## FUNDAMENTALS LESSON 09 – WEB SERVICES LAB

## Connecting to SOAP 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.

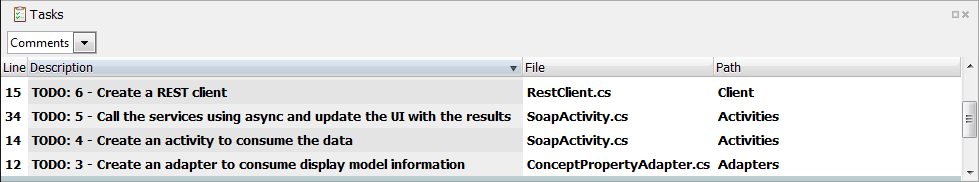
### Downloads

Included with this lab document is a folder with resources that you will need in order to complete the lab. The folder name is **Lab 09 Resources**. Make sure you have this folder before you begin.

### Lab Goals

This goal of this lab will be to introduce SOAP web services and how they can be integrated into Xamarin.Android and Xamarin.iOS applications. During the process, you will become familiar with the tools within Xamarin Studio to integrate SOAP web services, creating a SOAP 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](http://rxnav.nlm.nih.gov/APIsOverview.html).

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:



## 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 **Lab 09 Resources** folder included with this document
3. Locate the **Xamarin\_WebService\_Starter** folder – make sure it’s the starter and not the completed folder



1. Inside the Xamarin\_WebServices\_Starter folder you will find a Xamarin.WebServices.sln file – double click on this file to open the starter solution:



1. 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.

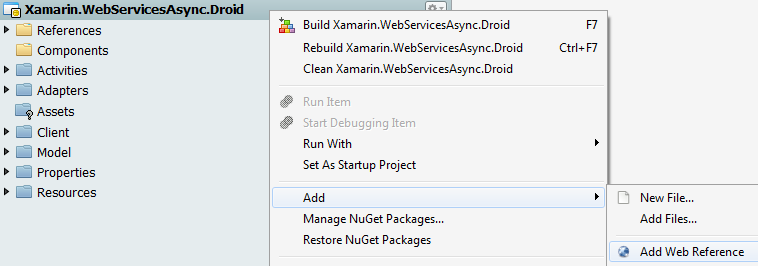
### Adding a SOAP Web Service to Xamarin.Android

We are going to add a **SOAP 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.Droid**



1. Right click on the project and select **Add > Add Web Reference**



1. An **Add Web Reference** wizard will appear
2. In the **Web Service Url** field, add the following url:  
   <http://rxnav.nlm.nih.gov/RxNormDBService.xml>



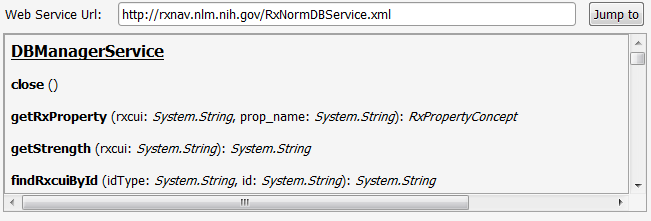
1. In the **Framework** dropdown, select **.Net 2.0 Web Services**



1. Click the **Jump To** button



1. Notice that after clicking the **Jump To** button, the results pane was populated with information about the web service. If you did not receive any response from the web service, please verify that the Web Service Url was configured correctly and notify the instructor



1. Change the **Reference** field to use the value **RxNav**



1. Select the **OK** button



1. A new folder named **Web References** will be added to the project and there will be a new entry named **RxNav** within it  
   
2. Open the **Client\SoapClient.cs** source file and locate the comment: //TODO: 1 - create a SOAP client. Go ahead and uncomment the code below

|  |
| --- |
| *//***TODO***: 1 - create a SOAP client* public async Task<IEnumerable<Model.ConceptProperty>> GetDataAsync(){     return await Task.Run (() => {         using(var soapClient = new DBManagerService ()){             *//Query for the drug named "aspirin"*             return SoapDtoToConceptProperty(soapClient.getDrugs("aspirin"));             }     }); } |

1. Next, locate the comment: //TODO: 2 - Convert the SOAP DTOs into model objects. Go ahead and uncomment the code below

|  |
| --- |
| *//***TODO***: 2 - Convert the SOAP DTOs into model objects* private IEnumerable<Model.ConceptProperty> SoapDtoToConceptProperty(RxConceptGroup[] rxConceptGroups){      var parsedConceptProperties = new List<Model.ConceptProperty>();      foreach (var conceptGroup in rxConceptGroups) {         foreach (var concept in conceptGroup.rxConcept) {             parsedConceptProperties.Add(                 new Model.ConceptProperty(){                 }             );         }     }      return parsedConceptProperties.OrderBy(cp => cp.Synonym); } |

1. Modify the **Model.ConceptProperty** declaration and add mappings for the following properties

|  |
| --- |
| new Model.ConceptProperty(){     Name = concept.STR,     Synonym = concept.SY } |

1. **Save** the project



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



1. Open the **Activities\SoapActivity.cs** source file and locate the comment: // TODO: 3 - Create an activity to consume the data. Uncomment the code below

|  |
| --- |
| Adapters.ConceptPropertyAdapter adapter;  protected override void OnCreate(Bundle bundle) {     base.OnCreate(bundle);      SetContentView (Resource.Layout.list\_with\_spinner); }  protected override void OnResume () {     base.OnResume ();      PrepareActivityAsync (); } |

1. Locate the comment: // 4 - Call the services using async and update the UI with the results. Uncomment the code below

|  |
| --- |
| public async Task PrepareActivityAsync(){     adapter.ConceptProperties.Clear ();     adapter.NotifyDataSetInvalidated ();      var soapClient = new Client.SoapClient ();      adapter.ConceptProperties.AddRange (await soapClient.GetDataAsync ());     adapter.NotifyDataSetChanged (); } |

1. **Save** the project

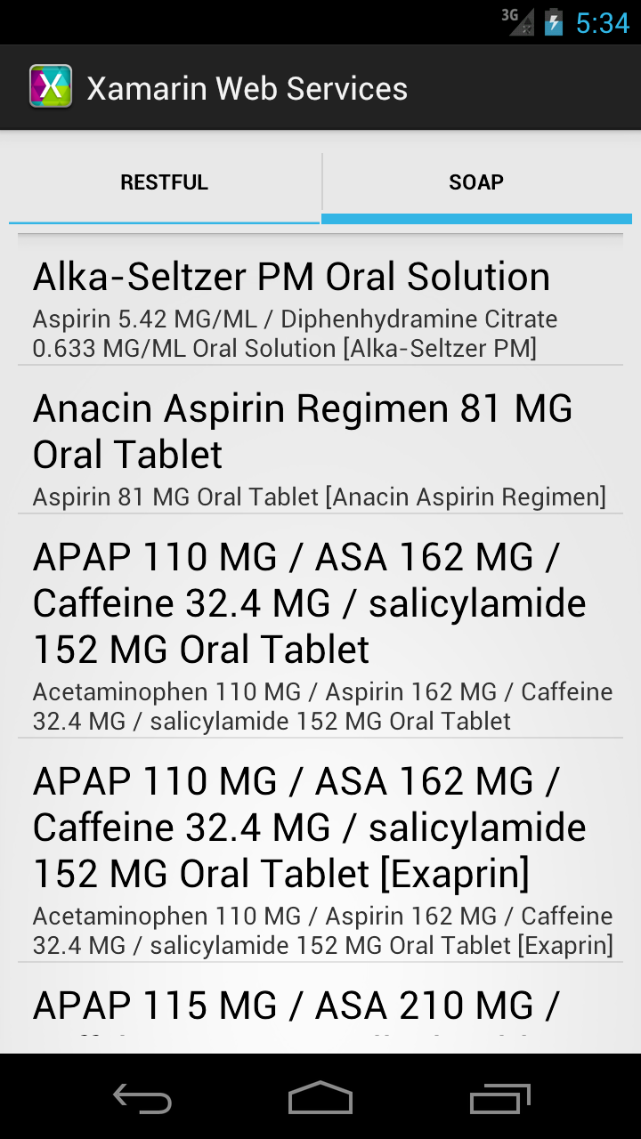


1. **Build** the project and ensure that there are no errors



### Testing the SOAP Web Services Client

1. Click the Play button to build and run our app.
2. Click the **SOAP** tab to view the results of the SOAP web service call



### Adding a RESTful Web Service to Xamarin.Android

We are going to add a **RESTful 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.Droid**



1. Open the **Client\RestClient.cs** source file and locate the comment: //TODO: 5 - Create a REST client and uncomment the code below

|  |
| --- |
| private const string      RestServiceBaseAddress = "http://rxnav.nlm.nih.gov/REST/",     AcceptHeaderApplicationJson = "application/json";  private HttpClient CreateRestClient(){     var client = new HttpClient(){ BaseAddress = new Uri(RestServiceBaseAddress) };      return client; } |

1. In the line after the **client** object is declared, add in the following code

|  |
| --- |
| client.DefaultRequestHeaders.Accept.Add (MediaTypeWithQualityHeaderValue.Parse(AcceptHeaderApplicationJson)); |

1. Locate the comment: //TODO: 6 - Call the web service and retrieve data and uncomment the code below

|  |
| --- |
| public async Task<IEnumerable<Model.ConceptProperty>> GetDataAsync(){     using (var client = CreateRestClient ()) {         var getDataResponse =  await client.GetAsync (  "drugs?name=aspirin", HttpCompletionOption.ResponseContentRead);          if (!getDataResponse.IsSuccessStatusCode)             return Enumerable.Empty<Model.ConceptProperty> ();          var jsonResponse =  JsonValue.Load(await getDataResponse.Content.ReadAsStreamAsync ());          return JsonToConceptProperty (jsonResponse);     } |

1. Locate the comment: //TODO: 7 - Convert JSON into model objects and uncomment the code below

|  |
| --- |
| private IEnumerable<Model.ConceptProperty> JsonToConceptProperty(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["conceptProperties"]) {                         conceptProperties.Add(                             new Model.ConceptProperty(){                             });                     }                 }             }         }     }      return conceptProperties.OrderBy(cp => cp.Synonym); } |

1. Modify the **Model.ConceptProperty** declaration and add in the following properties

|  |
| --- |
| new Model.ConceptProperty(){     Name = conceptProperty.ContainsKey("name")  ? (string)conceptProperty["name"] : String.Empty,     Synonym = conceptProperty.ContainsKey("synonym")  ? (string)conceptProperty["synonym"] : String.Empty }); |

1. **Save** the project



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



1. Open the **Activities\RestActivity.cs** source file and locate the comment: // TODO: 8 - Create an activity to consume the data. Uncomment the code below

|  |
| --- |
| Adapters.ConceptPropertyAdapter adapter;   protected override void OnCreate(Bundle bundle) {     base.OnCreate(bundle);      SetContentView (Resource.Layout.list\_with\_spinner);      adapter = new Adapters.ConceptPropertyAdapter(this, Enumerable.Empty<Model.ConceptProperty>());     ListView.Adapter = adapter; }  protected override void OnResume () {     base.OnResume ();      PrepareActivityAsync(); } |

1. Locate the comment: // 9 - Call the services using async and update the UI with the results. Uncomment the code below

|  |
| --- |
| private async Task PrepareActivityAsync(){     adapter.ConceptProperties.Clear ();     adapter.NotifyDataSetInvalidated ();      var restClient = new Client.RestClient ();      adapter.ConceptProperties.AddRange (await restClient.GetDataAsync ());     adapter.NotifyDataSetChanged (); } |

1. **Save** the project

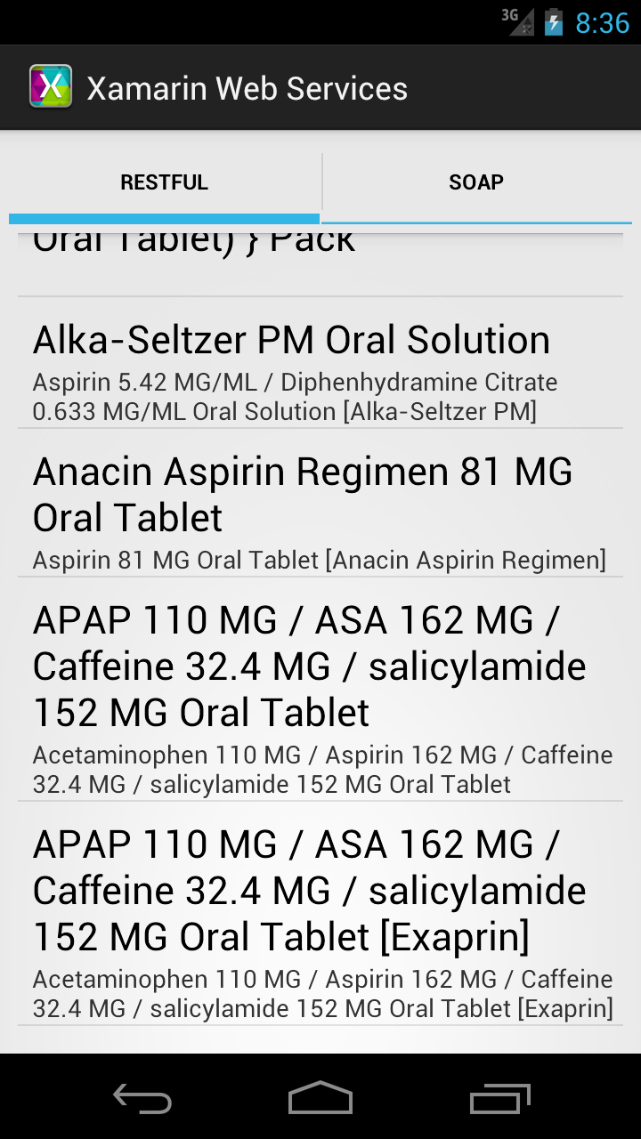


1. **Build** the project and ensure that there are no errors



### Testing the REST Web Services Client

1. Click the Play button to build and run our app.
2. Click the **RESTful** tab to view the results of the RESTful web service call



## Summary

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