Content

1. [WCF - Introduction](https://www.codeproject.com/Articles/884992/Simple-Demo-WCF-Web-Services#WCFIntro)
2. [What is a Service Contract, Operation Contract and a Data Contract](https://www.codeproject.com/Articles/884992/Simple-Demo-WCF-Web-Services#ServiceOperationDataContracts)
3. [A word about WCF configuration](https://www.codeproject.com/Articles/884992/Simple-Demo-WCF-Web-Services#WCFConfiguration)
4. [Look at our project defining simple Business Functionality](https://www.codeproject.com/Articles/884992/Simple-Demo-WCF-Web-Services#BusinessFunctionalityExplanation)
5. [Expose the above Business Functionality via a WCF Web Service](https://www.codeproject.com/Articles/884992/Simple-Demo-WCF-Web-Services#CreatingWCFWebService)
6. [Employ WCF Test Client to test out our newly created Web Service](https://www.codeproject.com/Articles/884992/Simple-Demo-WCF-Web-Services#TestingWCFWebService)
7. [Create a Proxy Client for our new Web Service](https://www.codeproject.com/Articles/884992/Simple-Demo-WCF-Web-Services#CreatingProxyClient)
8. [Test our above created proxy client](https://www.codeproject.com/Articles/884992/Simple-Demo-WCF-Web-Services#TestingProxyClient)
9. [Final Thoughts](https://www.codeproject.com/Articles/884992/Simple-Demo-WCF-Web-Services#FinalThoughts)

Introduction



In this article, we will have a look at a simple WCF Web Service.

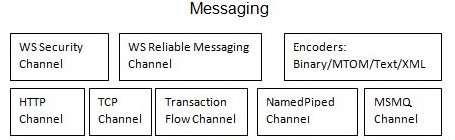
It is far easier to understand concepts through examples.

In this article, we will be using a total of 3 projects.  
Class Library Project TweetBL will represent a project that has some business functionality that needs to be service enabled. Then, we will look at a WCF Service Project which will expose our project TweetBL's business functionality as a Web Service. In our third and final project, we will create a proxy client which will consume our WCF Web Service.

Finally, it is worth adding that we will be testing each and every bit as we go. Stick around and don't panic I have tried to use simple code and just focus on WCF here!  
So let's go...

WCF - Introduction

WCF stands for Windows Communication Foundation.  
WCF is a framework designed for supporting message-based communication on both client and server.  
WCF supports multiple transports like SOAP over HTTP (our example Web Service is based on this), TCP, named pipes, MSMQ and some others.  
WCF supports multiple encodings like XML, text, binary and a few other options.  
Currently, to name some technologies that employ WCF are Windows Workflow, Silverlight, etc.



What is a Service Contract, Operation Contract and a Data Contract

In WCF, contracts are used to define functionality.  
Contracts basically are formal agreements between a client and a service to define a platform-neutral  
and standard way of describing what a service will and can do.

WCF defines 4 types of contracts:

1. Service Contracts
2. Operation Contracts
3. Data Contracts
4. Fault Contracts

We will be looking at the first three contracts in this article.



**Service Contracts** define the capabilities of a web service.  
It is essentially a Interface Contract that binds the application (business functionality - business operations  
being exposed) to provide a fixed set of operations (as stated in the interface contract) to the sources  
consuming the web service.  
It simply describes the client-callable operations, or methods, that are available on the service endpoint,  
and are exposed to the outside world.

In terms of code:  
All we need to do is to code and decorate a interface with ServiceContract attribute.  
This interface will have members (methods) that will be made available as part of the Tweet Service to the outside world.

Hide   Copy Code

*//*

*// Service Contract*

*// Wee need to decorate the Interface with the ServiceContract Attribute*

*//*

[ServiceContract]

interface ITweetService

{

*// Interface members*

*// Tweet Service Methods that need to be exposed to the outside world!*

}

**Operation Contracts** on the other hand defines the methods of the service that are accessible to the external systems.  
It is integral to the above created interface for Service Contract.  
Members marked with Operation Contract attribute in the Service Contract Interface are the ones which are  
exposed to external systems.  
Members not decorated with Operation Contract attribute in Service Contract Interface are **not** exposed  
as part of the web service.

In terms of code:  
It can be applied only on methods.  
It is used to decorate methods which belongs to **Service Contract** interfaces.

Hide   Copy Code

*//*

*// Service Contract - Operation Contact example*

*//*

[ServiceContract]

interface ITweetService

{

[OperationContract]

int AddNewTweet(string tweetBy, string tweet);

*// Other members...*

}

**Data Contracts** defines the type and the format of data that will be sent out and received by the web service operations.  
It is a formal agreement between the service and a client that abstractly describes the data to be exchanged.  
Data contract can be explicit or implicit. Simple type such as int, string, etc. has an implicit data contract.  
User defined object are explicit or Complex type, for which you have to define a Data contract using [DataContract] and [DataMember] attribute.

Some important factors regarding Data Contracts are mentioned below:

1. Describes the format of data, structure of data and types of data that is passed to and from the web service operations
2. Does the mapping between CLR type and XML Schema
3. Defines how the associated data is serialized and deserialized  
   Serialization = Converting data (object) into a sequence of bytes that can be transmitted over the Network  
   De-serialization = reassembling a sequence of bytes to it its original form -> data (object)
4. Is a loosely-coupled model that is defined outside the implementation of the service and is accessible by services in other platforms

In terms of code:  
To define a data contract, apply the DataContract attribute to the class to serialize the class by a serializer, and apply the DataMember attribute to the fields in the class that must be serialized.  
We need to include System.Runtime.Serialization reference to the project.  
This assembly holds the DataContract and DataMember attribute.

Hide   Copy Code

*//*

*// Data Contract - Data Member example*

*//*

*// Creating a user defined data type called SampleData. This data type should be identified*

*// for serialization and de-serialization by mentioning with [DataContract] and [DataMember] attribute.*

*//*

*// There should be: using System.Runtime.Serialization; reference in the using section.*

*//*

[DataContract]

public class SampleData

{

private string \_somePropertyPrivateVariable;

[DataMember]

public string SomeProperty

{

get { return \_somePropertyPrivateVariable;}

set { \_somePropertyPrivateVariable = value;}

}

*// Other properties...*

}

A Word about WCF Configuration

Since WCF supports so many technologies, transport protocols and encodings, it has a heavy reliance on configuration.

It all boils down to the below:

* **Endpoint**: This is the address where the server can send and receive data.
* **Binding**: This is what defines the transport and encoding used by a Endpoint.
* **Behaviour**: This is the final bit and is responsible for the runtime behaviour of a web service, endpoint, operation and/or client.

Behaviours are important as they influence error handling, credentials and many other important features.

We will be using default WCF Configuration in this article. The default WCf Configuration provided for every project is explained [here](https://www.codeproject.com/Articles/884992/Simple-Demo-WCF-Web-Services#DefaultWCFConfiguration).

Look At Our Project Defining Simple Business Functionality

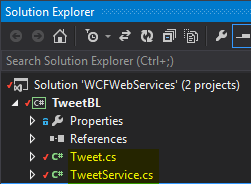
**We will expose this Business Functionality as a Web Service!**

The project that we will be exposing as a Web Service is a simple class library project called TweetBL (BL stands for Business Layer).  
Project TweetBL represents the business code that has been identified to be made available over the web via a web service.  
TweetBL project has the following business functionality (methods) that will be exposed as a service.

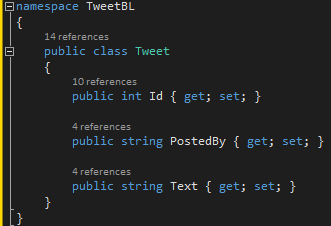
Business functionality methods are:

1. update tweet
2. save/insert a tweet
3. delete a tweet
4. retrieve tweet by id and
5. retrieve all tweets

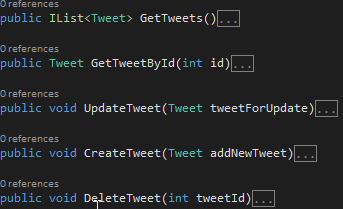
TweetBL Project:



***Tweet.cs***is simply the data class identifying all data members that we have to fetch/update/insert and delete to manage tweets. Think of Model (Data Model).

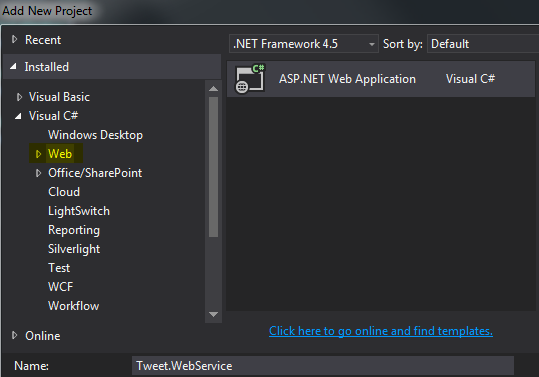


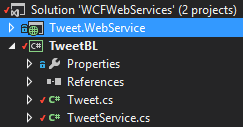
***TweetService.cs***is the class that houses the business methods that do all the necessary processing on the Tweet Data to manage and maintain it.  
**Note**: There is a bit of housekeeping code as well (unrelated to business functionality being exposed) because we are not using an actual database to keep things fairly simple.



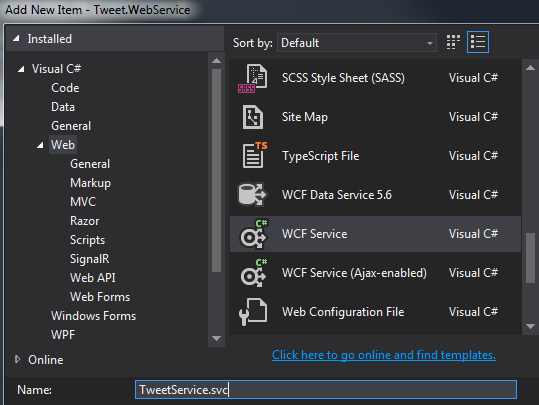
Expose the Above Business Functionality via a WCF Web Service

Now that we have had a quick trip of our TweetBL Project, it is time to jump into some real code...  
Let us start by adding a new project called Tweet.WebService - this will be our WCF project.

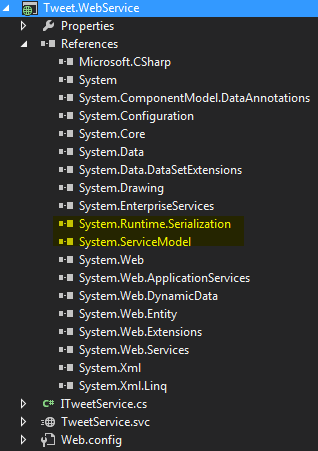




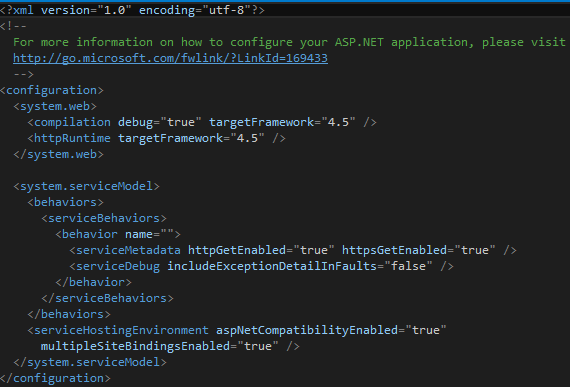
Now, we will add a WCF Service to Tweet.WebService project.  
Go to add new item (on Tweet.WebService project) and follow the screen-shot below:



As soon as we add the service, it makes a few changes to the project.  
Serialization and ServiceModal references have been added along with a few other files!



*Web.config*file has also been modified!  
One thing to note here is that as soon as we add a WCF Web Service, *Web.config*file gets updated with the default configuration for SOAP over HTTP based WCF Services...



We can see a collection of behaviour(s) has been defined for our WCF Service, especially, one for our Web Service.  
Behaviour defines run time aspects of our service.

You are more than welcome to add, remove or modify these settings but for this article we will stick to the defaults.  
Following the **KISS concept here - Keep it simple stupid!**

Below is a **short explanation of the provided WCF configuration:**

This setting allows to send Meta-data over HTTP and HTTPS using get verbs.

Hide   Copy Code

*//*

*// Web.Config snippet*

*//*

<serviceMetadata httpGetEnabled="true" httpsGetEnabled="true"/>

We are setting includeExceptionDetailInFaults to false so that if any error occurs on the server, we will not send details of the exception back to the client.

Hide   Copy Code

*//*

*// Web.Config snippet*

*//*

<serviceDebug includeExceptionDetailInFaults="false"/>

The configuration setting below defines how the Web Service will be hosted. Setting aspNetCompatibilityEnabled to true enables us to integrate with the ASP.NET pipeline in IIS.  
And finally, setting multipleSiteBindingsEnabled to true allows us to associate our web service with  
multiple site bindings configured on the IIS.

Hide   Copy Code

*//*

*// Web.Config snippet*

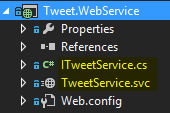
*//*

<serviceHostingEnvironment aspNetCompatibilityEnabled="true"

multipleSiteBindingsEnabled="true"/>

All the above settings are set for us by default... so we do not have to worry about any of it!

**Now, an Interface and an svc file were also added to the project**. Let us focus on them now.



WCF uses Interfaces to define the capabilities of each individual service and ***ITweetService.cs***represents  
that design and purpose.

Hide   Copy Code

*//*

*// ITweetService.cs snippet*

*//*

[ServiceContract]

public interface ITweetService

{

[OperationContract]

void DoWork();

}

ServiceContract attribute tells WCF that this interface *ITweetService.cs*can be used to serve out data as part of a Web Service.  
OperationContract attribute simply identifies the operations that can be executed by the service.

Finally, we have *TweetService.svc*.

Hide   Copy Code

*//*

*// ITweetService.svc (right-click: View Markup) snippet*

*//*

<%@ ServiceHost Language="C#" Debug="true"

Service="Tweet.WebService.TweetService" CodeBehind="TweetService.svc.cs"%>

The ServiceHost directive (above screen-shot) is specific to WCF - it enables all the binding and the configuration that we explored in the previous sections.

The code behind file is very clean as all the WCF specific configuration has been abstracted away and is specified  
in the ITweetService Interface.

Hide   Copy Code

*//*

*// ITweetService.svc.cs Code behind file snippet*

*//*

public class TweetService : ITweetService

{

public void DoWork()

{

*// Lets do some work*

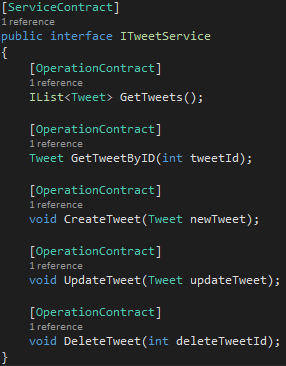
}

}

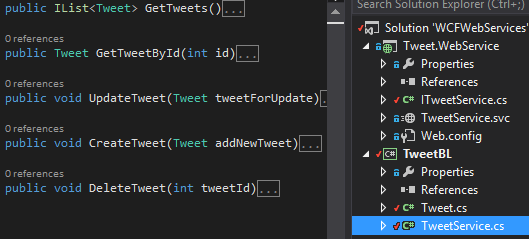
**Now Let Us Incorporate Our Business Code into this Newly Created WCF Web Service**

We will expose all the relevant methods from TweetBL Project - TweetService class to manage tweets as a Web Service.  
Let us start with exposing those methods via the ITweetService Interface of the Tweet.WebService Project.

Add methods to ITweetService Interface of the Tweet.WebService Project:



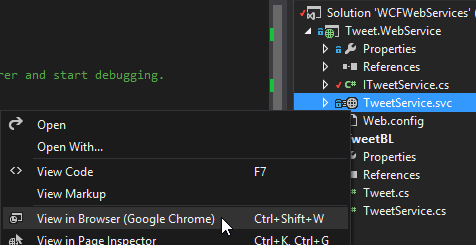
Now, we must implement the interface *ITweetService.cs*in class *TweetService.svc.cs*. I will implement the methods by tapping into our business functionality code defined in: Project TweetBL - Class *TweetService.cs*.



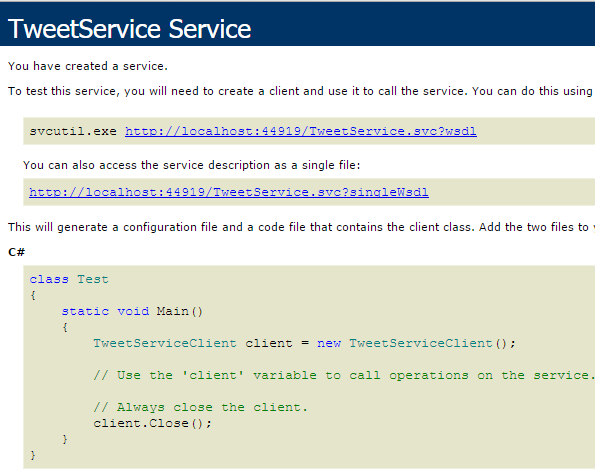
Implementing interface *ITweetService.cs*in class *TweetService.svc.cs*.



Now, after implementing & building the project - go to *TweetService.svc*, right click and select view in browser.

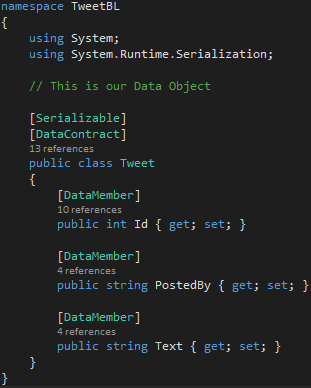


This will show us the generated HTML page for our web service, which we can use to build our proxy clients (the clients that consumes our service.)



Finally, we are just one step away from testing our service. Let us mark our data object *Tweet.cs*(our Transportclass) in project TweetBL with the following attributes:

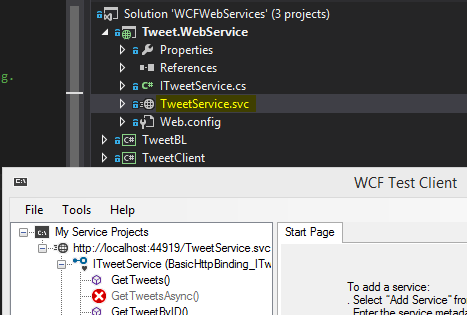
* [DataContract]: Marking our data transport class *Tweet.cs*(TweetBL Project) with this attribute
* [DataMember]: Marking all the fields in class *Tweet.cs*(TweetBL Project) with this attribute



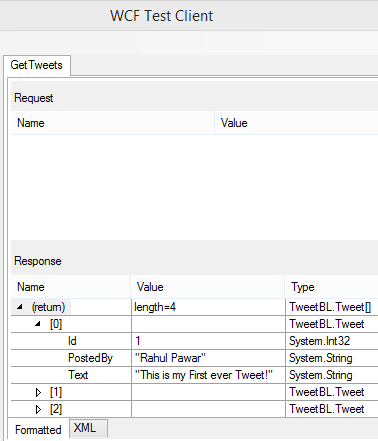
and now we are ready to Test our WCF Service!  
**Note\*** Data Contracts are explained [here](https://www.codeproject.com/Articles/884992/Simple-Demo-WCF-Web-Services#DataContractsExplained).

Employ WCF Test Client to Test Out Our Newly Created Web Service

In the previous section, we finalised the implementation of interface *ITweetService.cs*in class *TweetService.svc.cs*.  
It is now time to run a quick test... Set the Tweet.WebService Project as the start-up project.  
Select class *TweetService.svc*and press F5 to launch the WCF Test Client.



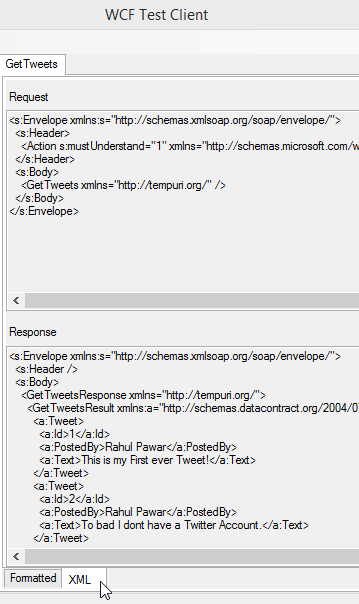
Select the GetTweets() method and double click - this will start a Form for us...  
The GetTweets() service method does not take any parameters so we can just click Invoke button:



The above screen-shot displays the result of GetTweets() service call - the result is displayed in a nice formatted data table!

It returned a list of 4 elements.  
Our Business Layer Project TweetBL - class *TweetService.cs*had been set up to create 4 initial tweets.  
All of those 4 records are returned.

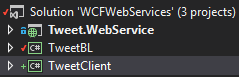
If you click on the **XML Tab**- you can look at the request and response SOAP envelopes.  
The Request envelope is generated by the client and the Response envelope is what we receive back from the WCF Service.



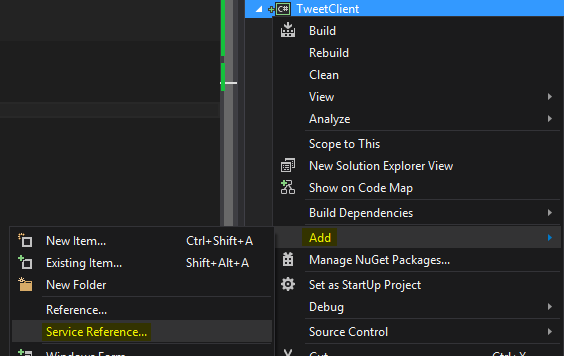
**Note**- You can test all the other Service Methods as well using WCF test client.  
I will leave the remaining service methods for you to play with on your own.

Create a Proxy Client for Our New Web Service

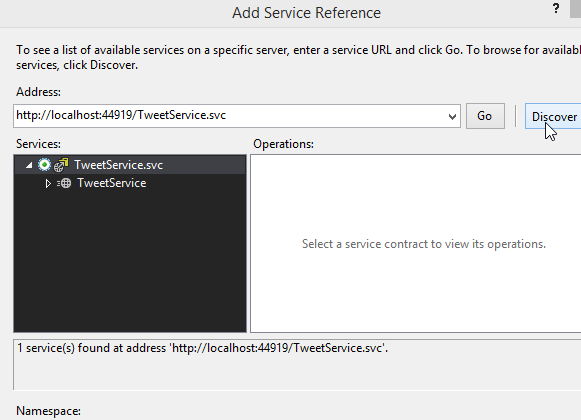
Add a new Console Application Project called TweetClient which will be the proxy client that will consume our Tweet Web Service.



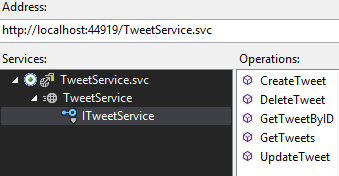
Now, let's add the Service Reference for our TweetService to TweetClient project:  
Right click on TweetClient project and go to Add -> Service Reference...



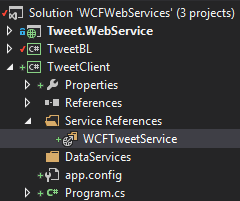
Click on the Discover button - as our service is in the same solution it will be discovered & displayed.



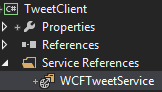
By expanding the TweetService, it will download the web service definition document from the service and it will get the service contract for the WCF Service.  
As you can see, we have our 5 operations that we exposed in our service contract being displayed.

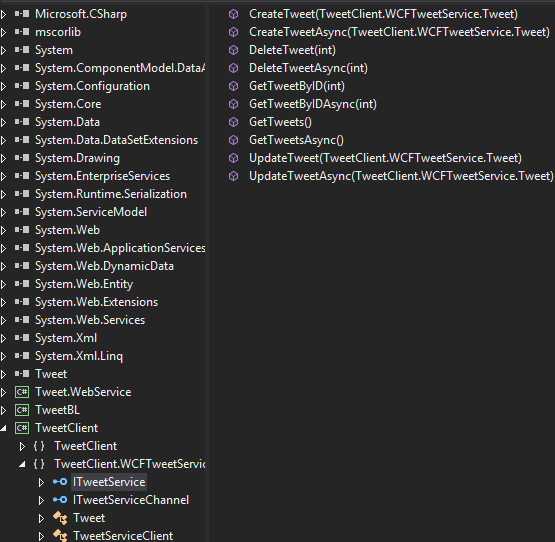


Click OK and the selected Service reference is added to a folder called '*Service References*'.



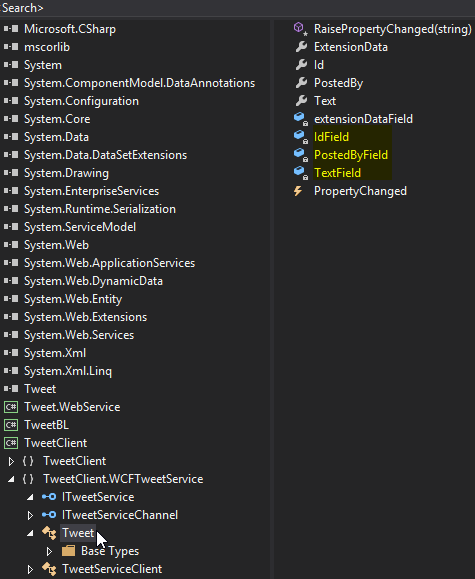
If you double click on the newly added service reference, it opens up the Object Browser.  
If you select the ITweetService interface, we have some interesting details here to look at:





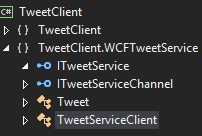
As you can see in the above screen-shot - for ITweetService interface, the Object Browser not only displays  
the 5 operations that we have exposed through our Service Contract, but it also displays Asynchronous versions of those methods. These Asynchronous methods have been added by default and it enables the client to create Asynchronous Web requests! to communicate with the server without blocking its main thread.

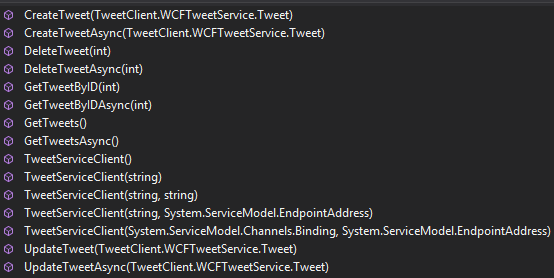
Also, have a quick look at our Tweet transport class...  
This is the transport class that we will use to send and receive data to and from the server for all our calls (example: create, get and update).  
It has all the fields that we require to do the above mentioned operations.



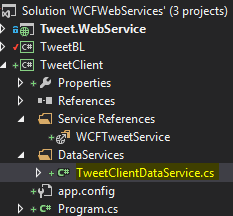
Finally, let's look at TweetServiceClient...  
This is the concrete implementation of ITweetService. Also it has 5 different constructor definitions.

**Note\*** the default constructor will use the Web Address that we used to add the Service Reference!  
The other constructors are made available to give us different options to configure our service to use different servers example a test server or a production server or use both.  
Using these constructors, you can define what Endpoint the service should be talking too.  
In this article, we will just employ the services of the default constructor.





Now that we have our service reference, let us define our TweetClientDataService.



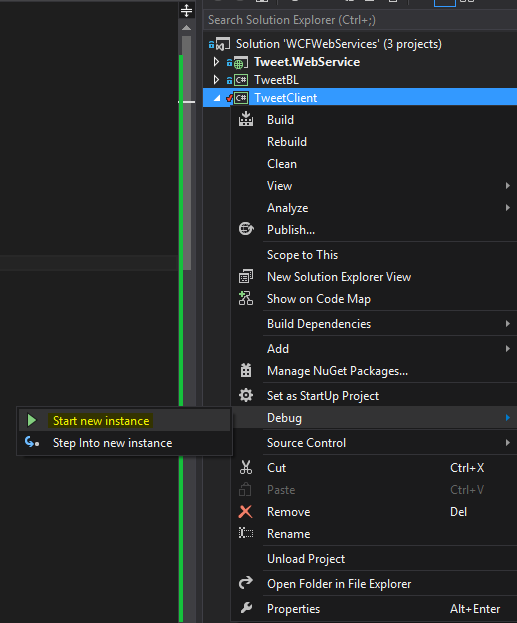


In TweetClientDataService implementation (above screen-shot), I am using reference to TweetServiceClient.  
TweetServiceClient is providing us with a lot of functionality under the covers!  
**Example**: TweetServiceClient - is doing all the serialization and de-serialization behind the scenes  
for request and response SOAP envelopes.  
It is also opening the connection for us and sending and receiving the information on our behalf over the network.  
So, it has really abstracted away a lot of complications that one has to face when communicating through  
RPC SAOP style Web Services!

Test Our Above Created Proxy Client

Wow! here we are... we have almost made it through!  
I have created a Program Class in the TweetClient project and have introduced some Command Prompt  
interface to interact with our Tweet Web Service operations. I will leave the Program Class in TweetClient  
Project for you to explore as it is fairly simple.

To run the Program Class in TweetClient Project, follow the screen shot below:



A pretty (you decide!) Command Prompt Interface will pop-up using which you can invoke our Tweet Web Service Operations!



**Have fun playing with our TweetService using TweetClient!**

Final Thoughts

It's time for me to sign off. Feel free to ask questions, provide feedback and everything in between as I am on the same boat with you.

P.S.: The Boat is called "Burn and Learn".