XAM150 – 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.

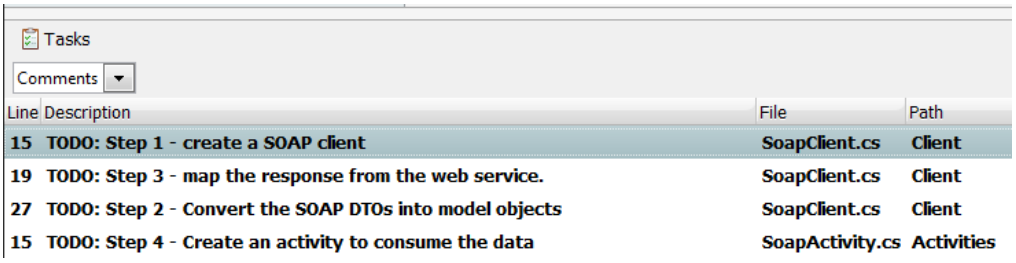
Downloads

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

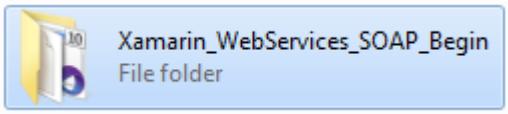
The lab has been provided as a starter solution with most of the code already filled in for you – as you following along with the instructor you will make small changes for each step, either writing a little code or uncommenting a block of code. Most of these steps are clearly marked in the supplied 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:



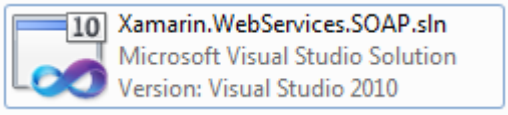
Steps

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 **Fundamentals Web Services Resources** folder included with this document
3. Locate the **Xamarin_WebService_SOAP_Begin** folder – make sure it's the starter and not the completed folder



4. Inside the **Xamarin_WebServices_SOAP_Begin** folder you will find a Xamarin.WebServices.Soap.sln file – double click on this file to open the starter solution:

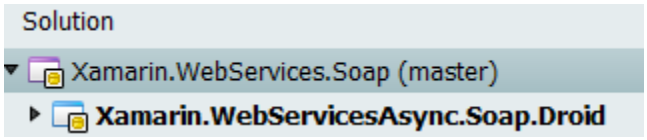


5. Go ahead and build and run the application in the emulator to make sure it compiles and your environment is ready. Let the instructor know if you have any trouble.

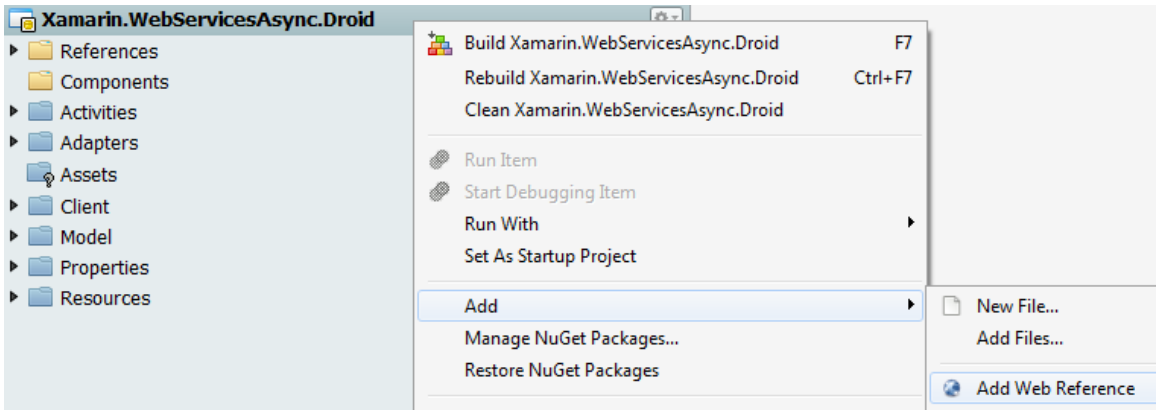
Adding a SOAP Web Service

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.

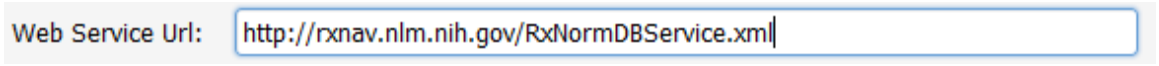
1. In **Xamarin Studio**, locate the project named **Xamarin.WebServicesAsync.Soap.Droid**



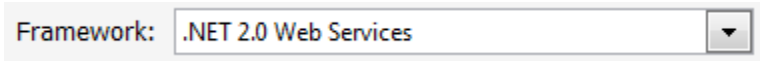
2. Right click on the project and select **Add > Add Web Reference**



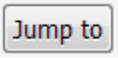
3. An **Add Web Reference** wizard will appear
4. In the **Web Service Url** field, add the following url: http://rxnav.nlm.nih.gov/RxNormDBService.xml



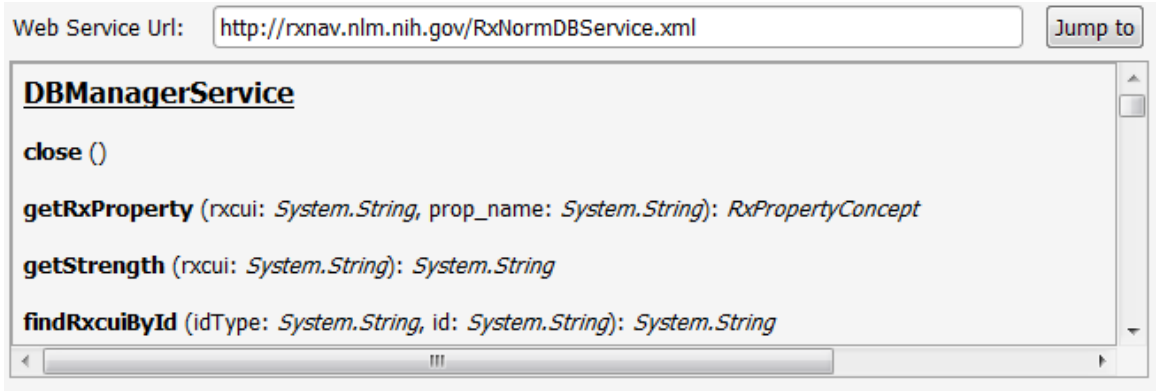
5. In the **Framework** dropdown, select **.Net 2.0 Web Services**



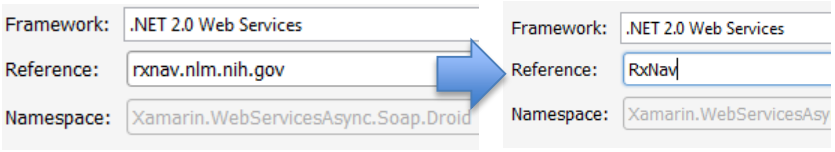
6. Click the **Jump To** button



7. 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

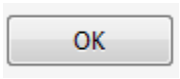


8. The Reference field will automatically populate with the value **rxnav.nlm.nih.gov**, change the **Reference** field to use the value **RxNav**

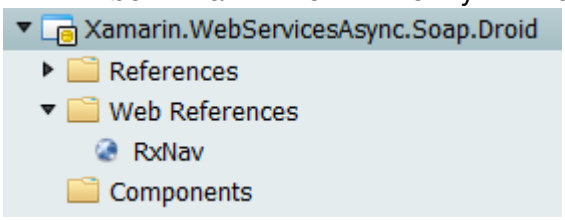


> **Tip:** The value in the Reference field will be appended to your default namespace. Try to tidy it up, if the generated namespace is too verbose or not descriptive enough.

9. Select the **OK** button



10. A new folder named **Web References** will be added to the project and there will be a new entry named **RxNav** within it



11. Open the **Client\SoapClient.cs** source file and locate the comment: //TODO: Step1 - create a SOAP client.

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

```
//TODO: Step1 - create a SOAP client
        public async Task<IEnumerable<Model.ConceptProperty>> GetDataAsync(){
            return await Task.Run (() => {
                using(var soapClient = new DBManagerService ()){
                    //TODO: Step 3 - map the response from
//                      //Query for the drug named "aspirin"
//                      var foundMatches = soapClient.getDrugs("aspirin");
//                         return MapSoapDtoToConceptProperty(foundMatches);
                }
            });
        }
```

12. Next, locate the comment: //TODO: 2 - Convert the SOAP DTOs into model objects. Uncomment the code below to map our DTO objects returned from the soap client to a model object.

```
//TODO: 2 - Convert the SOAP DTOs into model objects
private IEnumerable<Model.ConceptProperty> MapSoapDtoToConceptProperty(RxConce
```

```
ptGroup[] 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);
}
```

> **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.

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

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

14. Next, locate the comment: TODO: Step 3 - map the response from the web service. Uncomment the code below to map our DTO objects returned from the soap client to a model object

```
using(var soapClient = new DBManagerService ()){
    //TODO: Step 3 - map the response from the web service.
    //Query for the drug named "aspirin"
    var foundMatches = soapClient.getDrugs("aspirin");
    return MapSoapDtoToConceptProperty(foundMatches);
}
```

15. **Save** the project
16. **Build** the project and ensure that there are no errors
17. Open the **Activities\SoapActivity.cs** source file and locate the comment: // TODO: Step 4 - Create an activity to consume the data Uncomment the code below to associate a layout and an adapter to the activity

```
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;
}
```

18. Locate the comment: // Step 5 - 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.

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

    //Create a soap client
    var soapClient = new Client.SoapClient ();
    //Query
    var foundConceptProperties = await soapClient.GetDataAsync ();

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

19. Locate the comment: // Step 6 - Make the call to load the data.

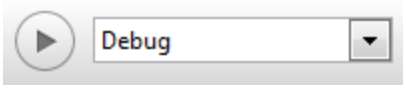
```
protected override void OnResume ()
{
    base.OnResume ();

    //TODO: Step 6 - 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.

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

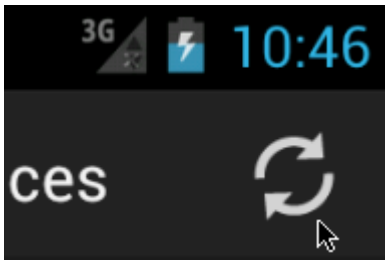
Testing the SOAP Web Services Client

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 SOAP - based web services. We, also, learned how to call those services in a way that keeps the user interface responsive.