Web APIs

Class Notes

Mobile apps frequently have to connect to a web-based API - that is, send some coded message to request information from a server and then decode the response (delivered in XML or JSON). It may be an API that you have developed yourself as a web back end or a component from a third party. For example, if you wanted to create a Twitter app (something Twitter would currently prefer you not to do), you would have to hook into Twitter's API.

Requesting information from a Web API is generally done with a simple URL. For example, this URL is a call to the Open Weather Map API to retrieve the weather for London in XML format.

http://api.openweathermap.org/data/2.5/weather?q=London&mode=xml&APPID=1f9a24c86c0b6155d4126ccb7ee6e61a

Okay, maybe "simple" was not the right word for it. The first part - before the question mark - is the actual URL. The remaining text is various options and information being passed to the API to let it know what we want back. In this case, we want the weather for London in XML format and we also provide an App ID so the API knows who we are and that we're allowed to access it.

If you click the click (or copy into the address bar in your browser), you'll see the information you get back. To use this in an app, you have to request the same information using the same URL in your C# code and then extract the information you want from the XML or JSON you get back.

Note that APIs generally charge a small amount for their use but they also usually have some basic free tier that allows you to test them with your app. The free tier is usually limited in some way (say, only 100 requests an hour) so you have to be careful.

HTTP Requests

To make a API request using a URL, you need to include a built in HTTP library with the following using statement…

```
using System.Net.Http;
```

To make and receive a HTTP request, do this…

```
public async void GetAPI ()
{
	var client = new HttpClient();
	var response = await client.GetAsync("https://www.whateveryourURLis.com");
	var responseString = await response.Content.ReadAsStringAsync();
}
```

The first line creates a HTTPClient class, which does all the work for us.

The second line uses the HTTPClient to make the request. The response is a HTTPResponseMessage object that contains the result, any errors and so forth.

The third line retrieves whatever it was we downloaded as a string. If it was an API call, it should be some JSON or XML. If it was a webpage, you'll get the HTML code.

Note that we are using asynchronous methods (as you would expect for something as slow as internet access).

Error Checking

Whenever your program touches any radio connectivity - Bluetooth, GPS or, in this case, the internet - then there are always a large amount of different problems and ways it can fail. You could be having trouble with drivers, network hardware, interference, internet access, CDNs or the website could just be down. Your program needs to account for as many of these as possible and fail gracefully, preferably with an error message to the user so they know something is wrong.

The HTTP response has a status code which will allow you to catch some of these errors. It's also a good idea to check to see if the response has anything in it…

```
var response = await client.GetAsync("https://www.whateveryourURLis.com");

if (response.StatusCode != System.Net.HttpStatusCode.OK ||
    response.Content == null)
{
      await DisplayAlert("Error", string.Format("Response contained status code:
      {0}", response.StatusCode), "OK");
      return;
}
```

Note that this is *not* a good error message for users. It's geeky and unhelpful. However, it allows you to see the StatusCode value and is useful for debugging. In a shipping product, you should be clearer and less technical.

It also doesn't fix all the errors. The remaining ones will trigger an exception so you will also need to use try-catch statements.