WPF Threading

Building Resoponsive User Interfaces

A Win32 window has to run inside a thread whose COM apartment state is `ApartmentState.STA` We achieve this by marking the entry method Main with the following attribute.

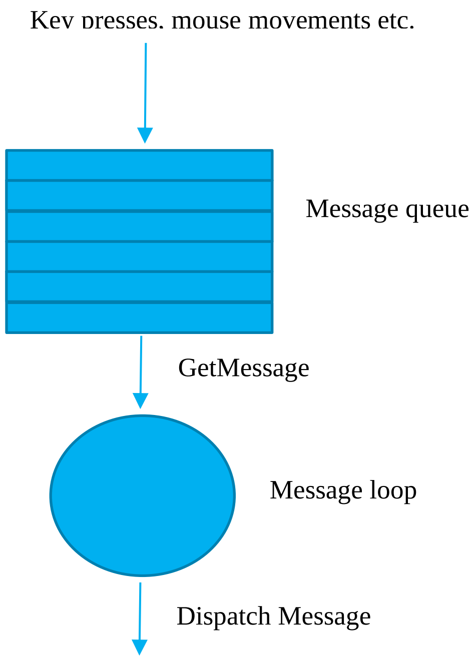
```
[System.STAThread]
public static void Main(string[] args)
```

In order for the thread to be a UI thread it needs to have a message loop. The message loop is setup by calling.

```
    System.Windows.Threading.Dispatcher.Run();
```

Inside this method WPF will setup the message loop. It is the message loop that makes the main thread a user interface thread. The message loop sits inside a loop, pulling messages from a message queue and dispatching them. Any window's input messages such as keystrokes and mouse presses are pushed onto the message queue by the operating system. It is then the message loop's job to pull these from the queue, process them and dispatch them such that handlers can respond to them.

If any thread other than the user interface thread wants to interact with elements created by the user interface thread in a thread safe fashion, they must do so via the dispatch method.



Any WPF object whose type extends `DispatcherObject` has what is known as thread affinity. This means it can only be safely accessed from the user interface thread that created it. In order to interact with such an object in a thread safe manner we need to use the dispatcher. The DispatcherObject type defines a Dispatcher object which can be used for this purpose. Almost all WPF types extend the type `DispatcherObject` and as such have access to the dispatcher associated with the thread which created them.

In actual fact a single WPF application can have multiple UI threads

Multiple UI Threads

The following code starts up multiple message loops.

```
public class B_MultipleUIThreads
{
    public static void Run()
    {
        CreateWindowOnNewUiThread(1.ToString());
        CreateWindowOnNewUiThread(2.ToString());
        CreateWindowOnNewUiThread(3.ToString());
    }

    public static void CreateWindowOnNewUiThread(string windowName)
    {
        var t = new Thread(Start);
        t.SetApartmentState(ApartmentState.STA);
        t.Start();
    }

    public static void Start(object o)
    {
        var w = new Window();
        w.Show();
        Dispatcher.Run();
    }
}
```

Communicating across UI Threads

Multiple UI threads and background threads communicate by using the dispatcher. We can then ask any WPF type that extends DispatcherObject to give us its Dispatcher and then we can use BeginInvoke to add a delegate to the message queue for the message loop to process.

```
public class C_Communicating_Across_UI_Threads : Window
{
    public C_Communicating_Across_UI_Threads()
    {
        Button b = new Button();
        this.Content = b;
        b.Content = "Click Me";
        b.Width = 100;
        b.Height = 100;
        b.Click += SwtichOtherWindowsColor;
    }


    private static C_Communicating_Across_UI_Threads _windowA;
    private static C_Communicating_Across_UI_Threads _windowB;

    public C_Communicating_Across_UI_Threads OtherWindow { get; set; }

    public static void Run()
    {
        var threadA = new Thread(() =>
        {
            _windowA = new C_Communicating_Across_UI_Threads();
            _windowA.Show();

            var threadB = new Thread(() =>
            {
                _windowB = new C_Communicating_Across_UI_Threads();
                _windowB.Show();
                _windowB.OtherWindow = _windowA;
                _windowA.OtherWindow = _windowB;
                Dispatcher.Run();
            });
            threadB.SetApartmentState(ApartmentState.STA);
            threadB.Start();

            Dispatcher.Run();
        });
        threadA.SetApartmentState(ApartmentState.STA);
        threadA.Start();
    }

    private void SwtichOtherWindowsColor(object sender, RoutedEventArgs e)
    {
        Action a = () => OtherWindow.Background = Brushes.Red;
        OtherWindow.Dispatcher.BeginInvoke(DispatcherPriority.Normal, a);
    }
}
```

WPF Threading Model - Questions

Basics

What is required for a thread to be a UI Thread?

- *Its COM apartment state must be set to STA (Single Threaded Affinity)*
- *It must be associated with a message loop and dispatcher*

How does a message loop work?

- *The OS pushes keystrokes, mouse clicks etc onto a message queue*
- *The message loop pulls messages from the queue and dispatches them*

What is the relationship between WPF objects and the UI Thread?

- *Any object which inherits from DispatcherObject must be created on a UI Thread.*
- *It is said to have thread affinity with the thread that created it.*

How can background threads safely update WPF Objects?

Non-UI threads must ask DependencyObject for its dispatcher and use the dispatcher

Given a WPF window w write code to safely set its background colour from a non-UI Thread?

```
w.Dispatcher.BeginInvoke( new Action( ()=>
{
    w.Background = new SolidColorBrush(System.Windows.Media.Colors.Red) ;
}));
```

How do you mark a main method such that it can serve as a WPF application main thread?

[STAThread]