Prerequisites

Getting started

The WPF framework is spread over three Dll's

- PresentationCore.dll
- PresentationFramework.dll
- WindowsBase.dll

For most .NET project we also need to include `System.xam.dll` A simple bare bones WPF MainClass.cs is as follows.

Listing 1Basic MainClass

```
using System;
using System.Windows;
using System.Windows.Threading;

public class MainClass
{
    [STAThread]
    private static void Main(String[] args)
    {
        Application application = new Application();
        bool b = application == Application.Current;
        Window w = new Window();
        application.Run(w);
    }
}
```

The Build

Although we will often use VisualStudio to build WPF applications it is instructive to see how to setup a MSBuild csproj file by hand

Listing 2Building a WPF Application

```
<?xml version="1.0" encoding="utf-8"?>
<Project  xmlns="http://schemas.microsoft.com/developer/msbuild/2003"
DefaultTargets="Build" ToolsVersion="4.5">
  <PropertyGroup>
    <AssemblyName>Kenny</AssemblyName>
    <PlatformTarget>AnyCPU</PlatformTarget>
    <OutputPath>.</OutputPath>
    <TargetFrameworkVersion>4.5</TargetFrameworkVersion>
  </PropertyGroup>
  <ItemGroup>
    <Compile Include="MainClass.cs"/>
    <Reference Include="System"/>
    <Reference Include="PresentationFramework"/>
    <Reference Include="PresentationCore"/>
    <Reference Include="WindowsBase"/>
    <Reference Include="System.xaml"/>
  </ItemGroup>
  <Target Name="Build">
    <Csc Sources="@(Compile)" References="@(Reference)"
    OutputAssembly="$(OutputPath)$(AssemblyName).exe" />
    </Target>
  <Import Project="$(MSBuildToolsPath)\Microsoft.CSharp.targets" />
</Project>
```

We would compile this against the three WPF Framework dlls to create our windows executable. For an example see the Appendix Simplest WPG Executable. This minimalist piece of code highlights some very important aspects of WPF applications;

- Every application requires at least one window
- The application requires a UI thread
- Most application have an instance of the Application class

The (Win32) Window

The first point is that a WPF application needs to contain at least one window. It is after all a user interface application. The window in a WPF application is just a Win32 window. The OS doesn't know the difference between a window with Win32 content and a window with WPF content.

We can create any number of windows. By default, as each window is closed the other windows remain running. If we want to create a *modeless dialog* which closes when its parent closes, we need to set the child windows `Owner` property to be the parent window.

The Threading Model

STA Thread

A Win32 window must 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)
```

Message loop

To make our new STAThread a UI Thread it needs to have an associated message loop. The message loop is setup by calling. 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. The following line of code makes the thread it is executing on a UI thread,

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

Thread Affinity

Any WPF object whose type extends `DispatcherObject` has what is known as thread affinity. It must be created on a UI thread and once created it has affinity with that UI thread. It can only be safely accessed from the user interface thread that created it.

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 that elements Dispatcher. The DispatcherObject type defines a Dispatcher 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.

The Application Class

Although not mandatory, most WPF executables will make use of an instance of the type Application .

> **One application per app domain**
>
> WPF ensures there is only Application per app domain. If we try to instantiate a second Application object in the same app domain WPF will throw an exception.

Appendices

Multiple UI Threads

```
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();
    }
}
```

Building a WPF Application

Simplest WPG Executable

First we show the source code of the simplest possible WPF application and the MSBuild file that takes it and links it into a WPF executable

Figure 1Simplest WPF Executable

```
using System;
using System.Windows;
using System.Windows.Threading;

public class MainClass
{
    [STAThread]
    private static void Main(String[] args)
    {
        Window w = new Window();
        w.Show();
        Dispatcher.Run();
    }
}
```

Listing 3SimplestWPF Exeutable Build

```
<?xml version="1.0" encoding="utf-8"?>
<Project  xmlns="http://schemas.microsoft.com/developer/msbuild/2003"
DefaultTargets="Build" ToolsVersion="4.5">
  <PropertyGroup>
    <AssemblyName>Kenny</AssemblyName>
    <PlatformTarget>AnyCPU</PlatformTarget>
    <OutputPath>.</OutputPath>
    <TargetFrameworkVersion>4.5</TargetFrameworkVersion>
  </PropertyGroup>
  <ItemGroup>
    <Compile Include="MainClass.cs"/>
    <Reference Include="System"/>
    <Reference Include="PresentationFramework"/>
    <Reference Include="PresentationCore"/>
    <Reference Include="WindowsBase"/>
    <Reference Include="System.xaml"/>
  </ItemGroup>
  <Target Name="Build">
    <Csc Sources="@(Compile)" References="@(Reference)"
    OutputAssembly="$(OutputPath)$(AssemblyName).exe" />
    </Target>
  <Import Project="$(MSBuildToolsPath)\Microsoft.CSharp.targets" />
</Project>
```

Standard Pattern

A standard WPF project setup inside visual studio will typically make sure the following structure exists. We need a xaml file for the application instance and a xaml file for the main window

Listing 4Application.xaml.cs

```
public partial class Application : System.Windows.Application {

}
```

Listing 5Application.xaml

```
<Application x:Class="Application"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             StartupUri="MainWindow.xaml">
    <Application.Resources/>
    </Application.Resources>
</Application>
```

Listing 6MainWindow.xaml.cs

```
public partial class MainWindow : System.Windows.Window
{
    public MainWindow()
    {
        InitializeComponent();
    }
}
```

Listing 7MainWindow.xaml

```
<Window x:Class="MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="MainWindow">
    <Button>Kenny</Button>
</Window>
```

The xml is this msbuild file cause the build to create the Application and the main window and wire them together . It also generated a main method

Listing 8csproj file

```
<?xml version="1.0" encoding="utf-8"?>
<Project  xmlns="http://schemas.microsoft.com/developer/msbuild/2003"
DefaultTargets="Build" ToolsVersion="4.5">
  <PropertyGroup>
    <AssemblyName>Kenny</AssemblyName>
    <PlatformTarget>AnyCPU</PlatformTarget>
    <OutputPath>.</OutputPath>
    <TargetFrameworkVersion>4.5</TargetFrameworkVersion>
  </PropertyGroup>
  <ItemGroup>
    <ApplicationDefinition Include="Application.xaml"/>
    <Page Include="MainWindow.xaml"/>
    <Compile Include="Application.xaml.cs">
      <DependentUpon>Application.xaml</DependentUpon>
      <SubType>Code</SubType>
    </Compile>
    <Compile Include="MainWindow.xaml.cs">
      <DependentUpon>MainWindow.xaml</DependentUpon>
      <SubType>Code</SubType>
    </Compile>
    <Reference Include="System"/>
    <Reference Include="PresentationFramework"/>
    <Reference Include="PresentationCore"/>
    <Reference Include="WindowsBase"/>
    <Reference Include="System.xaml"/>
  </ItemGroup>
  <Target Name="Build">
    <Csc Sources="@(Compile)" References="@(Reference)"
    OutputAssembly="$(OutputPath)$(AssemblyName).exe" />
    </Target>
  <Import Project="$(MSBuildToolsPath)\Microsoft.CSharp.targets" />
</Project>
```

Questions

What 3 libraries constitute the WPF framework?

PresentationCore

PresentationFramework

WindowsBase

What other DLL would we typically add for WPF development?

System.XAML

What is meant by modeless dialogue?

The dialogue does not block the main application until the user deals with the dialogue

How can we create a modeless dialogue in WPF?

Create a new window and set its owner property to the parent window and then use show

How can we modify a UI object on a thread other than the one that created it?

We can ask the UI object to give us its dispatcher and use that

What restrictions are there on the thread a WPF Window can run on?

A Win32 window must run inside a thread whose COM apartment state is ApartmentState.STA

How do we achieve this?

Mark the Main method with the System.STAThread attribute

In addition to being an STAThread what is needed for a thread to be a UI Thread?

It must have an associated message loop

What does the message loop do?

Windows input messages such as keystrokes and mouse presses are pushed onto the message queue by the OS

The message loop pulls them from the queue, processes them and dispatches them so handlers can respond to them

Give the code needed to make a thread executing it a message loop

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

In the context of WPF what is meant by thread affinity?

Any object which has DispatcherObject as a base must be created on a UI thread

It can only be safely accessed from the UI thread that create it

How can threads other than the UI thread interact with DispatcherObjects?

By asking it for its dispatcher and using that to dispatch methods