

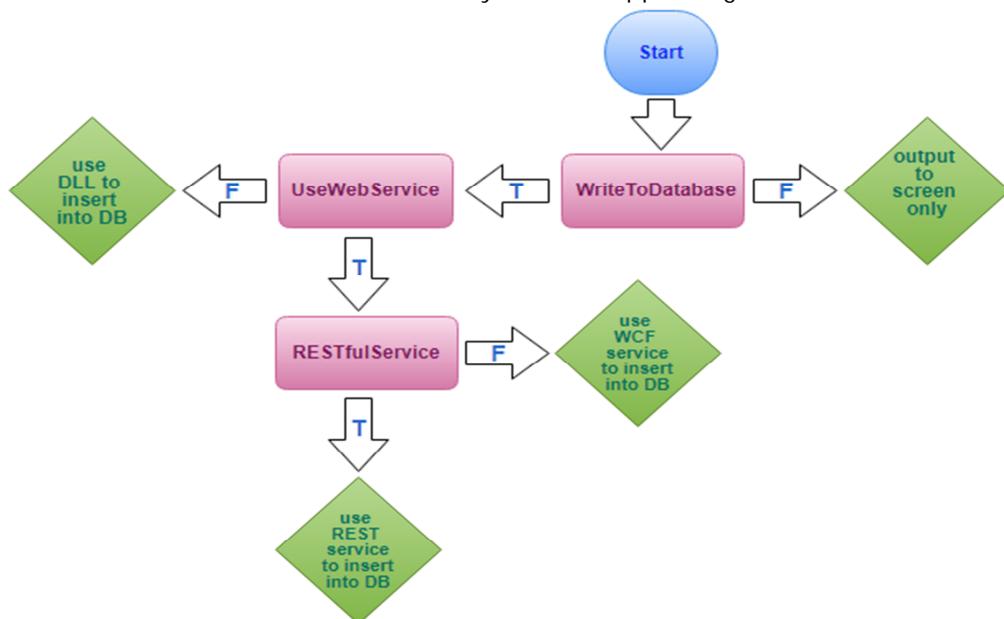
There are 5 projects in this solution:

1. Console app HelloWorld
2. DII Test
3. RESTfulDemo
4. WcfService1
5. UnitTestHelloWorld

I added 3 different APIs for this console app to use. There are options to select which API to use, or not to use any API, in app.config of HelloWorld project.

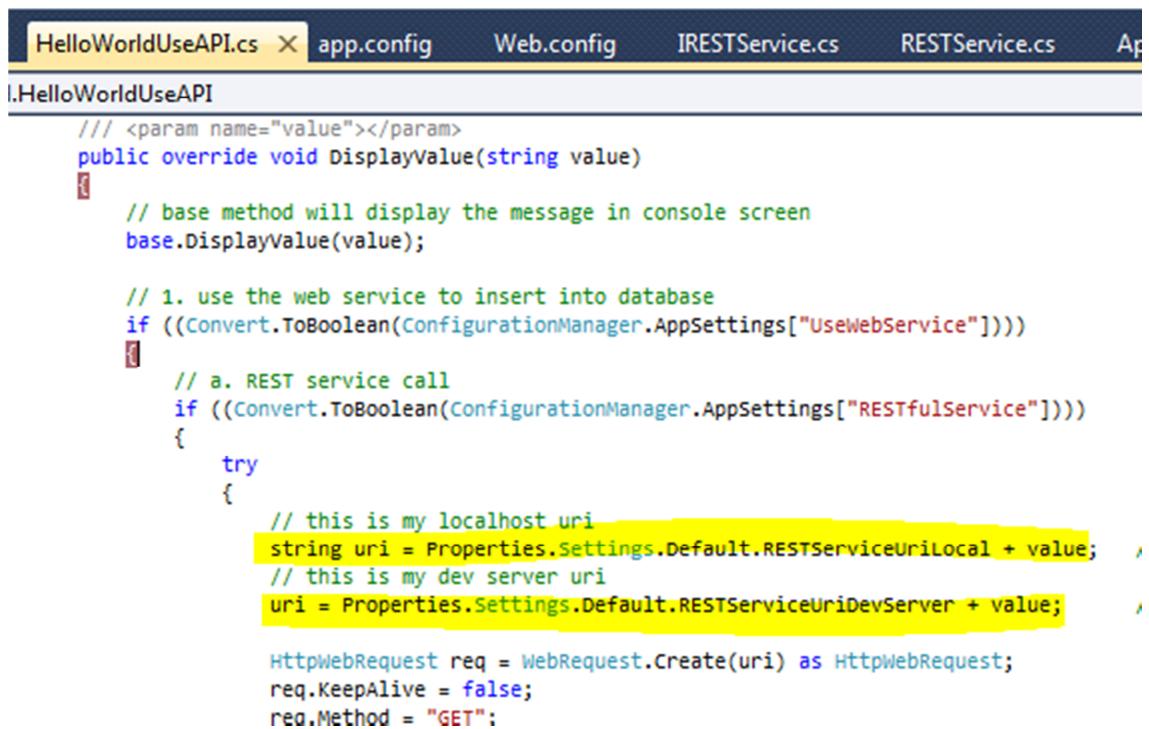
```
<?xml version="1.0"?>
<configuration>
  <configSections>
    <sectionGroup name="applicationSettings" type="System.Configuration">
      <section name="HelloWorld.Properties.Settings" type="System.Configuration.PropertySection, System.Configuration, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f1434493e" />
    </sectionGroup>
  </configSections>
  <appSettings>
    <add key="WriteToDatabase" value="true"/>
    <add key="UseWebService" value="true"/>
    <add key="RESTfulService" value="true"/>
  </appSettings>
  <startup><supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.0" />
  </startup>
  <applicationSettings>
    <HelloWorld.Properties.Settings>
      <setting name="HelloWorld_WCFServiceHW_Service1" serializeAs="String">
        <value>http://192.168.170.19:8018/Service1.svc</value>
      </setting>
    </HelloWorld.Properties.Settings>
  </applicationSettings>
</configuration>
```

Flowchart below shows how to set the key values in app.config to call different APIs



Some notes here:

1. In HelloWorldUseAPI.cs, when the code calls REST service:

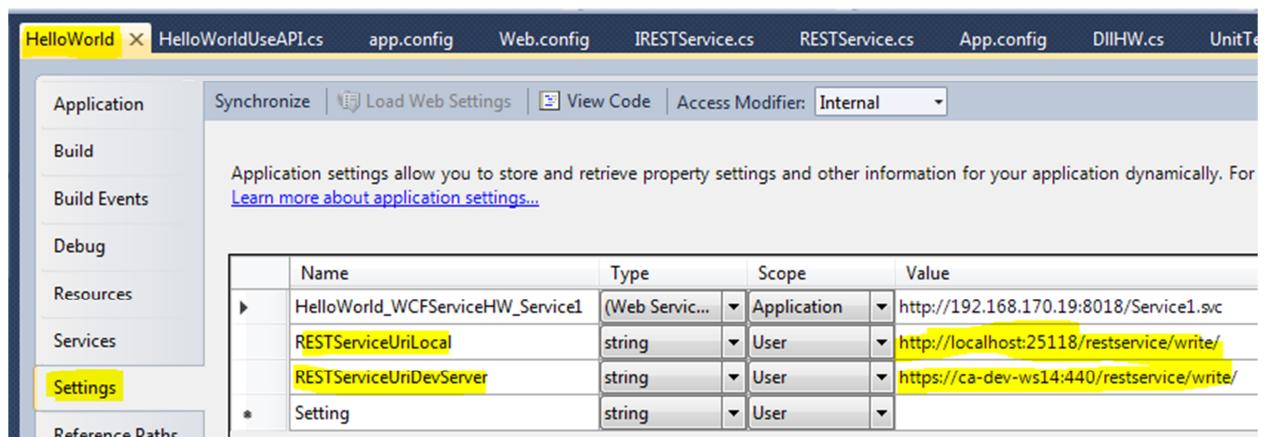


```
    /// <param name="value"></param>
    public override void DisplayValue(string value)
    {
        // base method will display the message in console screen
        base.DisplayValue(value);

        // 1. use the web service to insert into database
        if ((Convert.ToBoolean(ConfigurationManager.AppSettings["UseWebService"])))
        {
            // a. REST service call
            if ((Convert.ToBoolean(ConfigurationManager.AppSettings["RESTfulService"])))
            {
                try
                {
                    // this is my localhost uri
                    string uri = Properties.Settings.Default.RESTServiceUriLocal + value;
                    // this is my dev server uri
                    uri = Properties.Settings.Default.RESTServiceUriDevServer + value;

                    HttpWebRequest req = WebRequest.Create(uri) as HttpWebRequest;
                    req.KeepAlive = false;
                    req.Method = "GET";
                }
            }
        }
    }
}
```

The uri values for localhost and dev server are set in:



These uri's are for my localhost and my dev server, please change to your uri values to test.  
My dev server is internal so you will not be able to access it from internet.

A browser test result looks like this:

<https://ca-dev-ws14:440/restservice/write/LilyTest> has inserted the value into database table and returned "true" in the browser.

"Not secure" warning is because I made a self-signed certificate on my dev server which is not trusted by SSL.

This service works for HelloWorld console app too.

The screenshot shows a developer's workstation with two main windows open. At the top, a browser window displays the URL <https://ca-dev-ws14:440/restservice/write/LilyTest>. A red circle highlights the status bar message 'Not secure' and the URL itself. Below the browser, the status bar shows the response 'true'. In the bottom right corner of the browser window, another red circle highlights the URL again. To the right of the browser, the Microsoft SQL Server Management Studio (SSMS) interface is visible. It shows a query window titled 'SQLQuery1.sql - TRIPDBDEV.tdocs (ORT\lpan (52))' with the following script:

```
***** Script for SelectTopNRows command ****
SELECT * FROM [tdocs].[dbo].[TestTable]
```

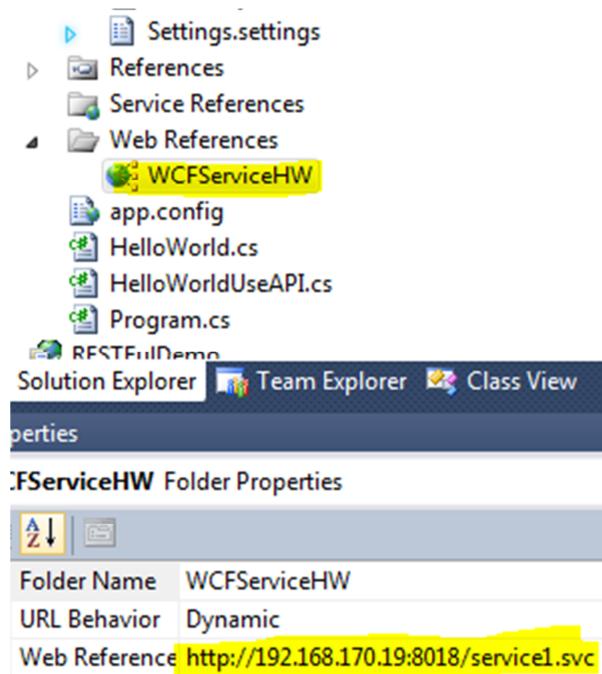
The results pane shows a table with two columns: 'ID' and 'Value'. The 'Value' column contains the value 'LilyTest'. This row is also circled in red. At the bottom of the SSMS window, a green checkmark indicates 'Query executed successfully.'

2. In HelloWorldUseAPI.cs, when the code calls WCF service, please comment one or the other, to try either the local reference or the web reference.

```
// b. WCF service call
else
{
    try
    {
        // when the service is added as a local reference
        WcfService1.Service1 service = new WcfService1.Service1();
        bool result = service.WriteValue(value);
        Console.WriteLine("Inserted into database by WCFService: " + result);
        Thread.Sleep(5000);

        // when the service is added as a web reference
        WCFSERVICEHW.Service1 svc = new WCFSERVICEHW.Service1();
        bool writeValueResult = false;
        bool writeValueResultSpecified = true;
        svc.WriteValue(value, out writeValueResult, out writeValueResultSpecified);
        Console.WriteLine("Inserted into database by WCFService: " + writeValueResult);
        Thread.Sleep(5000);
    }
}
```

When you try the web reference, please note to use my web reference. Because I published it on my internal dev server which will not be connected from outside.



Please publish the WCF service on your server, and then go to "Add Web Reference" with your own service link.

Add Web Reference

Navigate to a web service URL and click Add Reference to add all the available services.

URL: <http://192.168.170.19:8018/service1.svc>

## Service1 Service

You have created a service.

To test this service, you will need to create a client and use it to call the service. You can do this us the svcutil.exe tool from the command line with the following syntax:

```
svcutil.exe http://192.168.170.19:8018/Service1.svc?wsdl
```

You can also access the service description as a single file:

```
http://192.168.170.19:8018/Service1.svc?singleWsdl
```

This will generate a configuration file and a code file that contains the client class. Add the two files your client application and use the generated client class to call the Service. For example:

C#

```
class Test
{
    static void Main()
```

3. The connection string for the DLL is set in here:

I will delete my connection string value so you will put your own value in there.

The screenshot shows the Visual Studio Properties window for a project named "DIIHW [from metadata]". The "Settings" tab is selected. A note says: "Application settings allow you to store and retrieve property settings and other information for your application. Learn more about application settings...". A table lists three settings:

	Name	Type	Scope	Value
	ConnectionString	string	User	Data Source=tripdbdev.or...
*	useDatabase	bool	User	False
*	Setting	string	User	

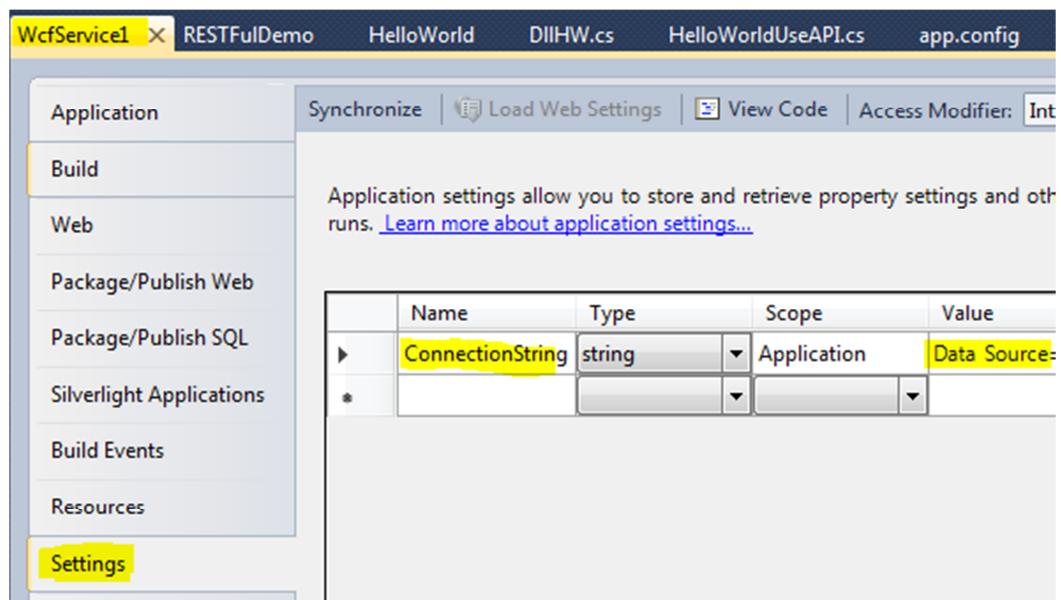
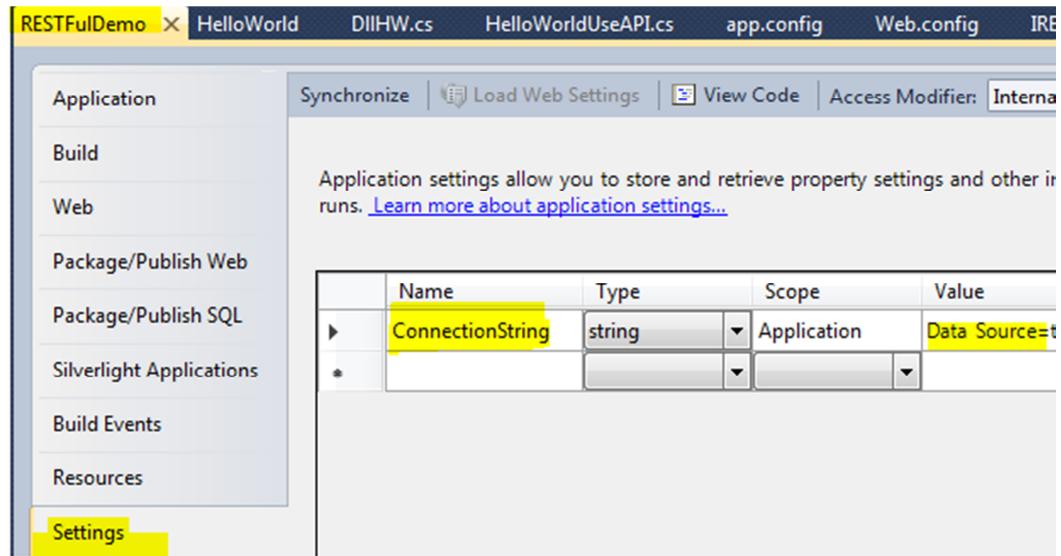
In DIIHW.cs, if you would like to test the database insert method, this is where you put your own query.

I have created a TestTable in dev DB and it works with my code.

```
try
{
    SqlConnection cn = new SqlConnection(dbConnectionString);
    cmd = cn.CreateCommand();
    cmd.CommandText = "INSERT INTO [TestTable] " +
        "(value) " +
        "VALUES (@Value)";
    cmd.Parameters.AddWithValue("@Value", value);
    cn.Open();
```

In the DLL I also added GetValue() method. I thought you also wanted to have options of either getting value from the database or just hard coding "Hello World". Then I realized you just asked for writing to database, but not reading from database. Anyways I added the option of reading from database too, and it's turned off for now in the config file: useDatabase is set to False.

4. I will also delete the connection string in RESTFulDemo and in WcfService1. Please add your own connection strings in there if you will test the code.



5. I did unit test for the DLL, to check if the return value is "Hello World", and the database insert method. I did not do the unit test for the REST service and WCF serve, because the only method is to insert into the database, and they will be the same as the insert test for the DLL.
6. So I think the DLL is easy to use, but it has to be added locally, and if there are changes in the DLL then we will have to rebuild the DLL, also the application that uses the DLL will have to re reference the DLL and rebuild too. The WCF service can be consumed from the server, and it's stable, but if there are changes in the service, the client app will also have to re reference and rebuild the code. Also we have to make sure all the configurations match on the server side and on client side, otherwise it will not work. Especially when SSL is enabled on the server, the configurations could be a little complicated for the client app. The REST service is easier for client app to use because we don't need to add reference, if there are changes in the service, as long as the uri is not changed, the client side doesn't have to change anything.