XAM150 – WEB SERVICES LAB

Connecting to REST Web Services

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.

Downloads

Included with this lab document is a folder with resources that you will need in order to complete the lab. The folder name is **Fundamentals Web Services Resources**. Make sure you have this folder before you begin.

Lab Goals

This goal of this lab will be to introduce REST web services and how they can be integrated into Xamarin.Android and Xamarin.iOS applications. For our lab today, we will use Xamarin.Android, but there is an accompanying Xamarin.iOS project included as well. During the process, you will become familiar with the tools within Xamarin Studio to integrate REST web services, creating a REST web service client and consuming the data within your application. For this lab, we will be integrating with a web service that provides pharmaceutical drug information. Additional information on the web service can be found at the [RxNav site](#).

The lab has been provided as a starter solution with most of the code already filled in for you – as you following along with the instructor you will make small changes for each step, either writing a little code or uncommenting a block of code. Most of these steps are clearly marked in the supplied solution with `//TODO:` comments. These comments are picked up by Xamarin Studio and shown in the Task Pad, which you can make visible either by clicking the Tasks button in the status bar of the application, or through the **View > Pads > Tasks** menu item. When the Tasks Pad is open, it will look like this:

| Line	Description	File	Path
15	TODO: Step 1 - Create a REST client	RestClient.cs	Client
28	TODO: Step 2 - Call the web service and retrieve data	RestClient.cs	Client
42	TODO: Step 3 - Convert JSON into model objects	RestClient.cs	Client
14	TODO: Step 4 - Create an activity to consume the data	RestActivity.cs	Activities

Steps

Open the Starter Solution

1. Launch **Xamarin Studio** using Spotlight or the application icon

2. Click **Open…** on the Xamarin Studio Welcome Screen and navigate to the **Fundamentals Web Services Resources** folder included with this document

3. Locate the **Xamarin_WebService_REST_Begin** folder – make sure it's the starter and not the completed folder
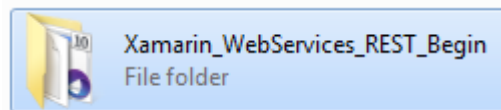
4. Inside the **Xamarin_WebServices_REST_Begin** folder you will find a `Xamarin.WebServices.Rest.sln` file – double click on this file to open the starter solution:
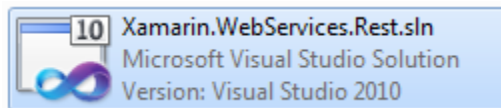

5. Go ahead and build and run the application in the emulator to make sure it compiles and your environment is ready. Let the instructor know if you have any trouble.

Adding a REST Web Service

We are going to add a **REST Web Service** to the application and build a client to display the results within a ListView in our app.

1. In **Xamarin Studio**, locate the project named **Xamarin.WebServicesAsync.Rest.Droid**
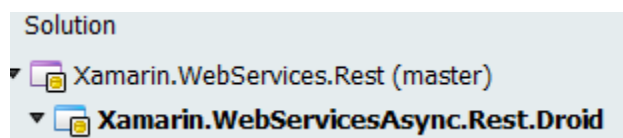
2. Open the **Client\RestClient.cs** source file and locate the comment: `//TODO:`

   `Step1     –     create     a     REST     client.`

   Uncomment the code below to create a new instance of the REST client

```
private const string
    RestServiceBaseAddress = "http://rxnav.nlm.nih.gov/REST/",
    AcceptHeaderApplicationJson = "application/json";

private HttpClient CreateRestClient(){
    var client = new HttpClient(){ BaseAddress = new Uri(RestServiceBaseAddres
s) };

    client.DefaultRequestHeaders.Accept.Add (MediaTypeWithQualityHeaderValue.P
arse(AcceptHeaderApplicationJson));

    return client;
}
```

3. Locate the comment: `//Step 2 - Call the web service and retrieve data.`
   Uncomment the code below to create a new client, call the web service and parse the results

```
public async Task<IEnumerable<Model.ConceptProperty>> GetDataAsync(){
    using (var client = CreateRestClient ()) {
        var getDataResponse = await client.GetAsync ("drugs?name=aspirin", Htt
pCompletionOption.ResponseContentRead);

        //If we do not get a successful status code, then return an empty set
        if (!getDataResponse.IsSuccessStatusCode)
            return Enumerable.Empty<Model.ConceptProperty> ();

        //Stream the response into a JSON Value object
        var jsonResponse = JsonValue.Load(await getDataResponse.Content.ReadAs
StreamAsync ());

        //TODO: Step 4 - call to map JSON to model object
        //return MapJsonToConceptProperty (jsonResponse);
    }
}
```

4. Next, locate the comment: `//TODO: Step 3 - Convert JSON into model objects.`
   Uncomment the code below to map our DTO objects returned from the REST client to a model object.

```
private IEnumerable<Model.ConceptProperty> MapJsonToConceptProperty(JsonValue
json)
{
```

```
    var conceptProperties = new List<Model.ConceptProperty>();

    if(json != null && json.ContainsKey("drugGroup")){
        var drugGroup = json ["drugGroup"];

        if (drugGroup != null && drugGroup.ContainsKey("conceptGroup")){
            foreach (JsonValue conceptGroup in drugGroup["conceptGroup"]) {
                if(conceptGroup.ContainsKey("conceptProperties")){
                    foreach (JsonValue conceptProperty in conceptGroup["concep
tProperties"]) {
                        conceptProperties.Add(
                            new Model.ConceptProperty(){
                                //TODO: Step 4 - Map the properties from our D
TO to custom object
                            }
                    }
                }
            }
        }
    }

    return conceptProperties.OrderBy(cp => cp.Synonym);
}
```

> **Tip:** It is recommended to map your DTO (Data Transfer Objects) from the web service to a local object/model definition. 3rd party services can often change the definitions of their messages and by mapping to a local object, we can minimize the impact that it will have on our apps.

5. Locate the comment: `TODO: Step 4 - Map the properties from our DTO to custom                                                     object.`
   Add in the following property mappings to the **Model.ConceptProperty** object

```
conceptProperties.Add(
    new Model.ConceptProperty(){
        //TODO: Step 4 - Map the properties from our DTO to custom object
        Name = conceptProperty.ContainsKey("name")
                ? (string)conceptProperty["name"]
                : String.Empty,
        Synonym = conceptProperty.ContainsKey("synonym")
                ? (string)conceptProperty["synonym"]
                : String.Empty
    }
```

6. Locate the comment: `TODO: Step 5 - call to map JSON to model object.`
   Uncomment the following line to map the JSON object to our model object

```
return MapJsonToConceptProperty (jsonResponse);
```

7. **Save** the project

8. **Build** the project and ensure that there are no errors

9. Open the **Activities\RestActivity.cs** source file and locate the comment: `//`
   `TODO: Step 6 - Create an activity to consume the data`
   Uncomment the code below to associate a layout and an adapter to the
   activity

```csharp
Adapters.ConceptPropertyAdapter adapter;

protected override void OnCreate(Bundle bundle)
{
    base.OnCreate(bundle);

    //Set our activity's layout
    SetContentView (Resource.Layout.list_with_spinner);

    //Assign an adapter
    adapter = new Adapters.ConceptPropertyAdapter(this, Enumerable.Empty<Model
.ConceptProperty>());
    ListView.Adapter = adapter;
}
```

10. Locate the comment: `// Step 7 - Call the services using async and update`
    `the                     UI                     with                     the.`
    Uncomment the code below to create a client and query for results. The
    results of the query will be added to the adapter and the adapter will notify
    that new data has been retrieved.

```csharp
public async Task LoadDataAsync(){
    //Remove any items which may be in the list
    adapter.ConceptProperties.Clear ();
    adapter.NotifyDataSetChanged ();

    //Create a REST client
    var restClient = new Client.RestClient ();
    //Query
    var foundConceptProperties = await restClient.GetDataAsync ();

    //Assign response to our adapter and notify of updates
    adapter.ConceptProperties.AddRange (foundConceptProperties);
    adapter.NotifyDataSetChanged ();
}
```

11. Locate the comment: `// Step 8 - Make the call to load the data.`

```csharp
protected override void OnResume ()
{
    base.OnResume ();

    //TODO: Step 8 - Make the call to load the data
    LoadDataAsync ();
}
```
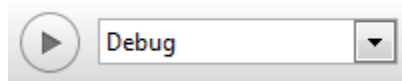
> **Tip:** Be mindful when calling methods with the async modifier from a synchronous method. These will *fire and forget*, so the order of operation is important.
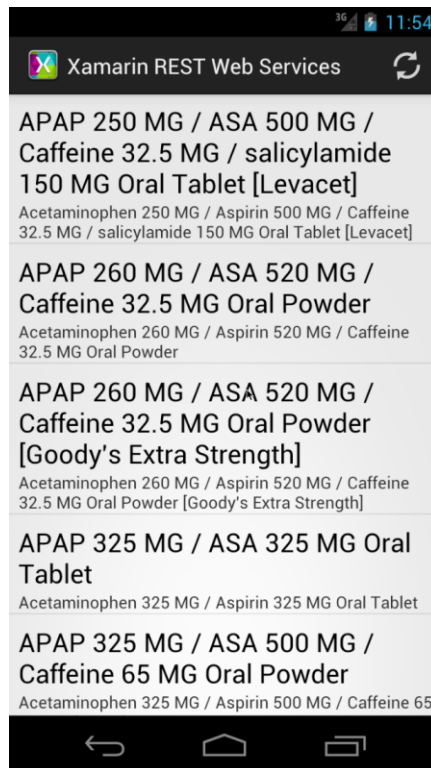
12. **Save** and **Build** the project and ensure that there are no errors
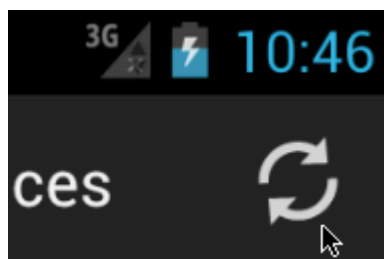
Testing the REST Web Services Client

1. Ensure your Android Emulator is running and from **Xamarin Studio, Debug** or **Run** the app

2. The application will automatically query and load the results into the list view

3. Click the **Refresh** button in the action bar to reload the results

Summary

Congratulations! You learned how to build clients that can connect to REST-based web services. We also learned how to call those services in a way that keeps the user interface responsive.