Thin Books

Licence Agreement
1) You are not allowed to distribute this book in any format.
2) You may not reproduce any part of this book
3) You add a link to your site to http://www.RealSoft.co.uk.
4) RealSoft reserves the right to remove this book without notice.
5) RealSoft reserves the right to change the terms of this agreement without notice.
6) The Term “Thin Books”™ is a trademark of RealSoft Limited
7) The Term “Thin Book”™ is a trademark of RealSoft Limited

The idea behind “Thin Books” is to provide you with the basics in any given topic, they do not try to dig into the depths but simply provide you with quick start and from there you can decide to, or not to, read more details by search the rest of the sprawling web.

Note this book does assume basic coding / c# knowledge

This is a Draft

Thin Books

Table of Contents

Thin Books

To Thread Or Not To Thread – That is the question?

This section will outline the situations where you should and should not use threading, don't forget nothing comes for free!

Use Threading When

1) In Winform / Client GUI application where the task being undertaken is long running and therefore would make the user experience rather frustrating.
2) The actual task at hand will benefit from splitting the task into discrete operations, where each operation is autonomous and thus results in faster execution (Think dual core)
3) Where the operation has multiple clients, if the operation was single threaded then subsequent clients would have to wait, and who likes to wait? (Think about web sites, would the internet work if you had to wait for someone else to finish?)

Don't Use Threading For the Fun Of It

AS the intro to this section mentioned, nothing comes for free, so why shouldn't you just create threads for everything?

1) Threads are expensive to create (It takes time)
2) Threads take up valuable resources, such as memory.
3) Threads for the CPU to do more work, the thread needs to be managed.
4) No benefit is gained.
5) Your code will be more complex.
6) You have to think about the problems associated with Threading, it hurts to think!

Thin Books

Threading Issues

The section will outline the types of problems that you may encounter and how to resolve them

Dead Lock

Problem Definition

A dead lock is the situation where 2 or more artefacts are waiting on each other.
Example: You have a pen and your friend has a piece of paper, you both wish to write a letter but neither of you can since you need both a pen and paper to achieve the task, one of you has to give up the artefact that you are holding or you both will just wait and wait and wait, get my drift!?

Solutions

1) Try and acquire both the pen and the paper, if you cannot get hold of both of them then release the one you have.

Race Condition

Problem Definition

This is an unwanted situation is a system where the order in which things occur is vital.

Example: You're counting sweets that are all lined up, and whilst you are counting your friend eats some of the sweets that you've already counted, so by the time you have finished counting the actual number of sweets that remain is no longer the number that you have counted.

Solutions

1) Lock the Sweets so that no-one can access them whilst you are counting them.

Thin Books

Locking Mechanisms In .Net

This section will outline the various locking mechanisms available.

System.Threading.Interlocked

This class ensures that updates to an instance variable are done to completion, the reason for saying this is that when you increment / decrement an instance member the following happens

1) Load the variable to the register
2) Increment / Decrement the variable in the Register
3) Copy the result back to the the variable

Since we have multiple steps there is the possibility that the thread may be interrupted prior to completing all the steps.

Sample

```
private static int _theInteger = 0;
public static int UpdateTheInteger(int newValue)
{
        InterLocked.Exchange(ref _theValue, newValue);
}
```

Other operations:
Interlocked.Increment, Interlock.Decrement, Interlocked.CompareExchanged

lock

This operation simple provides the means to lock a section (block) of code such that it completes prior to anyone else running the code

Sample

```
private object _theLock = new object(); //Note every object can be locked including “this”
public void CriticalWork(){
        lock(_theLock)
        {
                // Do Your Stuff Here
        }
}
```

System.Threading.Monitor (Static Class)

Use this when you wish to acquire an exclusive lock on a resource

Samples

```
System.Collections.Generic.List<int> _theList = new List<int>();

public void Sample1(){
        Monitor.Enter(_theList); //Acquire Exclusive lock on _theList
        _theList.Add(1);
}

public bool Sample2(){
        if( ! Monitor.TryEnter(_theList)){ //Try And Acquire Exclusive lock on _theList
        //NOTE : you can provide a second param to TryEnter, this indicates how long
        //to wait before giving up.
                return false; // Indicate that you failed to do the operation
        }
        else{
                _theList.Add(1);
                return true;
        }
}
```