Lab 01: Async / Await + Tasks

Prerequisites

You will need a development environment, either a Mac or Windows PC with the Android SDK and Xamarin tools installed. We will be using the Android emulator to test the code we are building, so make sure to have a virtual device already configured and ready to run. See the `Xamarin.Android` setup documentation if you need help getting your environment setup:

http://docs.xamarin.com/guides/android/getting_started/installation/

Downloads

Download the starting solution and a completed version from http://university.xamarin.com.

Lab Goals

The goal of this lab will be to demonstrate what happens when the UI thread is busy doing non-UI work. We will be using an Android application, however an iOS application would behave exactly the same way. Once we've identified the issue, we will correct it using the asynchronous capabilities of the Xamarin.Android framework.

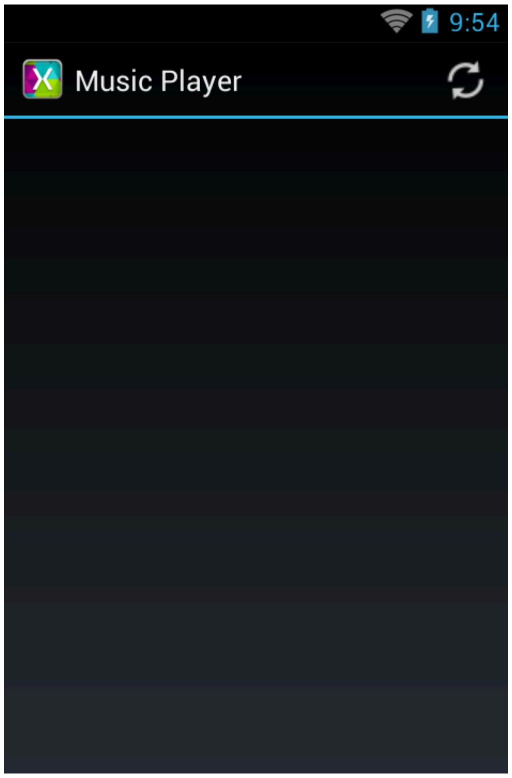
Steps

Open the Starting Solution

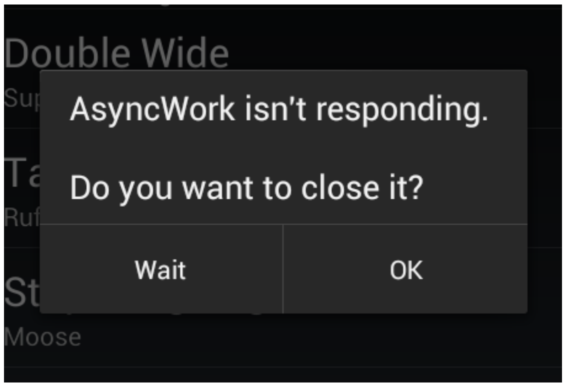
1. Launch Xamarin Studio and open the starting solution – `Lab1.AsyncWork.sln`.
2. Go ahead and run the application in the Android emulator or on a physical device.

> **Note**: We recommend you use the free Genymotion emulator, but the normal Android SDK emulator or a virtual machine will work as well.

3. It will present a very simple, empty UI with a refresh button in the action bar:



4. Tap the Refresh icon – notice the UI "freezes". If you attempt to work with it, it will get the dreaded "App Not Responding" dialog from Android:


AsyncWork isn't responding.

Do you want to close it?

5. It will eventually come back with data, but because it's loading that data using the main thread, the UI blocks until the data is ready to be displayed – which in this case takes several seconds.
6. Go ahead and shutdown the app.
7. The problem is that the songs are being loaded from a web service, but we are executing the fetch on the UI thread. Open the WebService.cs file and examine the `GetSongs` method. The call to `WebClient.DownloadString` is a synchronous call that is performed on the calling thread. We could move this to a task, but it turns out that most of the I/O related classes in .NET have async methods that match the synchronous ones.
8. Locate the comment `// TODO: Step 1 - add async version of method`

9. Uncomment the code that follows – this implements an async version of the GetSongs method. By convention, it is named GetSongsAsync. Notice that it returns a `Task` and is marked with the `async` keyword which turns on the async language support.
10. Next, let's use this method in our MainActivity.cs – locate the comment `TODO: Step 2 - call the async version of the service.` and uncomment the line that follows it – you will want to comment out the original line that calls `GetSongs` right before it as well.
11. Try to compile the code – you should get a compile error because of the `await` keyword. This is the other half of the C# async support – in order to use the keyword you need to decorate the method with the `async` keyword as shown below:

```
async void OnRefresh()
{
    try
    {
```

12. Run the application again – notice that now the UI thread continues to process events as evidenced by the progress bar running along the top. Watch the progress bar carefully. Notice that it pauses briefly just before the songs are displayed.
13. Go examine the GetSongsAsync method you just uncommented. Notice that it now loads the songs from the web service asynchronously, however it is parsing the JSON file into an object graph using a synchronous call!
14. In this case, we don't have an async form of the method – so now let's turn to our second tool, the Task object. Find the comment `TODO: Step 3 - add improved async version of method` and uncomment the method that follows. This is an improved form of the GetSongsAsync method, which takes the returned JSON data and then creates a Task to parse it into an object graph.
15. Change the MainActivity to call your new method:

```
var songs = await WebService.BetterGetSongsAsync();
```

16. Run the app one last time to verify that it now runs smoothly as it pulls and parses the data.

Summary

In this lab, we examined a common issue in UI development – keeping the device responsive while it processes various activities. In this instance we downloaded some data from a web service and then parsed it in order to display into the UI.