Web Services

Xamarin Android Level 2, Chapter 5

Overview

This chapter introduces how to integrate web service technologies with a Xamarin.Android. It examines various service implementations, evaluates available tools and libraries to integrate them, and provides sample patterns for consuming service data. Finally, it provides a basic overview of creating a RESTful web service for consumption with a Xamarin mobile application.

In this article we examine how to utilize the Xamarin mobile framework including built-in and third-party APIs, as well as the associated tools to integrate various web service technologies, including:

- ➔ REST Services – REST (or RESTful) Services are an increasingly common paradigm for creating web services because of their simplicity and inherent platform agnostic approach. Many of the major service providers use REST, such as Twitter, Flickr, Facebook, etc. REST allows for stateless, cacheable, and simple to consume client-server architecture over HTTP. This chapter examines how to use RestShaper, a third-party option for consuming REST services.
- ➔ WCF Services – WCF is a Microsoft specific web service technology that aims at abstracting implementation from transport and data technology. Xamarin mobile applications support the consumption of WCF services that are exposed via the `BasicHttpBinding` protocol, by using the Microsoft Silverlight `SLSvcUtil.exe` utility for client proxy generation.

Additionally, this article gives a basic overview of creating REST services for applications from the server perspective using the Microsoft ASP.NET MVC framework.

REST Services

With Xamarin.Android consuming RESTful services is an architecturally simple and performance conscious option. This section examines several mechanisms to interact with REST services from client applications, and introduce the various tools and patterns used to parse the common message formats.

The simplicity of standards-based RESTful architecture has helped make it the primary option for mobile applications, since client interaction requires basic access to HTTP in lieu of a complete SOAP runtime. In this section we introduce the following tools to help us interact with HTTP using it's inherent request-response pattern:

RestSharp is a library that includes support for HTTP requests ranging from simple to complex. This includes support for file download, complex authentication (including OAuth), async workflow, message serialization, chunking, etc.

USING RESTSHARP

Another way to consume REST services is using the RestSharp library. RestSharp encapsulates http requests, including support for retrieving results either as raw

string content or as a deserialized C# object. For example, the following code makes a request to the same RXTerm service used earlier, and retrieves the results as a JSON formatted string.

```
var request = new RestRequest(string.Format("{0}/allinfo", rxcui));
request.RequestFormat = DataFormat.Json;
var response = Client.Execute(request);
if(string.IsNullOrWhiteSpace(response.Content) || response.StatusCode != System.Net.HttpStatusCode.OK) {
        return null;
}
rxTerm = DeserializeRxTerm(response.Content);
```

In the above code `DeserializeRxTerm` is some method that will take the raw JSON string from the `RestSharp.RestResponse.Content` property and convert it into a C# object. Deserializing data returned from web services is discussed later in this article.

CALLING SERVICES ASYNCHRONOUSLY

When a web service is called synchronously, it blocks the main user interface thread, preventing the user from interacting the application. In order to alleviate this, use async operations to initiate the request on a background thread and callback to the main thread to display the data when the results are returned.

The following code from the sample application that accompanies this article illustrates how to call a service asynchronously:

TODO : Need an Android specific sample here

Options for Consuming RESTful Data

After returning the response from an HTTP request, we may be interested in consuming the data in a structured format. This can be accomplished by deserializing the response into defined model objects. Of course, one of the benefits associated with using the Xamarin mobile frameworks is potential re-use of ubiquitous model objects employed throughout the overall application architecture. In this section we will look at the available mechanisms to consume RESTful responses in Plain-Old-XML (POX) and JSON and map these responses to objects:

- ➔ System.Xml / System.Json – included in the BCL are the standard classes provided in the System.Xml and System.Json namespaces included in the Silverlight profile. “XLinq” or “Linq-to-XML” is also available in the System.Xml.Linq namespace.

USING SYSTEM.XML.LINQ

Xamarin has support for using LINQ, including Linq to XML within mobile applications. The following example shows how to parse XML and populate a C# object inline:

```
var doc = XDocument.Parse(xml);
var result = doc.Root.Descendants("rxtermsProperties")
.Select(x=> new RxTerm()
        {
```

```
            BrandName = x.Element("brandName").Value,
            DisplayName = x.Element("displayName").Value,
            Synonym = x.Element("synonym").Value,
            FullName = x.Element("fullName").Value,
            FullGenericName = x.Element("fullGenericName").Value,
            //bind more here...
            RxCUI = x.Element("rxcui").Value,
    });
```

USING SYSTEM.JSON

Xamarin mobile products also ship with support for JSON out of the box. By using a JsonObject, results can be retrieved as shown below:

```
var term = new RxTerm();

var obj = JsonObject.Parse(json);
var properties = obj["rxtermsProperties"];
term.BrandName = properties["brandName"];
term.DisplayName = properties["displayName"];
term.Synonym = properties["synonym"];
term.FullName = properties["fullName"];
term.FullGenericName = properties["fullGenericName"];
term.Strength = properties["strength"];
```

This is the familiar key-value coding style. However, it's important to be aware that the `System.Json` tools load the entirety of the data into memory. Additionally, Linq can also be used with JSON as well.

Summary

This document covered various options for consuming web services seamlessly with Xamarin.Android. It also evaluated several tools and patterns for consuming SOAP, WCF, and REST services. Finally, it examined a basic sample application designed to help get started with creating web services.