Creating a service search module in TSV2 - A Step By Step Guide

**Revision History**

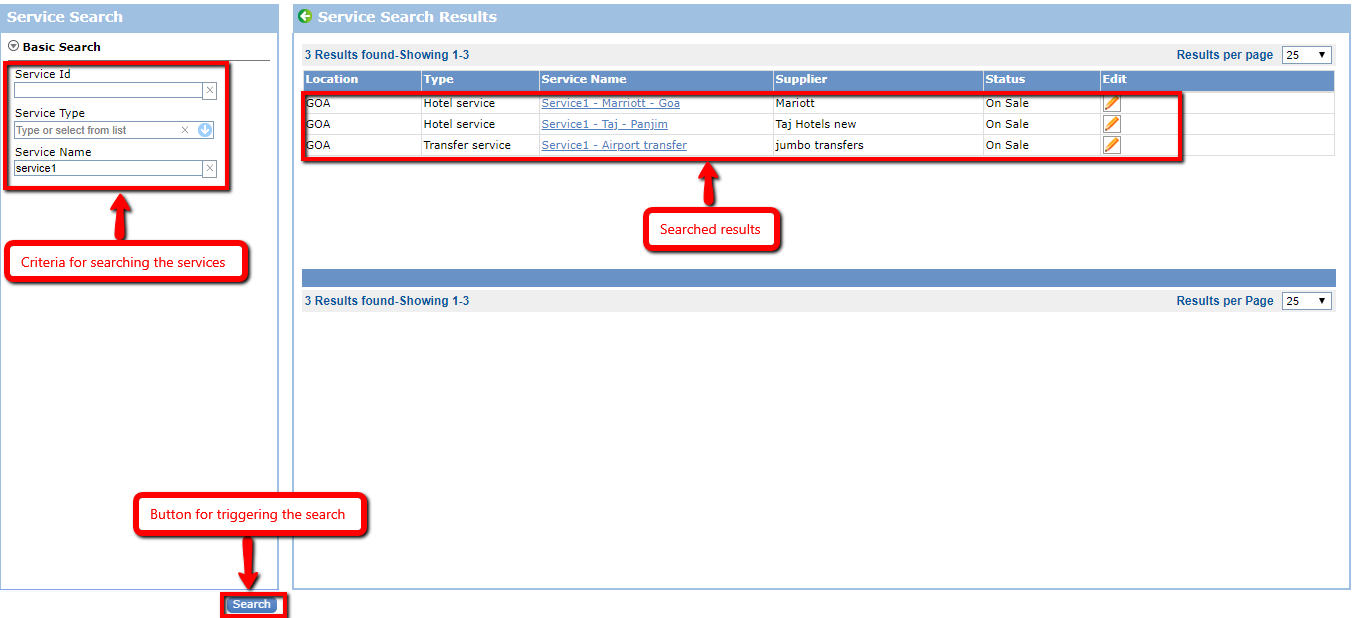
|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Date** | **Author** | **Comments** |
| 1.0 | 04 Apr 2018 | Ritesh Fulari | Initial Creation |
|  |  |  |  |

**Prerequisites**

1. Travel Studio Installed and Working.\*
2. Travel Studio Database.\*
3. You have to make sure that Travel Studio, Travel Studio Database, and the URL are of same TS version.

**OVERVIEW OF THE EXERCISE**

Basically you have to design a screen for searching the services based on desired criteria and display it in the grid as shown below.



Once the results are shown, you should be able to edit service name and status of the service by clicking on the edit link provided against the service details in the grid.

**NOTE:**

*1] You can make use of the existing web service. But you have to create your own web method.*

*2] You have to create your own data contracts, service contracts, business entities and stored procedure(optional).*

**LET’S BEGIN WITH THE DEVELOPMENT**

We will begin by writing API side code first.

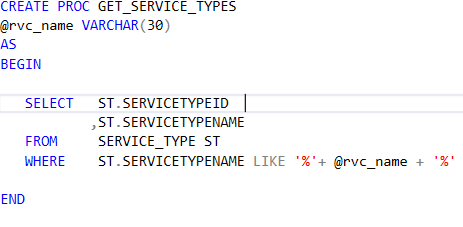
We have to write two web methods for fetching the required information from the database.

1] Web method to fetch types of services for auto complete functionality.

2] Web method to fetch available services based on the criteria provided.

**STEP 1:**

Write following stored procedure to fetch available service types in the TSV2 database to which your code is pointing to.



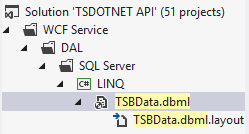
**STEP 2:**

Now for using this stored procedure with LINQ to SQL, we need to generate mapping classes in the TSV2 code to map to the stored procedure you have just written.

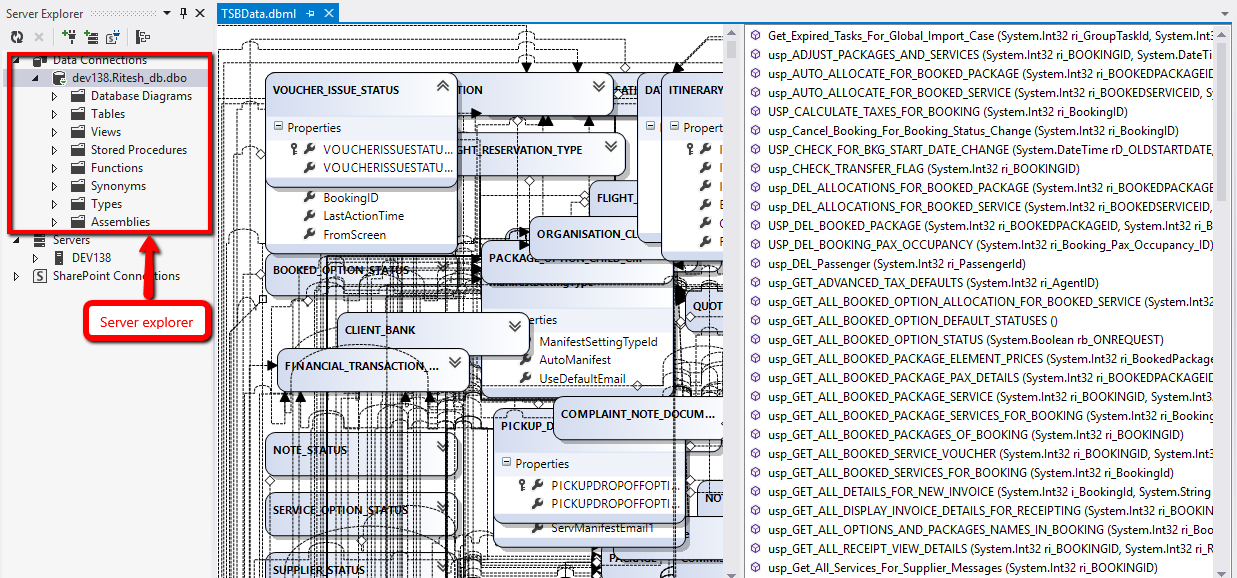
Go to the location

WCF Service 🡪 DAL 🡪 SQL server 🡪 LINQ (class library project)

Find and Double click on TSBData.dbml



This should open up the tab as shown below.



**STEP 3:**

Next go to “View” menu 🡪 Click on Server explorer

This should open up the server explorer as shown in the figure above.

Here make a new connection to point to your database 🡪 browse to “Stored Procedures” folder 🡪 Locate stored procedure GET\_SERVICE\_TYPES 🡪 drag and drop it in the right hand side of TSBData.dbml file were all the stored procedures are listed.

This should create required mapping classes for you under TSBData.designer.cs file.

**NOTE:**

*It becomes very cumbersome to drag and drop your stored procedure to TSBData.dbml file as shown above.*

*Work around is that, you can create a separate temp.dbml file in the project 🡪 add your stored procedures*

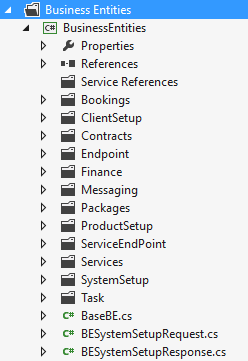
*And then you can copy the generated classes into a partial class “DBExtensions.cs”*

**STEP 4:**

Next you have to create business entities so that the data received from the database can be converted into the form which Business access layers (BAL) understand.

Go to the folder

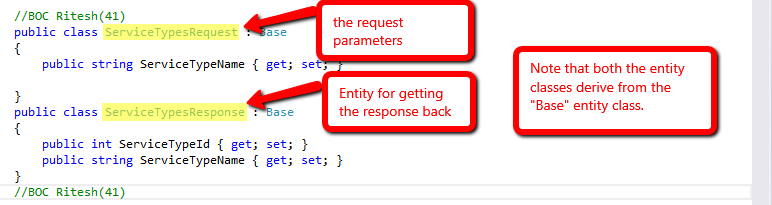
WCF Service 🡪 BAL 🡪 BusinessEntities (class library project)



Notice here that business entities are put under different folders based on the functionality.

Since the current functionality we are writing here is related to the setup of the product, we will put our entities under the folder “ProductSetup”.

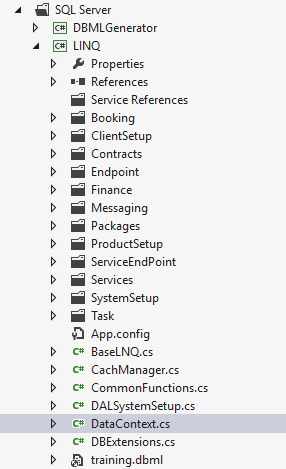
Since this is the common functionality which will be used across all the modules, locate for CommonBE.cs and create following business entities there.



**STEP 5:**

Now it’s the time to write some data access layer logic to fetch the data from the database.

Go to the folder

WCF Service 🡪 DAL 🡪 SQL server 🡪 LINQ (class library project)

Notice here that data access layer logic is put under different folders based on the functionality.

Since the current functionality we are writing here is related to the setup of the product (i.e. getting all service types), we will put our data access logic under the folder “ProductSetup”.

Since this is the common functionality which will be used across all the modules, locate for CommonLNQ.cs and write the following method there.



Notice that the above method takes Request Business Entity as the argument and returns back the Response Business Entity.

**STEP 6:**

Now let’s build a web service which will allow clients to fetch the data.

**NOTE:** *We will**make use of the**existing web service and add a new method in that.*

First let’s create data contracts.

Go to the folder

WCF Service 🡪 WCF 🡪 expand DataContracts (Class library project) 🡪 ProductSetup folder 🡪CommonDC.cs

Create Data Contracts and Message Contracts as shown below.



**STEP 7:**

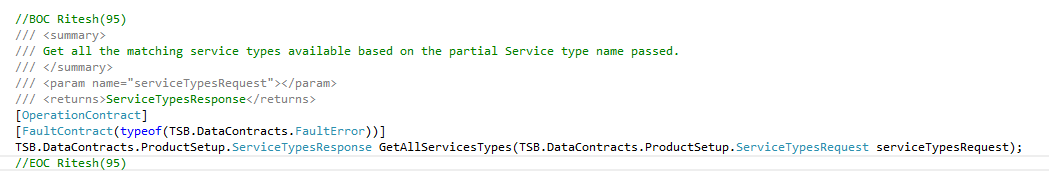
Now let us create a service contract.

Go to the folder

WCF Service 🡪 WCF 🡪 expand ServiceContracts (Class library project) 🡪 ProductSetup folder 🡪 Open IProduct.cs

Notice that this is an interface in which you have to add the Service Contract method declaration.

Go to the end of the page and within the interface IProduct which is the service contract, put the operation contract as follows.



**STEP 8:**

Now you have to provide the definition for this method in your web service class which implements this interface.

Open the web service file ProductSetup.svc and implement the method GetAllServicesTypes which comes from interface IProduct.

**NOTE :** *This method takes Message Contract for request as the argument and returns Message Contract for response.*

Write the following implementation in the code



**STEP 9:**

Now it is a golden rule that whatever logic you write in the BAL, it should be written strictly on the business entities. (Never pass data contracts to BAL).

So before sending request data to the BAL, Request Data Contract(DC) needs to be converted into Request Business Entity (BE).

This is done in the controller.

Go to the folder

WCF Service 🡪 BAL 🡪 Controller 🡪 expand Controller (Class library project) 🡪 ProductSetup 🡪 open CommonCTL.cs

Here write the following code



**STEP 10:**

Now make a call to a Core Business where any business specific logic will be applied on the BE.

Go to the folder

WCF Service 🡪 BAL 🡪 Core Business 🡪ProductSetup 🡪 open CommonCB.cs

In your case, you don’t have any specific logic to be applied here.

Just put the following code which will make a call to the DAL by passing the Request BE.

This method is going to return a Response BE back.



**STEP 11:**

Now your API side code is ready.

Just compile the solution and browse ProductSetup.svc from IIS.

Try to find a function GetAllServicesTypes in WSDL of the browsed service.

If everything went good, you shall be able to locate this method.

Congrats! Your API is now running for getting all Service Types available.

**STEP 12:**

Go ahead and write a web method for getting all the services available with the search criteria specified.

**Request object should have** 🡪 Service ID, Service Type ID and Service Name

**Response shall be list of objects having properties** 🡪 Service ID, Region Name, Service Type Name, Service Name, Supplier Name, Service Status Name

**STEP 13:**

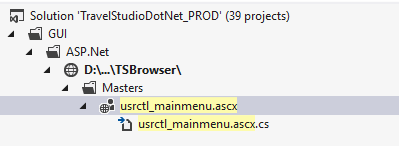
Go ahead and test for the method in the WSDL.

Once this is done, now only part left is to consume these methods in the client application (i.e. TSV2 Web Application)

**NEXT IS TO CREATE A WEBFORM FOR THE MODULE AND CONSUME WEB METHODS WRITTEN**

**STEP 14:**

To begin with designing, first replace the user control “usrctl\_mainmenu.ascx” which will be provided to you under the folder as shown in the figure below:



Go to the folder

Shared Control 🡪 MultilingualManager (class library) 🡪 Controls 🡪Menu 🡪 open Menu.en-US.resx

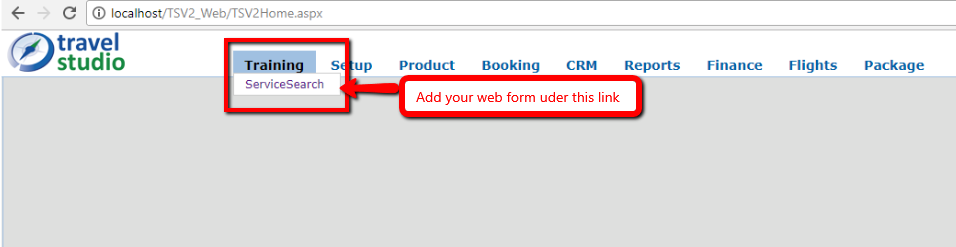
In the tab that opens, create a new record with following details:

Name 🡪 Search\_Service

Value 🡪 ServiceSearch||ServiceSearch

Browse Login.aspx and log in using your credentials.

You should be able to see a new Training menu added as shown in the figure below.



**STEP 15:**

Now go to the folder

GUI 🡪 ASP.Net 🡪 TSBrowser 🡪Pages

Here as you can see all the web forms are added in the folder as per their functionality.

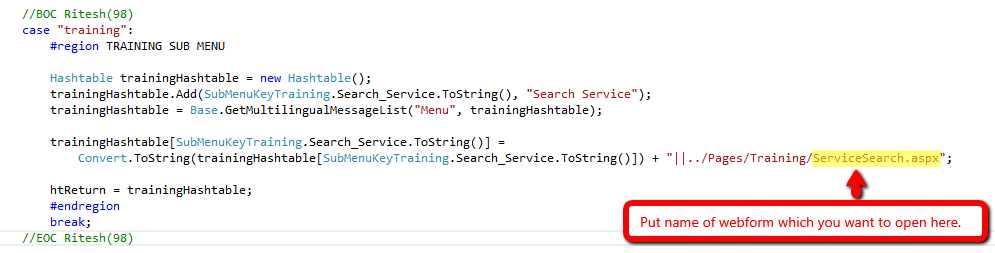
Now though you are creating a functionality related to service, for the training purpose you are going to create a new folder called “Training” under folder “Pages”.

Once that is done go ahead and put the web form “ServiceSearch.aspx” which will be provided, under Training folder.

**STEP 16:**

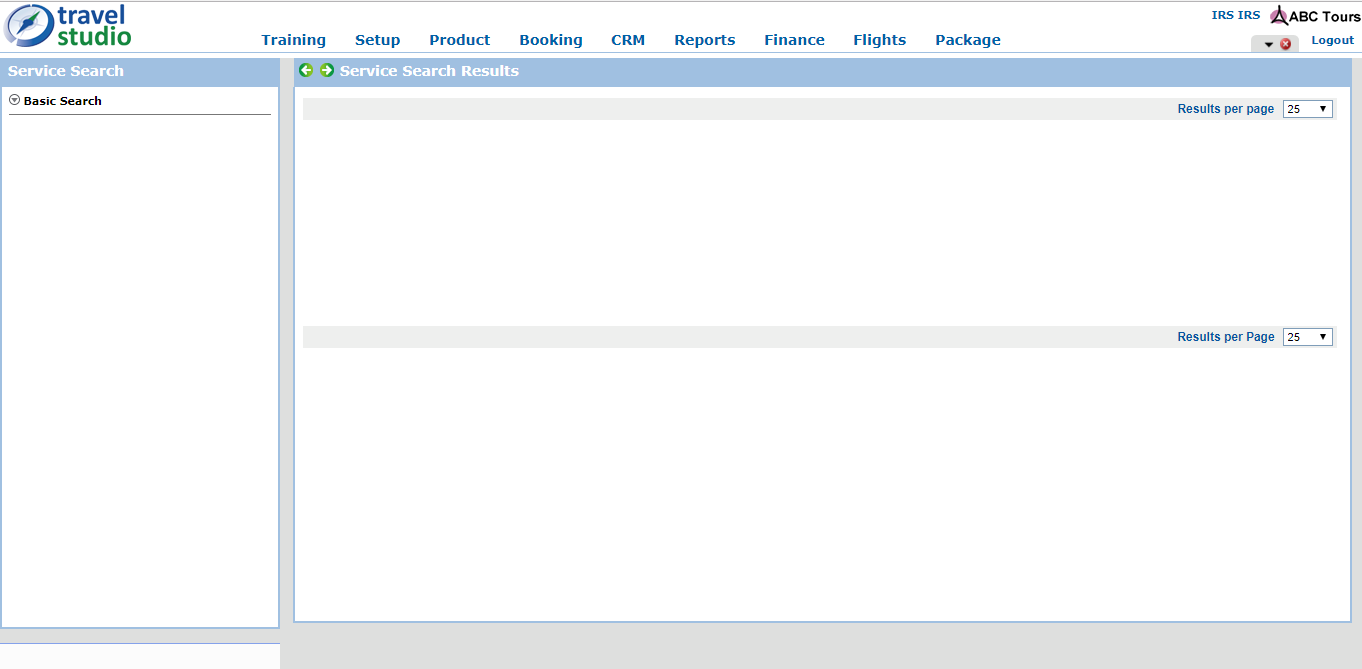
Open the code behind file of the user control “usrctl\_mainmenu.ascx” which you had replaced in the step 14.

Here you have to put link of the webform you just created.



Once done, browse and open the form.

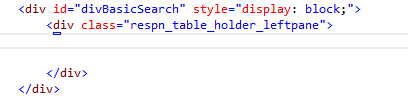
This should be a blank page as shown below.



**STEP 17:**

Now you have to create the required controls in the above blank form.

To create search criteria on the left hand side of the page, go to the following section in ServiceSearch.aspx

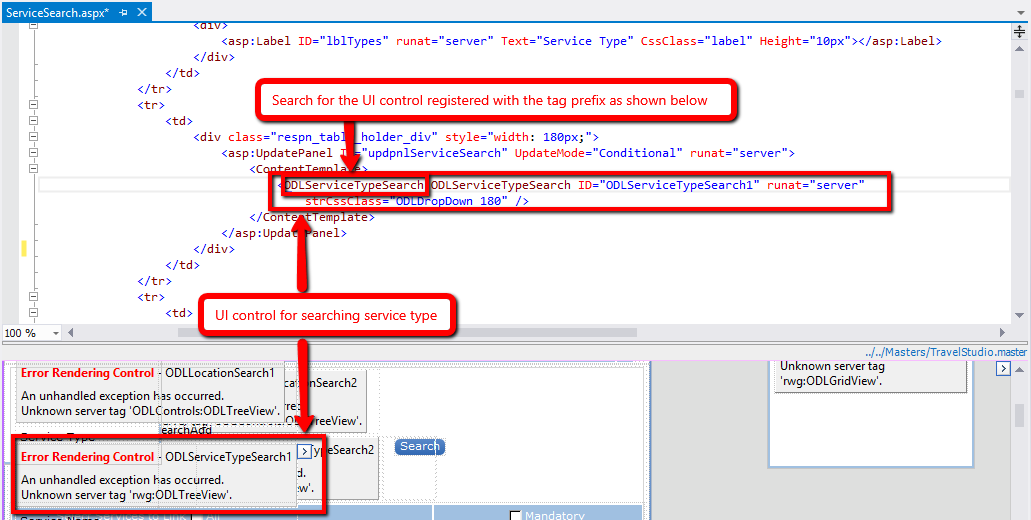


You can create controls for required search criteria here

You can follow the code shown below.



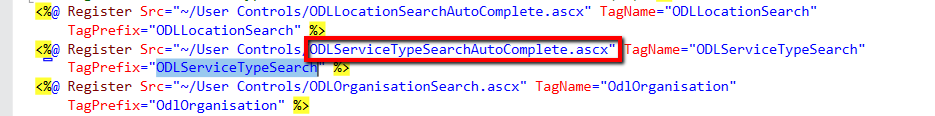
Notice that for filling all the service types which are available you have to use ODL’s custom user control designed for the same.



**NOTE:** *You are going to use the same user control for this exercise. But you are going to fetch details from the web method written by you.*

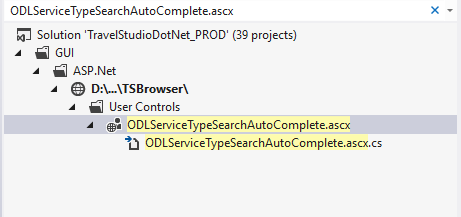
**STEP 18:**

To find the folder path in which custom user control is located, look for the User Control registered on the page with “TagPrefix = ODLServiceTypeSearch”.

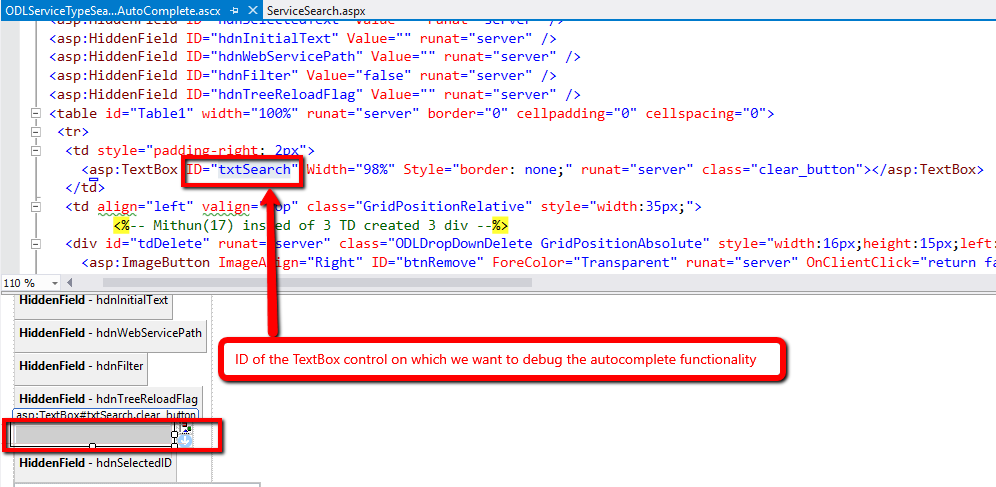


Find ODLServiceTypeSearchAutoComplete.ascx location in the solution explorer and open the file.

All the custom User Controls are placed under the User Controls folder.



In the User Control find for the Service Type Text Box on which we want to debug the auto complete functionality.



NOTE: *Auto complete is implemented on the Text Box using JQuery UI auto complete.*

*jQuery UI is a collection of GUI widgets, animated visual effects, and themes implemented with jQuery (a JavaScript library), Cascading Style Sheets, and HTML.*

*Link to JQuery UI auto complete -* [*https://jqueryui.com/autocomplete/*](https://jqueryui.com/autocomplete/)

**

*Ajax is a client-side script that communicates to and from a server/database without the need for a postback or a complete page refresh.*

*“the method of exchanging data with a server, and updating parts of a web page – without reloading the entire page.”*

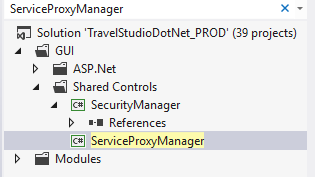
**NOTE:** *You are going to configure the URL in this AJAX call later in the exercise. First create the Proxy class for your Web Service.*

**STEP 19:**

First and foremost we need to keep our Web Service running and update the proxy class for ProductSetup.svc Web Service, so that classes are generated for our newly generated Web Service method.

To do this, follow these steps.

i ] Go to the location as shown below



ii] Open ServiceProxyManager in file explorer

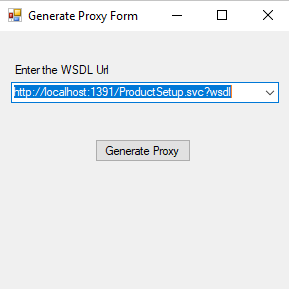
iii] Open file path as shown below

TravelStudioDotNet\_PROD\GUI\SharedControls\ServiceProxyManager\GenerateProxy\

GenerateProxy\bin\Debug

Locate GenerateProxy.exe and run the application as Administrator

Following window will pop up



iv] Enter the WSDL URL for ProductSetup.svc in the text box and click on Generate Proxy.

NOTE: *Ensure your Web Services is hosted and running.*

This should update your proxy class to include classes for the newly generated Web methods.

**STEP 20:** *Once proxy has been updated, go to the folder*

*GUI 🡪 Shared Controls 🡪 expand ServiceProxyManager 🡪 ProxyClasses 🡪 open ProductSetup.cs*

Here check for the classes generated for your Service Contracts and Data Contracts.

OR

Browse PoductSetup.svc from IIS 🡪 open WSDL document 🡪 Look for Service Contract.

**NOTE:** *If everything went fine, you should be able to find your method.*

*Repeat step 19 and 20 for Service.svc*

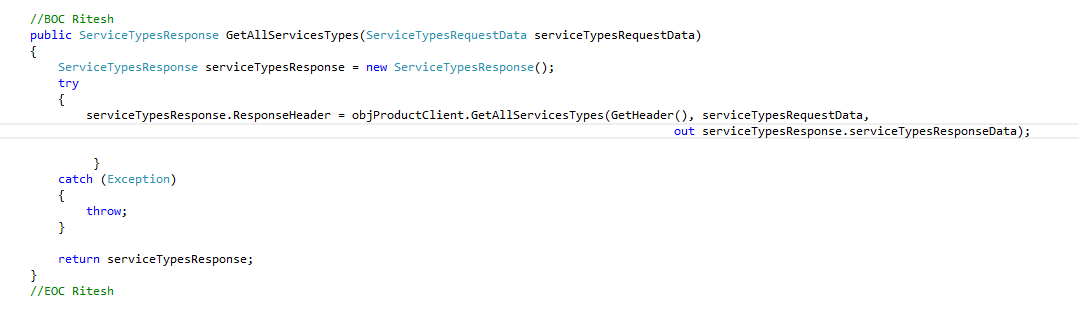
**STEP 21:**

Now you have to write a factory method.

Go to the folder

GUI 🡪 Shared Controls 🡪 expand ServiceProxyManager 🡪 Factory 🡪 GenericControls 🡪 open GenericControls.cs

Go at the end of the file and write the following method

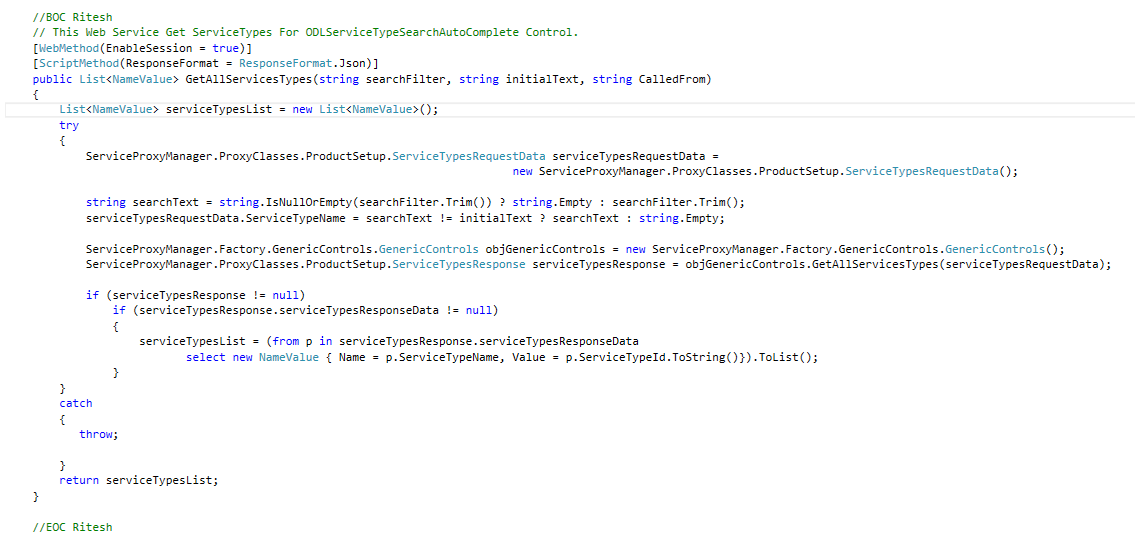
**

**STEP 22:**

Go to the folder

GUI 🡪 ASP.Net 🡪 TSBrowser 🡪 AjaxControlsWebService 🡪 open JUIAutoComplete.asmx

Go to the end and write the following method



**STEP 23:**

Open ODLServiceTypeSearchAutoComplete.ascx

Go to the URL section of the AJAX call and change the URL to match a method written by you.



**STEP 24:**

Browse and login to the application.

And now check for the autocomplete functionality.

If everything went fine than at this point your autocomplete should work.

**STEP 25:**

Go ahead and write the functionality for search button. It should populate the searched services in the grid.