CF Walkthrough

This walkthrough covers how a mobile application built with Xamarin can consume a WCF web service using the BasicHttpBinding.

PDF for Offline Use:
Download PDF

Sample Code:
HelloWorld.zip

Related Articles:
Windows Communication Foundation on MSDN
How to: Access a Service from Silverlight
Using SLsvcUtil.exe to Access a Service

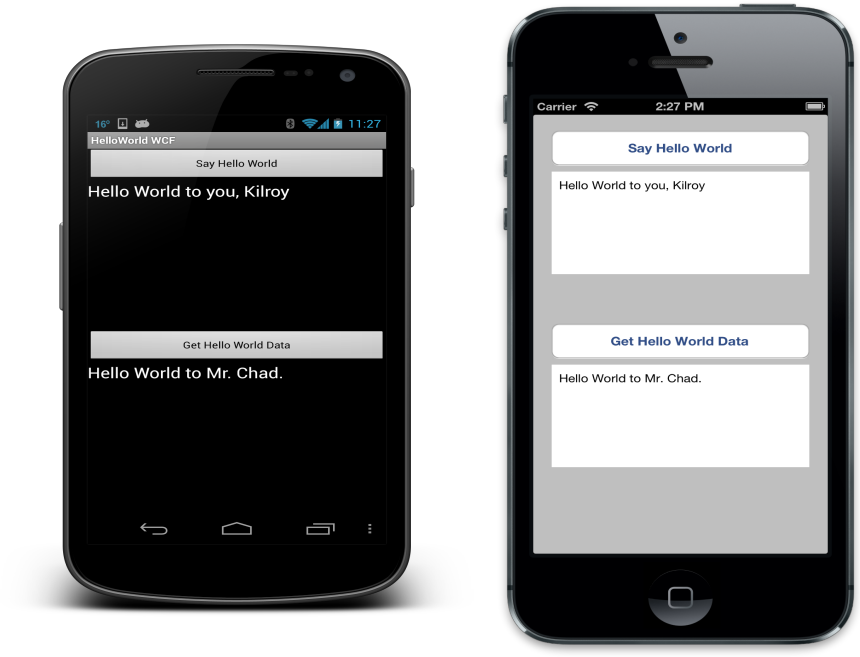
Related SDKs:
Microsoft Silverlight 5 SDK

Overview

It is a common requirement for mobile applications to be able to communicate with backend systems. There are many choices and options for backend frameworks, one of which is Windows Communication Foundation (WCF). This walkthrough will provide an example of how a Xamarin mobile application can consume a WCF service using the BasicHttpBinding, as outlined below:

1. **Create a WCF Service** - In this section we will create a very basic WCF service having two methods. The first method will take a string parameter, while the other method will take a C# object. This section will also discuss how to configure a developer's workstation to allow remote access to the WCF service.
2. **Create a Xamarin.Android Application** - Once the WCF service has been created, we will create a simple Xamarin.Android application that will use the WCF service. This section will cover how to create a WCF service proxy class to facilitate communication with the WCF service.
3. **Create a Xamarin.iOS Application** - The final part of this tutorial involves creating a simple Xamarin.iOS application that will use the WCF service.

The following screen shots shows the two applications running:



Requirements

This walkthrough assumes that you have some familiarity with creating and using WCF services.

In order to create the WCF service proxies, you will need the Silverlight 5 SDK installed. Download and run the

installer from Microsoft before proceeding with this walkthrough.

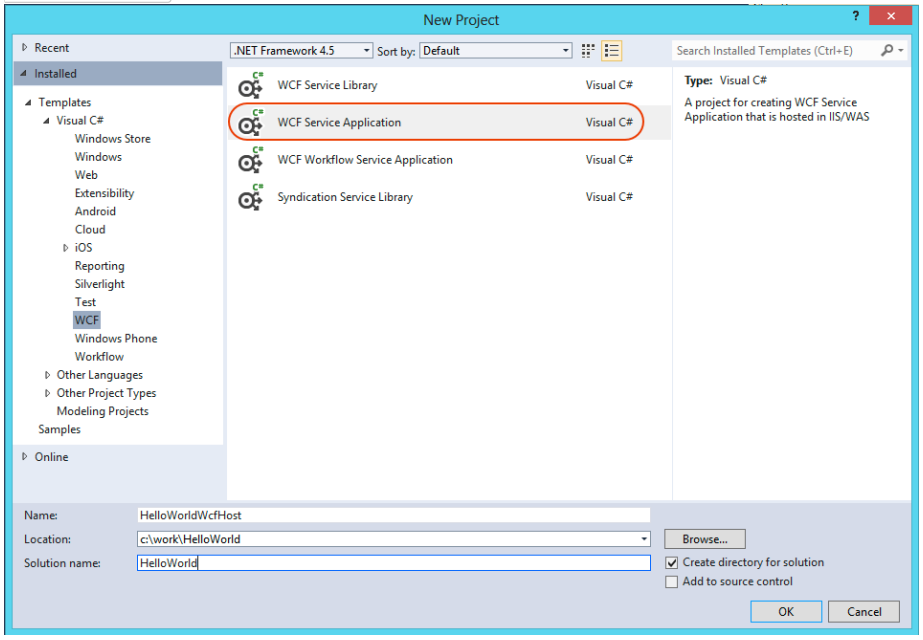
IIS Express will be used to host the WCF Service used in this walkthrough. This is the default web server for Visual Studio 2012. Installing IIS Express in older versions of Visual Studio is beyond the scope of this walkthrough.

> **Note:** This walkthrough was created on Windows 8 using Visual Studio 2012. If you are using on an older version of Windows or Visual Studio, be aware that some parts of this walkthrough may not be applicable to your development environment.

Creating a WCF Service

The first task before us is to create a WCF service for our mobile applications to communicate with.

1. Start up Visual Studio 2012, and press `Alt-F-N-P` to open the New Project dialog. Select the WCF Service Application template from the new project dialog as shown in the following screenshot:



   Name the project `HelloWorldWcfHost`, and name the solution `HelloWorld`. Click the OK button.

2. Next we need to create the service contract for the web service. Add an interface named `IHelloWorldService` and paste in the following code:

```
1 [ServiceContract]
2 public interface IHelloWorldService
3 {
4     [OperationContract]
5     string SayHelloTo(string name);
6
7     [OperationContract]
8     HelloWorldData GetHelloData(HelloWorldData helloWorldData);
9 }
```

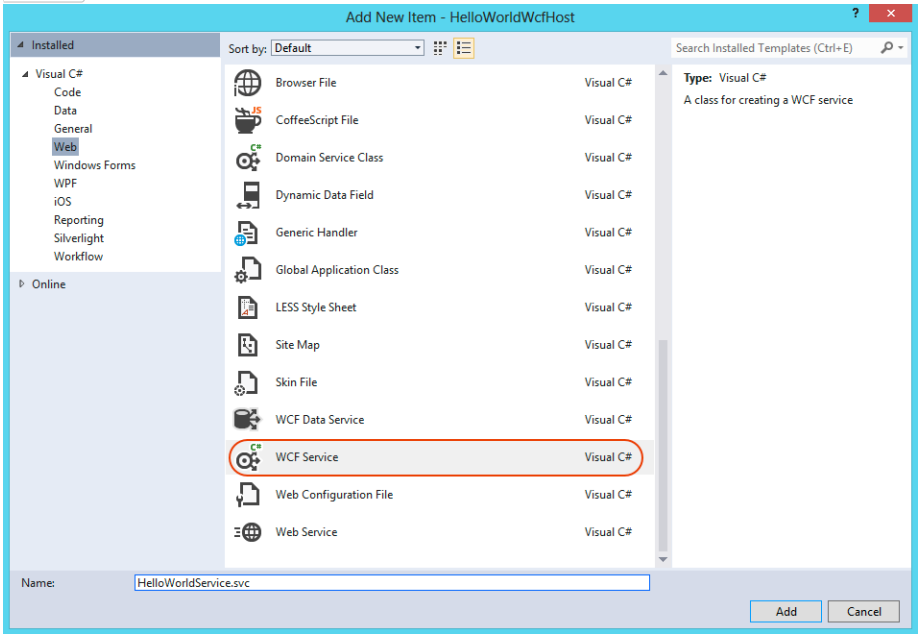
   This service provides two methods - one that takes a string for a parameter and another that takes a .NET object.

3. Our WCF service requires the class `HelloWorldData` as a parameter, so let's create this type next. Add a new class to the project named `HelloWorldData` with the following implementation:

```
1  [DataContract]
2  public class HelloWorldData
3  {
4      public HelloWorldData()
5      {
6          Name = "Hello ";
7          SayHello = false;
8      }
9
10     [DataMember]
11     public bool SayHello { get; set; }
```

```
12
13      [DataMember]
14      public string Name { get; set; }
15  }
```

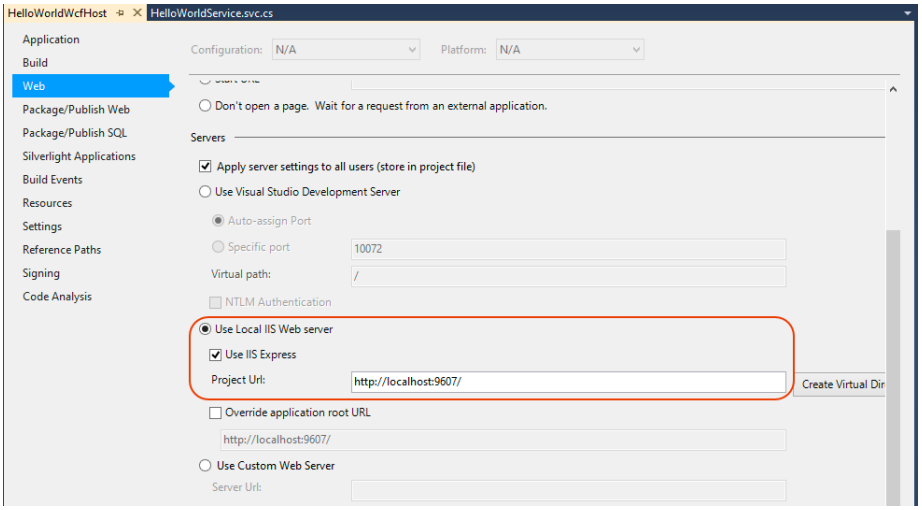
4. The final thing we need to do is to create the WCF Service class . Press `Ctrl-Shift-A` to bring up the Add New Item dialog:



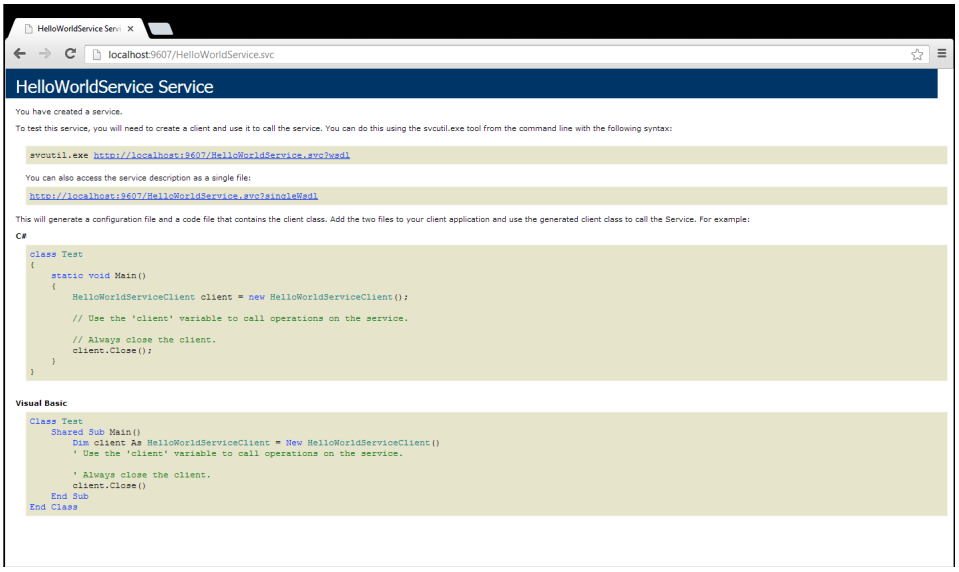
Add a new WCF Service class to the project named `HelloWorldService`. Edit the class so that it implements `IHelloWorldService` and contains the code from the following snippet:

```
1  public class HelloWorldService : IHelloWorldService
2  {
3      public HelloWorldData GetHelloData(HelloWorldData helloWorldData)
4      {
5          if (helloWorldData == null)
6          {
7              throw new ArgumentNullException("helloWorldData");
8          }
9
10         if (helloWorldData.SayHello)
11         {
12             helloWorldData.Name = String.Format("Hello World to {0}.", helloWorldData.Nam
13         }
14         return helloWorldData;
15     }
16
17     public string SayHelloTo(string name)
18     {
19         return string.Format("Hello World to you, {0}", name);
20     }
21 }
```

5. This final step is optional. In it we change the default port for the WCF service to use 9607 for connections being made from `localhost`. Press `Alt-P-P` to bring up the Project Properties dialog for the HelloWorldWcfHost project. Select the Web tab, and set the Project Url to `http://localhost:9607/`, as shown in the following screenshot:



At this point we should have a working WCF service. If we run the project, and point our browser to http://localhost:9067/HelloWorldService.svc, we should see the following page:



This current setup is sufficient if we only have to connect with the WCF service from our local workstation. However, remote devices (such as an Android device or an iPhone) do not have any access to the WCF service. The next section will cover how to configure Windows 8 and IIS Express to accept remote connections.

> **Note:** The following section is only necessary if you need to accept remote connections on a Windows 8 workstation. If you have an alternate platform on which to deploy this Web Service you can ignore following section.

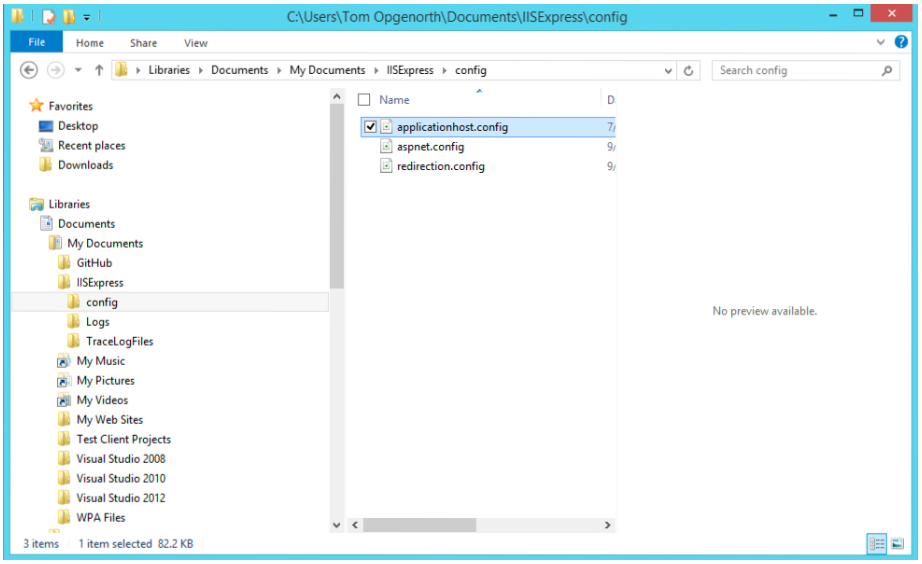
Configuring Remote Access to IIS Express

By default, Windows 8 and IIS Express will not accept remote connections. Before any remote devices, such as an Android device or an iPhone can communicate with our WCF service we must make the following changes:

1. **Configure IIS Express to Accept Remote connections** - This step involves editing the config file for IIS Express to accept remote connections on a specific port and then setting up a rule for IIS Express to accept the incoming traffic.
2. **Add an Exception to Windows Firewall** - We must open up a port through Windows Firewall that remote applications can use to communicate with the WCF service.

You will need to know the IP address of your workstation. For the purposes of this example we'll assume that our workstation has the IP address 192.168.1.143.

1. Let's begin by configuring IIS Express to listen for external requests. We do this by editing the configuration file for IIS Express at `%userprofile%\documents\iisexpress\config\applicationhost.config`, as shown in the following screenshot:



Locate the `site` element with the name `HelloWorldWcfHost`. It should look something like the following XML snippet:

```
<site name="HelloWorldWcfHost" id="2">
    <application path="/" applicationPool="Clr4IntegratedAppPool">
        <virtualDirectory path="/" physicalPath="\\vmware-host\Shared Folders\tom\work\xam
    </application>
    <bindings>
        <binding protocol="http" bindingInformation="*:9607:localhost" />
    </bindings>
</site>
```

We will need to add another `binding` to open up port 9608 to outside traffic. Add the following XML to the `bindings` element:

```
<binding protocol="http" bindingInformation="*:9608:192.168.1.143" />
```

This will configure IIS Express to accept HTTP traffic from any remote IP address on port 9608 on the external IP address of the computer. This above snippet assumes the IP address of the computer running IIS Express is 192.168.1.143. After the changes, the `bindings` element should look like the following:

```
<site name="HelloWorldWcfHost" id="2">
    <application path="/" applicationPool="Clr4IntegratedAppPool">
        <virtualDirectory path="/" physicalPath="\\vmware-host\Shared Folders\tom\work\xam
    </application>
    <bindings>
        <binding protocol="http" bindingInformation="*:9607:localhost" />
        <binding protocol="http" bindingInformation="*:9608:192.168.1.143" />
    </bindings>
</site>
```

2. Next, we need to configure IIS Express accept incoming connections on port 9608. Startup up an administrative command prompt, and run this command:

```
> netsh http add urlacl url=http://192.168.1.143:9608/ user=everyone
```

3. The final step is to configure Windows Firewall to permit external traffic on port 9608. From an administrative command prompt, run the following command:

```
> netsh advfirewall firewall add rule name="IISExpressXamarin" dir=in protocol=tcp localpo
```

This command will allow incoming traffic on port 9608 from all devices on the same subnet as the Windows 8 workstation.

At this point we have created a very basic WCF service hosted in IIS Express that will accept incoming connections from other devices or computers on our subnet. You can test this out by visiting http://localhost:9607/HelloWorldService.svc on your workstation and http://192.168.1.143:9608/HelloWorldService.svc from another computer on your subnet.

Creating a Xamarin.Android Application

Now that we have the WCF service working, let's move on to creating a Xamarin.Android application.

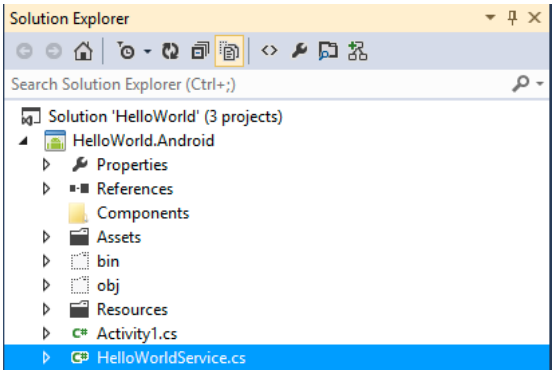
1. Add a new Android project to the solution and name it `HelloWorld.Android`.
2. Change the default namespace of the application from `HelloWorld.Android` to `HelloWorld.Droid`.
3. Next we need to create a proxy for the web service. To create the proxy we'll use the *Silverlight Service Model Proxy Generation Tool* (`SLsvcUtil.exe`). You can find this command line utility at the following location:

```
C:\Program Files (x86)\Microsoft SDKs\Silverlight\v5.0\Tools\SLsvcUtil.exe
```

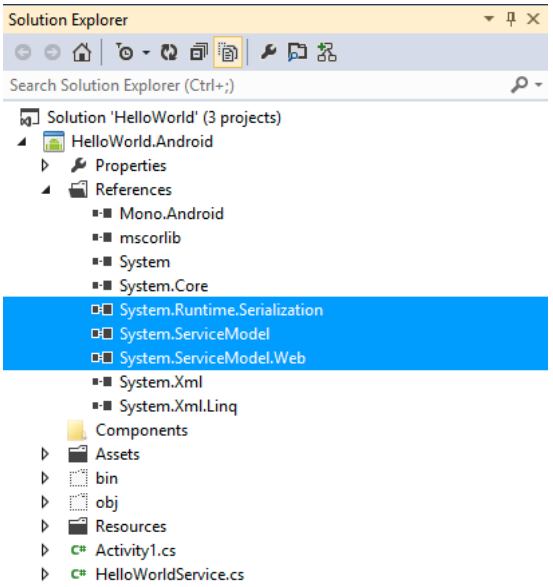
   Ensure that the WCF service we created in the previous section is running, and then run `SLsvcUtil.exe` with the following command line:

```
SLsvcUtil.exe /noConfig http://localhost:9607/HelloWorldService.svc
```

   This will create a service proxy called `HelloWorldService` in the file `HelloWorldService.cs`. Add this file to your Xamarin.Android project as shown in the following screenshot:



4. Before this generated proxy class will compile, we need to add to some references to our Xamarin.Android project, as shown in the follwoing screenshot:



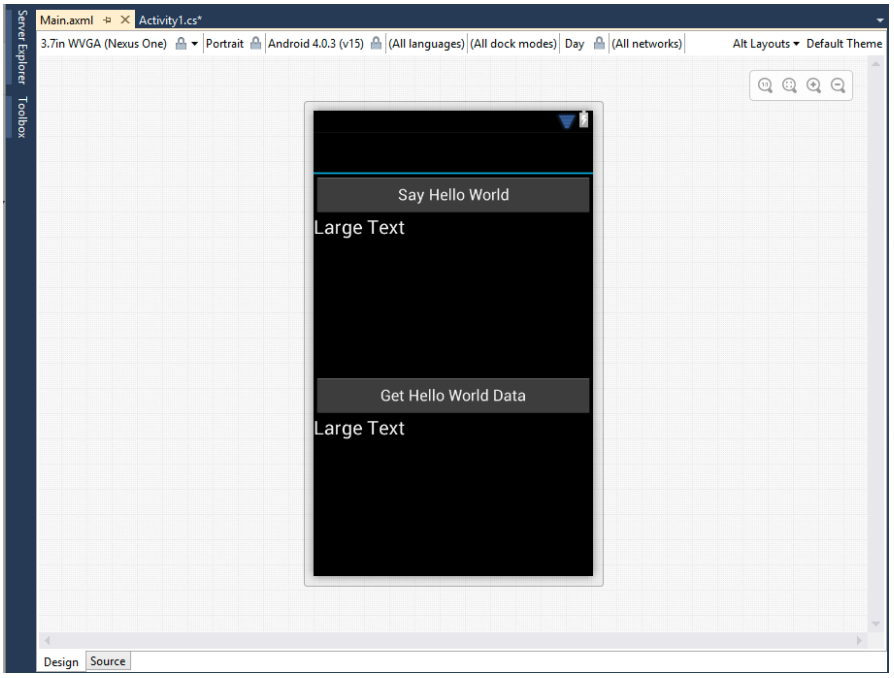
5. With the above infrastructure in place, we can finish up the Android application. Edit the file `Activity1.cs` and add the following instance variables:

```
[Activity(Label = "@string/app_name", MainLauncher = true, Icon = "@drawable/icon")]
public class Activity1 : Activity
{
    public static readonly EndpointAddress EndPoint = new EndpointAddress("http://192.168

    private HelloWorldServiceClient _client;
    private Button _getHelloWorldDataButton;
    private TextView _getHelloWorldDataTextView;
    private Button _sayHelloWorldButton;
    private TextView _sayHelloWorldTextView;

```

6. Next, update `Main.axml` with the following XML:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
              android:orientation="vertical"
              android:layout_width="fill_parent"
              android:layout_height="fill_parent">
    <LinearLayout
            android:orientation="vertical"
            android:layout_width="fill_parent"
            android:layout_height="0px"
            android:layout_weight="1">
        <Button
                android:id="@+id/sayHelloWorldButton"
                android:layout_width="fill_parent"
                android:layout_height="wrap_content"
                android:text="@string/say_hello_world" />
        <TextView
                android:text="Large Text"
                android:textAppearance="?android:attr/textAppearanceLarge"
                android:layout_width="fill_parent"
                android:layout_height="wrap_content"
                android:id="@+id/sayHelloWorldTextView" />
    </LinearLayout>
    <LinearLayout
            android:orientation="vertical"
            android:layout_width="fill_parent"
            android:layout_height="0px"
            android:layout_weight="1">
        <Button
                android:id="@+id/getHelloWorldDataButton"
                android:layout_width="fill_parent"
                android:layout_height="wrap_content"
                android:text="@string/get_hello_world_data" />
        <TextView
                android:text="Large Text"
                android:textAppearance="?android:attr/textAppearanceLarge"
                android:layout_width="fill_parent"
                android:layout_height="wrap_content"
                android:id="@+id/getHelloWorldDataTextView" />
    </LinearLayout>
</LinearLayout>

```

The following is a screenshot of what this UI looks like in the designer:



7. Now with the UI and instance variables in place, modify `OnCreate` to contain the following code:

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

    SetContentView(Resource.Layout.Main);

    InitializeHelloWorldServiceClient();

```

```
    // This button will invoke the GetHelloWorldData - the method that takes a C# object
    _getHelloWorldDataButton = FindViewById<Button>(Resource.Id.getHelloWorldDataButton);
    _getHelloWorldDataButton.Click += GetHelloWorldDataButtonOnClick;
    _getHelloWorldDataTextView = FindViewById<TextView>(Resource.Id.getHelloWorldDataText
    // This button will invoke SayHelloWorld - this method takes a simple string as a par
    _sayHelloWorldButton = FindViewById<Button>(Resource.Id.sayHelloWorldButton);
    _sayHelloWorldButton.Click += SayHelloWorldButtonOnClick;
    _sayHelloWorldTextView = FindViewById<TextView>(Resource.Id.sayHelloWorldTextView);
}
```

The code above initializes the instance variables for our class and wires up some event handlers.

8. Next we need to instantiate the client proxy class in our Activity. Edit `Activity1.cs` and add the following two methods to the class:

```
private void InitializeHelloWorldServiceClient()
{
    BasicHttpBinding binding = CreateBasicHttp();

    _client = new HelloWorldServiceClient(binding, EndPoint);
    _client.SayHelloToCompleted += ClientOnSayHelloToCompleted;
    _client.GetHelloDataCompleted += ClientOnGetHelloDataCompleted;
}

private static BasicHttpBinding CreateBasicHttp()
{
    BasicHttpBinding binding = new BasicHttpBinding
    {
        Name = "basicHttpBinding",
        MaxBufferSize = 2147483647,
        MaxReceivedMessageSize = 2147483647
    };
    TimeSpan timeout = new TimeSpan(0, 0, 30);
    binding.SendTimeout = timeout;
    binding.OpenTimeout = timeout;
    binding.ReceiveTimeout = timeout;
    return binding;
}
```

The code above instantiates and initializes a `HelloWorldService` object. The WCF proxy class can only call the WCF service asynchronously. Responses from the WCF service will handled by the `xxxCompleted` events that were generated by `SLsvcUtil.exe`.

9. Now we need to add the event handlers for the two buttons in our activity. Edit `Activity1.cs` and add the following two methods:

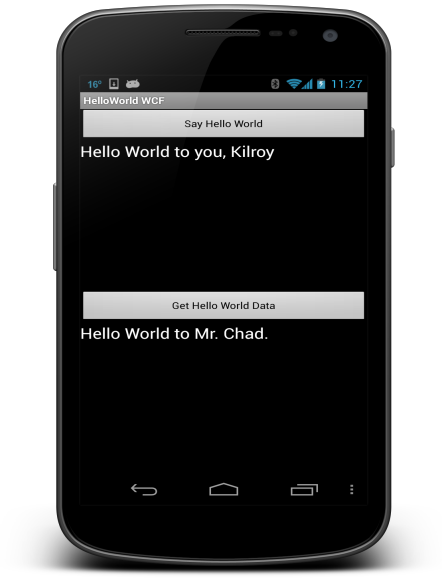
```
private void GetHelloWorldDataButtonOnClick(object sender, EventArgs eventArgs)
{
    HelloWorldData data = new HelloWorldData { Name = "Mr. Chad", SayHello = true };
    _getHelloWorldDataTextView.Text = "Waiting for WCF...";
    _client.GetHelloDataAsync(data);
}

private void SayHelloWorldButtonOnClick(object sender, EventArgs eventArgs)
{
    _sayHelloWorldTextView.Text = "Waiting for WCF...";
    _client.SayHelloToAsync("Kilroy");
}

```

10. Finally, we need to add event handlers for the `xxxCompleted` events of the `HelloWorldService` proxy client. Once again edit `Activity1.cs` and add the following methods:

```
private void ClientOnGetHelloDataCompleted(object sender, GetHelloDataCompletedEventArgs
{
    string msg = null;

    if (getHelloDataCompletedEventArgs.Error != null)
    {
        msg = getHelloDataCompletedEventArgs.Error.Message;
    }
    else if (getHelloDataCompletedEventArgs.Cancelled)
    {
        msg = "Request was cancelled.";
    }
    else
    {
        msg = getHelloDataCompletedEventArgs.Result.Name;
    }
    RunOnUiThread(() => _getHelloWorldDataTextView.Text = msg);
```

```
}

private void ClientOnSayHelloToCompleted(object sender, SayHelloToCompletedEventArgs sayH
{
    string msg = null;

    if (sayHelloToCompletedEventArgs.Error != null)
    {
        msg = sayHelloToCompletedEventArgs.Error.Message;
    }
    else if (sayHelloToCompletedEventArgs.Cancelled)
    {
        msg = "Request was cancelled.";
    }
    else
    {
        msg = sayHelloToCompletedEventArgs.Result;
    }
    RunOnUiThread(() =>_sayHelloWorldTextView.Text = msg);
}
```

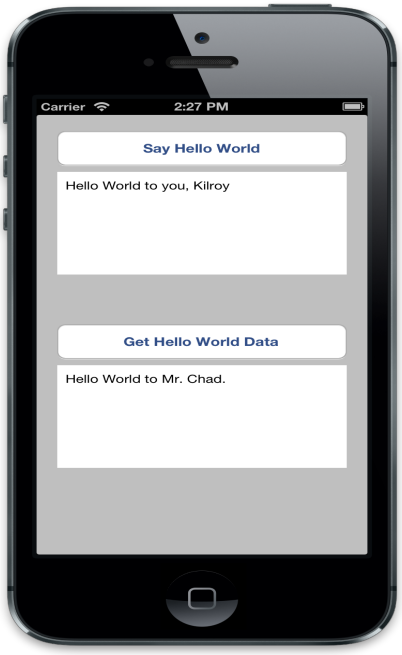
11. Run the application, and click on the two buttons. Our application will call the WCF asynchronously. Within 30 seconds a response should be received from each WCF method, and our application should look something like the following screenshot:



Now that we have a working Xamarin.Android application, let's create a Xamarin.iOS client.

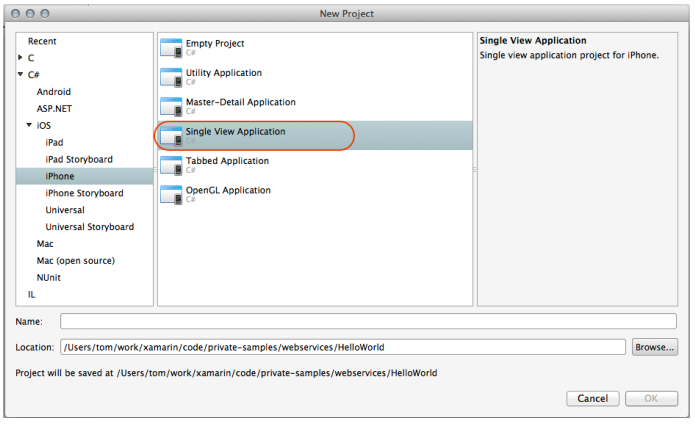
Creating a Xamarin.iOS Application

The Xamarin.iOS version is very similar to the Xamarin.Android application and will use the exact same WCF proxy client code that was generated by `SLsvcUtil.exe`. The following screenshot shows our application after it has successfully interacted with the WCF service that we created earlier on in this walkthrough:



Let's get started with the Xamarin.iOS application.

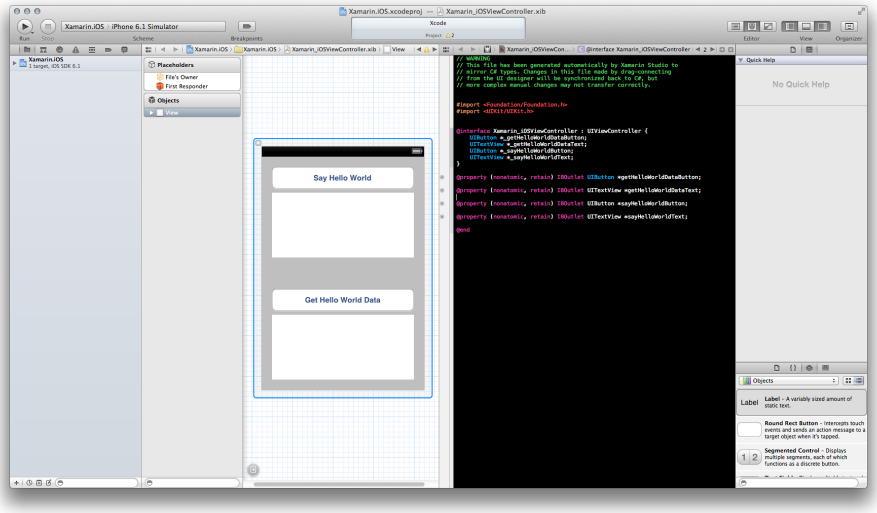
1. Start up Xamarin Studio, and open the HelloWorld solution. Add a new iPhone `Single View Application` to the solution, as shown in the following screenshot:



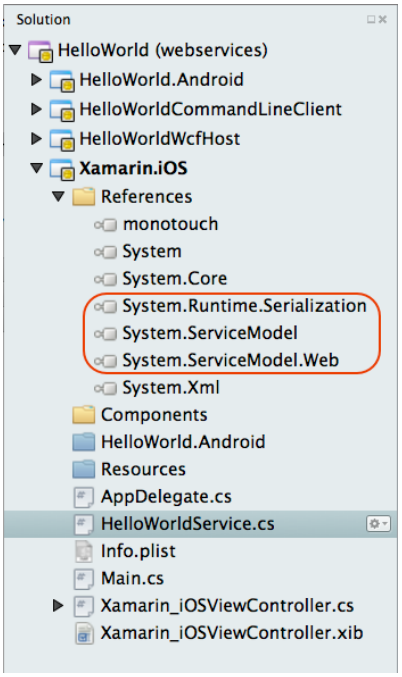
2. To create the user interface for our application, edit the `.XIB` in Interface Builder and add two `UIButton`s and two `UITextView`s. Add the outlets for each control according to the following table:

	Name	Text
`UIButton`	`sayHelloWorldButton`	Say Hello World
`UITextView`	`sayHelloWorldText`	
`UIButton`	`getHelloWorldDataButton`	Get Hello World Data
`UITextView`	`sayHelloWorldText`	

After creating the outlets, the UI should resemble the following screenshot:



3. Next, add the WCF proxy class `HelloWorldService.cs` to our project by pressing `Option-Cmd-A`. This will bring up the `Add files` dialog which you can use to locate and add the file to the project. Once that is done add references to `System.Runtime.Serialization`, `System.ServiceModel`, and `System.ServiceModel.Web` just as we did in the Android application. The following screenshot shows Solution Explorer after adding these references:



4. We need to add a couple variables to our view controller that the WCF client code will use. Edit the class `Xamarin_iOSViewController` and add two instance variables as shown in the following code snippet:

```
1 public partial class Xamarin_iOSViewController : UIViewController
2 {
3     public static readonly EndpointAddress EndPoint = new EndpointAddress("http://192.168.
4
5     private HelloWorldServiceClient _client;
6
```

You will probably have to change the IP address to that of the computer that is hosting your WCF service.

5. After adding the variable above, update the `ViewDidLoad` method to include the following code:

```
1 public override void ViewDidLoad()
2 {
3     base.ViewDidLoad();
4     InitializeHelloWorldServiceClient();
5 }
```

6. Next, add some helper methods to instantiate a `HelloWorkdService` client proxy instance and hook up a couple event handlers.. Add the following two methods to `Xamarin_iOSViewController`:

```
1 private void InitializeHelloWorldServiceClient()
2 {
```

```
    BasicHttpBinding binding = CreateBasicHttp();

    _client = new HelloWorldServiceClient(binding, EndPoint);
    _client.SayHelloToCompleted += ClientOnSayHelloToCompleted;
    _client.GetHelloDataCompleted += ClientOnGetHelloDataCompleted;

    getHelloWorldDataButton.TouchUpInside += GetHelloWorldDataButtonTouchUpInside;
    sayHelloWorldButton.TouchUpInside += SayHelloWorldDataButtonTouchUpInside;
}

private static BasicHttpBinding CreateBasicHttp()
{
    BasicHttpBinding binding = new BasicHttpBinding
        {
            Name = "basicHttpBinding",
            MaxBufferSize = 2147483647,
            MaxReceivedMessageSize = 2147483647
        };
    TimeSpan timeout = new TimeSpan(0, 0, 30);
    binding.SendTimeout = timeout;
    binding.OpenTimeout = timeout;
    binding.ReceiveTimeout = timeout;
    return binding;
}
```

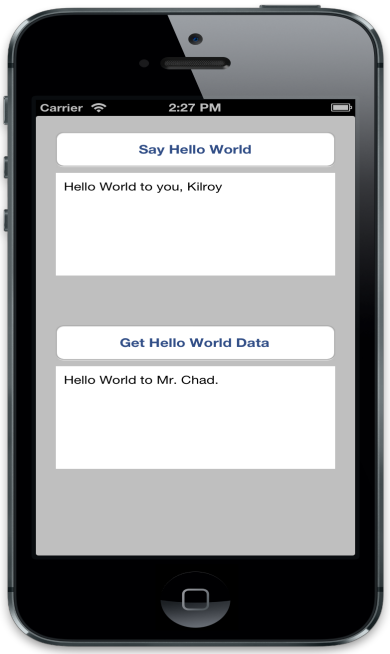
7. Next, create event handlers for the two `UIButton`s in the XIB file as shown below:

```
private void SayHelloWorldDataButtonTouchUpInside(object sender, EventArgs e)
{
    sayHelloWorldText.Text = "Waiting for WCF...";
    _client.SayHelloToAsync("Kilroy");
}

private void GetHelloWorldDataButtonTouchUpInside(object sender, EventArgs e)
{
    getHelloWorldDataText.Text = "Waiting WCF...";
    HelloWorldData data = new HelloWorldData { Name = "Mr. Chad", SayHello = true };
    _client.GetHelloDataAsync(data);
}
```

8. Finally, add the event handlers for the `xxxOnCompleted` events that the `HelloWorldService` proxy client will raise when the WCF service sends a reply back to our application. Add the following two methods to `Xamarin_iOSViewController`:

```
private void ClientOnGetHelloDataCompleted(object sender, GetHelloDataCompletedEventArgs
{
    string msg = null;

    if (e.Error != null)
    {
        msg = e.Error.Message;
    }
    else if (e.Cancelled)
    {
        msg = "Request was cancelled.";
    }
    else
    {
        msg = e.Result.Name;
    }

    InvokeOnMainThread(() => getHelloWorldDataText.Text = msg);
}

private void ClientOnSayHelloToCompleted(object sender, SayHelloToCompletedEventArgs e)
{
    string msg = null;

    if (e.Error != null)
    {
        msg = e.Error.Message;
    }
    else if (e.Cancelled)
    {
        msg = "Request was cancelled.";
    }
    else
    {
        msg = e.Result;
    }
    InvokeOnMainThread(() => sayHelloWorldText.Text = msg);
}

```

9. Now run the application, and click on each of the buttons in the UI. The WCF service will be called asynchronously. Within 30 seconds a response should be received from each WCF method, and our application should look like the following screenshot:



Summary

This tutorial covered how to work with a WCF service in a mobile application using Xamarin.Android and Xamarin.iOS. It showed how to create a WCF service and explained how to configure Windows 8 and IIS Express to accept connections from remote devices. It then discussed how to generate a WCF proxy client using the Silverlight Service Model Proxy Generation Tool (`SLsvcUtil.exe`) and demonstrated how to use the proxy client in both Xamarin.Android and Xamarin.iOS applications.