CountdownEvent Class

Reference

Definition

Namespace: System.Threading

Assembly: System.Threading.dll

Represents a synchronization primitive that is signaled when its count reaches zero.

C#

```
public class CountdownEvent : IDisposable
```

Inheritance Object → CountdownEvent

Implements IDisposable

Examples

The following example shows how to use a `CountdownEvent`:

C#

```
using System;
using System.Collections.Concurrent;
using System.Linq;
using System.Threading;
using System.Threading.Tasks;

class Example
{
    static async Task Main()
    {
        // Initialize a queue and a CountdownEvent
        ConcurrentQueue<int> queue = new ConcurrentQueue<int>
(Enumerable.Range(0, 10000));
        CountdownEvent cde = new CountdownEvent(10000); // initial count =
10000

        // This is the logic for all queue consumers
```

```
        Action consumer = () =>
        {
            int local;
            // decrement CDE count once for each element consumed from queue
            while (queue.TryDequeue(out local)) cde.Signal();
        };

        // Now empty the queue with a couple of asynchronous tasks
        Task t1 = Task.Factory.StartNew(consumer);
        Task t2 = Task.Factory.StartNew(consumer);

        // And wait for queue to empty by waiting on cde
        cde.Wait(); // will return when cde count reaches 0

        Console.WriteLine("Done emptying queue.  InitialCount={0},
CurrentCount={1}, IsSet={2}",
            cde.InitialCount, cde.CurrentCount, cde.IsSet);

        // Proper form is to wait for the tasks to complete, even if you know
that their work
        // is done already.
        await Task.WhenAll(t1, t2);

        // Resetting will cause the CountdownEvent to un-set, and resets
InitialCount/CurrentCount
        // to the specified value
        cde.Reset(10);

        // AddCount will affect the CurrentCount, but not the InitialCount
        cde.AddCount(2);

        Console.WriteLine("After Reset(10), AddCount(2): InitialCount={0},
CurrentCount={1}, IsSet={2}",
            cde.InitialCount, cde.CurrentCount, cde.IsSet);

        // Now try waiting with cancellation
        CancellationTokenSource cts = new CancellationTokenSource();
        cts.Cancel(); // cancels the CancellationTokenSource
        try
        {
            cde.Wait(cts.Token);
        }
        catch (OperationCanceledException)
        {
            Console.WriteLine("cde.Wait(preCanceledToken) threw OCE, as ex-
pected");
        }
        finally
        {
           cts.Dispose();
        }
```

```
        // It's good to release a CountdownEvent when you're done with it.
        cde.Dispose();
    }
}
// The example displays the following output:
//    Done emptying queue.  InitialCount=10000, CurrentCount=0, IsSet=True
//    After Reset(10), AddCount(2): InitialCount=10, CurrentCount=12,
IsSet=False
//    cde.Wait(preCanceledToken) threw OCE, as expected
```

Constructors

CountdownEvent(Int32)	Initializes a new instance of CountdownEvent class with the specified count.

Properties

CurrentCount	Gets the number of remaining signals required to set the event.
InitialCount	Gets the numbers of signals initially required to set the event.
IsSet	Indicates whether the CountdownEvent object's current count has reached zero.
WaitHandle	Gets a WaitHandle that is used to wait for the event to be set.

Methods

AddCount()	Increments the CountdownEvent's current count by one.
AddCount(Int32)	Increments the CountdownEvent's current count by a specified value.
Dispose()	Releases all resources used by the current instance of the CountdownEvent class.
Dispose(Boolean)	Releases the unmanaged resources used by the CountdownEvent, and optionally releases the managed resources.
Equals(Object)	Determines whether the specified object is equal to the current object. (Inherited from Object)

GetHashCode()	Serves as the default hash function. (Inherited from Object)
GetType()	Gets the Type of the current instance. (Inherited from Object)
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
Reset()	Resets the CurrentCount to the value of InitialCount.
Reset(Int32)	Resets the InitialCount property to a specified value.
Signal()	Registers a signal with the CountdownEvent, decrementing the value of CurrentCount.
Signal(Int32)	Registers multiple signals with the CountdownEvent, decrementing the value of CurrentCount by the specified amount.
ToString()	Returns a string that represents the current object. (Inherited from Object)
TryAddCount()	Attempts to increment CurrentCount by one.
TryAddCount(Int32)	Attempts to increment CurrentCount by a specified value.
Wait()	Blocks the current thread until the CountdownEvent is set.
Wait(CancellationToken)	Blocks the current thread until the CountdownEvent is set, while observing a CancellationToken.
Wait(Int32)	Blocks the current thread until the CountdownEvent is set, using a 32-bit signed integer to measure the timeout.
Wait(Int32, CancellationToken)	Blocks the current thread until the CountdownEvent is set, using a 32-bit signed integer to measure the timeout, while observing a CancellationToken.
Wait(TimeSpan)	Blocks the current thread until the CountdownEvent is set, using a TimeSpan to measure the timeout.
Wait(TimeSpan, CancellationToken)	Blocks the current thread until the CountdownEvent is set, using a TimeSpan to measure the timeout, while observing a CancellationToken.

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8
.NET Framework	4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.6, 2.0, 2.1
UWP	10.0
Xamarin.iOS	10.8
Xamarin.Mac	3.0

Thread Safety

All public and protected members of CountdownEvent are thread-safe and may be used concurrently from multiple threads, with the exception of Dispose(), which must only be used when all other operations on the CountdownEvent have completed, and Reset(), which should only be used when no other threads are accessing the event.

See also

- CountdownEvent