Threading in C#

Authored by : Sushant Banerjee
Email : sushantba@cybage.com

Presented by : Sushant Banerjee
Extn. 7210

Agenda

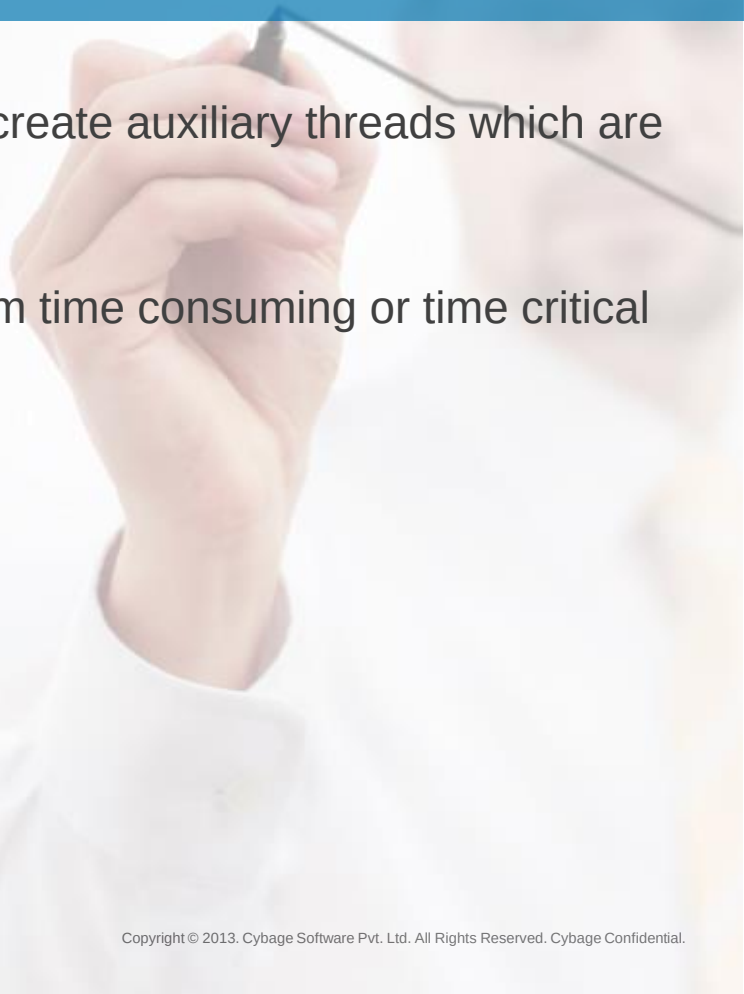
- Introduction
- Creating New Threads
- Multithreading
- Setting Priority
- Synchronization

What is Threading

- Threading enables you to perform **concurrent processing**.
- In other words, you can do **more than one operation** at a time.
- In .Net Framework **System.Threading** namespace used for threading.
- By default a program has one thread which is also known as **primary thread**.

Worker Threads

- In addition to primary thread you can create auxiliary threads which are also known as **worker threads.**
- Worker threads can be used to perform time consuming or time critical tasks.



Multithreading

- Multithreading is a process in which different tasks execute on **separate threads**.
- A multithreaded applications can **perform multiple tasks** at the same time.
- Multithreading is also known as **free threading**.
- In multithreaded applications the user interface **always remains active**.
- Multithreading can be used to write scalable applications which means that you can add new threads as the workload increases.

Creating Thread

```
//create a new worker thread and pass method reference
  to threadstart delegate
Thread workerThread = new Thread(PrintEvenNumbers);

//start worker thread
workerThread.Start();
```

Thread Methods

Method	Action
Start	Causes a thread to start to run.
Sleep	Pauses a thread for a specified time.
Suspend	Pauses a thread.
Abort	Stops a thread.
Resume	Restarts a suspended thread

Thread Properties

Method	Action
IsAlive	Contains the value True if a thread is active
Name	Gets or sets the name of a thread. Most frequently used to discover individual threads when you debug.
Priority	Gets or sets a value that is used by the operating system to prioritize thread scheduling.
ThreadState	Contains a value that describes a thread's state or states.

Thread Priorities

- The operating system allocates
  - Longer time slices to high priority threads and
  - Shorter time slices to low priority threads.
- You can use priority property of threads to set the priority level.
- Threads are scheduled for execution based on their priority.

```
workerThread.Priority = ThreadPriority.Highest;
```

Thread Priorities

Member name	Description
AboveNormal	The Thread can be scheduled after threads with Highest priority and before those with Normal priority.
BelowNormal	The Thread can be scheduled after threads with Normal priority and before those with Lowest priority.
Highest	The Thread can be scheduled before threads with any other priority.
Lowest	The Thread can be scheduled after threads with any other priority.
Normal	The Thread can be scheduled after threads with AboveNormal priority and before those with BelowNormal priority. Threads have Normal priority by default.

Thread Synchronization

- In multithreaded applications each thread executes asynchronously.
- That means each thread can access same resources such as files, networks and memory at the same time.
- This asynchronous nature of threads result into unpredictable data corruption.
- Thread synchronization helps you to handle these situations using lock and monitors.

Lock Statement

- The lock statement can be used to ensure that a block of code runs and completes its execution without interruption by other threads.

```
private System.Object lockThis = new System.Object();
        public void Process()
        {
            lock (lockThis)
            {
                // Access thread-sensitive resources.
            }
        }
```

Monitors

- Just like lock keyword, monitor also prevents blocks of code from simultaneous execution by multiple threads.

```
public void Execute()
{
  object obj = new object();
  System.Threading.Monitor.Enter(obj);
  Try
        {PrintEvenNumbers();}
  Finally
        { System.Threading.Monitor.Exit(obj); }
}
```

Deadlocks – The Worst Situations

- Even after implementing thread synchronization there is always a danger of creating a deadlock in multithreaded applications.
- When multiple threads are waiting for each other, the application halts and the situation is called deadlock.
- You need to plan your threads carefully to avoid deadlocks.

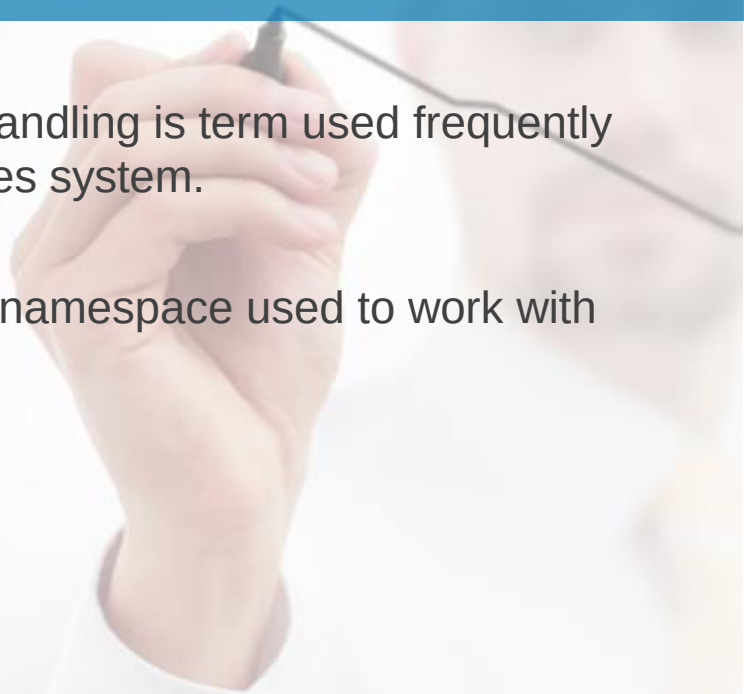
File I/O

Agenda

- Introduction
- File Stream
- Stream Writer
- Stream Reader

What is File I/O

- File I/O or File Input / Output or File Handling is term used frequently when you work with files and directories system.
- In .NET Framework **System.IO** is the namespace used to work with files and directories.



Files and Streams

- A file is a collection of persistent data stored in the disk using a file name with a directory path.
- When you open a file for reading or writing it becomes a stream.
- A stream is a sequence of bytes traveling from a source to destination over a communication path.
- Streams may be input stream and output stream.
  - The input stream is used for reading data from files.
  - The output stream is used for writing data into files.

FileStream Class

- The FileStream class is used to read from, write to, open and close files on a file system.
- When you create an object of the FileStream class you pass different parameters to the constructor of this class.
  - String path
  - FileMode
  - FileAccess
  - FileShare

FileMode Enum

- **Append** : Opens the file if it exists and seeks to the end of the file or creates a new file.
- **Create** : The OS will create a new file. If the file already exists, it will be overwritten.
- **CreateNew** : The OS will create a new file. If the file already exists a System.IO.IOException is thrown.
- **Open** : The OS should open an existing file. If the file does not exist a System.IO.FileNotFoundException is thrown.
- **OpenOrCreate** : The OS should open a file if it exists, otherwise a new file should be created.
- **Truncate** : The OS should open a file and truncates its size to zero bytes.

FileAccess Enum

- **Read** : Read access to the file. Data can be read from the file.
- **Write** : Write access to the file. Data can be written to the file.
- **ReadWrite** : Read and Write access to the file. Data can be written to and read from the file.

FileShare Enum

- **Read** : Allows subsequent opening of the file for reading.
- **Write** : Allows subsequent opening of the file for writing.
- **ReadWrite** : Allows subsequent opening of the file for reading or writing.
- **None** : Declines sharing of the current file. Any request to open the file will fail until the file is closed.
- **Delete** : Allows subsequent deleting of a file.

Character Data I/O

- The following classes can be used to perform read and write operation
- **StreamWriter** : This class can be used to write character data to a stream.
- **StreamReader** : This class can be used to read character from a byte stream.

Binary Data I/O

- The following classes are used specially for input and output operations on binary data :
- **BinaryReader** : Reads primitive data types as binary values.
- **BinaryWriter** : Writes primitive data types in binary format.

Bibliography, Important Links

- http://msdn.microsoft.com/en-us/library/hyz69czz(v=vs.110).aspx
- http://msdn.microsoft.com/en-us/library/1c9txz50(v=vs.110).aspx
- http://www.codeproject.com/Articles/274062/Improved-Multi-Threading-Support-in-Net
- http://msdn.microsoft.com/en-us/library/k3352a4t(v=vs.110).aspx

Any Questions?

Thank you!