage collector from
20 finalizing the current instance.

21 **Exceptions**

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

22
23

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

14
15]

16