Threads in .Net

A thread is defined as the execution path of a program. Each thread defines a unique flow of control. If your application involves complicated and time consuming operations like database access or some intense I/O operations, then it is often helpful to set different execution paths or threads, with each thread performing a particular job.

Threads are lightweight processes. One common example of use of thread is implementation of concurrent programming by modern operating systems. Use of threads saves wastage of CPU cycle and increase efficiency of an application.

So far we have written programs where a single thread runs as a single process which is the running instance of the application. However, this way the application can perform one job at a time. To make it execute more than one task at a time, it could be divided into smaller threads.

In .NET, the threading is handled through the System.Threading namespace. Creating a variable of the *System.Threading.Thread* type allows you to create a new thread to start working with. It allows you to create and access individual threads in a program.

**Creating Thread:**

A thread is created by creating a Thread object, giving its constructor a ThreadStart reference.

ThreadStart childthreat = new ThreadStart(childthreadcall);

**The Thread Life Cycle:**

The life cycle of a thread starts when an object of the System.Threading.Thread class is created and ends when the thread is terminated or completes execution.

Following are the various states in the life cycle of a thread :

* **The Unstarted State:** it is the situation when the instance of the thread is created but the Start method has not been called.
* **The Ready State:** it is the situation when the thread is ready to run and waiting CPU cycle.
* **The Not Runnable State:** a thread is not runnable, when:
  + Sleep method has been called
  + Wait method has been called
  + Blocked by I/O operations
* **The Dead State:** it is the situation when the thread has completed execution or has been aborted.

**The Thread Priority:**

The Priority property of the Thread class specifies the priority of one thread with respect to other. The .Net runtime selects the ready thread with the highest priority.

The priorities could be categorised as:

* Above normal
* Below normal
* Highest
* Lowest
* Normal

Once a thread is created its priority is set using the Priority property of the thread class.

NewThread.Priority = ThreadPriority.Highest;

**Example:**

Working with single thread

Thread Start\_thread=new Thread(NewThread);

NewThread.start();

Working with multi thread:

------------------------------------------------------------

ThreadStart job = new ThreadStart(ThreadJob);

Thread thread = new Thread(job);

thread.Start();

Working with Threadpool:

ThreadPool.QueueUserWorkItem(new Program().testThread1);

            ThreadPool.QueueUserWorkItem(new Program().testThread2);

Public void testThread1(object o)

{

}

Public void testThread2()

{

}

Thread Methods

sleep();🡪Makes the thread for sleep for some time

Suspend();->Makes the thread for pause for sometime

Abort();🡪Makes the thread for terminating.

## Features and Benefits of Threads

Mutually exclusive tasks, such as gathering user input and background processing can be managed with the use of threads. Threads can also be used as a convenient way to structure a program that performs several similar or identical tasks concurrently.

One of the advantages of using the threads is that you can have multiple activities happening simultaneously. Another advantage is that a developer can make use of threads to achieve faster computations by doing two different computations in two threads instead of serially one after the other.

### **How do they work**

A multitasking operation system divides the available processor time among the processes and threads that need it. A thread is executed in the given time slice, and then it is suspended and execution starts for next thread/process in the queue. When the OS switches from one thread to another, it saves thread context for preempted thread and loads the thread context for the thread to execute.

The length of time slice that is allocated for a thread depends on the OS, the processor, as also on the priority of the task itself.

### **Scheduling Threads**

Every thread has a thread priority assigned to it. Threads created within the common language runtime are initially assigned the priority of ThreadPriority.Normal. Threads created outside the runtime retain the priority they had before they entered the managed environment. You can get or set the priority of any thread with theThread.Priority property.

Threads are scheduled for execution based on their priority. Even though threads are executing within the runtime, all threads are assigned processor time slices by the operating system. The details of the scheduling algorithm used to determine the order in which threads are executed varies with each operating system. Under some operating systems, the thread with the highest priority (of those threads that can be executed) is always scheduled to run first. If multiple threads with the same priority are available, the scheduler cycles through the threads at that priority, giving each thread a fixed time slice in which to execute. As long as a thread with a higher priority is available to run, lower priority threads do not get to execute. When there are no more run able threads at a given priority, the scheduler moves to the next lower priority and schedules the threads at that priority for execution. If a higher priority thread becomes run able, the lower priority thread is preempted and the higher priority thread is allowed to execute once again. On top of all that, the operating system can also adjust thread priorities dynamically as an application's user interface is moved between foreground and background. Other operating systems might choose to use a different scheduling algorithm.

### **Pausing and Resuming threads**

After you have started a thread, you often want to pause that thread for a fixed period of time. Calling [Thread.Sleep](http://www.codeproject.com/Articles/8694/_target)causes the current thread to immediately block for the number of milliseconds you pass to Sleep, yielding the remainder of its time slice to another thread. One thread cannot call Sleep on another thread. CallingThread.Sleep(Timeout.Infinite) causes a thread to sleep until it is interrupted by another thread that calls[Thread.Interrupt](http://msdn.microsoft.com/library/en-us/cpref/html/frlrfSystemThreadingThreadClassInterruptTopic.asp) or is aborted by [Thread.Abort](http://msdn.microsoft.com/library/en-us/cpref/html/frlrfSystemThreadingThreadClassAbortTopic.asp" \t "_blank).

**Most Common Instance Member of the System.Threading.Thread class**

The following are the most common instance members of the System.Threading.Thread class:

* **Name**   
    
  A property of string type used to get/set the friendly name of the thread instance.
* **Priority**   
    
  A property of type System.Threading.ThreadPriority to schedule the priority of threads.
* **IsAlive**   
    
  A Boolean property indicating whether the thread is alive or terminated.
* **ThreadState**   
    
  A property of type System.Threading.ThreadState, used to get the value containing the state of the thread.
* **Start()**Starts the execution of the thread.
* **Abort()**   
    
  Allows the current thread to stop the execution of the thread permanently.
* **Suspend()**   
    
  Pauses the execution of the thread temporarily.
* **Resume()**Resumes the execution of a suspended thread.
* **Join()**  
    
  Make the current thread wait for another thread to finis