Create your first RESTful service with WCF 3.5

What is REST: REST is a term coined by Roy Fielding in his Ph.D. dissertation to describe an architecture style of networked systems. REST is an acronym standing for Representational State Transfer.

Read more here about REST on my blog

Problem with SOAP: In particular web scenarios, it is required to have high interoperability and we need to restrict to the use to of just XML message for transmitting over HTTP and can't take advantage of high end WS* specification part of protocol. For security, we can still reply in HTTPS and SSL. So now it's becoming pretty common to use SOAP without WS* specifications when we need to achieve high interoperability. This is where REST comes into picture. In nutshell

Hence many used SOAP without WS specification*

1) Often called simple SOAP

2) Increases interoperability potential

3) Framework / tools hide XML / HTTP, i.e. tool support for generating clients and server from WDSL definitions which create XML schema automatically and that schema hides underlying XML details. So that way developers still go for SOAP because of tool support from all major vendors, even though that may not be able to take advantage of lots if additional protocols that were designed to support.

What is POX: When we do not really need SOAP?

It is just exchanging XML message over HTTP without SOAP, referred to as plain –Old XML (POX).

1) Using this interoperability is virtually guaranteed.

2) But working with this involves direct coding for XML and HTTP, which is complex at times. Have to use various XML API for writing XML based codes, and has to be familiar with HTTP stack for actually programming logic for sending and receiving messages. This is one main reason why lots of developer switch back to SOAP programming model, which have tool support and automatic code generation especially at the client side.

POX applications can be defined in a RESTful way when actions are defined by HTTPS verbs.

Understanding REST

1) REST is not a protocol like SOAP, it an architectural design.

2) An architecture for building services that build on HTTP.

Fundamentally different from SOAP

1) SOAP defines a transport neutral model focused on custom operations

2) REST defines transport specific (HTTP) model focused on resources.

Builds on a uniform interface and common data formats

1) HTTP methods: GET, POST, PUT, DELETE, etc.

In SOAP we define and rely on custom service contracts/ operations.

In REST, we focus on defining resources and use unique URI to identify and represent these resources. Then we build services around these uniform interfaces instead of custom service contract that all services will use. So when we use HTTP to implement RESTful services, these uniform interfaces are defined by HTTP methods, i.e. GET, POST etc, so those operations that would be able to invoke on our resource. And when we get a resource then would be retrieving a representation of that resource over the protocol. So what we get is a representation of our resource and we can sue wide variety of formats to represent that resource. But at the end we will always use uniform resource to interact with those resources that are exposed by service.

2) Data Formats: HTML, XML, JSON, RSS

REST versus SOAP:

SOAP emphasizes Verbs while REST emphasizes nouns or resources.

With SOAP, we define custom operations (new verbs) tunnels through POST.

With REST, we define resources and use a uniform interface to operate on them (HTTP verbs).

So I can GET a User {}, to retrieve a representation, or I can POST a User {} to create one, or I can DELETE User {} to delete a representation and I can PUT the User {} to update in the system. So I can use any that of standard HTTP operations with any of these resources. So that's the beauty of REST, we have standard uniform interface that can be apply equally to all of our resource that we expose through that service, and they provide standard CRUD operation that we typical exposed through SOAP services as well, and when we do that we define standard representation for those resource that we would be sending back to the wire.

What is Resource orientated architecture (ROA)

With REST, we build resource oriented architecture, because our focus is now the resource rather than operations. While writing ROA based application, we focus on following;

1) We focus on identifying and naming resources URIs.

2) How to represent then XML formats.

3) How to navigate between then HREFs

4) And we use a Uniform interface to interact with them.

HTTP defines the uniform interface, the HTTP verbs

1) Constraining operations set simplifies model.

Primary Benefits

1) Widespread interoperability, basically anyone who supports HTTP and XML message can use your service.

2) Scalability, because we are leveraging HTTP, we can now take advantage of the entire wide spread web infrastructure and optimization build around that specific protocol and those specific operations.

Importance of GET:

Web is primarily built on GET, in fact 90% of web traffic is GET, which is well defined operations and is thus have no side effects.

1) GET simply means retrieves the representation of a resource.

2) Should not cause any unsafe side effects.

3) POST, PUT, DELETE provides additional semantics. But with no guarantee.

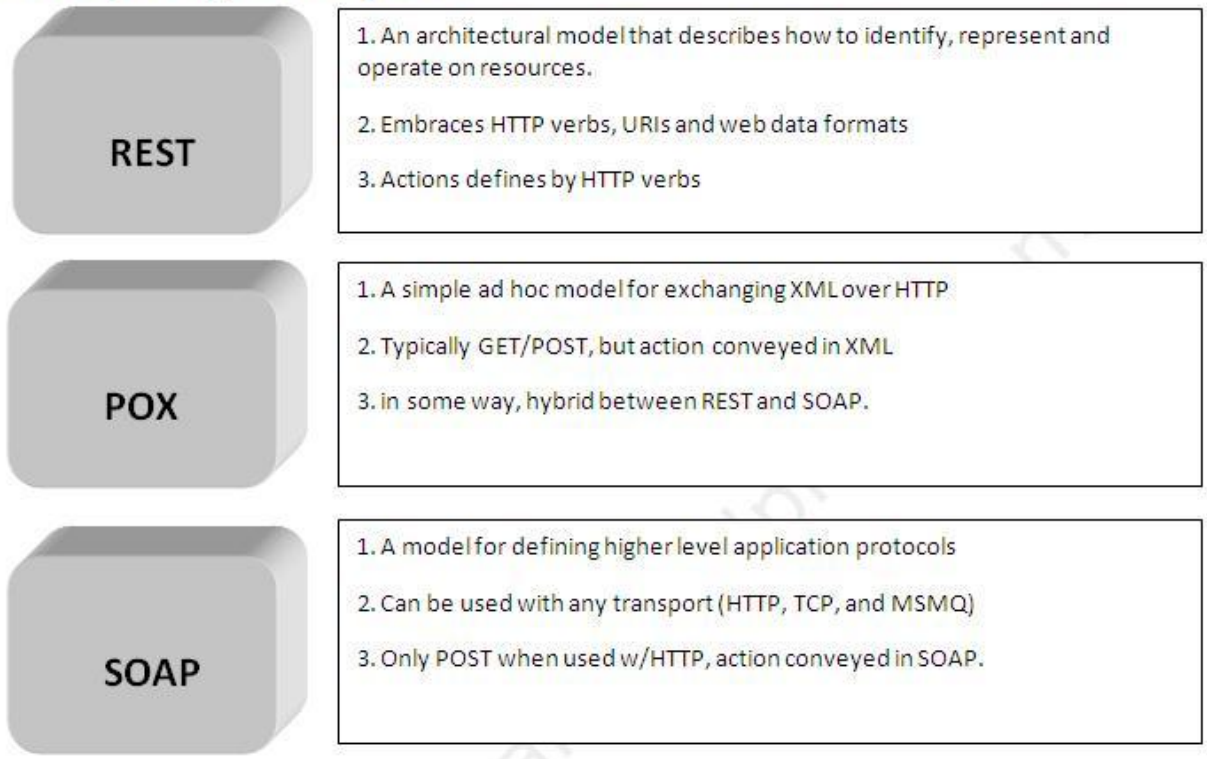
REST uses / embraces the importance of GET, unlike SOAP which uses POST.

1) While SOAP largely ignored its importance.

2) With REST we get the full advantage of web infrastructure.

3) With SOAP, we have to build that stuff ourselves.

Web Programming model comparison:



So BIG question. When to use what?

Each architecture, style and format has its own pros and cons.

*Use SOAP + WS**

1) In large enterprises

2) In complex situations where we need COM+ like capabilities.

3) Can be overkill in many distributed scenarios today.

4) Good tools support provided by the big SOAP vendors.

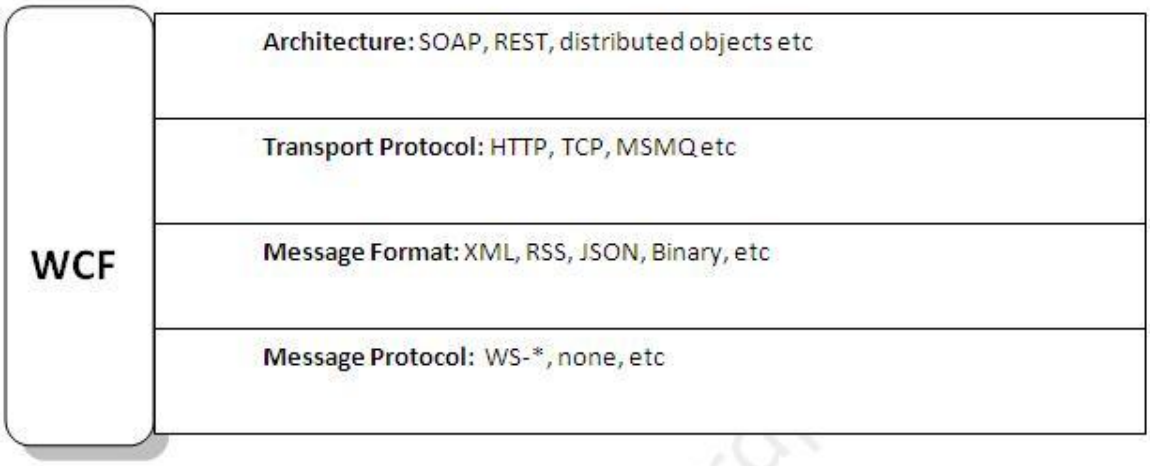
Use REST

1) When we need interoperable, scalable transfer of information.

2) Large web vendors adopting it (Amazon, Yahoo, Flickr)

3) But have limited tool support beyond HTTP and XML Stack.

WCF and support for programming models: WCF doesn't take sides, it provides a unified programming model and allows to use SOAP, POX, REST, and whatever data format.



WCF and support for REST.

1) Most of built in WCF bindings uses SOAP and WS* by default.

2) These binding have to be configured manually to disable SOAP.

3) WCF 3.5 comes with a new Web (REST) programming model, found in **System.ServiceModel.Web.dll.** This allows us to map HTTP requests to methods Via URI templates.

4) We can enable the web model with a new binding / behavior, apply to messaging layer using **WebHttpBinding** and Apply to dispatcher using **WebHttpBehavior**.

Now we will take WCF REST attributes one by one and will explore their purpose. They can found in System.ServiceModel.Web.dll

A) WebHttpBinding:

1) WebHttpBinding produces an appropriate HTTP based channel.

2) Produce an HTTP transport channel.

3) We can customize certain setting like cookies, proxy, security etc.

4) Security is handled by **WebHttpSecurity** (HTTP vs HTTS)

5) Produces a WebMessageEncoder (Support XML and JSON formatted messages,)

B) WebHttpBehavior.

After configuring binding, we enable web HTTP behavior on particular endpoint which uses web http binding.

1) WebHttpBehavior customizes the HTTP based dispatching logic.

2) It Overrides operation selection, serialization, and invocation.

C) WebGet Attribute

1) Use to actually map incoming HTTP GET requests to particular WCF operations.

2) We do so by providing a URI template to defining URI mapping

3) The URI template variable map method parameters by name.

4) Request/ response format control body format, i.e. which message format to use, JSON etc.

```
[ServiceContract]
    public interface IEvalService
    {

        [WebGet(UriTemplate="eval/{id}")]
        [OperationContract]
        Eval GetEval(string id);
```

We added WebGet attribute to define mapping from the incoming URI to the method. So here UriTemplate parameter `{id}` matches the method signature parameter `(string id)`. So when a request arrives with this parameter URI, those request will be handled by this method.

D) WebInvoke Attribute

1) It is meant to be used for all other HTTP verbs, other than GET, to WCF operations.

2) It add a mehods property for specifying the verb (default is POST).

3) Allows mapping UriTemplate variable.

4) Body is deserialiazed into remaining parameter.

```
[ServiceContract]
    public interface IEvalService
    {
        [WebInvoke(Method="POST", UriTemplate="evals")]
        [OperationContract]
        void SubmitEval(Eval eval);
```

E) UriTemplate

1) System.UriTemplate implements URI template syntax

2) UriTemplate syntax allows you to specify variable in URI space.

3) UriTemplate.Bind fills variable with actual values.

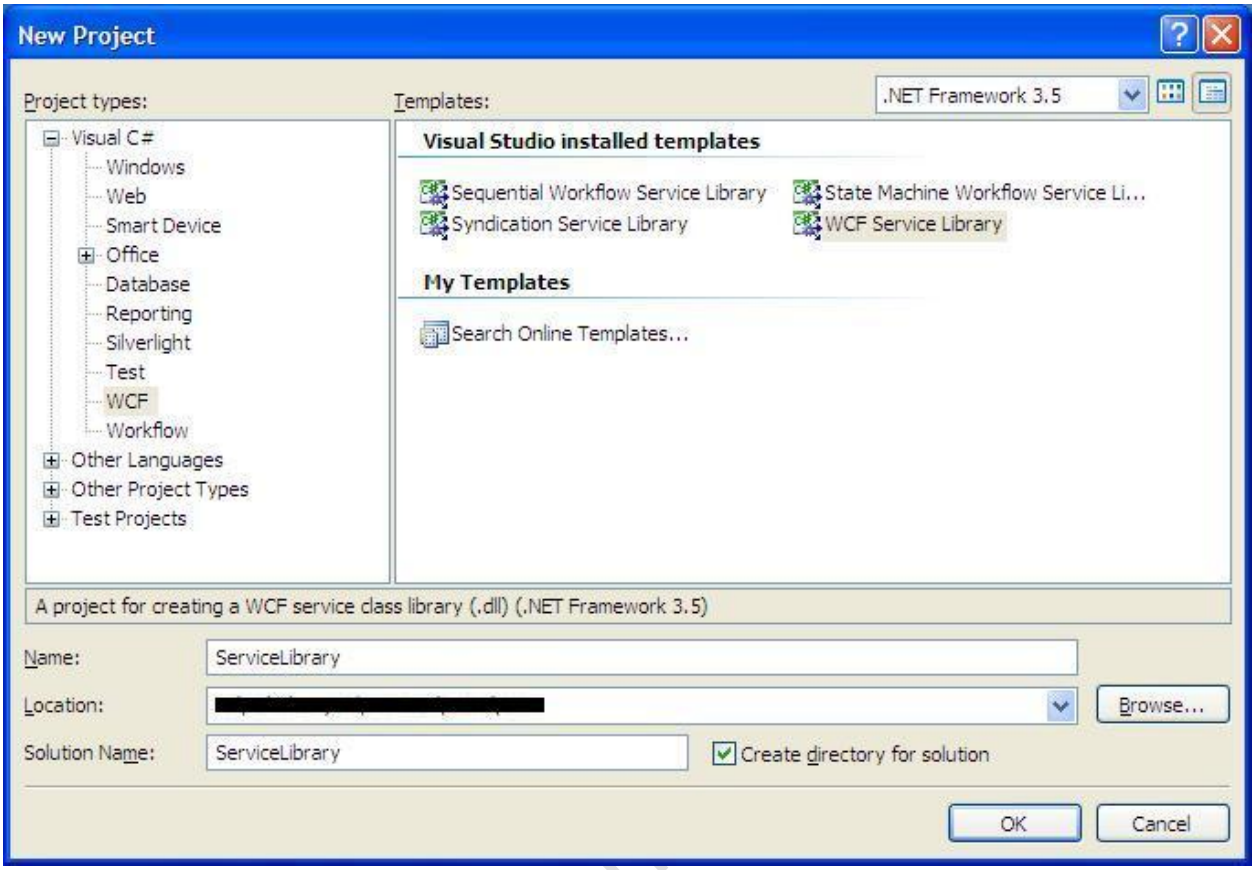
4) UriTemplate.Match verifies match and exact actual values.

5) Can use wildcard characters "*" to match anything.

```
UriTemplate = "services/evals?name={name}&detailed={detailed}
```

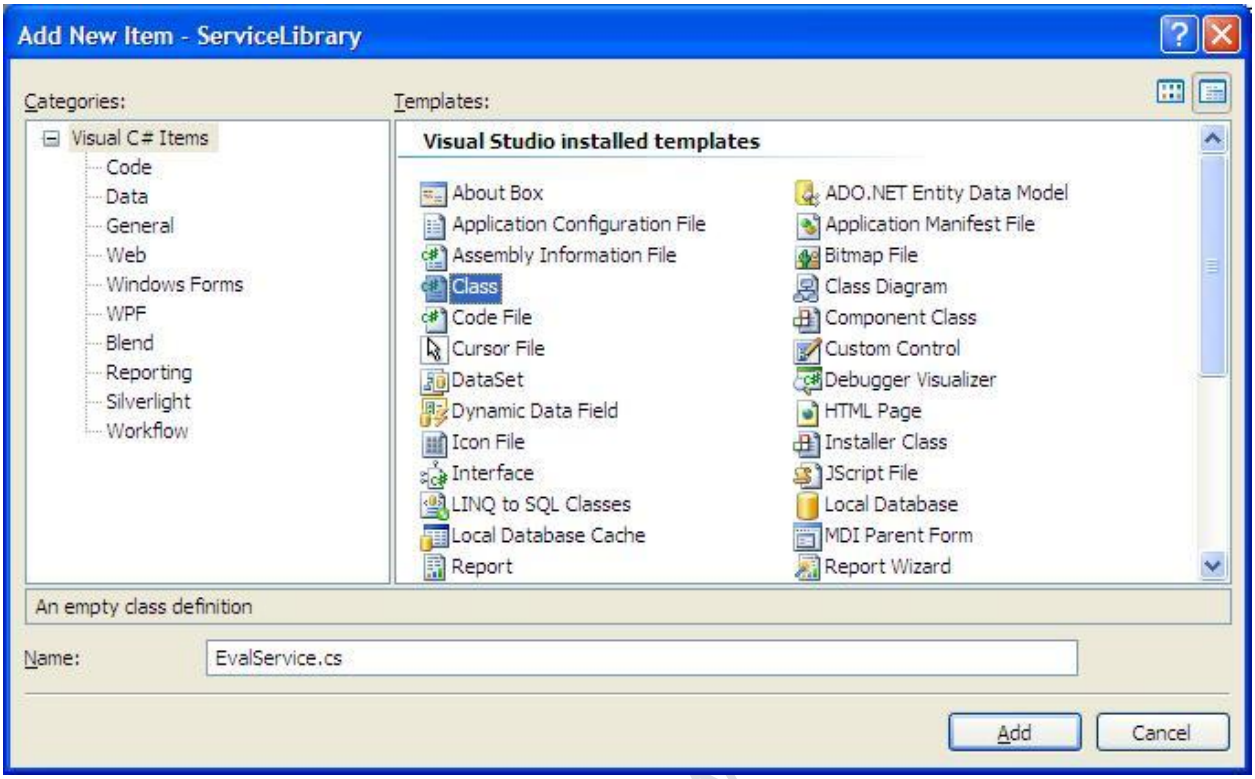
By now we know the important attributes , so its time we can start writing our first REST service with WCF.

Step 1: Create a WCF project and name solution file as RESTFulService. Give "ServiceLibrary" name to the project.



Step 2: Open ServiceLibrary project and delete all the default files.

Step 3: Add a class as new item to ServiceLibrary project and name is as" EvalService.cs"



Step 4: Open **EvalService**.cs and create class like below. This is actually a data contract for our service; we call it Eval class, i.e. evaluation class, which will have message submitter name, id, timesend, and comment.

```
[DataContract(Namespace = "http://vishalnayan.wordpress.com/evals")]
    public class Eval
    {
        [DataMember]
        public string Id;
        [DataMember]
        public string Submitter;
        [DataMember]
        public DateTime Timesent;
        [DataMember]
        public string Comments;
    }
```

Step 5: Next create a interface called **IEvalService** and put [**ServiceContract]** attribute.

```
  [ServiceContract]
    public interface IEvalService
    {
```

```
        [OperationContract]
        void SubmitEval(Eval eval);

        [OperationContract]
        Eval GetEval(string id);

        [OperationContract]
        List<Eval> GetAllEvals();

        [OperationContract]
        List<Eval> GetEvalsBySubmitter(string submitter);

        [OperationContract]
        void RemoveEval(string id);
    }
```

Step 6: Create a class **EvalService** which will implement this interface. We will attribute this class to support single instance mode.

```
[ServiceBehavior(InstanceContextMode = InstanceContextMode.Single)]
    public class EvalService : IEvalService
    {
        List<Eval> evals = new List<Eval>();
        int evalCount = 0;

        #region IEvalService Members

        public void SubmitEval(Eval eval)
        {
            eval.Id = (++evalCount).ToString();
            evals.Add(eval);
        }

        public Eval GetEval(string id)
        {
            return evals.First(e => e.Id.Equals(id));
        }

        public List<Eval> GetAllEvals()
        {
            return this.GetEvalsBySubmitter(null);
        }

        public List<Eval> GetEvalsBySubmitter(string submitter)
        {
            if (submitter == null || submitter.Equals(""))
                return evals;
            else
                return evals.Where(e =>
e.Submitter.ToLower().Equals(submitter.ToLower())).ToList<Eval>();
        }

        public void RemoveEval(string id)
        {
            evals.RemoveAll(e => e.Id.Equals(id));
        }
```

Lets take a brief look at what all this method does;

Before that we have to creat a collection object List of eval type for storage.

```
List<Eval> evals = new List<Eval>();
```

1)
```
public void SubmitEval(Eval eval)
{
    eval.Id = (++evalCount).ToString();
    evals.Add(eval);
}
```

Description: It take eval and add it in List. We auto generate the id before we add. This id will uniquely identify single eval submission.

2)
```
public Eval GetEval(string id)
{
    return evals.First(e => e.Id.Equals(id));
}
```

Description: it return the eval for the id submitted.

3)
```
public List<Eval> GetAllEvals()
{
    return this.GetEvalsBySubmitter(null);
}
```

Description: It return all the evals submitted. It actually call another method `GetEvalsBySubmitter()`.

4)
```
public List<Eval> GetEvalsBySubmitter(string submitter)
{
    if (submitter == null || submitter.Equals(""))
        return evals;
    else
        return evals.Where(e =>
e.Submitter.ToLower().Equals(submitter.ToLower())).ToList<Eval>();
}
```

Description: it take submitter name to return evals submitted by him/her. If no any name is provided it will return all evals.

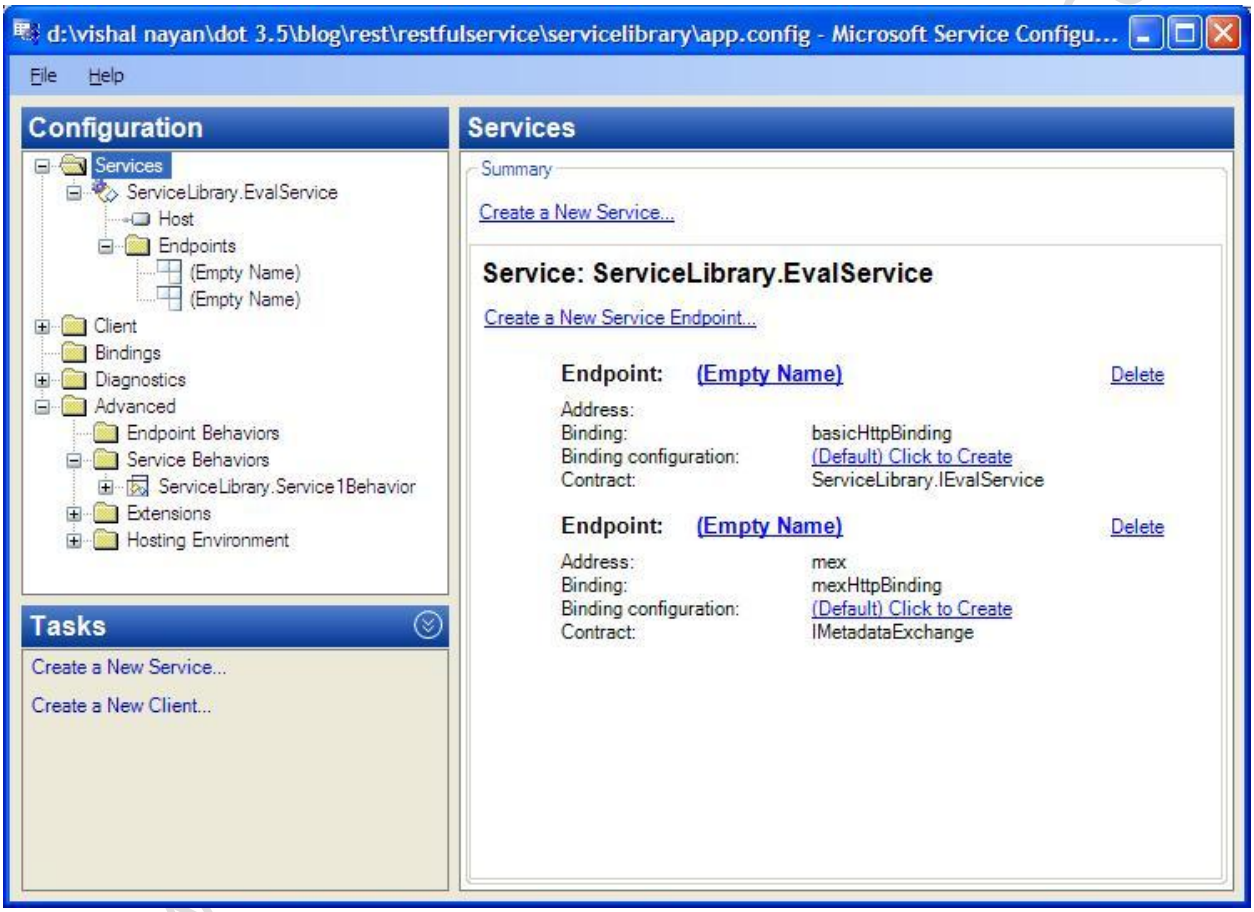
Step 7: Now open app.config to create and expose endpoints. Right click the app.config file and open it in WCF configuration wizard

a. create a mex endpoint with mexhttpbinding. Name it as mex

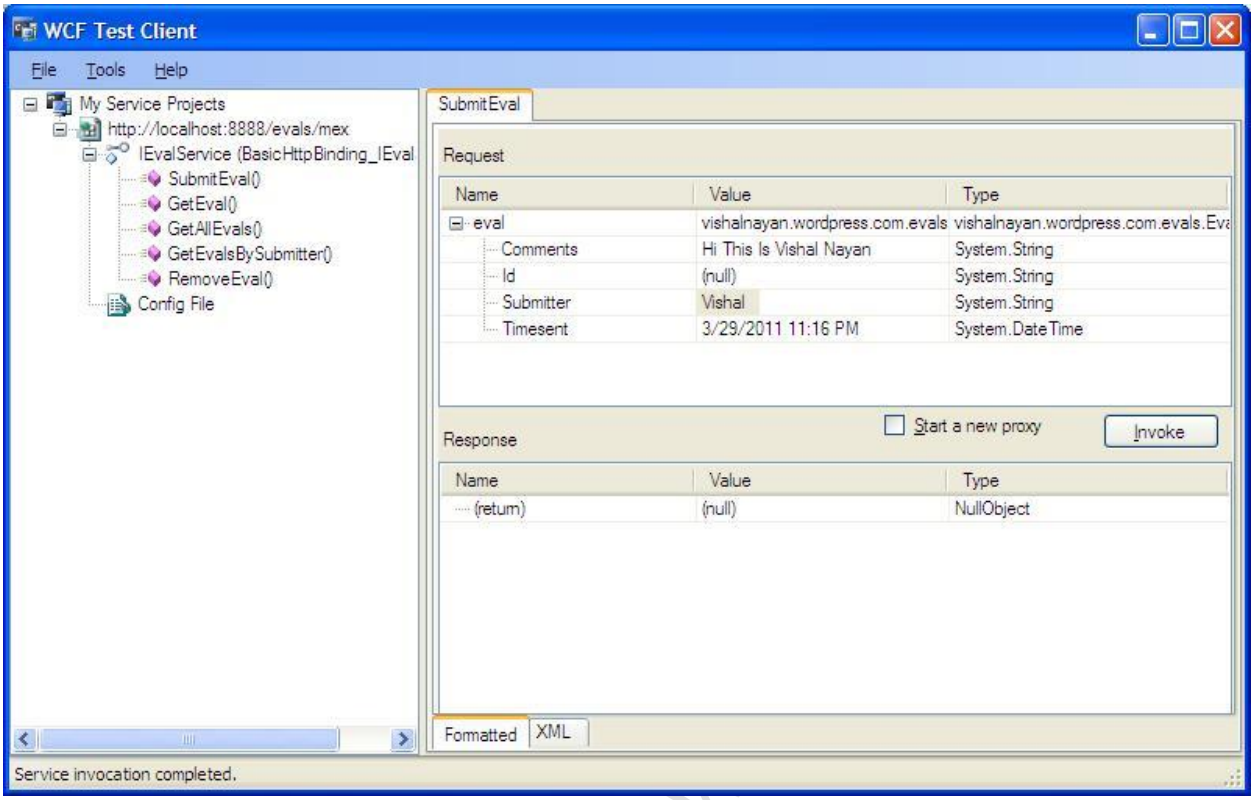
b. create a EvalService endpoint with basichttpbinding.

c. create a base address, http://localhost:8888/evals

It should look like this below;

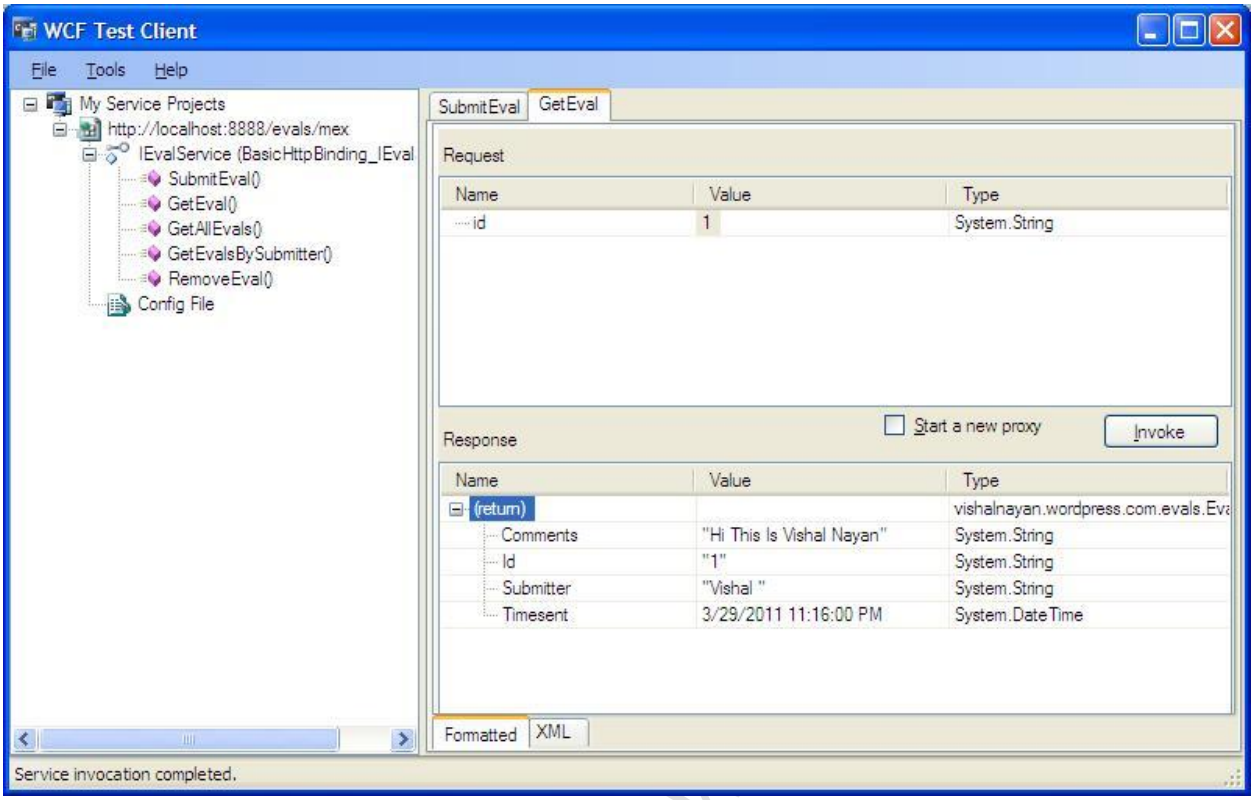


Step 8: Our service endpoint infrastructure is ready so press F5 to launch WCF cleint to test out service methods exposed by this endpoint



We are using this WCF test client to invoke SubmitEval method. Double click SubmitEval() from left hand side and you can see I am sending comment, submitter values and then click to invoke button to send this across proxy.

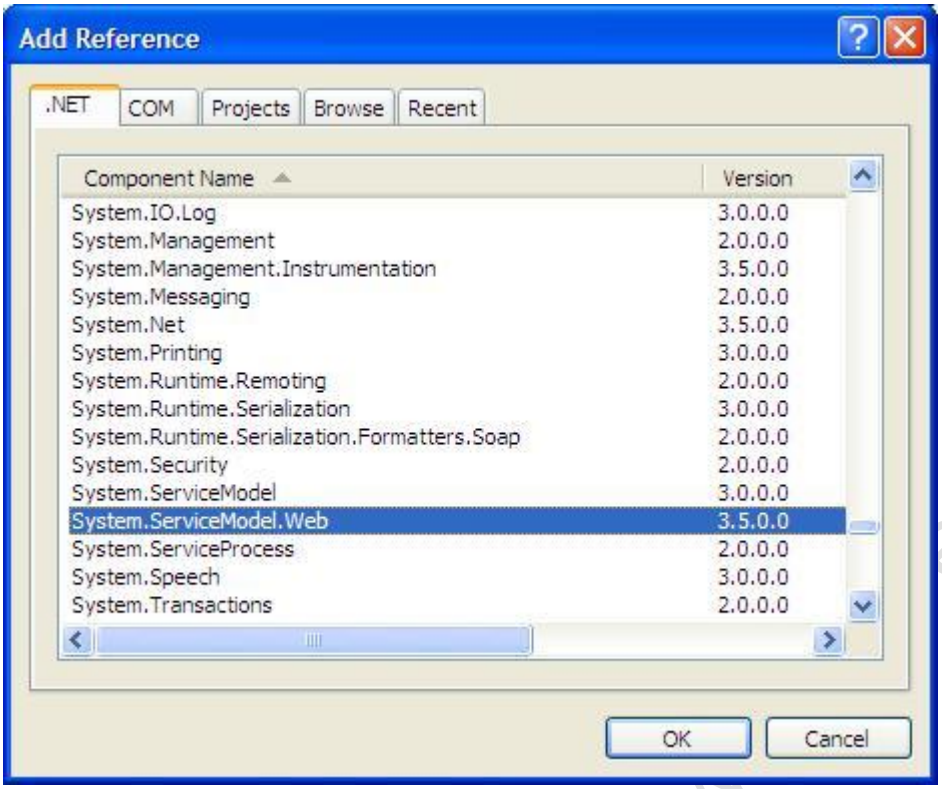
Later to see whether what we submitted was successfully got saved as eval in our list collection object , so for this we will invoke GetEval() method , so now double click to GetEval() and provide 1 as id .



You can see it return the evals message which we submitted through SubmitEval().

Step 9: Now this is happeing as SOAP message, so now we have to make our service method to support REST ful way of invocation.

1. Come back to ServiceLibrary project and add **System.ServiceModel.Web assembly** .



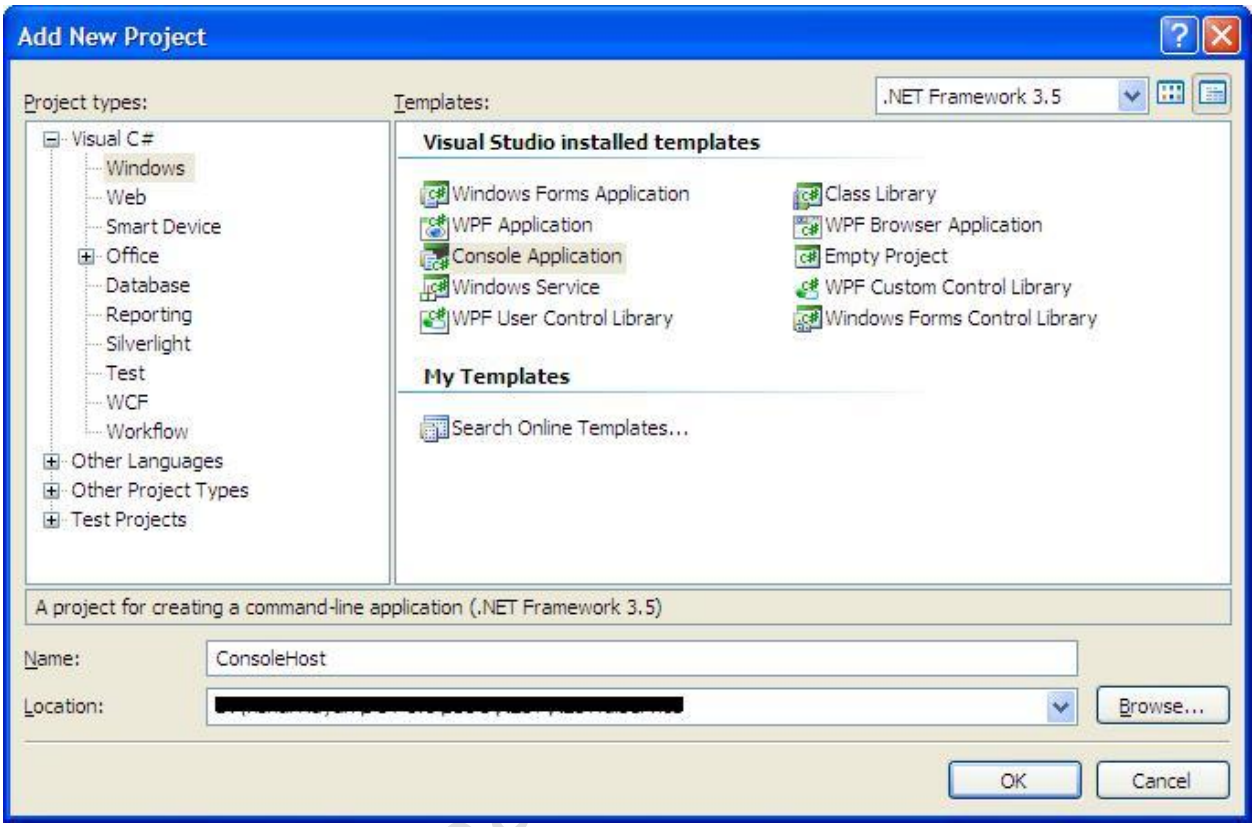
2. Come back to IEvalService interface and update it like below

```
[ServiceContract]
    public interface IEvalService
    {
        [WebInvoke(Method = "POST", UriTemplate = "evals")]
        [OperationContract]
        void SubmitEval(Eval eval);

        [WebGet(UriTemplate = "eval/{id}")]
        [OperationContract]
        Eval GetEval(string id);

        [WebGet(UriTemplate = "evals")]
        [OperationContract]
        List<Eval> GetAllEvals();

        [WebGet(UriTemplate = "evals/{submitter}")]
        [OperationContract]
        List<Eval> GetEvalsBySubmitter(string submitter);

        [WebInvoke(Method = "DELETE", UriTemplate = "eval/{id}")]
        [OperationContract]
        void RemoveEval(string id);
    }
```

Read the above part of article where we have discussed about **WebInvoke** and **WebGet** attributes.

There will be no change in EvalService class implementing this interface.

Step 10:. Right click RESTfulService solution and add a console project which will host out REST service now. Call it as ConsolHost.

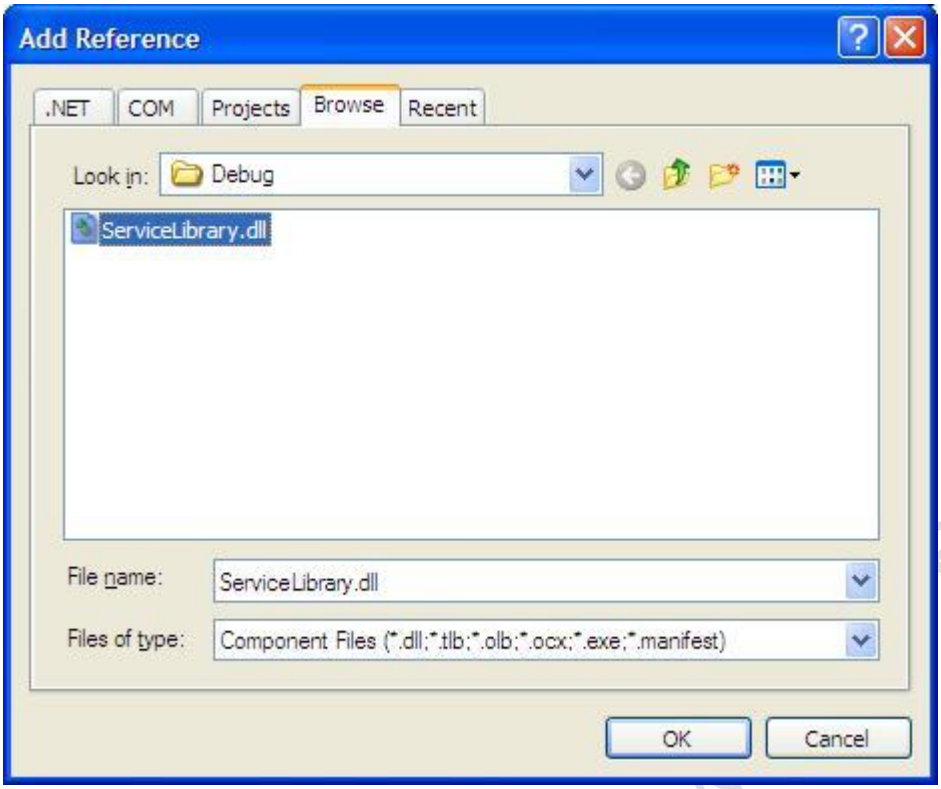


Step 11: Open ConsolHost and Left click reference to add following .NET assemblies

1) System.ServiceModel.Description;

2) System.ServiceModel;

Also browse and add ServiceLibrary assembly



Step 12: Open program.cs and write below codes;

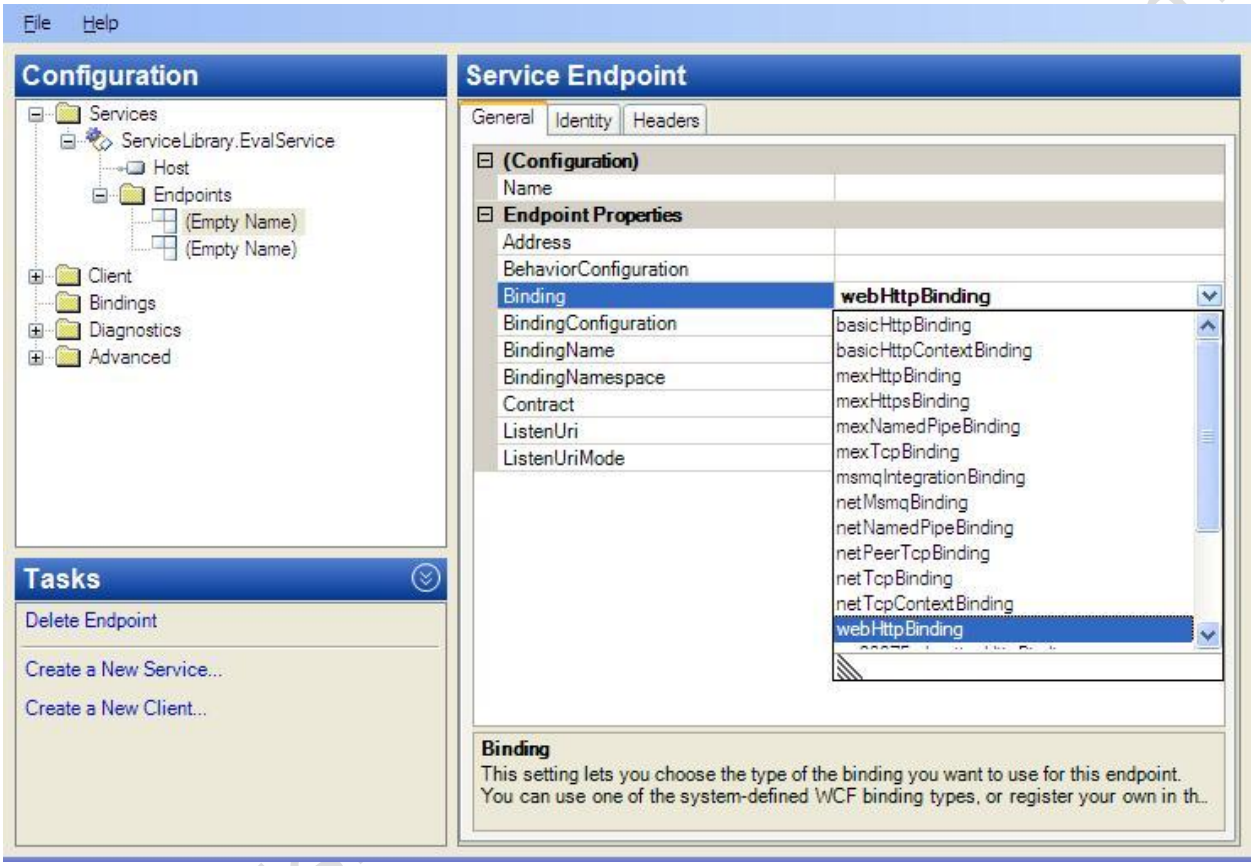
```
class Program
    {
        static void Main(string[] args)
        {
            ServiceHost host = new ServiceHost(typeof(EvalService));
            try
            {
                host.Open();
                PrintServiceInfo(host);
                Console.ReadLine();
                host.Close();

            }
            catch(Exception e)
            {
                Console.WriteLine(e);
                host.Abort();

            }
        }
        static void PrintServiceInfo(ServiceHost host)
        {
            Console.WriteLine("{0} is running with these endpoints",
host.Description.ServiceType);
            foreach (ServiceEndpoint se in host.Description.Endpoints)
                Console.WriteLine(se.Address);
```

```
        }
    }
```

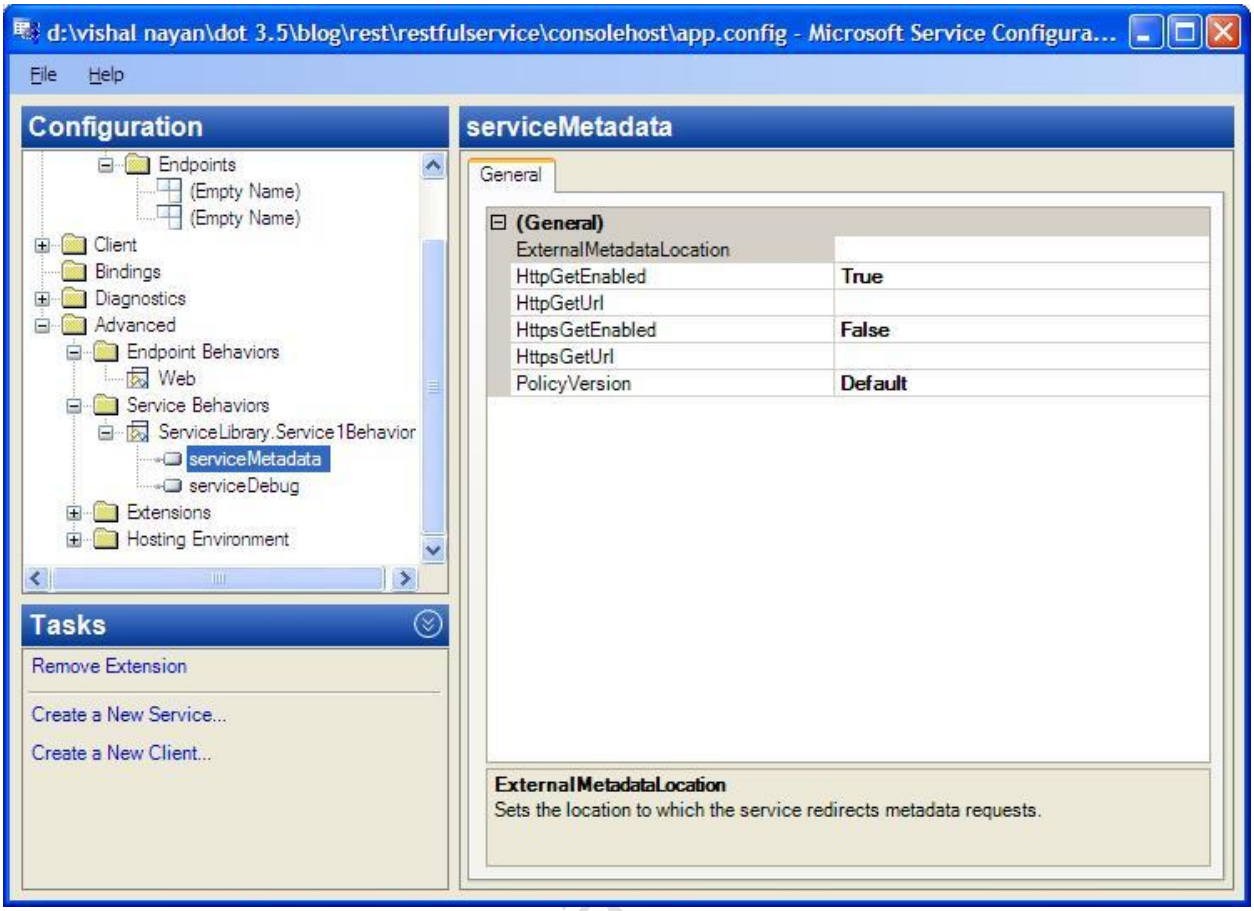
Step 13: Now we need configure host app.config file to expose REST methods via endpoint.

So open app.config file and configure it like below;

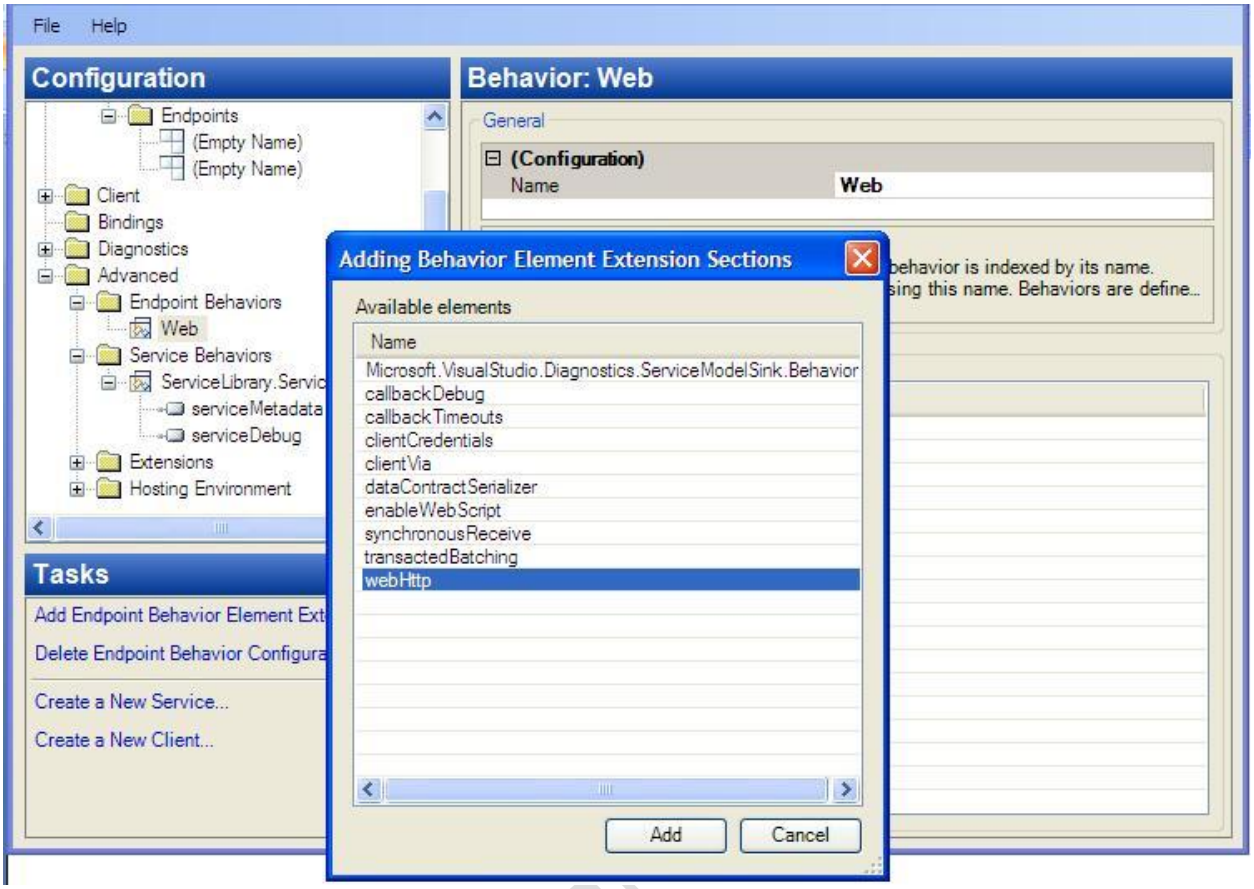
1. **Create a endpoint with WebHttpBinding binding configuration which support REST calls**



2. Create a service behavior configuration and get HttpGetEnabled true;



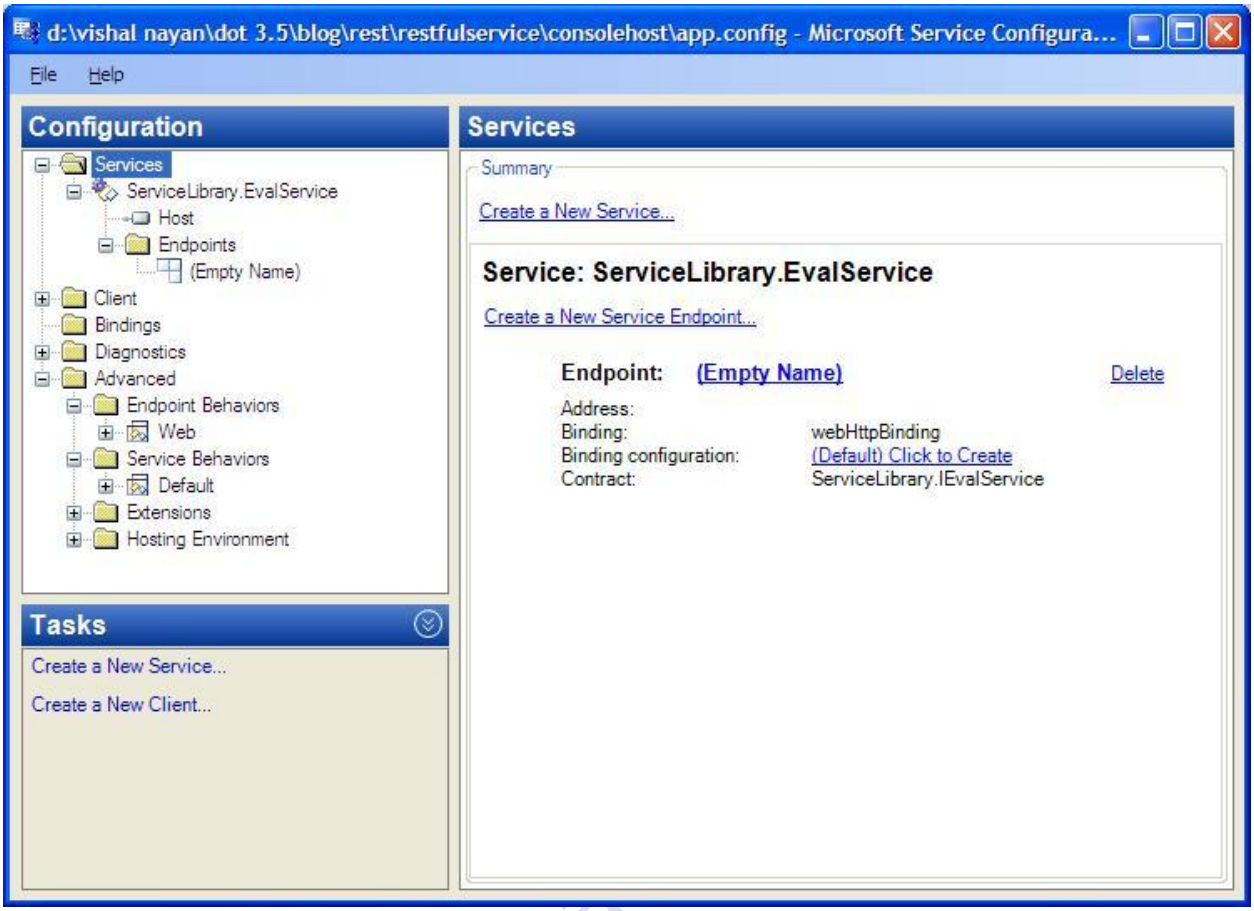
3. Create a new endpoint behavior configuration, name it Web and add web Http as endpoint behavior element extention. This is important to support REST invocation.



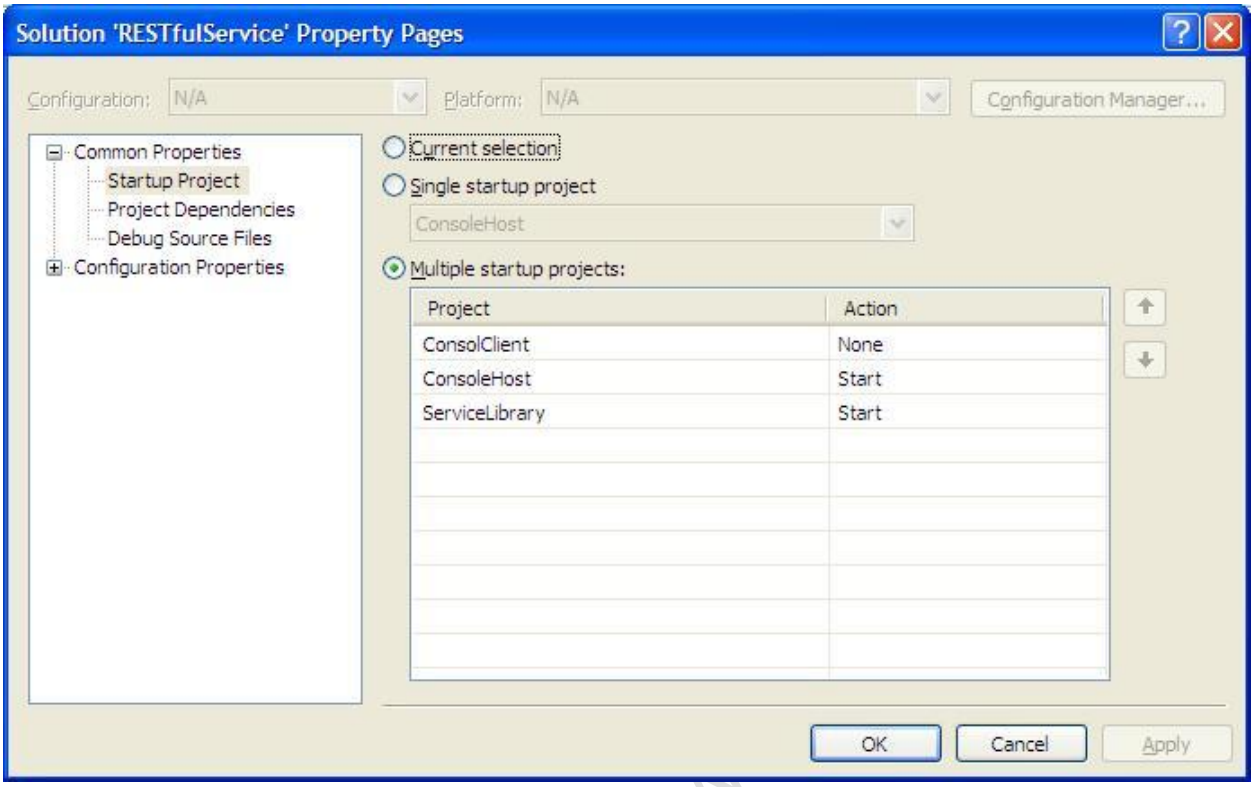
4. Add a Host Base address

```
<baseAddresses>
          <add baseAddress="http://localhost:8181/evalservice" />
        </baseAddresses>
```

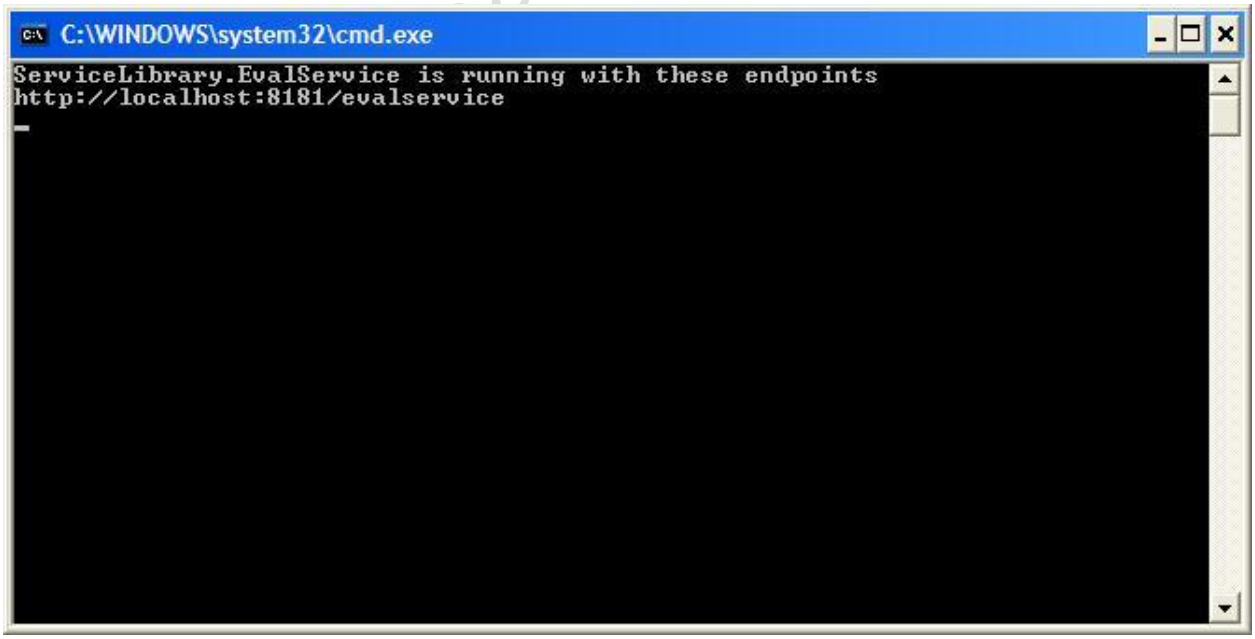
5. Now whole Host app.config file will look like this.

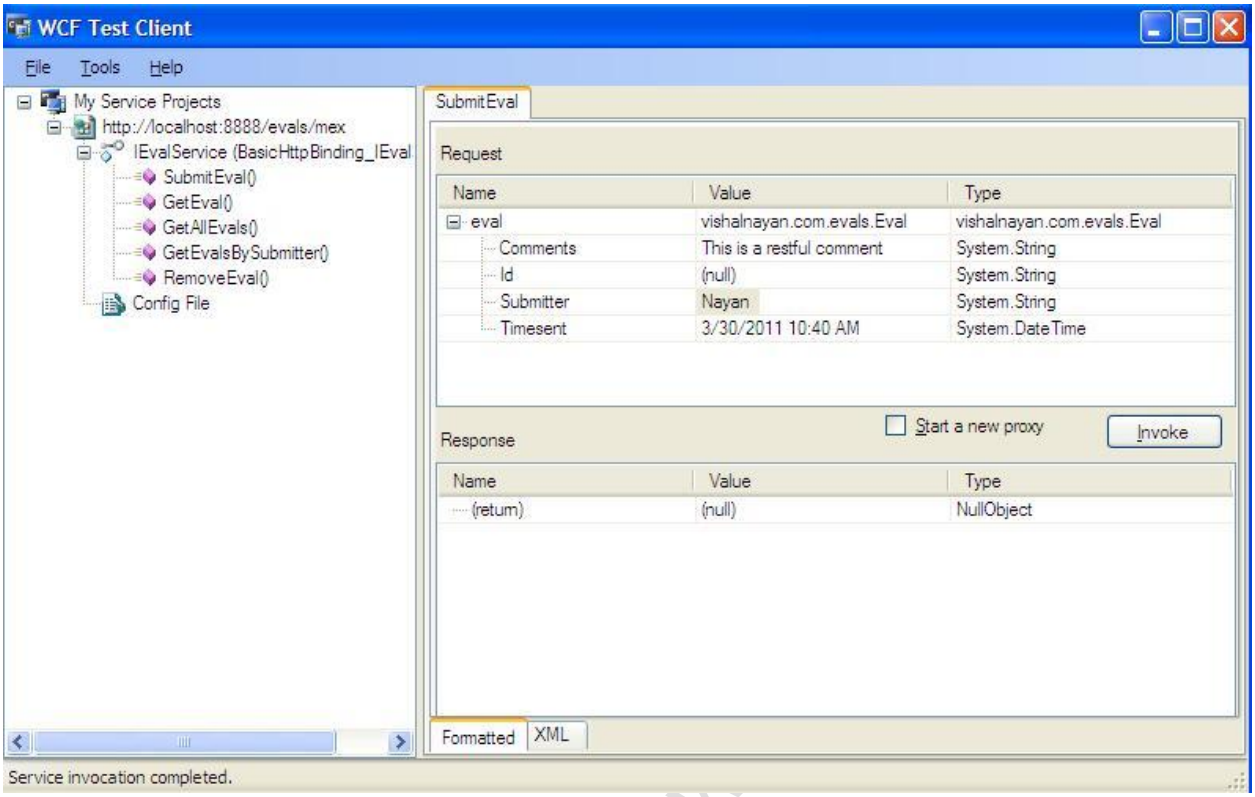


Step 14: Now we are ready to launch our service which is listening to all incoming calls . Set multiple project startup option;



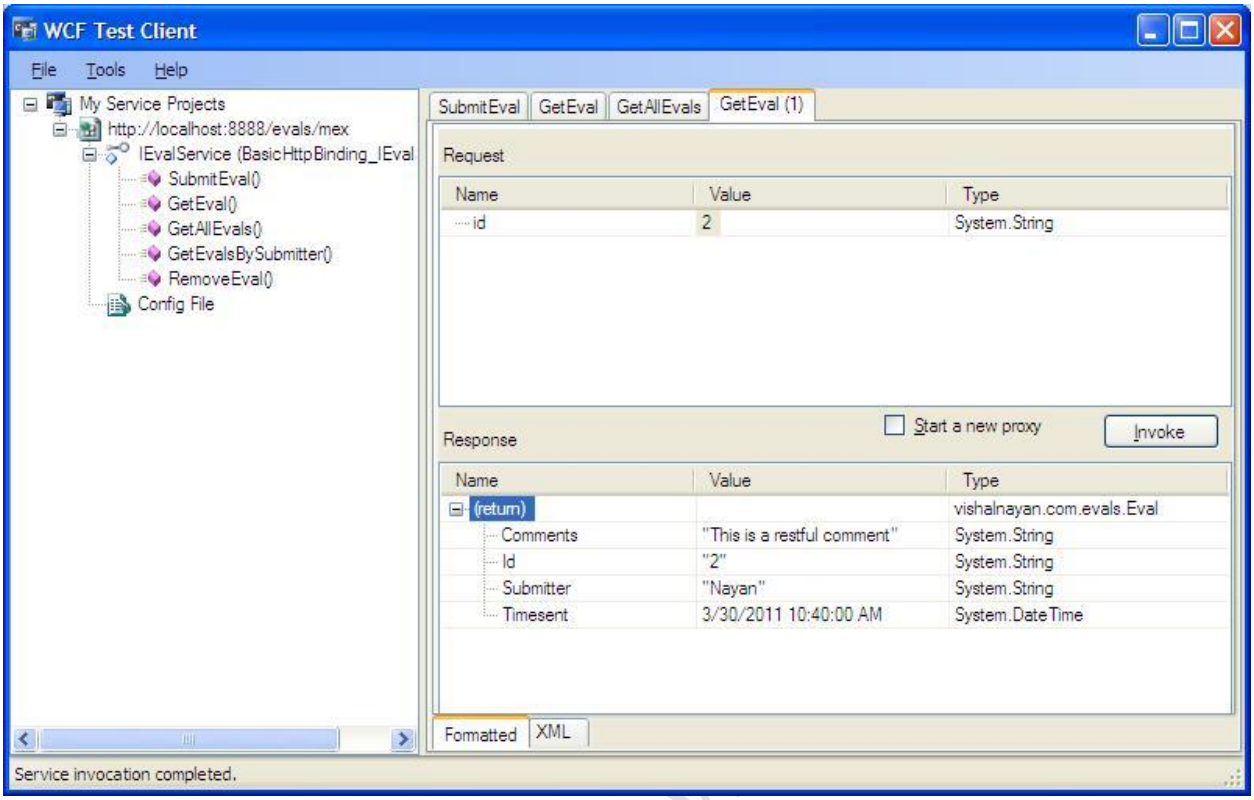
Now press F5 and this will launch our Host console screen showing detail of endpoint. Also WCF client application will be launched.





Double click SubmitEval() and write comment and submitter name. now this message is not using SOAP it is using REST.

Click to GetEval() and provide a id to see the message back.



This is same way we were doing with SOAP way using basicHttpBinding. Now we are sending message using REST , using webHttpBinding .

Step 16: Now lets check in browser and try to access any eval method .

Copy base address in URL and then followed by URI template.

i.e. base address : http://localhost:8181/evalservice

URITemplate :"evals"

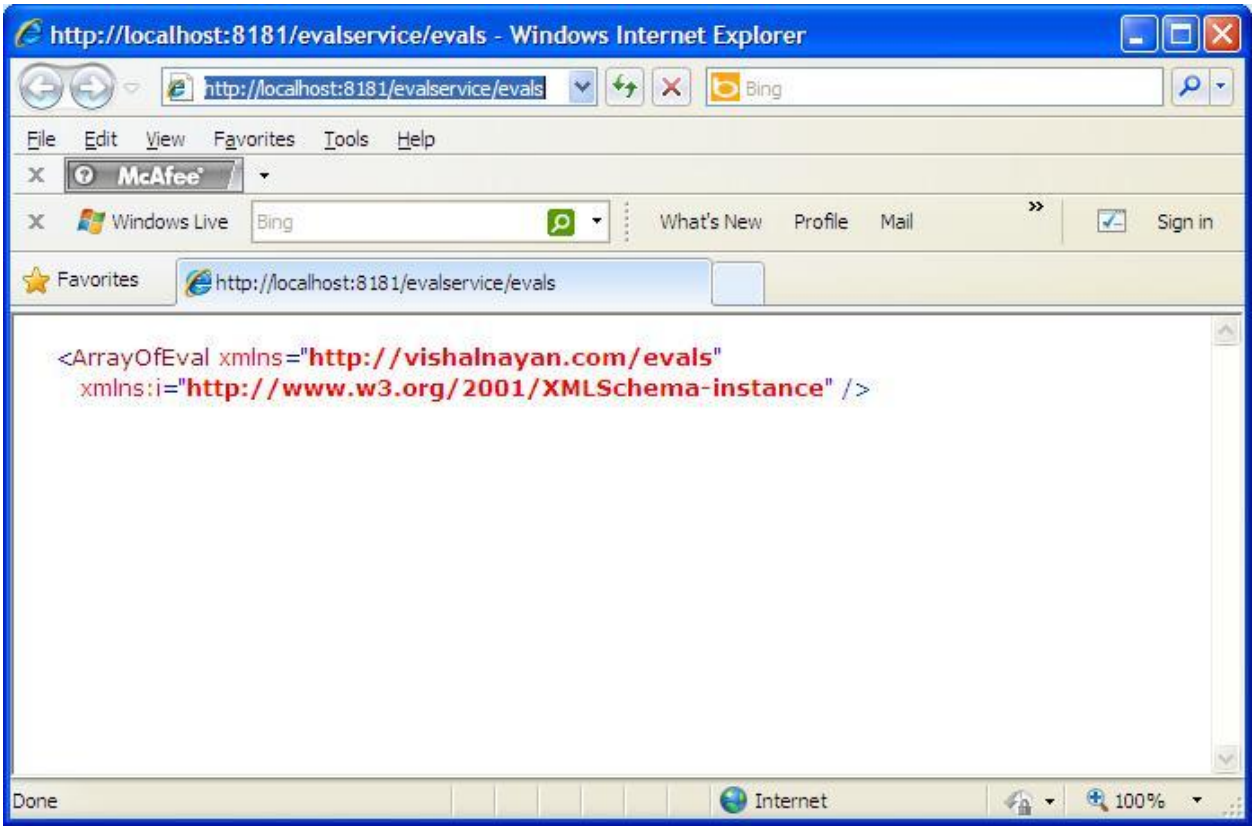
So the URL becomes like this ; http://localhost:8181/evalservice/evals

Important : UriTemplate = "evals", this actually means we are adding evals to our base address, i.e.

http://localhost:8181/evalservice/evals , and corresponding to this URI we have below method

```
[WebGet(UriTemplate = "evals")]
[OperationContract]
List<Eval> GetAllEvals();
```

So when we call ; http://localhost:8181/evalservice/evals URI , GetAllEvals() , will get called. Similarly for other URITemplates.



Step 17: Using httputil.js to call methods REST ful way.

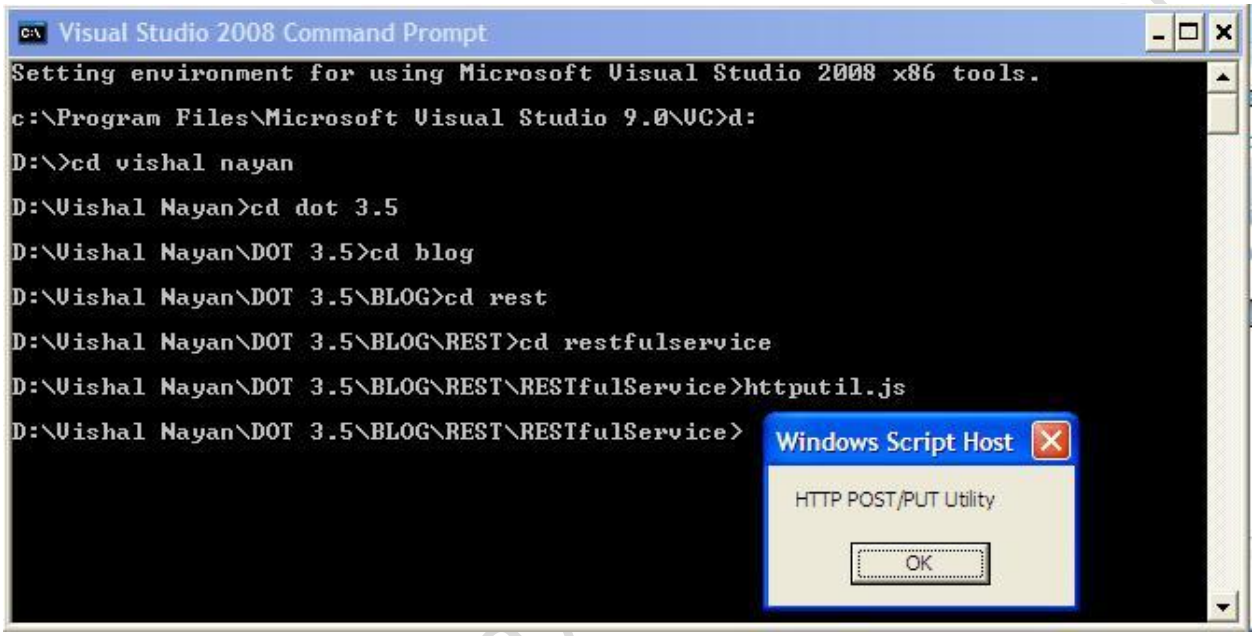
Httputil.js is a javascript utility for calling via HTTP verbs, i.e POST, DELETE etc. so we will use this utility to send XML message to REST methods. Below I have created a xml file , which compliment to our eval class variable,

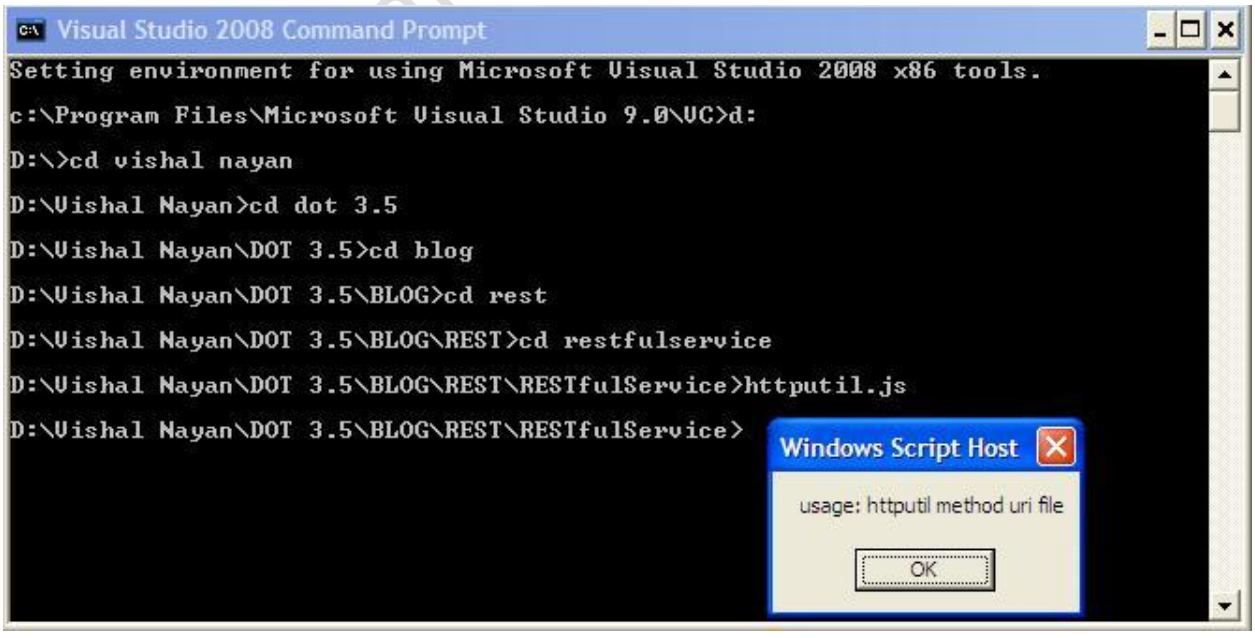
```
<Eval xmlns="http://vishalnayan.com/evals">
    <Comments>This came from Vishal!</Comments>
    <Submitter>Vishal</Submitter>
    <Timesent>2012-04-22T20:30:15.674-06:00</Timesent>
</Eval>
```

Copy httputil.js and this xml file in your solution directory and see below how to invoke REST method using this utility. Also keep the Host application up and running to listen to any incoming calls.

1) Open visual studio command prompt.

Browse to you directory where you have copied XML files and httputil.js file

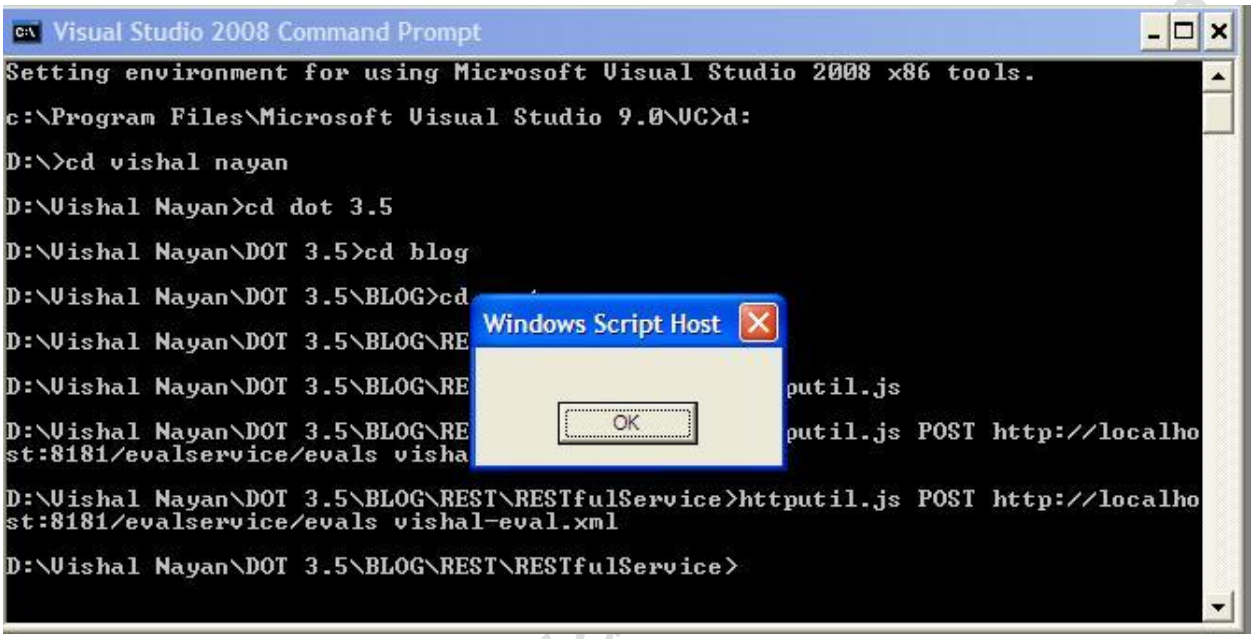




So the path for invoking is like this

httputil.js <HTTP Verb> <URI for file,i.e. xml file>

httputil.js POST http://localhost:8181/evalservice/evals vishal-eval.xml



Ok, it looks our message has been send successfully.

Note we are calling URI template evals with webinvoke POST , so URI template evals will navigate it to below method;

```
[WebInvoke(Method="POST", UriTemplate="evals")]
[OperationContract]
void SubmitEval(Eval eval);
```

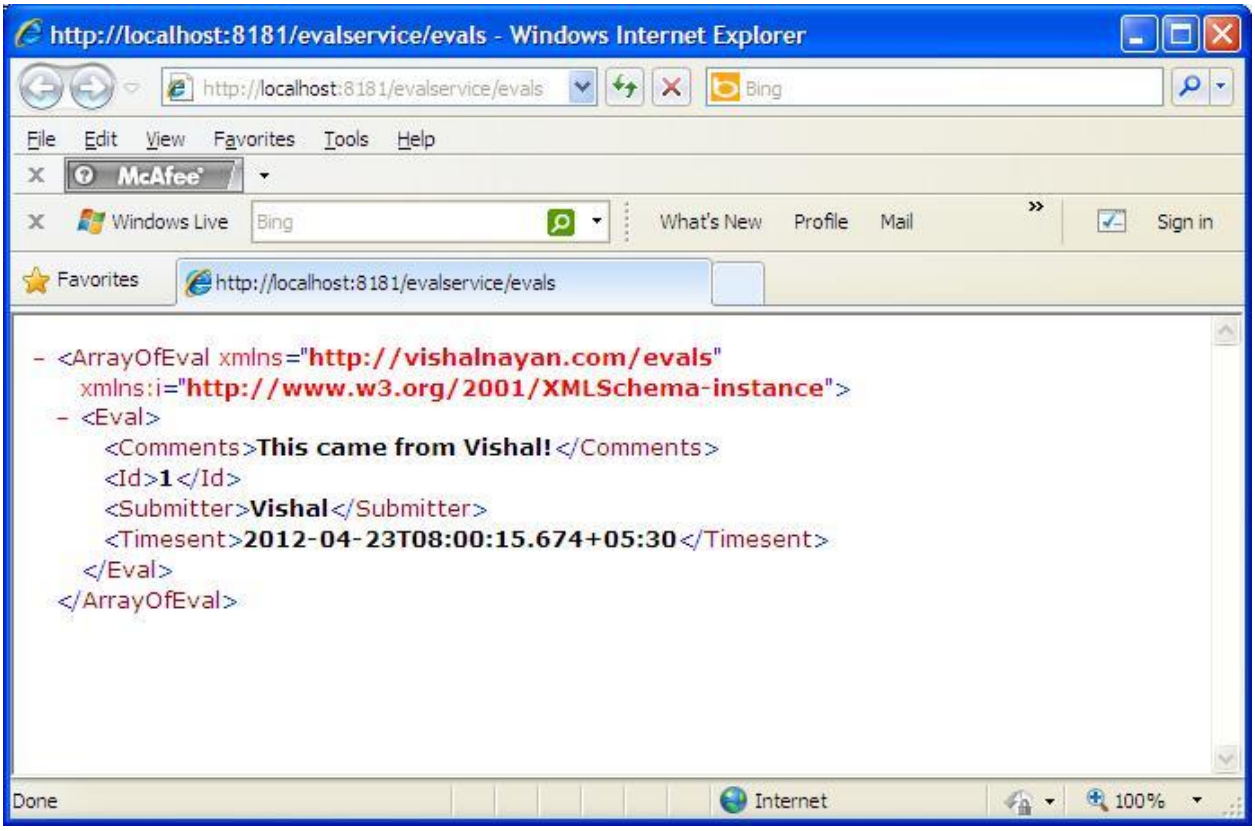
so SubmitEval() will be called.

Now let us check where eval got submitted or not.

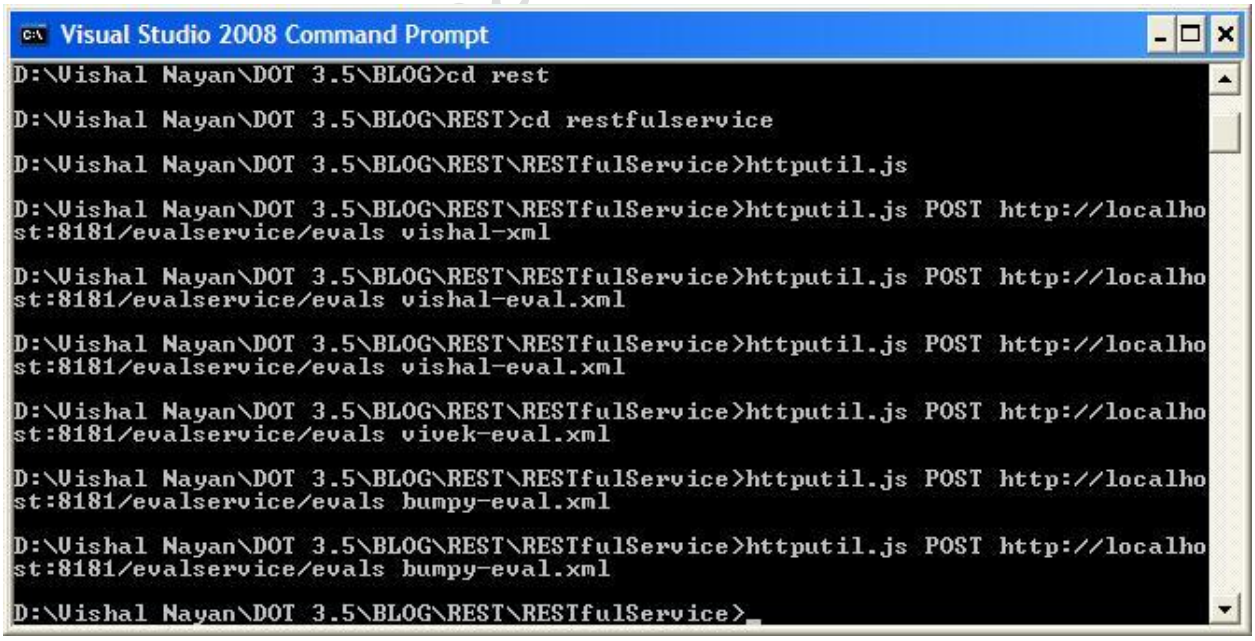
Copy this URL http://localhost:8181/evalservice/evals in browser and hit . now here

```
[WebGet(UriTemplate = "evals" /*, ResponseFormat=WebMessageFormat.Json*/)]
[OperationContract]
List<Eval> GetAllEvals();
```

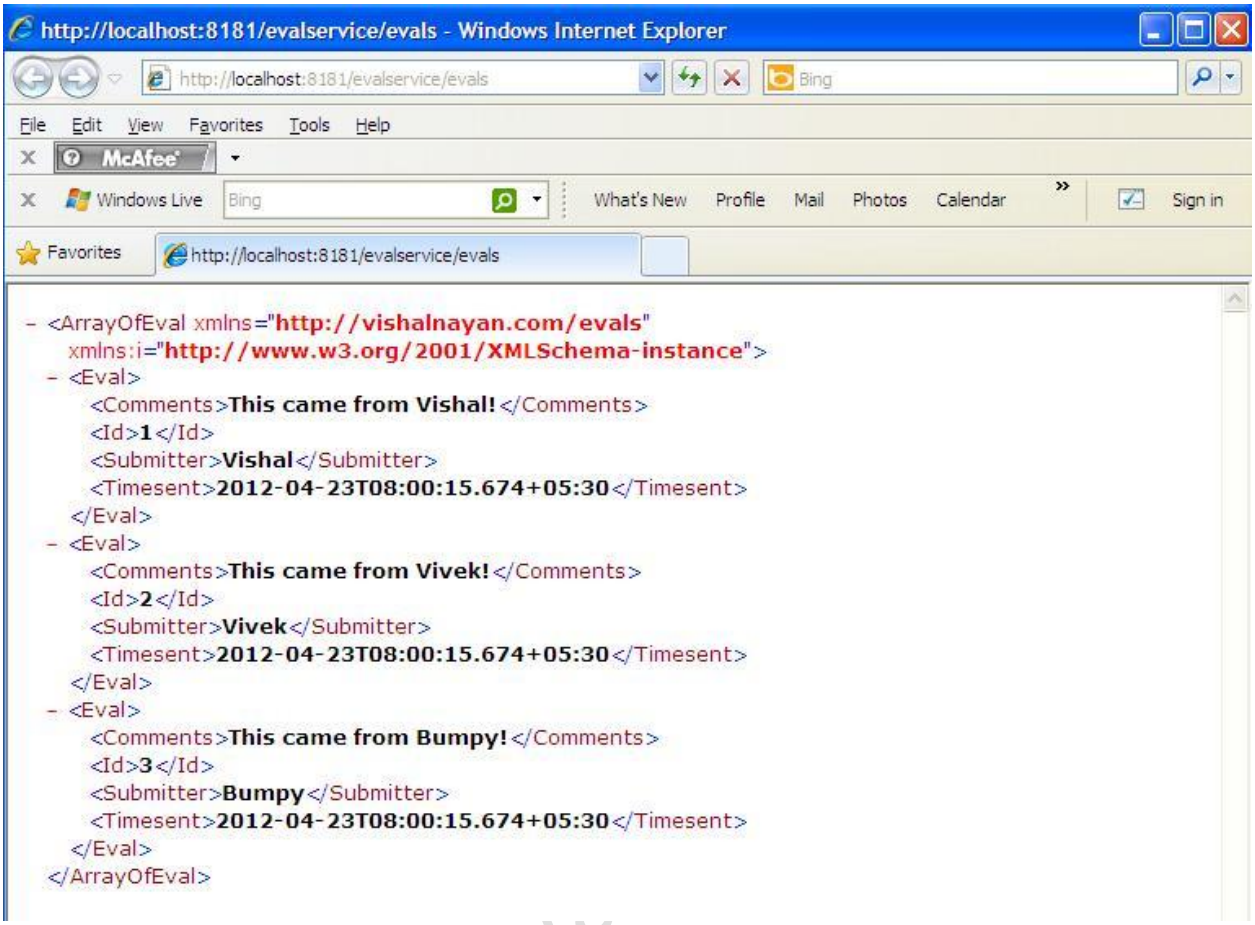
Will be called.



Cool, you can see our eval was submitted. Try sending more message like this



Refresh the browser and see all the evals submitted.



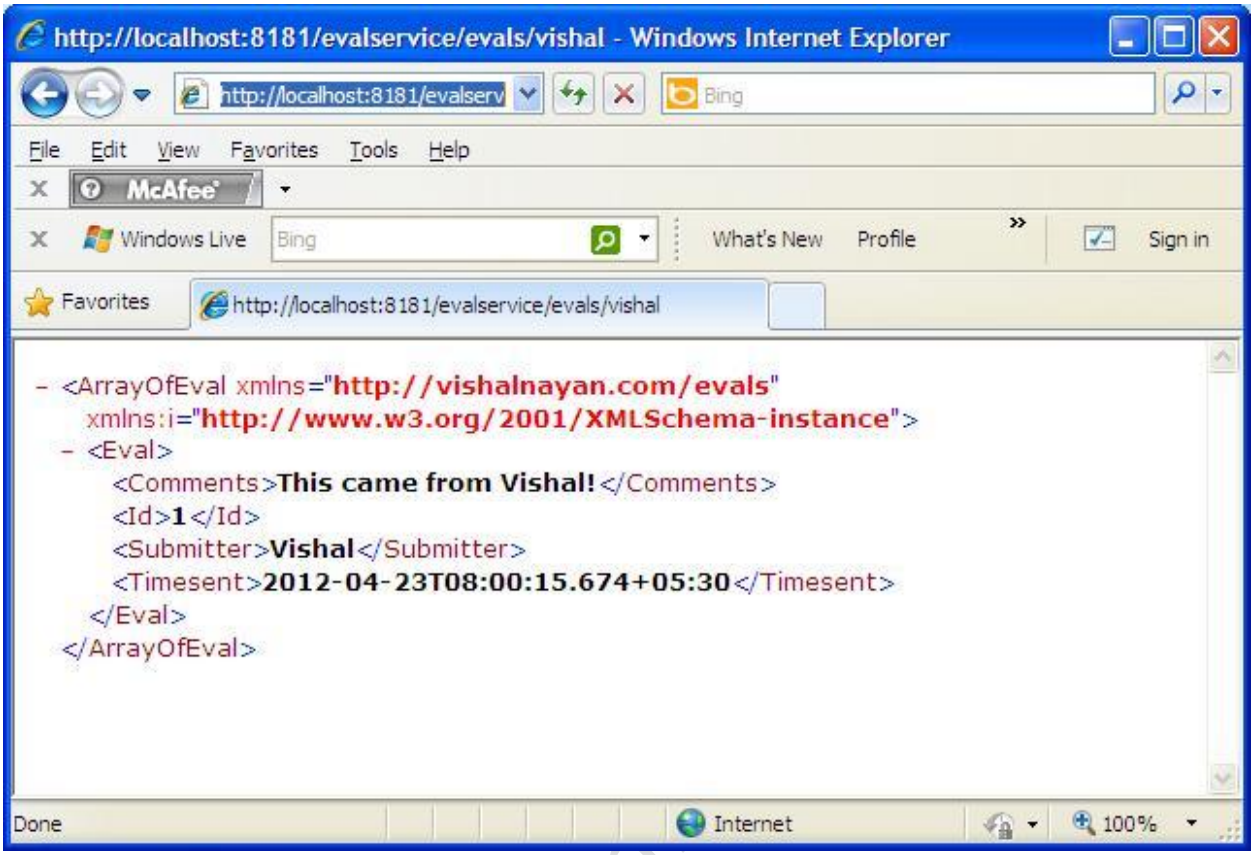
Now we are interested in knowing evals submitted by a particular submitter say vishal. For this we have call below URI template , which will point GetEvalsBySUbmitter method.

```
[WebGet(UriTemplate = "evals/{submitter}")]
        [OperationContract]
        List<Eval> GetEvalsBySubmitter(string submitter);
```

So the URI will look like this now

http://localhost:8181/evalservice/evals/vishal

hit and see the result, it will show all evals submitted by only vishal.



Now we are interested in finding the evals , by providing the id

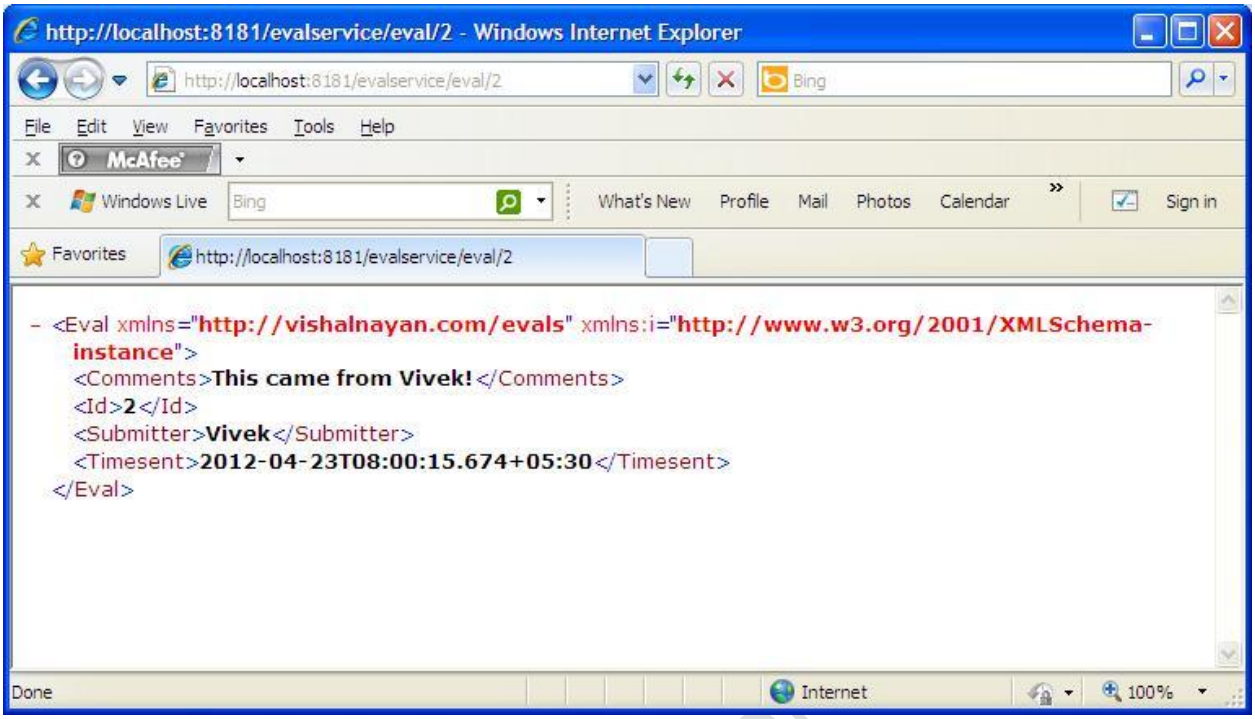
So the URI template for this is

```
[WebGet(UriTemplate = "eval/{id}")]
        [OperationContract]
        Eval GetEval(string id);
```

And URL will look like this

http://localhost:8181/evalservice/eval/1

and result would look like this;



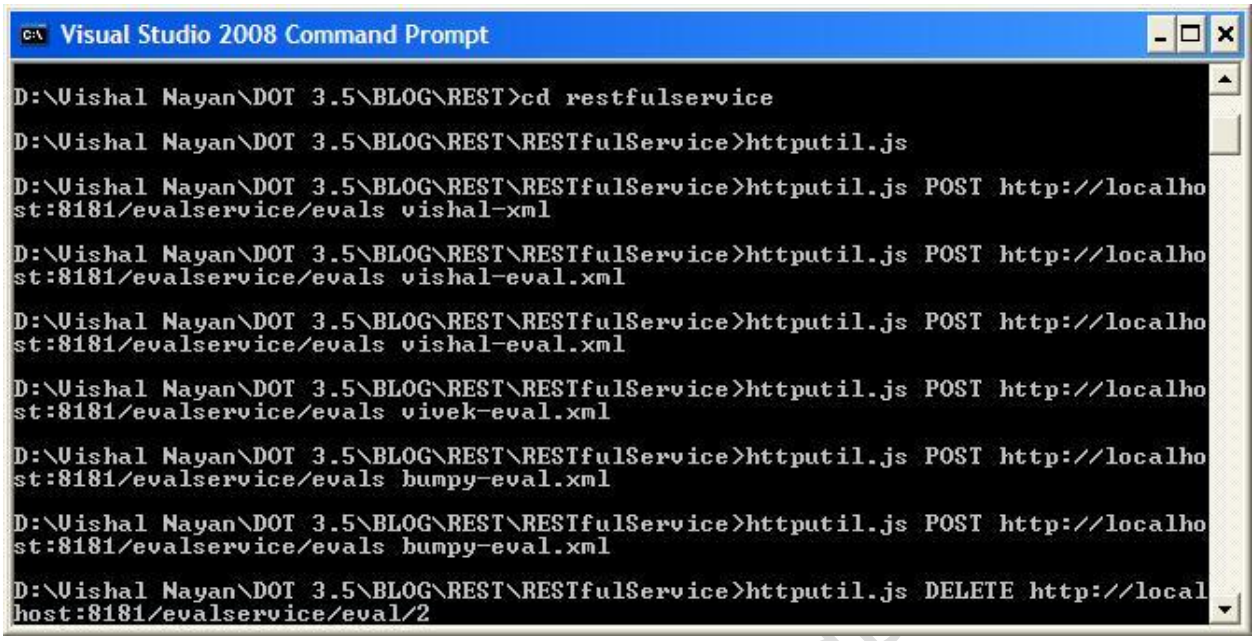
Step 18: Let us try some more HTTP Verbs for deleting any eval.

Type below path;

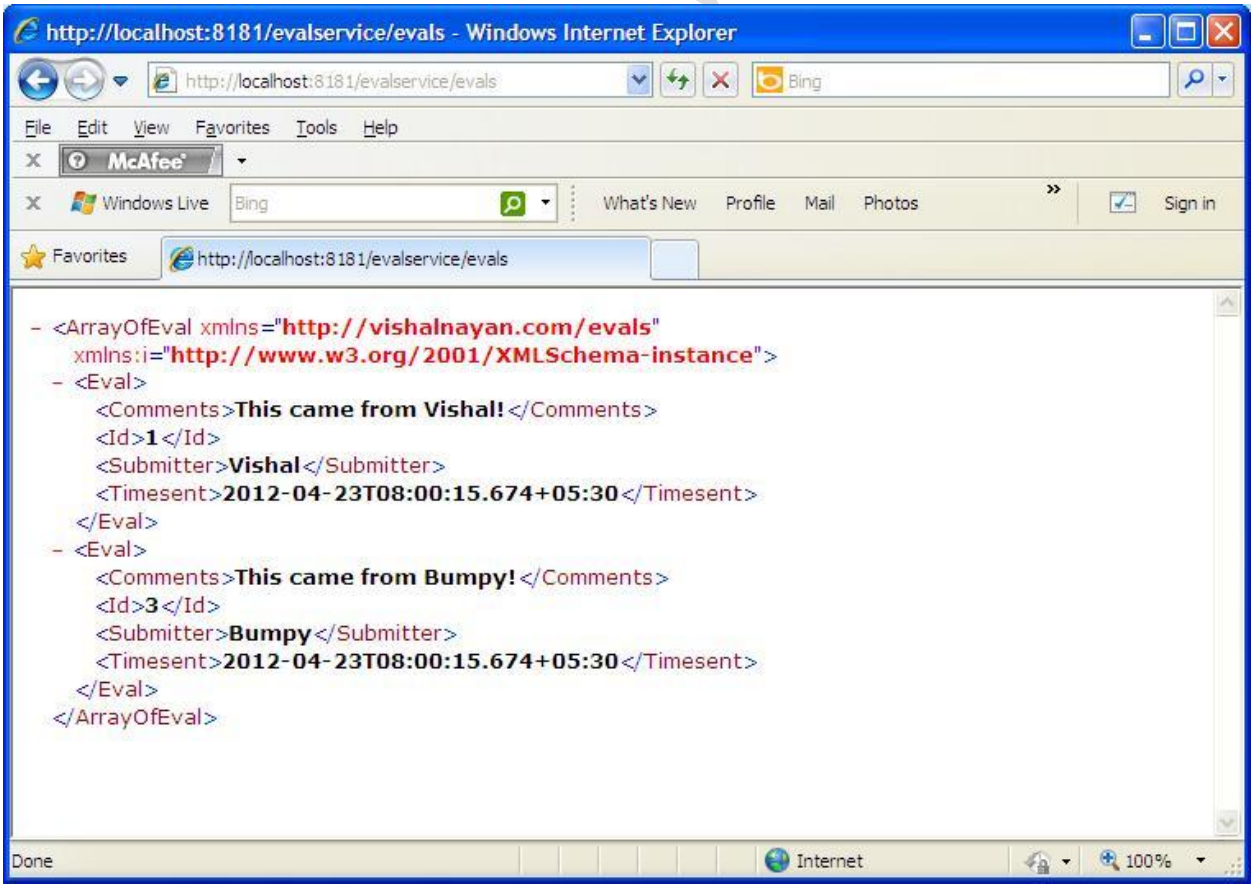
httputil.js DELETE http://localhost:8181/evalservice/eval/2

so here we are call URI to delete a eval. In this case below method will be called

```
[WebInvoke(Method = "DELETE", UriTemplate = "eval/{id}")]
        [OperationContract]
        void RemoveEval(string id);
```



Refresh the browser , you will see now that eval of id=2 , will be deleted and is not availabe.



Cool , isn't.

Step 19: Exploring WebServiceHost / Factory.

So by this we saw how to use create restful operations and how to configure app.config file to support restful configuration. We also saw how to call using WCF cleint application and Httputil.js utility.

If you notice we are hosting our service using follwing things;

1) we are using **ServiceHost** Class to host our service

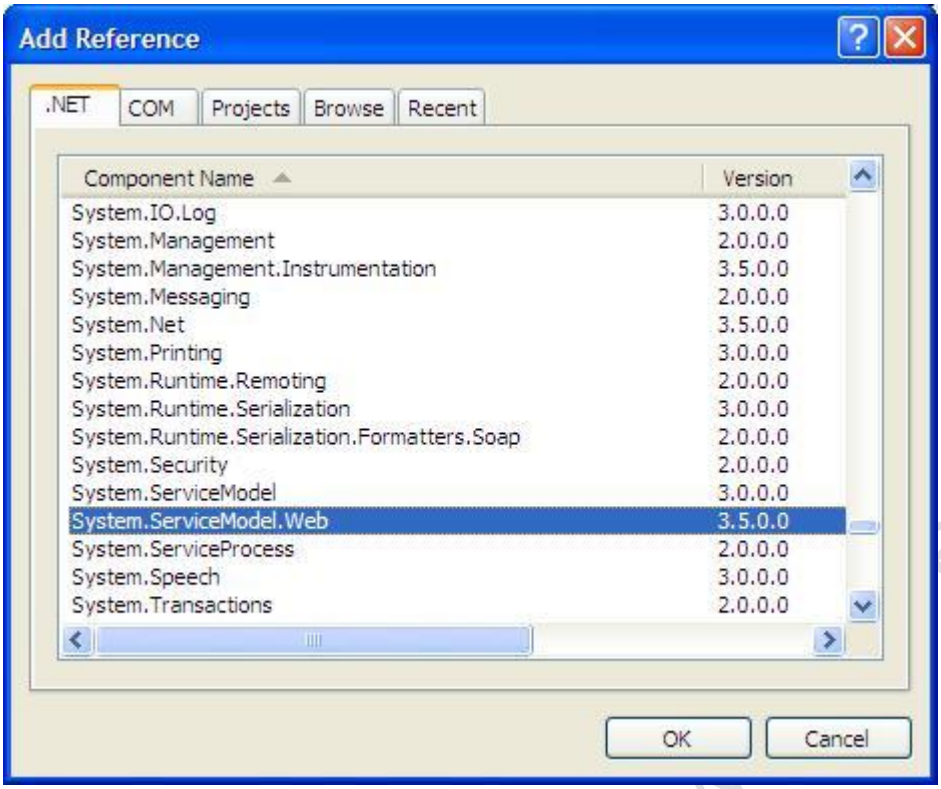
2) we have configured endpoint behavior with web http extension for REST support.

```
<endpointBehaviors>
        <behavior name="Web">
          <webHttp />
        </behavior>
      </endpointBehaviors>
```

3) we have created service behavior

```
<serviceBehaviors>
        <behavior name="Default">
          <serviceMetadata httpGetEnabled="true" />
          <serviceDebug includeExceptionDetailInFaults="false" />
        </behavior>
      </serviceBehaviors>
```

Well now we will use WebService Host class to manage all these. For this add System.ServiceModel.Web assembly in ConsolHost Project.



Open app.config file and delete all keeping base address only. So after deleting rest , app.config file will look like below;

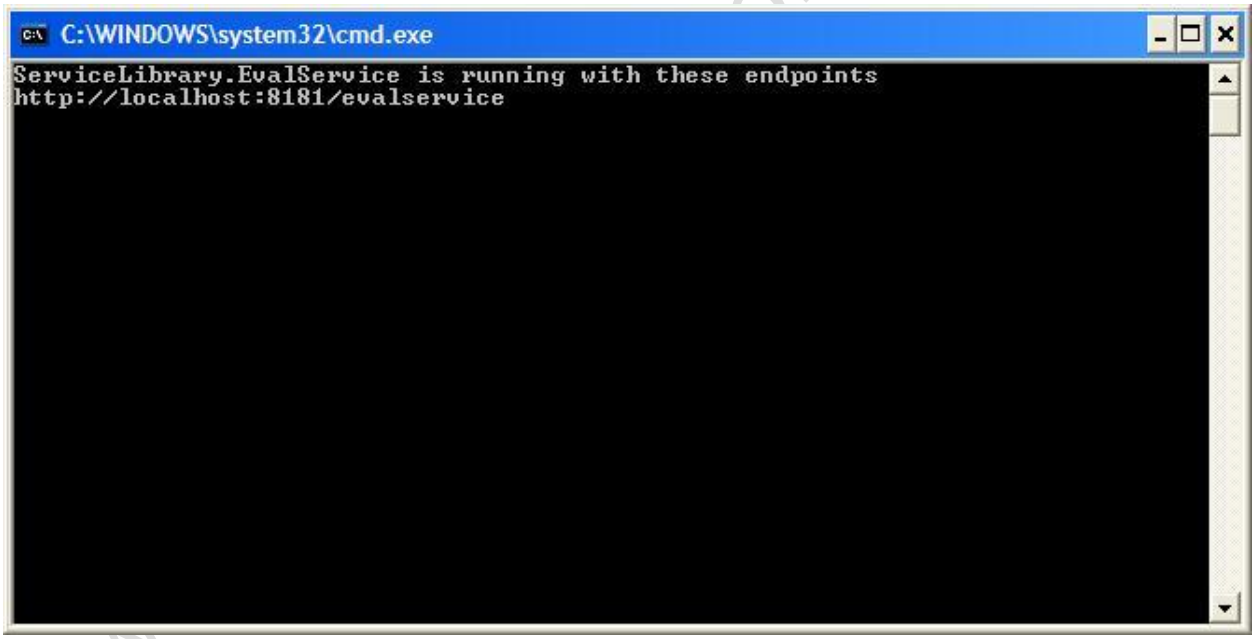
```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.web>
    <compilation debug="true" />
  </system.web>
  <!-- When deploying the service library project, the content of the config
file must be added to the host's
  app.config file. System.Configuration does not support config files for
libraries. -->
  <system.serviceModel>
    <services>
      <service name="ServiceLibrary.EvalService">
        <host>
          <baseAddresses>
            <add baseAddress="http://localhost:8181/evalservice" />
          </baseAddresses>
        </host>
      </service>
    </services>
  </system.serviceModel>
</configuration>
```

Open Program.cs and change the ServiceHost to WebServiceHost class. So now the program.cs look like below

```
static void Main(string[] args)
        {
            WebServiceHost host = new WebServiceHost(typeof(EvalService));
            try
            {
                host.Open();
                PrintServiceInfo(host);
                Console.ReadLine();
                host.Close();

            }
            catch(Exception e)
            {
                Console.WriteLine(e);
                host.Abort();

            }
        }
```

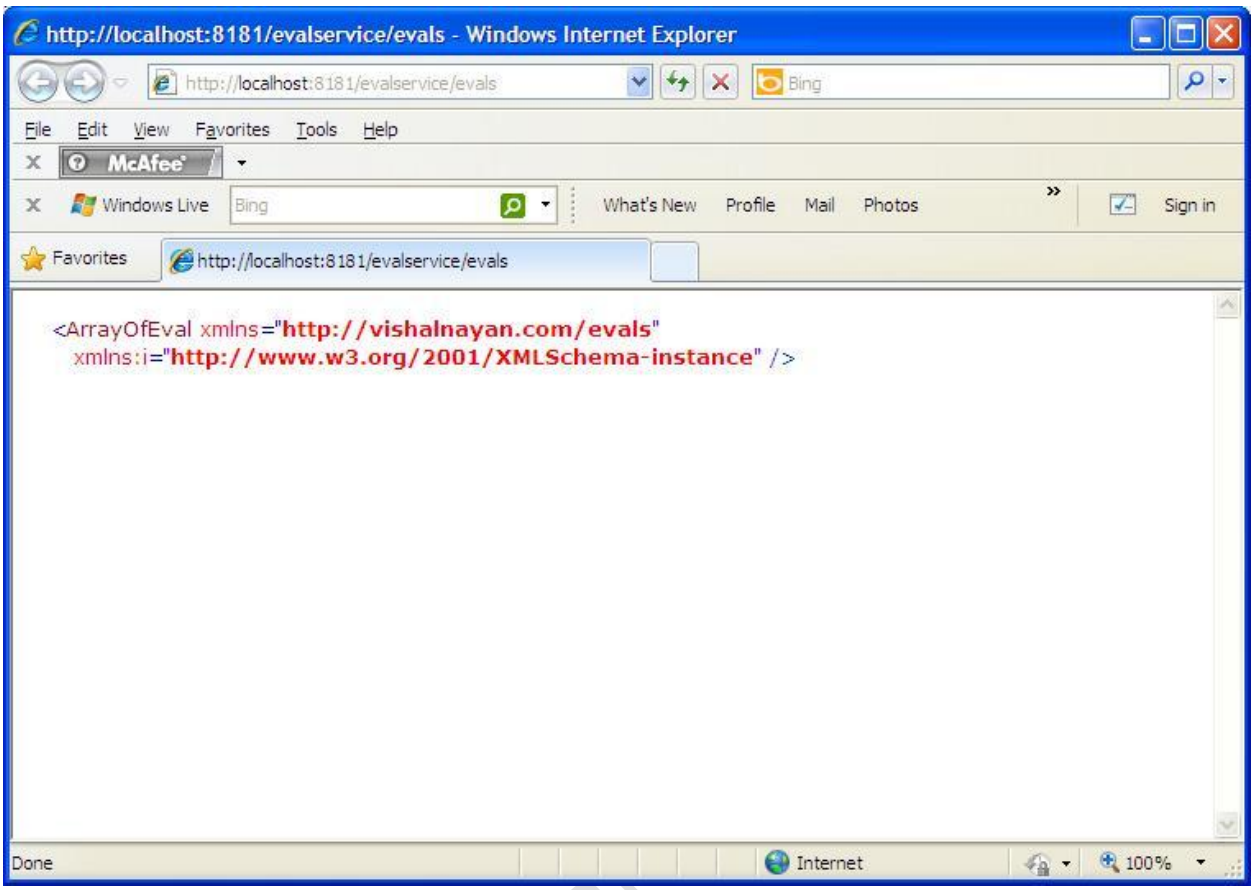
That's all , with the help of WebServiceHost , rest of app configuration settings are not required.

Now press F5 and run the host applicaion.



All is perfect. Lets check in the browser

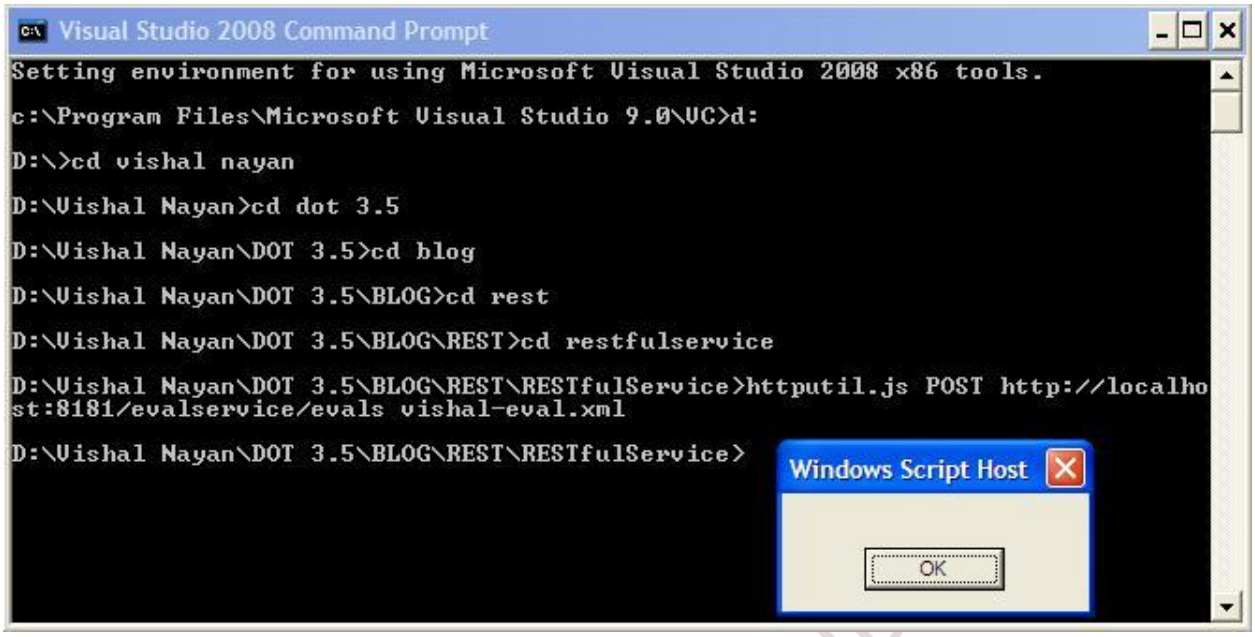
http://localhost:8181/evalservice/evals



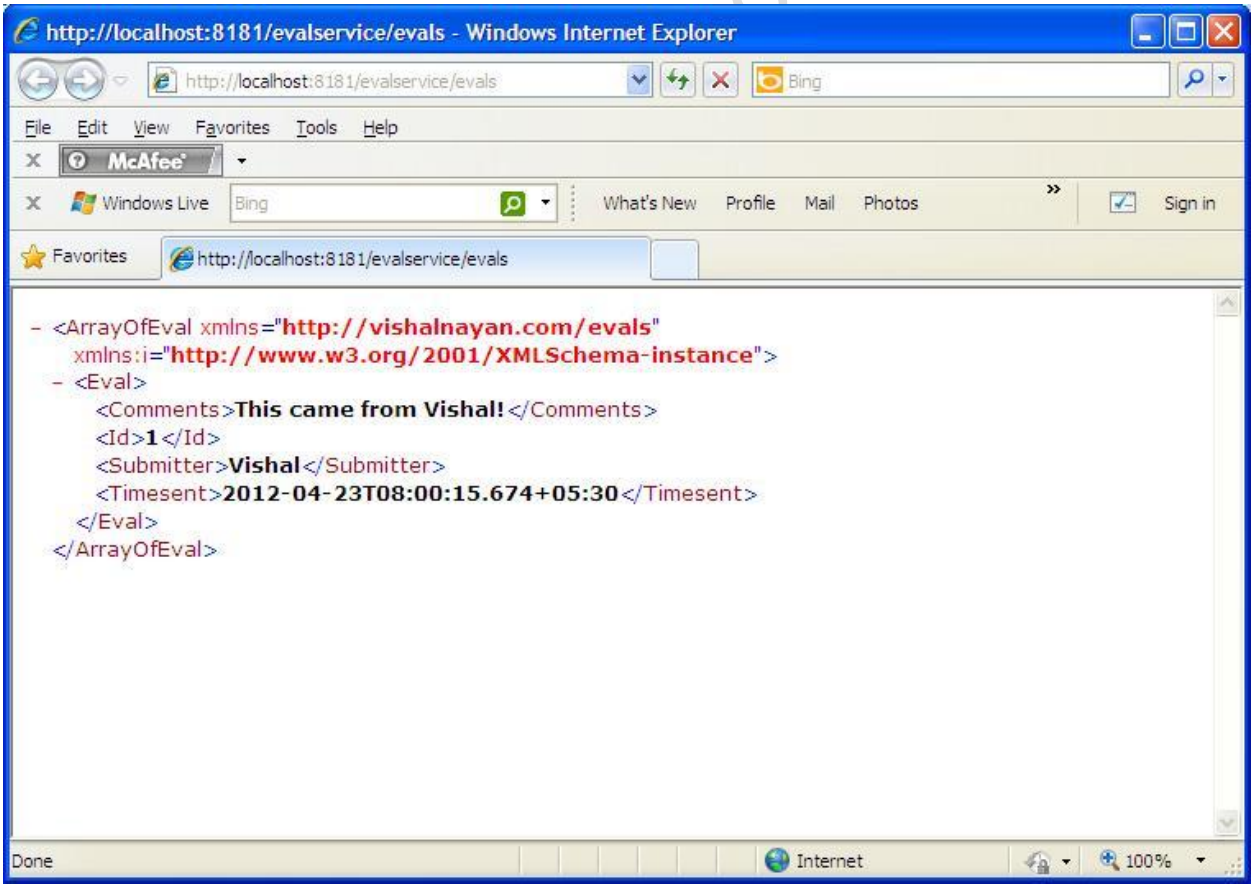
This too seems to be perfect. Now lets go back to httputil.js again and POST the xml message which have.

So type this path in visual studio command prompt .

Httputil.js POST http://localhost:8181/evalservice/evals vishal-eval.xml



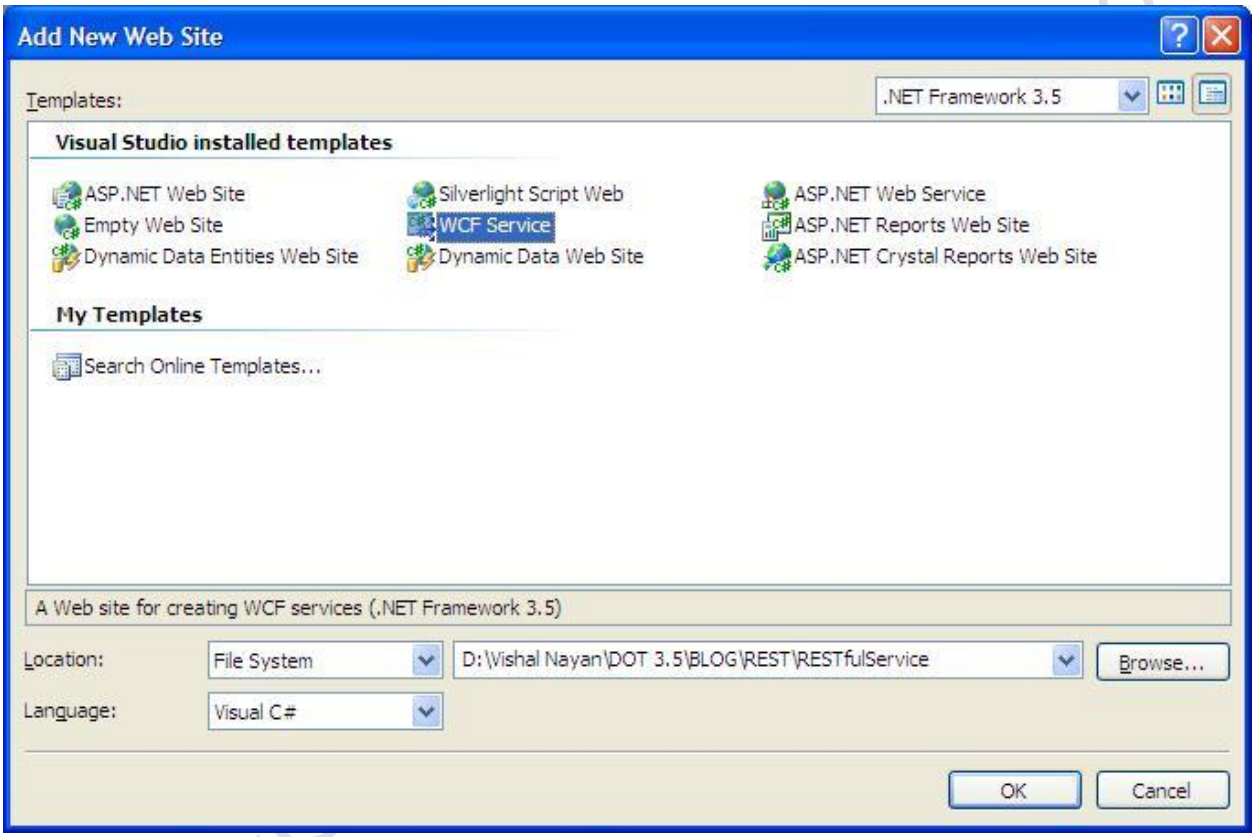
Refresh the browser



Cools, seems after we applied WebServiceHost class, all is working like before. We need not to do anything extra to call our method,webservicehost takes care of everything. This also handy when we host our service with IIS.

Step 20: Hosting our service with IIS and working with WebServiceHost

1) Add a new WCF website in RestfulService solution. Name is EvalSite



2. Delete all the files udner App_Code

3. rename service.svc to eval.svc

4. open web.config and delete entire system.servicemodel setting. So we don't have any WCF configuration setting what so ever.

5. Add a reference to this site to ServiceLibrary assembly.

6. Now open eval.svc and you will have something like below;

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

```
Service="Service" CodeBehind="~/App_Code/Service.cs" %>
```

Delete few attributes and change it to something like below;

```
<%@ ServiceHost Service="ServiceLibrary"
Factory="System.ServiceModel.Activation.WebServiceHostFactory" %>
```

Its going to create a web service host object for us behind the scene when this svc file is activated, and that web service host object will add endpoints for us automatically , at the base address for this service, and that base address will be address of this svc file.

http://localhost:3688/EvalSite/eval.svc

And then automatically add webhttpbinding endpoint and configure with webhttpbehavior. So now we have a configure free mechanism to host REST full service with WCF with no WCF configuration settings.

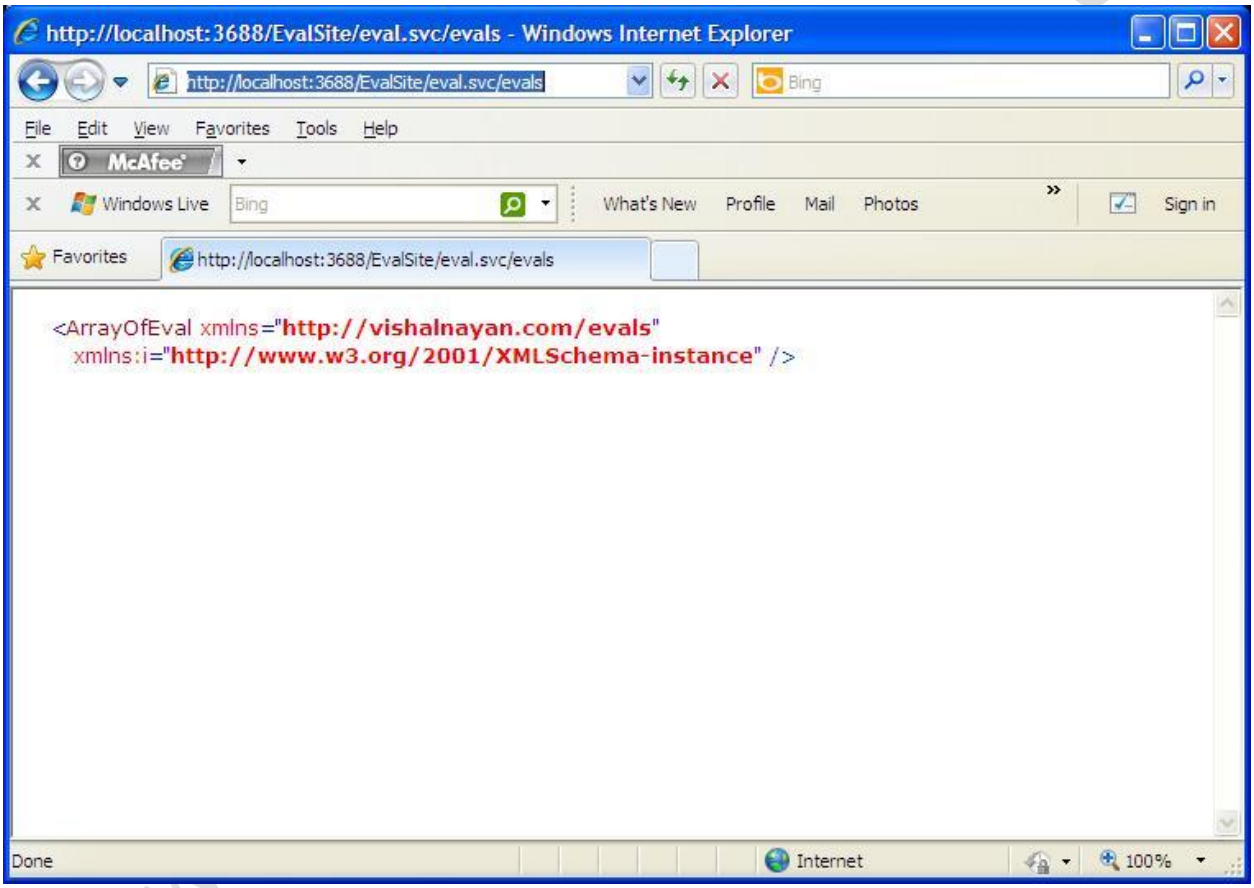
7. So we are ready now, so right click eval.svc and click “View in Browse”

So see there is no service. This is because we haven't pointed our URI to any REST method.

So type this in browser

http://localhost:3688/EvalSite/eval.svc/evals

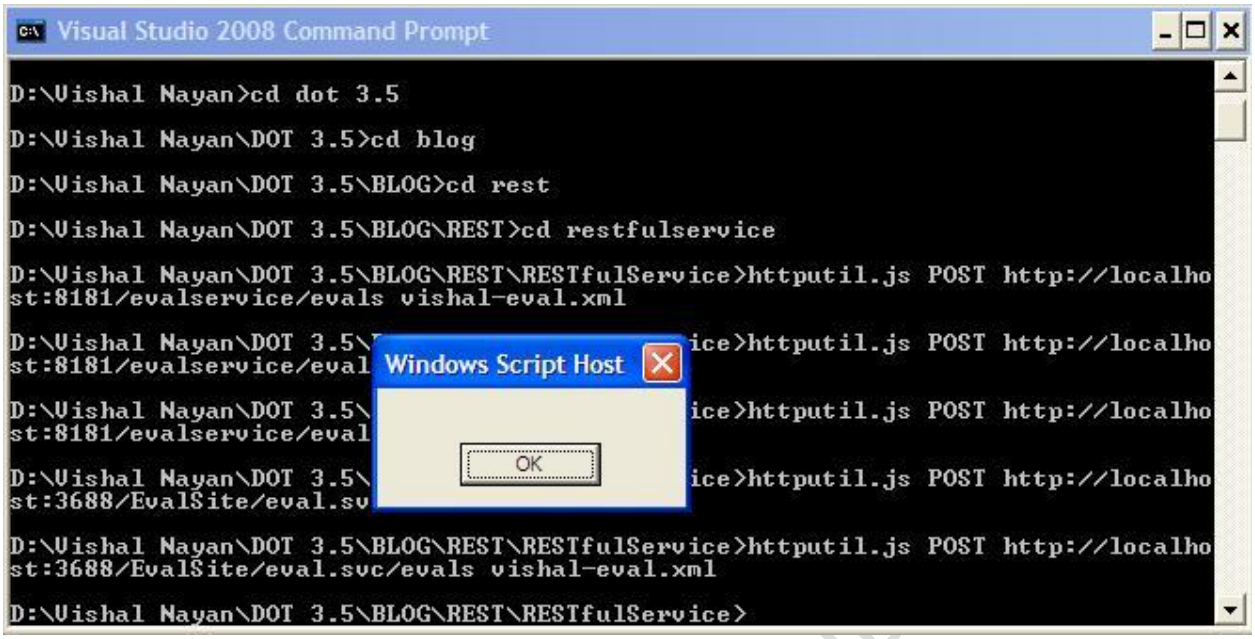
http://localhost:3688/EvalSite/eval.svc is our base address, and /evals is relative address which is URI template pointing to `GetAllEvals` () method. So here in this WCF site example, our base address got changed but relative address remains same.



Notice we got empty array of eval, so our service is working fine. Now we will use again httputil.js utility to send message,

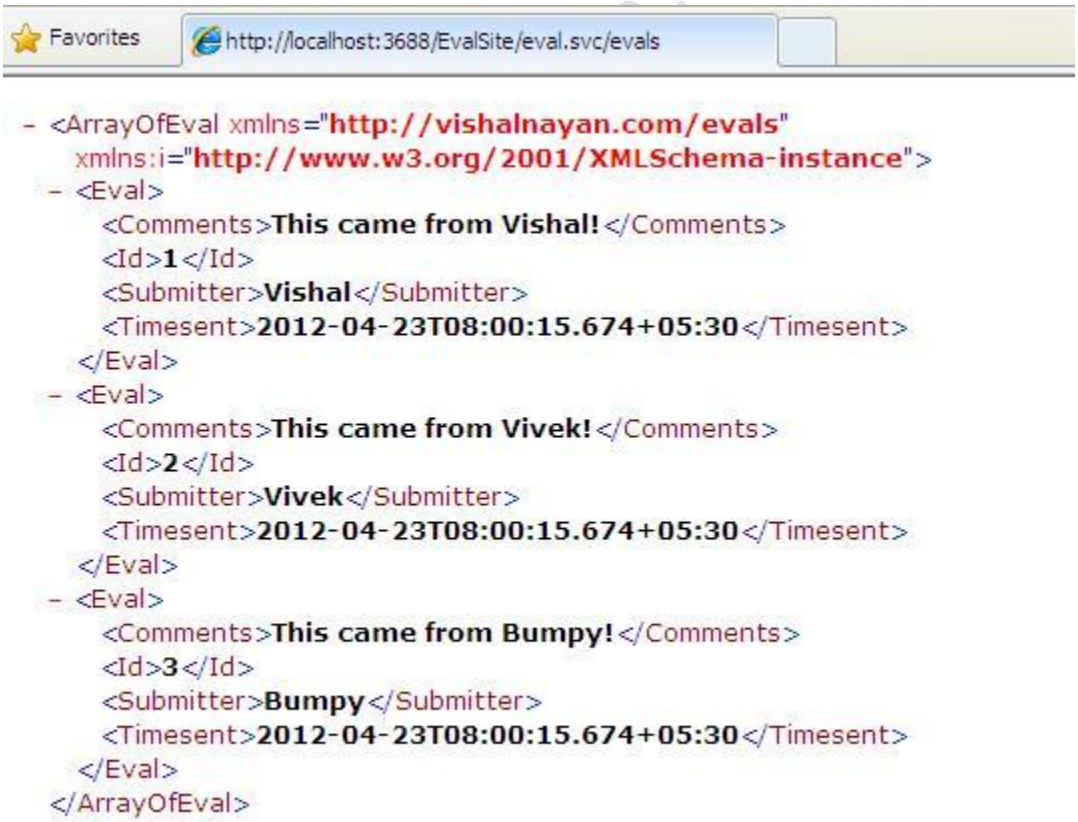
So open the command prompt and type below URL, this is changed to WCF

httputil.js POST http://localhost:3688/EvalSite/eval.svc/evals vivek-eval.xml



So we will do it for rest of xml files as well.

Now refresh the browser and see all evals or by evals id or by eval name.



Cool, we got to see all evals through this new URL http://localhost:3688/EvalSite/eval.svc/evals, which IIS is hosting for us. And benefit is that we need not to do any configuration al all, we mapped our service to svc file and told to use **`WebServiceHostFactory,`** which behind the scene use webservicehost class which do auto configurations.

Step 21: Creating a Client project for Calling REST service programitically

Till here we saw how can we host and call REST services in IIS and calling via wed browser. But in most cases we will write some client code programitically to actually calls to REST service. So let us create a cleint consol project.

For this we need service contract annotated with webget and webinvoke for the client to use similar way we have at service side. We can use same contract what we used at service side at the cleint side as well. So we will use **IEvalService** servuice contract on the cleint and then use this in conjunction with **special channel factory that know how to translate these methods calls into http request., which is** `WebChannelFactory.`

So we will add refrecence directly to ServiceLibrary to cleint console project. We will not use add service refrence because it uses SOAP API.

1. Add a new consol client project to RESTfulService Solution.

2. Add ServiceLibrary assembly to this project directly. Don't use add service reference.

3. Add System.ServiceModel.Web assembly.

3. Open program.cs. write below code

```
Console.WriteLine("**** Calling WCF service RESTway***");
WebChannelFactory<ServiceLibrary.IEvalService> cf =
    new WebChannelFactory<ServiceLibrary.IEvalService>(
        new Uri("http://localhost:8181/evalservice"));

ServiceLibrary.IEvalService client = cf.CreateChannel();
```

`WebChannelFactory` is like service host which automatically configures underlying runtime with webhttpbinding and webhttpbehavior doing job for us.

So the whole code looks like this;

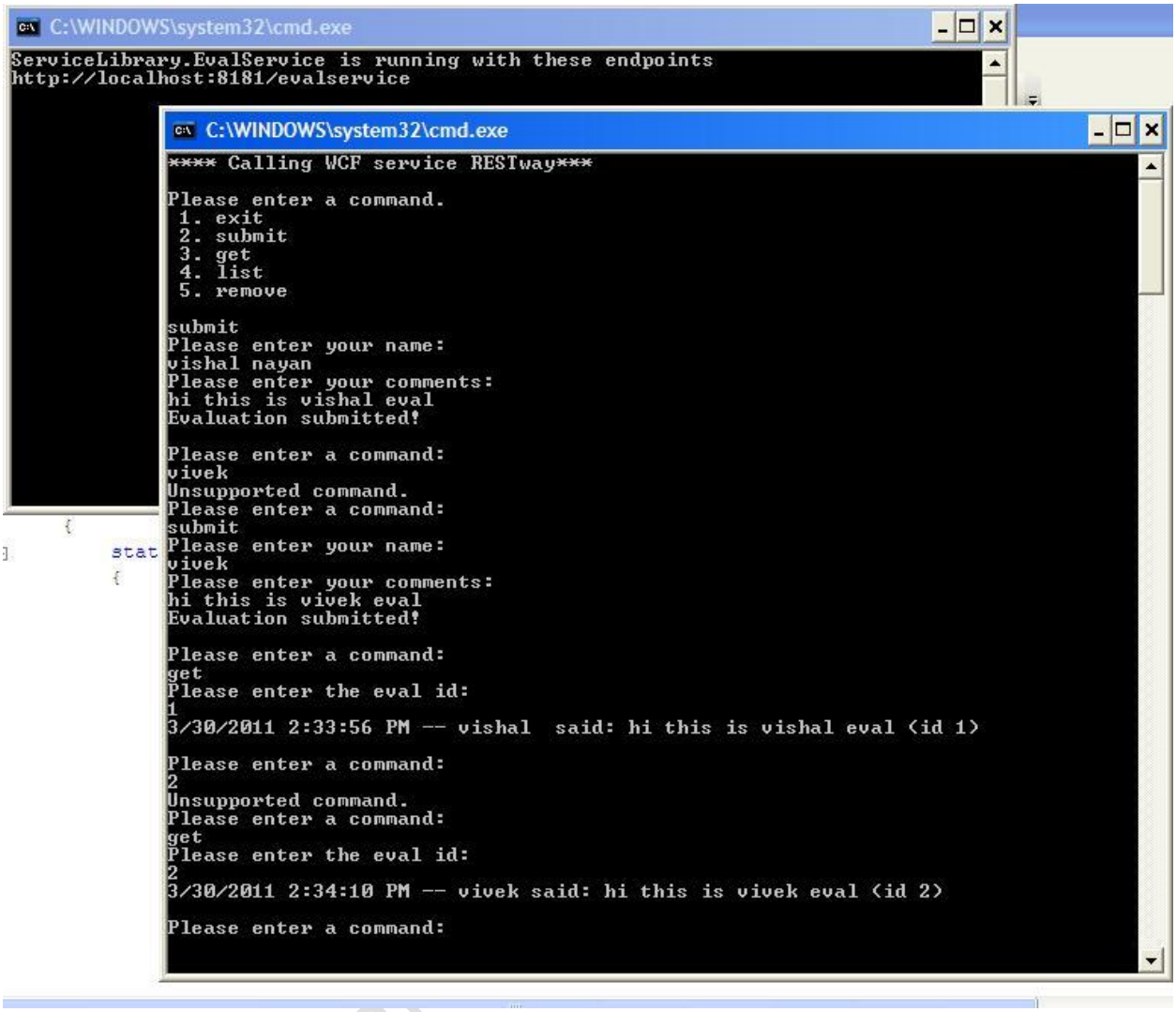
```
static void Main(string[] args)
        {
```

```
            Console.WriteLine("**** Calling WCF service RESTway***");
            WebChannelFactory<ServiceLibrary.IEvalService> cf =
                new WebChannelFactory<ServiceLibrary.IEvalService>(
                    new Uri("http://localhost:8181/evalservice"));

            ServiceLibrary.IEvalService client = cf.CreateChannel();

            Console.WriteLine("\nPlease enter a command.\n 1. exit \n 2.
submit \n 3. get \n 4. list \n 5. remove \n");
            string command = Console.ReadLine();

            while (!command.Equals("exit"))
            {
                switch (command)
                {
                    case "submit":

                        Console.WriteLine("Please enter your name:");
                        string name = Console.ReadLine();
                        Console.WriteLine("Please enter your comments:");
                        string comments = Console.ReadLine();

                        Eval eval = new Eval();
                        eval.Timesent = DateTime.Now;
                        eval.Submitter = name;
                        eval.Comments = comments;
                        client.SubmitEval(eval);

                        Console.WriteLine("Evaluation submitted!\n");
                        break;

                    case "get":
                        Console.WriteLine("Please enter the eval id:");
                        string id = Console.ReadLine();

                        Eval fe = client.GetEval(id);
                        Console.WriteLine("{0} -- {1} said: {2} (id {3})\n",
fe.Timesent, fe.Submitter, fe.Comments, fe.Id);
                        break;

                    case "list":

                        Console.WriteLine("Please enter the submitter
name:");
                        name = Console.ReadLine();

                        List<Eval> evals = client.GetEvalsBySubmitter(name);

                        evals.ForEach(e => Console.WriteLine("{0} -- {1}
said: {2} (id {3})", e.Timesent, e.Submitter, e.Comments, e.Id));
                        Console.WriteLine();
                        break;

                    case "remove":

                        Console.WriteLine("Please enter the eval id:");
                        id = Console.ReadLine();
```

```
                    client.RemoveEval(id);

                    Console.WriteLine("Evaluation {0} removed!\n", id);
                    break;

                default:
                    Console.WriteLine("Unsupported command.");
                    break;
            }

            Console.WriteLine("Please enter a command: ");
            command = Console.ReadLine();
        }
    }
```

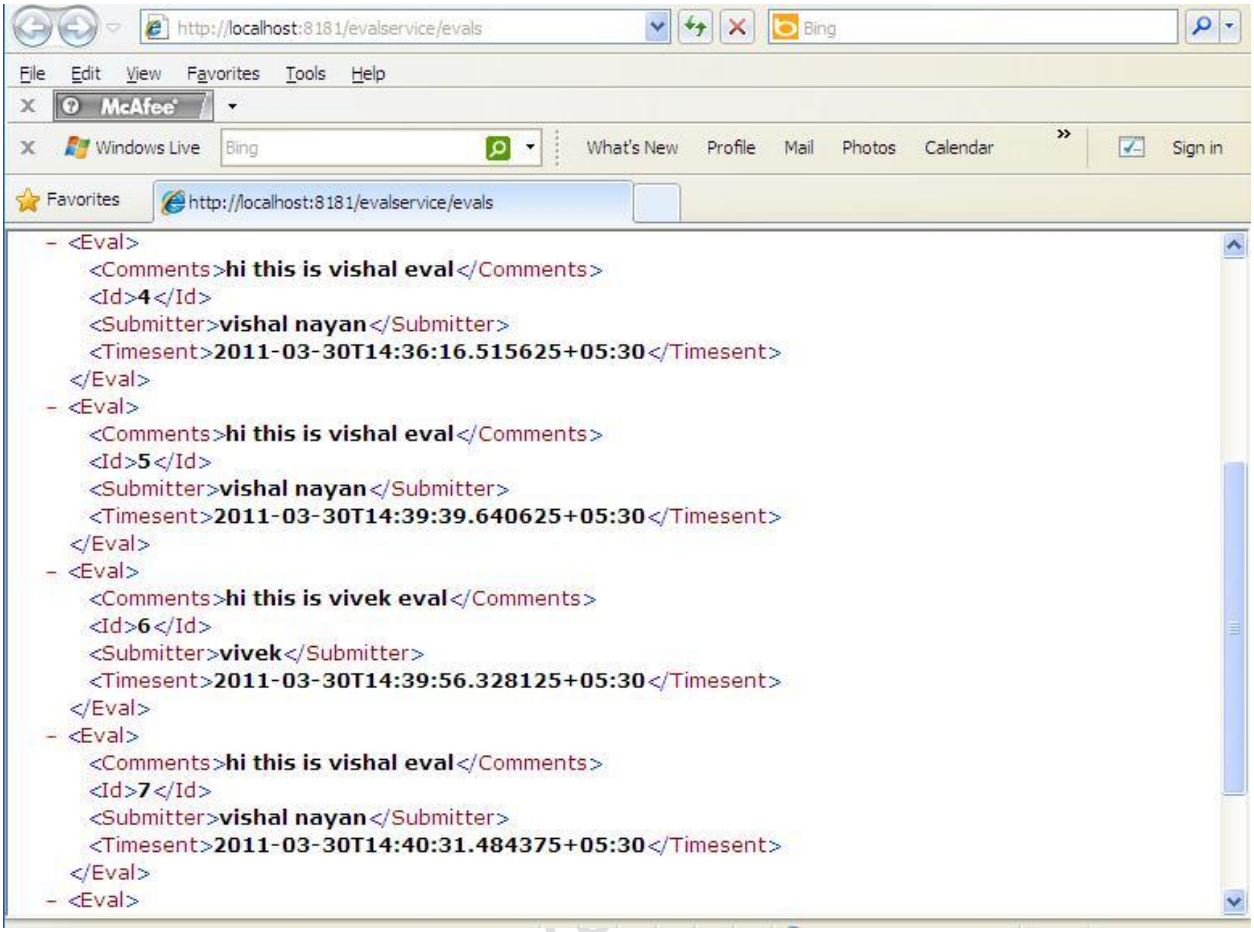
Code is self explainatory as we are asking for command and then calling particular method.

So now run Host program and then run client console application.



Cool, its working fine.

Lets check in the browser as well



Hope you enjoyed this RESTful ride and now can create your own REST ful services.

Read more here about REST on my blog

CHEERS