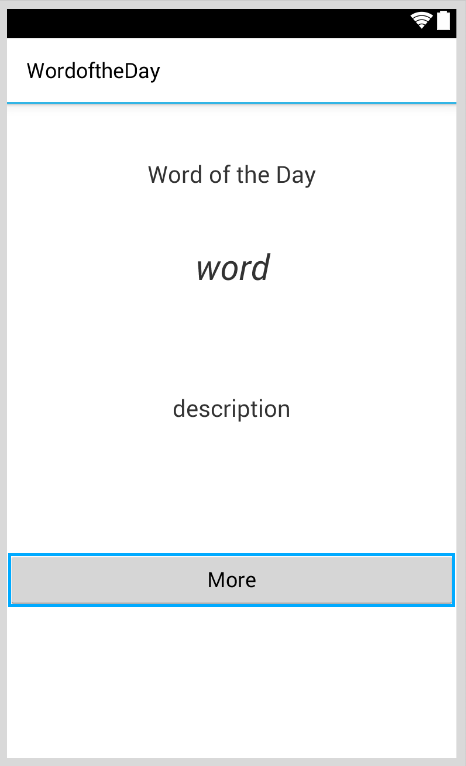
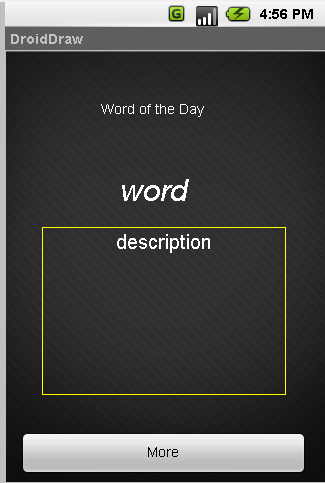
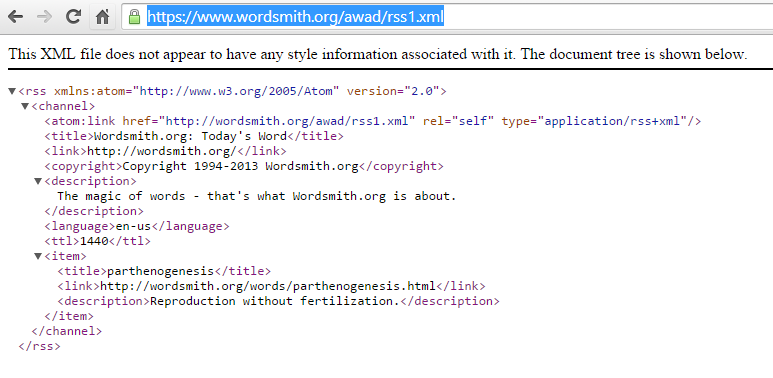
Learning Outcomes

1. Handling REST api calls
2. Understanding asynchronous operation using async and await



API endpoint

**https://www.wordsmith.org/awad/rss1.xml**



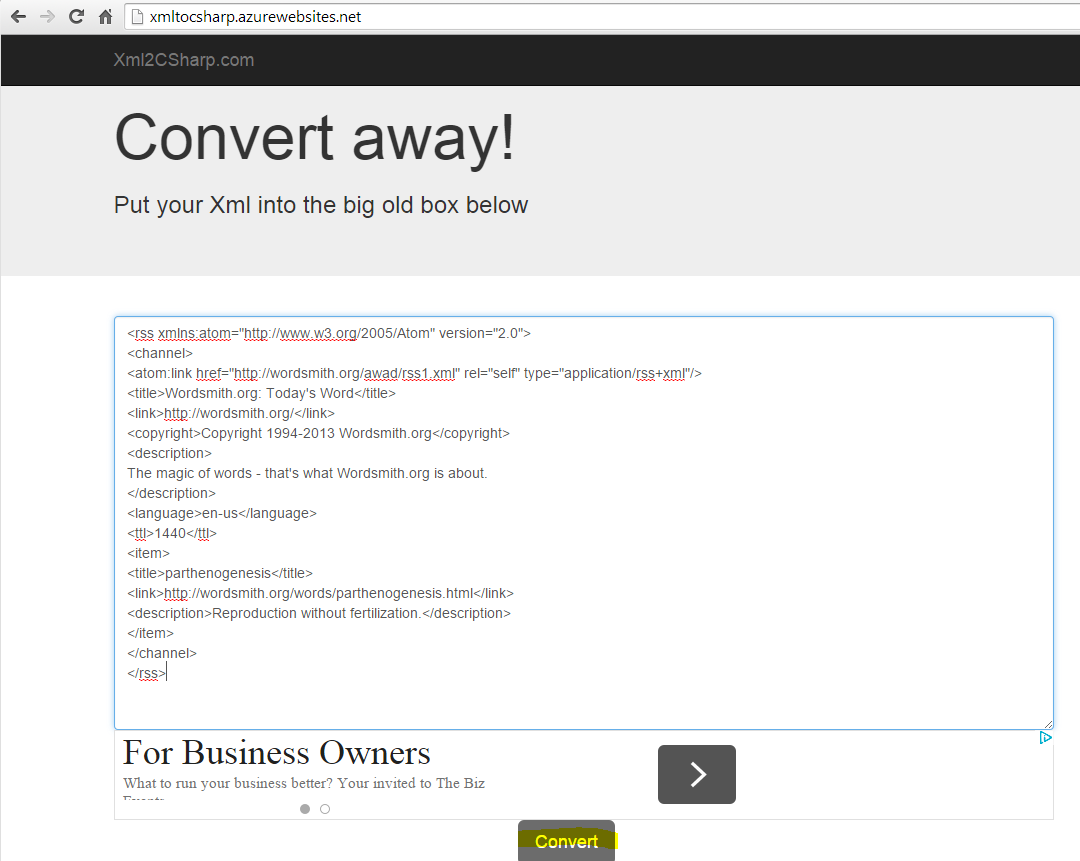
**Deserializing the XML**

Serialization is a process by which an object's state is transformed in some serial data format, such as XML or binary format. Deserialization on the other hand is used to convert the byte of data, such as XML or binary data, to object type. Serialization is the process of converting an object into a form that can be readily transported. For example, you can serialize an object and transport it over the Internet using HTTP between a client and a server. On the other end, deserialization reconstructs the object from the stream. XML serialization results in strongly typed classes with public properties and fields that are converted to a serial format (in this case, XML) for storage or transport

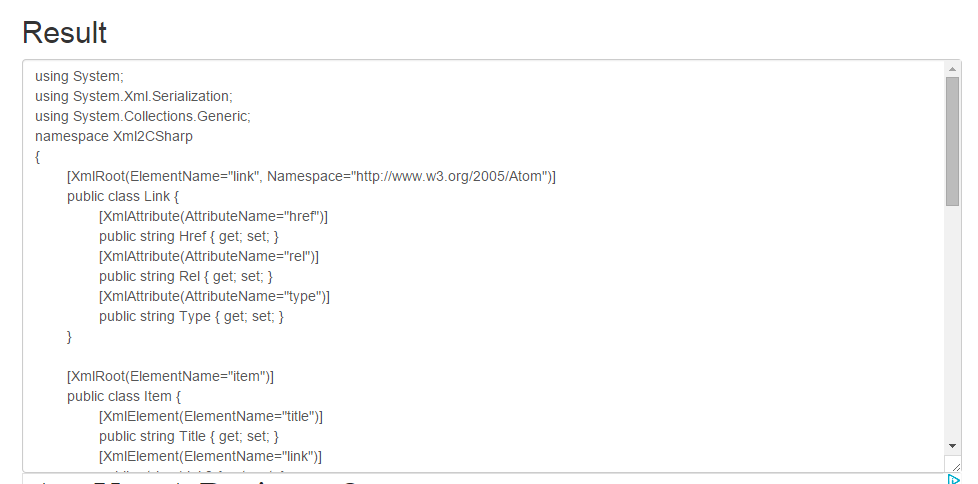
To deserialize the XML we need to create a class that maps the XML document we want to read. There is an easier way to do this.

Go to Google and type XML to C# or go to [**xml**tocsharp.azurewebsites.net](xmltocsharp.azurewebsites.net)

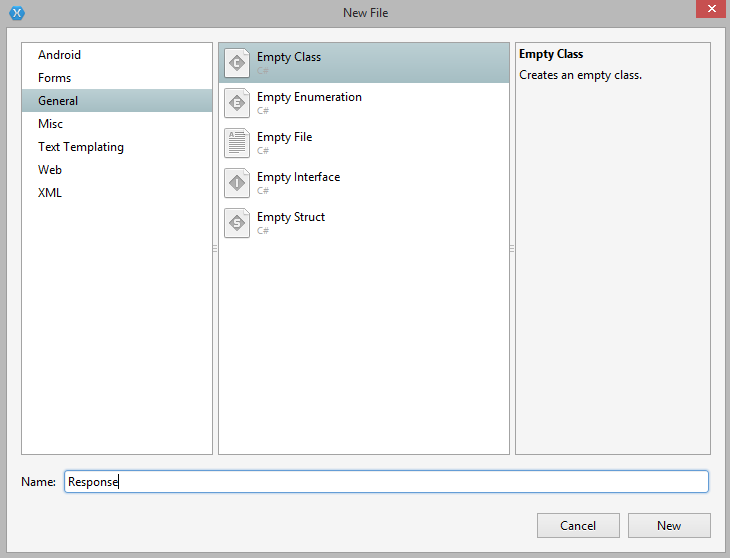
Paste your XML file and Hit convert



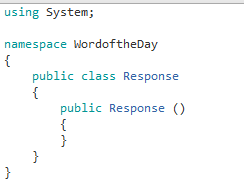
You get the result as shown below



Create a new class file and name it Response.cs



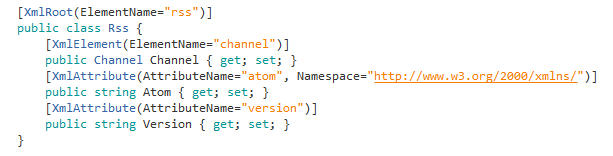
Delete the default class generated



And paste the classes generated from the XML.

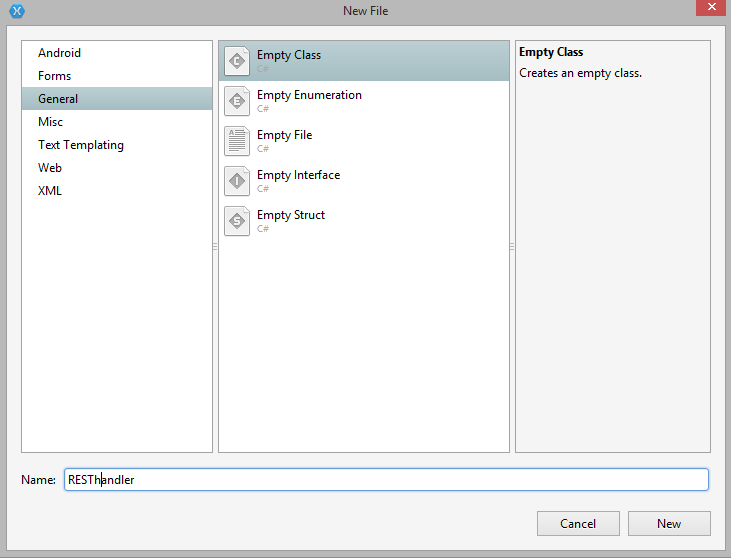


It’s important to understand that our root class is rss which has a channel class as its sub class. The channel class has further subclasses as link and Item.

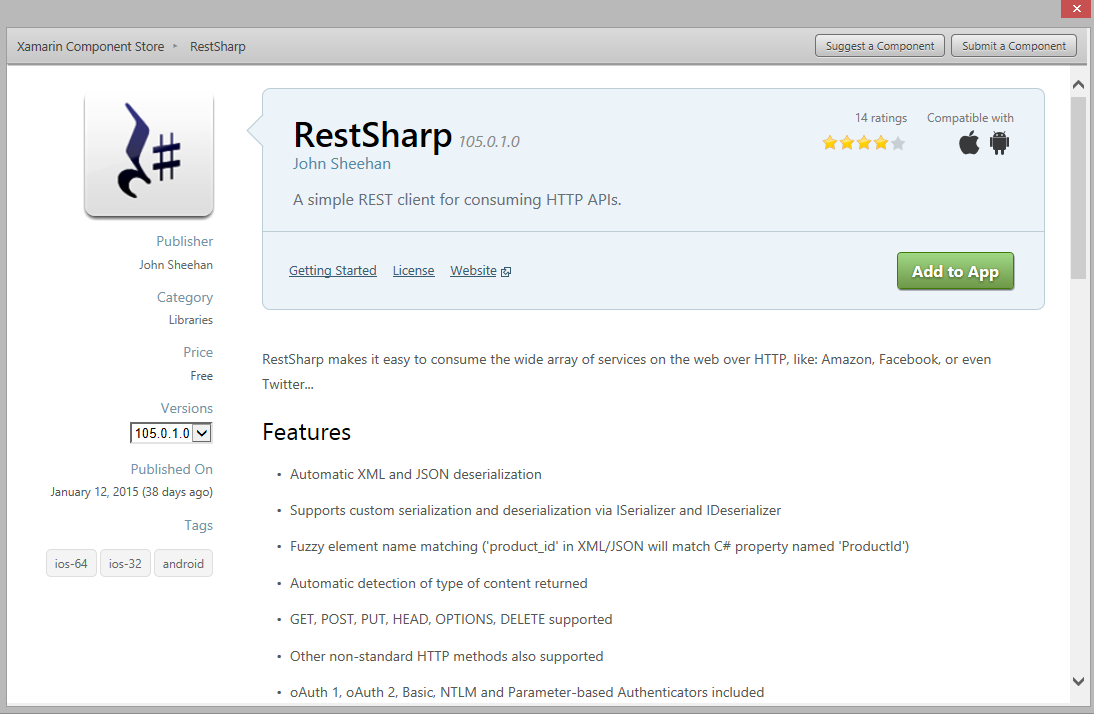


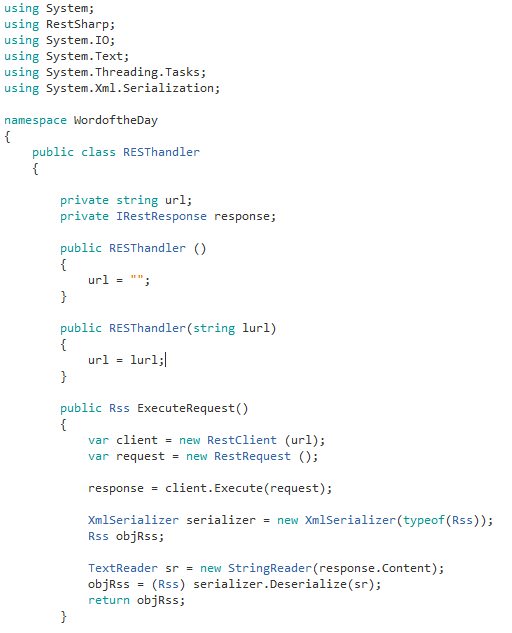
We create another class that will send the RESTRequest and deserialize the RESTResponse.

We will call it the RESTHandler.cs.

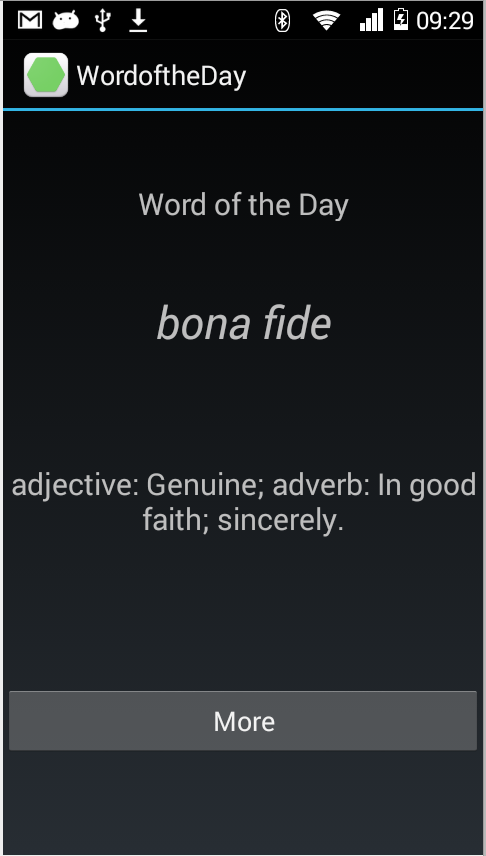


We need to add the RESTSharp component to handle the REST API call.









The problem with this approach is that the entire UI is unresponsive (does not load) until the word is fetched from the RESTcall. However we would like the UI to be responsive and the fetching of the REST call to happen in the background.

The latest version of the C# language – version 5 – introduced two new keywords to express asynchronous operations: **async and await**. These keywords let you write simple code that utilizes the Task Parallel Library to execute long running operations (such as network access) in another thread and easily access the results on completion.

**Using async & await**

async and await are new C# language features that work in conjunction with the Task Parallel Library to make it easy to write threaded code to perform long-running tasks without blocking the main thread of your application.

**async**

**Declaration**

The async keyword is placed in a method declaration (or on a lambda or anonymous method) to indicate that it contains code that can run asynchronously, ie. not block the caller’s thread.

A method marked with async should contain at least one await expression or statement. If no awaits are present in the method then it will run synchronously (the same as if there were no async modifier). This will also result in a compiler warning (but not an error).

**Return Types**

An async method should return a Task, Task<TResult> or void.

Specify the Task return type if the method does not return any other value.

Specify Task<TResult> if the method needs to return a value, where TResult is the type being returned (such as an int, for example).

The void return type is used mainly for event handlers which require it. Code that calls void-returning asynchronous methods can’t await on the result.

**Parameters**

Asynchronous methods cannot declare ref or out parameters.

**await**

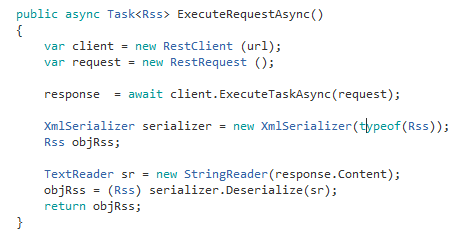
The await operator can be applied to a Task inside a method marked as async. It causes the method to stop execution at that point and wait until the task completes.

Using await does not block the caller’s thread – rather control is returned to the caller. This means that the calling thread is not blocked, so for example the user interface thread would not be blocked when awaiting a task.

When the task completes, the method resumes executing at the same point in the code. This includes returning to the try scope of a try-catch-finally block (if one is present). await cannot be used in a catch or finally block.

**Async Version of Word of the Day**

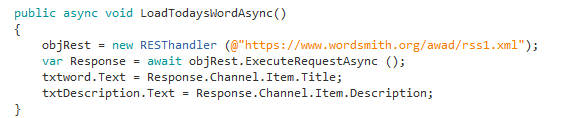
We write an Async version of ExecuteRequest in RestHandler.cs



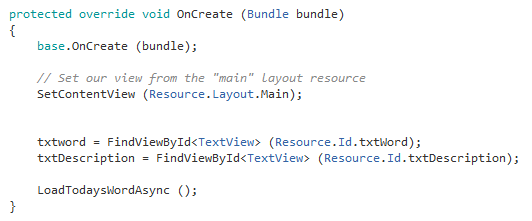
The function must return void or type Task<T>. It should contain a await keyword else it runs synchronously.

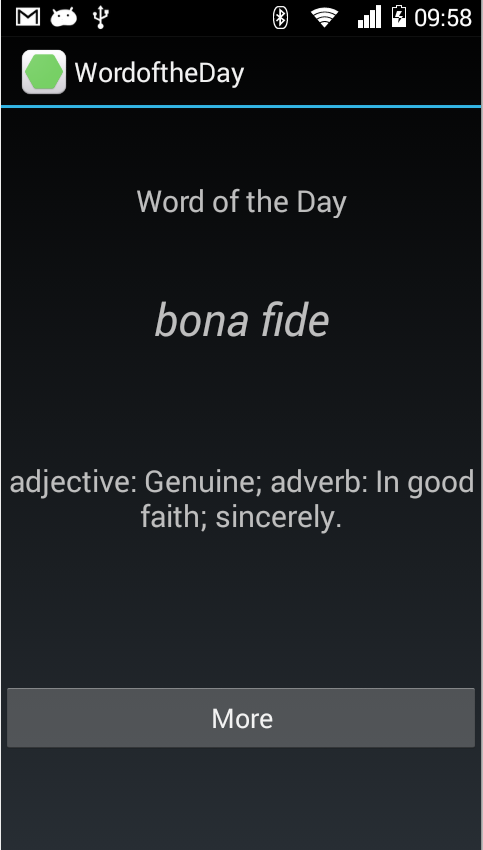
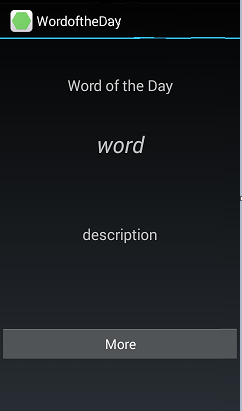
We similarly write an async version to LoadTodaysWord in MainActivity.cs

**MainActivity.cs**



Modify your OnCreate to call LoadTodaysWordAsync()





**Exercise**

1. Try adding a progress bar before the word loads (search for the component ANDHUD )
2. On click of the More button open up a WebView and display the results