Multi-Threading

Tasks (introduced in .Net 4.0)

A Task is a higher level of abstraction- it represents a concurrent operation that **may** or **may not** be backed by thread. Tasks ***can*** use the Thread pool to lessen the start up time. The CLR runs tasks on pooled threads, which is ideal for short running tasks, which are by default background threads. This means that when the main thread ends, so do any tasks that you create. Hence you must block the main thread after starting the task (using Task.Wait()).

You can use the [Task.Run](https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.task.run) (introduced in .Net 4.5) method to create and start a task in one operation. These are “hot” tasks as they are kickstarted from initiation. The Run methods are the preferred way to create and start tasks when more control over the creation and scheduling of the task is not needed.

“Cold” tasks (tasks in stopped state) are created using Task’s constructor, although not used in practice.

Long Running Tasks

By default, The CLR runs tasks on pooled threads, which is ideal for short running tasks. For longer running tasks and blocking operations, we can prevent the use of Thread Pools as such:

Task t = Task.Factory.StartNew(()=> …., **TaskCreationOptions.LongRunning**);

Returning Values

Task<int> T = Task.Run(() =>{ return a + 10; });

Task has a generic subclass called Task<TResult> that allows a task to emit a return value. We can obtain the result by querying the Result property. If the task hasn’t finished, accessing this property will block the current thread until the task finishes.

Exceptions

If the task faults we can know from IsFaulted and IsCanceled properties of the Task. If both the properties return false, no error occurred. If ‘IsCanceled’ is true, an OperationCanceledException was thrown for that task. If ‘IsFaulted’ is true, some other exception has happened and details of the same will be available in the Exception object.

Additionally, if the task throws an unhandled exception, the CLR wraps the exception in “AggregateException” in order to play well with Parallel Programming scenarios.

Task.Delay

Task.Delay is asynchronous to Thread.Sleep().