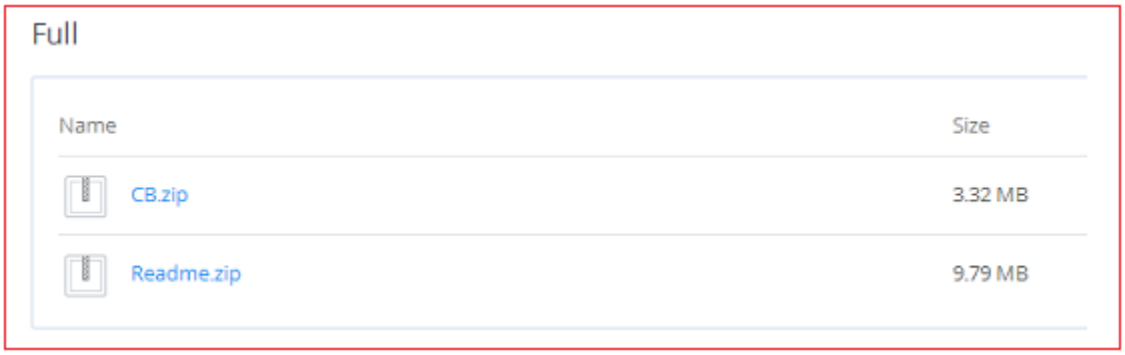
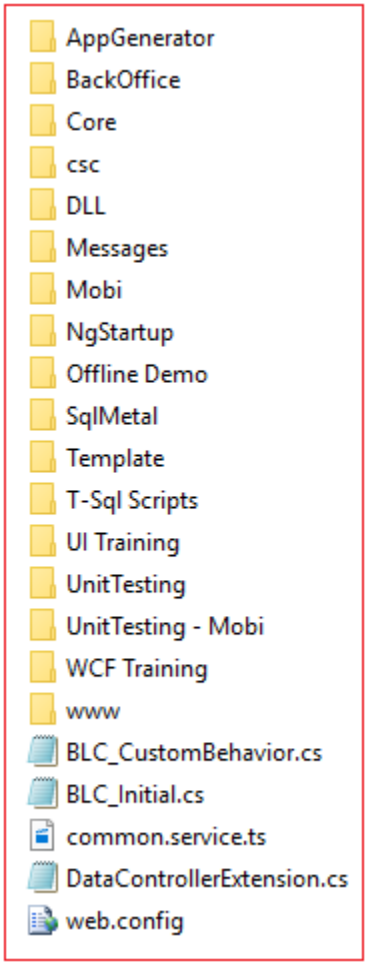
Create an Application Using Code Booster

*. Visit the following GitHub Repository and download both CB.zip & Readme.zip

*. If you press the previously mentioned link you will be redirected to Drop box link where you have to download the following zipped folders

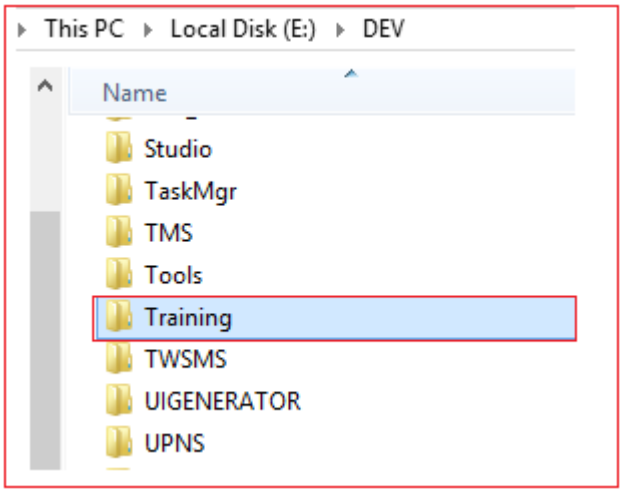


***. Extract the CB Folder. (You will find the following content)**

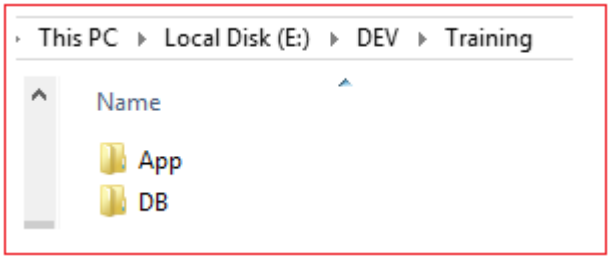


***. In order to start:**

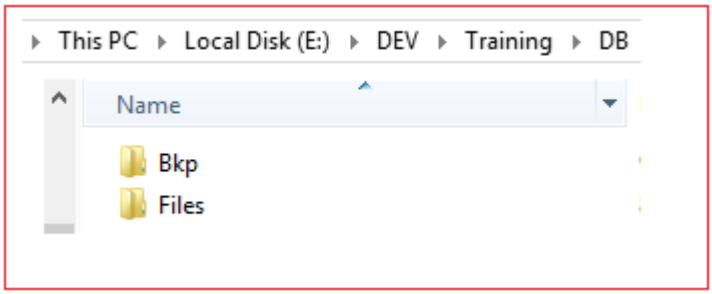
*. Create a folder named “**Training**” under your preferred drive (in my case E:\DEV\)



*. Under the previously created folder (**Training**), create two folders "**App**" & "**DB**" as the following

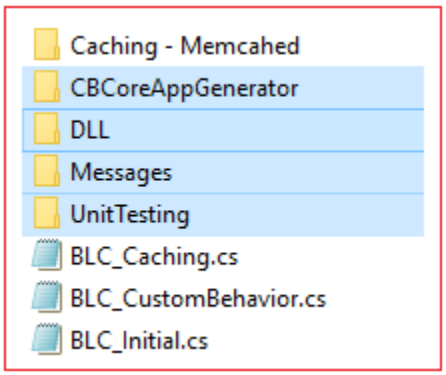


*. Under the "**DB**" folder (previously created) create 2 sub folders "**Bkp**" & "**Files**" as the following



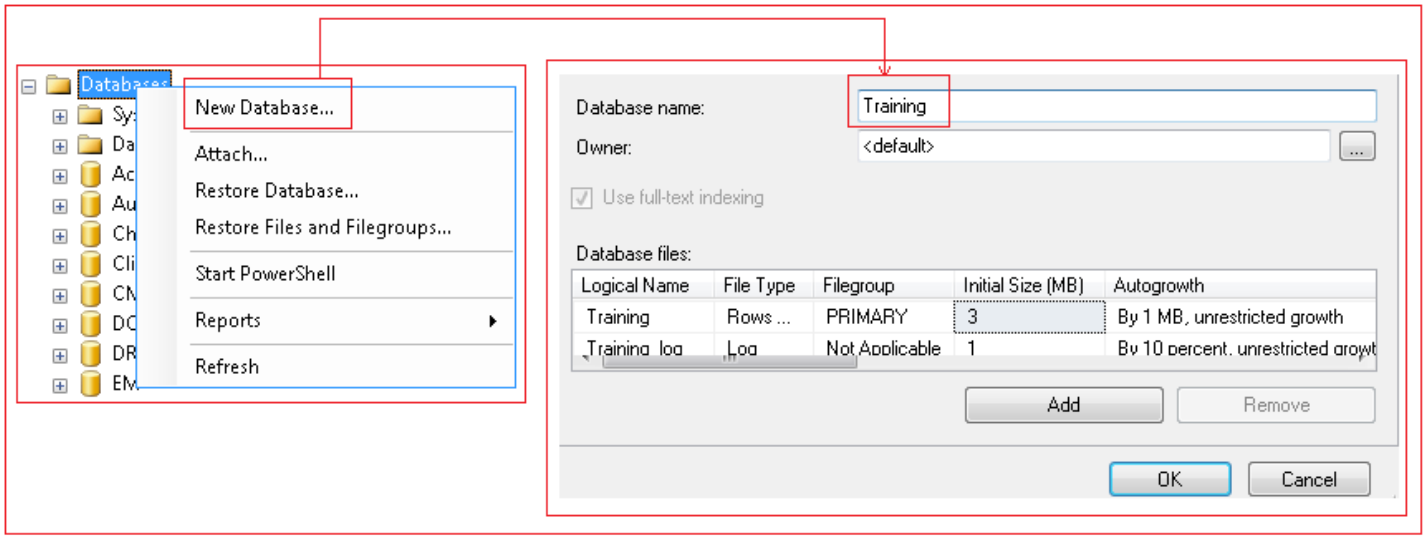
*. From the "**CB\Core**" folder (Already downloaded)

copy the following folders & Files to “App" folder you created under "Training" folder.



N. B: You will find under "**Readme**" folder a .pdf document "**10_Sql_Server_Installation.pdf**" showing how to install SQL Server with correct features.

***. Create an empty database named training & Assure that the .mdf & .ldf files will be created under "DB\Files" (already created) folder**

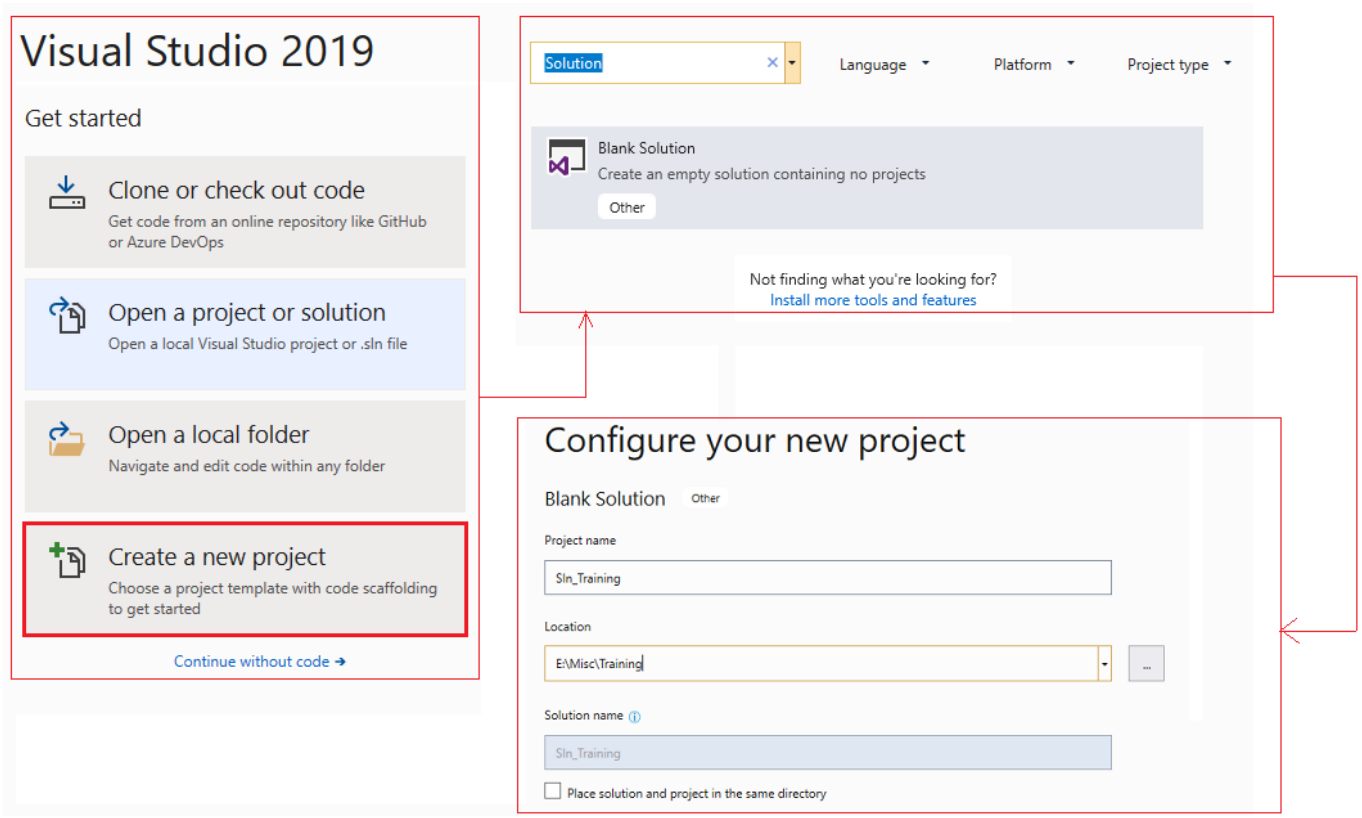


N.B:

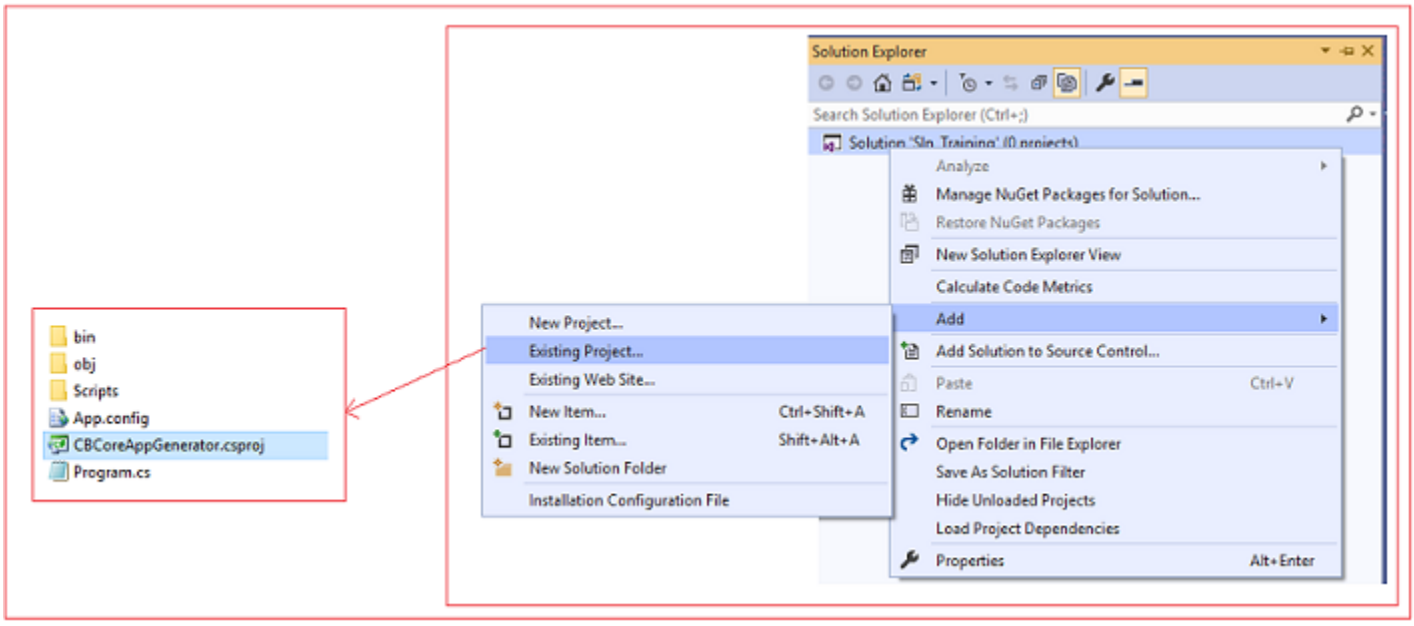
By default, the Sql Server will create database files under the Sql Server installation folder (mainly under c:\Program files) and this is not recommended.

Always assure that when you create a new database (or restoring one) to inform the sql server to create corresponding database files under **DB\Files** folder (previously created)

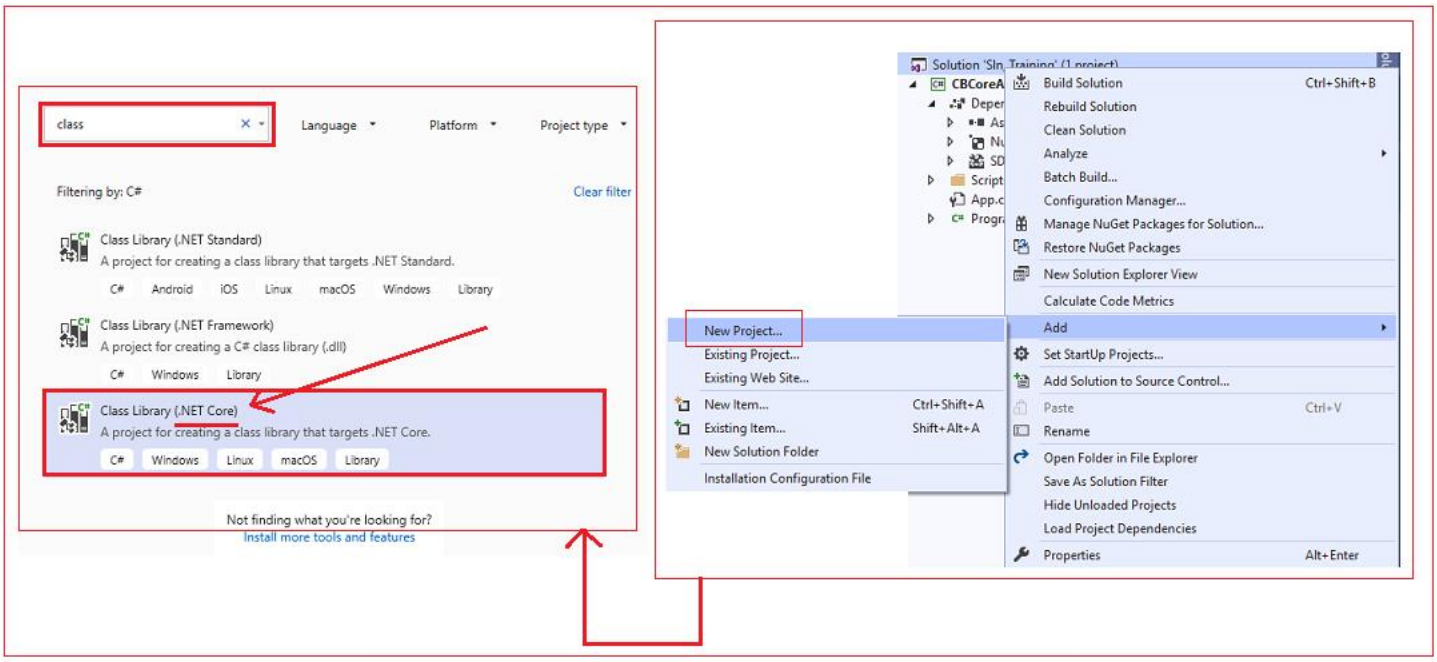
***. Create an empty solution under the "App" (already created) folder**



***. Add the AppGenerator Console Application, Found under the "App" Folder, to the previously created solution.**

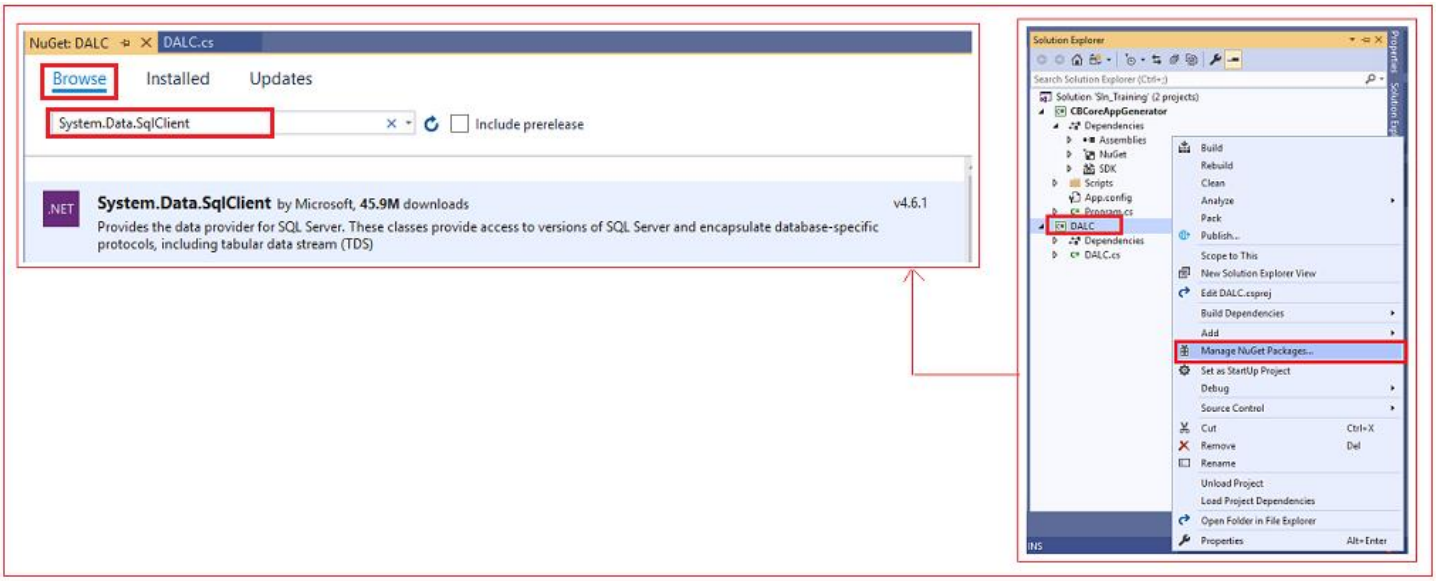


***. Add a Class Library Called DALC to the solution (it should be located under "App" folder)**

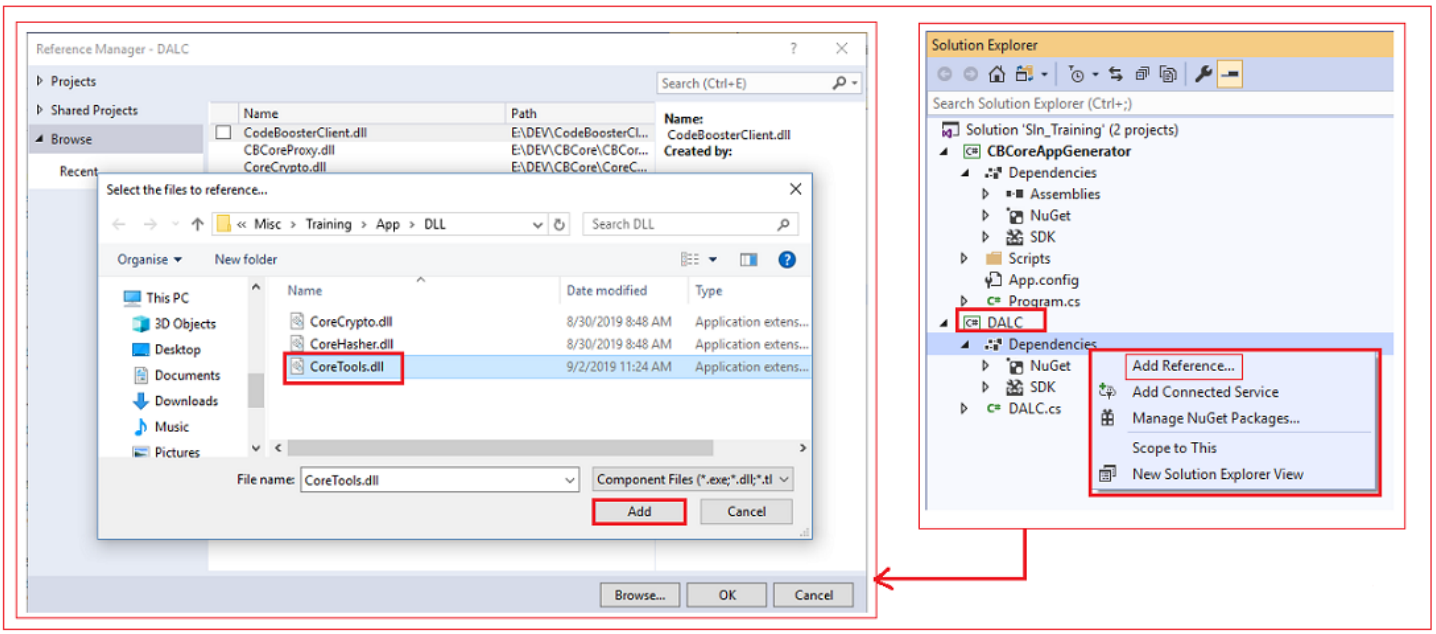


***. Rename the Class1.cs file name to DALC.cs**

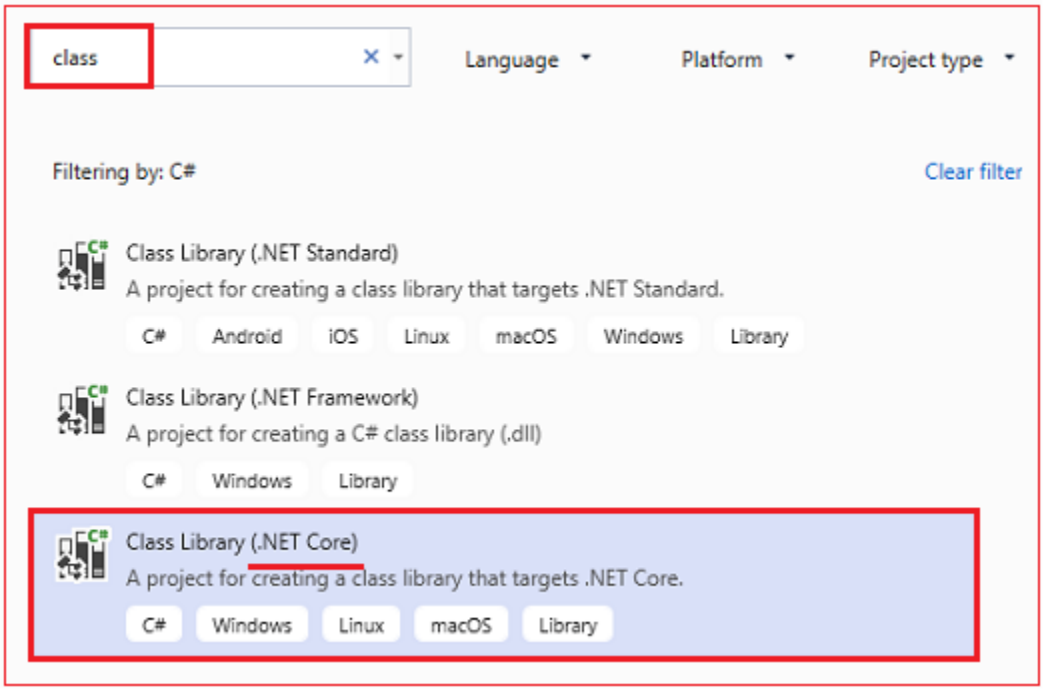
***. In DALC project, you should add nugget package named System.Data.SqlClient as the following**



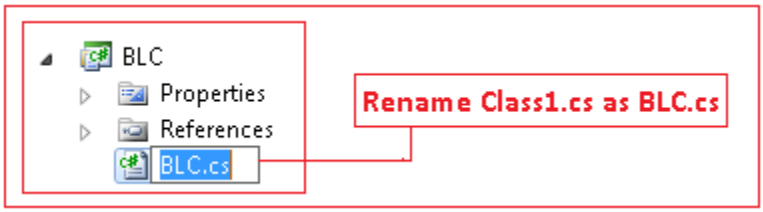
***. In DALC project, add reference to the CoreTools.dll file (that exists in the DLL folder under App Folder)**



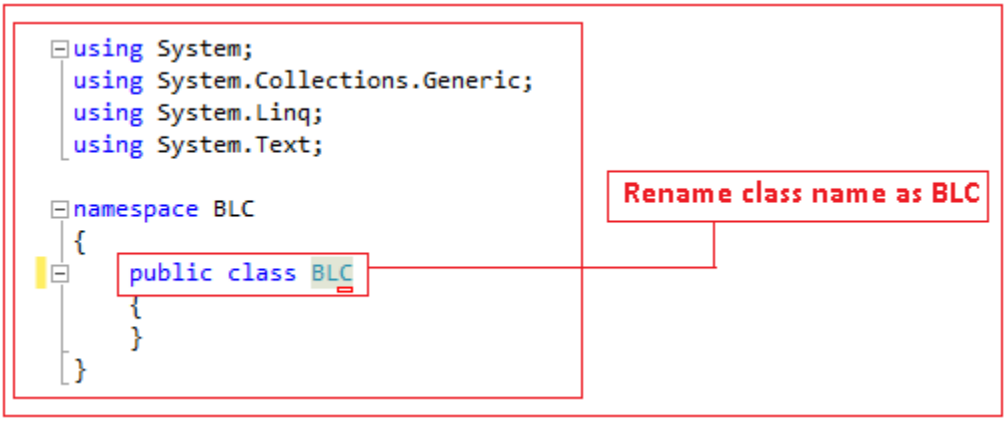
***. Add a Class Library Called BLC to the solution (it should be located under "App" folder)**



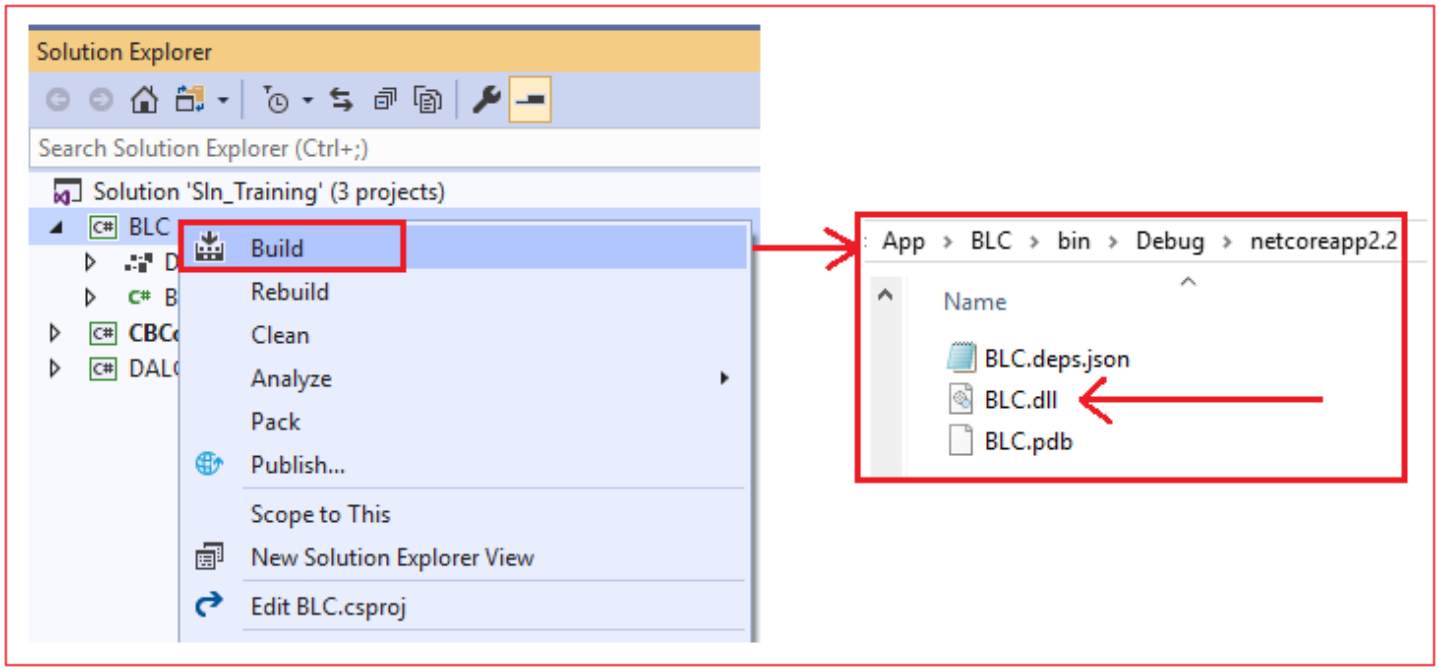
***. Rename the Class1.cs file name to BLC.cs**



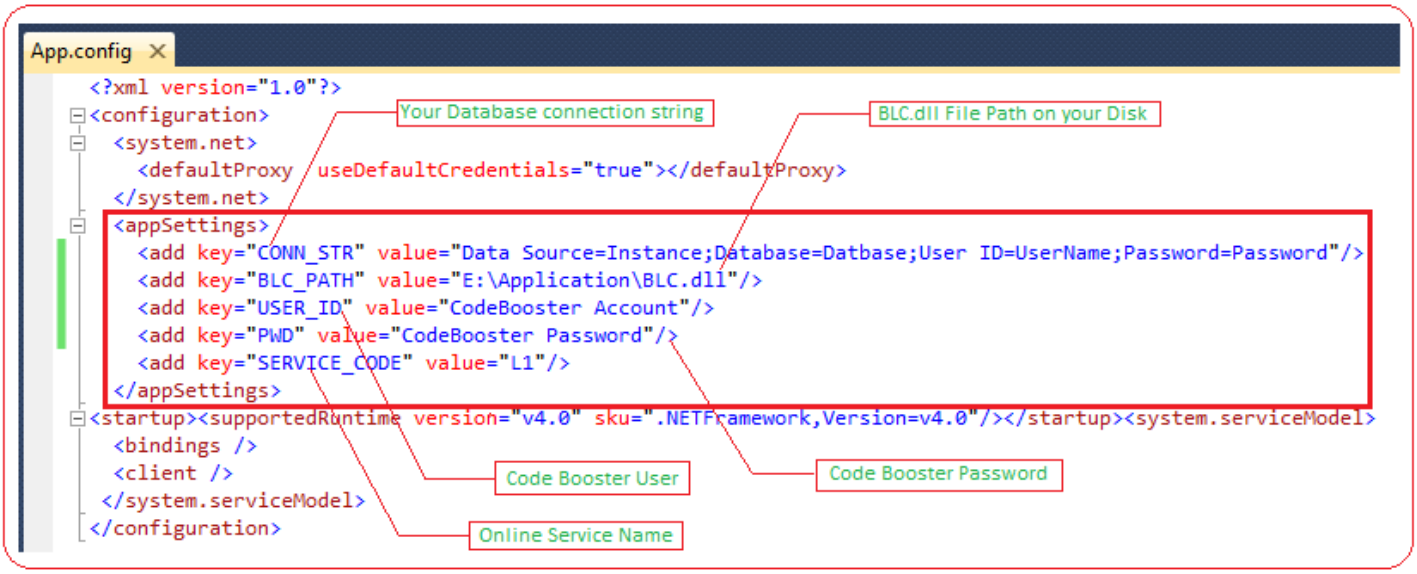
***. Rename the class name from Class1 to BLC**



*. **Build the BLC Class Library in order to generate the BLC.dll under the bin folder (to be used in the next steps)**



*. **Change the App.config of the AppGenerator console application:**



CONN_STR: Database connection string

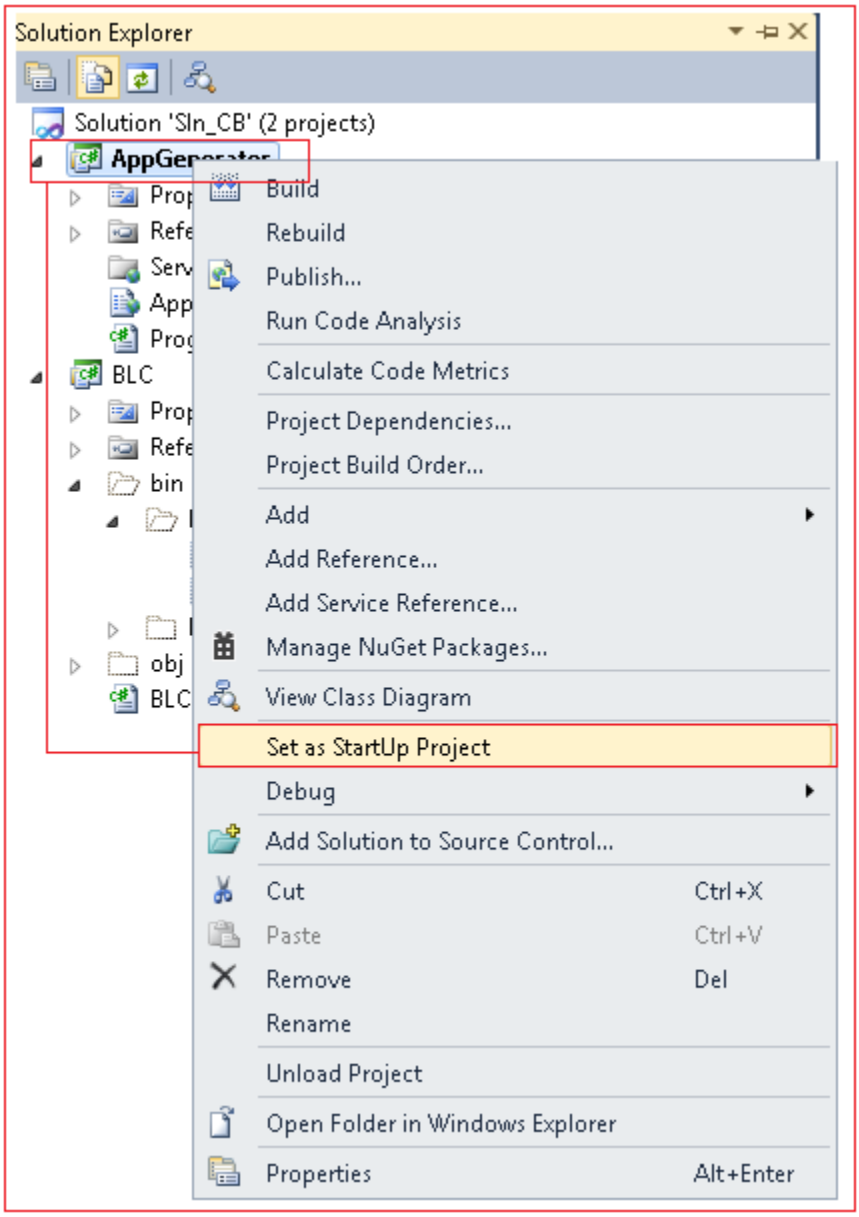
(In case you are using windows authentication, instead of Username & Password use Trusted Connection=True ;) as the following:

```
<add key="CONN_STR" value="Data Source=Instance;Database=Training;Trusted_Connection=True;"/>
```

BLC_PATH: The physical path of BLC.dll file already created when you built the BLC project, it should end by **BLC.dll.**

USER_ID, PWD & SERVICE_CODE will be provided by the Code booster service provider

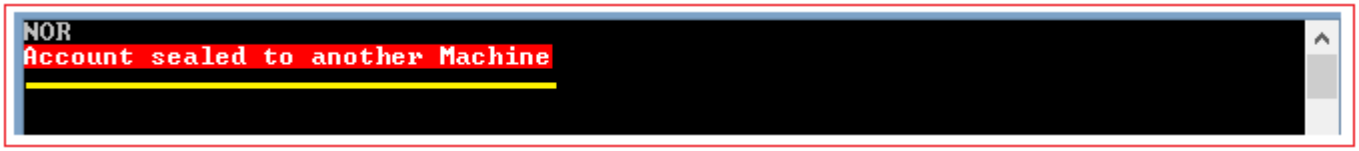
***. Set the AppGenerator as startup project.**



***. Run the AppGenerator console application and select option "001" (By typing 001 and press enter)**

***. Sealed Account**

It's mandatory to know that the first time you use code booster account it will get linked (Sealed) to the machine you are using & u cannot use the same account on different machine unless you send us an email on info@online-developer.net mentioning clearly that you want to un-link the account to be able to use it on different machine.



***. A Patch will be downloaded in the "D:\" directory.**

In case you don't have drive "D:\", you can change the location where the patch will be downloaded.

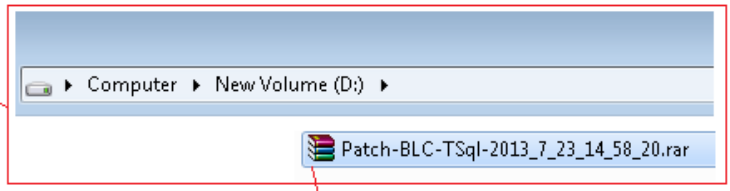
By changing the following line in the program.cs file under the AppGenerator console application

oCodeBoosterClient.Local_Patch_Folder = @"D:\";

N.B:

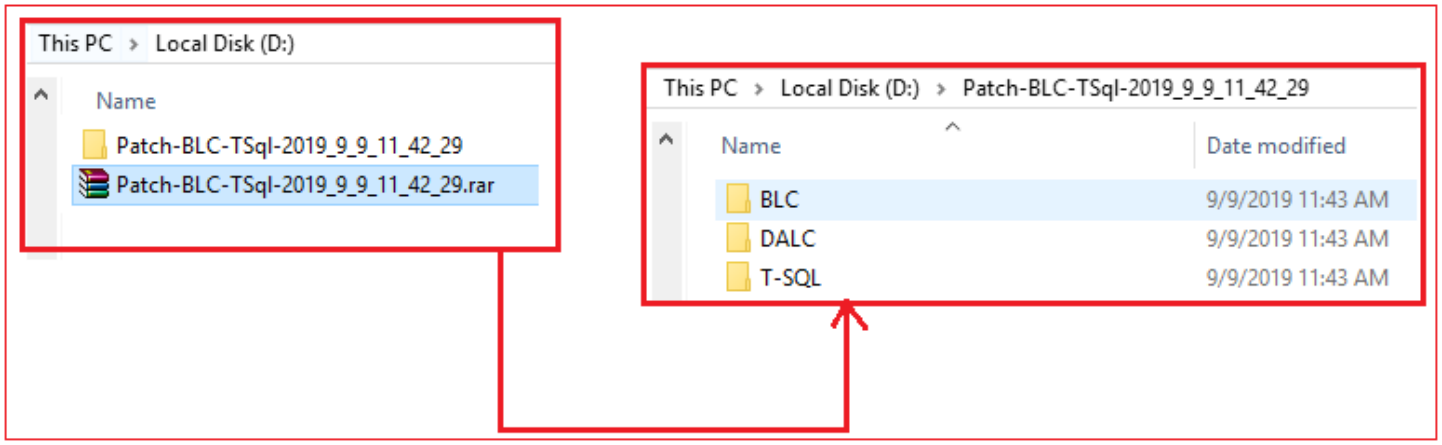
***.** In windows 8/8.1/10, changing the **Local_Patch_Folder** to "**C:**" will not be enough as writing on the Root is not allowed even with administrator privileges. A folder should be created for this purpose too i.e., C:\Temp

```
L1
Enter An Option:
001 --> Create SP's & BLC Layer
002 --> Generate WCF / JSON Code
003 --> Generate UI
001
Assure Required DB Objects existence : Start
Assure Required DB Objects existence : End
Assure Profiling DB Objects Existence : Start
Assure Profiling DB Objects Existence : End
Fetching DB Fields : Start
Fetching DB Fields : Done
Fetching DB Indexes : Start
Fetching DB Indexes : Done
Fetching DB FKs : Start
Fetching DB FKs : Done
Fetching UI Svc Actions : Start
Fetching UI Svc Actions : Done
Fetching UI Widgets : Start
Fetching UI Widgets : Done
Uploading DB Fields : Start
Uploading DB Fields : Done
Uploading DB Indexes : Start
Uploading DB Indexes : Done
Uploading FKs : Start
Uploading FKs : Done
Uploading UI Svc Actions : Start
Uploading UI Svc Actions : Done
Uploading UI Widgets : Start
Uploading UI Widgets : Done
Uploading Methods with Event : Start
Uploading Methods with Event : Done
Uploading WCF Methods : Start
Uploading WCF Methods : Done
Uploading Keys Mapper : Start
Uploading Keys Mapper : Done
Uploading BLC : Start
Uploading BLC : End
Downloading Patch : Patch-BLC-TSql-2013_7_23_14_58_20.rar : Start
Downloading Patch : Patch-BLC-TSql-2013_7_23_14_58_20.rar : End
Press Any Key To Exit
```

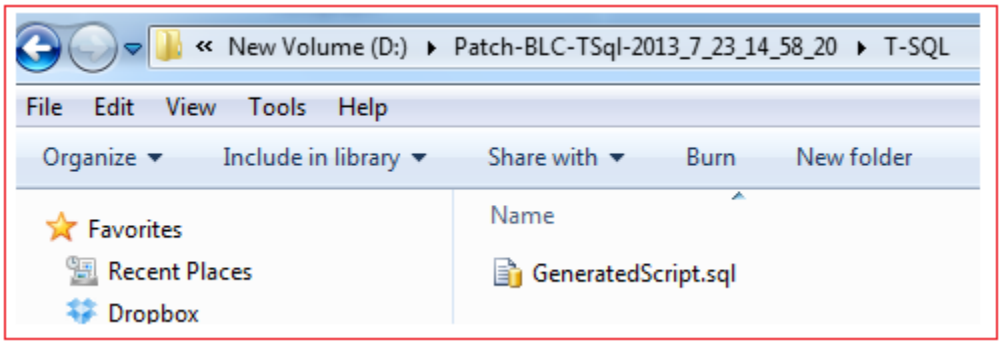


Patch generated and downloaded

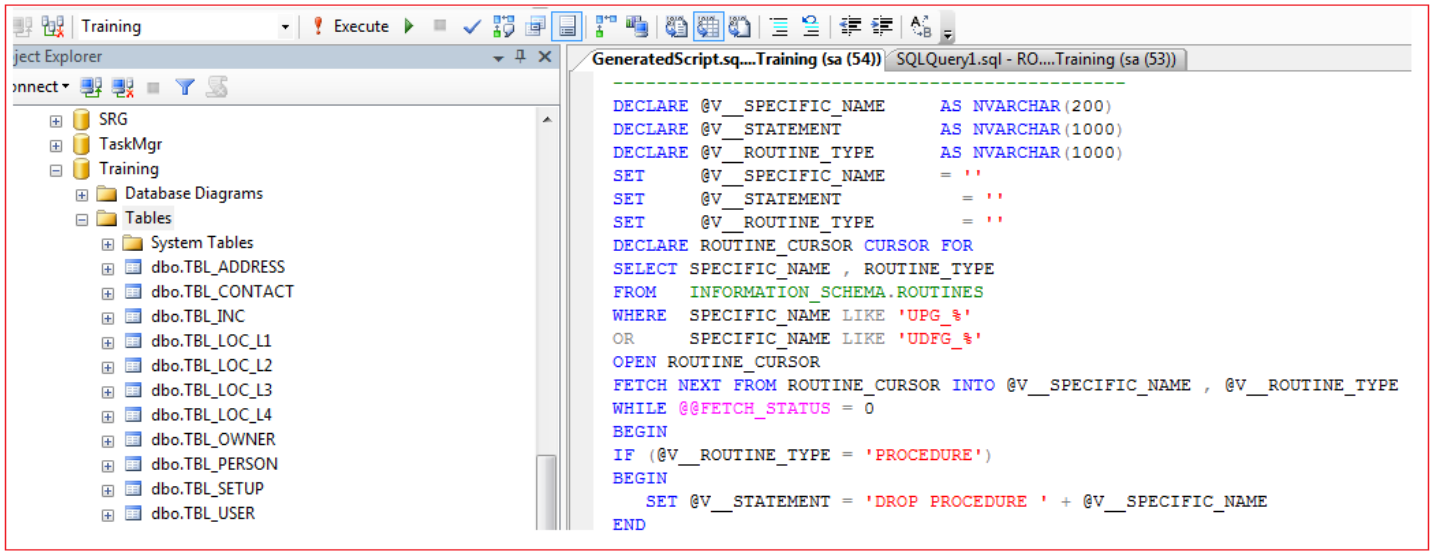
***. Unzip the patch (it's a .rar) and you will get 3 folders BLC, T-SQL & DALC**



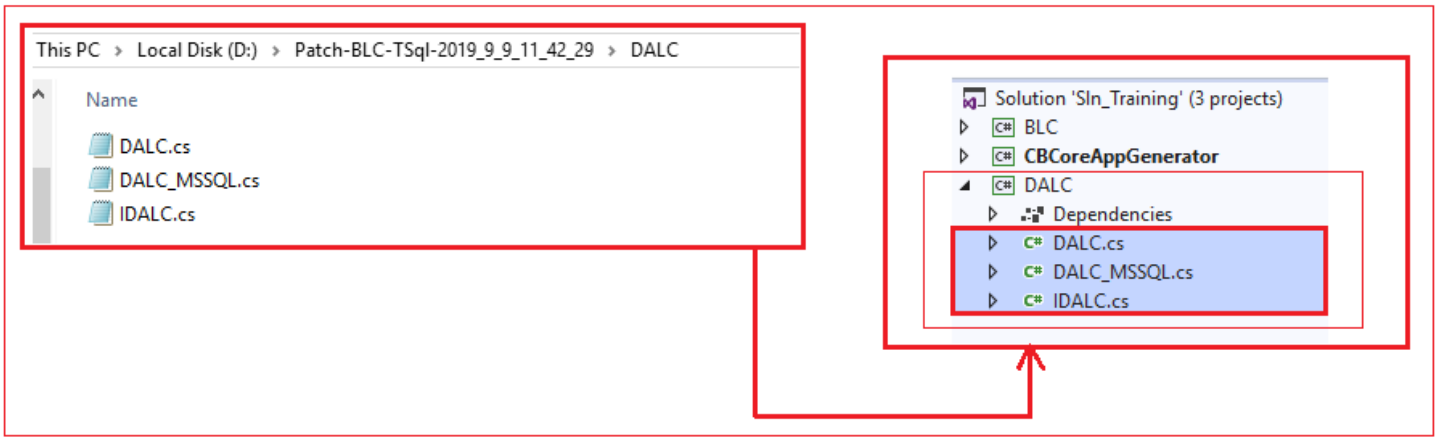
***. Under the T-SQL Folder you will find GeneratedScript.sql**



***. Run GeneratedScript.sql script on the newly created database.**

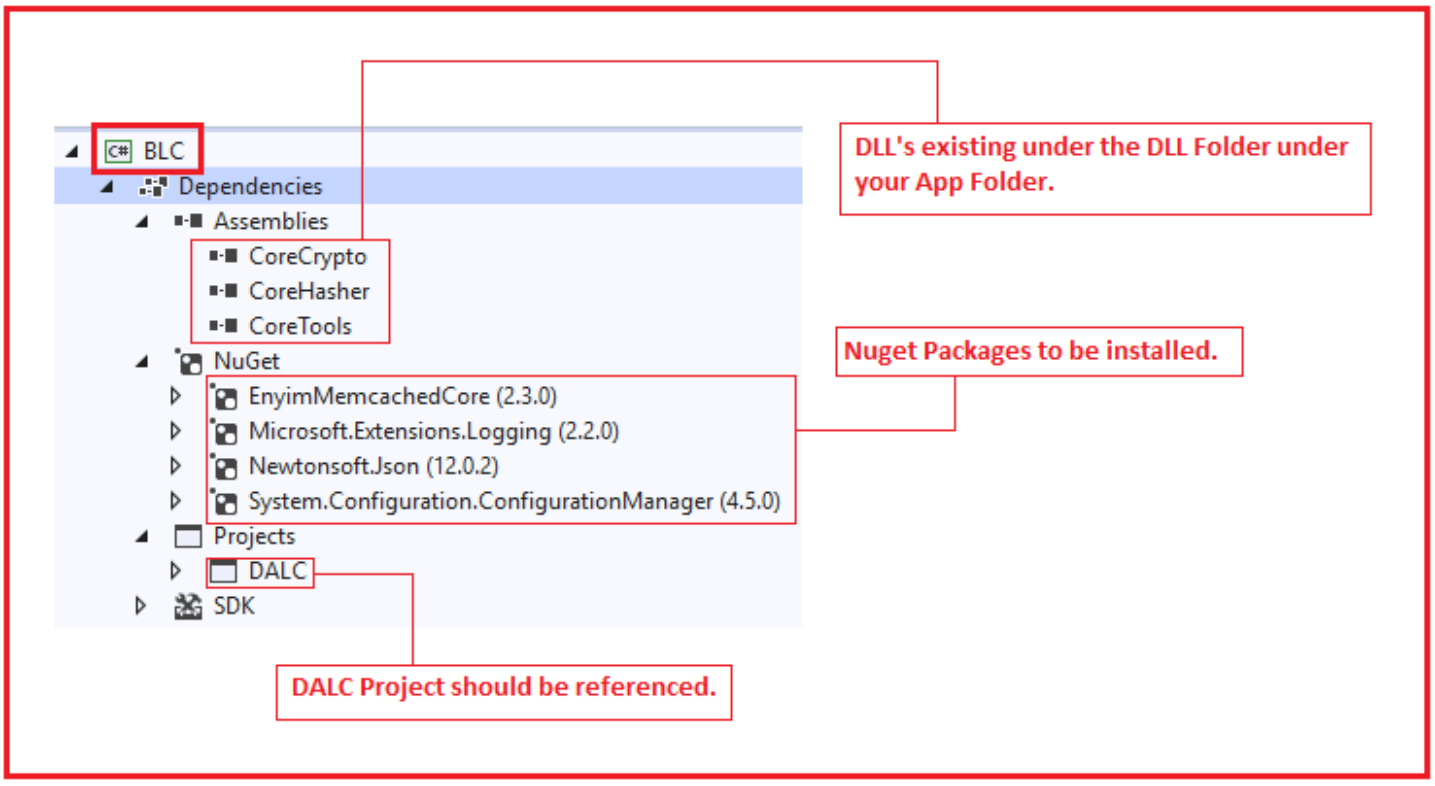


***. Override the files that exist under the DALC folder in the DALC Project**



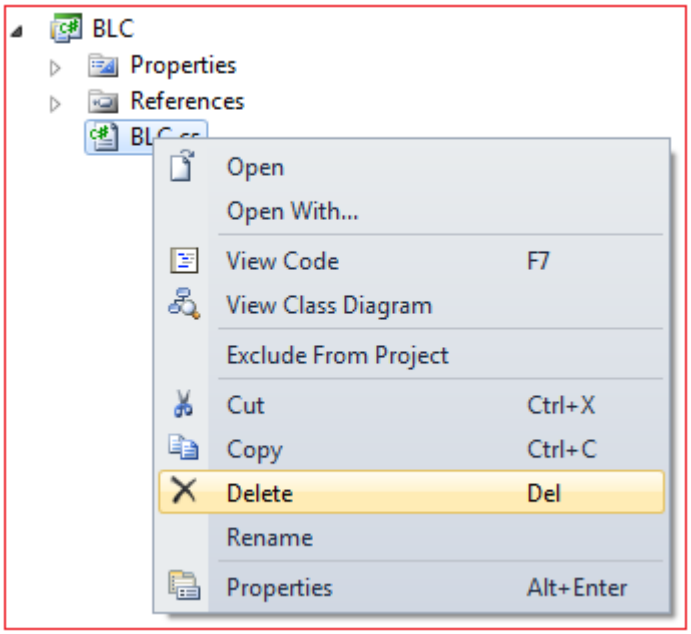
***. Build the DALC Project to assure that everything is Ok.**

***. In BLC Project, you should reference the following:**

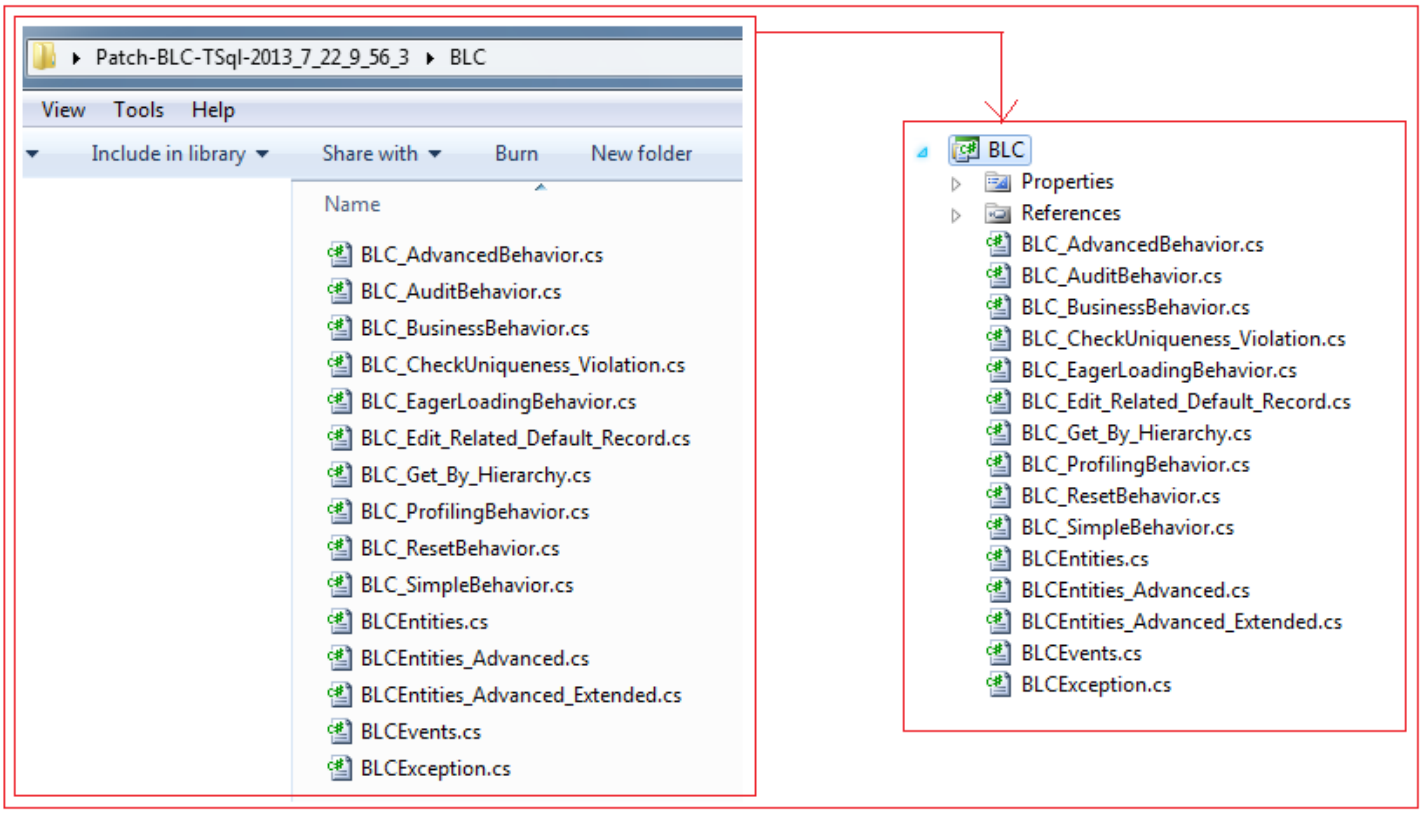


***. In BLC Project:**

***. Delete the BLC.cs file (Initially created empty)**



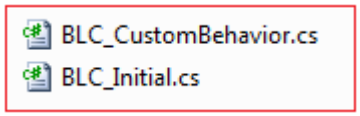
***. Copy/Paste the content of the BLC Folder (located under the patch folder) under the BLC Project.**



***. Add to BLC project the following files [You can find them under CB\Core Folder]:**

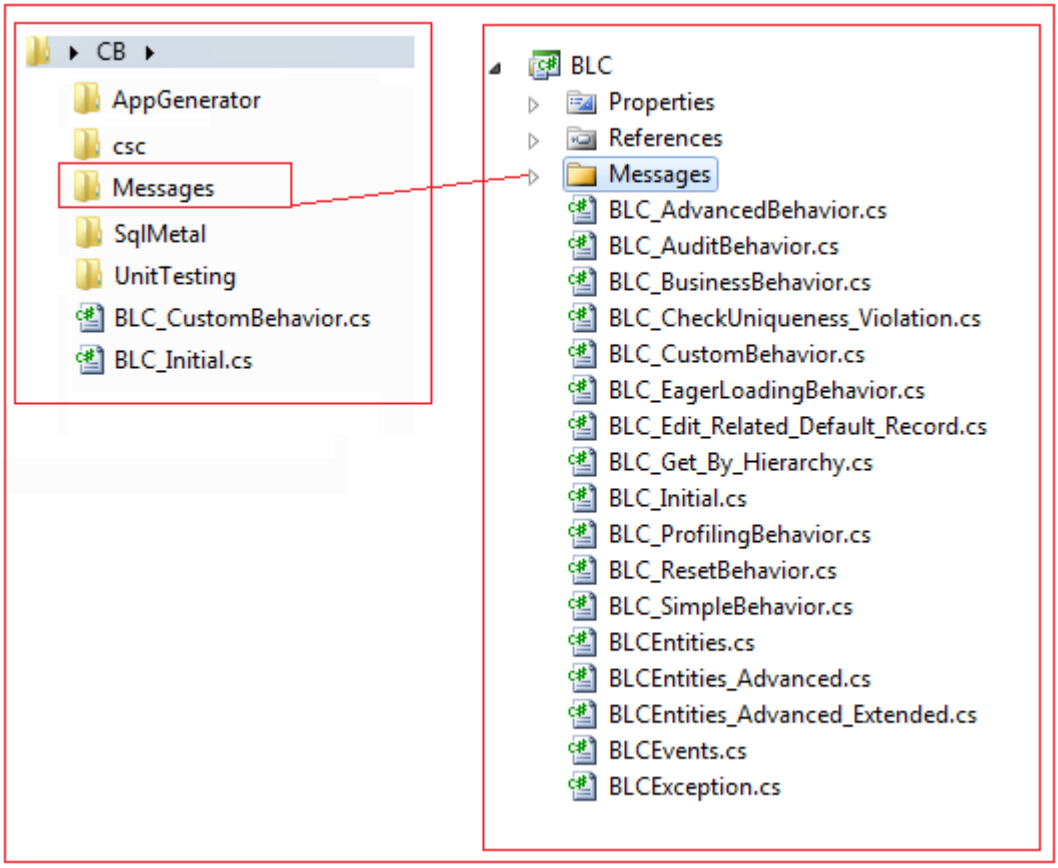
***. BLC_Initial.cs**

***. BLC_CustomBehavior.cs**

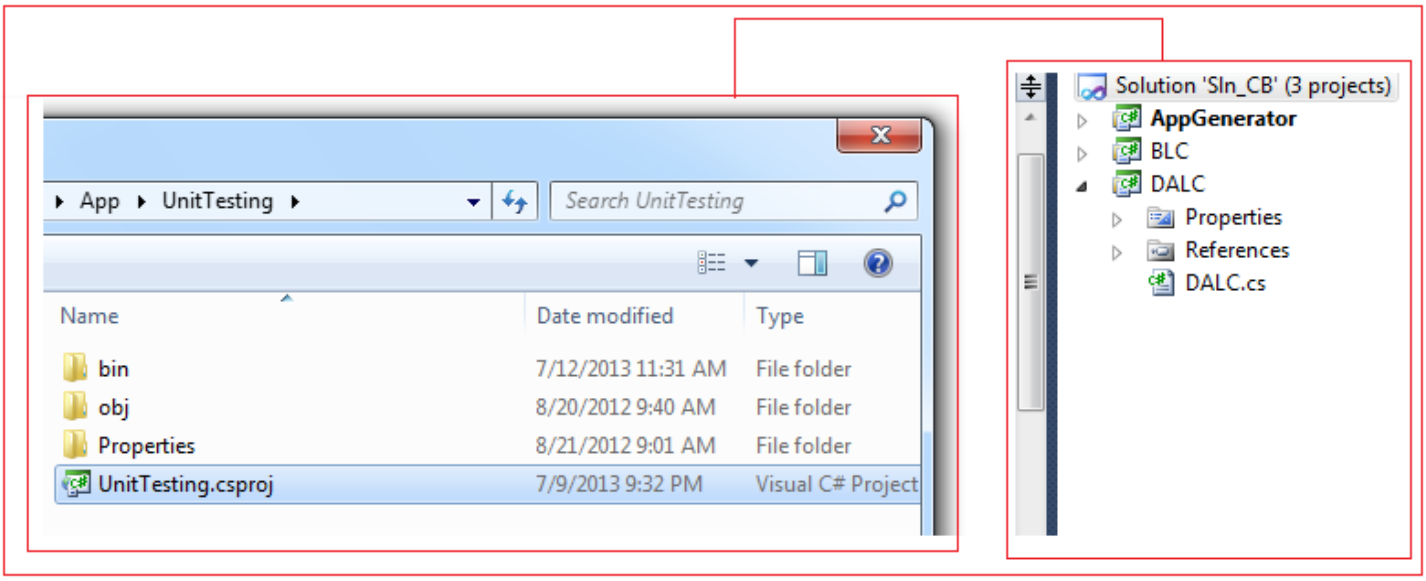


***. Build the BLC Project to assure that everything is working correctly.**

***. Under the "App" (already created) folder, you will find a folder called Messages that should be copied under the BLC Project**



***. Add to the solution the "UnitTesting" console application [residing under the App (already created) Folder].**

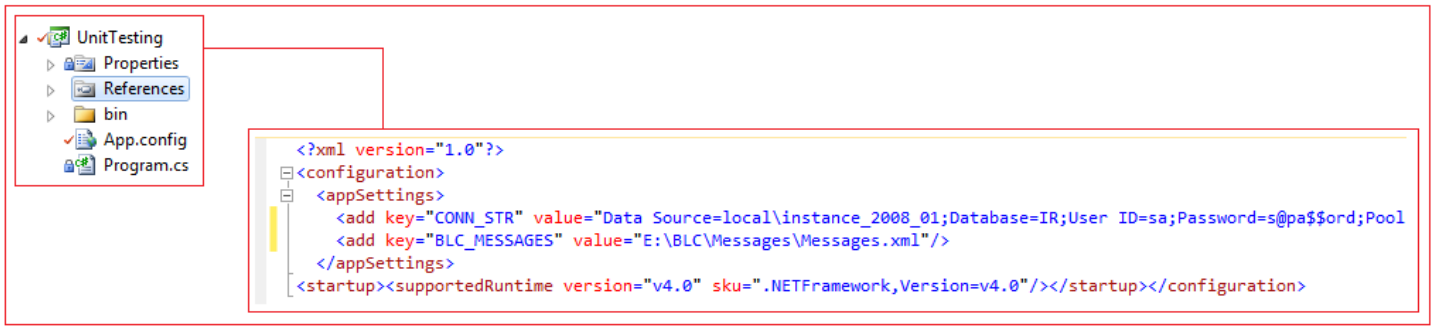


***. In App.config of the "UnitTesting" console application, change the following:**

***. CONN_STR : Pointing to your database**

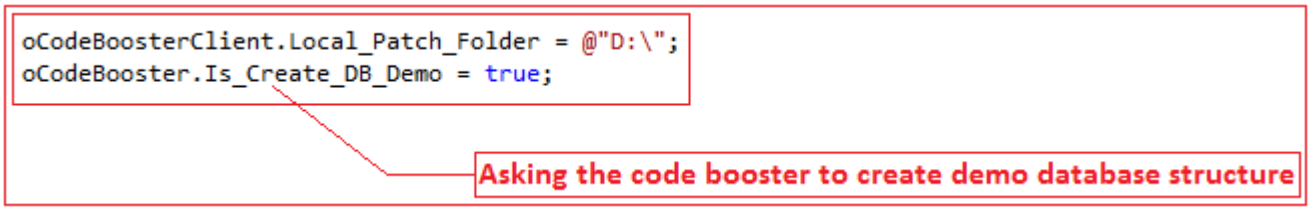
***. BLC_MESSAGES: pointing to the Messages.xml**

(You already created the corresponding folder under the BLC Project folder)



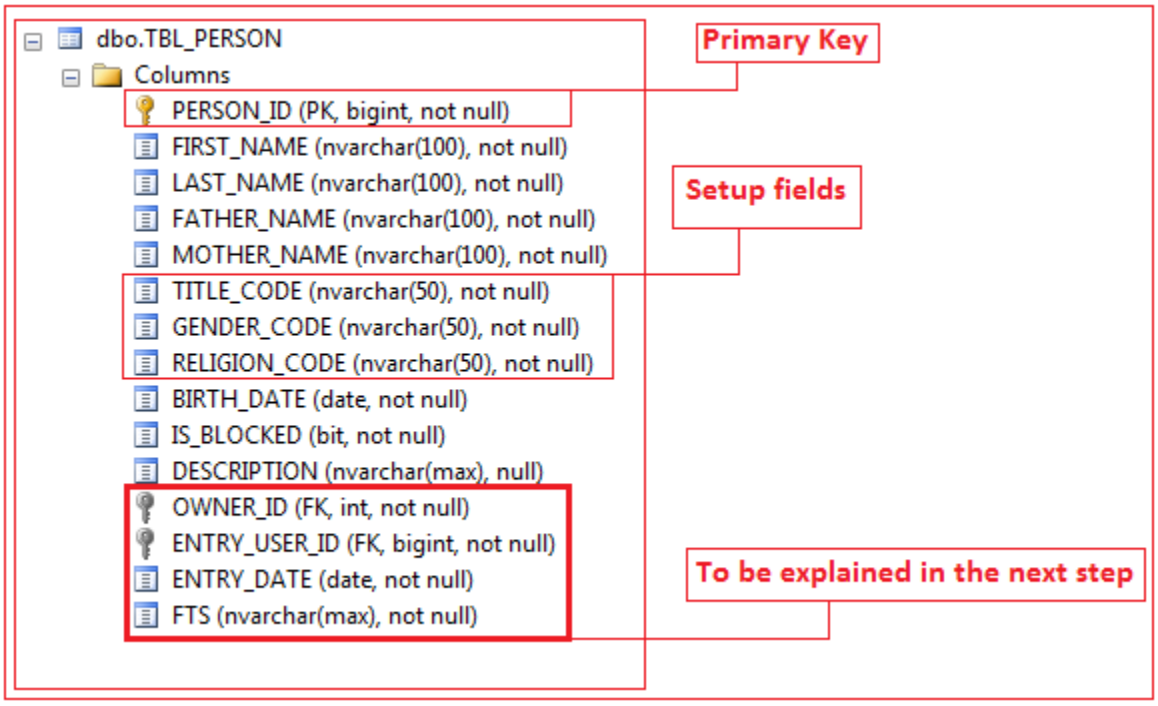
***. Database existence:**

We started this tutorial by asking you to create an empty database, but if you check the database you will find that some tables (stored procedures & functions) have been created: this is due to the fact that you asked the code booster to create a Demo database by writing the following line (in Program.cs of the AppGenerator Console application)



***. Database structure:**

*. TBL_PERSON table has the following structure.

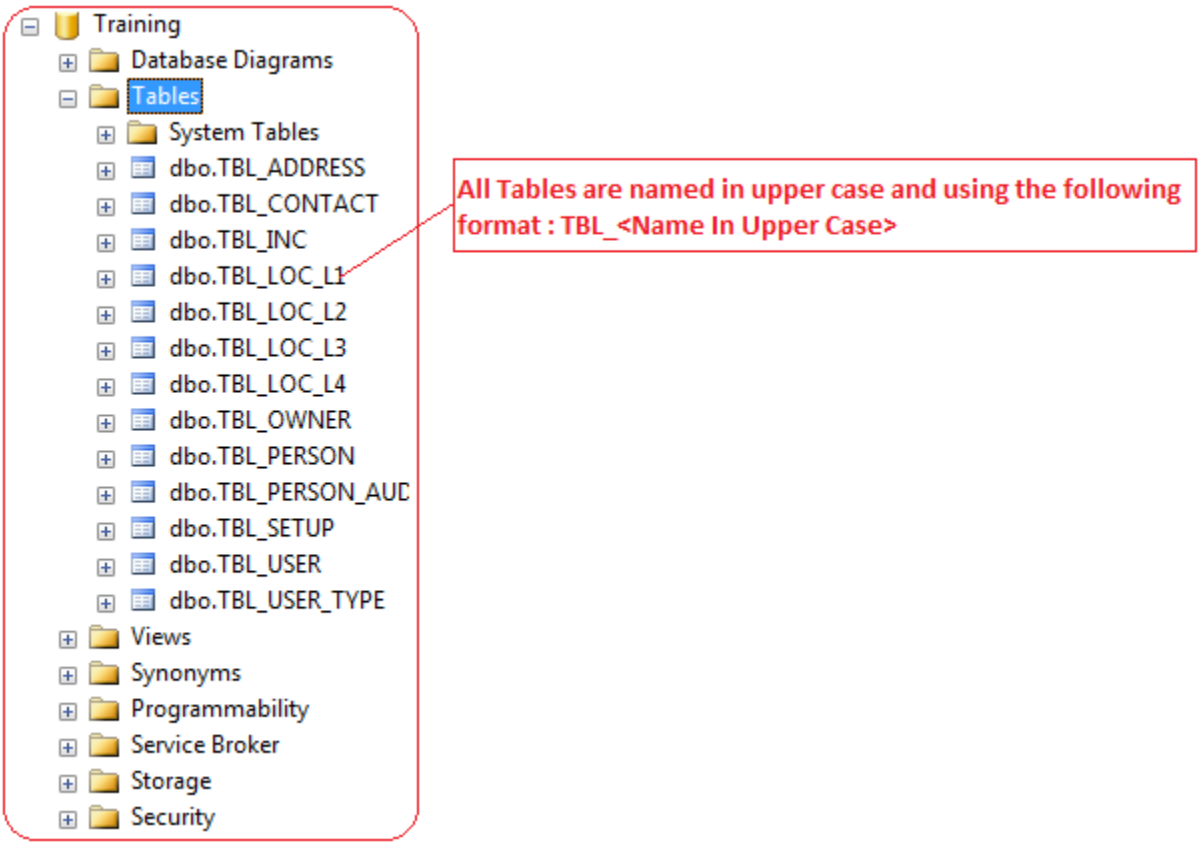


As you noticed that there is a convention while naming table and fields (and later stored procedures and user defined functions) as it will be explained in the next section.

Rule #1: Naming tables

We must name table as the following format **TBL_<name in upper case>**

Ex: In case we must create Person table, we have to name it as **TBL_PERSON**

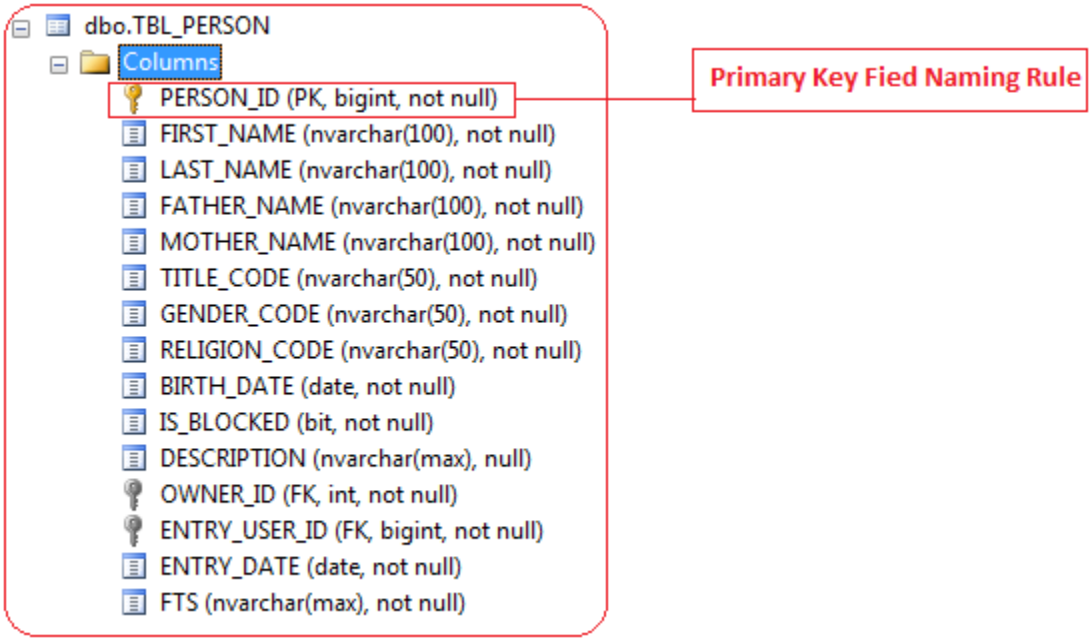


Rule #2: Naming the Primary Key Field

Each Table must have a primary key field named as the following:

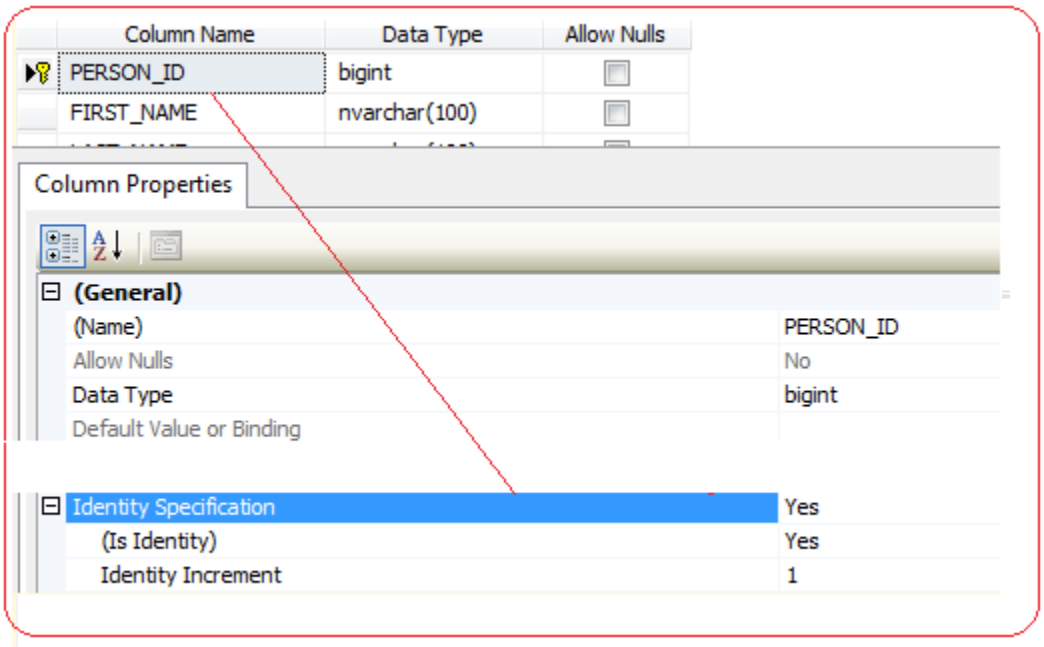
1. The Table name and remove the **TBL_** from it.
2. Add **_ID** at the end.

Ex: TBL_PERSON → PERSON_ID

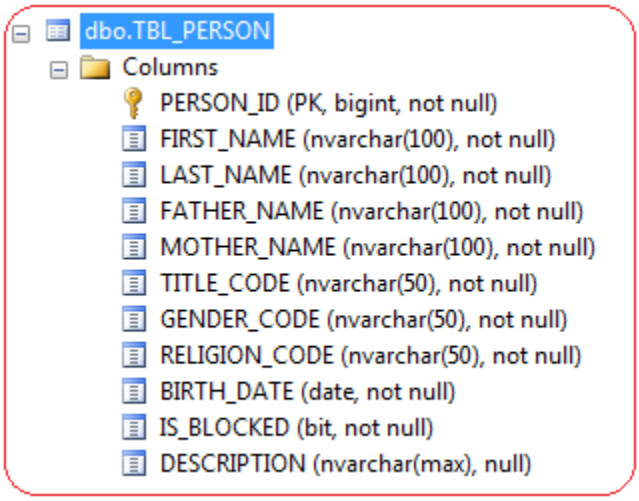


Rule #3: The Primary Key should be an identity field

The table primary key field should be an identity field.



Rule #4: Field name should be in upper case.

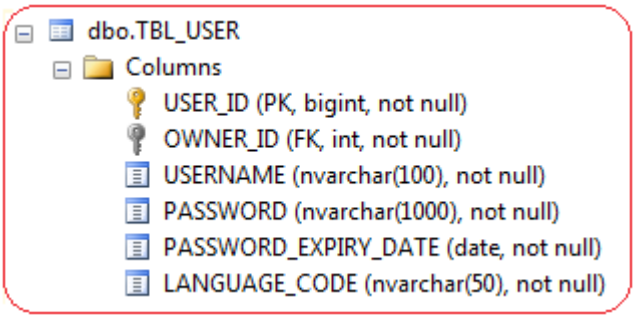


All fields are named in upper case.

Rule #5: Tables

The following Tables should exist:

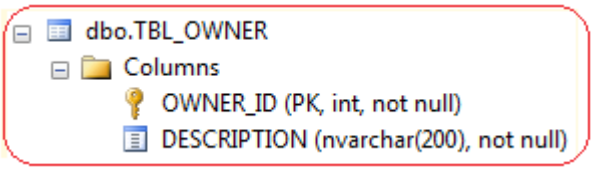
1. TBL_USER



N.B: You can disable this feature/ table by setting the following option

```
oCodeBooster.Is_InLine_Audit_Disabled = true;
```

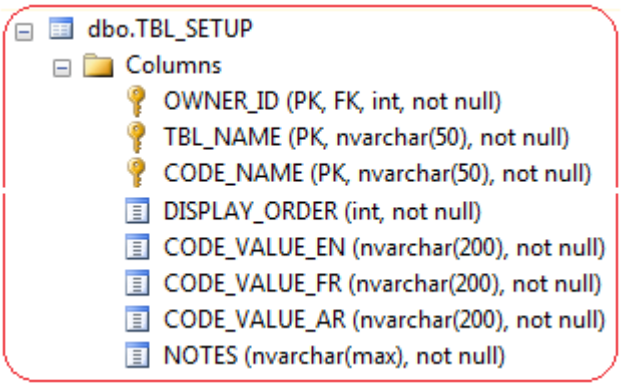
2. TBL_OWNER



N.B: You can disable this feature/ table by setting the following option

```
oCodeBooster.Is_SaaS_Disabled = true;
```

3. TBL_SETUP



N.B: You can disable this feature/table by setting the following option.

```
oCodeBooster.Is_Setup_Disabled = true;
```

***. Why there is a table called TBL_OWNER?**

As application development is heading to be SaaS (Software as a Service), you should be able to host in one database all possible clients (in our case Owners).

So we created this table TBL_OWNER in order to store all possible clients and in each table (check the TBL_PERSON table) you should have OWNER_ID field related to TBL_OWNER table so we will able to get records specific for a certain owner.

N.B: You can disable this table by setting the following option.

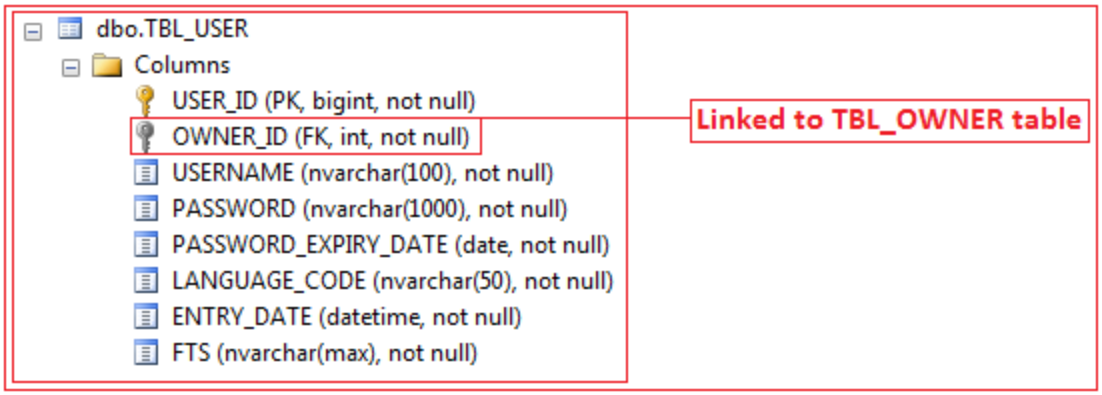
```
oCodeBooster.Is_SaaS_Disabled = true;
```

***. Why there is a table called TBL_USER?**

It goes without saying that any business application must have a list of users having privileges and related transactions; this is why we created this table. And in this table TBL_USER you will find OWNER_ID field, because users are related to a certain owner.

N.B: You can disable this feature/table by setting the following option.

```
oCodeBooster.Is_InLine_Audit_Disabled = true;
```



***. ENTRY_USER_ID & ENTRY_DATE fields exist in all tables.**

For auditing purposes, each table should have ENTRY_USER_ID & ENTRY_DATE fields to have the last user (and date) that has created (or modified) a certain record.

***. FTS field:** abbreviation of "Full text search".

By definition the full text search indexing lets users and applications run full-text queries against character-based data in SQL Server tables.

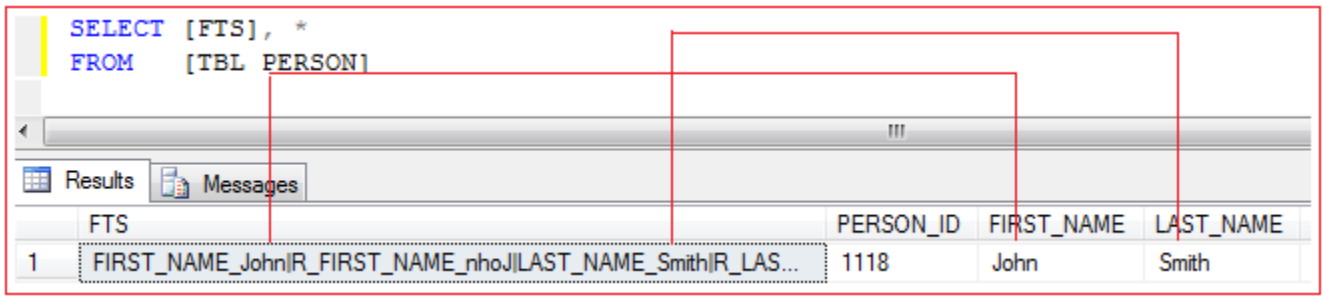
N.B: You can disable this feature by setting the following option.

```
oCodeBooster.Is_FTS_Disabled = true;
```

***. Ok deal, but why there is a field called FTS in each table?**

In order to not create full text search index [From now on we will call it FTS Index], we created only one field called FTS in the TBL_PERSON table and created a FTS index on it.

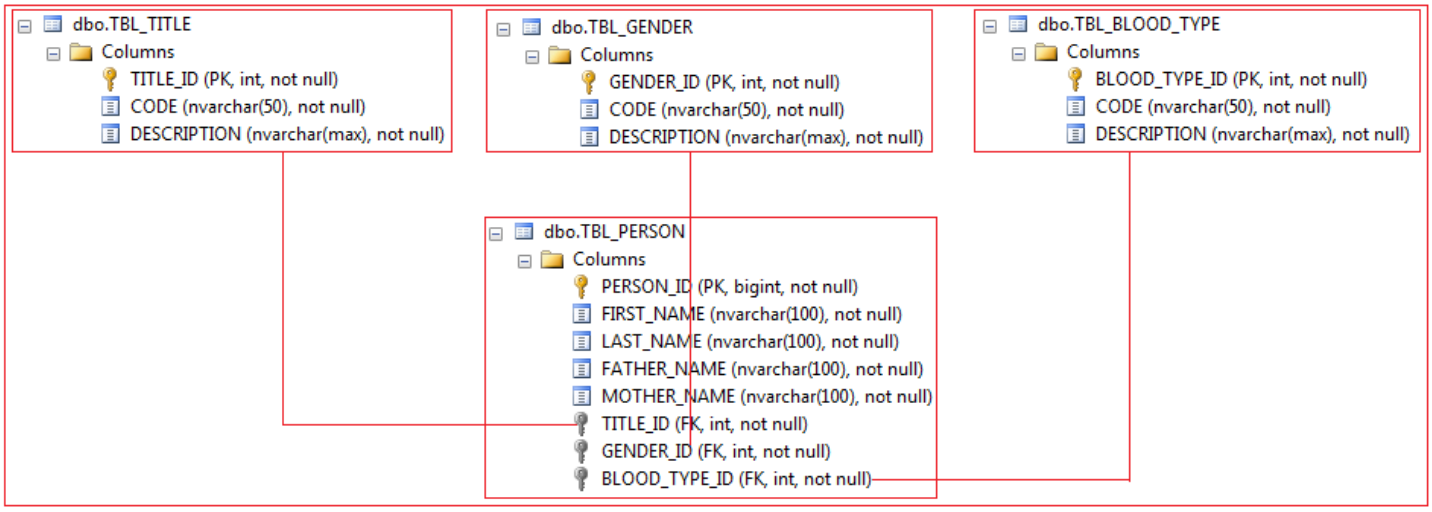
The value of this field is a kind of combination of the rest of field's values [check the following image]



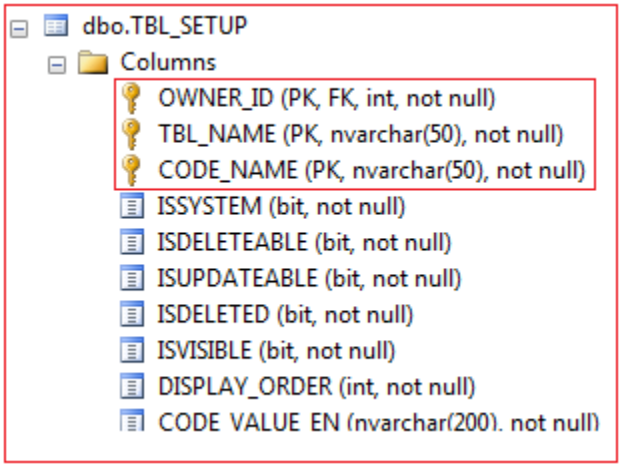
***. Why there is a table called TBL_SETUP**

Let's consider a scenario where we need to add to TBL_PERSON the Title, gender and blood type attributes.

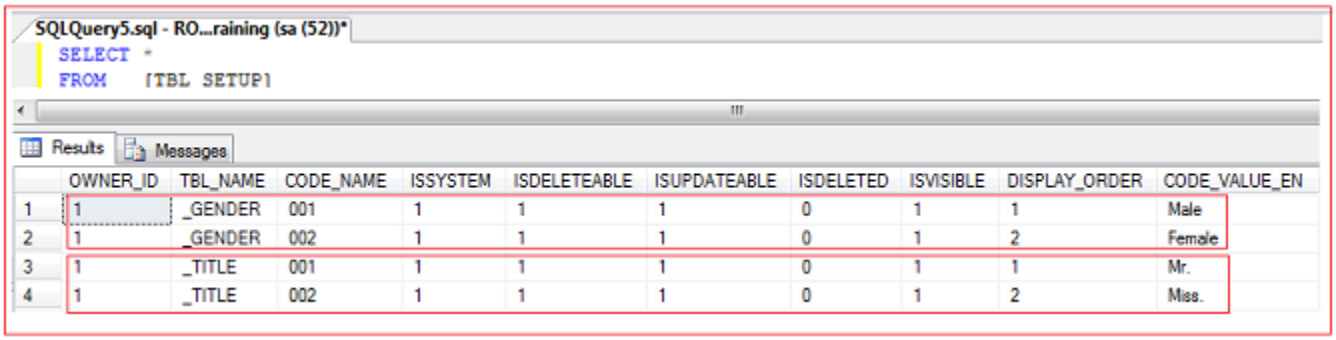
The traditional way is to create 3 tables as the following



As you noticed, the 3 tables are having the same structure and if you are working on middle-to-large scale applications you will face more than 100 such setup tables. So, as best practice, create one table called TBL_SETUP having the following structure.



And find below a sample of data where we created both Title and Gender setup tables in one place.

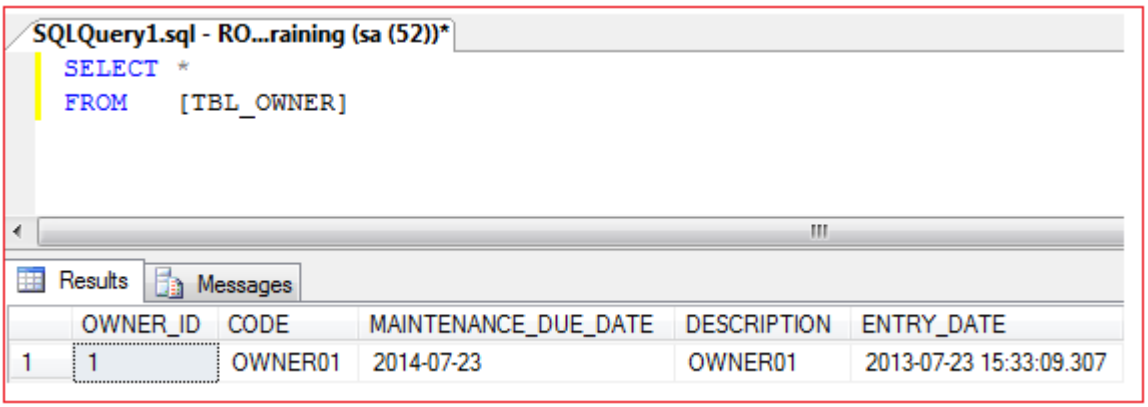


	OWNER_ID	TBL_NAME	CODE_NAME	ISSYSTEM	ISDELETEABLE	ISUPDATEABLE	ISDELETED	ISVISIBLE	DISPLAY_ORDER	CODE_VALUE_EN
1	1	_GENDER	001	1	1	1	0	1	1	Male
2	1	_GENDER	002	1	1	1	0	1	2	Female
3	1	_TITLE	001	1	1	1	0	1	1	Mr.
4	1	_TITLE	002	1	1	1	0	1	2	Miss.

***. Database initial data**

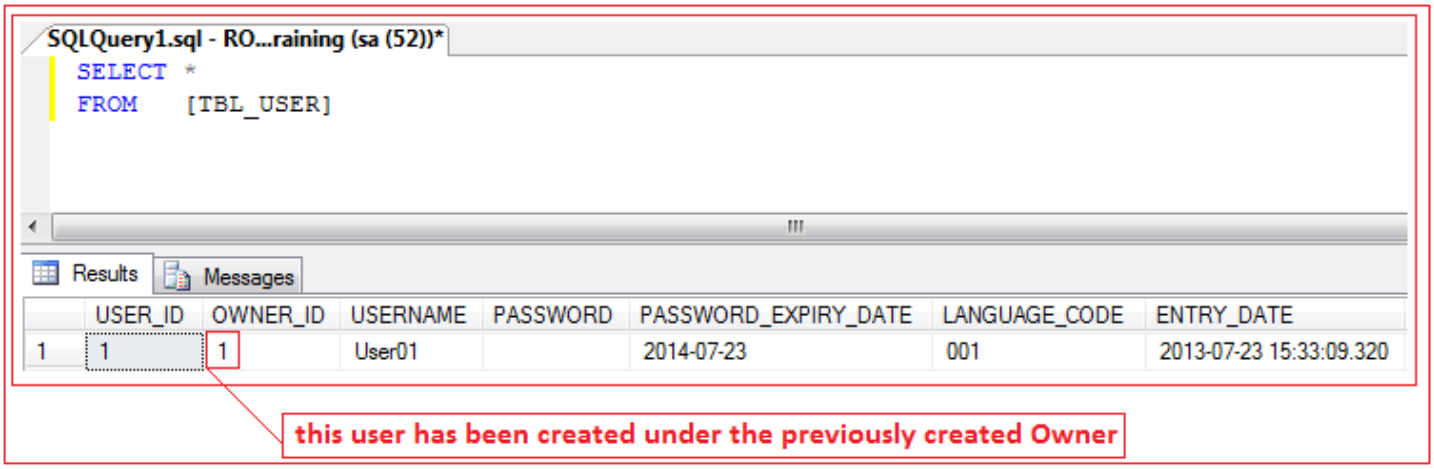
The code booster has not only created the database structure, but also created initial data in the following tables.

***. TBL_OWNER**



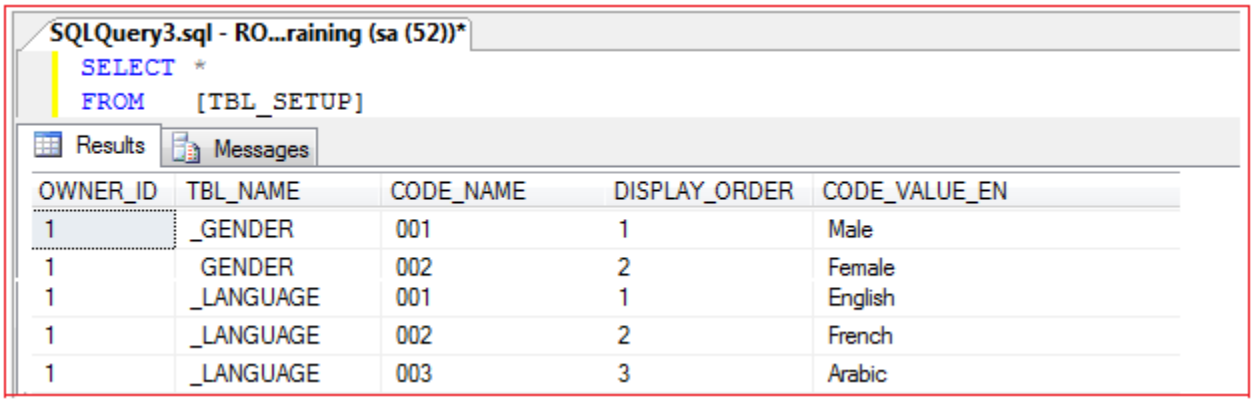
	OWNER_ID	CODE	MAINTENANCE_DUE_DATE	DESCRIPTION	ENTRY_DATE
1	1	OWNER01	2014-07-23	OWNER01	2013-07-23 15:33:09.307

***. TBL_USER**



	USER_ID	OWNER_ID	USERNAME	PASSWORD	PASSWORD_EXPIRY_DATE	LANGUAGE_CODE	ENTRY_DATE
1	1	1	User01		2014-07-23	001	2013-07-23 15:33:09.320

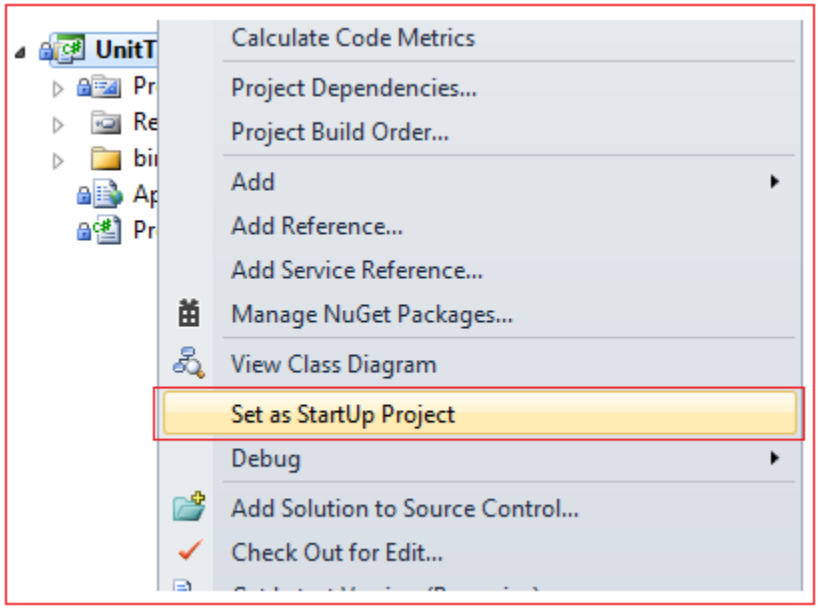
***. TBL_SETUP**



OWNER_ID	TBL_NAME	CODE_NAME	DISPLAY_ORDER	CODE_VALUE_EN
1	_GENDER	001	1	Male
1	_GENDER	002	2	Female
1	_LANGUAGE	001	1	English
1	_LANGUAGE	002	2	French
1	_LANGUAGE	003	3	Arabic

***. Prepare our testing environment (and specifically the UnitTesting console application)**

***. Set as startup project the UnitTesting console application (already added to the solution).**



***. At this point, you are ready to start using the code generated by Code Booster to manage your database in terms of creating, update, deleting & retrieving data.**

For every single table in your database, beside the fact that needed stored procedures were created in the database, the following methods will be created in BLC Layer (We will take **TBL_PERSON** as an example)

*. A class named **Person** will be created in BLC Layer (and specifically in **BLCEntities.cs** file) as the following.

```
#region Person
public partial class Person
{
    #region Properties
    public long? PERSON_ID { get; set; }
    public string FIRST_NAME { get; set; }
    public string LAST_NAME { get; set; }
    public string FATHER_NAME { get; set; }
    public string MOTHER_NAME { get; set; }
    public string TITLE_CODE { get; set; }
    public string GENDER_CODE { get; set; }
    public string RELIGION_CODE { get; set; }
    public string BIRTH_DATE { get; set; }
    public bool? IS_BLOCKED { get; set; }
    public string DESCRIPTION { get; set; }
    public Int32? OWNER_ID { get; set; }
    public long? ENTRY_USER_ID { get; set; }
    public string ENTRY_DATE { get; set; }
    #endregion
}
#endregion
```

*. **Edit_Person**: this method will be created in your BLC layer and specifically in **BLC_SimpleBehavior.cs** file as the following.

```
#region Edit_Person
public void Edit_Person(Person i_Person)
{
    Declaration And Initialization Section.
    PreEvent_General
    Body Section.
    PostEvent_General
}
#endregion
```

It accepts the **Person** object as parameter.

*. In case you want to create a new record in TBL_PERSON, prepare the Person object while setting PERSON_ID = -1 and then call Edit_Person method as the following

```
Person oPerson = new Person();
oTools.SetPropertiesDefaultValue(oPerson);
oPerson.PERSON_ID = -1;
oPerson.FIRST_NAME = "Peter";
oPerson.LAST_NAME = "Semaan";
oBLC.Edit_Person(oPerson);
```

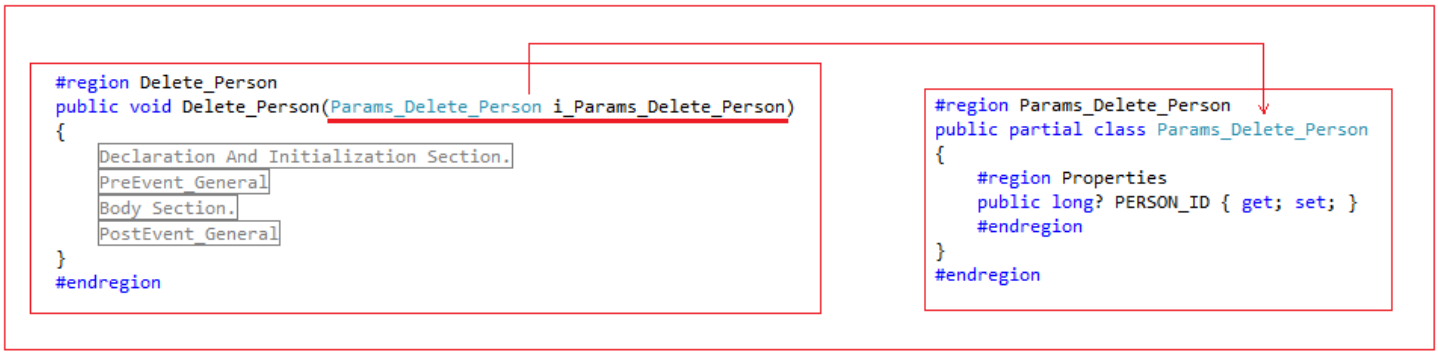
PERSON_ID = -1 to create a new record in TBL_PERSON

*. In case you want to update an already existing record in TBL_PERSON, just prepare the Person object with the corresponding PERSON_ID as the following

```
Person oPerson = new Person();
oTools.SetPropertiesDefaultValue(oPerson);
oPerson.PERSON_ID = 17;
oPerson.FIRST_NAME = "Peter - Updated";
oPerson.LAST_NAME = "Semaan - Updated";
oBLC.Edit_Person(oPerson);
```

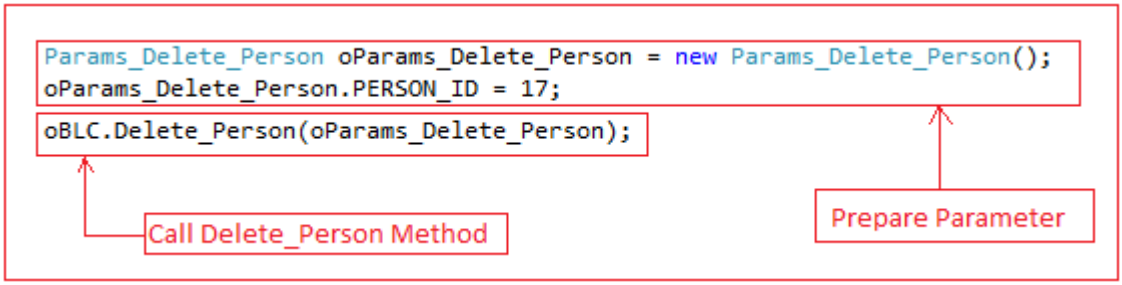
Update TBL_PERSON record having the primary key PERSON_ID = 17

***. Delete_Person:** this method will be created in your BLC layer and specifically in **BLC_SimpleBehavior.cs** file as the following.



As you noticed, the **Delete_Person** method accepts a complex type named **Params_Delete_Person** (this type is generated in BLCEntities.cs file) and it (the complex type) has one single property named PERSON_ID

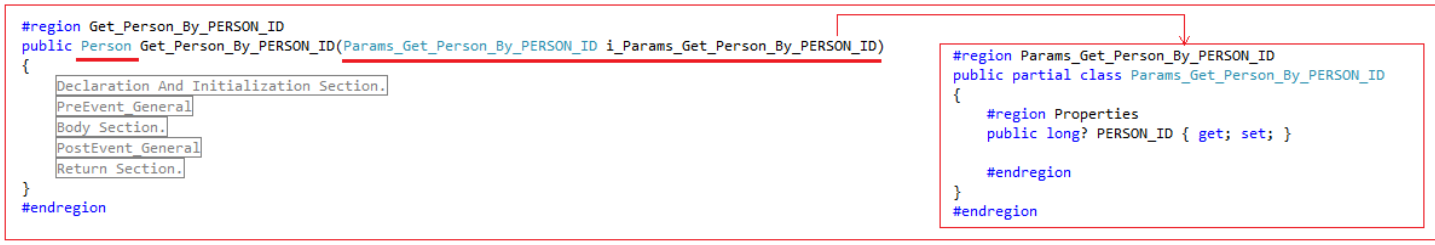
This method can be called as the following:



***. Get_Person_By_PERSON_ID:** This method will be created in your BLC layer and specifically in **BLC_SimpleBehavior.cs** file as the following.

```
#region Get_Person_By_PERSON_ID
public Person Get_Person_By_PERSON_ID(Params_Get_Person_By_PERSON_ID i_Params_Get_Person_By_PERSON_ID)
{
    Declaration And Initialization Section.
    PreEvent_General
    Body Section.
    PostEvent_General
    Return Section.
}
#endregion
```

As you noticed this method accepts a complex type named **Params_Get_Person_By_PERSON_ID** (this type is generated in BLCEntities.cs file) and it (the complex type) has one single property named PERSON_ID and it returns **Person** object



And this method can be called as the following.

```
Params_Get_Person_By_PERSON_ID oParams_Get_Person_By_PERSON_ID = new Params_Get_Person_By_PERSON_ID();
oParams_Get_Person_By_PERSON_ID.PERSON_ID = 17;
Person oPerson = oBLC.Get_Person_By_PERSON_ID(oParams_Get_Person_By_PERSON_ID);
```

*. **Get_Person_By_OWNER_ID**: This method will be created in your BLC layer and specifically in **BLC_SimpleBehavior.cs** file as the following.

```
#region Get_Person_By_OWNER_ID
public List<Person> Get_Person_By_OWNER_ID(Params_Get_Person_By_OWNER_ID i_Params_Get_Person_By_OWNER_ID)
{
    Declaration And Initialization Section.
    PreEvent_General
    Body Section.
    PostEvent_General
    Return Section.
}
#endregion
```

```
#region Params_Get_Person_By_OWNER_ID
public partial class Params_Get_Person_By_OWNER_ID
{
    #region Properties
    public Int32? OWNER_ID { get; set; }

    #endregion
}
#endregion
```

As you noticed this method accepts a complex type named **Params_Get_Person_By_OWNER_ID** (this type is generated in BLCEntities.cs file) and it (the complex type) has one single property named OWNER_ID and it returns a generic list of **Person** objects.

You can call this method as the following.

```
Params_Get_Person_By_OWNER_ID oParams_Get_Person_By_OWNER_ID = new Params_Get_Person_By_OWNER_ID();
oParams_Get_Person_By_OWNER_ID.OWNER_ID = 1;
List<Person> oList_Person = oBLC.Get_Person_By_OWNER_ID(oParams_Get_Person_By_OWNER_ID);
```

*. **Get_Person_By_Where**: This method will be created in your BLC layer and specifically in **BLC_SimpleBehavior.cs** file as the following.

```
#region Get_Person_By_Where
public List<Person> Get_Person_By_Where(Params_Get_Person_By_Where i_Params_Get_Person_By_Where)
{
    Declaration And Initialization Section.
    PreEvent_General
    Body Section.
    PostEvent_General
    Return Section.
}
#endregion
```

```
#region Params_Get_Person_By_Where
public partial class Params_Get_Person_By_Where
{
    #region Properties

    public Int32? OWNER_ID { get; set; }
    public string FIRST_NAME { get; set; }
    public string LAST_NAME { get; set; }
    public string FATHER_NAME { get; set; }
    public string MOTHER_NAME { get; set; }
    public string TITLE_CODE { get; set; }
    public string GENDER_CODE { get; set; }
    public string RELIGION_CODE { get; set; }
    public string DESCRIPTION { get; set; }
    public long? START_ROW { get; set; }
    public long? END_ROW { get; set; }
    public long? TOTAL_COUNT { get; set; }
    #endregion
}
#endregion
```

This method allows you to fetch the TBL_PERSON by any character field with the paging mechanism.

(By paging mechanism we meant the ability to retrieve a chunk of result instead of retrieving all data in one shot

As you do while doing google search where you have the click on a certain button to get next 20 results).

Just retrieve the first 10 records matching the criteria

```
Params_Get_Person_By_Where oParams_Get_Person_By_Where = new Params_Get_Person_By_Where();
oTools.SetPropertiesDefaultValue(oParams_Get_Person_By_Where);
oParams_Get_Person_By_Where.FIRST_NAME = "Peter";
oParams_Get_Person_By_Where.LAST_NAME = "Semaan";
oParams_Get_Person_By_Where.START_ROW = 0;
oParams_Get_Person_By_Where.END_ROW = 10;
List<Person> oList_Person = oBLC.Get_Person_By_Where(oParams_Get_Person_By_Where);
```

Fetching records in TBL_PERSON where FIRST_NAME contains "Peter" and LAST_NAME contains "Semaan"

*. **Get_Person_By_PERSON_ID_List**: This method will be created in your BLC layer and specifically in **BLC_SimpleBehavior.cs** file as the following.

```
#region Get_Person_By_PERSON_ID_List
public List<Person> Get_Person_By_PERSON_ID_List(Params_Get_Person_By_PERSON_ID_List i_Params_Get_Person_By_PERSON_ID_List)
{
    Declaration And Initialization Section.
    PreEvent_General
    Body Section.
    PostEvent_General
    Return Section.
}
#endregion
```

```
#region Params_Get_Person_By_PERSON_ID_List
public partial class Params_Get_Person_By_PERSON_ID_List
{
    #region Properties
    public List<long?> PERSON_ID_LIST { get; set; }
    #endregion
}
#endregion
```

As you noticed this method accepts a complex type named **Params_Get_Person_By_PERSON_ID_List** (this type is generated in BLCEntities.cs file) and it (the complex type) has one single property named **PERSON_ID_LIST** and it returns a generic list of **Person** objects.

You can call this method as the following.

```
Params_Get_Person_By_PERSON_ID_List oParams_Get_Person_By_PERSON_ID_List = new Params_Get_Person_By_PERSON_ID_List();
oParams_Get_Person_By_PERSON_ID_List.PERSON_ID_LIST = new List<long?>() { 1,47,58};
oList_Person = oBLC.Get_Person_By_PERSON_ID_List(oParams_Get_Person_By_PERSON_ID_List);
```

*. **Edit_Person_List**: This method will be created in your BLC layer and specifically in **BLC_SimpleBehavior.cs** file as the following

```
#region Edit_Person_List
public void Edit_Person_List(List<Person> i_List_Person)
{
    Declaration And Initialization Section.
    PreEvent_General
    Body Section.
    PostEvent_General
}
#endregion
```

This method accepts a collection of Persons, so in one shot you can Edit/Create a list of Persons as the following.

```
oList_Person = new List<Person>();
oList_Person.Add(new Person() { PERSON_ID = -1, FIRST_NAME = "Peter" , LAST_NAME = "Semaan" });
oList_Person.Add(new Person() { PERSON_ID = 45, FIRST_NAME = "Jhon - Updated", LAST_NAME = "Smith - Updated" });
oBLC.Edit_Person_List(oList_Person);
```

*. **What we talked about till now regarding the TBL_PERSON is applicable of all tables handled by Code booster**

For example, for TBL_ADDRESS you have the following BLC methods: Edit_Address, Delete_Address, Get_Address_By_ADDRESS_ID, Get_Address_By_OWNER_ID, Get_Address_By_Where

BUT, due to the fact that TBL_ADDRESS is linked to other tables, Code booster will be generated in addition the following methods

*. **Get_Address_By_PERSON_ID**: It returns a generic list of Address objects for a certain PERSON_ID as the following

```
Params_Get_Address_By_PERSON_ID oParams_Get_Address_By_PERSON_ID = new Params_Get_Address_By_PERSON_ID();
oParams_Get_Address_By_PERSON_ID.PERSON_ID = 17;
List<Address> oList_Address = oBLC.Get_Address_By_PERSON_ID(oParams_Get_Address_By_PERSON_ID);
```

*. **Get_Address_By_LOC_L1_ID**: It returns a generic list of Address objects for a certain LOC_L1_ID as the following

```
Params_Get_Address_By_LOC_L1_ID oParams_Get_Address_By_LOC_L1_ID = new Params_Get_Address_By_LOC_L1_ID();
oParams_Get_Address_By_LOC_L1_ID.LOC_L1_ID = 45;
List<Address> oList_Address = oBLC.Get_Address_By_LOC_L1_ID(oParams_Get_Address_By_LOC_L1_ID);
```

*. **Get_Address_By_Where_InList**: This method allows you to fetch TBL_ADDRESS table by any character field that exists in it plus the ability to search by a parent id (Ex: PERSON_ID)

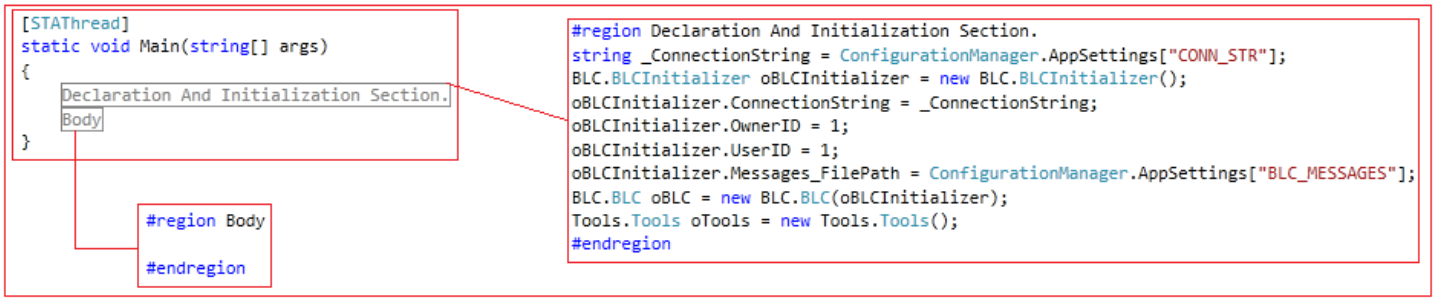
```
Params_Get_Address_By_Where_InList oParams_Get_Address_By_Where_InList = new Params_Get_Address_By_Where_InList();
oTools.SetPropertiesDefaultValue(oParams_Get_Address_By_Where_InList);
oParams_Get_Address_By_Where_InList.BUILDING = "My Building";
oParams_Get_Address_By_Where_InList.PERSON_ID_LIST = new List<long?>() { 17 };
List<Address> oList_Address = oBLC.Get_Address_By_Where_InList(oParams_Get_Address_By_Where_InList);
```

*. **_Adv methods**: Any get method (ex: Get_Address_By_ADDRESS_ID) has an “Advanced Version” named Get_XXX_By_YYY_Adv (Ex: Get_Address_By_ADDRESS_ID_Adv), it accepts that same parameter, but the result will be the joint with all One to One related tables as the following

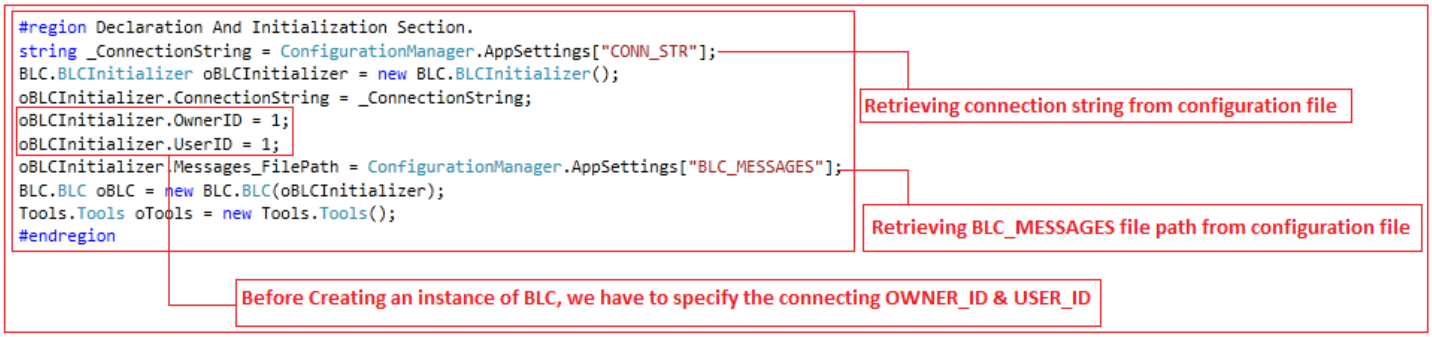
```
Params_Get_Address_By_PERSON_ID oParams_Get_Address_By_PERSON_ID = new Params_Get_Address_By_PERSON_ID();
oParams_Get_Address_By_PERSON_ID.PERSON_ID = 17;
List<Address> oList_Address = oBLC.Get_Address_By_PERSON_ID_Adv(oParams_Get_Address_By_PERSON_ID);
foreach (var item in oList_Address)
{
    Console.WriteLine(item.My_Person.FIRST_NAME);
    Console.WriteLine(item.My_Loc_l1.DESCRIPTION);
}
```

***. Let's create the first ever record in TBL_PERSON**

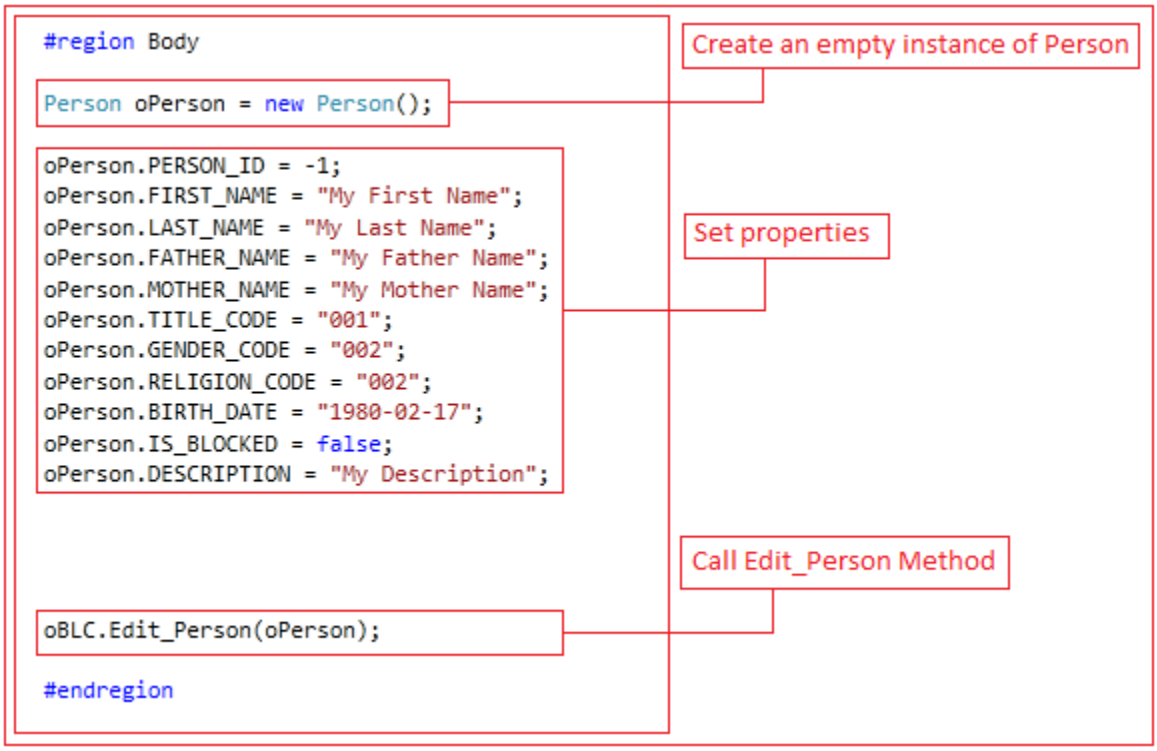
*. In program.cs file under the UnitTesting console application and specifically in the "Main" method you will find the following 2 regions.



*. In the "Declaration and Initialization section" we are initializing the BLC instance to be used in the next steps



***. Let us call the Edit_Person method to create a record in TBL_PERSON table**



***. We have 2 technical comments here:**

***.** We specified PERSON_ID = -1, in order to inform the Edit_Person method that we are creating a new entry.

```
oPerson.PERSON_ID = -1;
```

***.** BIRTH_DATE (which is a date field in the database) is set as string (property) with an ISO format.

YYYY-MM-DD to assure that the Sql Server will handle it without any kind of conflict related

To the regional settings or the installation process of the database engine itself

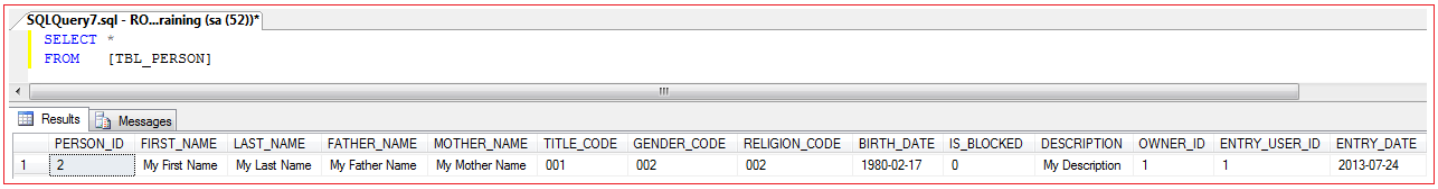
```
oPerson.BIRTH_DATE = "1980-02-17";
```

*. ENTRY_USER & OWNER_ID are retrieved from the oBLC instance (set in the declaration section as the following)

Speicfying the connected OWNED_ID & USER_ID

```
#region Declaration And Initialization Section.
string _ConnectionString = ConfigurationManager.AppSettings["CONN_STR"];
BLC.BLCInitializer oBLCInitializer = new BLC.BLCInitializer();
oBLCInitializer.ConnectionString = _ConnectionString;
oBLCInitializer.OwnerID = 1;
oBLCInitializer.UserID = 1;
oBLCInitializer.Messages_FilePath = ConfigurationManager.AppSettings["BLC_MESSAGES"];
BLC.BLC oBLC = new BLC.BLC(oBLCInitializer);
Tools.Tools oTools = new Tools.Tools();
#endregion
#region Body

#endregion
```

***. If we check the TBL_PERSON table, we find the following**



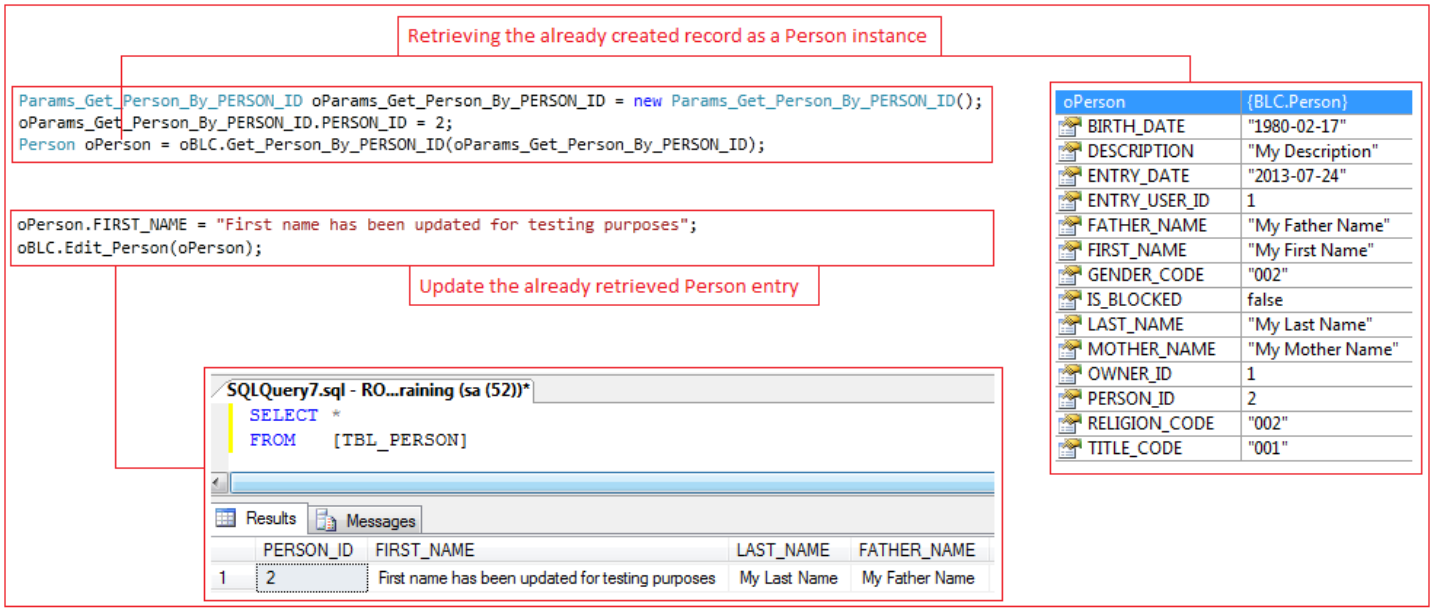
SQLQuery7.sql - RO...raining (sa (52))*

```
SELECT *
FROM   [TBL_PERSON]
```

Results | Messages

	PERSON_ID	FIRST_NAME	LAST_NAME	FATHER_NAME	MOTHER_NAME	TITLE_CODE	GENDER_CODE	RELIGION_CODE	BIRTH_DATE	IS_BLOCKED	DESCRIPTION	OWNER_ID	ENTRY_USER_ID	ENTRY_DATE
1	2	My First Name	My Last Name	My Father Name	My Mother Name	001	002	002	1980-02-17	0	My Description	1	1	2013-07-24

***. Let us retrieve the already created record in TBL_PERSON in order to update it.**



Retrieving the already created record as a Person instance

```
Params_Get_Person_By_PERSON_ID oParams_Get_Person_By_PERSON_ID = new Params_Get_Person_By_PERSON_ID();
oParams_Get_Person_By_PERSON_ID.PERSON_ID = 2;
Person oPerson = oBLC.Get_Person_By_PERSON_ID(oParams_Get_Person_By_PERSON_ID);
```

oPerson	{BLC.Person}
BIRTH_DATE	"1980-02-17"
DESCRIPTION	"My Description"
ENTRY_DATE	"2013-07-24"
ENTRY_USER_ID	1
FATHER_NAME	"My Father Name"
FIRST_NAME	"My First Name"
GENDER_CODE	"002"
IS_BLOCKED	false
LAST_NAME	"My Last Name"
MOTHER_NAME	"My Mother Name"
OWNER_ID	1
PERSON_ID	2
RELIGION_CODE	"002"
TITLE_CODE	"001"

```
oPerson.FIRST_NAME = "First name has been updated for testing purposes";
oBLC.Edit_Person(oPerson);
```

Update the already retrieved Person entry

SQLQuery7.sql - RO...raining (sa (52))*

```
SELECT *
FROM   [TBL_PERSON]
```

Results | Messages

	PERSON_ID	FIRST_NAME	LAST_NAME	FATHER_NAME
1	2	First name has been updated for testing purposes	My Last Name	My Father Name

***. Let's delete the already created record in TBL_PERSON**

```
Params_Delete_Person oParams_Delete_Person = new Params_Delete_Person();
oParams_Delete_Person.PERSON_ID = 2;
oBLC.Delete_Person(oParams_Delete_Person);
```

***. Let's fetch out the Tools.dll library.**

This Tools.dll is a custom library written by our team containing many useful methods that make the developer life easier.

For example, the following methods exist under Tools.dll library.

*. **GetDateString**

```
Person oPerson = new Person();
//oPerson.BIRTH_DATE = "2013-07-24";
oPerson.BIRTH_DATE = oTools.GetDateString(DateTime.Today);
```

This method returns a string representing date in ISO format

*. **SetPropertiesDefaultValue (Calling it, default value of each property will be set)**
Ex: Int →0 , String → "" , Boolean → false

```
Person oPerson         = new Person();
oPerson.PERSON_ID      = -1;
oPerson.FIRST_NAME     = "My First Name";
oPerson.LAST_NAME      = "";
oPerson.FATHER_NAME    = "";
oPerson.MOTHER_NAME    = "";
oPerson.TITLE_CODE     = "001";
oPerson.GENDER_CODE    = "001";
oPerson.RELIGION_CODE  = "001";
oPerson.BIRTH_DATE     = oTools.GetDateString(DateTime.Today);
oPerson.IS_BLOCKED     = false;
oPerson.DESCRIPTION    = "";

oBLC.Edit_Person(oPerson);
```

```
Person oPerson         = new Person();
oTools.SetPropertiesDefaultValue(oPerson);
oPerson.PERSON_ID      = -1;
oPerson.FIRST_NAME     = "My First Name";

oBLC.Edit_Person(oPerson);
```

Instead of setting default values for many properties, use SetPropertiesDefaultValue Method

***. Let us create 100 entries in TBL_PERSON**

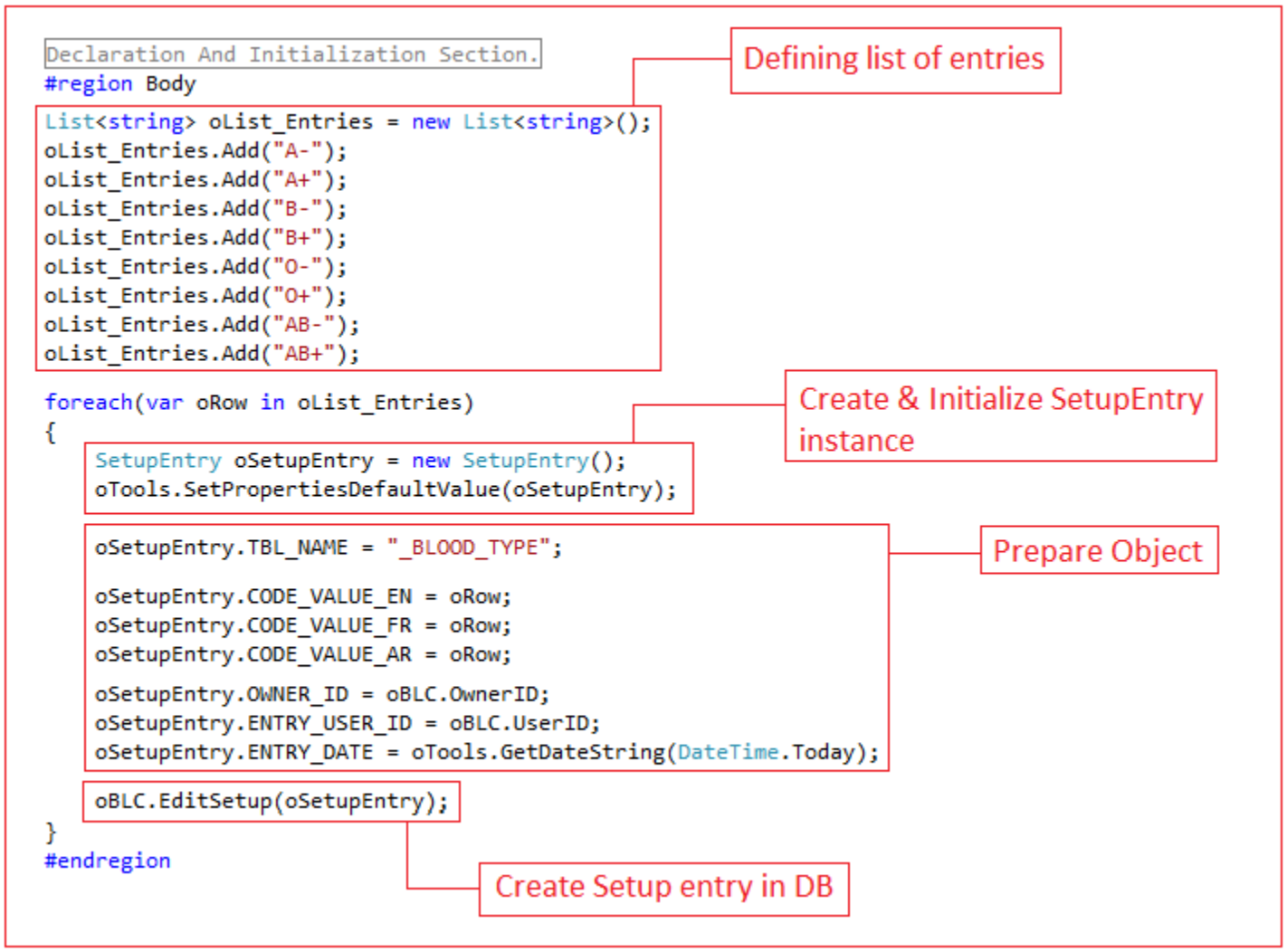
```
for (int i = 0; i <= 100; i++)
{
     Person oPerson        = new Person();
     oTools.SetPropertiesDefaultValue(oPerson);

     oPerson.PERSON_ID     = -1;
     oPerson.FIRST_NAME = string.Format("My first name is {0}",i.ToString());
     oPerson.TITLE_CODE = "001";
     oPerson.GENDER_CODE = "001";
     oPerson.RELIGION_CODE = "001";

     oBLC.Edit_Person(oPerson);
}
```

***. Let's delete all entries in TBL_PERSON**

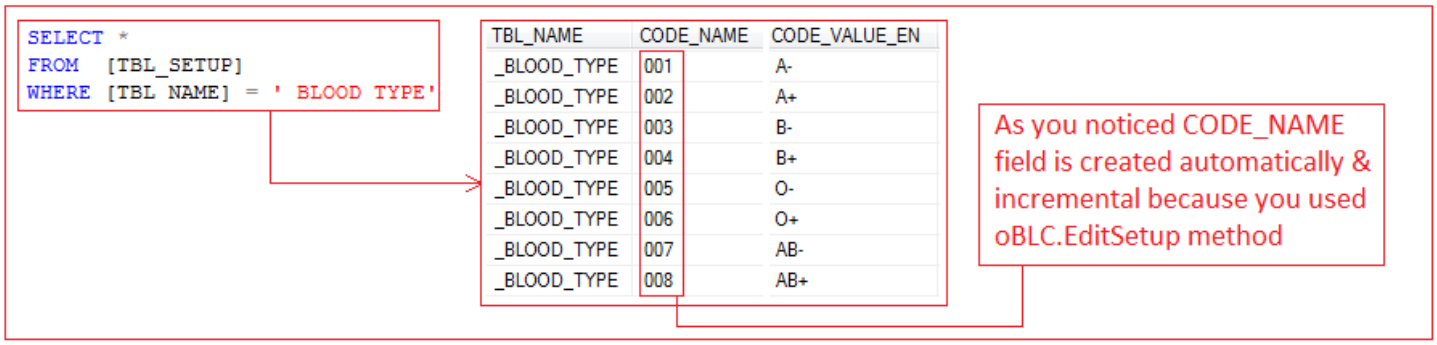
```
Params_Delete_Person_By_OWNER_ID oParams_Delete_Person_By_OWNER_ID = new Params_Delete_Person_By_OWNER_ID();
oParams_Delete_Person_By_OWNER_ID.OWNER_ID = 1;
oBLC.Delete_Person_By_OWNER_ID(oParams_Delete_Person_By_OWNER_ID);
```

***. Since TBL_SETUP table is a mandatory table in any business application you might develop, it is important to know how to create setup entries.**

In the unit testing console application and especially in the program.cs file, write the following code (below image) in order to create setup entries (in TBL_SETUP) for possible blood types (for ex)



And if you query the TBL_SETUP table, you will find the following data.



***. Using events to inject business rules.**

***.** Let us suppose that you have a business rule to prevent creating person record having a FIRST_NAME less than 4 characters.

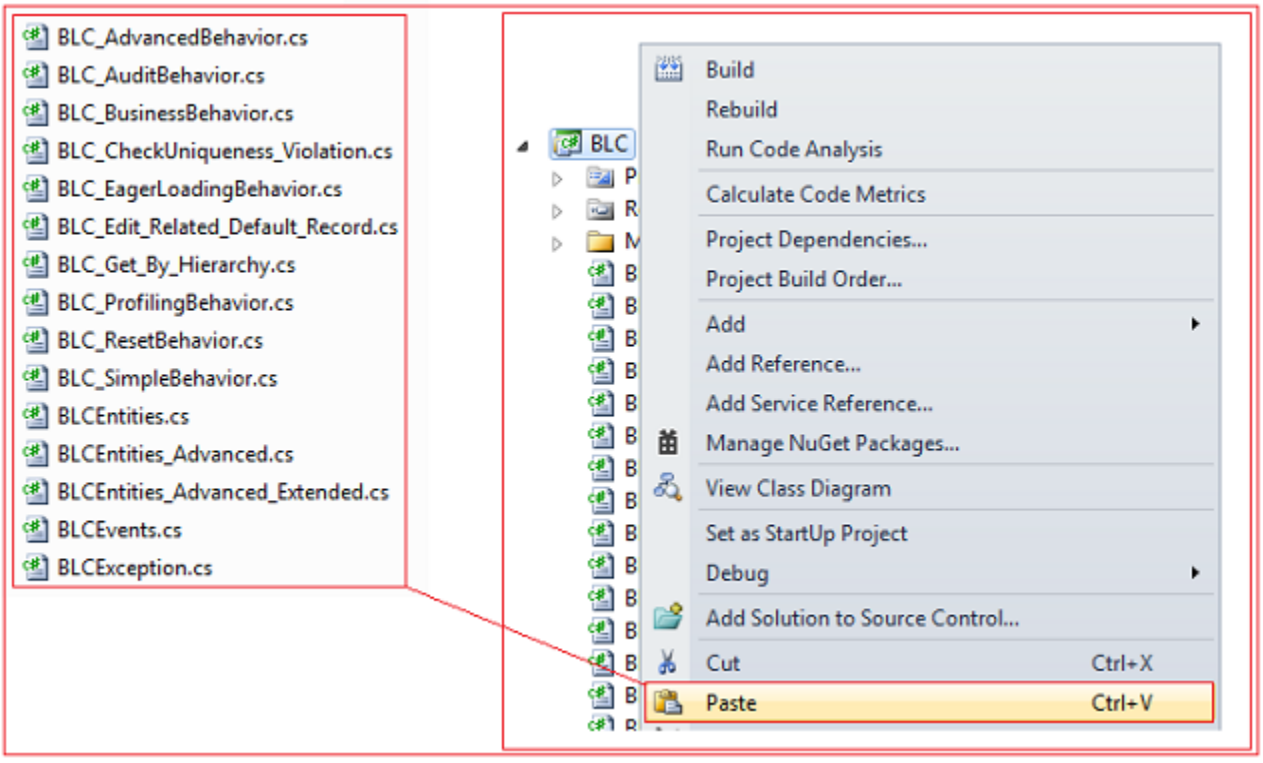
*. First of all, you ask the code booster engine to generate for Edit_Person method, pre- and post-events by writing the following in AppGenerator console application before calling option “001”.

```
static void Main(string[] args)
{
    Initialization & Authentication
    #region Events
    oCodeBooster.Methods_With_Events.Add("Edit_Person");
    #endregion
    Excluding Tables From 12M Hanlder
    #region Body Section
    Console.WriteLine("Enter An Option:");
    Console.WriteLine("001 --> Create SP's & BLC Layer");
    Console.WriteLine("002 --> Generate WCF / JSON Code");
    Console.WriteLine("003 --> Generate UI");

    string str_Option = Console.ReadLine();
    Options
    ByPassing Notification
    switch (str_Option)
    {
        #region case "001":
        case "001":
            oCodeBoosterClient.GenerateAllSPAndBLCLayer();
            break;
        #endregion
        case "002":
        case "003":
    }
    Console.WriteLine("Press Any Key To Exit");
    Console.ReadLine();
    #endregion
```

Informing the code booster engine to generate Pre & Post events for Edit_Person Method

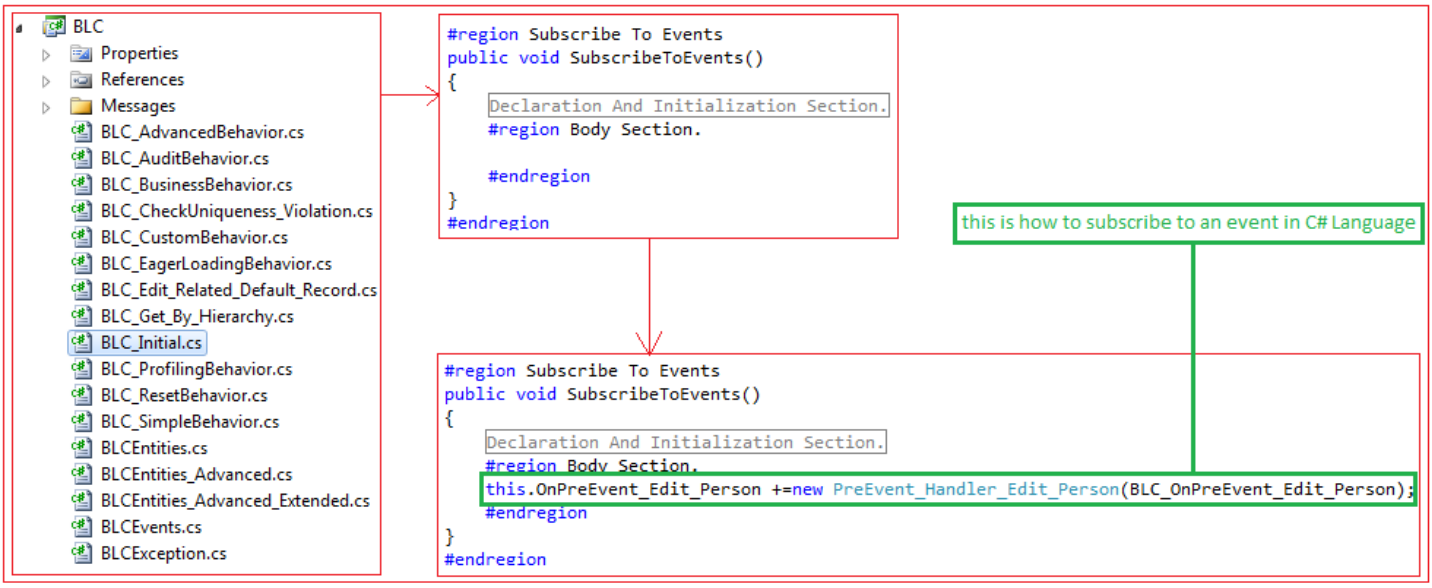
```
C:\Windows\system32\cmd.exe
001 --> Create SP's & BLC Layer
002 --> Generate WCF / JSON Code
003 --> Generate UI
001
Assure Required DB Objects existence : Start
Assure Required DB Objects existence : End
Assure Demo DB Objects Existence : Start
Assure Demo DB Objects Existence : End
Fetching DB Fields : Start
Fetching DB Fields : Done
Fetching DB Indexes : Start
Fetching DB Indexes : Done
Fetching DB FKs : Start
Fetching DB FKs : Done
Uploading DB Fields : Start
Uploading DB Fields : Done
Uploading DB Indexes : Start
Uploading DB Indexes : Done
Uploading FKs : Start
Uploading FKs : Done
Uploading Methods with Event : Start
Uploading Methods with Event : Done
Uploading WCF Methods : Start
Uploading WCF Methods : Done
Uploading Keys Mapper : Start
Uploading Keys Mapper : Done
Uploading BLC : Start
Uploading BLC : End
Downloading Patch : Patch-BLC-TSql-2013_7_25_8_47_50.rar : Start
Downloading Patch : Patch-BLC-TSql-2013_7_25_8_47_50.rar : End
Press Any Key To Exit
```

***. Override the content of the BLC folder (under the newly created patch) over the BLC Library project**



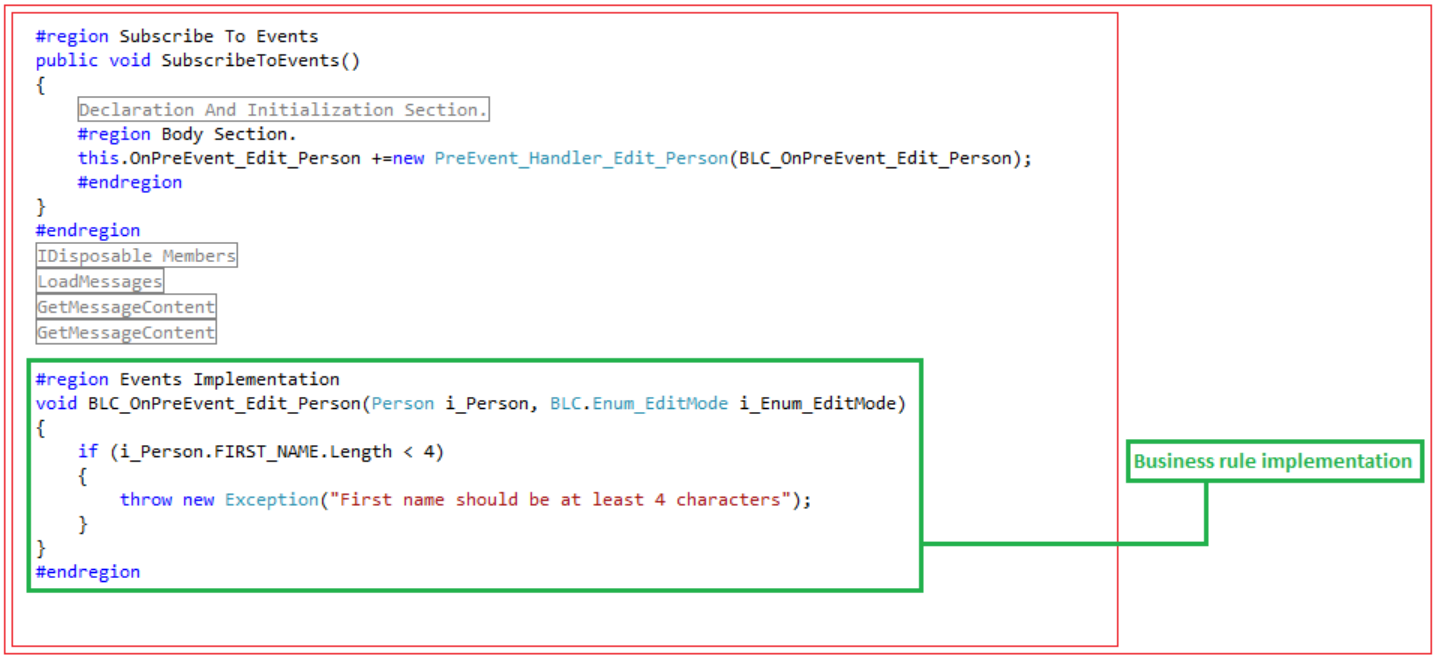
***. Let's now implement the business rule (FIRST_NAME length should not be less than 4 characters)**

*. In BLC_Initial.cs file and specifically in the SubscribeToEvents method you have to add a function to the onPreEvent_Edit_Person events pool by writing the following:



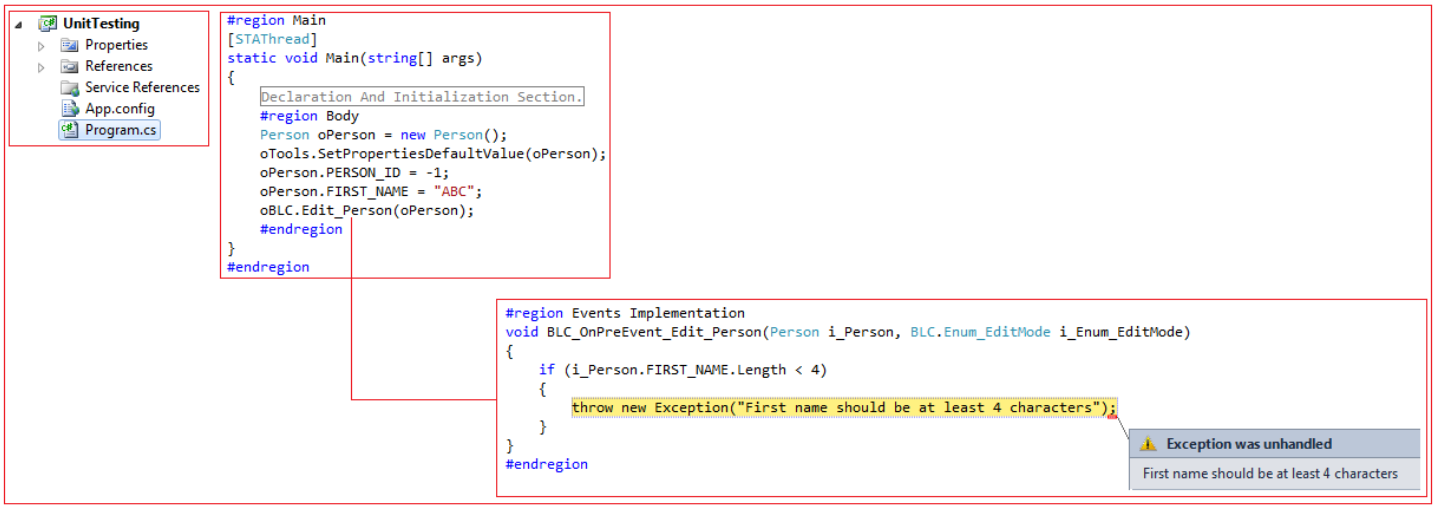
***. The last step is implementing the BLC_OnPreEvent_Edit_Person:**

In the same file (BLC_Initial.cs) and specifically in "Events Implemenation" region implement your function like the following



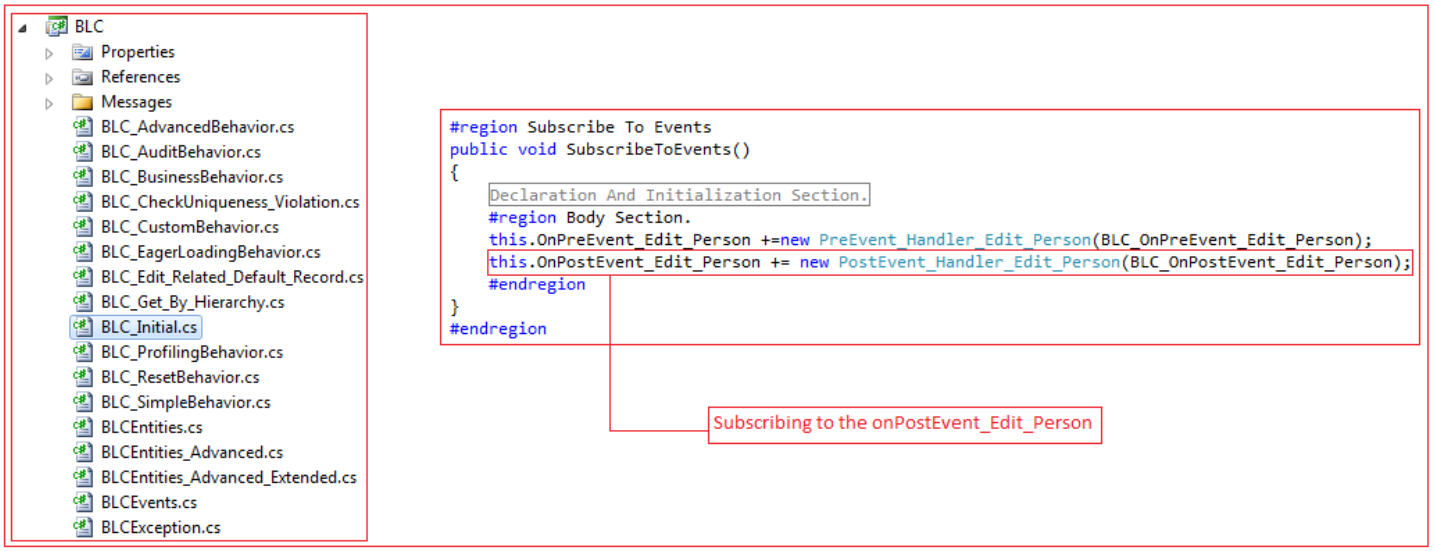
***. Let's try to create a person with FIRST_NAME less than 4 characters**

In the UnitTesting console application and specifically in the Main Method, try to create a person with FIRST_NAME less than 4 characters and you will notice that an exception will be thrown preventing you proceeding.



***. In case you have another business rule saying that after creating a new person, an email should be sent to the administrator.**

In the same file (BLC_Initial.cs) and specifically in "Events Implemenation" region implement your function like the following



***. Implementing the BLC_OnPostEvent_Edit_Person:**

In the same file (BLC_Initial.cs) and specifically in “Events Implemenation” region implement your function like the following

```
#region Subscribe To Events
public void SubscribeToEvents()
{
    Declaration And Initialization Section.
    #region Body Section.
    this.OnPreEvent_Edit_Person +=new PreEvent_Handler_Edit_Person(BLC_OnPreEvent_Edit_Person);
    this.OnPostEvent_Edit_Person += new PostEvent_Handler_Edit_Person(BLC_OnPostEvent_Edit_Person);
    #endregion
}
#endregion
IDisposable Members
LoadMessages
GetMessageContent
GetMessageContent
#region Events Implementation
void BLC_OnPreEvent_Edit_Person(Person i_Person, BLC.Enum_EditMode i_Enum_EditMode)...
void BLC_OnPostEvent_Edit_Person(Person i_Person, BLC.Enum_EditMode i_Enum_EditMode)
{
    // Send email to Administrator
}
#endregion
```

*. **As you noticed the easy way to inject business rules without affecting the caller method , in our case the UnitTesting console application has not been modified while we added as much as we want business rules**

```
#region Main
[STAThread]
static void Main(string[] args)
{
    Declaration And Initialization Section.
    #region Body
    Person oPerson = new Person();
    oTools.SetPropertiesDefaultValue(oPerson);
    oPerson.PERSON_ID = -1;
    oPerson.FIRST_NAME = "ABC";
    oBLC.Edit_Person(oPerson);
    #endregion
}
#endregion
```

The caller method has not been affected by any added business rule

*. **Let's add another business rule saying that once you created a person record , you should not be able to update his/her FIRST_NAME unless you are an administrator**

*. In the same file (BLC_Initial.cs) and specifically in "Events Implemenation" region implement your function like the following

As you noticed , you are able to add as much as you want business rules to Pre (or Post) events

```
#region Subscribe To Events
public void SubscribeToEvents()
{
    Declaration And Initialization Section.
    #region Body Section.
    this.OnPreEvent_Edit_Person +=new PreEvent_Handler_Edit_Person(BLC_OnPreEvent_Edit_Person);
    this.OnPostEvent_Edit_Person += new PostEvent_Handler_Edit_Person(BLC_OnPostEvent_Edit_Person);
    this.OnPreEvent_Edit_Person += new PreEvent_Handler_Edit_Person(BLC_OnPreEvent_Edit_Person_02);
    #endregion
}
#endregion
```

***. Implementing the BLC_OnPreEvent_Edit_Person_02:**

```
#region Subscribe To Events
public void SubscribeToEvents()
{
    Declaration And Initialization Section.
    #region Body Section.
    this.OnPreEvent_Edit_Person +=new PreEvent_Handler_Edit_Person(BLC_OnPreEvent_Edit_Person);
    this.OnPostEvent_Edit_Person += new PostEvent_Handler_Edit_Person(BLC_OnPostEvent_Edit_Person);
    this.OnPreEvent_Edit_Person += new PreEvent_Handler_Edit_Person(BLC_OnPreEvent_Edit_Person_02);
    #endregion
}
#endregion
IDisposable Members
LoadMessages
GetMessageContent
GetMessageContent
#region Events Implementation
void BLC_OnPreEvent_Edit_Person(Person i_Person, BLC.Enum_EditMode i_Enum_EditMode)...
void BLC_OnPreEvent_Edit_Person_02(Person i_Person, BLC.Enum_EditMode i_Enum_EditMode)
{
    switch(i_Enum_EditMode)
    {
        case Enum_EditMode.Add:
            break;
        case Enum_EditMode.Update:
            // if the connected user is not administrator throw an exception
            break;
    }
}
void BLC_OnPostEvent_Edit_Person(Person i_Person, BLC.Enum_EditMode i_Enum_EditMode)...
```

***. Differentiating between “Add” & “Update” Mode:**

As you noticed in the previous print screen, you can check if you are in “Add” or “Update” mode and take business decisions based on them.

```
void BLC_OnPreEvent_Edit_Person_02(Person i_Person, BLC.Enum_EditMode i_Enum_EditMode)
{
    switch(i_Enum_EditMode)
    {
        case Enum_EditMode.Add:
            break;
        case Enum_EditMode.Update:
            // if the connected user is not administrator throw an exception
            break;
    }
}
```

***. Let’s add another business rule saying that you cannot delete a person record unless you have administrative privileges.**

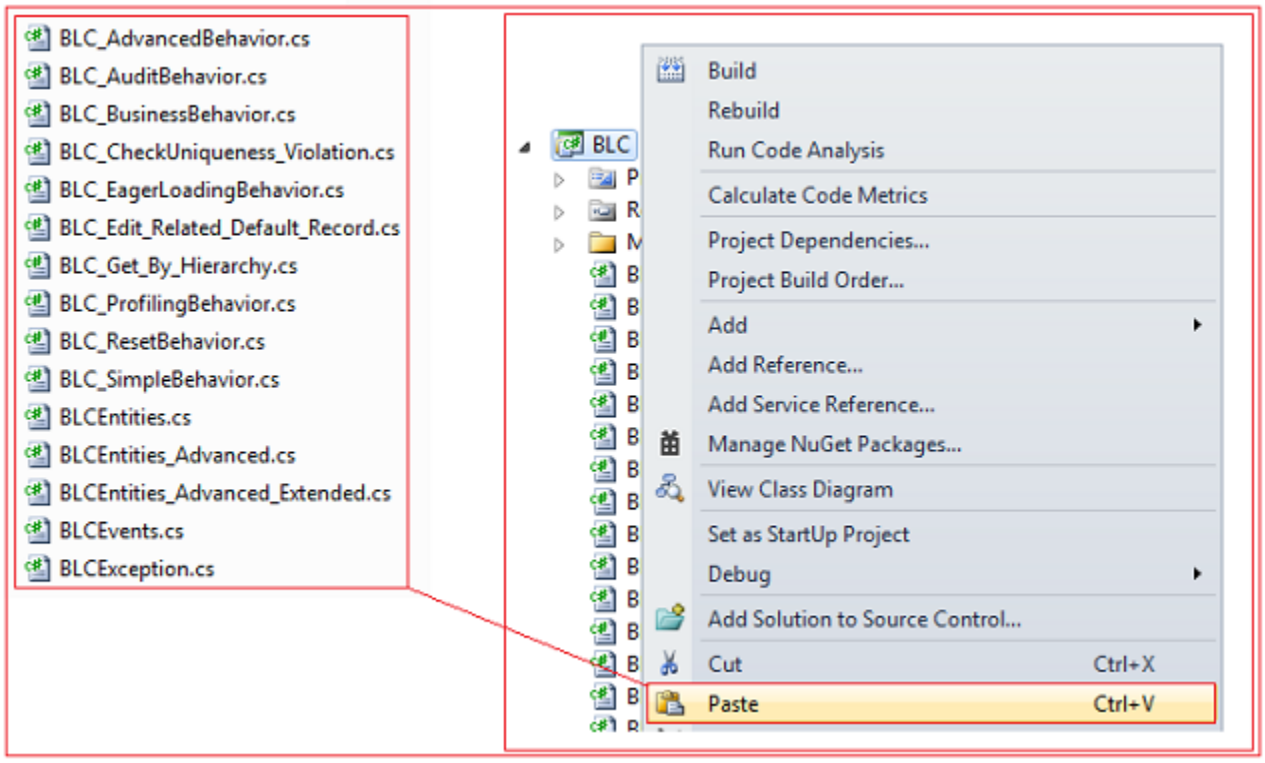
***. First, you ask the code booster engine to generate for Delete_Person method, pre- and post-events by writing the following in AppGenerator console application before calling option “001”.**

```
#region Main
static void Main(string[] args)
{
    Initialization & Authentication
    #region Events
    oCodeBooster.Methods_With_Events.Add("Edit_Person");
    oCodeBooster.Methods_With_Events.Add("Delete_Person");
    #endregion
    Excluding Tables From 12M Hanlder
    #region Body Section
    Console.WriteLine("Enter An Option:");
    Console.WriteLine("001 --> Create SP's & BLC Layer");
    Console.WriteLine("002 --> Generate WCF / JSON Code");
    Console.WriteLine("003 --> Generate UI");

    string str_Option = Console.ReadLine();
    Options
    ByPassing Notification
    switch (str_Option)
    {
        #region case "001":
        case "001":
            oCodeBoosterClient.GenerateAllSPAndBLCLayer();
            break;
        #endregion
        case "002":
        case "003":
    }
    Console.WriteLine("Press Any Key To Exit");
    Console.ReadLine();
```

Informing the code booster engine to generate Pre & Post events for Delete_Person Method

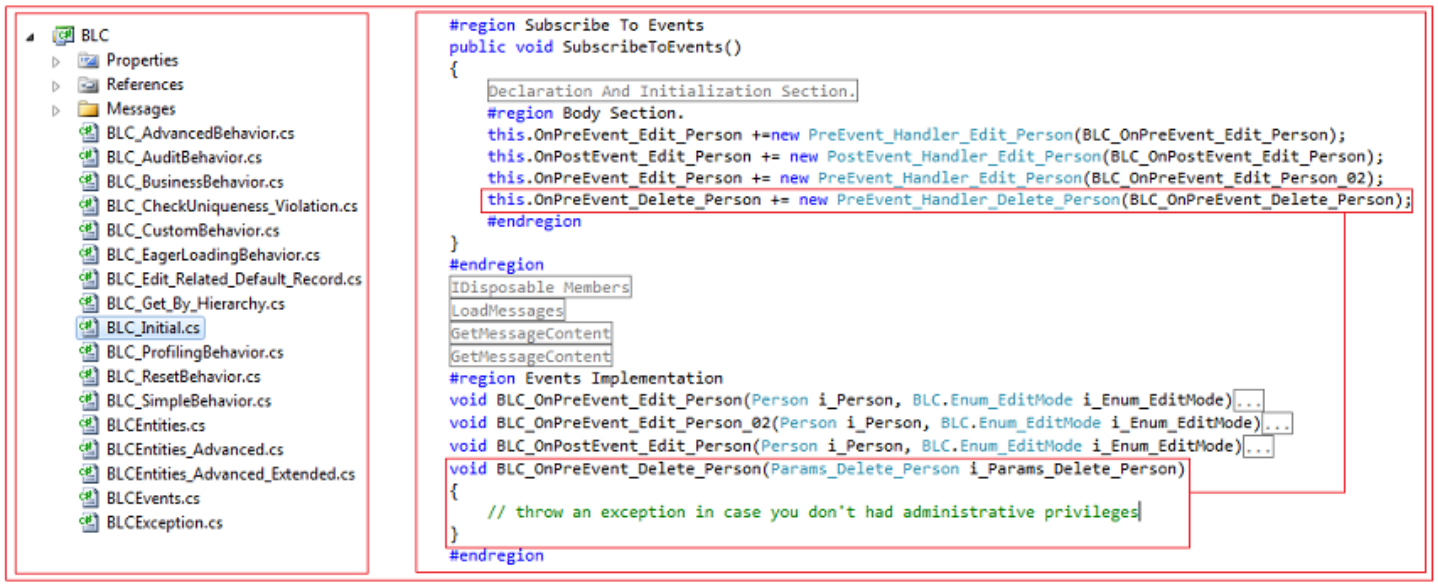
```
Enter An Option:
001 --> Create SP's & BLC Layer
002 --> Generate WCF / JSON Code
003 --> Generate UI
001
Assure Required DB Objects existence : Start
Assure Required DB Objects existence : End
Assure Demo DB Objects Existence : Start
Assure Demo DB Objects Existence : End
Fetching DB Fields : Start
Fetching DB Fields : Done
Fetching DB Indexes : Start
Fetching DB Indexes : Done
Fetching DB FKs : Start
Fetching DB FKs : Done
Uploading DB Fields : Start
Uploading DB Fields : Done
Uploading DB Indexes : Start
Uploading DB Indexes : Done
Uploading FKs : Start
Uploading FKs : Done
Uploading Methods with Event : Start
Uploading Methods with Event : Done
Uploading WCF Methods : Start
Uploading WCF Methods : Done
Uploading Keys Mapper : Start
Uploading Keys Mapper : Done
Uploading BLC : Start
Uploading BLC : End
Downloading Patch : Patch-BLC-TSql-2013_7_25_15_23_35.rar : Start
Downloading Patch : Patch-BLC-TSql-2013_7_25_15_23_35.rar : End
Press Any Key To Exit
```

***. Override the content of the BLC folder (under the newly created patch) over the BLC Library project**



***. Let's now implement our business rule (**you cannot delete a person record unless you have administrative privileges**)**

*. In BLC_Initial.cs file and specifically in the SubscribeToEvents method you have to add a function to the onPreEvent_Delete_Person events pool and implement it in "Events Implementation" region.



***. Let's add a new business rule saying that after deleting a person you have to send email to the system administrator containing the full information about the deleted person record.**

*. As we previously asked the code booster to generate events for "Delete_Person" method, both Pre & Post events were created and there is no need to re-generate a new patch.

***. Let's now implement our business rule**

*. In BLC_Initial.cs file and specifically in the SubscribeToEvents method you have to add a function to the onPostEvent_Delete_Person events pool and implement it in "Events Implementation" region.

-

```
public void SubscribeToEvents()
{
    Declaration And Initialization Section.
    #region Body Section.

    Initialize_Eager_Loading_Mechanism();
    Initialize_Reset_Mechanism();

    this.OnPreEvent_Edit_Person +=new PreEvent_Handler_Edit_Person(BLC_OnPreEvent_Edit_Person);
    this.OnPreEvent_Edit_Person += new PreEvent_Handler_Edit_Person(BLC_OnPreEvent_Edit_Person_02);
    this.OnPostEvent_Edit_Person += new PostEvent_Handler_Edit_Person(BLC_OnPostEvent_Edit_Person);
    this.OnPostEvent_Delete_Person += new PostEvent_Handler_Delete_Person(BLC_OnPostEvent_Delete_Person);

    this.OnPreEvent_Delete_Person += new PreEvent_Handler_Delete_Person(BLC_OnPreEvent_Delete_Person);
    #endregion
}
#endregion
IDisposable Members
LoadMessages
GetMessageContent
GetMessageContent
#region Events Implementation
void BLC_OnPreEvent_Edit_Person(Person i_Person, BLC.Enum_EditMode i_Enum_EditMode)...
void BLC_OnPreEvent_Edit_Person_02(Person i_Person, BLC.Enum_EditMode i_Enum_EditMode)...
void BLC_OnPostEvent_Edit_Person(Person i_Person, BLC.Enum_EditMode i_Enum_EditMode)...
void BLC_OnPreEvent_Delete_Person(Params_Delete_Person i_Params_Delete_Person)...
void BLC_OnPostEvent_Delete_Person(Params_Delete_Person i_Params_Delete_Person)
{
    // For testing purposes we will not send an email
    // but printing out (on console) the Person.FIRST_NAME & Person.LAST_NAME properties

    // We are able to access the properties of the deleted record (EVEN AFTER DELETION) using _Person instance
    Console.WriteLine(_Person.FIRST_NAME);
    Console.WriteLine(_Person.LAST_NAME);
}
#endregion
```

We are able to access the properties (Even after deletion) of the corresponding record

As you noticed , we were able to access the Person properties even after record deletion by using _Person instance

***. Let's create a new business rule saying that you have to retrieve persons sorted by the FIRST_NAME field.**

*. In program.cs file under the UnitTesting console application and specifically in the "Main" method let us delete existing entries in TBL_PERSON Table and then create 4 entries as the following

```
static void Main(string[] args)
{
    Declaration And Initialization Section.
    #region Body

    Params_Delete_Person_By_OWNER_ID oParams_Delete_Person_By_OWNER_ID = new Params_Delete_Person_By_OWNER_ID();
    oParams_Delete_Person_By_OWNER_ID.OWNER_ID = 1;
    oBLC.Delete_Person_By_OWNER_ID(oParams_Delete_Person_By_OWNER_ID);

    Person oPerson = null;

    oPerson = new Person();
    oTools.SetPropertiesDefaultValue(oPerson);
    oPerson.PERSON_ID = -1;
    oPerson.FIRST_NAME = "X First Name";
    oBLC.Edit_Person(oPerson);

    oPerson = new Person();
    oTools.SetPropertiesDefaultValue(oPerson);
    oPerson.PERSON_ID = -1;
    oPerson.FIRST_NAME = "C First Name";
    oBLC.Edit_Person(oPerson);

    oPerson = new Person();
    oTools.SetPropertiesDefaultValue(oPerson);
    oPerson.PERSON_ID = -1;
    oPerson.FIRST_NAME = "A First Name";
    oBLC.Edit_Person(oPerson);

    oPerson = new Person();
    oTools.SetPropertiesDefaultValue(oPerson);
    oPerson.PERSON_ID = -1;
    oPerson.FIRST_NAME = "B First Name";
    oBLC.Edit_Person(oPerson);

    #endregion
}
#endregion
```

Deleting all entries in TBL_PERSON table

Creating 4 entries in TBL_PERSON Table

```
SELECT *
FROM   [TBL_PERSON]
```

Results | Messages

	PERSON_ID	FIRST_NAME	LAST_NAME	FATHER_NAME	MOTHER_NAME
1	1114	X First Name			
2	1115	C First Name			
3	1116	A First Name			
4	1117	B First Name			

*. In order to retrieve all records in TBL_PERSON table, we must call Get_Person_By_OWNER_ID as the following

```
Params_Get_Person_By_OWNER_ID oParams_Get_Person_By_OWNER_ID = new Params_Get_Person_By_OWNER_ID();
oParams_Get_Person_By_OWNER_ID.OWNER_ID = 1;
List<Person> oList_Person = oBLC.Get_Person_By_OWNER_ID(oParams_Get_Person_By_OWNER_ID);
```

```
foreach (var oRow_Person in oList_Person)
{
    Console.WriteLine(oRow_Person.FIRST_NAME);
}
```

```
C:\Windows\system32\cmd.exe
X First Name
C First Name
A First Name
B First Name
Press any key to continue . . .
```

As you noticed, by default records are retrieved by the data entry order (**Not ordered by FIRST_NAME field**)

*. In order to sort records retrieved by FIRST_NAME, let's create a Post event for Get_Person_By_OWNER_ID method where we will sort records as we want.

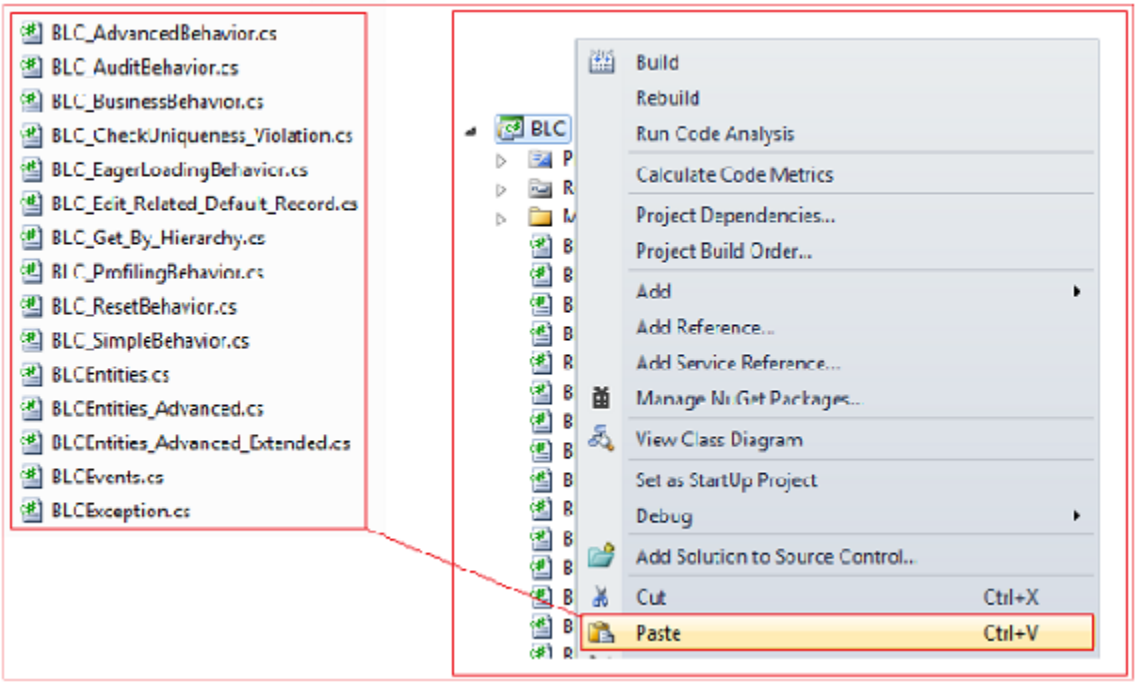
*. Ask the code booster engine to generate for Get_Person_By_OWNER_ID method, pre- and post-events by writing the following in AppGenerator console application before calling option "001".

```
static void Main(string[] args)
{
    Initialization & Authentication
    #region Events
    oCodeBooster.Methods_With_Events.Add("Edit_Person");
    oCodeBooster.Methods_With_Events.Add("Delete_Person");
    oCodeBooster.Methods_With_Events.Add("Get_Person_By_OWNER_ID");
    #endregion
    Excluding Tables From 12M Hanlder
    EagerLoading

    ResetTopology
    #region Body Section
    Console.WriteLine("Enter An Option:");
    Console.WriteLine("001 --> Create SP's & BLC Layer");
    Console.WriteLine("002 --> Generate WCF / JSON Code");
    Console.WriteLine("003 --> Generate UI");

    string str_Option = Console.ReadLine();
    Options
    ByPassing Notification
    switch (str_Option)
    {
        #region case "001":
        case "001":
            oCodeBoosterClient.GenerateAllSPAndBLCLayer();
            break;
        #endregion
        case "002":
        case "003":
    }
    Console.WriteLine("Press Any Key To Exit");
    Console.ReadLine();
    #endregion
```

Informing the code booster to generate Pre & Post events for Get_Person_By_OWNER_ID method

```
L1
Enter An Option:
001 --> Create SP's & BLC Layer
002 --> Generate WCF / JSON Code
003 --> Generate UI
001
Assure Required DB Objects existence : Start
Assure Required DB Objects existence : End
Assure Demo DB Objects Existence : Start
Assure Demo DB Objects Existence : End
Fetching DB Fields : Start
Fetching DB Fields : Done
Fetching DB Indexes : Start
Fetching DB Indexes : Done
Fetching DB FKs : Start
Fetching DB FKs : Done
Uploading DB Fields : Start
Uploading DB Fields : Done
Uploading DB Indexes : Start
Uploading DB Indexes : Done
Uploading FKs : Start
Uploading FKs : Done
Uploading Methods with Event : Start
Uploading Methods with Event : Done
Uploading WCF Methods : Start
Uploading WCF Methods : Done
Uploading Keys Mapper : Start
Uploading Keys Mapper : Done
Uploading BLC : Start
Uploading BLC : End
Downloading Patch : Patch-BLC-TSql-2013_7_29_17_16_41.rar
Downloading Patch : Patch-BLC-TSql-2013_7_29_17_16_41.rar
```

***. Override the content of the BLC folder (under the newly created patch) over the BLC Library project**



***. Let's now implement our business rule (**Sorting person records by FIRST_NAME**)**

*. In BLC_Initial.cs file and specifically in the SubscribeToEvents method you have to add a function to the onPostEvent_Get_Person_By_OWNER_ID events pool and implement it in "Events Implementation" region

```
#region Subscribe To Events
public void SubscribeToEvents()
{
    Declaration And Initialization Section.
    #region Body Section.

    Initialize_Eager_Loading_Mechanism();
    Initialize_Reset_Mechanism();

    this.OnPreEvent_Edit_Person +=new PreEvent_Handler_Edit_Person(BLC_OnPreEvent_Edit_Person);
    this.OnPreEvent_Edit_Person += new PreEvent_Handler_Edit_Person(BLC_OnPreEvent_Edit_Person_02);
    this.OnPostEvent_Edit_Person += new PostEvent_Handler_Edit_Person(BLC_OnPostEvent_Edit_Person);
    this.OnPostEvent_Delete_Person += new PostEvent_Handler_Delete_Person(BLC_OnPostEvent_Delete_Person);
    this.OnPreEvent_Delete_Person += new PreEvent_Handler_Delete_Person(BLC_OnPreEvent_Delete_Person);

    this.OnPostEvent_Get_Person_By_OWNER_ID += new PostEvent_Handler_Get_Person_By_OWNER_ID(BLC_OnPostEvent_Get_Person_By_OWNER_ID);
    #endregion
}

#endregion
IDisposable Members
LoadMessages
GetMessageContent
GetMessageContent
#region Events Implementation
void BLC_OnPreEvent_Edit_Person(Person i_Person, BLC.Enum_EditMode i_Enum_EditMode)...
void BLC_OnPreEvent_Edit_Person_02(Person i_Person, BLC.Enum_EditMode i_Enum_EditMode)...
void BLC_OnPostEvent_Edit_Person(Person i_Person, BLC.Enum_EditMode i_Enum_EditMode)...
void BLC_OnPreEvent_Delete_Person(Params_Delete_Person i_Params_Delete_Person)...
void BLC_OnPostEvent_Delete_Person(Params_Delete_Person i_Params_Delete_Person)...
void BLC_OnPostEvent_Get_Person_By_OWNER_ID(List<Person> i_Result, Params_Get_Person_By_OWNER_ID i_Params_Get_Person_By_OWNER_ID)
{
    if (i_Result != null)
    {
        i_Result = i_Result.OrderBy(x => x.FIRST_NAME).ToList();
    }
}
#endregion
```

Sorting the returned collection by FIRST_NAME field

*. In the UnitTesting console application and specifically in the "Main" method let us call Get_Person_By_OWNER_ID to check how records are sorted

```
Params_Get_Person_By_OWNER_ID oParams_Get_Person_By_OWNER_ID = new Params_Get_Person_By_OWNER_ID();
oParams_Get_Person_By_OWNER_ID.OWNER_ID = 1;
List<Person> oList_Person = oBLC.Get_Person_By_OWNER_ID(oParams_Get_Person_By_OWNER_ID);
```

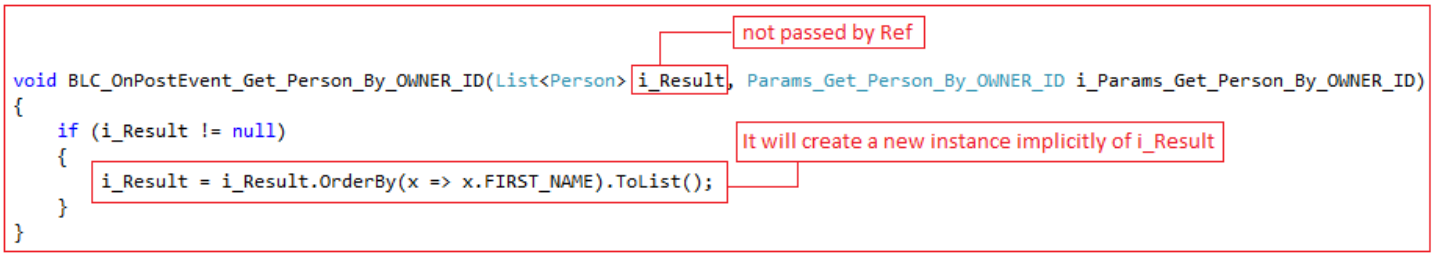
```
foreach (var oRow_Person in oList_Person)
{
    Console.WriteLine(oRow_Person.FIRST_NAME);
}
```

```
C:\Windows\system32\cmd.exe
X First Name
C First Name
A First Name
B First Name
Press any key to continue . . .
```

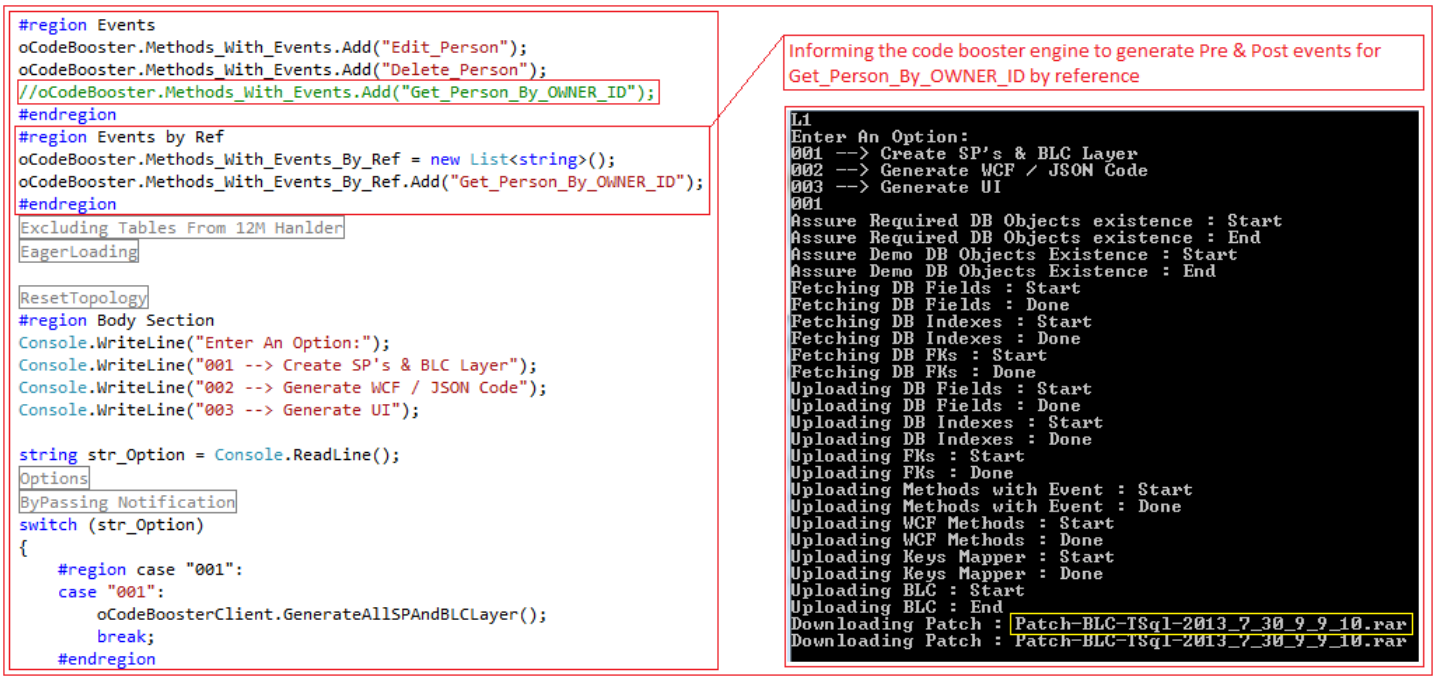
But records are not sorted by FIRST_NAME !!!!!!!

!!!!! BUT RECORDS ARE NOT SORTED BY FIRST_NAME FIELD!!!!!! WHY???

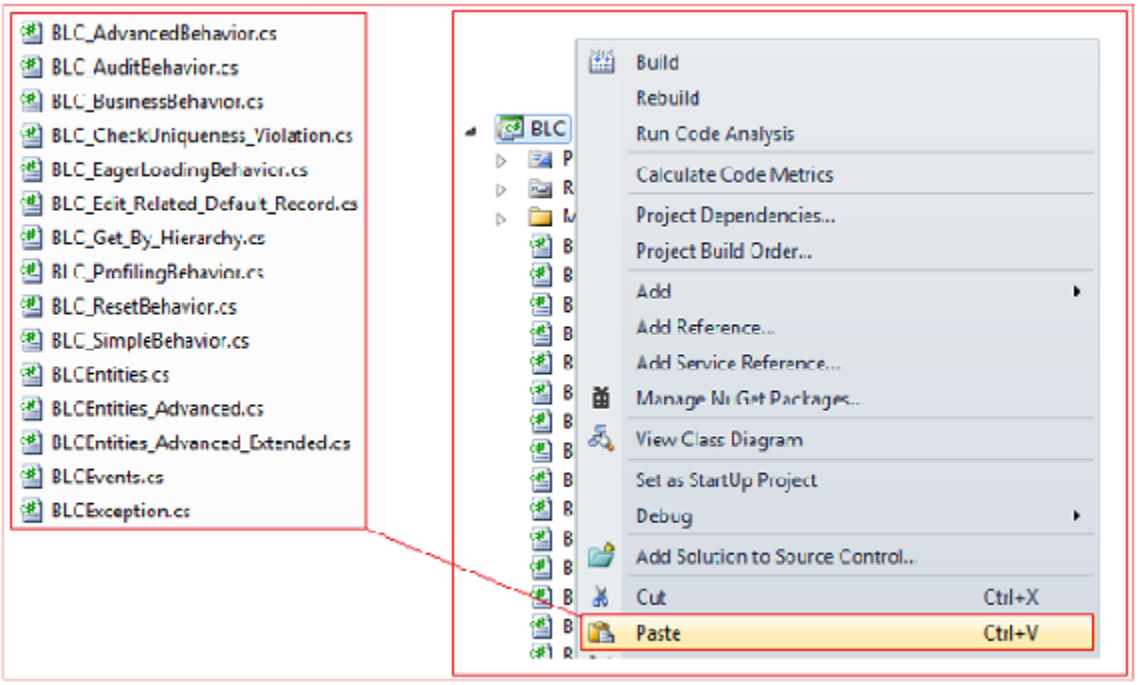
*. The fact that we are sorting the list in the onPostEvent_Get_Person_By_OWNER_ID will lead to such behavior because the sorting method will create a new instance of the list and as the i_Result is not passed by reference it will not be affected by the sorting method.



*. In order to work correctly, you have asked the code booster to generate events for Get_Person_By_OWNER_ID method using **Methods_With_Events_By_Ref** instead of **Methods_With_Events_By** as the following



***. Override the content of the BLC folder (under the newly created patch) over the BLC Library project**



*. In BLC_Initial.cs file and specifically in the SubscribeToEvents method you have to add a function to the onPostEvent_Get_Person_By_OWNER_ID events pool and implement it in “Events Implementation” region

```
#region Subscribe To Events
public void SubscribeToEvents()
{
    Declaration And Initialization Section.
    #region Body Section.
    Initialize_Eager_Loading_Mechanism();
    Initialize_Reset_Mechanism();

    this.OnPreEvent_Edit_Person +=new PreEvent_Handler_Edit_Person(BLC_OnPreEvent_Edit_Person);
    this.OnPreEvent_Edit_Person += new PreEvent_Handler_Edit_Person(BLC_OnPreEvent_Edit_Person_02);
    this.OnPostEvent_Edit_Person += new PostEvent_Handler_Edit_Person(BLC_OnPostEvent_Edit_Person);
    this.OnPostEvent_Delete_Person += new PostEvent_Handler_Delete_Person(BLC_OnPostEvent_Delete_Person);
    this.OnPreEvent_Delete_Person += new PreEvent_Handler_Delete_Person(BLC_OnPreEvent_Delete_Person);
    this.OnPostEvent_Get_Person_By_OWNER_ID += new PostEvent_Handler_Get_Person_By_OWNER_ID(BLC_OnPostEvent_Get_Person_By_OWNER_ID);
    #endregion
}
#endregion
IDisposable Members
LoadMessages
GetMessageContent
GetMessageContent
#region Events Implementation
void BLC_OnPostEvent_Get_Person_By_OWNER_ID(ref List<Person> i_Result, Params_Get_Person_By_OWNER_ID i_Params_Get_Person_By_OWNER_ID)
{
    if (i_Result != null)
    {
        i_Result = i_Result.OrderBy(x => x.FIRST_NAME).ToList();
    }
}
```

i_Result is now passed by Ref

*. In the UnitTesting console application and specifically in the “Main” method let us call Get_Person_By_OWNER_ID to check how records are sorted

```
#region Main
[STAThread]
static void Main(string[] args)
{
    Declaration And Initialization Section.
    #region Body

    Params_Get_Person_By_OWNER_ID oParams_Get_Person_By_OWNER_ID = new Params_Get_Person_By_OWNER_ID();
    oParams_Get_Person_By_OWNER_ID.OWNER_ID = 1;
    List<Person> oList_Person = oBLC.Get_Person_By_OWNER_ID(oParams_Get_Person_By_OWNER_ID);

    foreach (var oRow_Person in oList_Person)
    {
        Console.WriteLine(oRow_Person.FIRST_NAME);
    }

    #endregion
}
#endregion
```

Returned result is now sorted by FIRST_NAME field

```
A First Name
B First Name
C First Name
X First Name
```

***. Audit Feature**

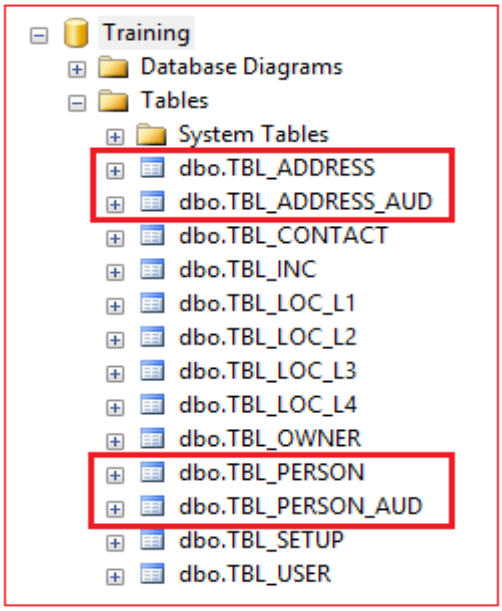
Anytime you find yourself in need to create an audit table where you have to log any operation (INSERT, UPDATE & DELETE) for a certain table in your database, all what you have to do is to inform code booster about it by filling the following collection List_Tables_To_Audit by needed tables to audit as the following

```
#region Audit
oCodeBooster.List_Tables_To_Audit = new List<string>();
oCodeBooster.List_Tables_To_Audit.Add("[TBL_PERSON]");
oCodeBooster.List_Tables_To_Audit.Add("[TBL_ADDRESS]");
#endregion
```

1. Run option "001" (asking the code booster to send us the TSQL & BLC patch)

2. Once the patch is downloaded, apply the T-SQL Script on the corresponding database, generate the DALC & override BLC files.

3. You will notice that for each table you asked the code booster to start audit, a new table has been created named TBL_XXX_AUD (AUD stands for Audit) as the following image



4. If you compare the TBL_PERSON & TBL_PERSON_AUD, you will notice that all fields of TBL_PERSON exists in TBL_PERSON_AUD in addition to "AUDIT_VERSION_CODE","HOUR","MINUTE","SECOND" fields

5. One last step before start benefiting from the audit module, is to inform the BLC layer to activate the audit mechanism by adding the following line of code in SubscribeToEvents() method as the following

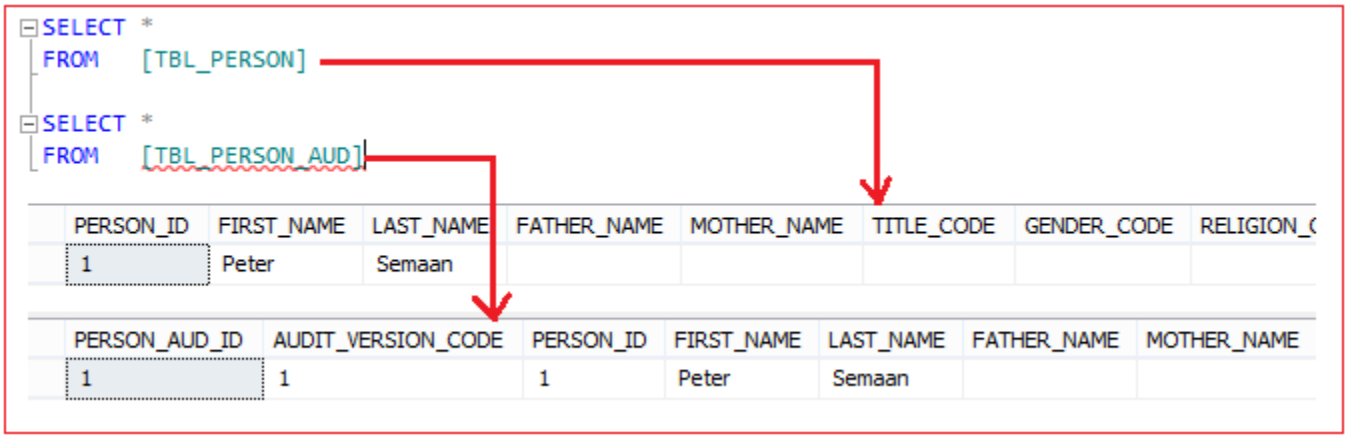
```
#region Subscribe To Events
public void SubscribeToEvents()
{
    Declaration And Initialization Section.
    #region Body Section.

    Initialize_Audit_Mechanism();

    #endregion
}
#endregion
```

6. In order to check what kind of audit data will be created in TBL_PERSON_AUD, just try to insert, update or delete records in TBL_PERSON as the following

*. **Add Record**

```
Person oPerson = new Person();
oTools.SetPropertiesDefaultValue(oPerson);
oPerson.PERSON_ID = -1;
oPerson.FIRST_NAME = "Peter";
oPerson.LAST_NAME = "Semaan";
oBLC.Edit_Person(oPerson);
```

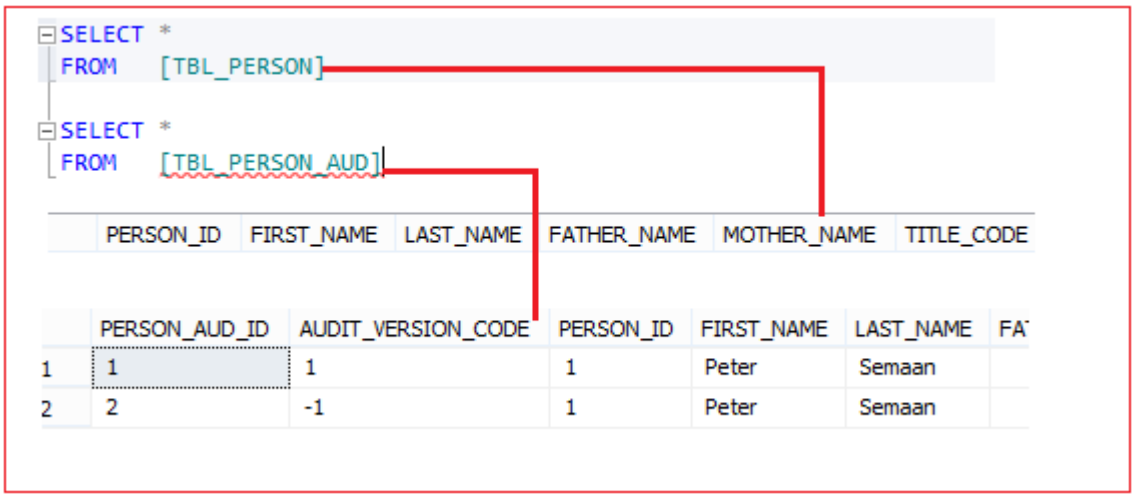
and if you check the database, you will find (beside the record normally created in TBL_PERSON) another record will be created in TBL_PERSON_AUD as the following



*. **Remove a record**

```
Params_Delete_Person oParams_Delete_Person = new Params_Delete_Person();
oParams_Delete_Person.PERSON_ID = 1;
oBLC.Delete_Person(oParams_Delete_Person);
```

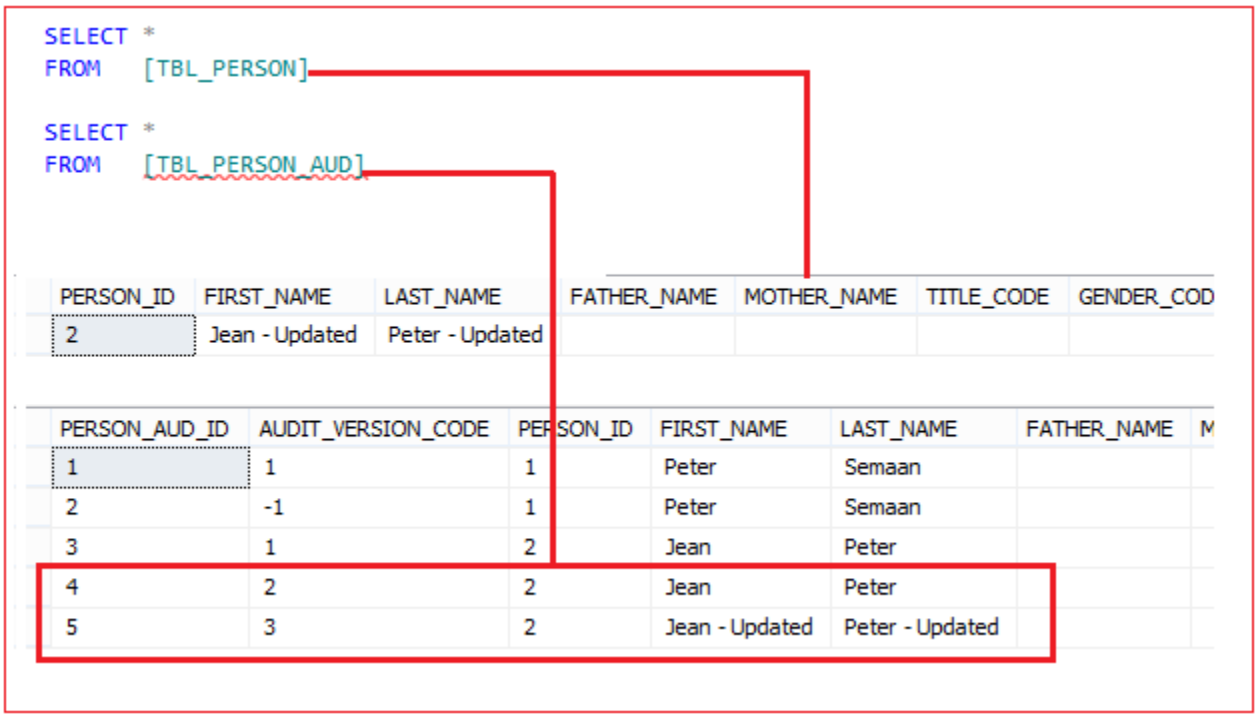
and if you check the database , beside the fact that the record has been deleted from TBL_PERSON, another record has been created in TBL_PERSON_AUD as the following



*. **Edit existing record.**

```
Person oPerson = new Person();
oTools.SetPropertiesDefaultValue(oPerson);
oPerson.PERSON_ID = 2;
oPerson.FIRST_NAME = "Jean - Updated";
oPerson.LAST_NAME = "Peter - Updated";
oBLC.Edit_Person(oPerson);
```

And in the database you will find that beside the fact that the record in question has been updated, two records have been created in the TBL_PERSON_AUD representing the before & after images of the record in question as the following



```
SELECT *
FROM   [TBL_PERSON]

SELECT *
FROM   [TBL_PERSON_AUD]
```

PERSON_ID	FIRST_NAME	LAST_NAME	FATHER_NAME	MOTHER_NAME	TITLE_CODE	GENDER_COD
2	Jean - Updated	Peter - Updated				

PERSON_AUD_ID	AUDIT_VERSION_CODE	PERSON_ID	FIRST_NAME	LAST_NAME	FATHER_NAME	M
1	1	1	Peter	Semaan		
2	-1	1	Peter	Semaan		
3	1	2	Jean	Peter		
4	2	2	Jean	Peter		
5	3	2	Jean - Updated	Peter - Updated		

*. **Methods monitoring**

One of the biggest questions in any enterprise level application is where the bottleneck in our code in terms of performance, and if you do not have the ability / luxury to have a performance monitoring on your code usability. you will spend a lot of time to find the bottleneck reason.

While using Code Booster, it is a matter of setting a simple flag named Is_Method_Monitoring_Enabled.

In order to test it, do the following:

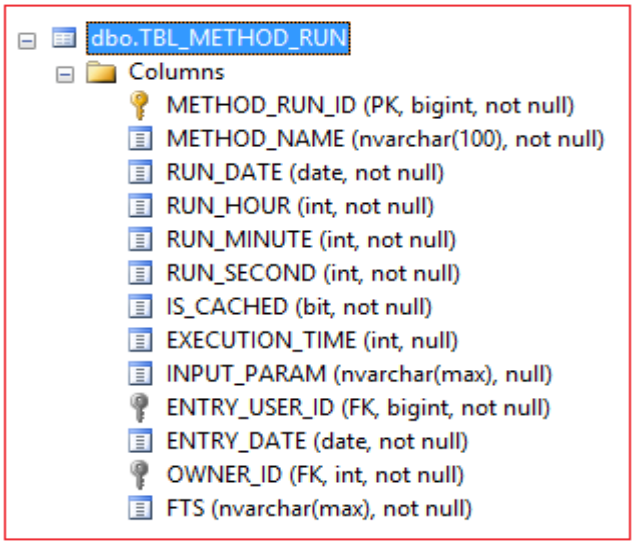
1. Set the Is_Method_Monitoring_Enabled = true as the following

```
#region Method Monitoring.
oCodeBooster.Is_Method_Monitoring_Enabled = true;
#endregion
```

2. Run option "001" (Asking the code booster to send us a new patch)

3. Once the patch is downloaded, apply the T-SQL Script on the corresponding database, generate the DALC & override BLC files.

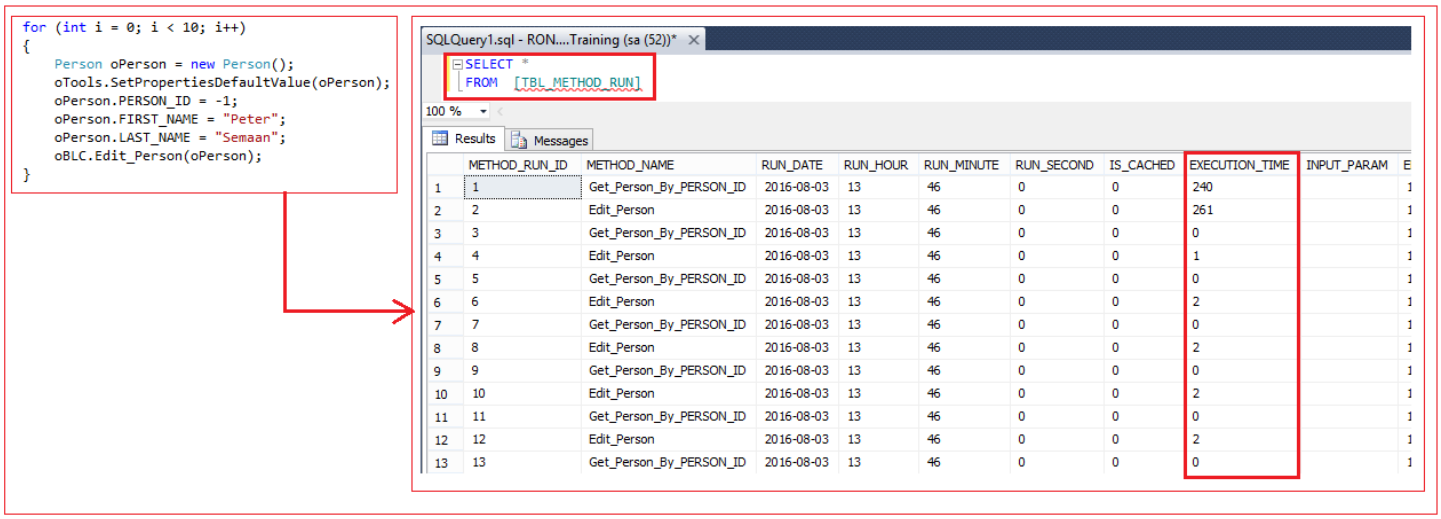
4. You will notice that a new table has been created in your database named TBL_METHOD_RUN



5. One last step before start benefiting from the audit module, is to inform the BLC layer to activate the monitoring mechanism by adding the following line of code in SubscribeToEvents() method as the following

```
#region Subscribe To Events
public void SubscribeToEvents()
{
    Declaration And Initialization Section.
    #region Body Section.

    Initialize_Audit_Mechanism();

    Initialize_Monitoring_Mechanism();

    #endregion
}
#endregion
```

6. Now, in order to check how it works, try to call any method(s) and you will notice that every single call for any BLC method will be logged in the database showing how much time it takes in mill second.



```
for (int i = 0; i < 10; i++)
{
    Person oPerson = new Person();
    oTools.SetPropertiesDefaultValue(oPerson);
    oPerson.PERSON_ID = -1;
    oPerson.FIRST_NAME = "Peter";
    oPerson.LAST_NAME = "Semaan";
    oBLC.Edit_Person(oPerson);
}
```

SQLQuery1.sql - RON....Training (sa (52))*

```
SELECT *
FROM [TBL_METHOD_RUN]
```

100 %

Results | Messages

	METHOD_RUN_ID	METHOD_NAME	RUN_DATE	RUN_HOUR	RUN_MINUTE	RUN_SECOND	IS_CACHED	EXECUTION_TIME	INPUT_PARAM	E
1	1	Get_Person_By_PERSON_ID	2016-08-03	13	46	0	0	240		1
2	2	Edit_Person	2016-08-03	13	46	0	0	261		1
3	3	Get_Person_By_PERSON_ID	2016-08-03	13	46	0	0	0		1
4	4	Edit_Person	2016-08-03	13	46	0	0	1		1
5	5	Get_Person_By_PERSON_ID	2016-08-03	13	46	0	0	0		1
6	6	Edit_Person	2016-08-03	13	46	0	0	2		1
7	7	Get_Person_By_PERSON_ID	2016-08-03	13	46	0	0	0		1
8	8	Edit_Person	2016-08-03	13	46	0	0	2		1
9	9	Get_Person_By_PERSON_ID	2016-08-03	13	46	0	0	0		1
10	10	Edit_Person	2016-08-03	13	46	0	0	2		1
11	11	Get_Person_By_PERSON_ID	2016-08-03	13	46	0	0	0		1
12	12	Edit_Person	2016-08-03	13	46	0	0	2		1
13	13	Get_Person_By_PERSON_ID	2016-08-03	13	46	0	0	0		1

*. **Caching**

What we mean by caching is the ability to save the result of calling a BLC method so next time we call the same method with the same parameters, the result will be retrieved from the cache avoiding a hit on the database which makes the application very responsive.

In order to "cache" a certain method (BLC Method), all what you must do is to inform code booster about the method you want to cache by filling the following collection.

Enable Caching feature by setting the following flag:

```
oCodeBooster.Is_Caching_Enabled = true;
```

```
#region Cache
oCodeBooster.Methods_With_Caching = new List<Caching_Topology>();
oCodeBooster.Methods_With_Caching.Add(new Caching_Topology() { Method_Name = "Get_Person_By_PERSON_ID" });
#endregion
```

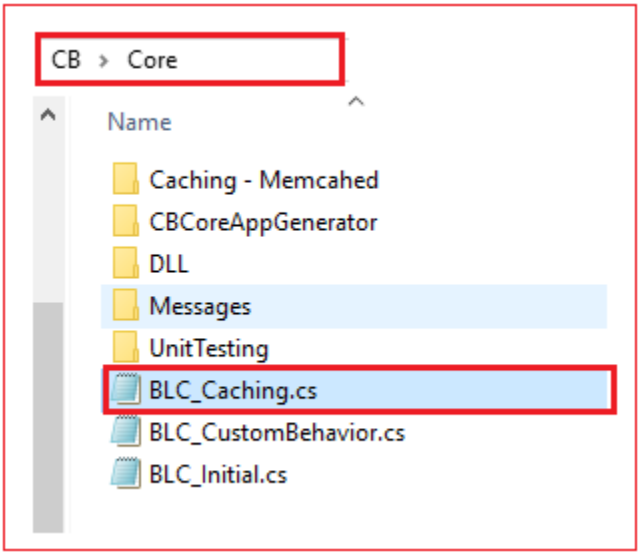
It's mandatory to know that the caching is applicable for both Web & Mobile Application consuming API.

- Run option "001" (Asking the code booster to send us a new patch)

- Once the patch is downloaded, apply the T-SQL Script on the corresponding database, generate the DALC & override BLC files.

- Before start benefiting from the caching feature, you must:

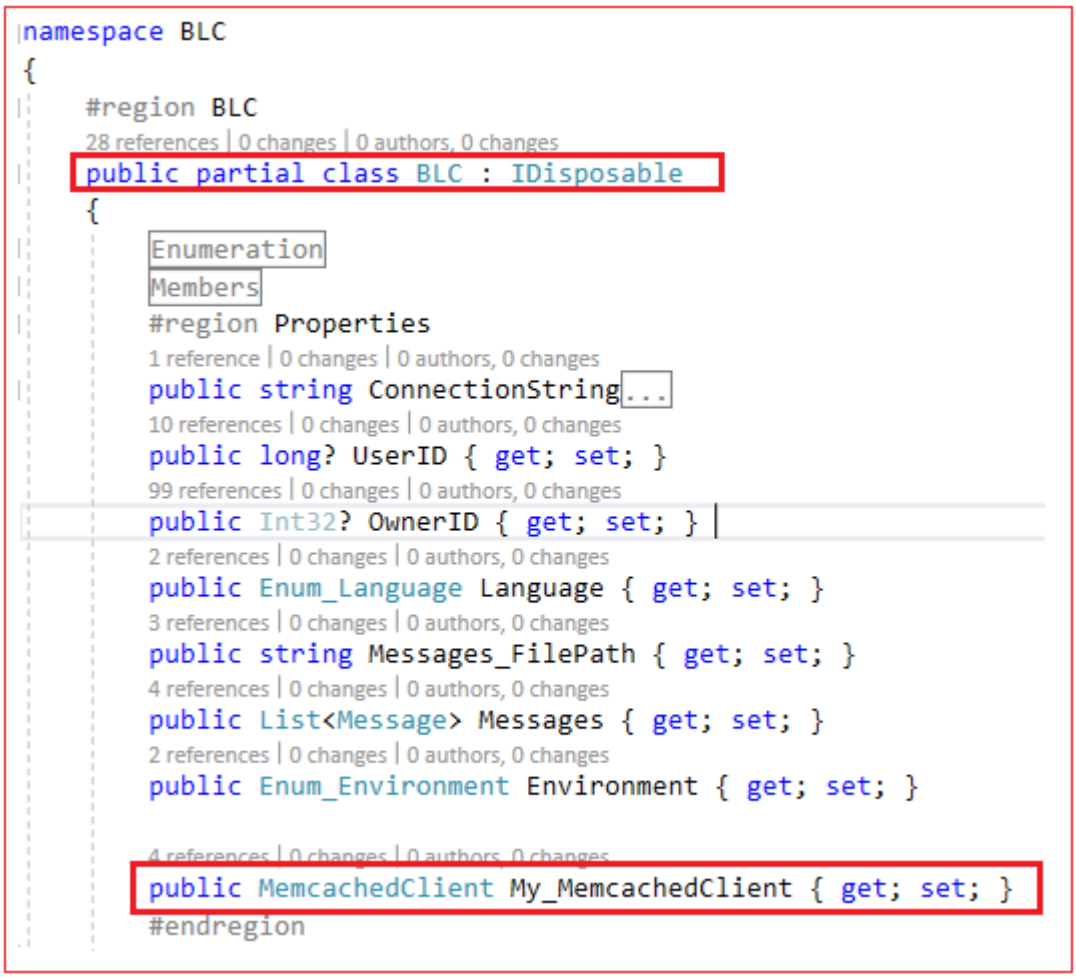
    *. Add to your BLC Project the BLC_Caching.cs file that can be found under CB\Core folder



*. In the BLC_Initial.cs :

*. Add the following property:

public MemcachedClient My_MemcachedClient { get; set; }



*. In the Parameterized constructor add the following code

this.My_MemcachedClient = Init_MemcachedClient();

```
1 reference | 0 changes | 0 authors, 0 changes
public BLC(BLCInitializer i_BLCInitializer)
{
    Declaration And Initialization Section.
    #region Body Section.
    // ---------------------
    ConnectionString = i_BLCInitializer.ConnectionString;
    UserID = i_BLCInitializer.UserID;
    OwnerID = i_BLCInitializer.OwnerID;
    Language = i_BLCInitializer.Language;
    Messages_FilePath = i_BLCInitializer.Messages_FilePath;
    _AppContext = new DALC.MSSQL_DALC(_ConnectionString);
    this.My_MemcachedClient = Init_MemcachedClient();
    // ---------------------

    // ---------------------
    LoadMessages();
    SubscribeToEvents();
    //Initialize_Audit_Mechanism();
    // ---------------------
```

*. The BLC_Caching.cs file contains a caching implementation using **Memcached** as the caching server.

*. Under the **CB\Core** folder you will find a folder named **memcached-server** which contains **2** files

memcached.exe & **pthreadGC2.dll**

*. Copy the "memcached-server" folder to your favorite directory on your PC

*. and you can start the caching server by issuing the following command (in cmd)

memcached.exe -vv -I 5M -m 1024

*. In The App.config file of your application assure you have the following entries

```
<appSettings>
  .....
  .....
  <add key="CACHING_ENABLED" value="1"/>
  <add key="MEMCACHED_IP" value="192.168.1.5"/>
  <add key="MEMCACHED_PORT" value="11211"/>
</appSettings>
```

(**Sure, the IP should be the server's IP**)

*. **Cascade Delete**

In case you want upon deleting a certain entry in a certain table to delete all related entries in different tables while applying the business rules applied upon deletion, you case use the following collection

```
#region Cascade
oCodeBooster.List_Cascade_Tables = new List<Cascade>();
oCodeBooster.List_Cascade_Tables.Add
    (
        new Cascade()
        {
            ParentTable = "[TBL_PERSON]",
            ChildTables = new List<string>()
            {
                "[TBL_ADDRESS]",
                "[TBL_CONTACT]"
            }
        }
    );
#endregion
```

In this scenario, we informed the code booster that upon deleting any entry in TBL_PERSON table, you must delete first related entries in TBL_ADDRESS & TBL_CONTACT tables.

- Run option "001" (Asking the code booster to send us a new patch)

- Once the patch is downloaded, apply the T-SQL Script on the corresponding database, generate the DALC & override BLC files.

- One last step before start benefiting from the cascading feature, is to inform the BLC layer to activate the cascade mechanism by adding the following line of code in SubscribeToEvents () method as the following

```
#region Subscribe To Events
public void SubscribeToEvents()
{
    Declaration And Initialization Section.
    #region Body Section.

    Initialize_Audit_Mechanism();

    Initialize_Cascade_Mechanism();

    #endregion
}
#endregion
```

*. **Custom Procedure BLC Wrappers:**

It's known by now that the code booster engine generates required stored procedures & user defined functions in the database and at the same time it generates the corresponding BLC Methods in (BLC_SimpleBehavior.cs, BLC_AdvancedBehavior.cs).

And in case you have created your own custom stored procedures you can ask the code booster to generate the corresponding DALC methods that call these custom procedures by just setting the following flag in CBCoreAppGenerator Console Application (especially in program.cs)

oCodeBoosterClient.Is_Handle_Custom_Procedures = true;

Once you run option "001" and you receive the patch, you will replace DALC files under the DALC class library and you will notice that new methods have been added to the IDALC & the corresponding implementation and you can use it in your BLC Project (for example) as the following:

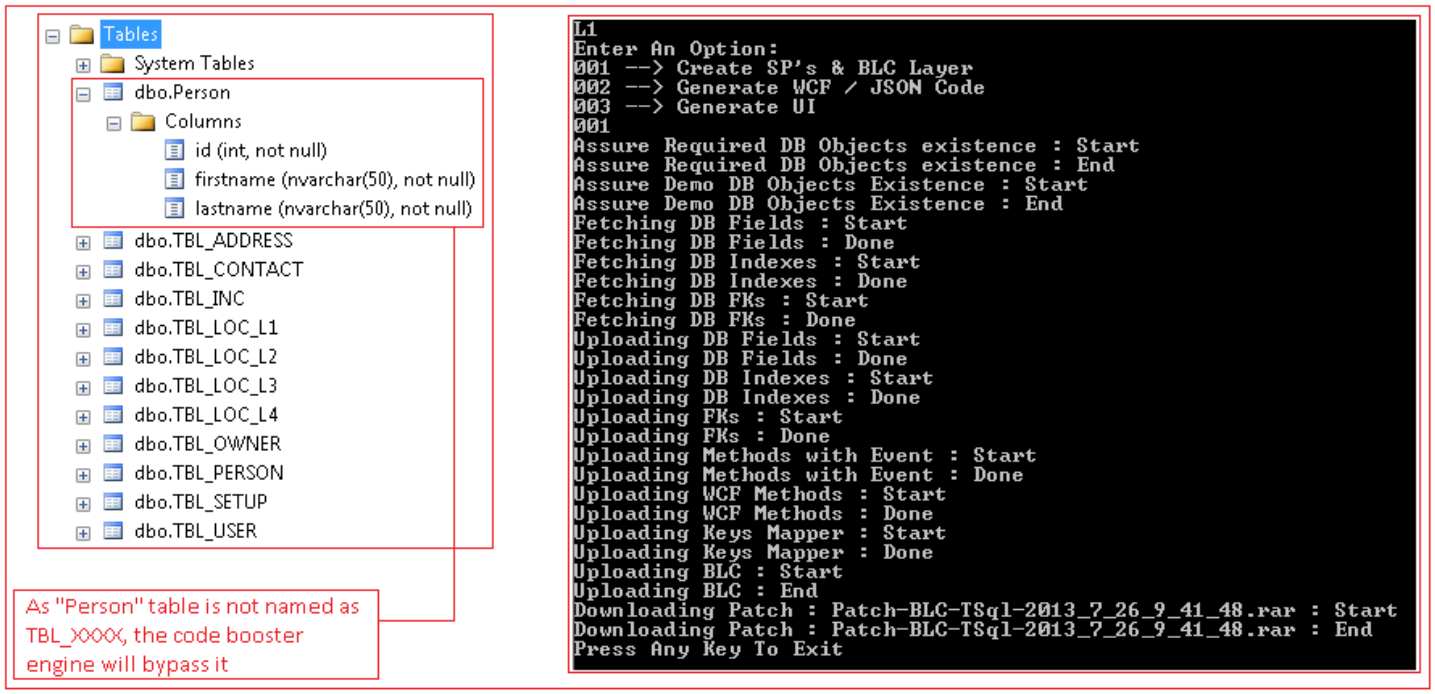
```
public List<SetupEntry> Get_SetupEntries_Per_Table(Params_Get_SetupEntries_Per_Table i_Params_Get_SetupEntries_Per_Table)
{
    #region Declaration And Initilization Section.
    Tools.Tools oTools = new Tools.Tools();
    SetupEntry oSetupEntry = new SetupEntry();
    List<SetupEntry> oList = new List<SetupEntry>();
    #endregion
    #region Body Section.
    var oQuery = from oItem in _AppContext.GET_TBL_SETUP()
                 where (oItem.TBL_NAME == i_Params_Get_SetupEntries_Per_Table.TBL_NAME)
                 select oItem;
    if (oQuery != null)
    {
        foreach (var oRow in oQuery)
        {
            oSetupEntry = new SetupEntry();
            oTools.CopyPropValues(oRow, oSetupEntry);
            oList.Add(oSetupEntry);
        }
    }
    #endregion
    #region Return Section
    return oList;
    #endregion
}
```

```
public UserInfo Authenticate(Params_Authenticate i_Params_Authenticate)
{
    #region Declaration And Initialization Section.
    Crypto.MiniCryptoHelper oCrypto = new Crypto.MiniCryptoHelper();
    string str_Ticket_PlainText = string.Empty;
    string str_Ticket_Encrypted = string.Empty;
    Int32? i_ExpiryPeriod = 240; // In Minutes
    long? i_MinutesElapsedSinceMidnight = 0;

    Tools.Tools oTools = new Tools.Tools();

    #endregion
    #region Body Section.
    var oQuery = from oItem in _AppContext.GET_TBL_USER()
                 where
                    (
                        (oItem.USERNAME == i_Params_Authenticate.My_UserInfo.UserName)
                    )
                 select oItem;

    if (oQuery.Count() == 1)
    {
        var oResult = oQuery.First();
        if
            (
                (oCrypto.Decrypt(oResult.PASSWORD) == i_Params_Authenticate.My_UserInfo.Password)
            )
        {
```

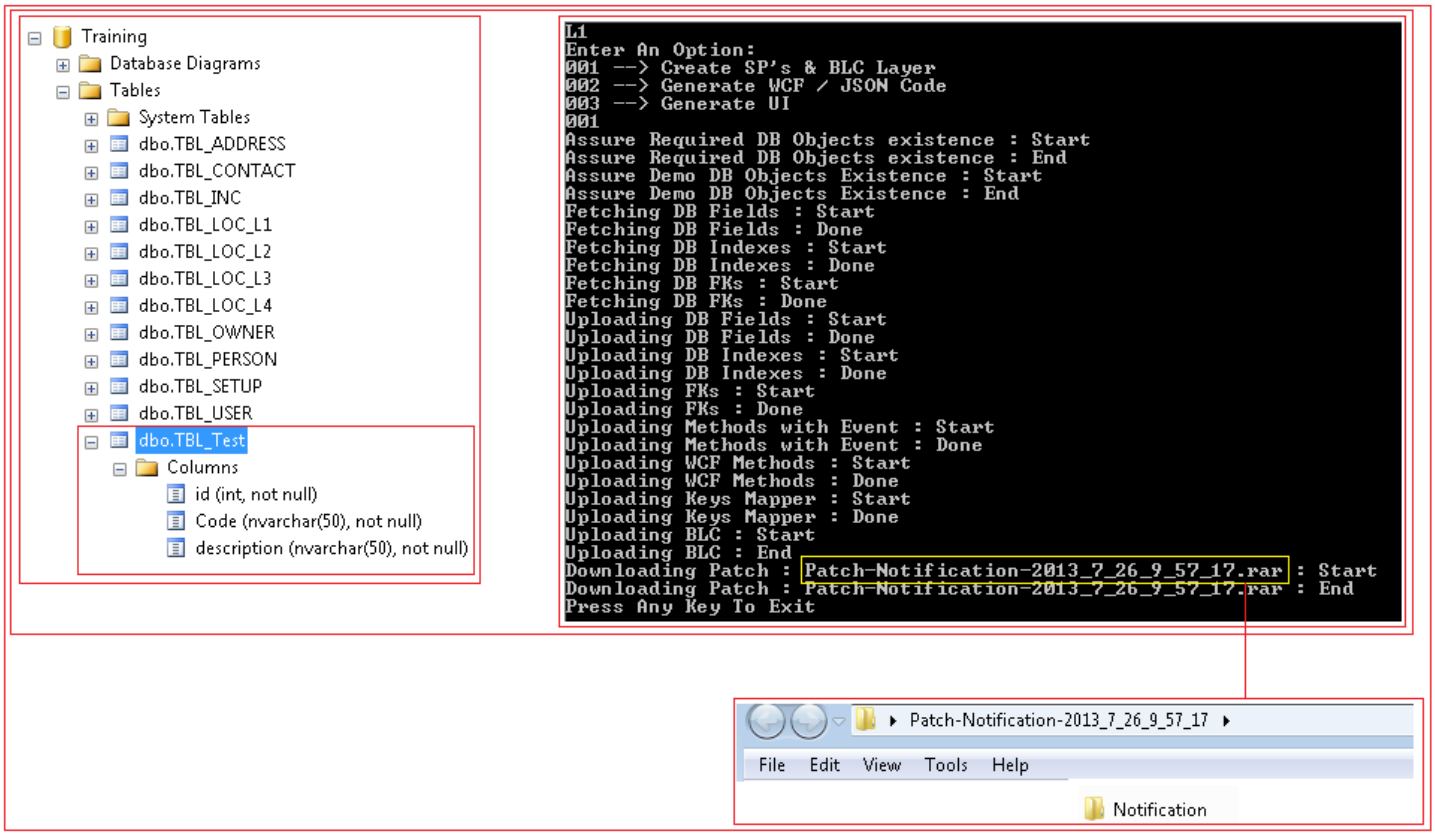
Notifications

***. When you ask the Code Booster engine to generate a patch for you, indirectly you are asking it to assure that you created your database(and code as we will explain later on) in a standard way and having all required fields and tables to be named as enterprise database.**

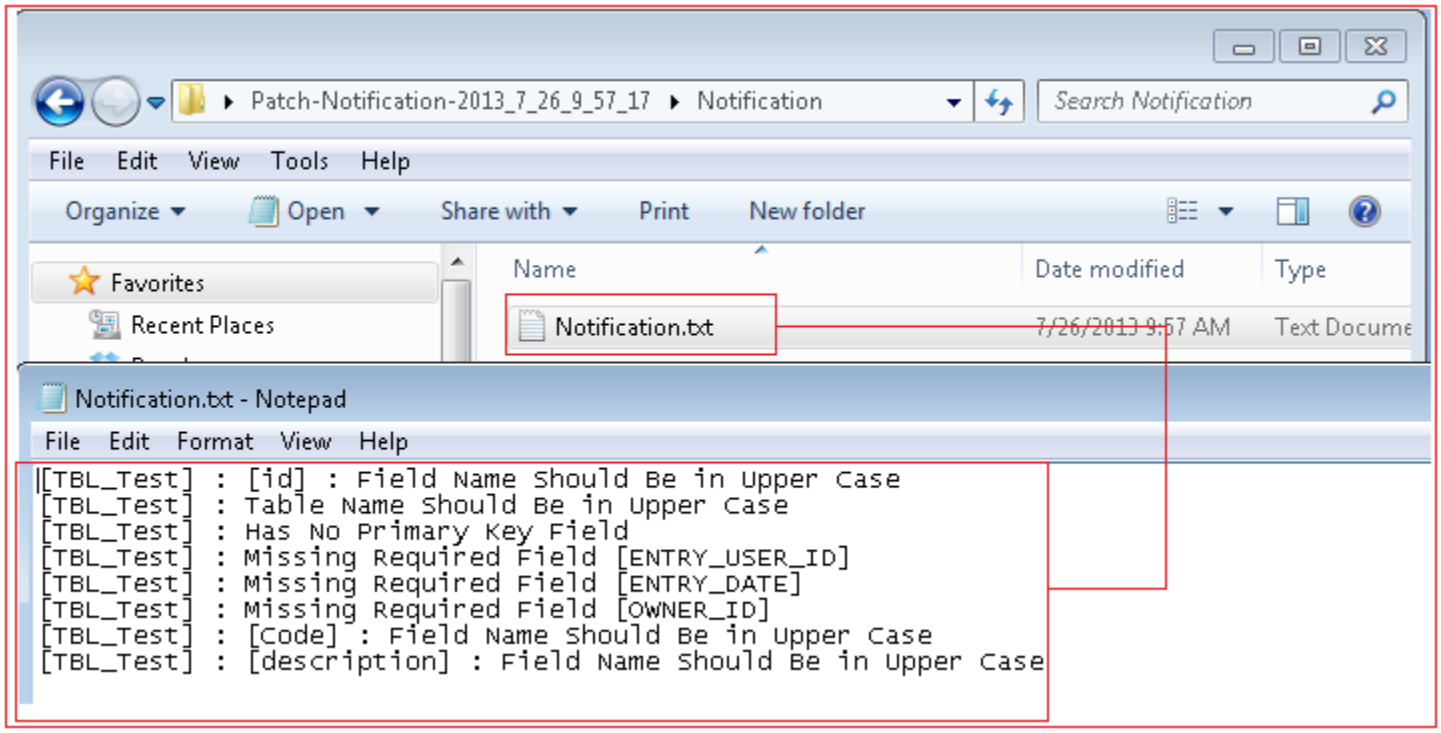
***. The Code booster engine will only check tables named as TBL_XXX or it will be bypassed by the generation process**



***. Let's create table named "TBL_Test" having id, Code and description fields and let's fetch out what notifications will be sent from the code booster engine.**

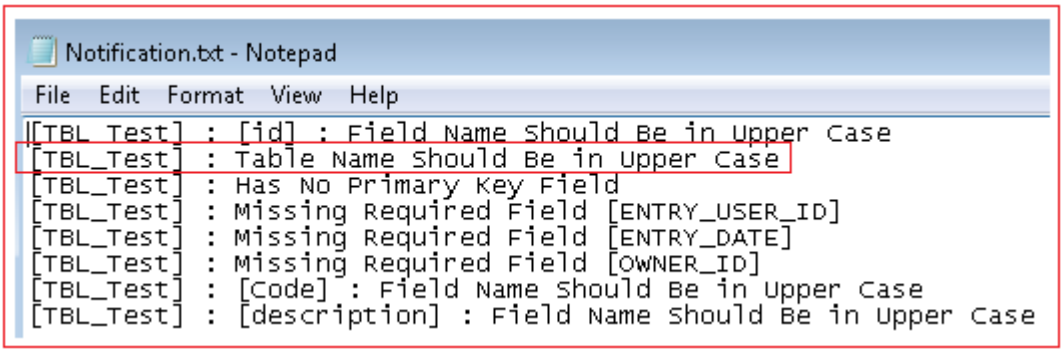


The code booster engine will not send you in this case a Patch but a notification file as the following.

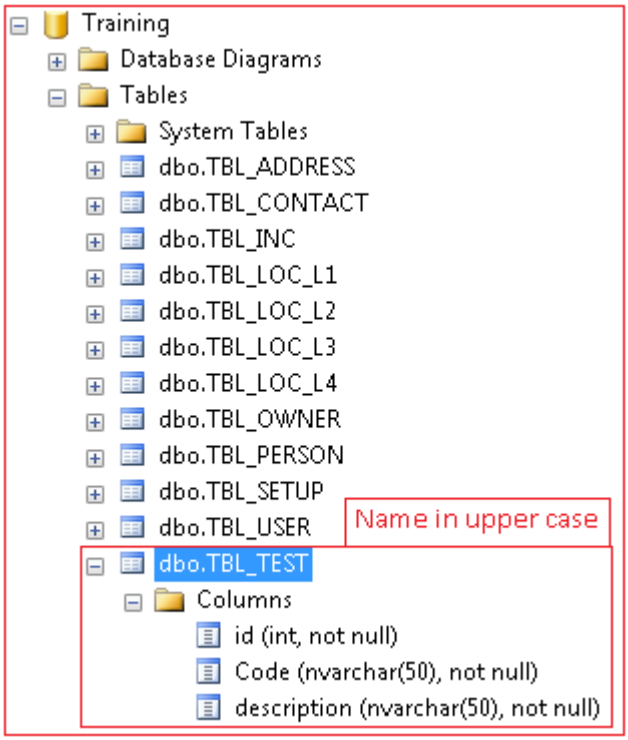


***. We will tackle each notification entry on its own in order to show how to fix it.**

***. First of all, let's handle the "Table name should be in upper case"**

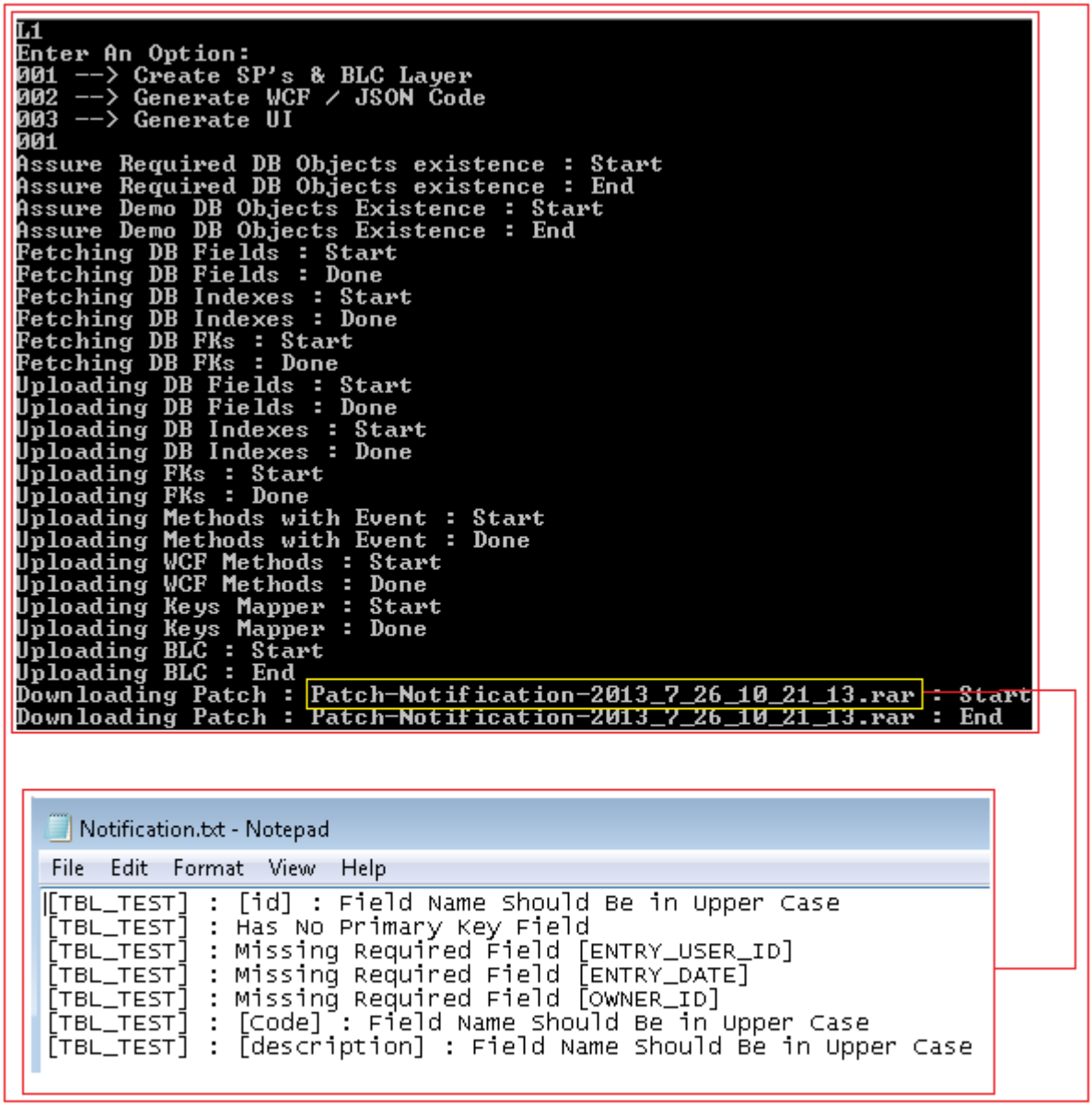


***. Assure that you named your tables in upper case.**



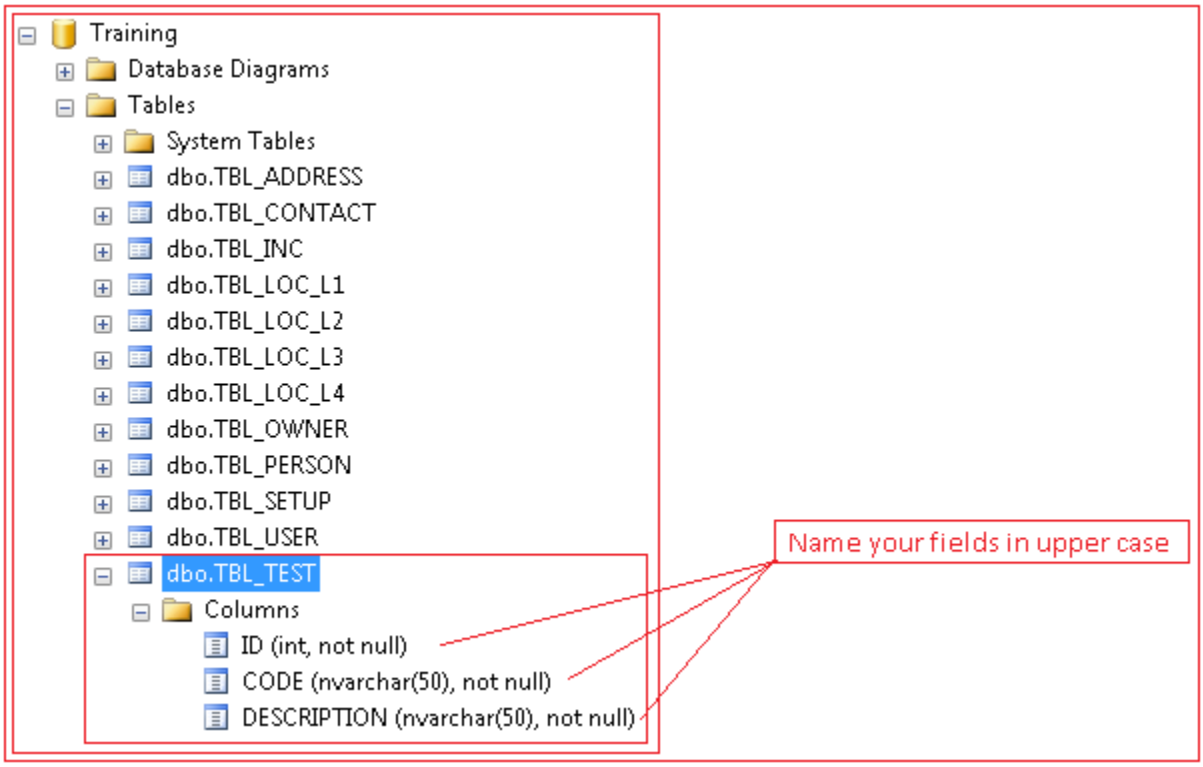
***. Ask the code booster to generate a new patch.**

Sure, a notification file will be generated (Because we fixed only the “Field name should be in upper case”) but the rest of notifications are still not solved.



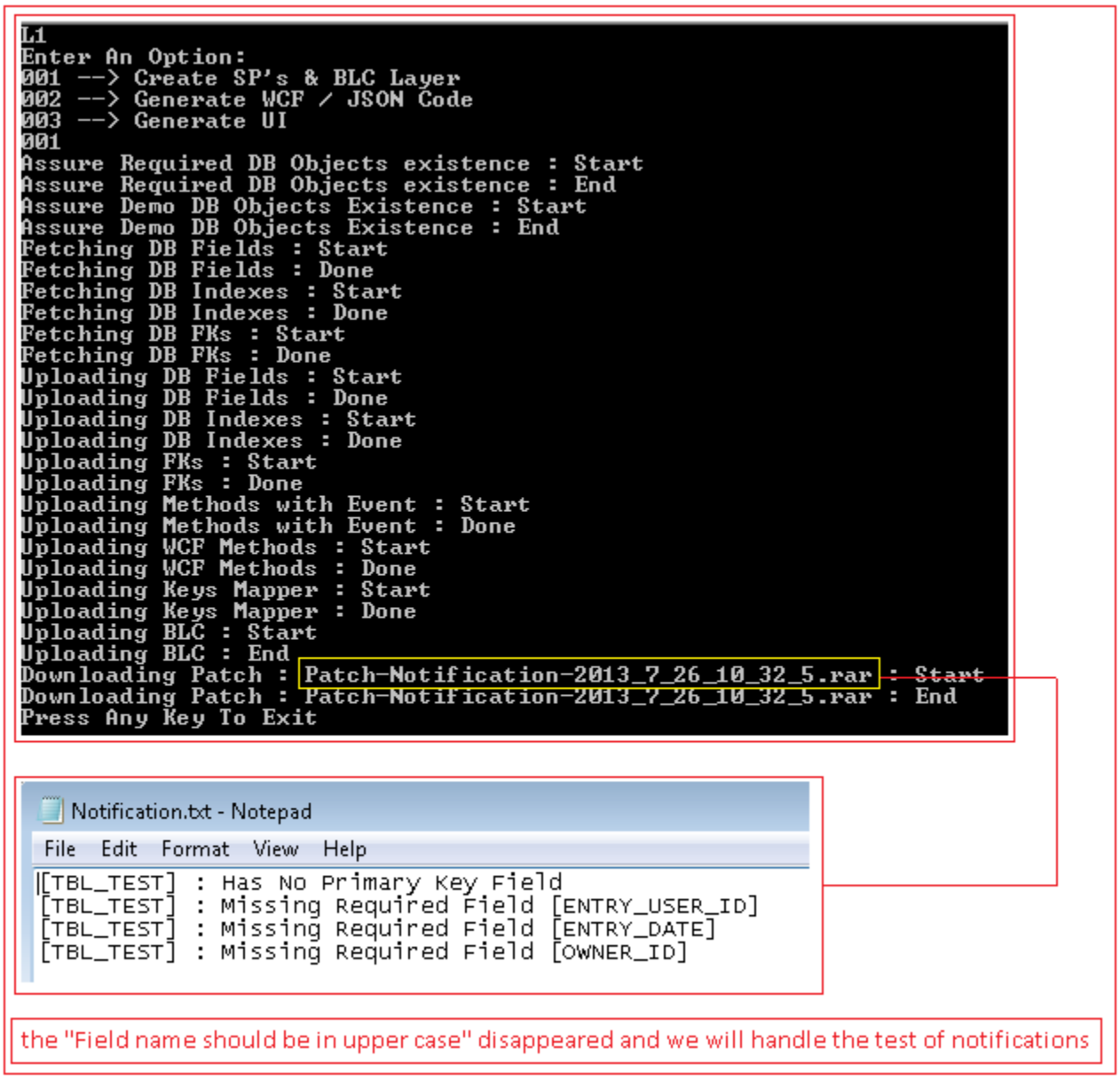
The "Table name should be in upper case" disappeared and we will handle the rest of notification entries.

***. Assure you named your fields in upper case.**

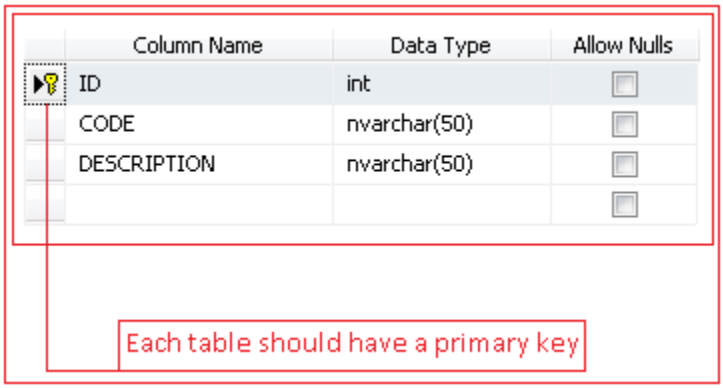


***. Ask the code booster engine to generate a new patch.**

Sure, a notification file will be generated (Because we fixed only the “Table name should be in upper case”) but the rest of notifications are still not solved.

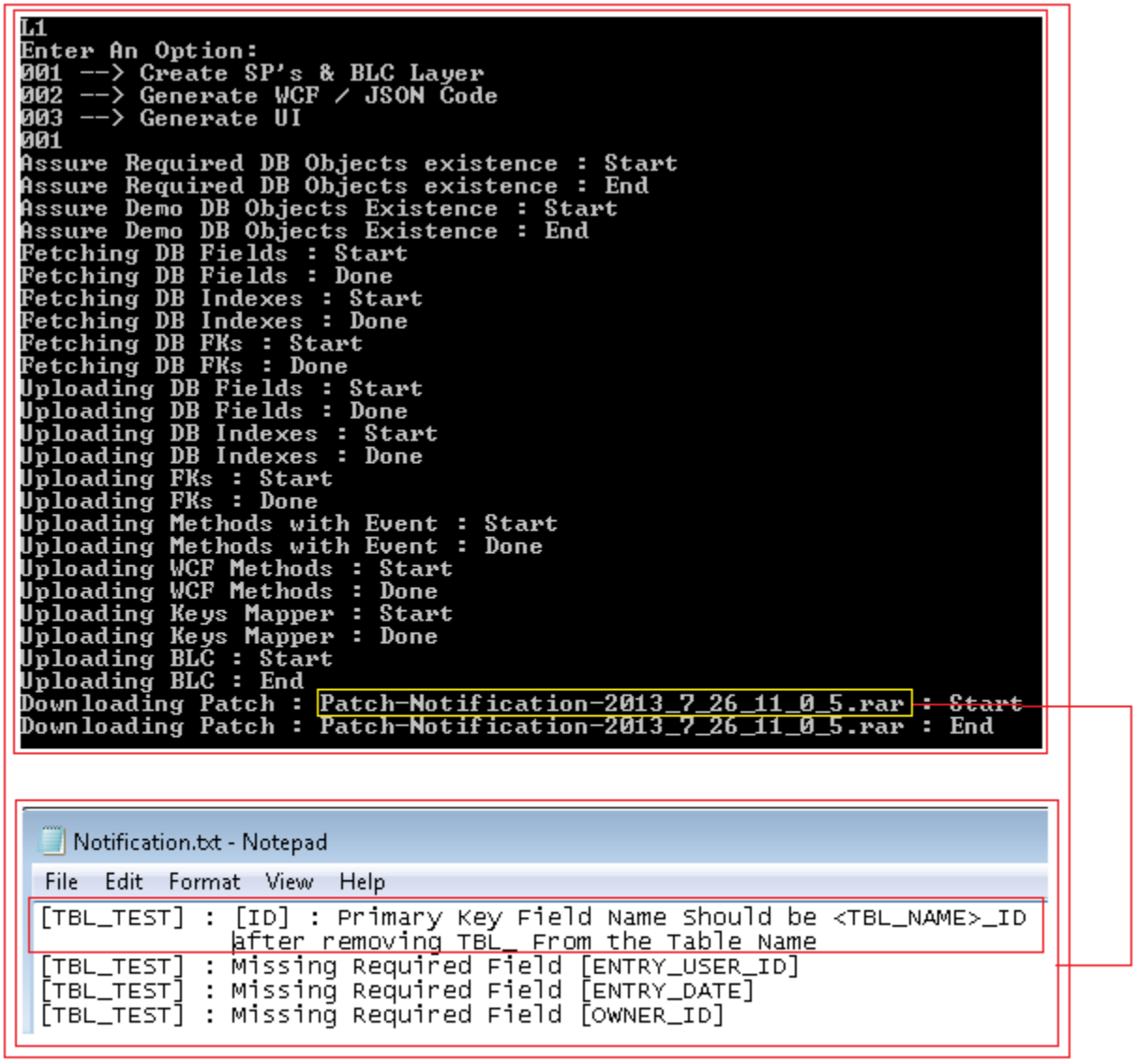


***. Assure that you have primary for each table.**

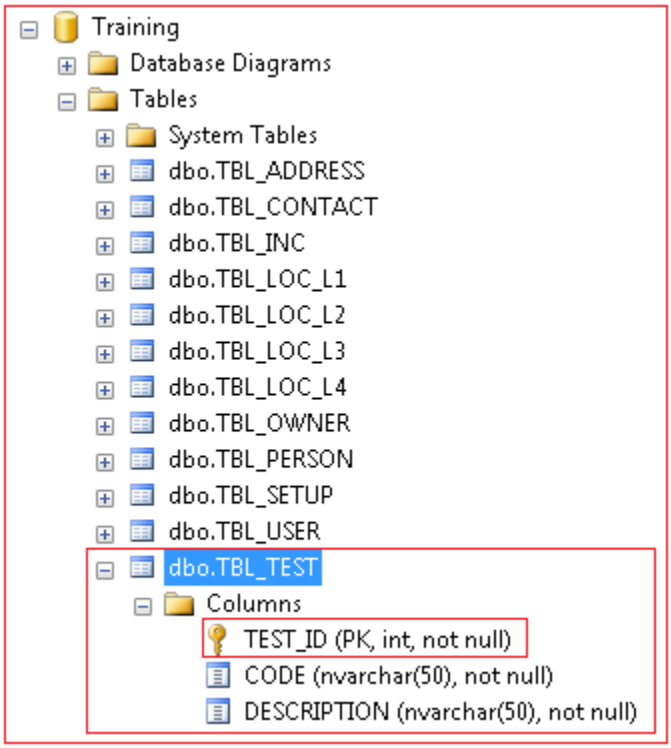


***. Ask the code booster engine to generate a new patch.**

You will notice in the notification file, that the “Has no primary key field” disappeared, but a new notification entry is shown saying “Primary Key Field Name should be <TBL_NAME>_ID after removing TBL_ from the Table Name”

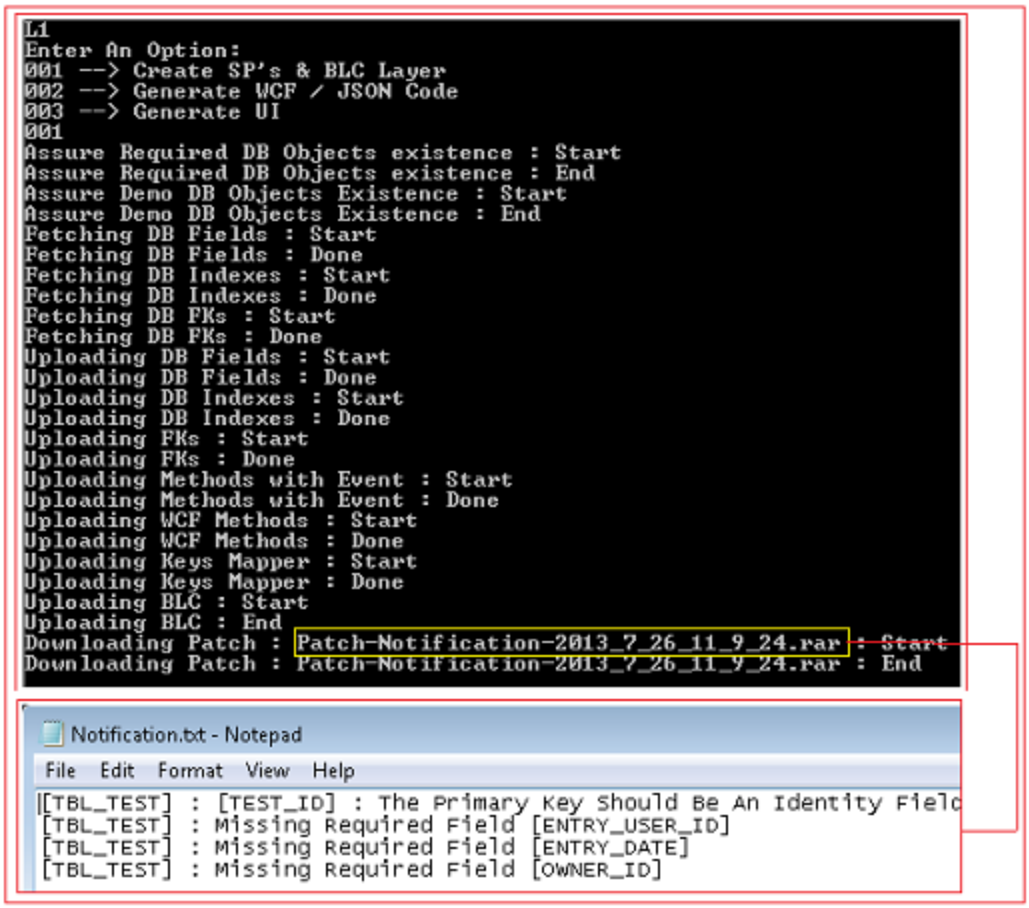


***. Assure you named the primary key field as “TEST_ID”**

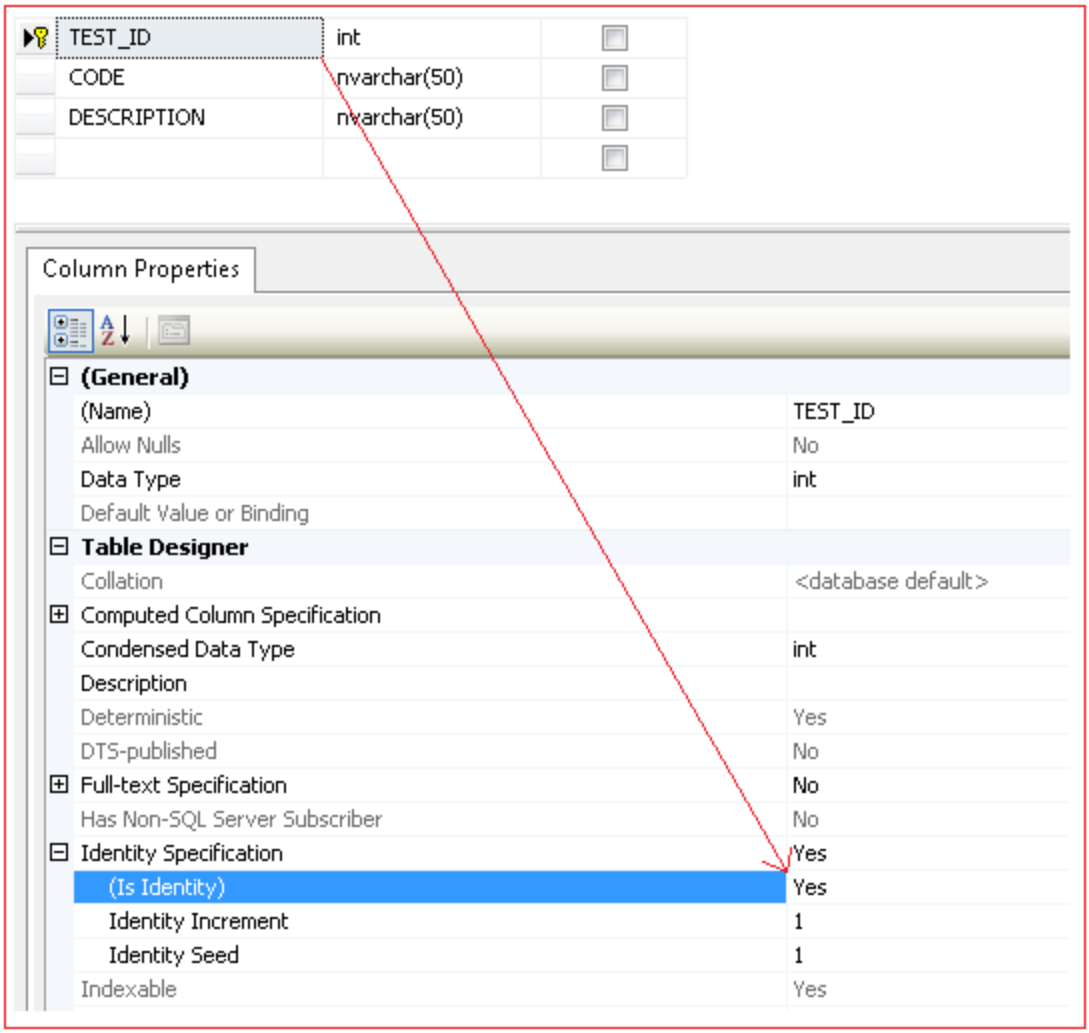


***. Ask the code booster to generate a new patch.**

A new notification is now generated informing you that you have to set it as identity (auto-increment) the primary key.

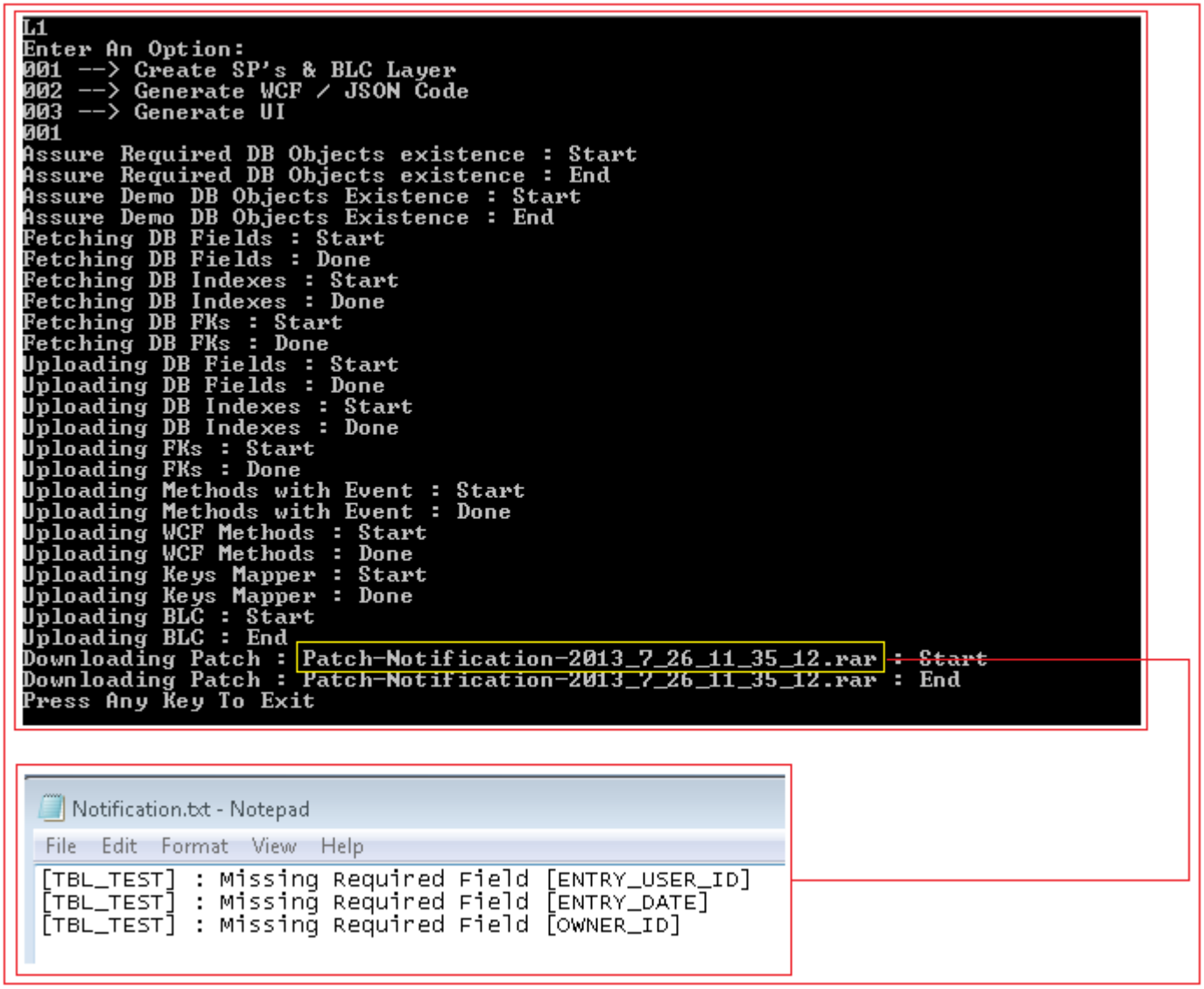


***. Let's set the primary field "TEST_ID" as identity field.**


TEST_ID
int
CODE
nvarchar(50)
DESCRIPTION
nvarchar(50)
Column Properties
(General)
(Name)
TEST_ID
Allow Nulls
No
Data Type
int
Default Value or Binding
Table Designer
Collation
<database default>
Computed Column Specification
Condensed Data Type
int
Description
Deterministic
Yes
DTS-published
No
Full-text Specification
No
Has Non-SQL Server Subscriber
No
Identity Specification
Yes
(Is Identity)
Yes
Identity Increment
1
Identity Seed
1
Indexable
Yes

***. Ask the code booster to generate a new patch.**

Finally, all notifications related to the primary key (in terms of naming and identity nature) are gone and we still have notifications mentioning that you have missing fields to be added to this table.

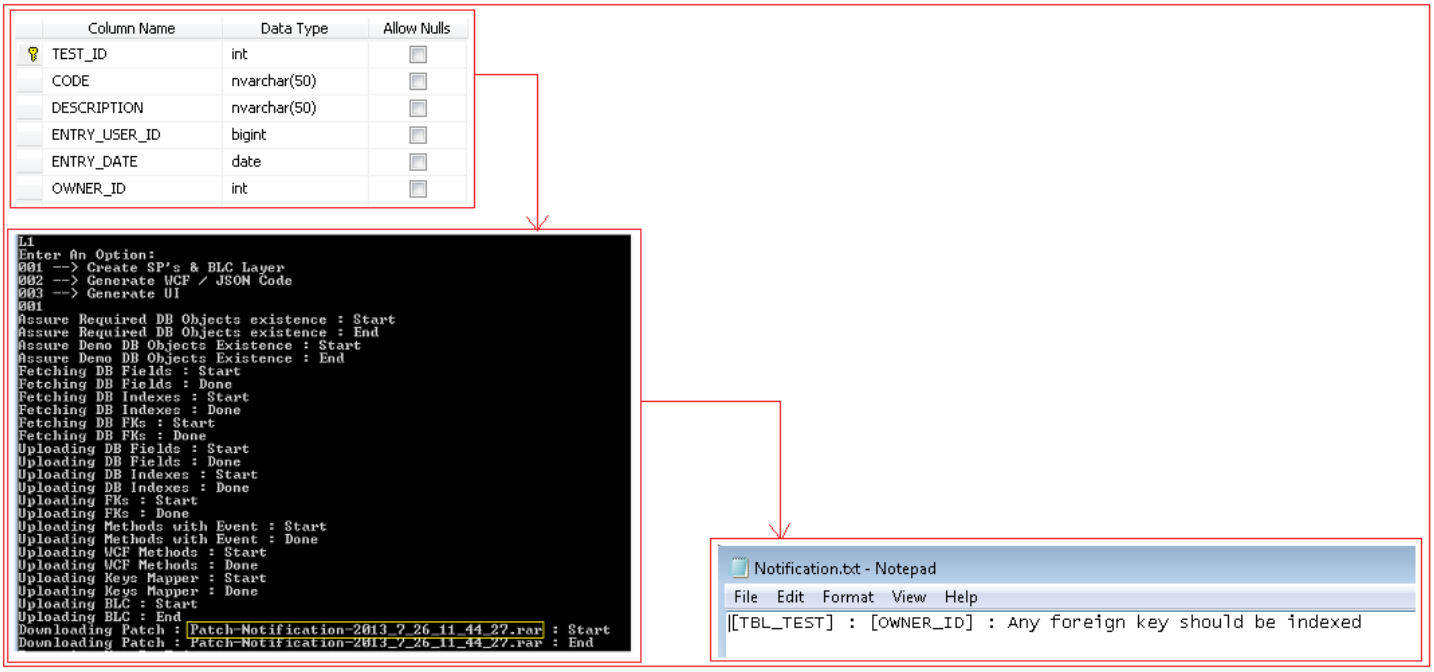


***. Let us add missing fields:**

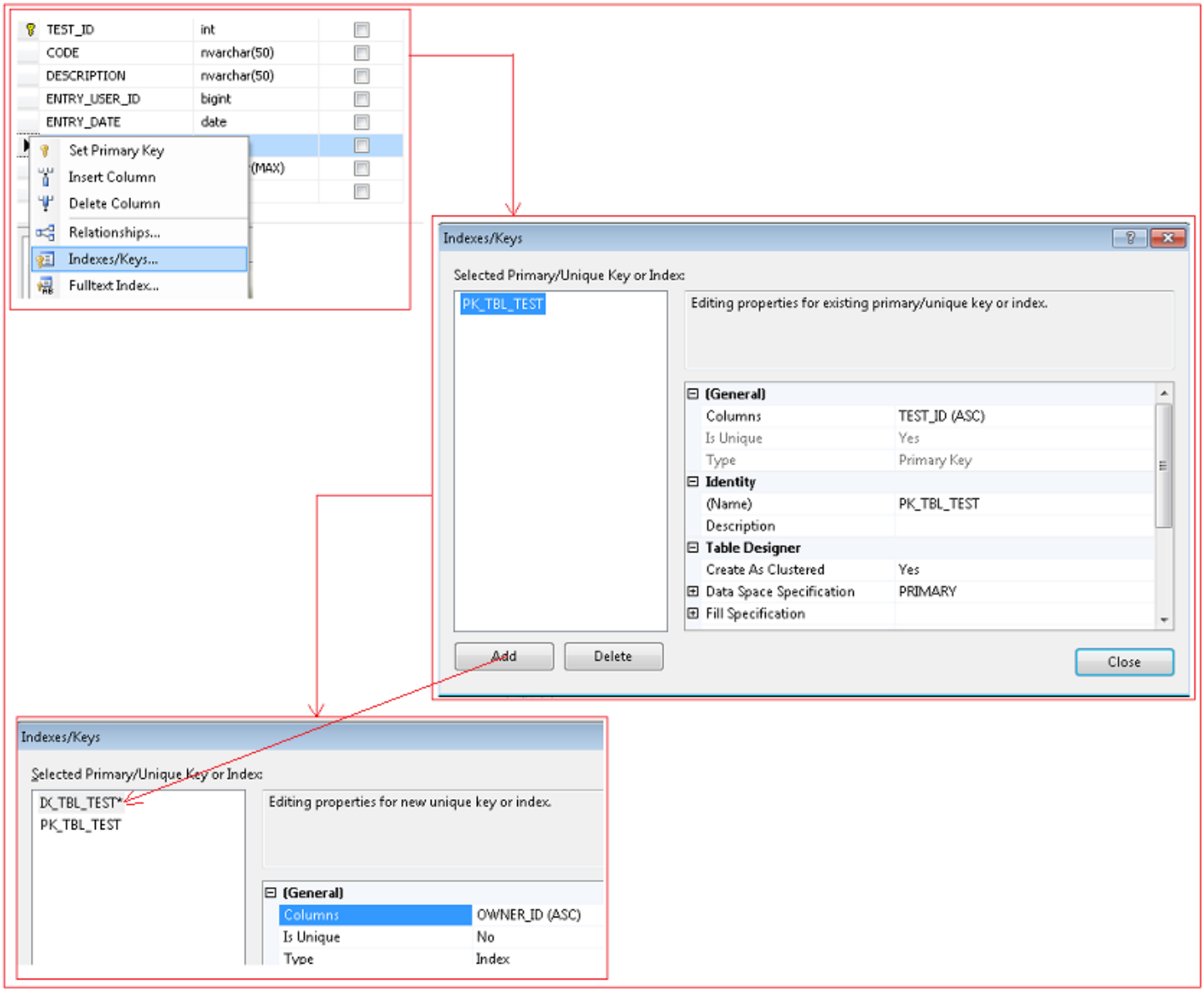
*. ENTRY_USER_ID: this field will store the last user that has managed a certain record (create or update)

*. ENTRY_DATE: this field will store the last date this record has been managed.

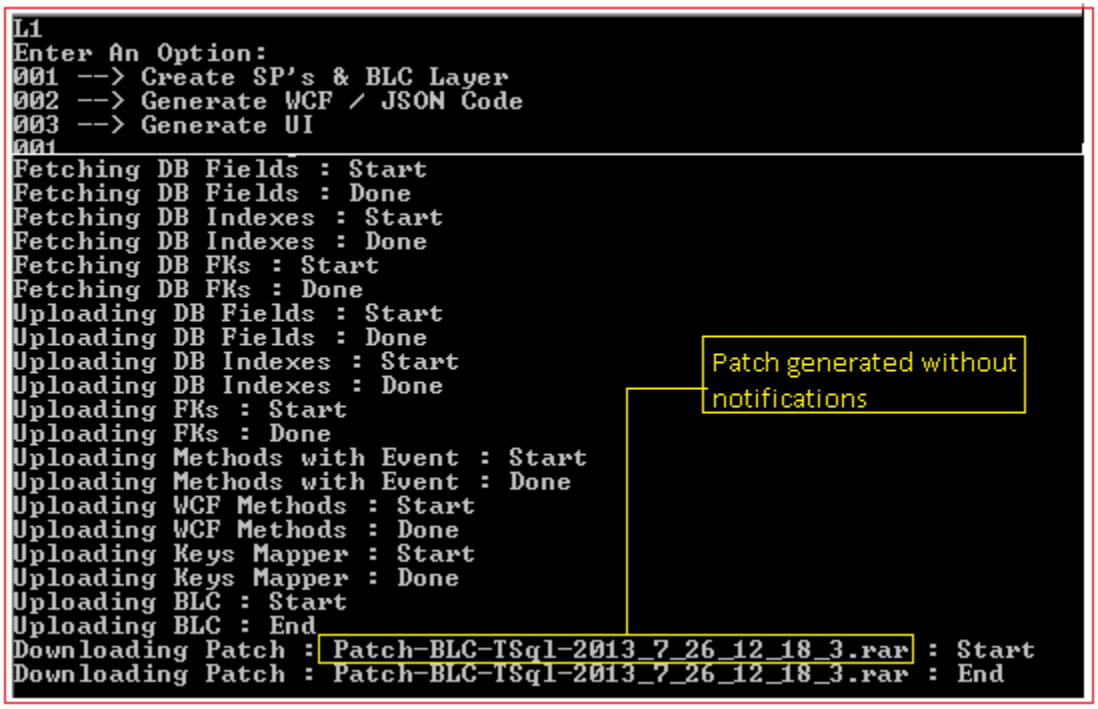
*. OWNER_ID: as previously mentioned, this field is used for SaaS development.



As OWNER_ID is the primary key of TBL_OWNER table, the code booster will inform you that you must create an index on it (to make queries faster) and sure you have to link it (as foreign key to TBL_OWNER)

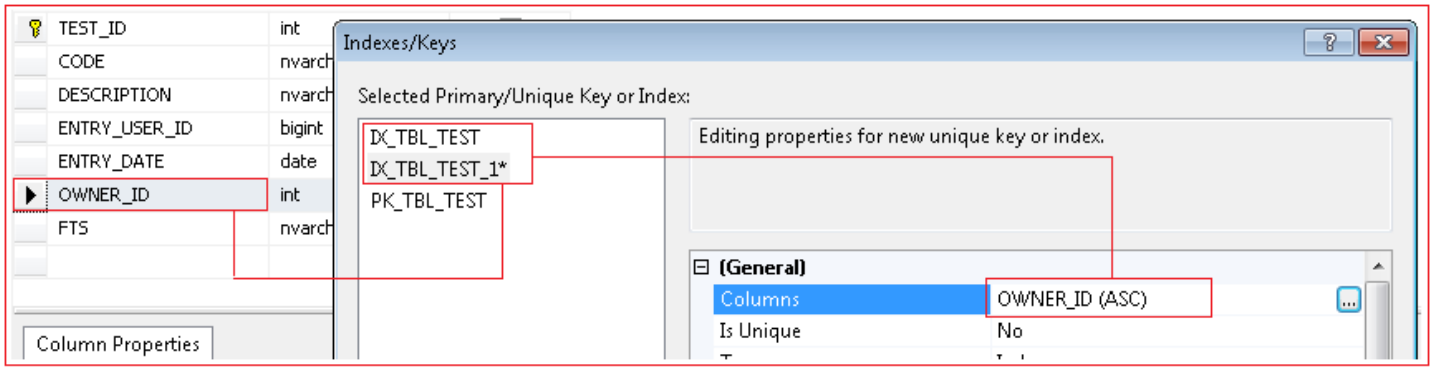


***. Now, if you ask the code booster to generate a patch, it will generate it without any notification**

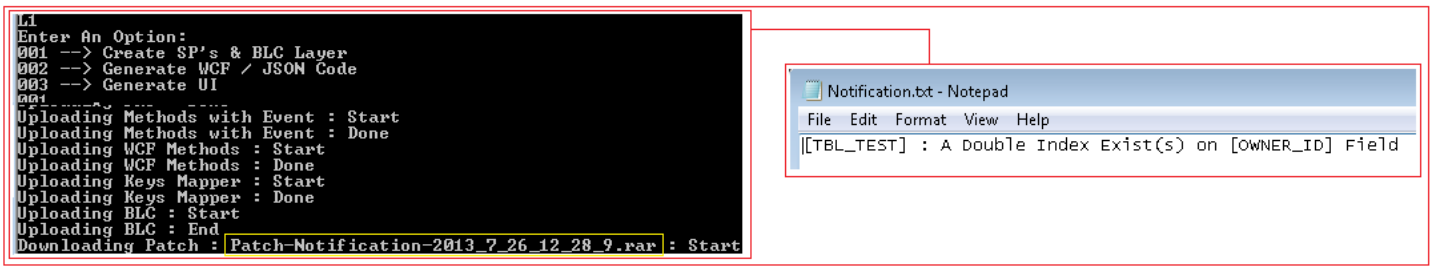


***. A common mistake is committed by developers by creating more than one index on the same table field.**

(In the following picture, we are creating another index on OWNER_ID field beside the one we created before)

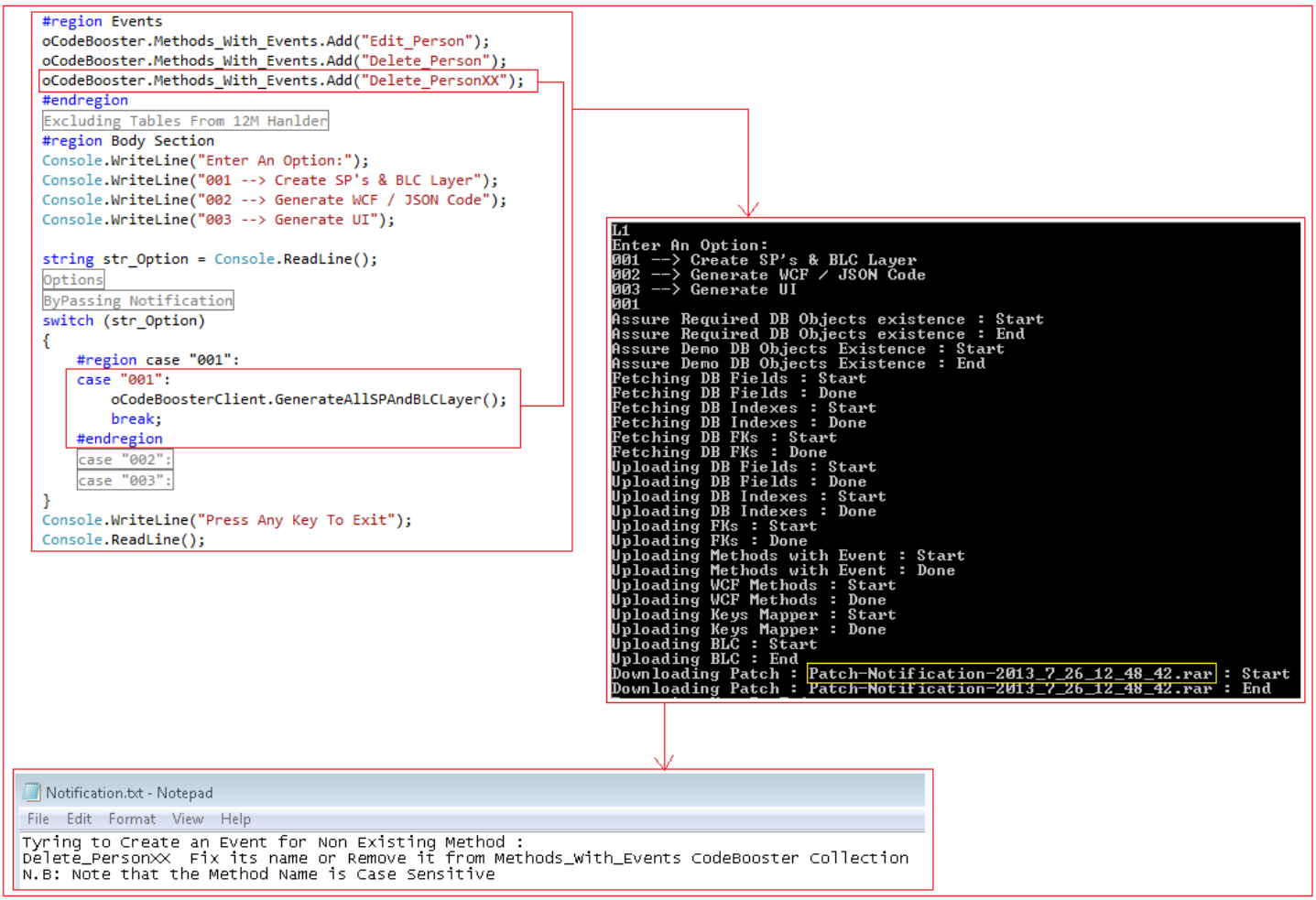


***. If you ask the code booster to generate a patch, it will notify that you are you having a double indexed field.**

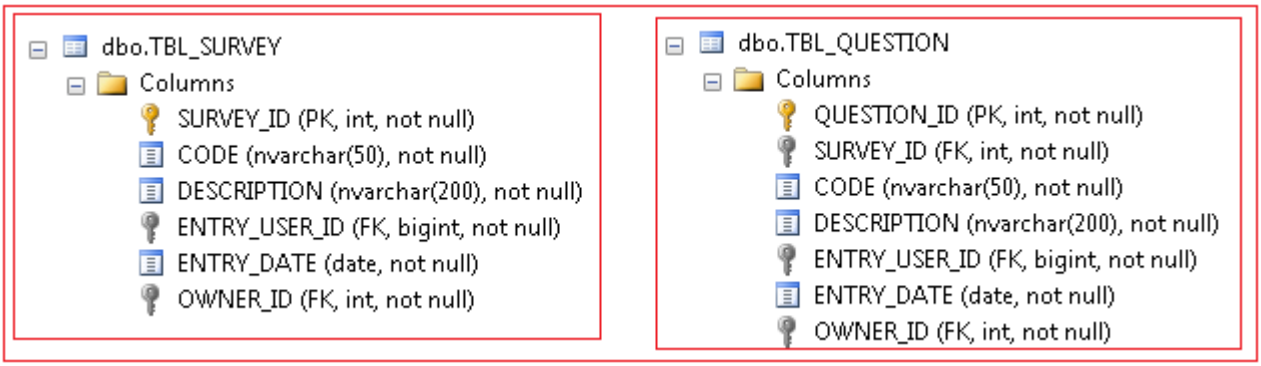


***. In case you ask the code booster engine to create Pre/Post event for a non-existing method (or misspelled)**

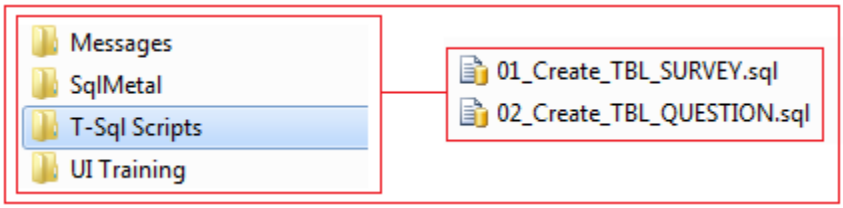
You will get the following notification (N.B: this is a case sensitive checking)



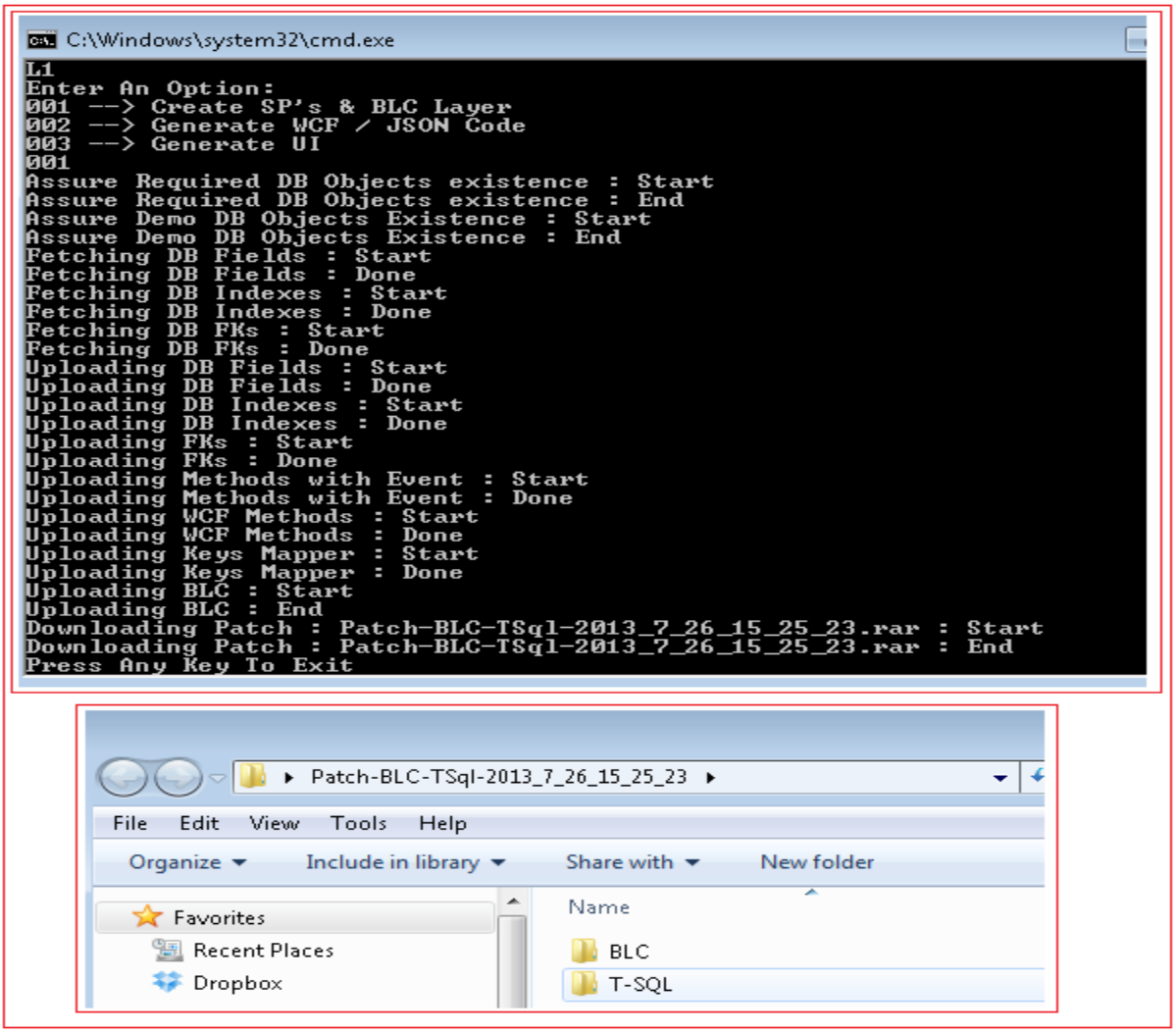
***. Let's create two related tables (Parent/Child) named TBL_SURVEY & TBL_QUESTION**



You will find the corresponding scripts under the T-Sql Scripts folder residing under the "CB" folder.



***. Ask the Code Booster to generate the corresponding patch**



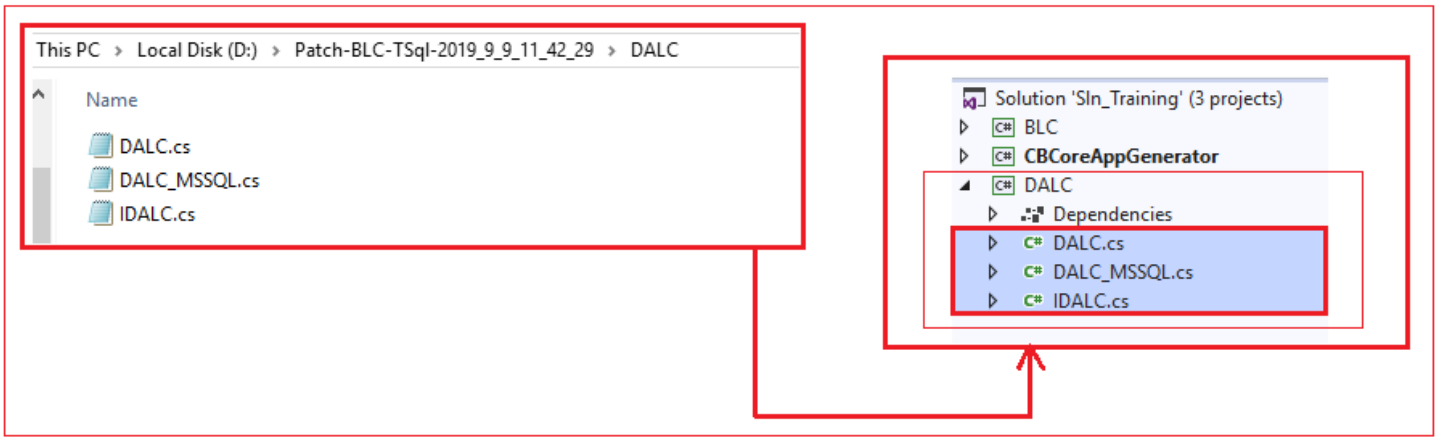
***. Apply the T-Sql Script (GeneratedScript.sql) on your database.**

GeneratedScript.sq...(53)) Executing...

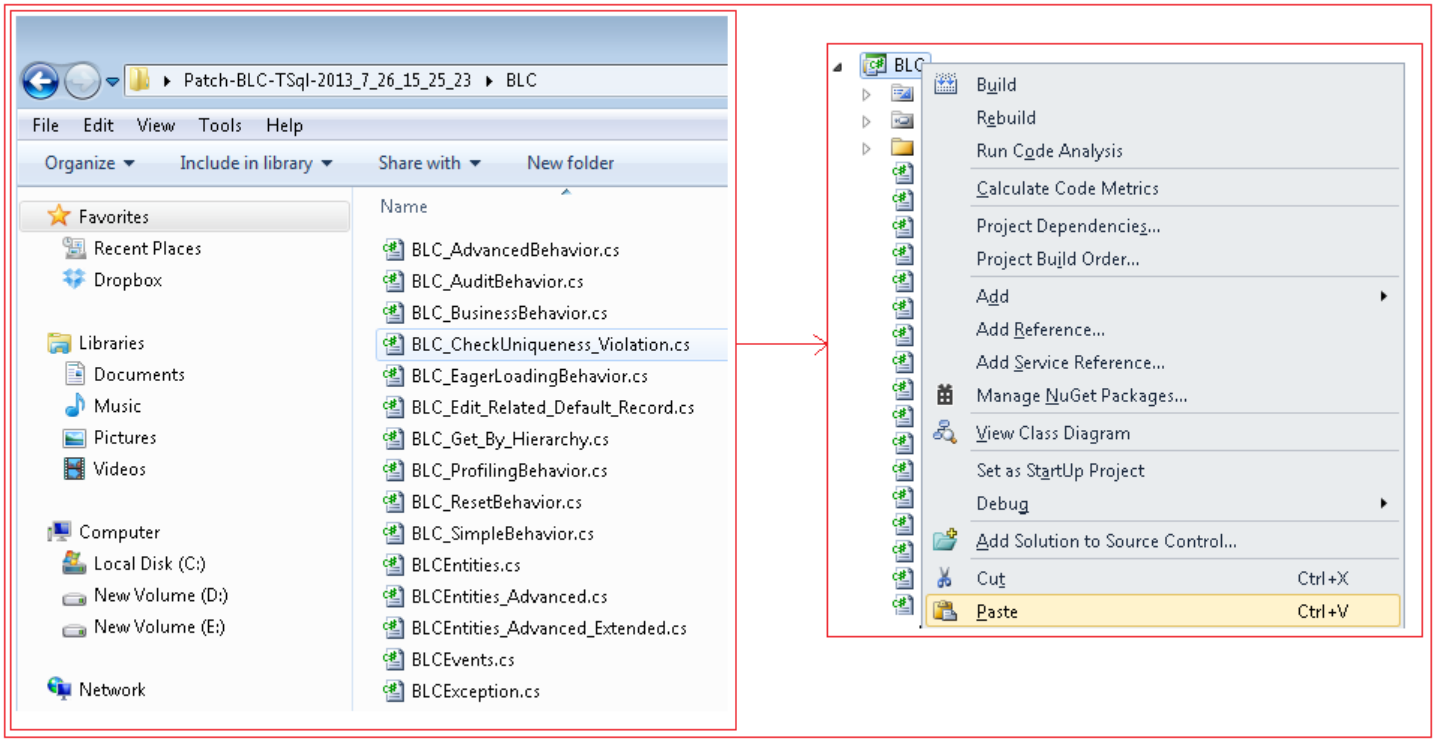
```
--------------------------------------------
DECLARE @V__SPECIFIC_NAME      AS NVARCHAR(200)
DECLARE @V__STATEMENT          AS NVARCHAR(1000)
DECLARE @V__ROUTINE_TYPE       AS NVARCHAR(1000)
SET     @V__SPECIFIC_NAME      = ''
SET     @V__STATEMENT            = ''
SET     @V__ROUTINE_TYPE         = ''
DECLARE ROUTINE_CURSOR CURSOR FOR
SELECT SPECIFIC_NAME , ROUTINE_TYPE
FROM   INFORMATION_SCHEMA.ROUTINES
WHERE  SPECIFIC_NAME LIKE 'UPG_%'
OR     SPECIFIC_NAME LIKE 'UDFG_%'
OPEN ROUTINE_CURSOR
FETCH NEXT FROM ROUTINE_CURSOR INTO @V__SPECIFIC_NAME , @V__ROUTINE_TYPE
WHILE @@FETCH_STATUS = 0
BEGIN
IF (@V__ROUTINE_TYPE = 'PROCEDURE')
BEGIN
   SET @V__STATEMENT = 'DROP PROCEDURE ' + @V__SPECIFIC_NAME
END
IF (@V__ROUTINE_TYPE = 'FUNCTION')
BEGIN
```

Results | Messages

***. Override the files that exist under the DALC folder in the DALC Project**

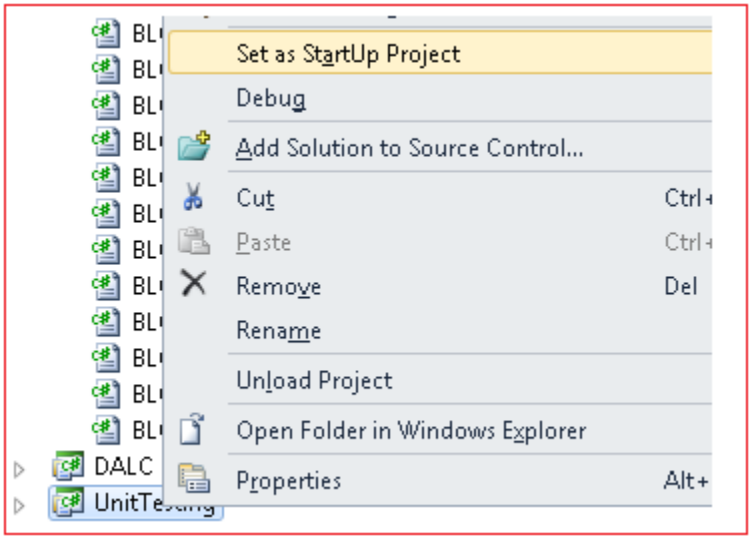


***. Override generated BLC files under the BLC Library project**

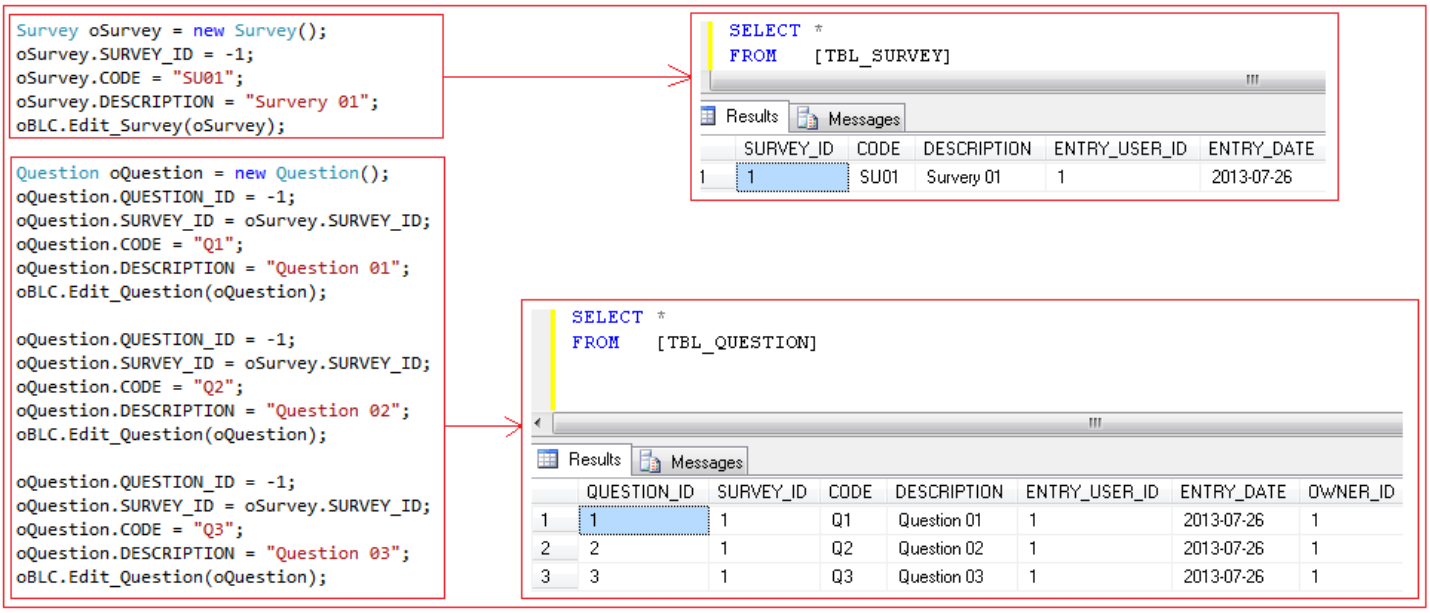


***. Rebuild the BLC & UnitTesting console applications.**

***. Set the UnitTesting console application as the startup project.**



***. Using the UnitTesting console application let us create a survey and under this survey let's create 3 questions.**



```
Survey oSurvey = new Survey();
oSurvey.SURVEY_ID = -1;
oSurvey.CODE = "SU01";
oSurvey.DESCRIPTION = "Survery 01";
oBLC.Edit_Survey(oSurvey);

Question oQuestion = new Question();
oQuestion.QUESTION_ID = -1;
oQuestion.SURVEY_ID = oSurvey.SURVEY_ID;
oQuestion.CODE = "Q1";
oQuestion.DESCRIPTION = "Question 01";
oBLC.Edit_Question(oQuestion);

oQuestion.QUESTION_ID = -1;
oQuestion.SURVEY_ID = oSurvey.SURVEY_ID;
oQuestion.CODE = "Q2";
oQuestion.DESCRIPTION = "Question 02";
oBLC.Edit_Question(oQuestion);

oQuestion.QUESTION_ID = -1;
oQuestion.SURVEY_ID = oSurvey.SURVEY_ID;
oQuestion.CODE = "Q3";
oQuestion.DESCRIPTION = "Question 03";
oBLC.Edit_Question(oQuestion);
```

```
SELECT *
FROM   [TBL_SURVEY]
```

Results | Messages

	SURVEY_ID	CODE	DESCRIPTION	ENTRY_USER_ID	ENTRY_DATE
1	1	SU01	Survery 01	1	2013-07-26

```
SELECT *
FROM   [TBL_QUESTION]
```

Results | Messages

	QUESTION_ID	SURVEY_ID	CODE	DESCRIPTION	ENTRY_USER_ID	ENTRY_DATE	OWNER_ID
1	1	1	Q1	Question 01	1	2013-07-26	1
2	2	1	Q2	Question 02	1	2013-07-26	1
3	3	1	Q3	Question 03	1	2013-07-26	1

***. In order to make the creation of survey and corresponding questions under one transaction check the following [Using TransactionScope]**

```
using (TransactionScope oScope = new TransactionScope())
{
    Survey oSurvey = new Survey();
    oSurvey.SURVEY_ID = -1;
    oSurvey.CODE = "SU01";
    oSurvey.DESCRIPTION = "Survery 01";
    oBLC.Edit_Survey(oSurvey);

    Question oQuestion = new Question();
    oQuestion.QUESTION_ID = -1;
    oQuestion.SURVEY_ID = oSurvey.SURVEY_ID;
    oQuestion.CODE = "Q1";
    oQuestion.DESCRIPTION = "Question 01";
    oBLC.Edit_Question(oQuestion);

    oQuestion.QUESTION_ID = -1;
    oQuestion.SURVEY_ID = oSurvey.SURVEY_ID;
    oQuestion.CODE = "Q2";
    oQuestion.DESCRIPTION = "Question 02";
    oBLC.Edit_Question(oQuestion);

    oQuestion.QUESTION_ID = -1;
    oQuestion.SURVEY_ID = oSurvey.SURVEY_ID;
    oQuestion.CODE = "Q3";
    oQuestion.DESCRIPTION = "Question 03";
    oBLC.Edit_Question(oQuestion);

    oScope.Complete();
}
```

***. Let's retrieve all questions related to a certain survey.**

Retrieving all Questions related to a certain Survery

```
Params_Get_Question_By_SURVEY_ID oParams_Get_Question_By_SURVEY_ID = new Params_Get_Question_By_SURVEY_ID();
oParams_Get_Question_By_SURVEY_ID.SURVEY_ID = 1;
List<Question> oList_Question = oBLC.Get_Question_By_SURVEY_ID(oParams_Get_Question_By_SURVEY_ID);
```

```
if (oList_Question != null)
{
    foreach (var oRow_Question in oList_Question)
    {
        Console.WriteLine(oRow_Question.DESCRIPTION);
    }
}
```

Print out on console the DESCRIPTION of each question

***. Reset Topology:**

***. You can ask the Code booster to allow you to save a survey and the corresponding collection of questions in one shot.**

***. To do so, there is a collection called "List_Reset_Topology" that should be initialized as the following.**

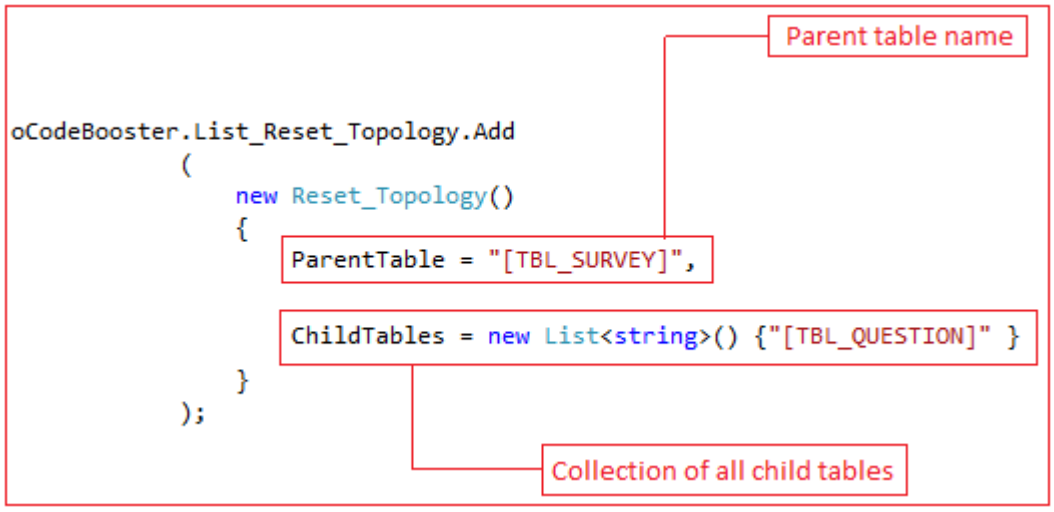
```
oCodeBooster.List_Reset_Topology = new List<Reset_Topology>();
```

***. Then you must add to the previously initialized collection a new instance of type Reset_Topology**

***. The Reset_Toplogy object has the following properties:**

***. ParentTable: The Parent table name (In our case TBL_SURVEY)**

***. ChidlTables: a collection mentioning all child tables (In our case it contains only TBL_QUESTION)**

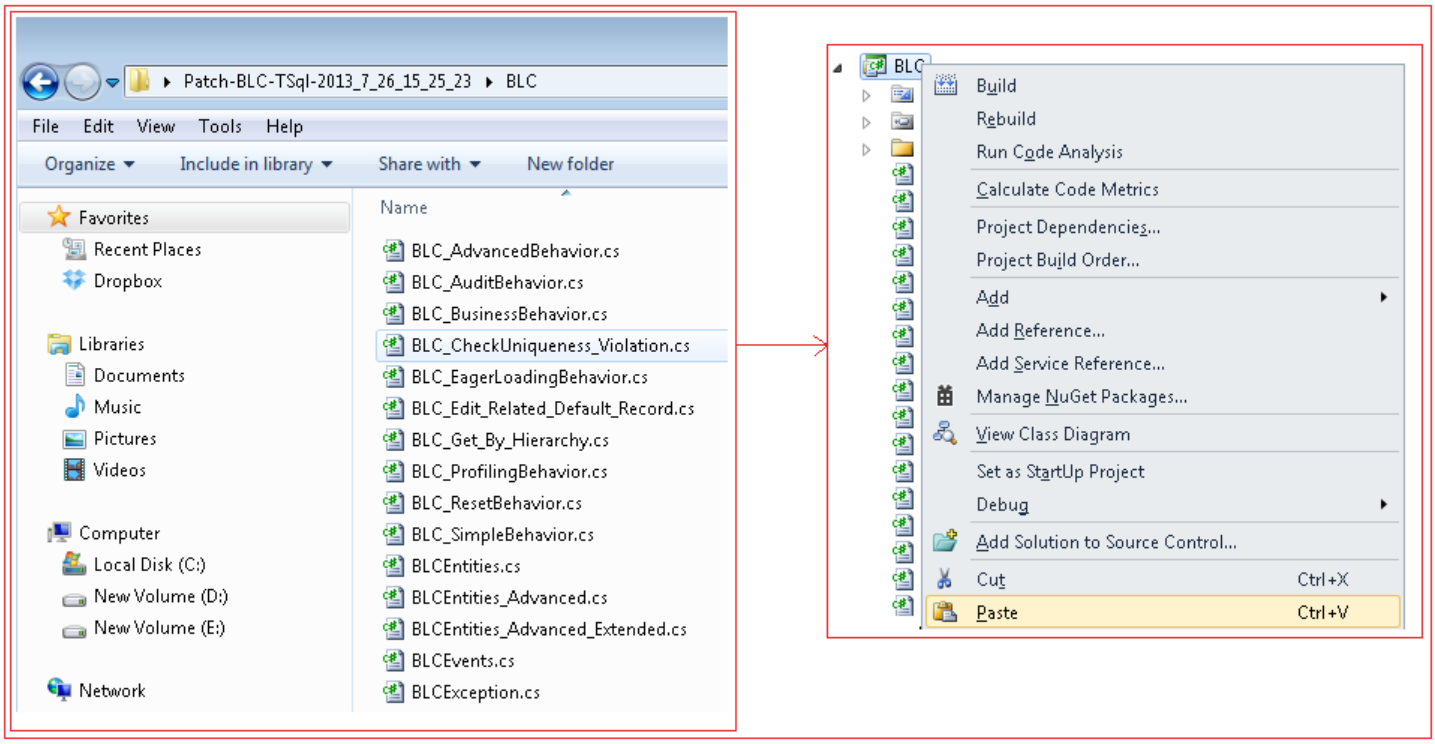


***. This is how the Program.cs (under the AppGenerator console application) should look while asking the code booster to generate the ResetTopology behavior**

```
#region ResetTopology
oCodeBooster.List_Reset_Topology = new List<Reset_Topology>();

oCodeBooster.List_Reset_Topology.Add
            (
                new Reset_Topology()
                {
                    ParentTable = "[TBL_SURVEY]",
                    ChildTables = new List<string>() {"[TBL_QUESTION]" }
                }
            );
#endregion
#region Body Section
Console.WriteLine("Enter An Option:");
Console.WriteLine("001 --> Create SP's & BLC Layer");
Console.WriteLine("002 --> Generate WCF / JSON Code");
Console.WriteLine("003 --> Generate UI");

string str_Option = Console.ReadLine();
Options
ByPassing Notification
switch (str_Option)
{
    #region case "001":
    case "001":
        oCodeBoosterClient.GenerateAllSPAndBLCLayer();
        break;
    #endregion
    case "002":
    case "003":
}
```

```
L1
Enter An Option:
001 --> Create SP's & BLC Layer
002 --> Generate WCF / JSON Code
003 --> Generate UI
001
Assure Required DB Objects existence : Start
Assure Required DB Objects existence : End
Assure Demo DB Objects Existence : Start
Assure Demo DB Objects Existence : End
Fetching DB Fields : Start
Fetching DB Fields : Done
Fetching DB Indexes : Start
Fetching DB Indexes : Done
Fetching DB FKs : Start
Fetching DB FKs : Done
Uploading DB Fields : Start
Uploading DB Fields : Done
Uploading DB Indexes : Start
Uploading DB Indexes : Done
Uploading FKs : Start
Uploading FKs : Done
Uploading Methods with Event : Start
Uploading Methods with Event : Done
Uploading WCF Methods : Start
Uploading WCF Methods : Done
Uploading Keys Mapper : Start
Uploading Keys Mapper : Done
Uploading BLC : Start
Uploading BLC : End
Downloading Patch : Patch-BLC-TSql-2013_7_26_16_58_51.rar
Downloading Patch : Patch-BLC-TSql-2013_7_26_16_58_51.rar
```

***. Override generated BLC files under the BLC Library project**

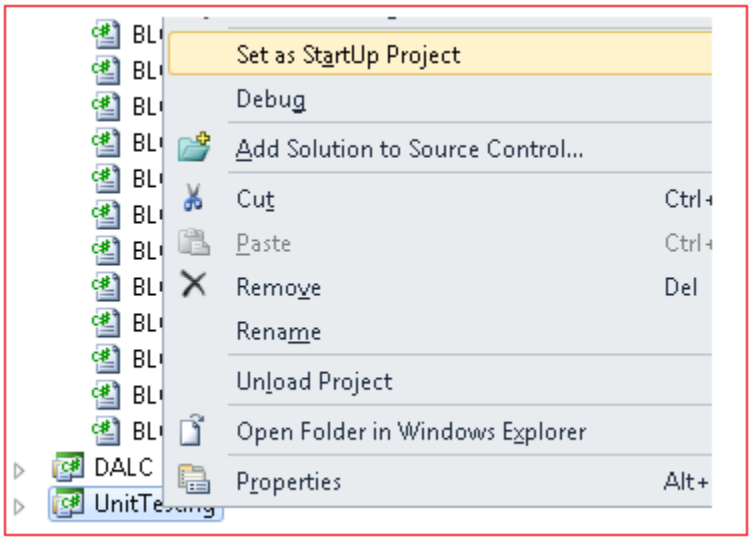


***. In the BLC_Initial.cs file and specifically in SubscribeToEvents method add the following line (as per picture)**

```
#region Subscribe To Events
public void SubscribeToEvents()
{
    Declaration And Initialization Section.
    #region Body Section.

    Initialize Eager Loading Mechanism();
    Initialize_Reset_Mechanism();

    this.OnPreEvent_Edit_Person +=new PreEvent_Handler_Edit_Person(BLC_OnPreEvent_Edit_Person);
    this.OnPostEvent_Edit_Person += new PostEvent_Handler_Edit_Person(BLC_OnPostEvent_Edit_Person);
    this.OnPreEvent_Edit_Person += new PreEvent_Handler_Edit_Person(BLC_OnPreEvent_Edit_Person_02);
    this.OnPreEvent_Delete_Person += new PreEvent_Handler_Delete_Person(BLC_OnPreEvent_Delete_Person);
    #endregion
}
#endregion
```

***. Rebuild the BLC & UnitTesting console applications.**

***. Set the UnitTesting console application as the startup project.**



***. In the UnitTesting Console application, create a survey object and fill the collection of questions then in one short by calling Edit_Survey creates the survey itself and the corresponding questions.**

```
Survey oSurvey = new Survey();
oSurvey.SURVEY_ID = -1;
oSurvey.CODE = "SU02";
oSurvey.DESCRIPTION = "Survery 02";
oSurvey.My_Question = new List<Question>();
oSurvey.My_Question.Add(new Question() { QUESTION_ID= -1,  CODE = "Q21", DESCRIPTION = "Question 21" });
oSurvey.My_Question.Add(new Question() { QUESTION_ID = -1, CODE = "Q22", DESCRIPTION = "Question 22" });
oSurvey.My_Question.Add(new Question() { QUESTION_ID = -1, CODE = "Q23", DESCRIPTION = "Question 23" });

oBLC.Edit_Survey(oSurvey);
```

```
SELECT *
FROM   [TBL_SURVEY]
```

Results | Messages

	SURVEY_ID	CODE	DESCRIPTION	ENTRY_USER_ID	ENTRY_DATE	OWNER_ID
1	1	SU01	Survery 01	1	2013-07-26	1
2	2	SU02	Survery 02	1	2013-07-26	1

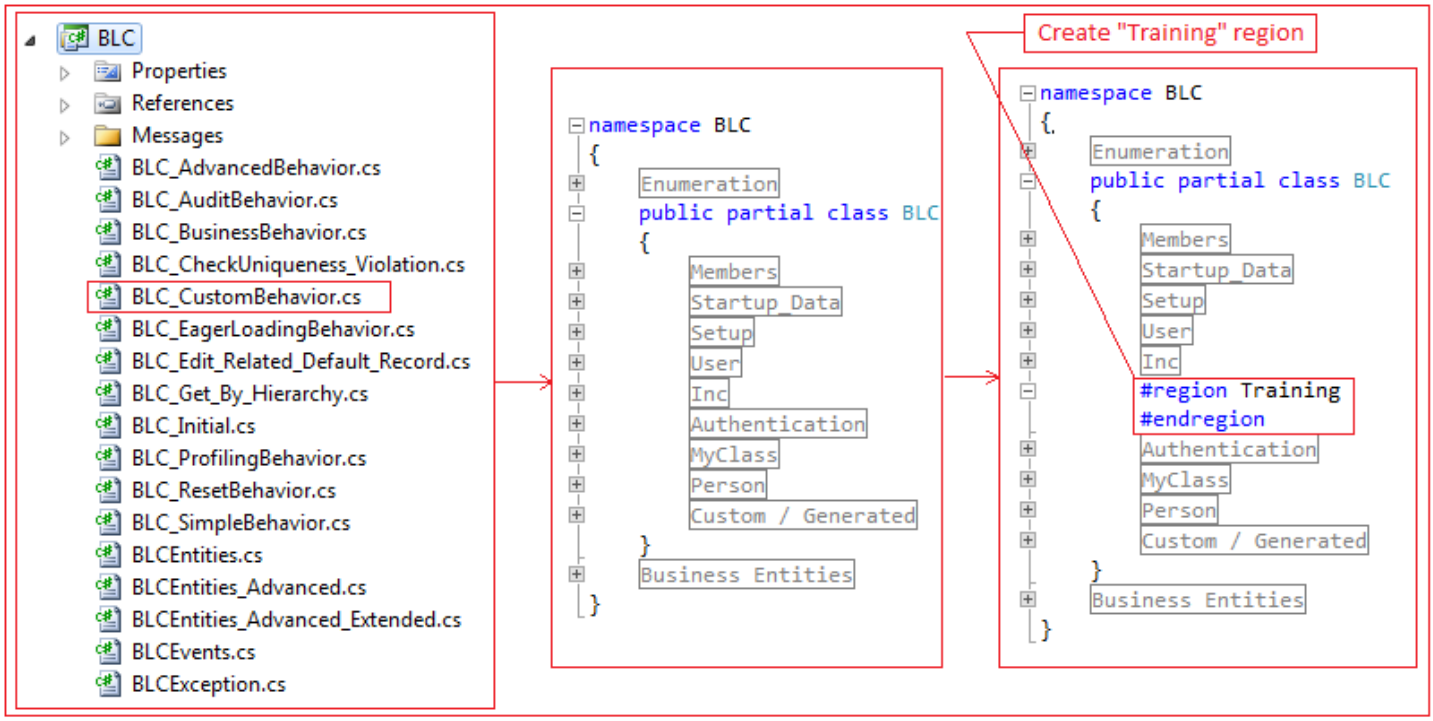
```
SELECT *
FROM   [TBL_QUESTION]
```

Results | Messages

	QUESTION_ID	SURVEY_ID	CODE	DESCRIPTION	ENTRY_USER_ID	ENTRY_DATE	OWNER_I
	1	1	Q1	Question 01	1	2013-07-26	1
	2	1	Q2	Question 02	1	2013-07-26	1
	3	1	Q3	Question 03	1	2013-07-26	1
	4	2	Q21	Question 21	1	2013-07-26	1
	5	2	Q22	Question 22	1	2013-07-26	1
	6	2	Q23	Question 23	1	2013-07-26	1

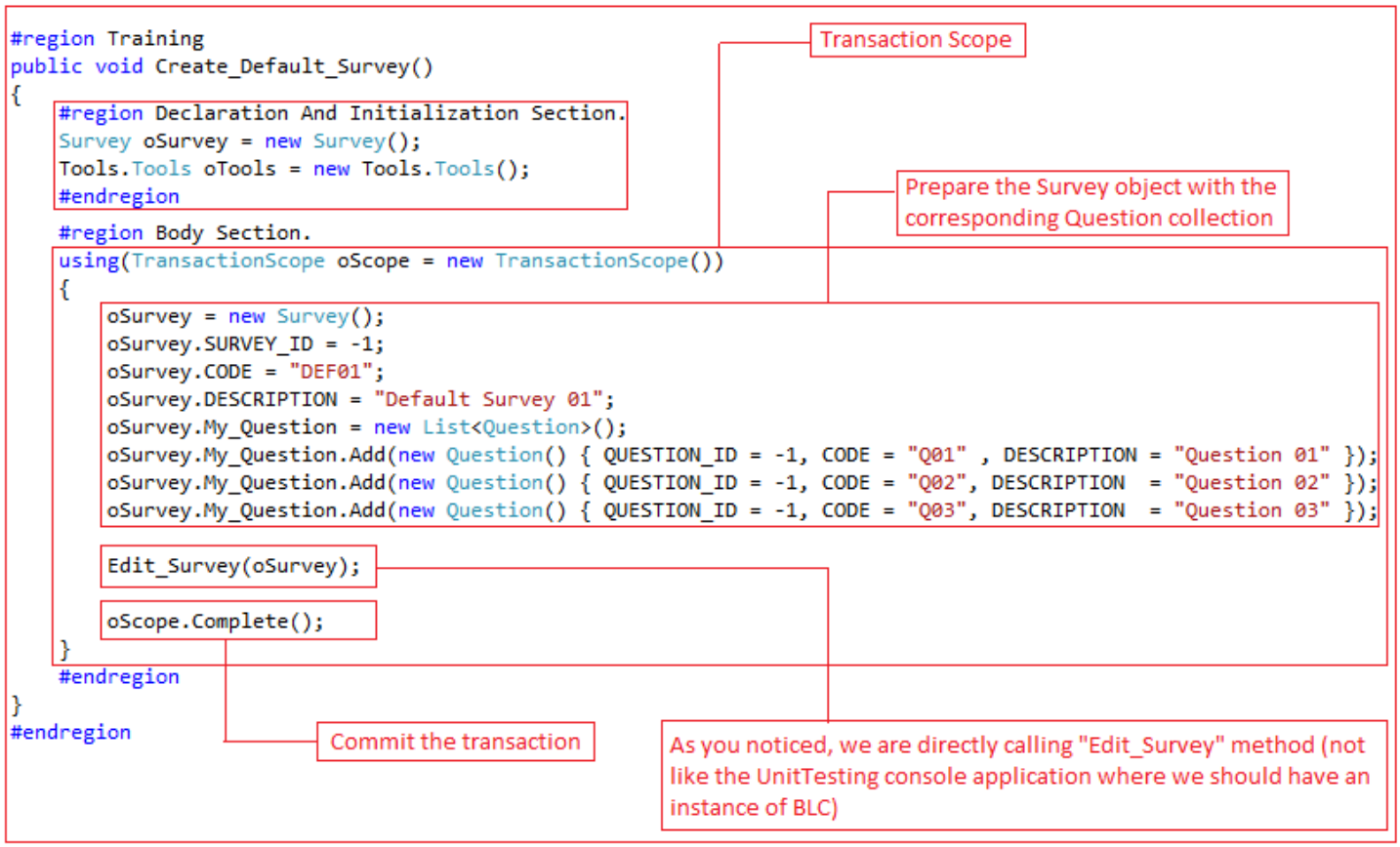
*. Till now we have only used generated methods (ex: Edit_Person, Delete_Person, Get_Person_By_OWNER_ID)

*. Under the BLC project and specifically in BLC_CustomBehavior.cs file create a region called “Training”

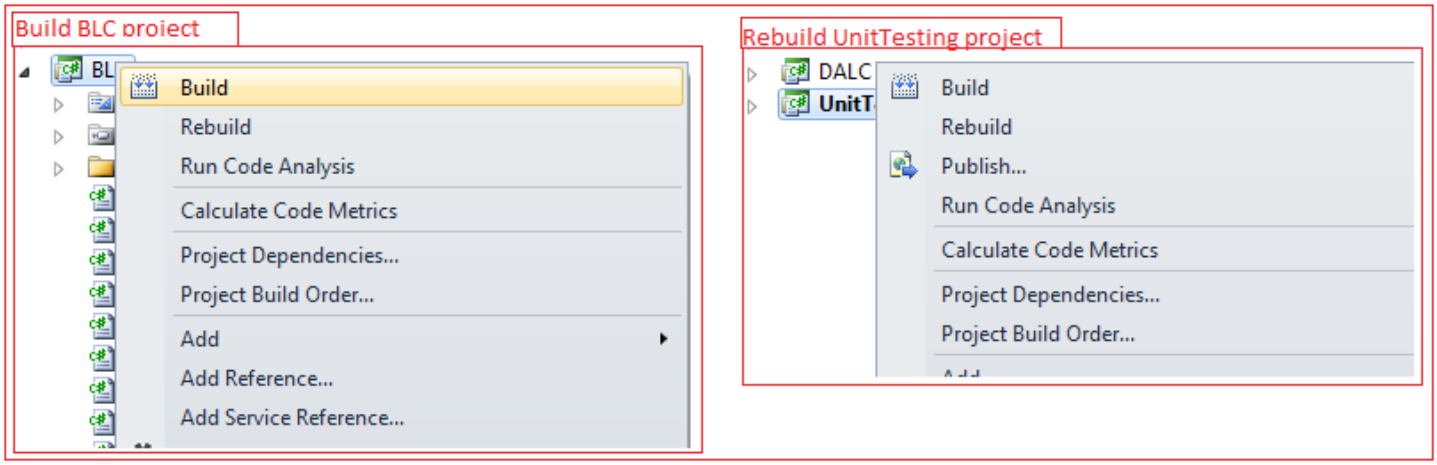


*. Create a method called “Create_Default_Survey”:

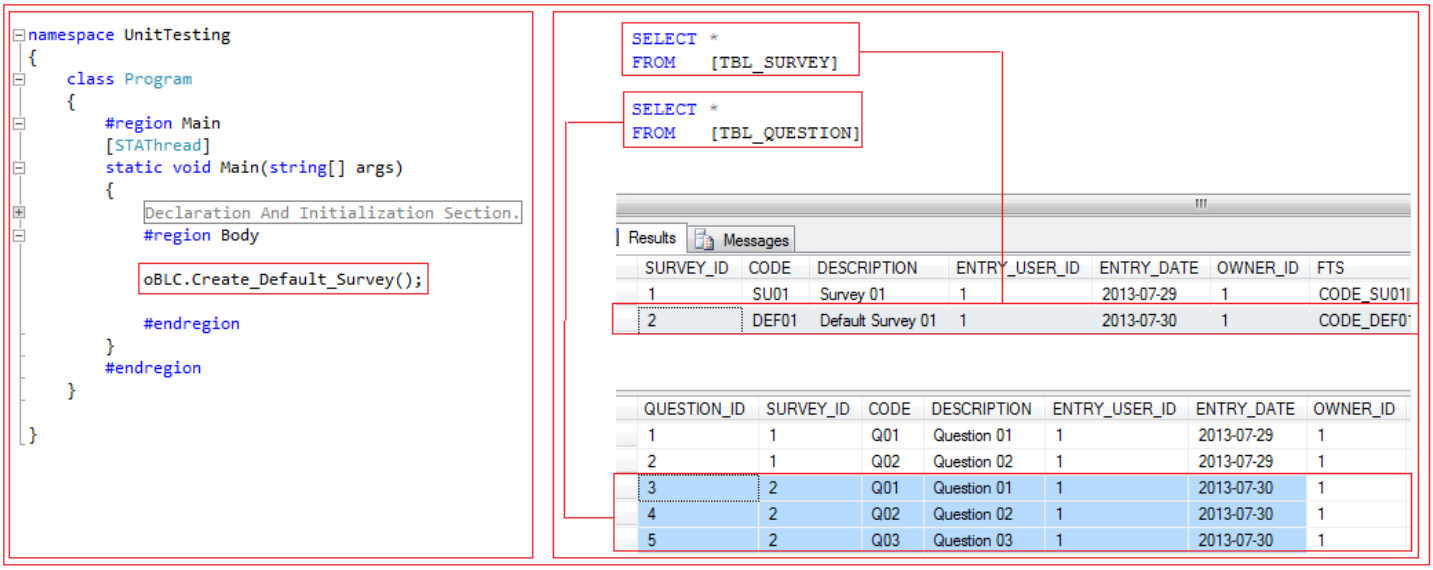
It does not accept any parameter and it creates s default survey with 3 questions.



*. Build the BLC Project and Rebuild the UnitTesting Console application to test the previously created BLC method.



*. In the UnitTesting console application and specifically in the “Main” method let’s call Create_Default_Survey method



```
namespace UnitTesting
{
    class Program
    {
        #region Main
        [STAThread]
        static void Main(string[] args)
        {
            Declaration And Initialization Section.
            #region Body

            oBLC.Create_Default_Survey();

            #endregion
        }
        #endregion
    }
}
```

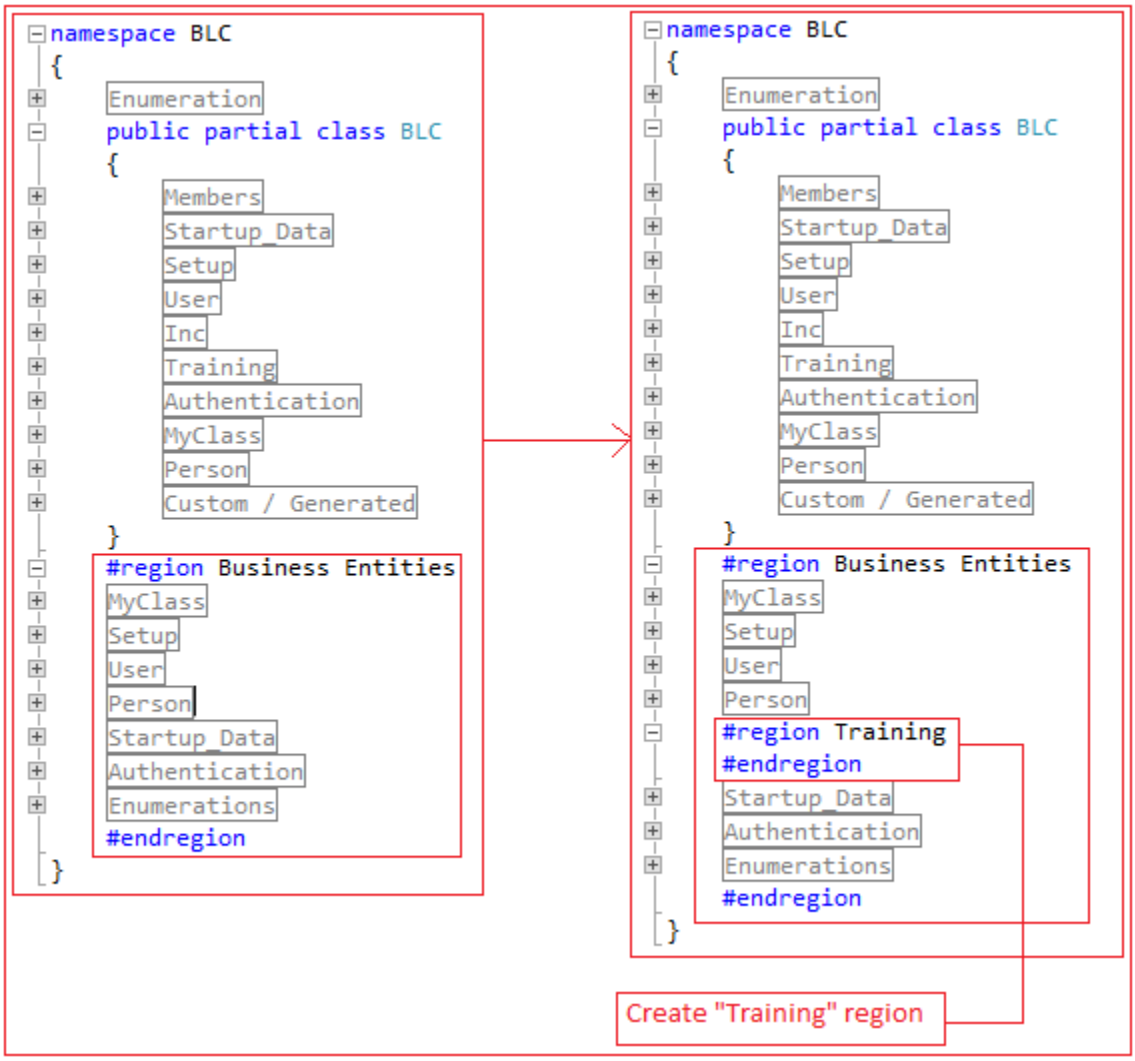
```
SELECT *
FROM   [TBL_SURVEY]

SELECT *
FROM   [TBL_QUESTION]
```

Results | Messages

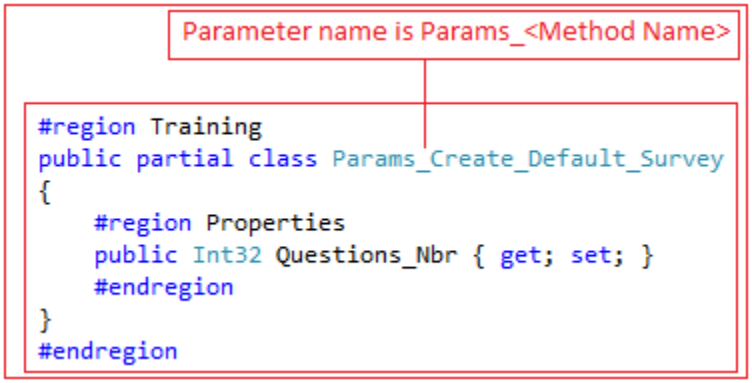
SURVEY_ID	CODE	DESCRIPTION	ENTRY_USER_ID	ENTRY_DATE	OWNER_ID	FTS
1	SU01	Survey 01	1	2013-07-29	1	CODE_SU01
2	DEF01	Default Survey 01	1	2013-07-30	1	CODE_DEF0

QUESTION_ID	SURVEY_ID	CODE	DESCRIPTION	ENTRY_USER_ID	ENTRY_DATE	OWNER_ID
1	1	Q01	Question 01	1	2013-07-29	1
2	1	Q02	Question 02	1	2013-07-29	1
3	2	Q01	Question 01	1	2013-07-30	1
4	2	Q02	Question 02	1	2013-07-30	1
5	2	Q03	Question 03	1	2013-07-30	1

*. In the BLC project and specifically in BLC_CustomBehavior.cs file, you have to create a region called “Training” under “Business Entities” region as the following.



*. Create a new type called Params_Create_Default_Survey



*. Now change the signature of Create_Default_Survey method as the following.

```
#region Training
public void Create_Default_Survey(Params_Create_Default_Survey i_Params_Create_Default_Survey)
{
    #region Declaration And Initialization Section.
    Survey oSurvey = new Survey();
    Tools.Tools oTools = new Tools.Tools();
    #endregion
    #region Body Section.
    using(TransactionScope oScope = new TransactionScope())
    {
        oSurvey = new Survey();
        oSurvey.SURVEY_ID = -1;
        oSurvey.CODE = "DEF01";
        oSurvey.DESCRIPTION = "Default Survey 01";
        oSurvey.My_Question = new List<Question>();
        oSurvey.My_Question.Add(new Question() { QUESTION_ID = -1, CODE = "Q01" , DESCRIPTION = "Question 01" });
        oSurvey.My_Question.Add(new Question() { QUESTION_ID = -1, CODE = "Q02", DESCRIPTION  = "Question 02" });
        oSurvey.My_Question.Add(new Question() { QUESTION_ID = -1, CODE = "Q03", DESCRIPTION  = "Question 03" });

        Edit_Survey(oSurvey);

        oScope.Complete();
    }
    #endregion
}
#endregion
```

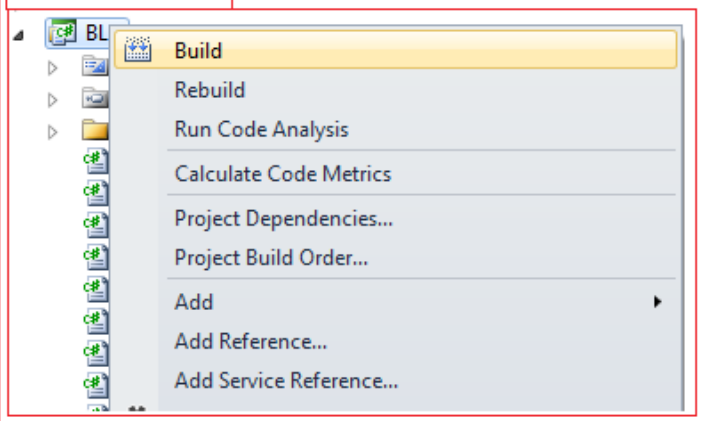
*. We will use the input parameter to create as much questions as much is “Questions_Nbr” equals

```
public void Create_Default_Survey(Params_Create_Default_Survey i_Params_Create_Default_Survey)
{
    Declaration And Initialization Section.
    #region Body Section.
    using(TransactionScope oScope = new TransactionScope())
    {
        oSurvey = new Survey();
        oSurvey.SURVEY_ID = -1;
        oSurvey.CODE = "DEF01";
        oSurvey.DESCRIPTION = "Default Survey 01";
        oSurvey.My_Question = new List<Question>();

        for (int i = 0; i < i_Params_Create_Default_Survey.Questions_Nbr; i++)
        {
            oSurvey.My_Question.Add
                (
                    new Question()
                    {   QUESTION_ID = -1,
                        CODE = string.Format("QX{0}",i.ToString()),
                        DESCRIPTION = string.Format("Question {0}",i.ToString())
                    }
                );
        }

        Edit_Survey(oSurvey);

        oScope.Complete();
    }
    #endregion
}
```

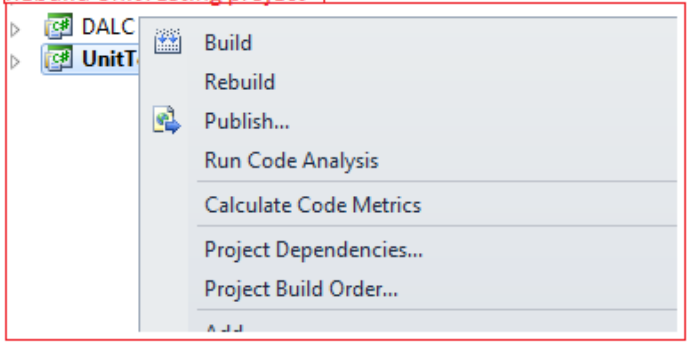
Create as much questions as specified in Questions_Nbr property

*. Build the BLC Project and Rebuild the UnitTesting Console application to test the previously created BLC method

Build BLC project



Rebuild UnitTesting project



*. In the UnitTesting console application and specifically in the "Main" method let's call Create_Default_Survey method

Prepare corresponding parameter before calling method

```
namespace UnitTesting
{
    class Program
    {
        #region Main
        [STAThread]
        static void Main(string[] args)
        {
            Declaration And Initialization Section.
            #region Body

            Params_Create_Default_Survey oParams_Create_Default_Survey = new Params_Create_Default_Survey();
            oParams_Create_Default_Survey.Questions_Nbr = 5;
            oBLC.Create_Default_Survey(oParams_Create_Default_Survey);

            #endregion
        }
        #endregion
    }

}
```

As you noticed, even if the method is accepting only one primitive type (Int32 in this case) we created a complex input parameter in order to not change the method signature for any additional parameter we might need in the future

```
SELECT *
FROM   [TBL_SURVEY]

SELECT *
FROM   [TBL_QUESTION]
```

Results | Messages

	SURVEY_ID	CODE	DESCRIPTION	ENTRY_USER_ID	ENTRY_DATE	OWNER_ID	FTS
1	4	DEF01	Default Survey 01	1	2013-07-30	1	CODE_DEF0

	QUESTION_ID	SURVEY_ID	CODE	DESCRIPTION	ENTRY_USER_ID	ENTRY_DATE	OWNER_ID
1	26	4	QX0	Question 0	1	2013-07-30	1
2	27	4	QX1	Question 1	1	2013-07-30	1
3	28	4	QX2	Question 2	1	2013-07-30	1
4	29	4	QX3	Question 3	1	2013-07-30	1
5	30	4	QX4	Question 4	1	2013-07-30	1

***. EagerLoading: You can ask the code booster that you want, while retrieving a certain survey, to directly retrieve the corresponding questions in one shot.**

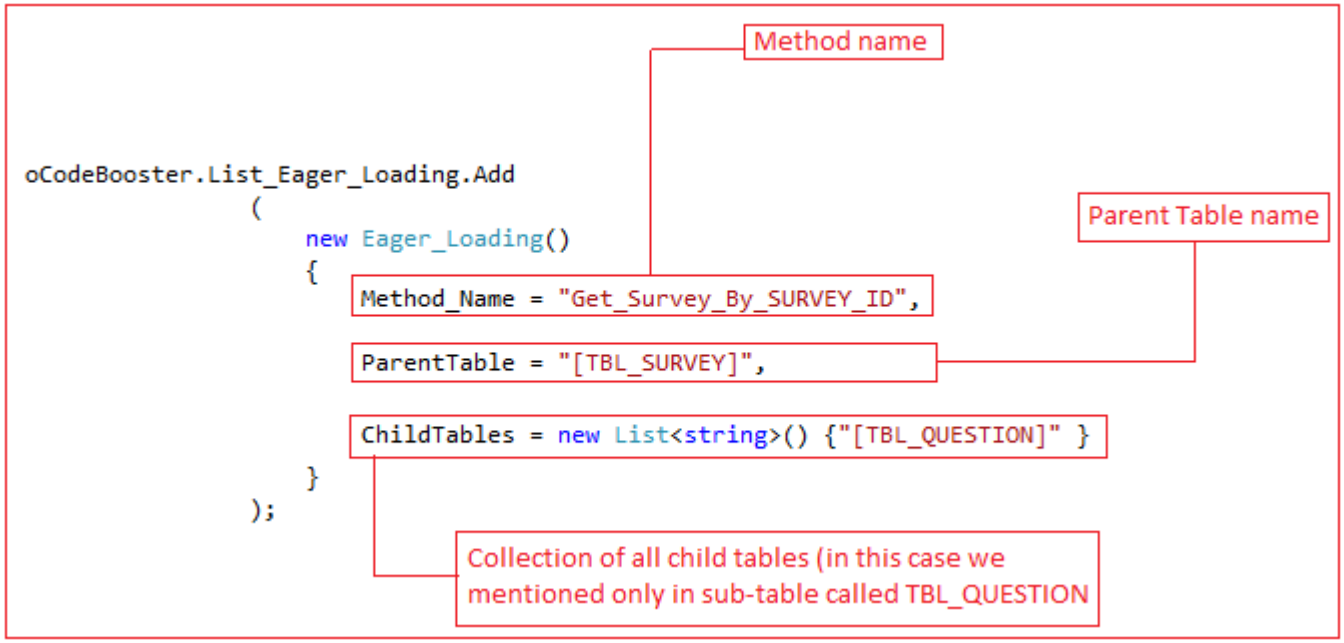
***. To do so, there is a collection called "List_Eager_Loading" that should be initialized as the following.**

```
oCodeBooster.List_Eager_Loading = new List<Eager_Loading>();
```

***. Then you must add to the previously initialized collection a new instance of type Eager_Loading**

The Eager_Loading object has the following properties:

1. **Method_Name : this is the name of the method where the eager loading will work**
2. **ParentTable : The main table to be returned (in our sample , this is TBL_SURVEY table)**
3. **ChildTables: this is a collection of all sub-tables to be returned with TBL_SURVEY table.**



("Questions" will be returned as a collection property under the survey type).

***. This is how the Program.cs (under the AppGenerator console application) should look while asking the code booster to generate the EagerLoading behavior.**

```
oCodeBooster.List_Eager_Loading = new List<Eager_Loading>();

oCodeBooster.List_Eager_Loading.Add
                (
                    new Eager_Loading()
                    {
                        Method_Name = "Get_Survey_By_SURVEY_ID",
                        ParentTable = "[TBL_SURVEY]",
                        ChildTables = new List<string>() {"[TBL_QUESTION]" }
                    }
                );

#endregion
ResetTopology
#region Body Section
Console.WriteLine("Enter An Option:");
Console.WriteLine("001 --> Create SP's & BLC Layer");
Console.WriteLine("002 --> Generate WCF / JSON Code");
Console.WriteLine("003 --> Generate UI");

string str_Option = Console.ReadLine();
Options
ByPassing Notification
switch (str_Option)
{
    #region case "001":
    case "001":
        oCodeBoosterClient.GenerateAllSPAndBLCLayer();
        break;
    #endregion
```

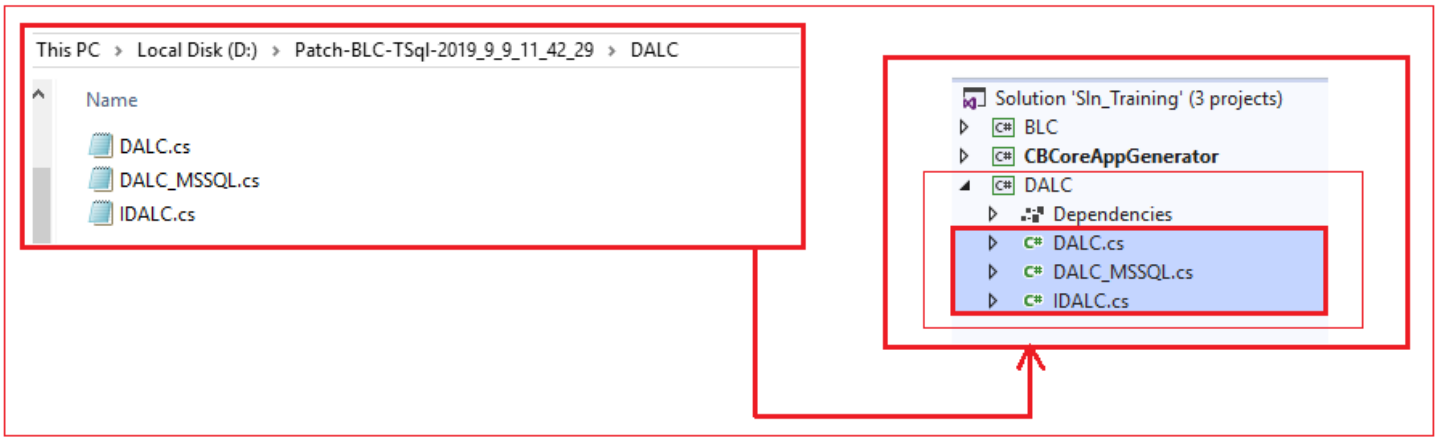
***. Apply the T-Sql Script (GeneratedScript.sql) on your database.**

GeneratedScript.sq...(53)) Executing...

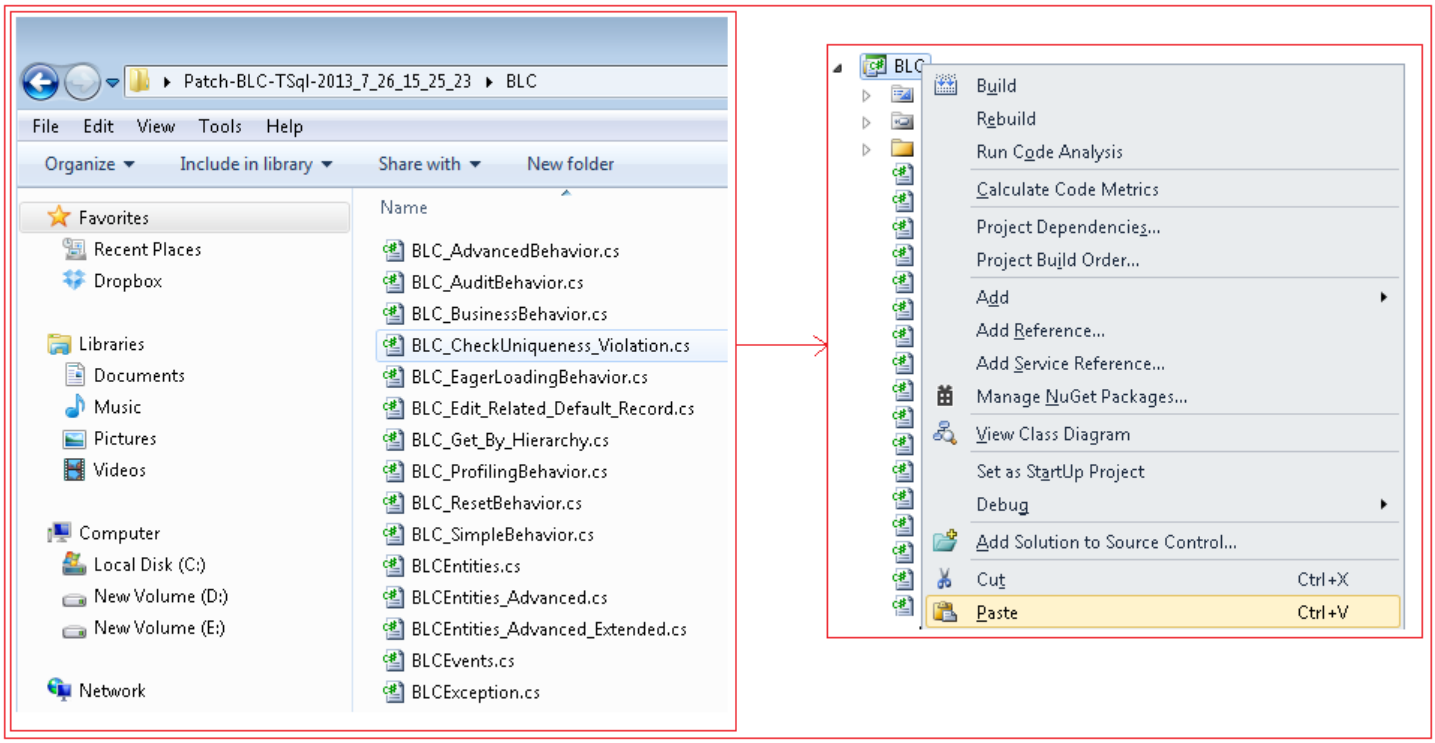
```
--------------------------------------------
DECLARE @V__SPECIFIC_NAME      AS NVARCHAR(200)
DECLARE @V__STATEMENT          AS NVARCHAR(1000)
DECLARE @V__ROUTINE_TYPE       AS NVARCHAR(1000)
SET     @V__SPECIFIC_NAME      = ''
SET     @V__STATEMENT            = ''
SET     @V__ROUTINE_TYPE         = ''
DECLARE ROUTINE_CURSOR CURSOR FOR
SELECT SPECIFIC_NAME , ROUTINE_TYPE
FROM   INFORMATION_SCHEMA.ROUTINES
WHERE  SPECIFIC_NAME LIKE 'UPG_%'
OR     SPECIFIC_NAME LIKE 'UDFG_%'
OPEN ROUTINE_CURSOR
FETCH NEXT FROM ROUTINE_CURSOR INTO @V__SPECIFIC_NAME , @V__ROUTINE_TYPE
WHILE @@FETCH_STATUS = 0
BEGIN
IF (@V__ROUTINE_TYPE = 'PROCEDURE')
BEGIN
   SET @V__STATEMENT = 'DROP PROCEDURE ' + @V__SPECIFIC_NAME
END
IF (@V__ROUTINE_TYPE = 'FUNCTION')
BEGIN
```

Results | Messages

***. Override the files that exist under the DALC folder in the DALC Project**



***. Override generated BLC files under the BLC Library project**

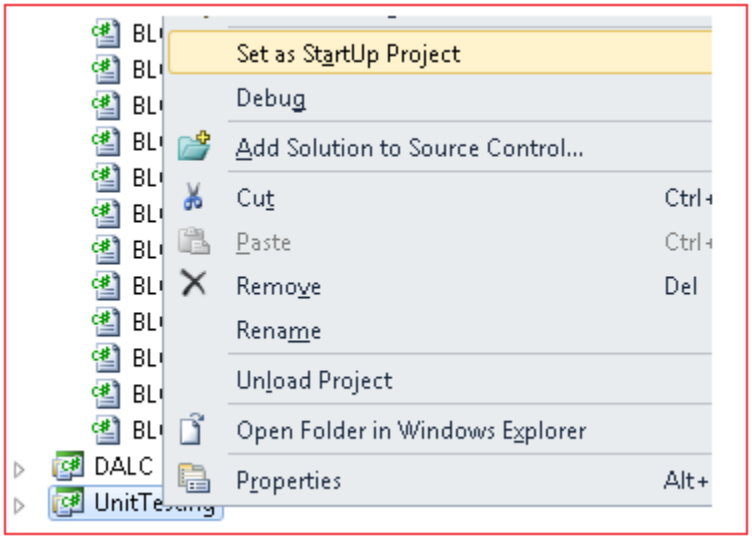


***. In the BLC_Initial.cs file and specifically in SubscribeToEvents method add the following line (as per picture)**

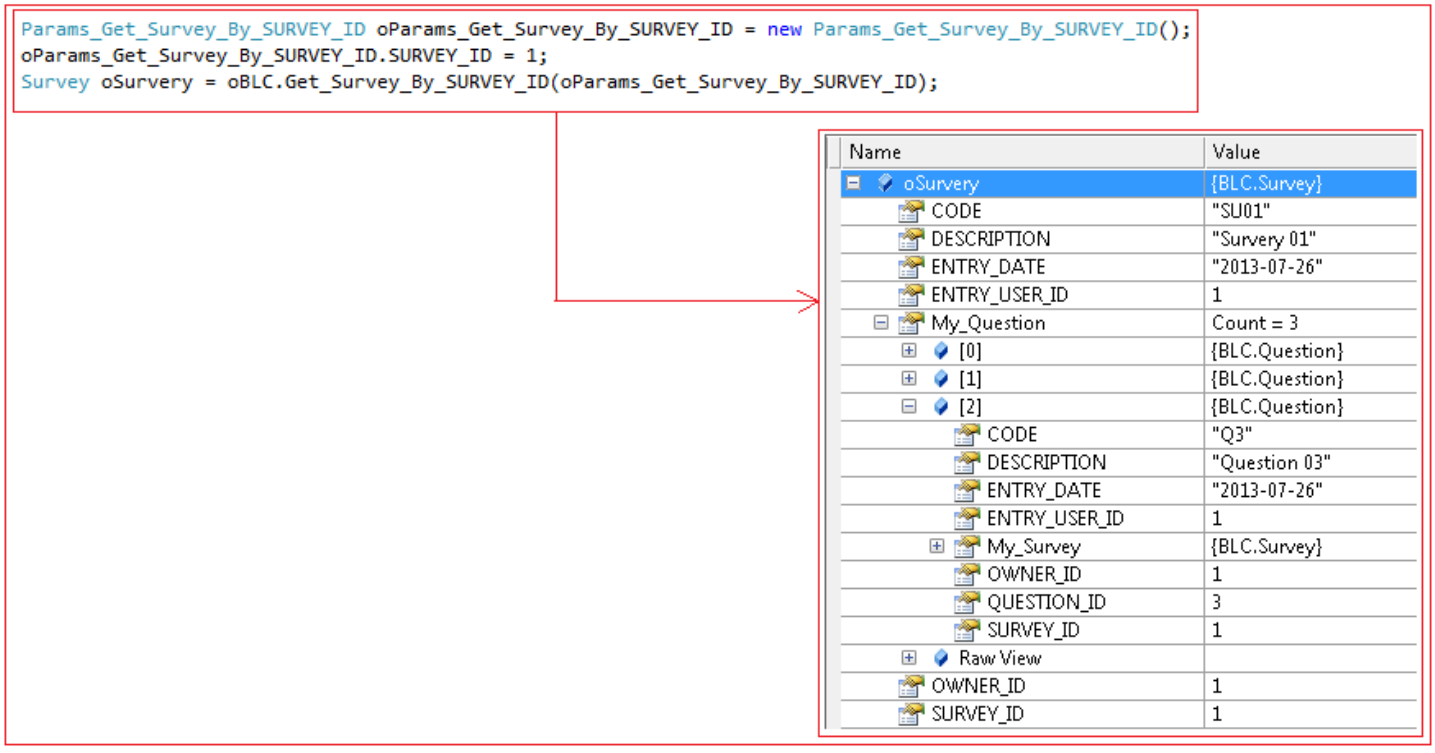
```
#region Subscribe To Events
public void SubscribeToEvents()
{
    Declaration And Initialization Section.
    #region Body Section.

    Initialize_Eager_Loading_Mechanism();

    this.OnPreEvent_Edit_Person +=new PreEvent_Handler_Edit_Person(BLC_OnPreEvent_Edit_Person);
    this.OnPostEvent_Edit_Person += new PostEvent_Handler_Edit_Person(BLC_OnPostEvent_Edit_Person);
    this.OnPreEvent_Edit_Person += new PreEvent_Handler_Edit_Person(BLC_OnPreEvent_Edit_Person_02);
    this.OnPreEvent_Delete_Person += new PreEvent_Handler_Delete_Person(BLC_OnPreEvent_Delete_Person);
    #endregion
}
```

***. Rebuild the BLC & UnitTesting console applications.**

***. Set the UnitTesting console application as the startup project.**



***. In the UnitTesting console application, retrieve the Survey by ID and you noticed that related questions will be retrieved in one shot.**



```
Params_Get_Survey_By_SURVEY_ID oParams_Get_Survey_By_SURVEY_ID = new Params_Get_Survey_By_SURVEY_ID();
oParams_Get_Survey_By_SURVEY_ID.SURVEY_ID = 1;
Survey oSurvery = oBLC.Get_Survey_By_SURVEY_ID(oParams_Get_Survey_By_SURVEY_ID);
```

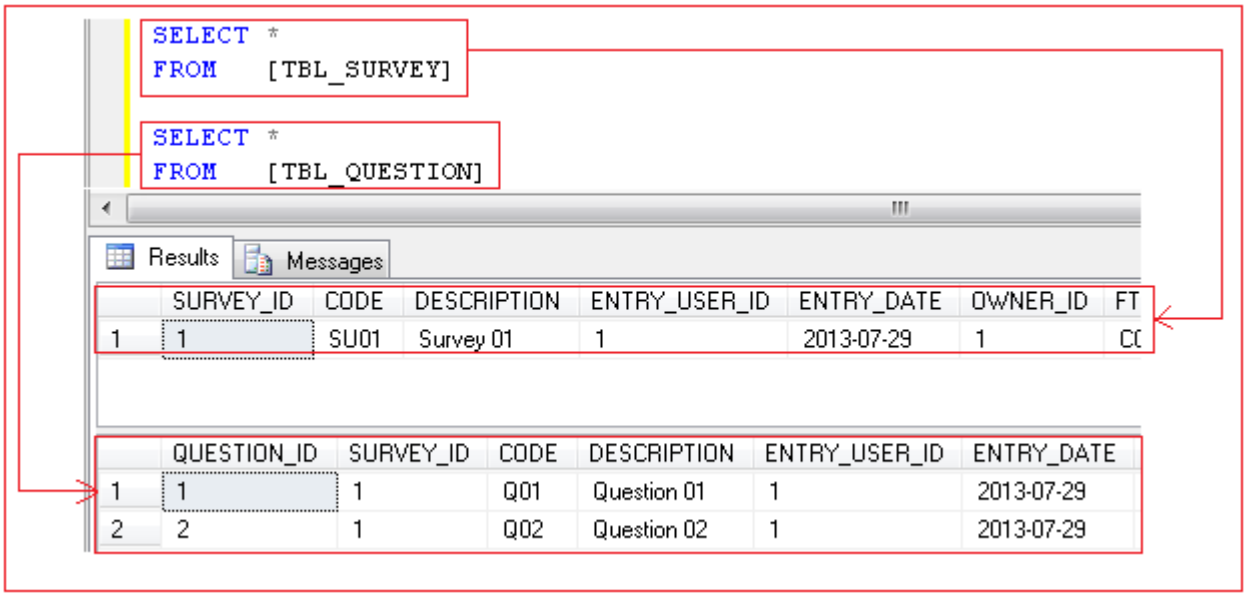
Name	Value
oSurvery	{BLC.Survey}
CODE	"SU01"
DESCRIPTION	"Survery 01"
ENTRY_DATE	"2013-07-26"
ENTRY_USER_ID	1
My_Question	Count = 3
[0]	{BLC.Question}
[1]	{BLC.Question}
[2]	{BLC.Question}
CODE	"Q3"
DESCRIPTION	"Question 03"
ENTRY_DATE	"2013-07-26"
ENTRY_USER_ID	1
My_Survey	{BLC.Survey}
OWNER_ID	1
QUESTION_ID	3
SURVEY_ID	1
Raw View	
OWNER_ID	1
SURVEY_ID	1

As you noticed, this survey related questions have been retrieved as a collection of Question type under this survey and each Question entry is retrieved in Advanced Model (_Adv)

In case you want to retrieve the Question (or any child) in simple mode (not advanced) just set the following attribute to true

```
#region Eager Loading
oCodeBooster.List_Eager_Loading = new List<Eager_Loading>();
oCodeBooster.List_Eager_Loading.Add
    (
        new Eager_Loading()
        {
            Method_Name = "Get_Person_By_PERSON_ID_Adv",
            ParentTable = "[TBL_PERSON]",
            ChildTables = new List<string>() { "[TBL_ADDRESS]" },
            Is_Simple_Child = true
        }
    );
```

***. Let's assume that you have the following data in your database (previously created during training)**

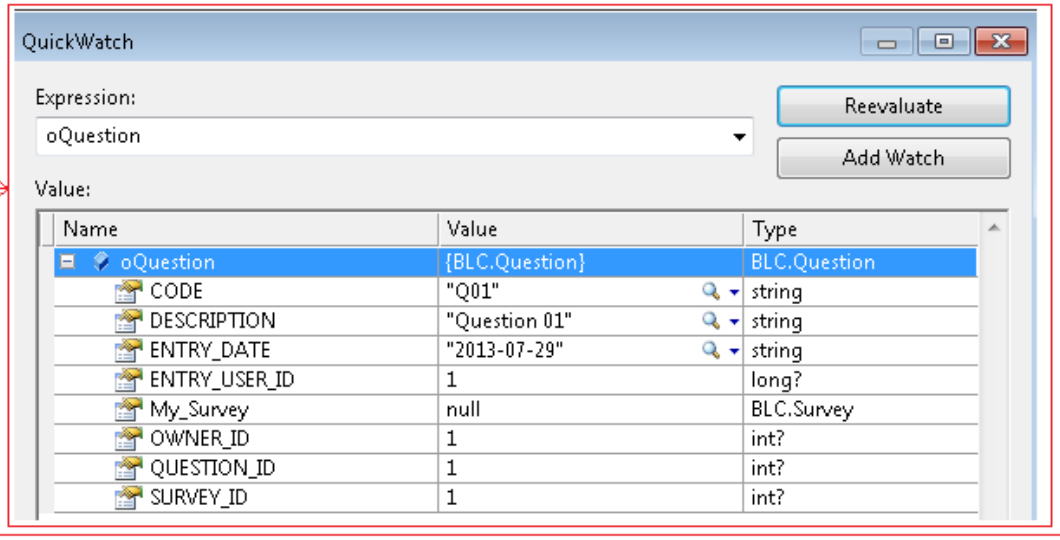


	SURVEY_ID	CODE	DESCRIPTION	ENTRY_USER_ID	ENTRY_DATE	OWNER_ID	FT
1	1	SU01	Survey 01	1	2013-07-29	1	C(

	QUESTION_ID	SURVEY_ID	CODE	DESCRIPTION	ENTRY_USER_ID	ENTRY_DATE
1	1	1	Q01	Question 01	1	2013-07-29
2	2	1	Q02	Question 02	1	2013-07-29

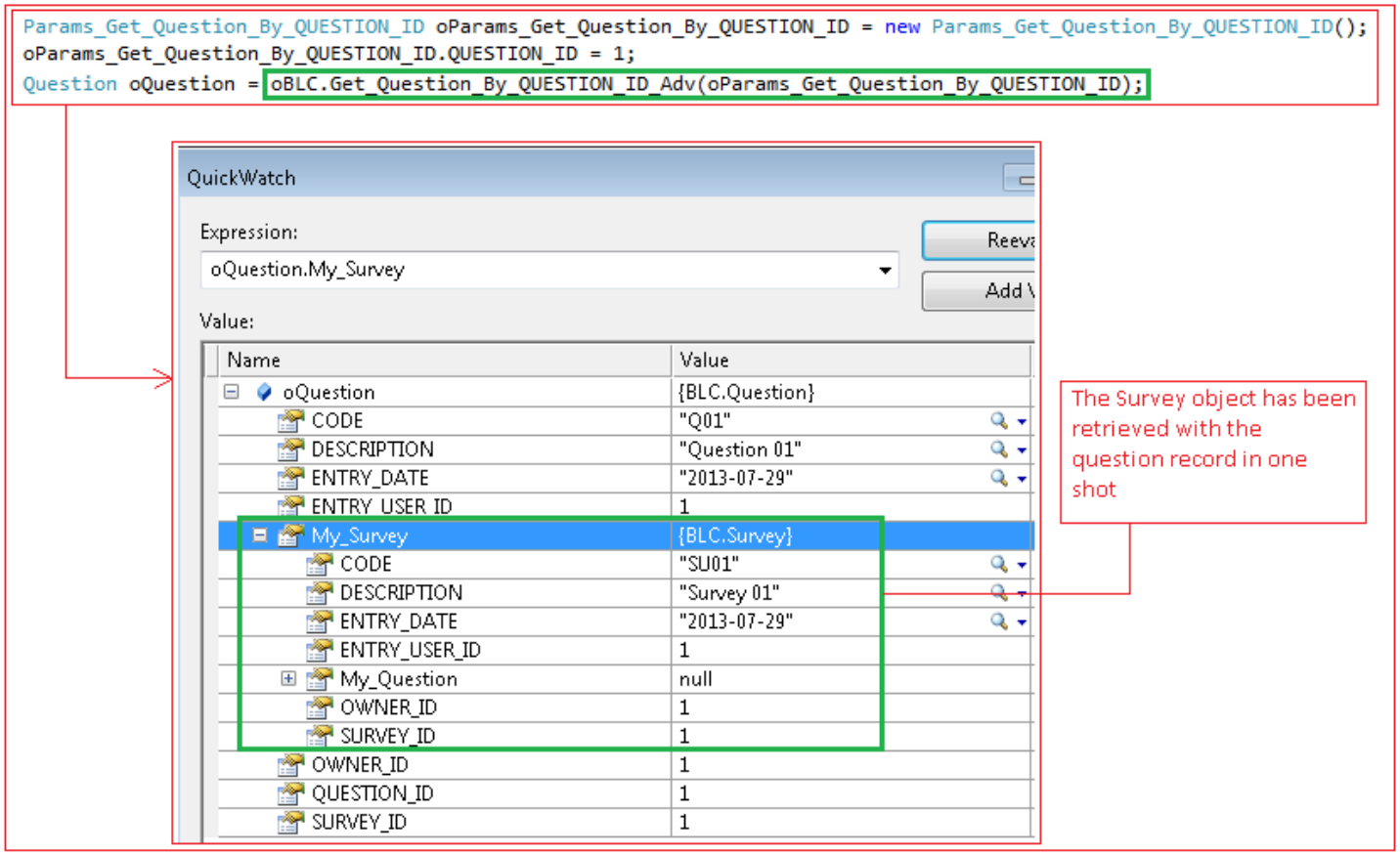
***. If you run the following code, you will return a Question object representing exactly the existing record in database table.**

```
Params_Get_Question_By_QUESTION_ID oParams_Get_Question_By_QUESTION_ID = new Params_Get_Question_By_QUESTION_ID();
oParams_Get_Question_By_QUESTION_ID.QUESTION_ID = 1;
Question oQuestion = oBLC.Get_Question_By_QUESTION_ID(oParams_Get_Question_By_QUESTION_ID);
```

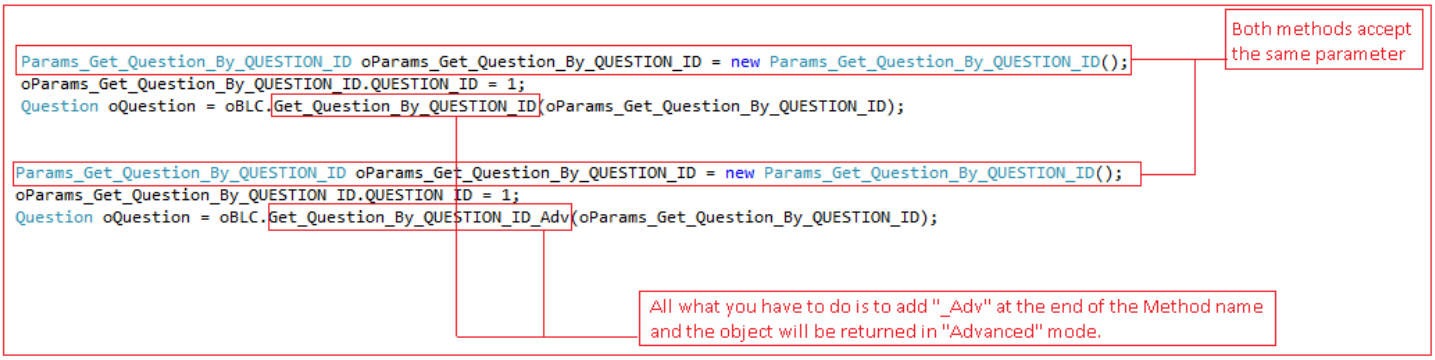


Name	Value	Type
oQuestion	{BLC.Question}	BLC.Question
CODE	"Q01"	string
DESCRIPTION	"Question 01"	string
ENTRY_DATE	"2013-07-29"	string
ENTRY_USER_ID	1	long?
My_Survey	null	BLC.Survey
OWNER_ID	1	int?
QUESTION_ID	1	int?
SURVEY_ID	1	int?

***. In case you want to return on the level of "Question" object the corresponding Survey Objet (Not only its SURVEY_ID) you have to issue the following**



***. As you noticed both methods accept the same parameter "Params_Get_Question_By_QUESTION_ID", but the second method has the same name while adding "_Adv", Get_Question_By_QUESTION_ID_Adv**



***. Take for granted, any "Get" method has an advanced version accepting the same parameter and returns the advanced version of the corresponding returned type.**

Ex: Get_Question_By_QUESTION_ID → Get_Question_By_QUESTION_ID_Adv [as per previous example]

Get_Question_By_OWNER_ID → Get_Question_By_OWNER_ID_Adv [as the following]

Accepting same parameter but the second method will return the advanced version of Question type

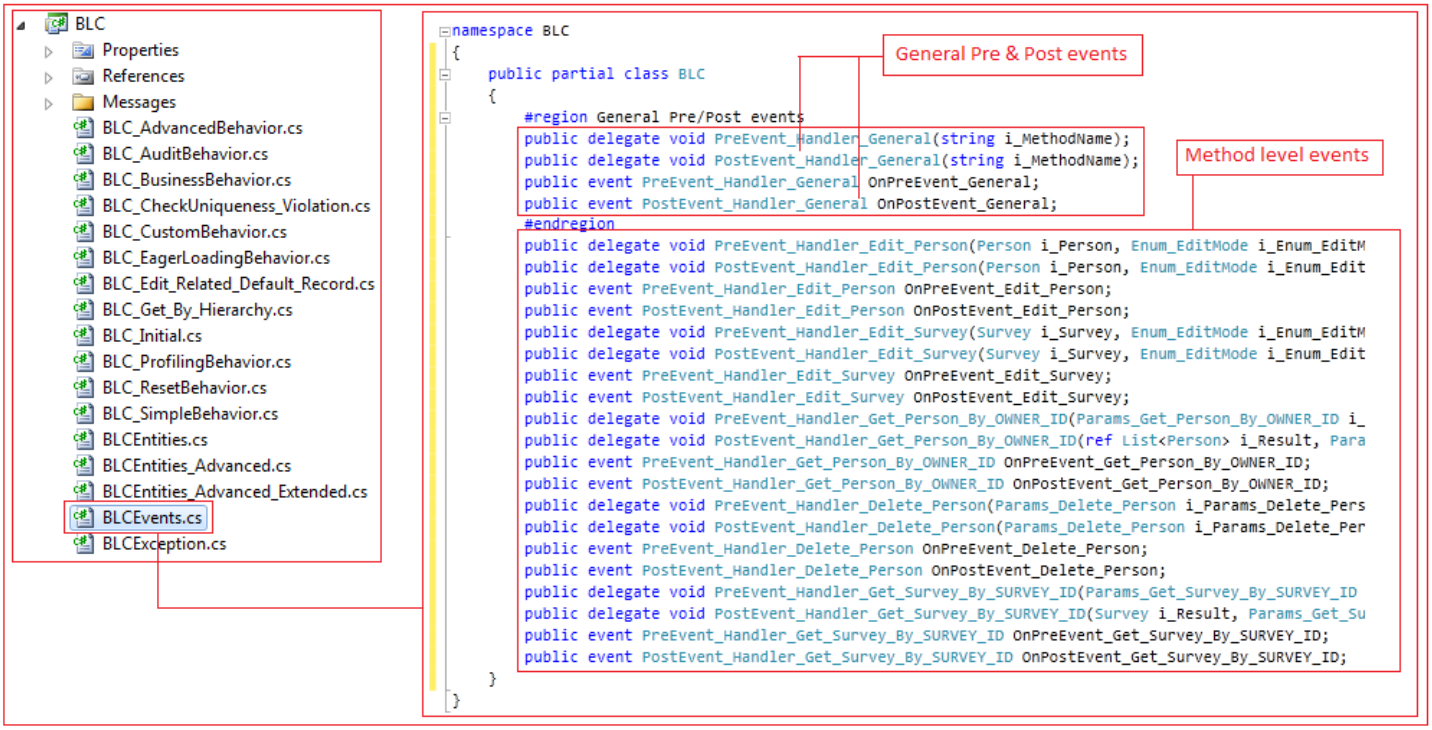
```
Params_Get_Question_By_OWNER_ID oParams_Get_Question_By_OWNER_ID = new Params_Get_Question_By_OWNER_ID();
oParams_Get_Question_By_OWNER_ID.OWNER_ID = 1;
List<Question> oList_Question = oBLC.Get_Question_By_OWNER_ID(oParams_Get_Question_By_OWNER_ID);


Params_Get_Question_By_OWNER_ID oParams_Get_Question_By_OWNER_ID = new Params_Get_Question_By_OWNER_ID();
oParams_Get_Question_By_OWNER_ID.OWNER_ID = 1;
List<Question> oList_Question = oBLC.Get_Question_By_OWNER_ID_Adv(oParams_Get_Question_By_OWNER_ID);
```

***. Now, let's check out the Pre & Post General events**

Previously, we demonstrated how we can add Pre/ Post events to any method we want in order to inject rules before and after the actual execution.

In case you need a certain code to be fired before (or after) any method call in your application (logging for example), you can use the OnPreEvent_General & OnPostEvent_General events.

Both events are always generated by the code booster (so there is no need to inform the engine to generate them like method level event) in BLCEvents.cs file (under the BLC project).



***. In BLC_Initial.cs file and specifically in SubscribeToEvents method, subscribe a method to OnPreEvent_General event and implement it in "Events implementation" region, like the following.**

```
public void SubscribeToEvents()
{
    Declaration And Initialization Section.
    #region Body Section.

    this.OnPreEvent_General += new PreEvent_Handler_General(BLC_OnPreEvent_General);


    Initialize_Eager_Loading_Mechanism();
    Initialize_Reset_Mechanism();

    this.OnPreEvent_Edit_Person +=new PreEvent_Handler_Edit_Person(BLC_OnPreEvent_Edit_Person);
    this.OnPreEvent_Edit_Person += new PreEvent_Handler_Edit_Person(BLC_OnPreEvent_Edit_Person_02);
    this.OnPostEvent_Edit_Person += new PostEvent_Handler_Edit_Person(BLC_OnPostEvent_Edit_Person);
    this.OnPostEvent_Delete_Person += new PostEvent_Handler_Delete_Person(BLC_OnPostEvent_Delete_Person);
    this.OnPreEvent_Delete_Person += new PreEvent_Handler_Delete_Person(BLC_OnPreEvent_Delete_Person);
    this.OnPostEvent_Get_Person_By_OWNER_ID += new PostEvent_Handler_Get_Person_By_OWNER_ID(BLC_OnPostEvent_Get_Person_By_OWNER_ID);
    #endregion
}
```

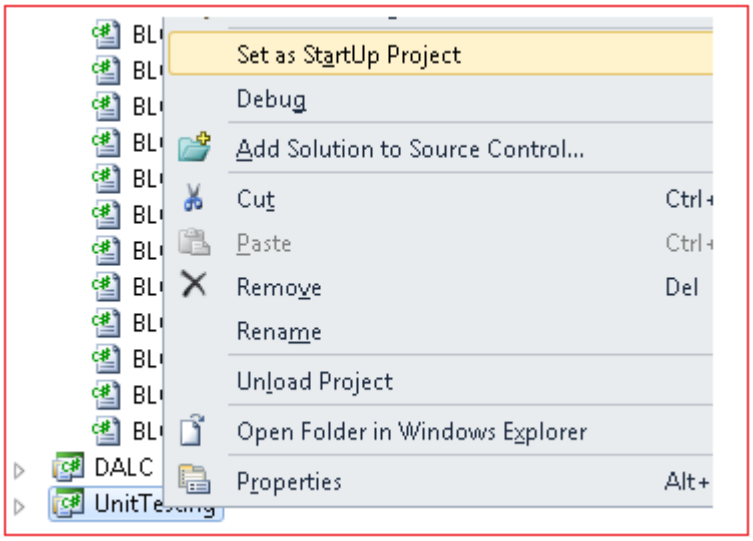
```
#region Events Implementation
void BLC_OnPreEvent_General(string i_MethodName)
{
    Console.WriteLine(i_MethodName);
}
```

Implementing the subscribed method

```
void BLC_OnPreEvent_Edit_Person(Person i_Person, BLC.Enum_EditMode i_Enum_EditMode)...
void BLC_OnPreEvent_Edit_Person_02(Person i_Person, BLC.Enum_EditMode i_Enum_EditMode)...
void BLC_OnPostEvent_Edit_Person(Person i_Person, BLC.Enum_EditMode i_Enum_EditMode)...
void BLC_OnPreEvent_Delete_Person(Params_Delete_Person i_Params_Delete_Person)...
void BLC_OnPostEvent_Delete_Person(Params_Delete_Person i_Params_Delete_Person)...
#endregion
```

***. Rebuild the BLC & UnitTesting console applications.**

***. Set the UnitTesting console application as the startup project.**



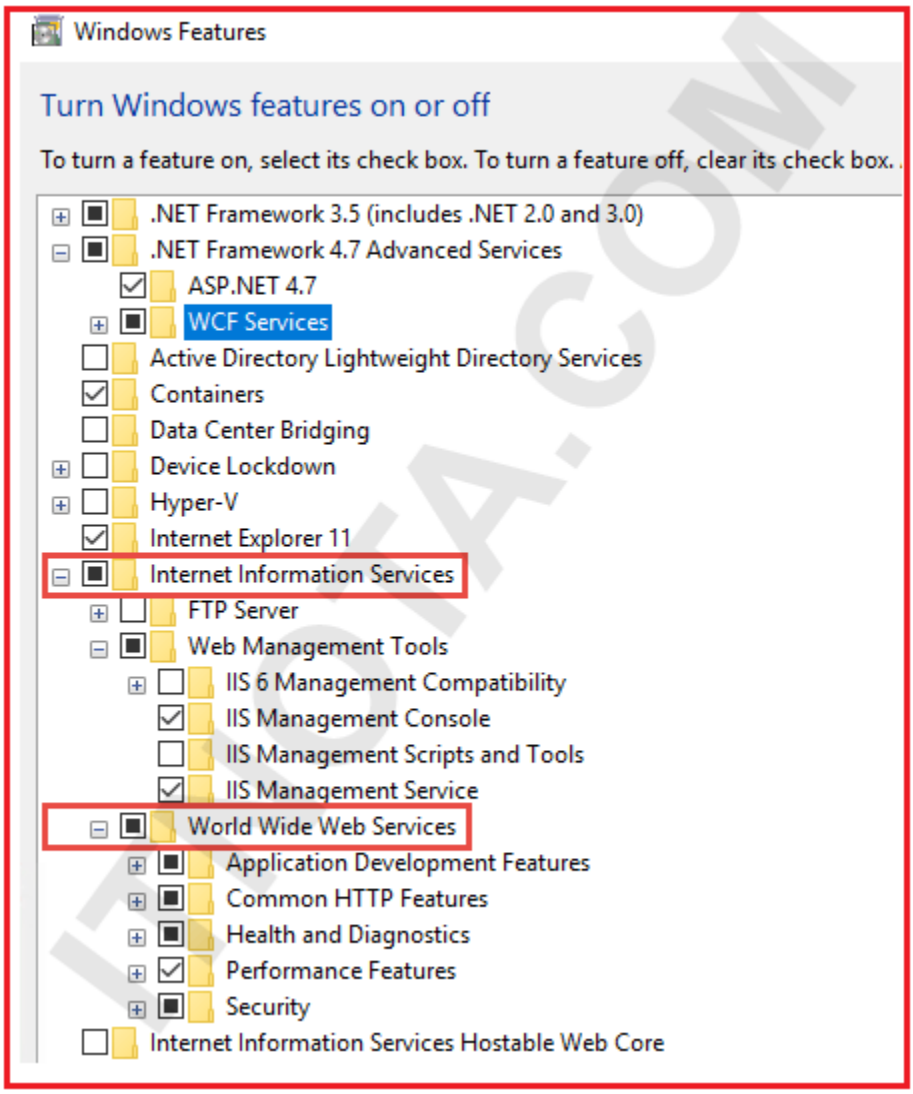
***. In the UnitTesting Console application, call two different methods and you will notice that the method name will be printed twice because the OnPreEvent_General _General event has been fired on each method.**

```
namespace UnitTesting
{
    class Program
    {
        #region Main
        [STAThread]
        static void Main(string[] args)
        {
            Declaration And Initialization Section.
            #region Body

            Params_Get_Survey_By_SURVEY_ID oParams_Get_Survey_By_SURVEY_ID = new Params_Get_Survey_By_SURVEY_ID();
            oParams_Get_Survey_By_SURVEY_ID.SURVEY_ID = 4;
            Survey oSurvey = oBLC.Get_Survey_By_SURVEY_ID(oParams_Get_Survey_By_SURVEY_ID);

            Params_Get_Question_By_SURVEY_ID oParams_Get_Question_By_SURVEY_ID = new Params_Get_Question_By_SURVEY_ID();
            oParams_Get_Question_By_SURVEY_ID.SURVEY_ID = 4;
            List<Question> oList_Question = oBLC.Get_Question_By_SURVEY_ID(oParams_Get_Question_By_SURVEY_ID);


            #endregion
        }
        #endregion
    }

}
```

*. The next step will be creating a web API, but first you have to assure that you installed IIS (Internet information services) on your PC (Laptop) by doing so:

You can find step by step how you install IIS on Windows 10 by visiting the following link

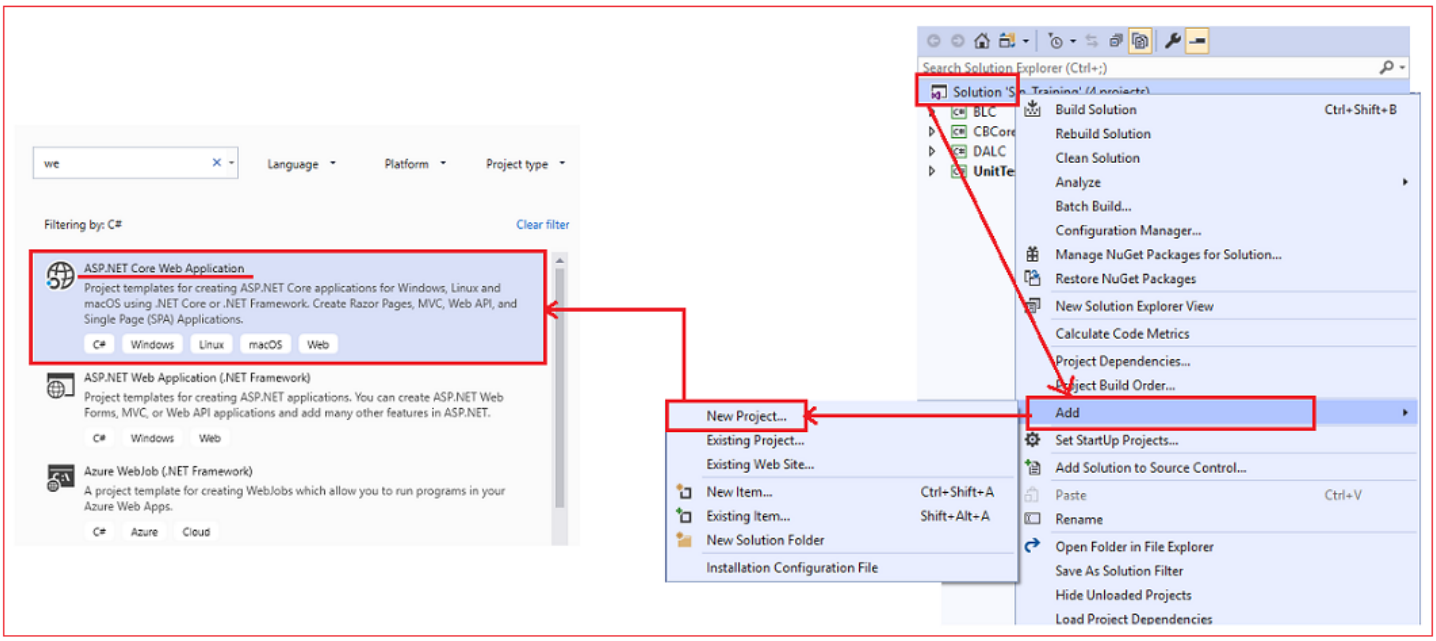
https://www.itnota.com/install-iis-windows/

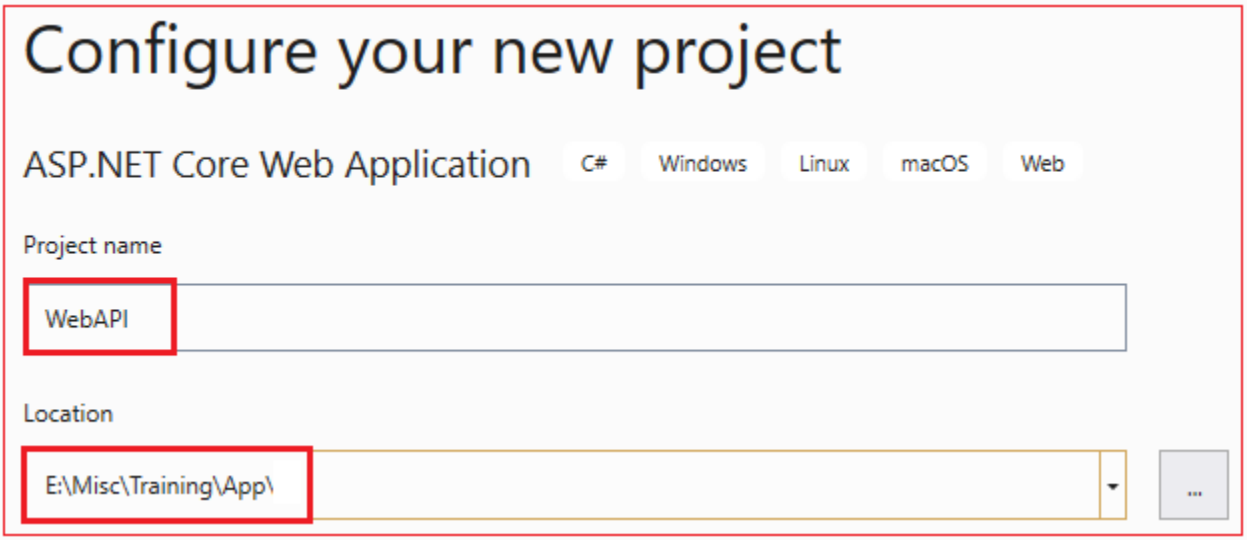


*. In order to create a Web API Project, you must do the following:

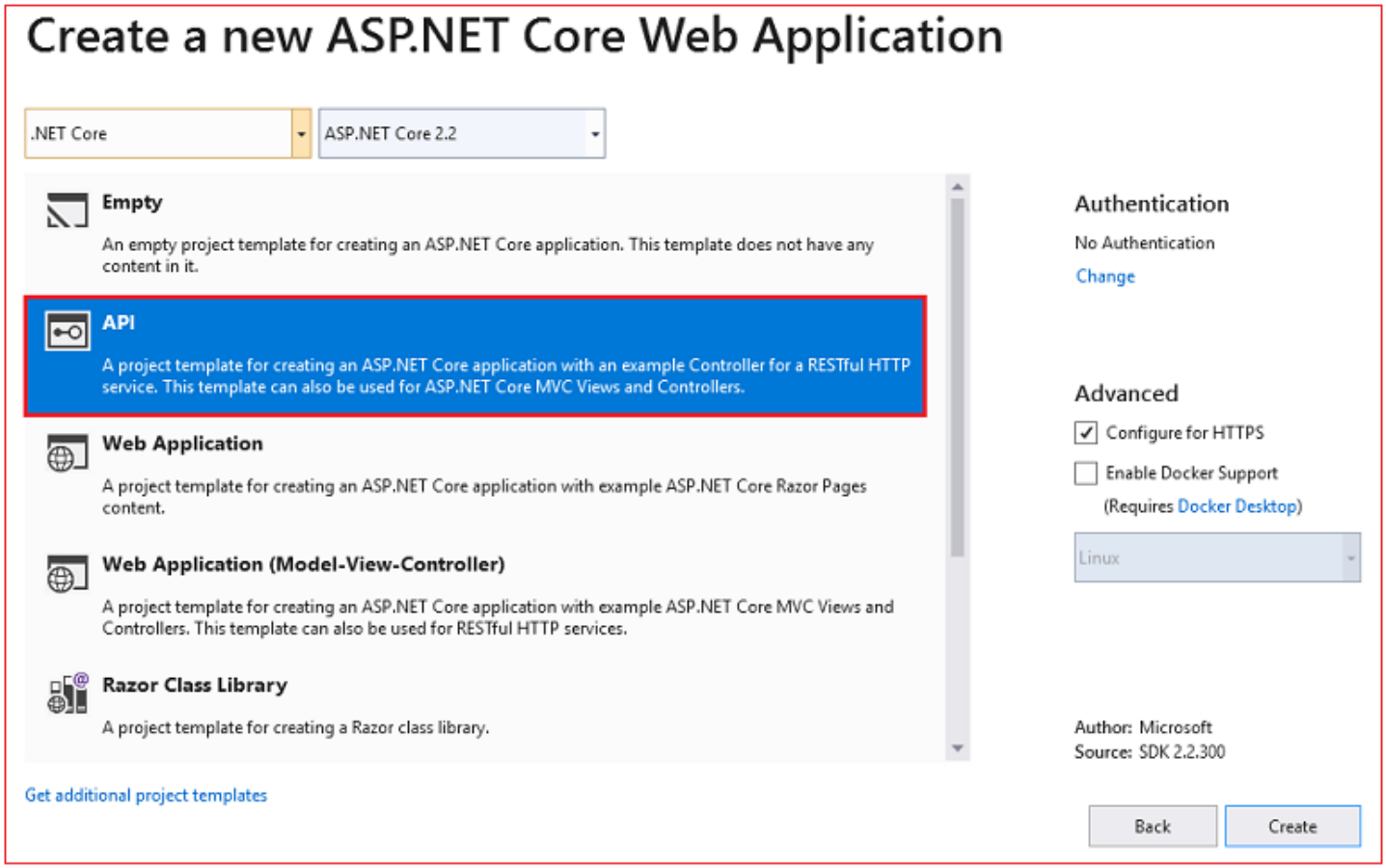
*. Right click on the previously created solution & Add a new **ASP.Net Core Web Application** (C# Language) as the following

N.B: Name it as Web API and save it in your working folder.

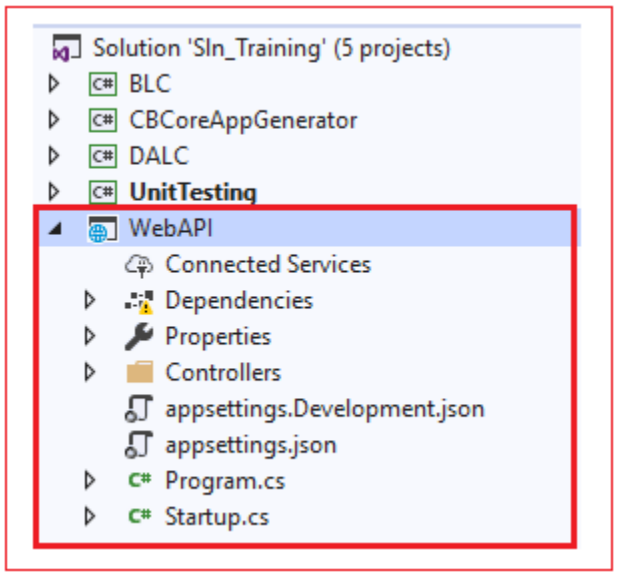




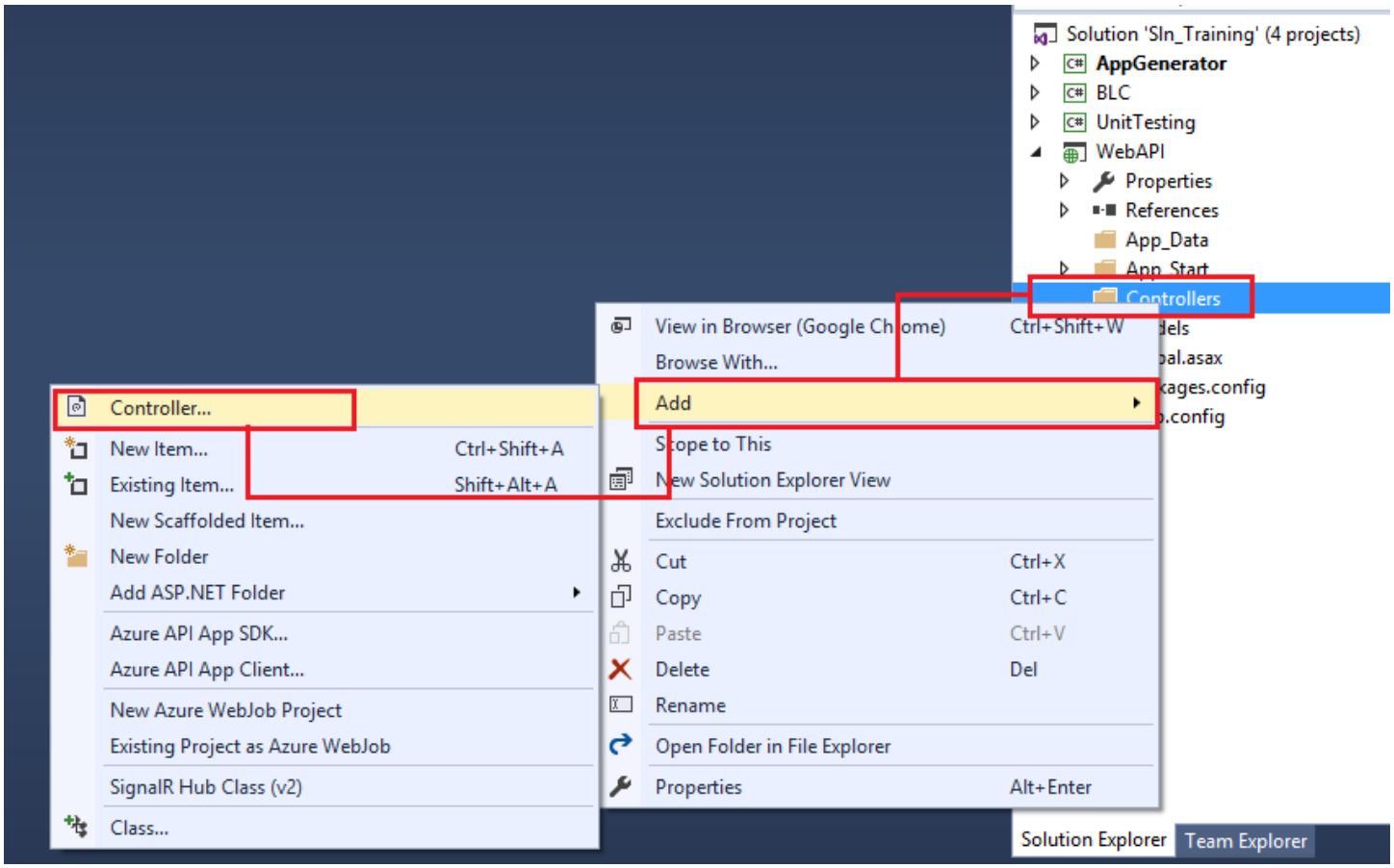
*. From the Provided templates, select "API" as the following:



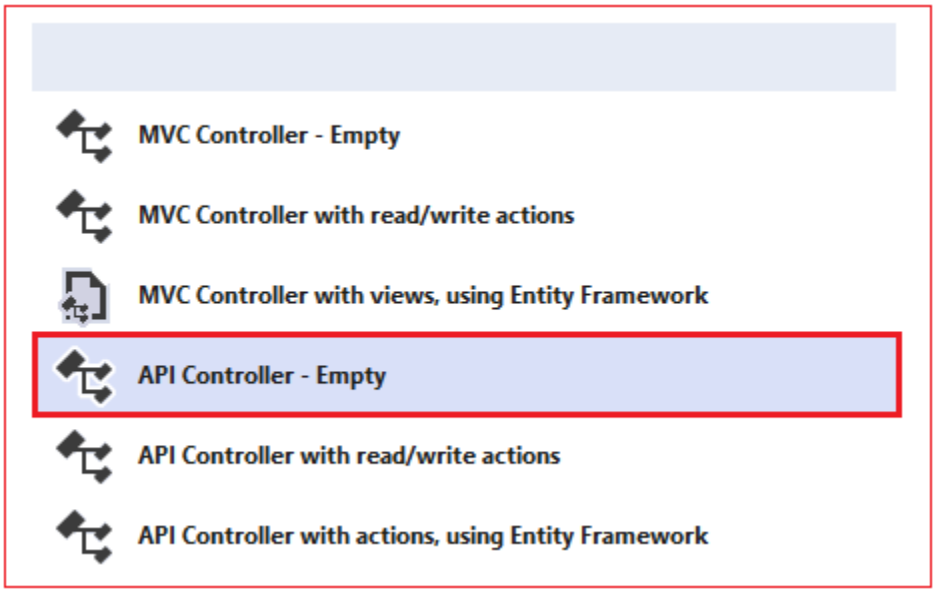
*. The following project will be created.



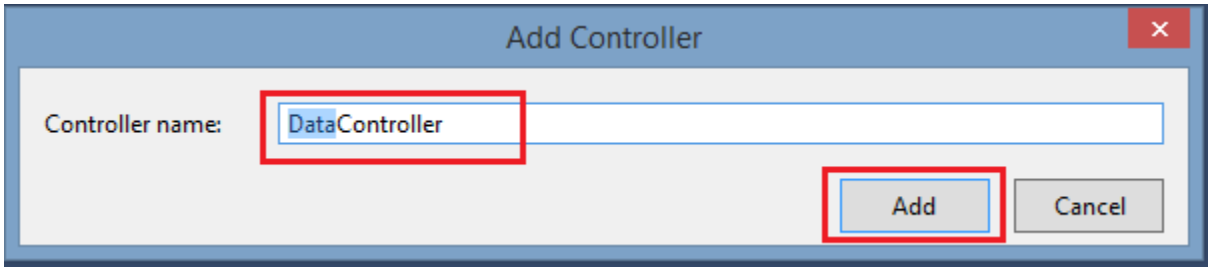
*. Right Click on the "Controllers" folder & add a new controller item as the following.



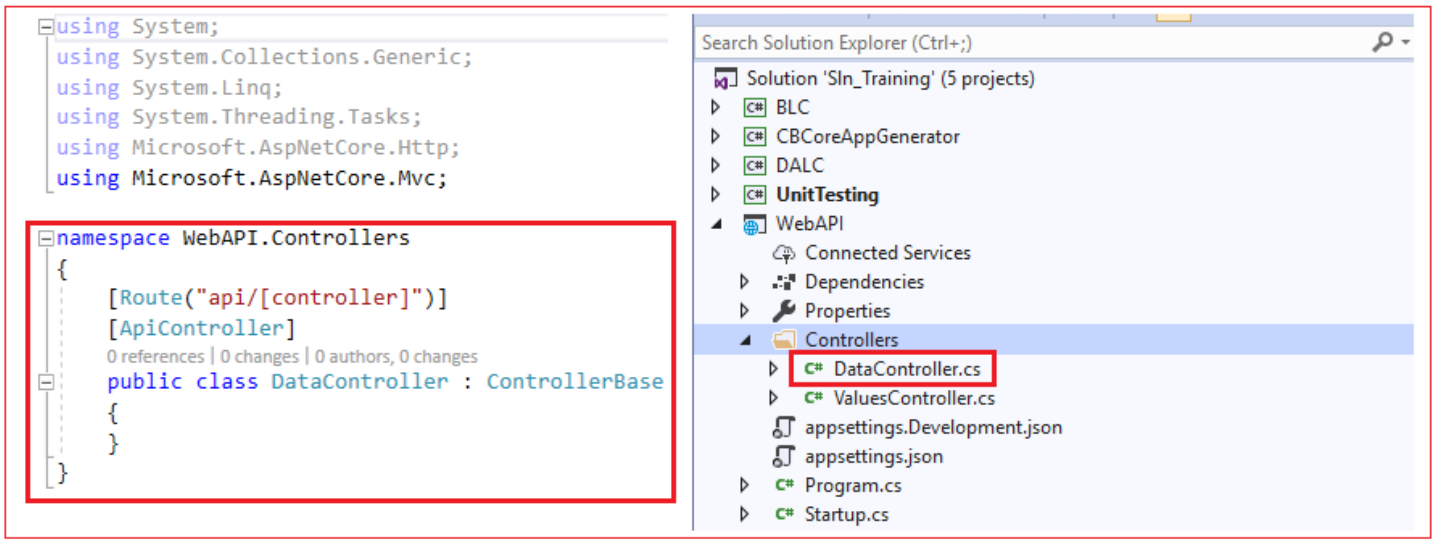
*. Select "API Controller - Empty” option as the following, then click on Add button



*. Name it as "DataController", and press "Add" as the following



*. A new file named "DataControllers.cs" file will be created under "Controllers" folder as the following.

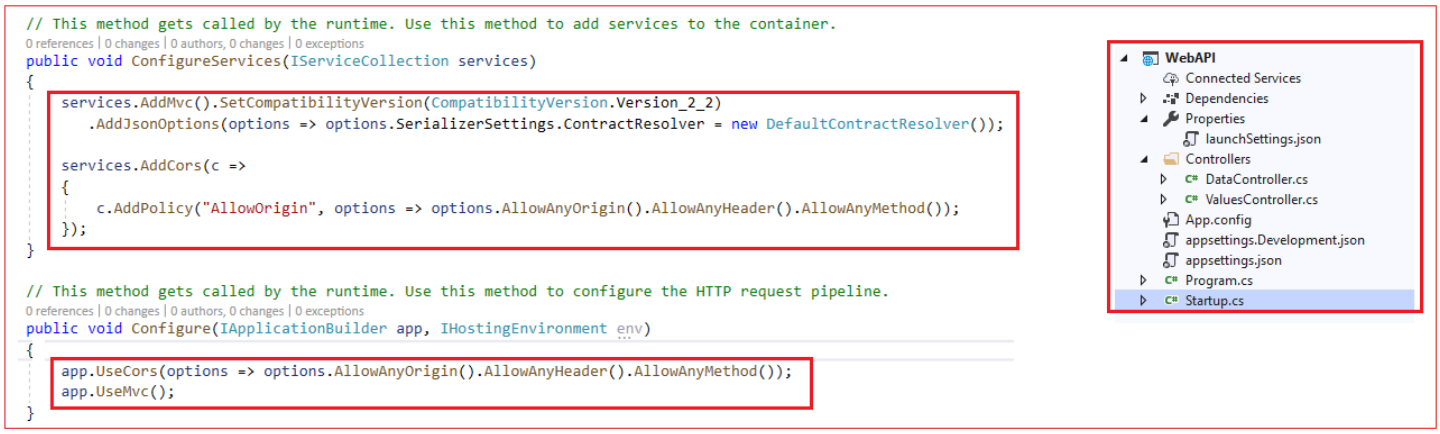


```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;

namespace WebAPI.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    0 references | 0 changes | 0 authors, 0 changes
    public class DataController : ControllerBase
    {
    }
}
```

```
Search Solution Explorer (Ctrl+;)
Solution 'Sln_Training' (5 projects)
BLC
CBCoreAppGenerator
DALC
UnitTesting
WebAPI
  Connected Services
  Dependencies
  Properties
  Controllers
    DataController.cs
    ValuesController.cs
  appsettings.Development.json
  appsettings.json
  Program.cs
  Startup.cs
```

*. In case you are working with .Net 2.2 (or lower version) you have to the following

In the Startup.cs file and especially in ConfigureServices method assure you added the mentioned line of code.



```
// This method gets called by the runtime. Use this method to add services to the container.
0 references | 0 changes | 0 authors, 0 changes | 0 exceptions
public void ConfigureServices(IServiceCollection services)
{
    services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_2)
        .AddJsonOptions(options => options.SerializerSettings.ContractResolver = new DefaultContractResolver());

    services.AddCors(c =>
    {
        c.AddPolicy("AllowOrigin", options => options.AllowAnyOrigin().AllowAnyHeader().AllowAnyMethod());
    });
}

// This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
0 references | 0 changes | 0 authors, 0 changes | 0 exceptions
public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    app.UseCors(options => options.AllowAnyOrigin().AllowAnyHeader().AllowAnyMethod());
    app.UseMvc();
}
```

```
WebAPI
  Connected Services
  Dependencies
  Properties
    launchSettings.json
  Controllers
    DataController.cs
    ValuesController.cs
  App.config
  appsettings.Development.json
  appsettings.json
  Program.cs
  Startup.cs
```

*. In case you working with .Net Core 3.0 and above you have to do the following:

*. First you have to add the following nugget package.

Microsoft.AspNetCore.Mvc.NewtonsoftJson (Version 3.0.0 or Above)

*. In Startup.cs file you assure the following is written (**You can find Startup.cs file under the CB\Core folder**)

```
readonly string MyAllowSpecificOrigins = "_myAllowSpecificOrigins";
public IConfiguration Configuration { get; }

public void ConfigureServices(IServiceCollection services)
{

    services.AddControllers().AddNewtonsoftJson(options =>
        {
            options.SerializerSettings.ContractResolver = new DefaultContractResolver();
        });

    services.AddCors(options =>
    {
        options.AddPolicy(MyAllowSpecificOrigins,
        builder =>
        {
            builder.AllowAnyOrigin().AllowAnyHeader().AllowAnyMethod();
        });
    });

}
```

```
public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
        if (env.IsDevelopment())
        {
            app.UseDeveloperExceptionPage();
        }

        //app.UseHttpsRedirection();

        app.UseRouting();

        app.UseCors(MyAllowSpecificOrigins);

        app.UseAuthorization();

        app.UseEndpoints(endpoints =>
        {
            endpoints.MapControllers();
        });
}
```

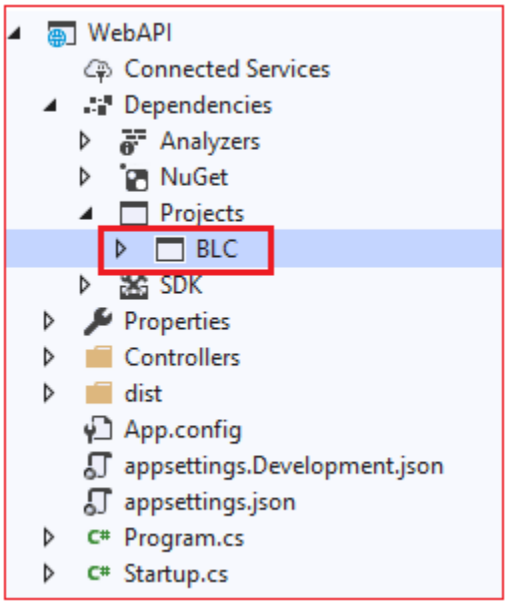
*. Add file App.config to the WebAPI Project. (**Copy it from UnitTesting Project if you want**)

IT"S MANDATORY TO UNDERSTAND THAT IN .NET CORE : THE ASP.NET CORE WEB APPLICATIONS ARE JUST CONSOLE APPLICATIONS (*if you run them on Kestrel Development Web Server*)

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <appSettings>
    <add key="CONN_STR" value="Data Source=RONI-MOBILE\INSTANCE_2K17_01;Database=Training;User ID=sa;Password=sapassword"/>
    <add key="BLC_MESSAGES" value="E:\Misc\CB101\App002\App\BLC\Messages\Messages.xml" />
  </appSettings>
</configuration>
```

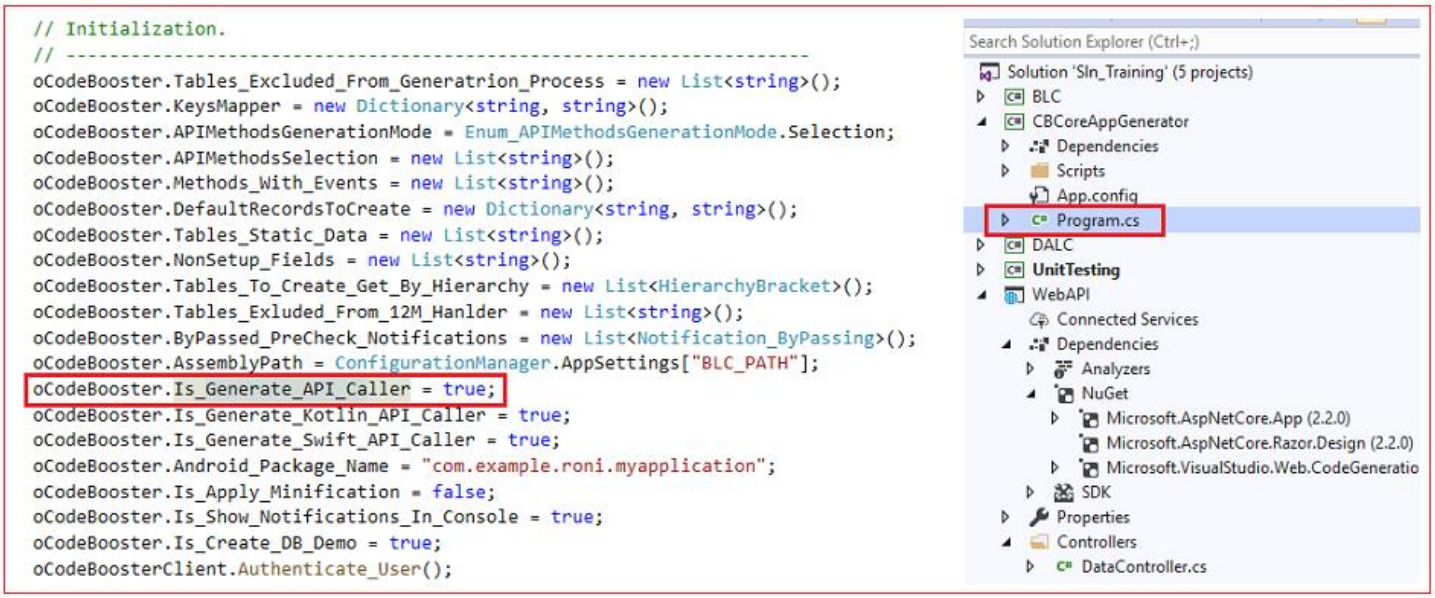
Sure you have to change the database connection string and the path of Messages.xml file to match your environment.

*. On WebAPI Project, assure that you added a reference to the BLC Project (as the following)



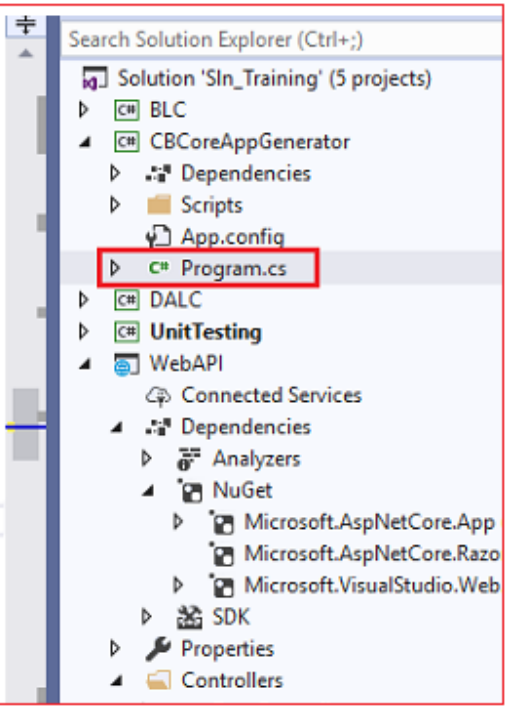
*. Now, in the AppGenerator console application, and especially in program.cs file assure that you have the following

*. In Program.cs also, assure that you have the following flag (Is_Generate_API_Caller) is set to true

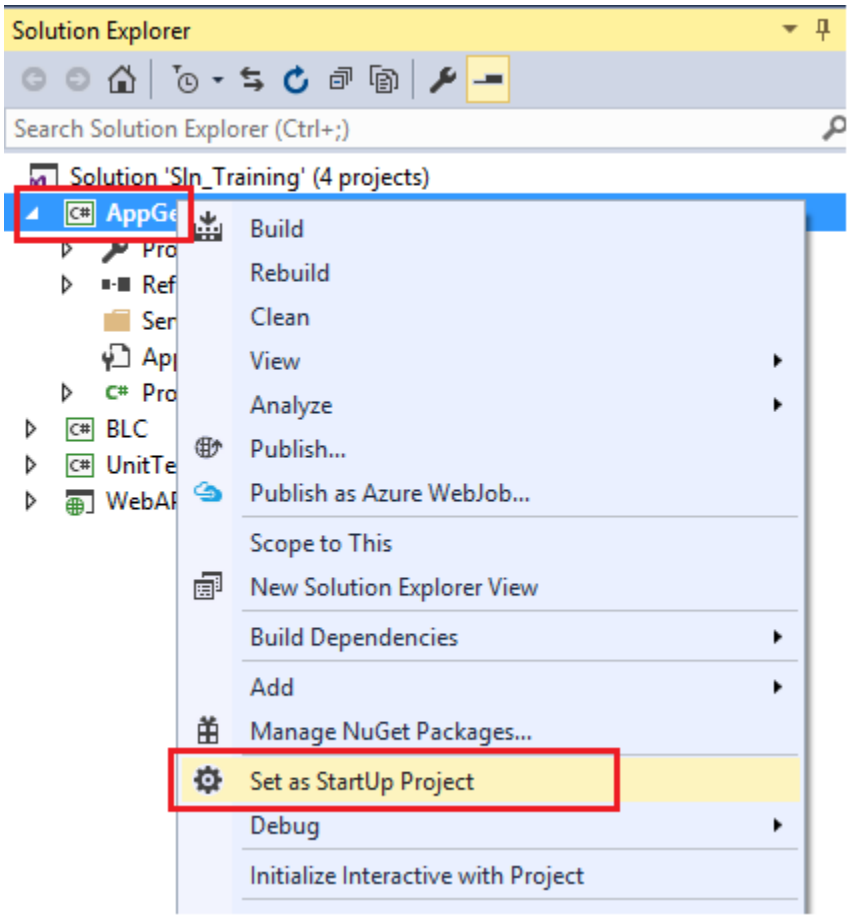


*. Again, in AppGenerator console application, add the following lines under the region named ("#region API")

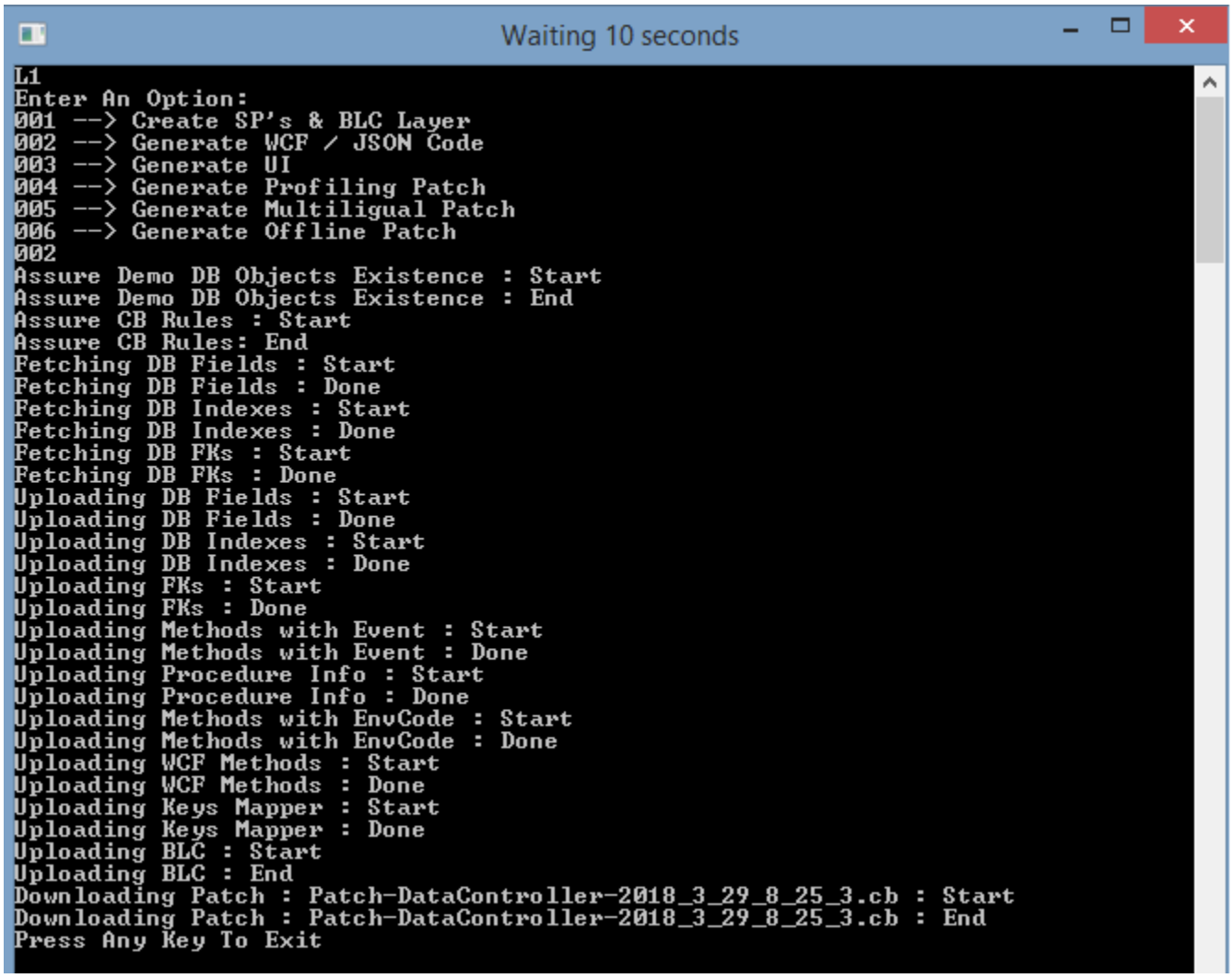
```
#endregion
#region Body Section
Console.WriteLine("Enter An Option:");
Console.WriteLine("001 --> Create SP's & BLC Layer");
Console.WriteLine("002 --> Generate API / JSON Code");
Console.WriteLine("003 --> Generate UI");
Console.WriteLine("051 --> Generate Mobile Native UI");

str_Option = Console.ReadLine();

ByPassing Notification
Caching
Cascade
#region API

// Person
//----------------------------
//oCodeBooster.APIMethodsSelection.Add("Get_Person_By_OWNER_ID");
//oCodeBooster.APIMethodsSelection.Add("Delete_Person");
//----------------------------
```



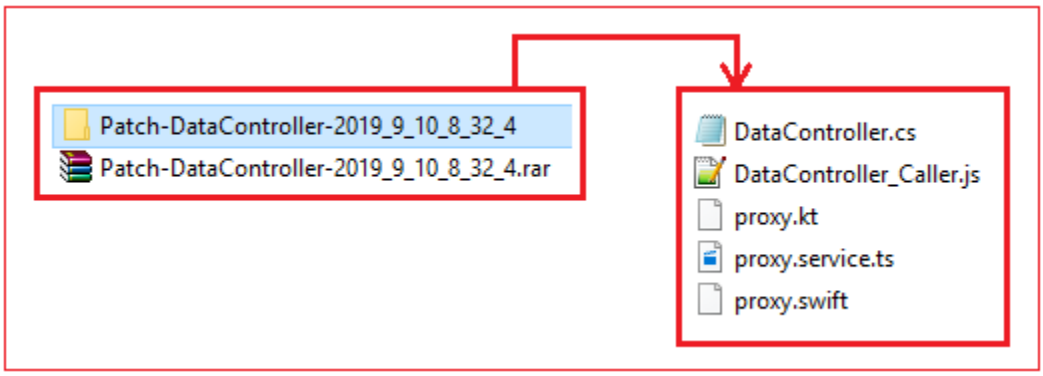
*. Set the AppGenerator console application as startup project.



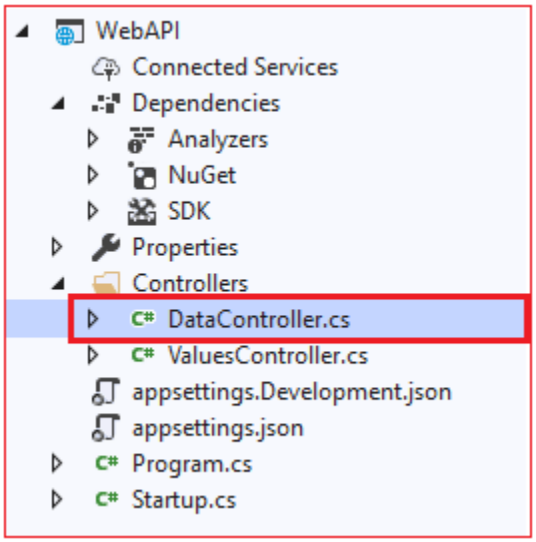
*. Run the AppGenerator Console Application and enter "002" as option and hit enter as the following:



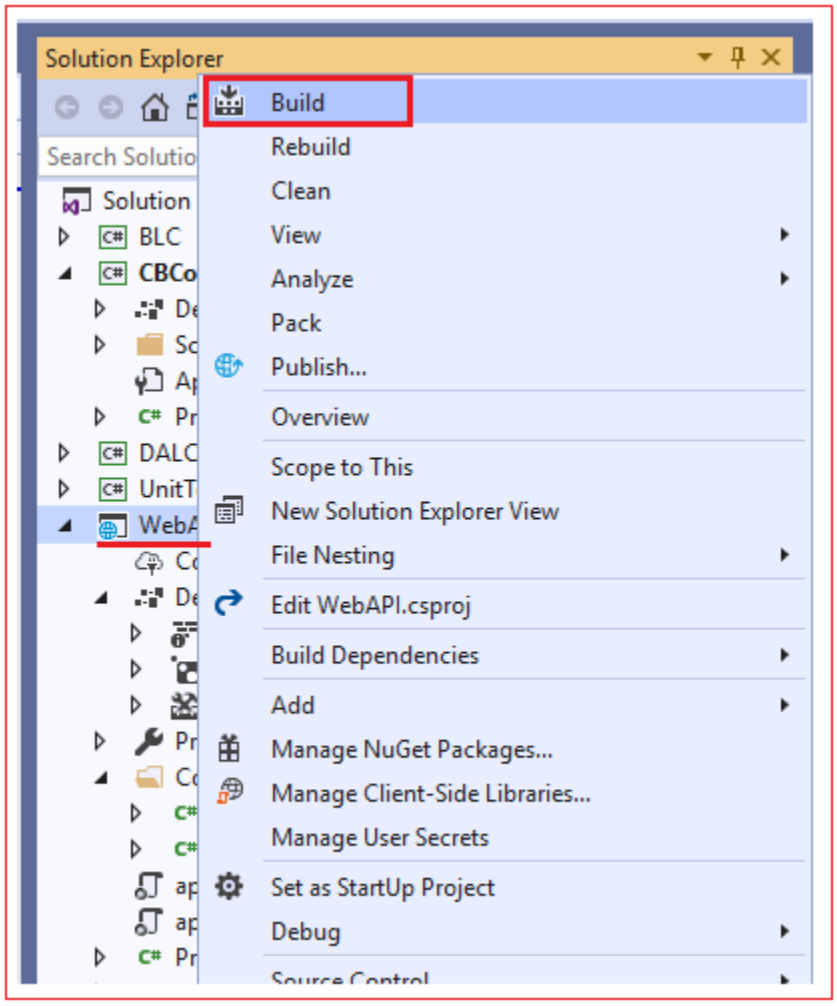
*. A new patch will be downloaded on your machine (under the path already specified in Local_Patch_Folder)



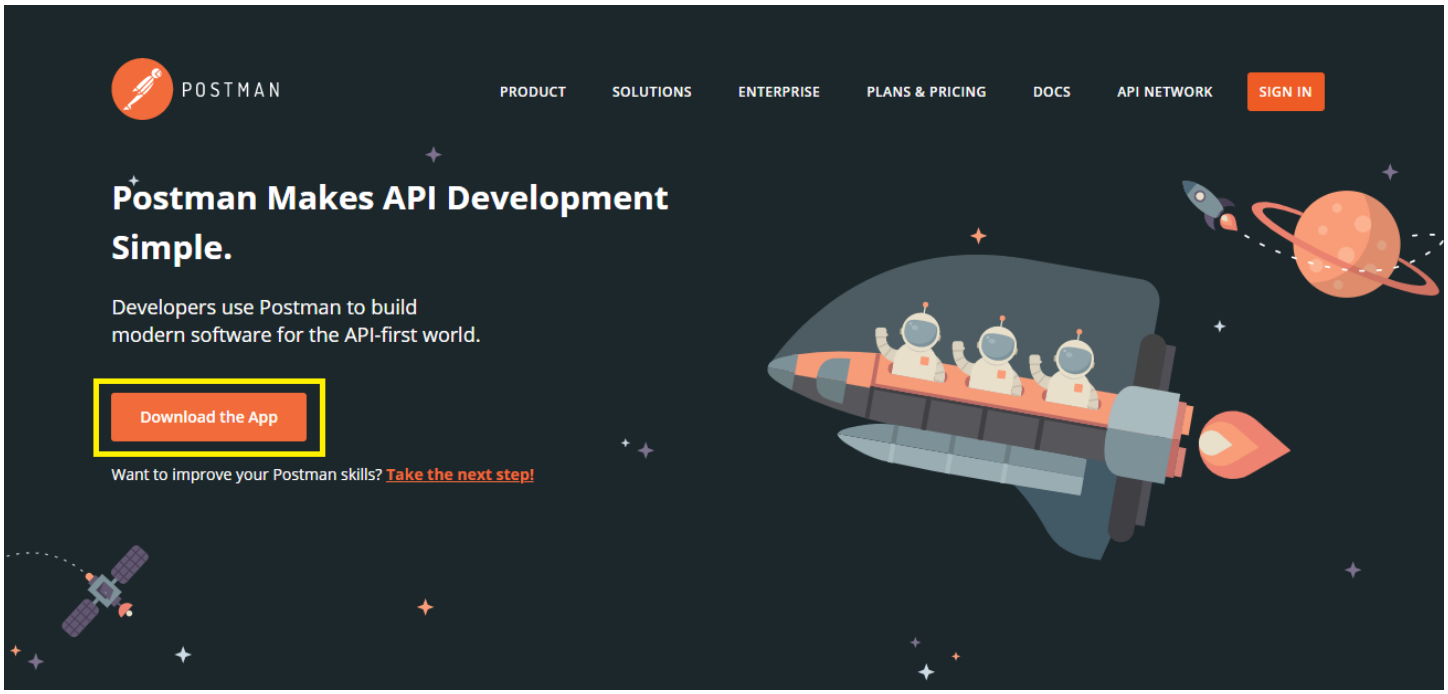
*. Copy the DataController.cs file and pasted it in the WebAPI project and specifically under “Controllers" folder as the following



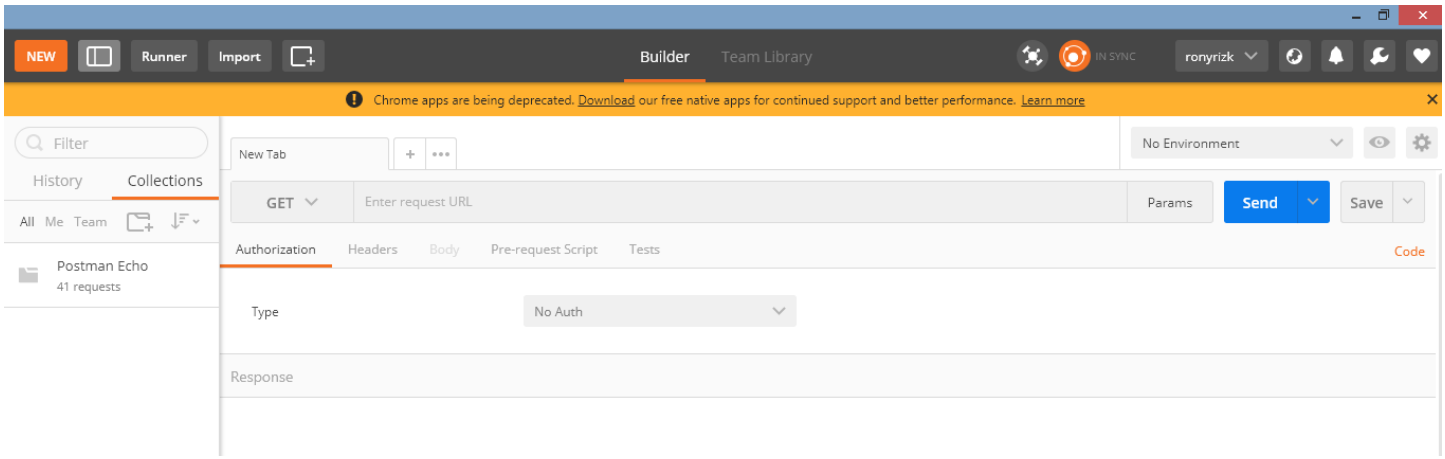
*. Right Click on the WebAPI project and click on build to assure that everything is working correctly.



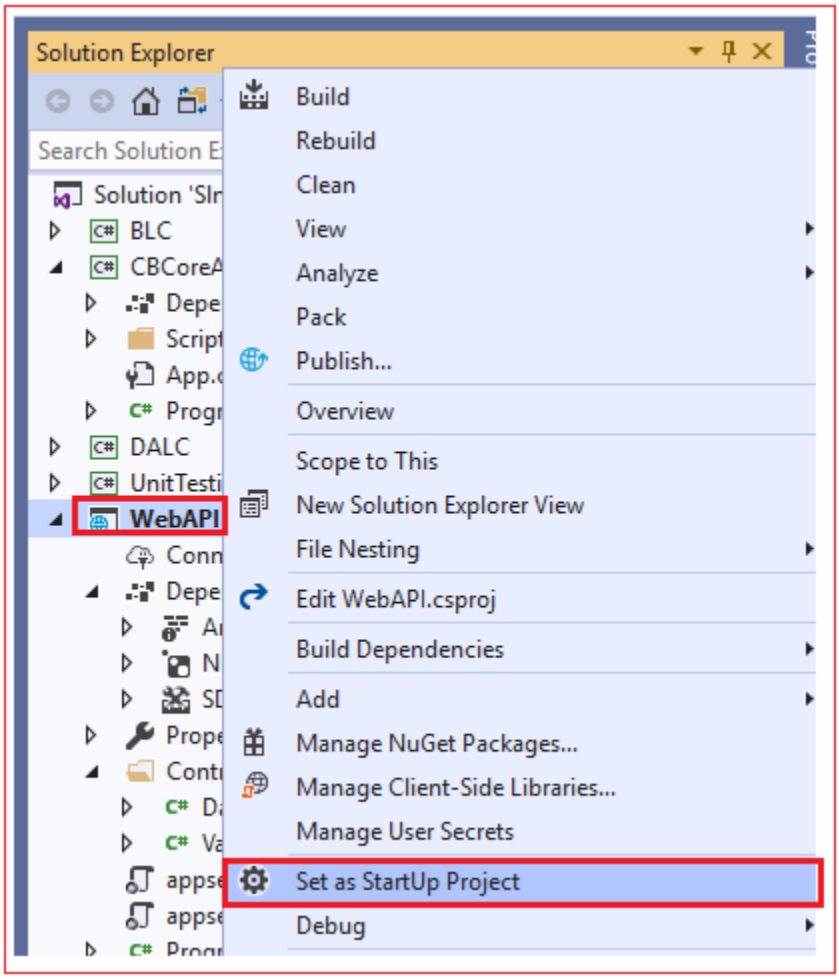
*. In Order to check of the Web API is working correctly, install the Postman application via the following link **https://www.getpostman.com**



*. Once Postman is installed, run it and you will get the following screen

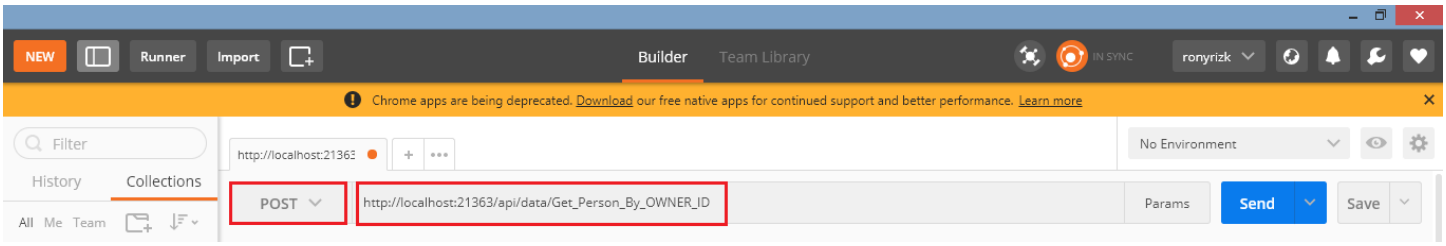


*. In order to be able to test the Web API, set the WebAPI as startup project (in your solution) and press on the run button as the following.



In my case the port number is 21263 (in case you started the WebAPI on IIS else it would be 5000), and this is for sure not the same number you are seeing in your browser by now.

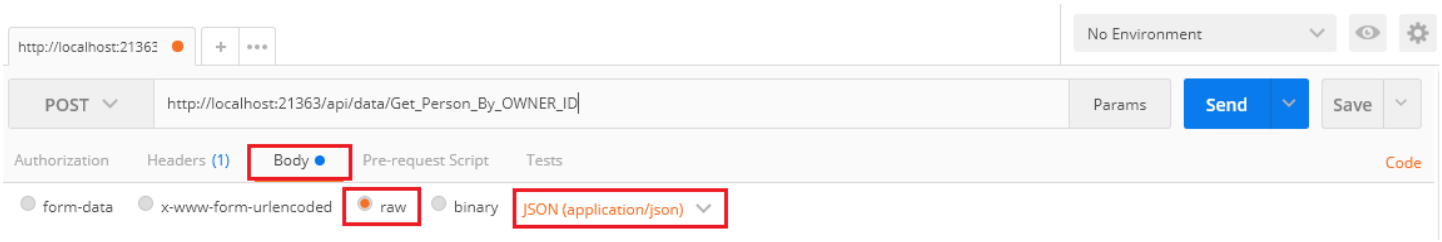
*. Now, in order to test if you can call Get_Person_By_OWNER_ID method using Postman do the following.



Assure you selected "POST" in the drop Down list , and then write in the text box the following url

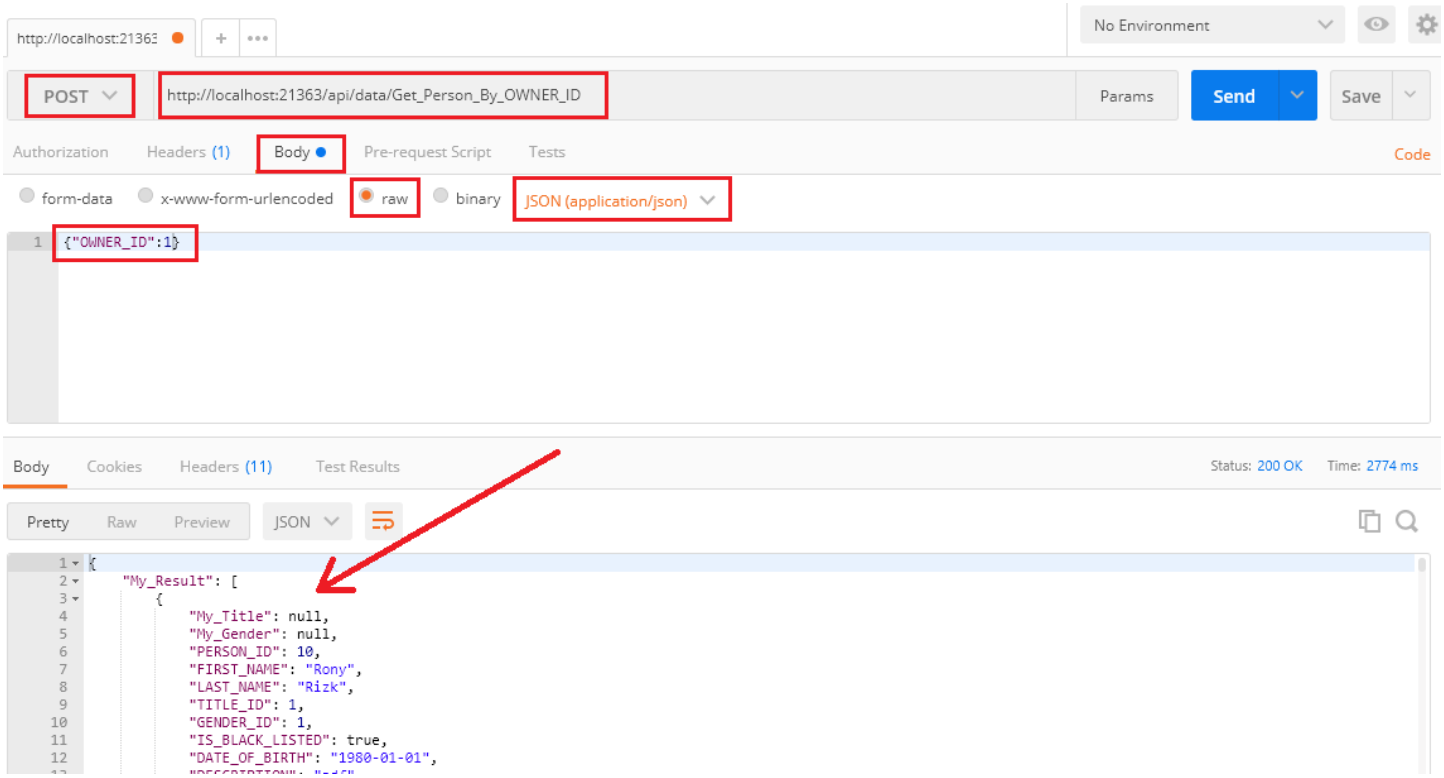
http://localhost:21363/api/data/Get_Person_By_OWNER_ID, sure the port number (marked in yellow) should be replaced by the one provided by your Web API project.

Then , assure you selected the following (Body, raw & JSON (applcation/json)



and finally add in the bottom textbox the following {"OWNER_ID":1} as the below image

And finally press on "Send" button, and you should view the data retrieved by the Web API in the bottom as the following

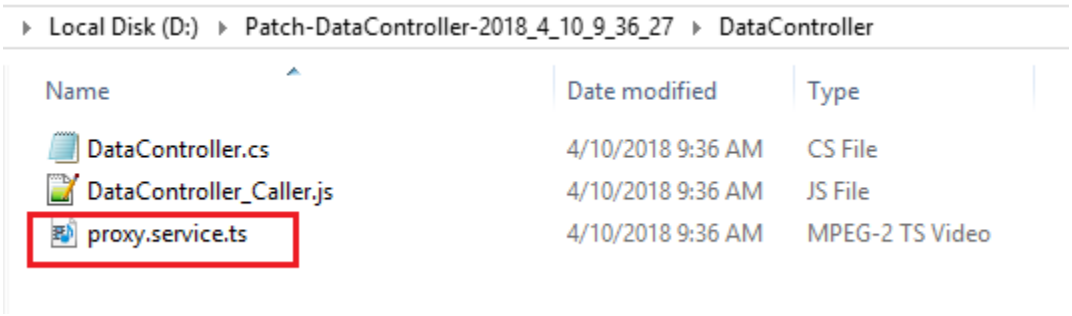


*. For Sure, Postman is used for testing purposes to assure the Web API is working correctly, but at the end we are going to call the Web API from a Web or Mobile Applications, in the following points we will show you how to use it (the Web API) from an Angular Web Applications (It applies also on Ionic based mobile applications)

*. In order to call a Web API from Angular Project, assure you imported the HttpClientModule in the app.module.ts file as the following

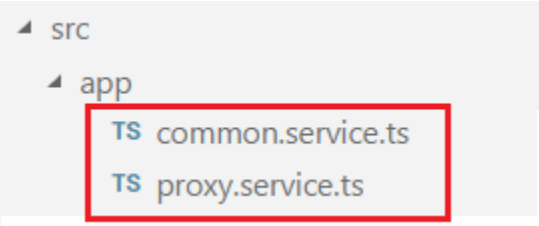
```
import { HttpClientModule } from '@angular/common/http';
@NgModule({
  declarations: [
    AppComponent
  ],
  entryComponents: [
    DeleteConfirmationComponent
  ],
  imports: [
    BrowserModule,
    FormsModule,
    HttpClientModule,
```

*. The Good news is that you can find under the Patch downloaded by code Booster (option "002"), a file name proxy.service.ts as the following image



*. **Just copy/paste this file under the "App" folder of your Angular Project.**

*. and create another file named 'common.service' , it should also exist near the proxy.service.ts file under the Angular Application as the following



*. Open the common.service.ts file and assure you have the following code.

N.B: Change the APIUrl property to point to your Web API, as the following:

```
import { Injectable } from '@angular/core';

@Injectable()
export class CommonService {
  public APIUrl = 'http://localhost:5000/api/Data';
  public ticket = '';
  constructor() { }

  ShowMessage(message: string, d: number = 1000) {
    alert(message);
  }
  Handle_Exception(msg) {
    if ((msg != null) && (msg !== '')) {
      this.ShowMessage(msg , 3000);
    }
  }
}
```

*. common.service.ts & proxy.service.ts represent 2 services and any service should be declared in the app.module.ts as the following (in the providers array)

```
providers: [
            Proxy,
            CommonService
           ],
```

*. Now, in any angular component, just inject the proxy service in the constructor as dependency as the following.

```
constructor(private proxy: Proxy) {}
```

*. Now, you can call any Web API exposed method from your angular project, as you did from Postman application.

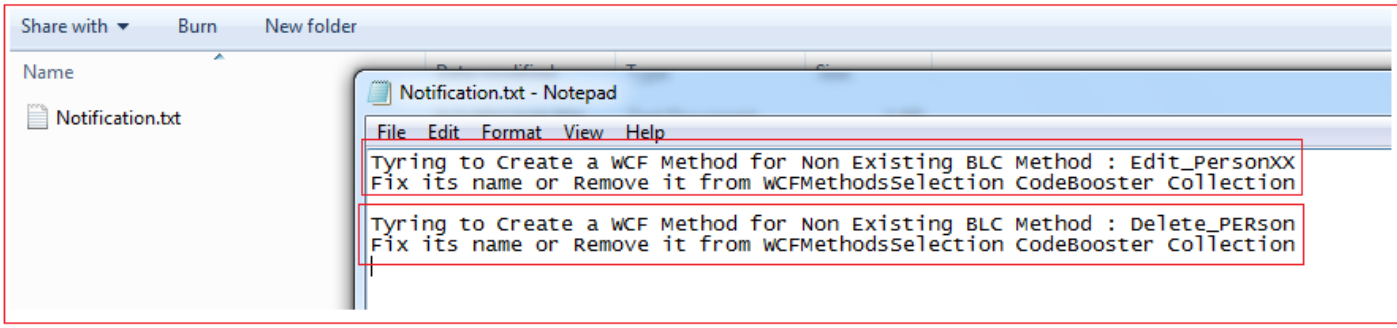
```
constructor(
  private proxy: Proxy
) {
    const p = new Params_Get_Person_By_OWNER_ID();
    p.OWNER_ID = 1;
    this.proxy.Get_Person_By_OWNER_ID(p).subscribe(data => alert(data));
}
```

*. If you try to generate Web API methods for non-existing BLC method, the code booster engine will send you notification instead of patch mentioning that you are trying to create Web API methods for non-existing BLC methods.

N.B: Method name is case sensitive

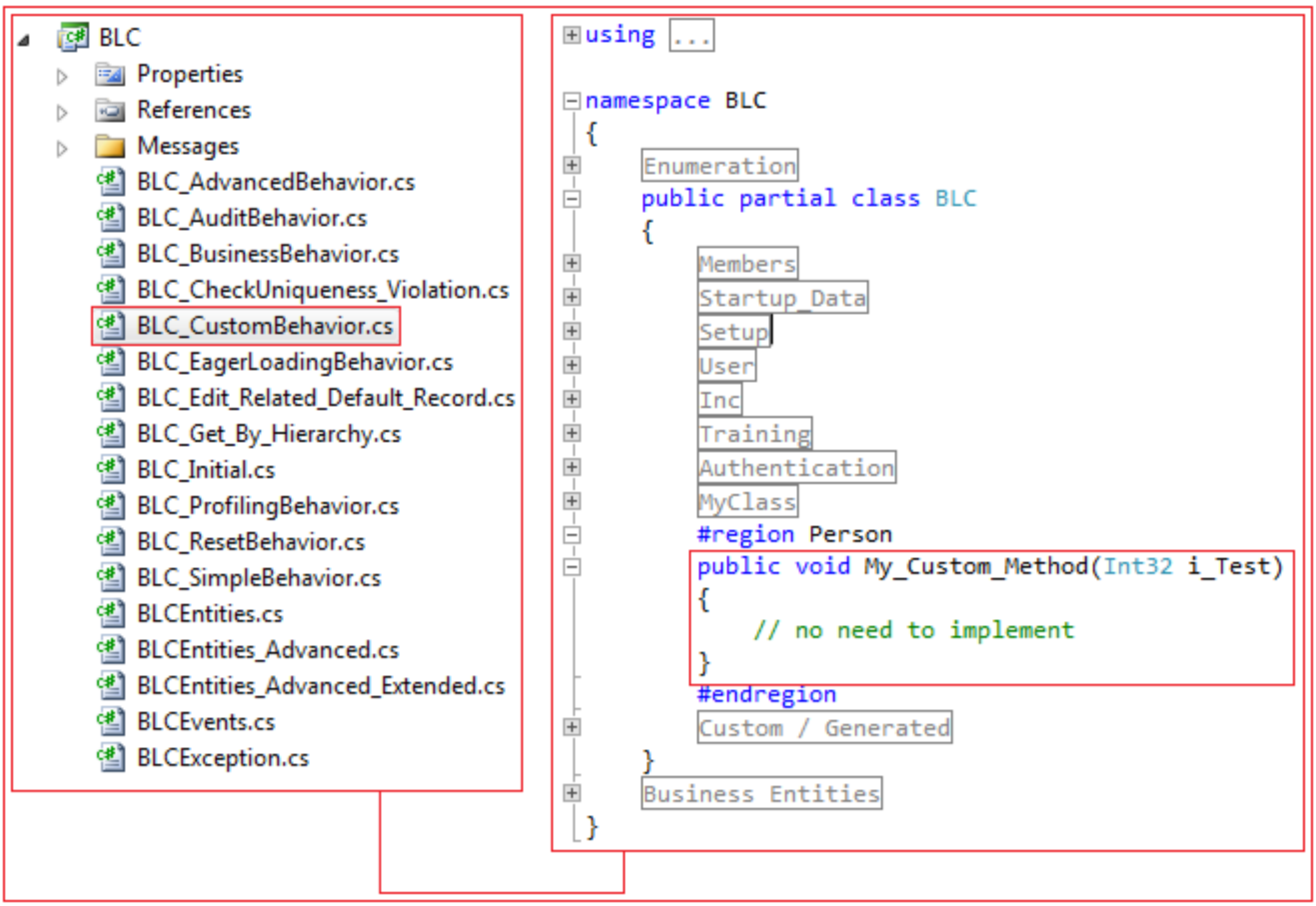
```
// non existing methods
// ------------------------------------------------------------------------
oCodeBooster.WCFMethodsSelection.Add("Edit_PersonXX");
oCodeBooster.WCFMethodsSelection.Add("Delete_PERson");
// ------------------------------------------------------------------------
```

```
L1
Enter An Option:
001 --> Create SP's & BLC Layer
002 --> Generate WCF / JSON Code
003 --> Generate UI
002
Assure Required DB Objects existence : Start
Assure Required DB Objects existence : End
Assure Demo DB Objects Existence : Start
Assure Demo DB Objects Existence : End
Fetching DB Fields : Start
Fetching DB Fields : Done
Fetching DB Indexes : Start
Fetching DB Indexes : Done
Fetching DB FKs : Start
Fetching DB FKs : Done
Uploading DB Fields : Start
Uploading DB Fields : Done
Uploading DB Indexes : Start
Uploading DB Indexes : Done
Uploading FKs : Start
Uploading FKs : Done
Uploading Methods with Event : Start
Uploading Methods with Event : Done
Uploading WCF Methods : Start
Uploading WCF Methods : Done
Uploading Keys Mapper : Start
Uploading Keys Mapper : Done
Uploading BLC : Start
Uploading BLC : End
Downloading Patch : Patch-Notification-2013_8_3_18_16_50.rar : Start
Downloading Patch : Patch-Notification-2013_8_3_18_16_50.rar : End
Press Any Key To Exit
```

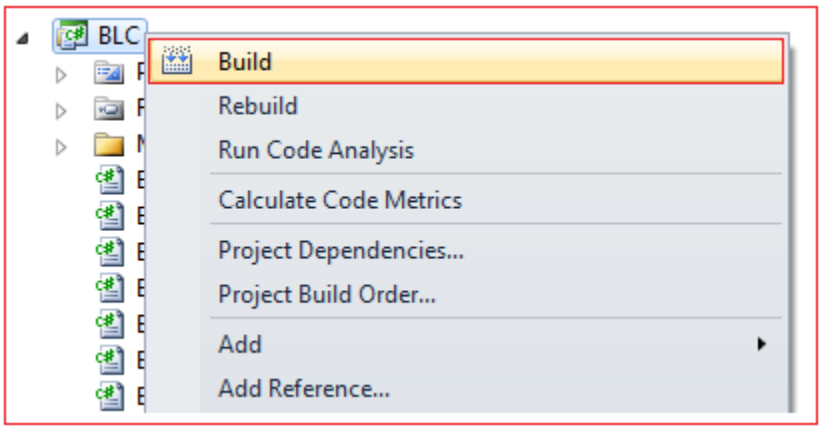


*. As a best practice, you should only expose BLC methods having one single parameter of complex type having all possible parameters as properties instead of sending many primitive types as parameter, in other terms if you try to ask the code booster to expose BLC method which is expecting a primitive type as parameter it (the code booster) will generate a notification instead of patch.

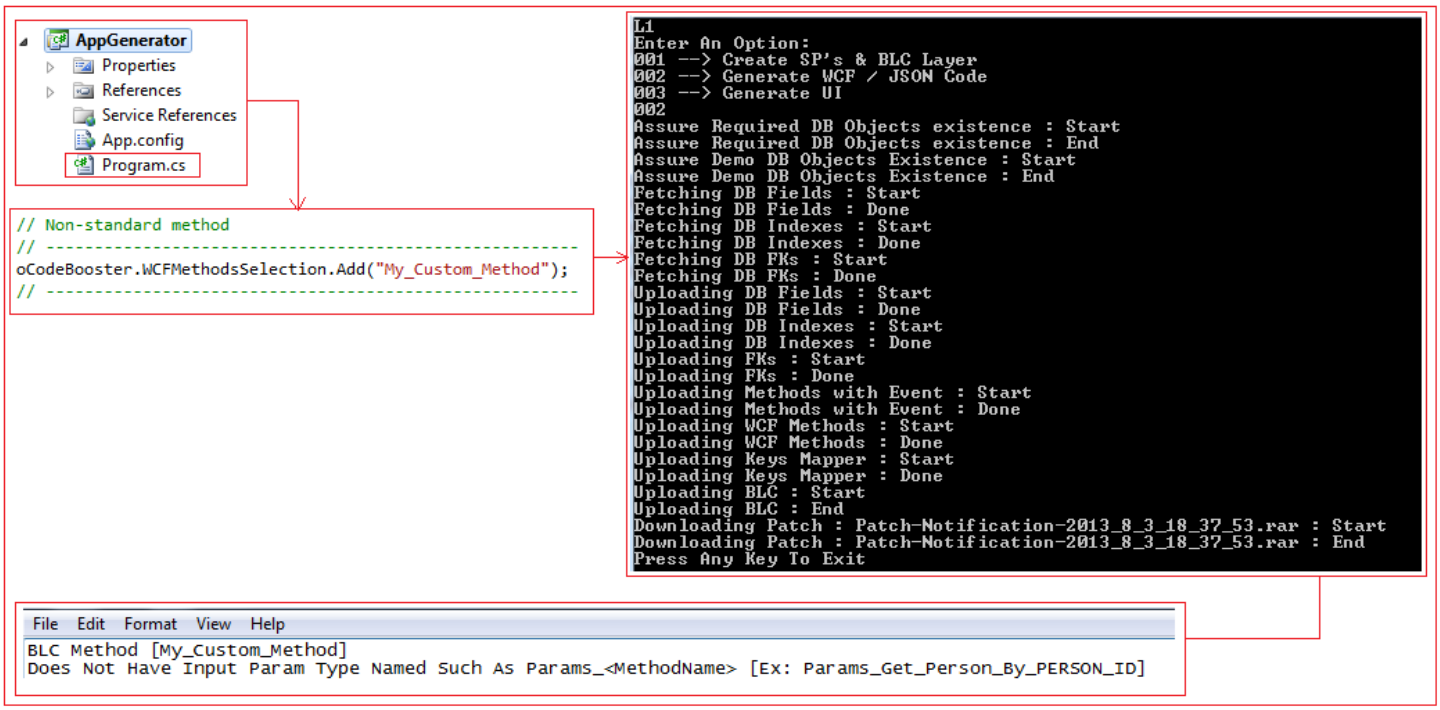
*. To clarify the previous point, let us create in the BLC project and specifically in BLC_CustomBehavior.cs file a custom method called My_Custom_Method expecting one primitive parameter.



*. Build the BLC project



*. In the AppGenerator console application (and specifically in program.cs) adds this custom method to the collection of BLC methods that you want the code booster to generate the corresponding API methods and then run option “002”



As you noticed you will get a notification mentioning that your method (in this case My_Custom_Method) should have a parameter (complex type) named Params_<Method Name>

. **Ticketing Mechanism**

By default, a Web API published on a certain IP will be accessed by anyone who knows the URL, and this is something that we don't want to happen in our enterprise level applications.

What I am trying to show you here is how to secure your API from being called by non-authorized code using something named ticketing.

Simply, no one can call your WebAPI unless the caller has a certain ticket (encrypted string) that should be passed in the URL as query string parameter each time the caller wants to interact with your WebAPI.

That ticket should be generated at a certain time, mainly upon login, by the server code that will send (in case the end user is authentic) a ticket that later one the caller will add it to the URL.

Using Code booster, this mechanism (ticketing) is totally handled (WebAPI Generation) while allowing you to implement your custom ticket generation.

The whole checking can be stopped if you set in the App.config "ENABLE_TICKET = 0" (or totally removed) as you noticed in the code, the ticketing mechanism will only be applied if ENABLE_TICKET flag is set to 1

- In most of the applications, the ticket is generated and returned to the client code upon successful login, so in your authentication method (and after checking if it is an authentic user) return a ticket to the client which is a string that contains required data to identity the client on any later on WebAPI call, an example of Ticket is the following

```
USER_ID:1[~!@]OWNER_ID:1[~!@]START_DATE:2016-08-03[~!@]START_MINUTE:288[~!@]SESSION_PERIOD:240
```

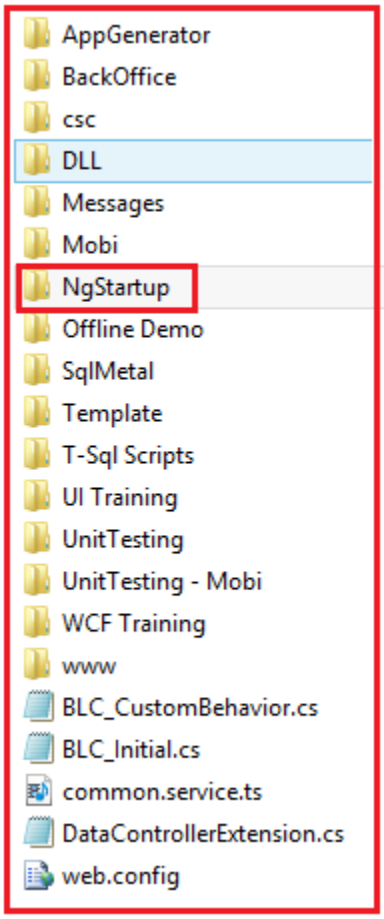
and all what you must do is to implement in your BLC a method named "ResolveTicket" that assures that the ticket is in correct format.

- In case you need to bypass the ticketing mechanism for certain WebAPI methods (ex: Forgot Password Action), you can inform code booster about that by filling the following collection by required methods

```
#region  ByPass Ticketing
oCodeBooster.List_ByPass_Ticketing = new List<string>();
oCodeBooster.List_ByPass_Ticketing.Add("Edit_Person");
#endregion
```

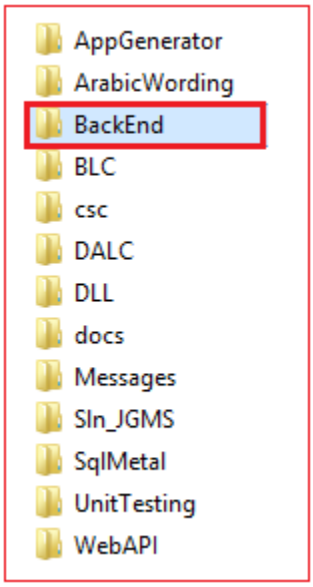
*. Now let's create a Web Application (Angular Based).

Under the CB Folder (already downloaded), you will find a folder named “NGStartup" as the following

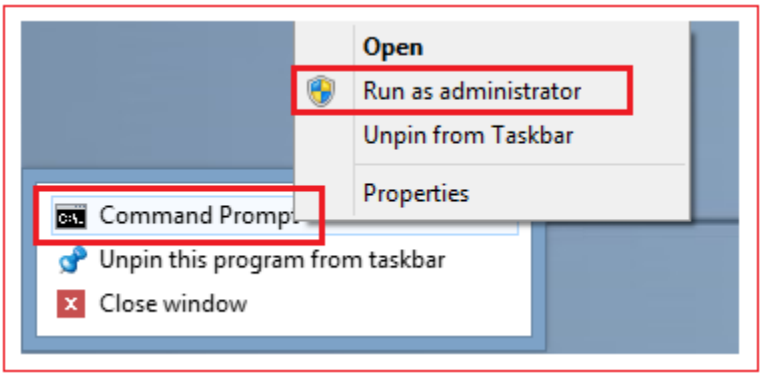


*. Copy the "NGStartup" folder under your already created "App" folder.

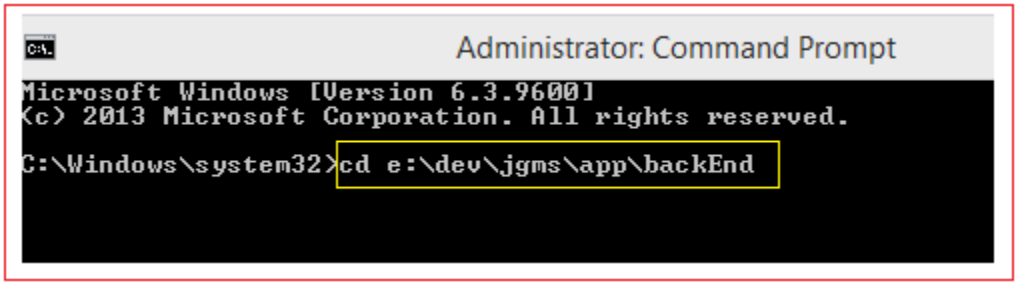
*. Rename the "NGStartup" folder to "BackEnd" as the following



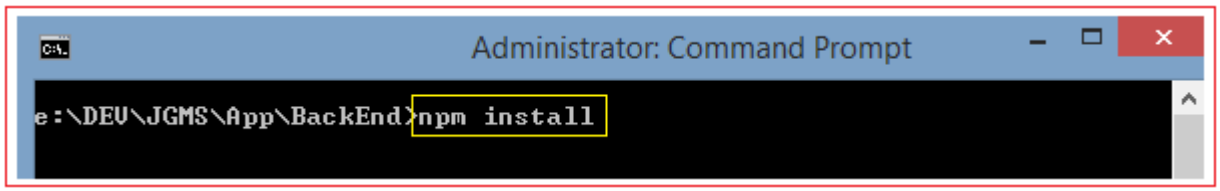
*. Open the command line (cmd) as administrator.



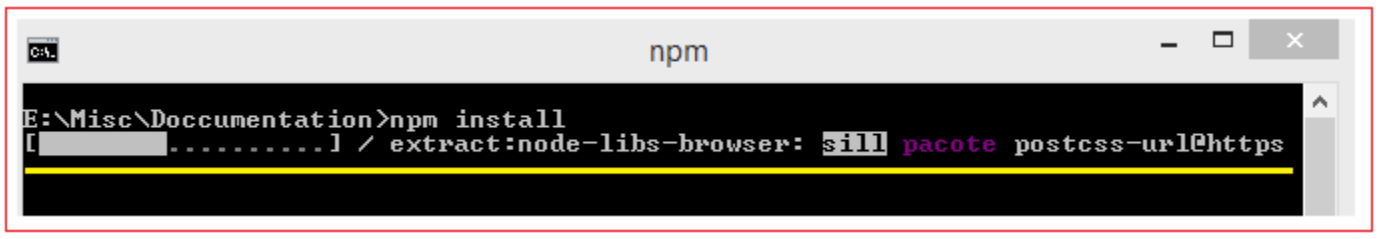
*. In the command line, Change the directory to point to the "BackEnd" folder you just created



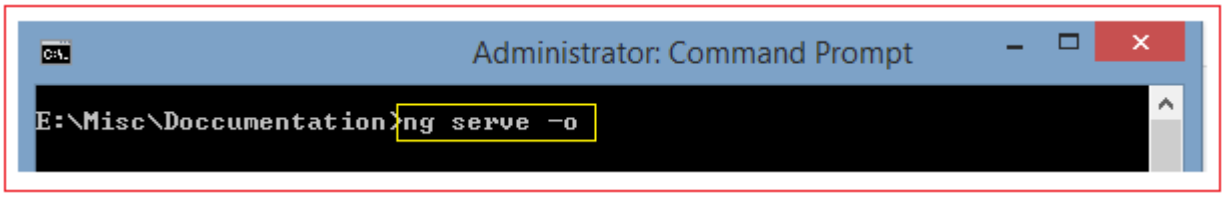
*. After being in the BackEnd folder (from the command Line), run "npm install" command.



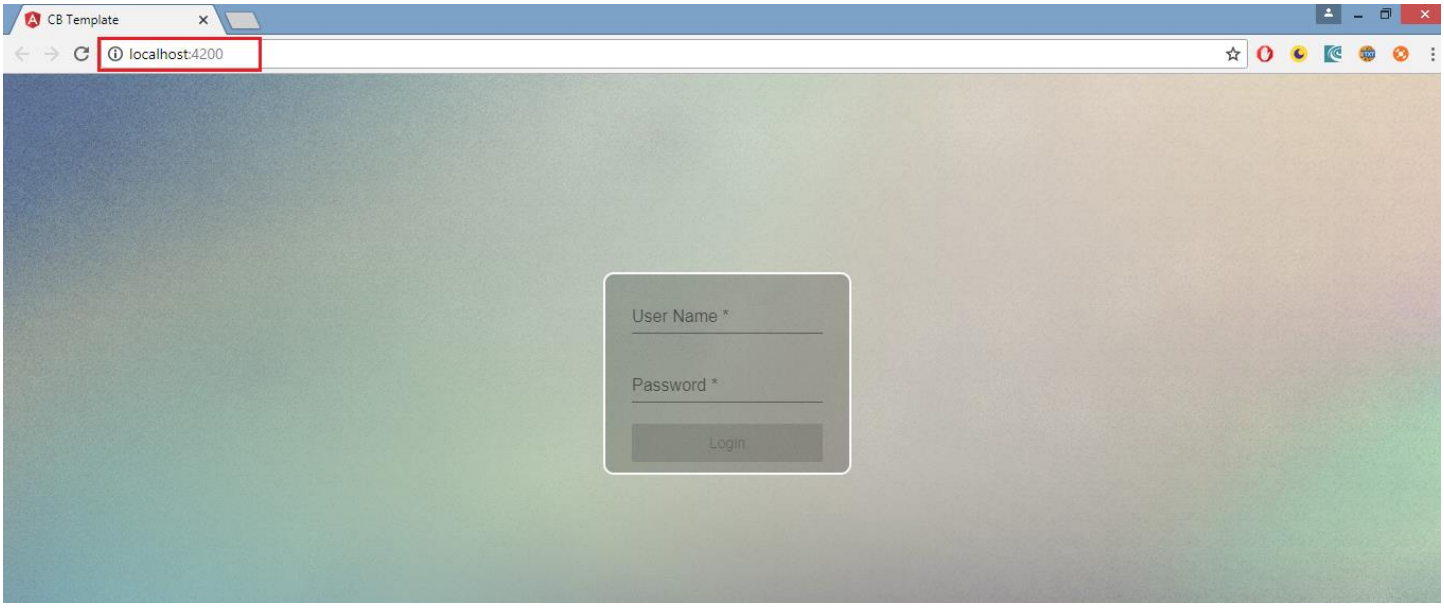
*. You have to wait till the installation process (of all required packages) ends.



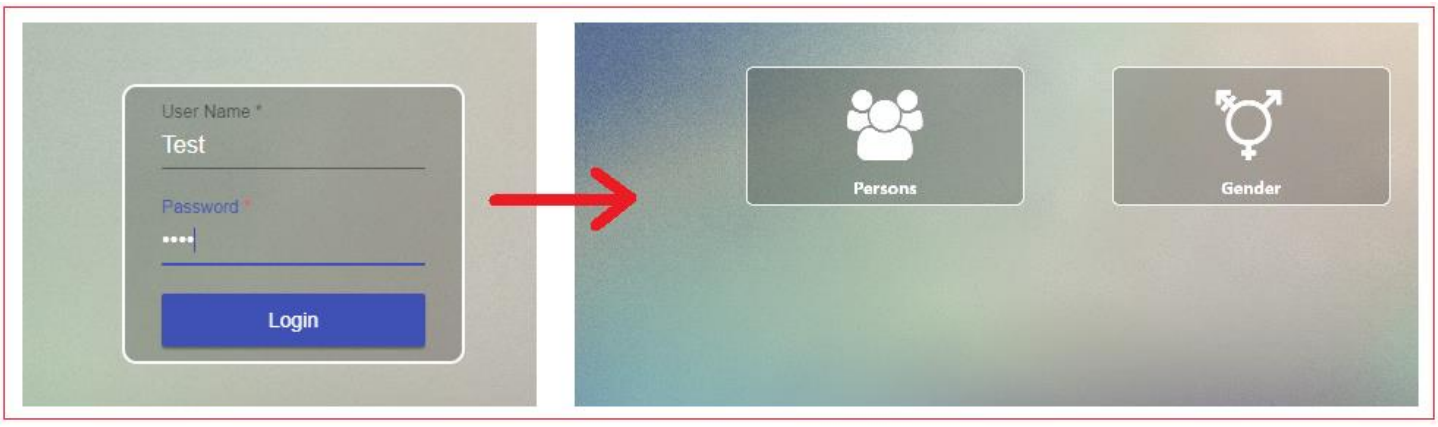
*. Once the npm installation is done, issue the following command (ng serve -o)



and your default browser [Hopefully Google Chrome :)] will open the following web application



*. Enter whatever you want as username & password, and press "Login" so you should see the following page.

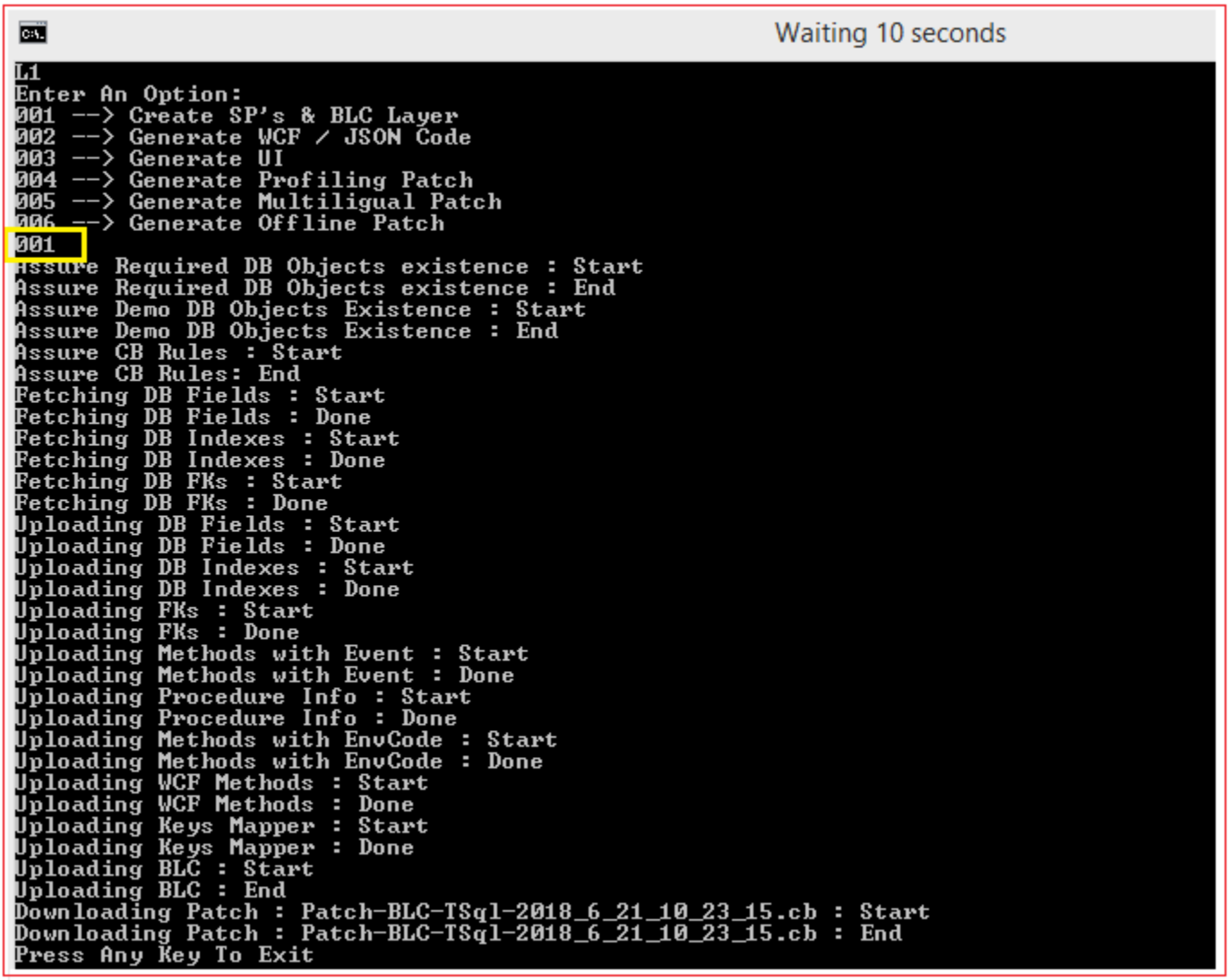


*. Now, assure you have a table named TBL_CURRENCY in your database, having the following structure

TBL_CURRENCY

Column Name	Data Type	Allow Nulls
CURRENCY_ID	int	☐
NAME	nvarchar(50)	☐
SYMBOLE	nvarchar(10)	☐
		☐

*. Generate BLC/T-SQL Patch (by running AppGenerator and selection option "001")



After the patch is downloaded, assure that the following steps are done:

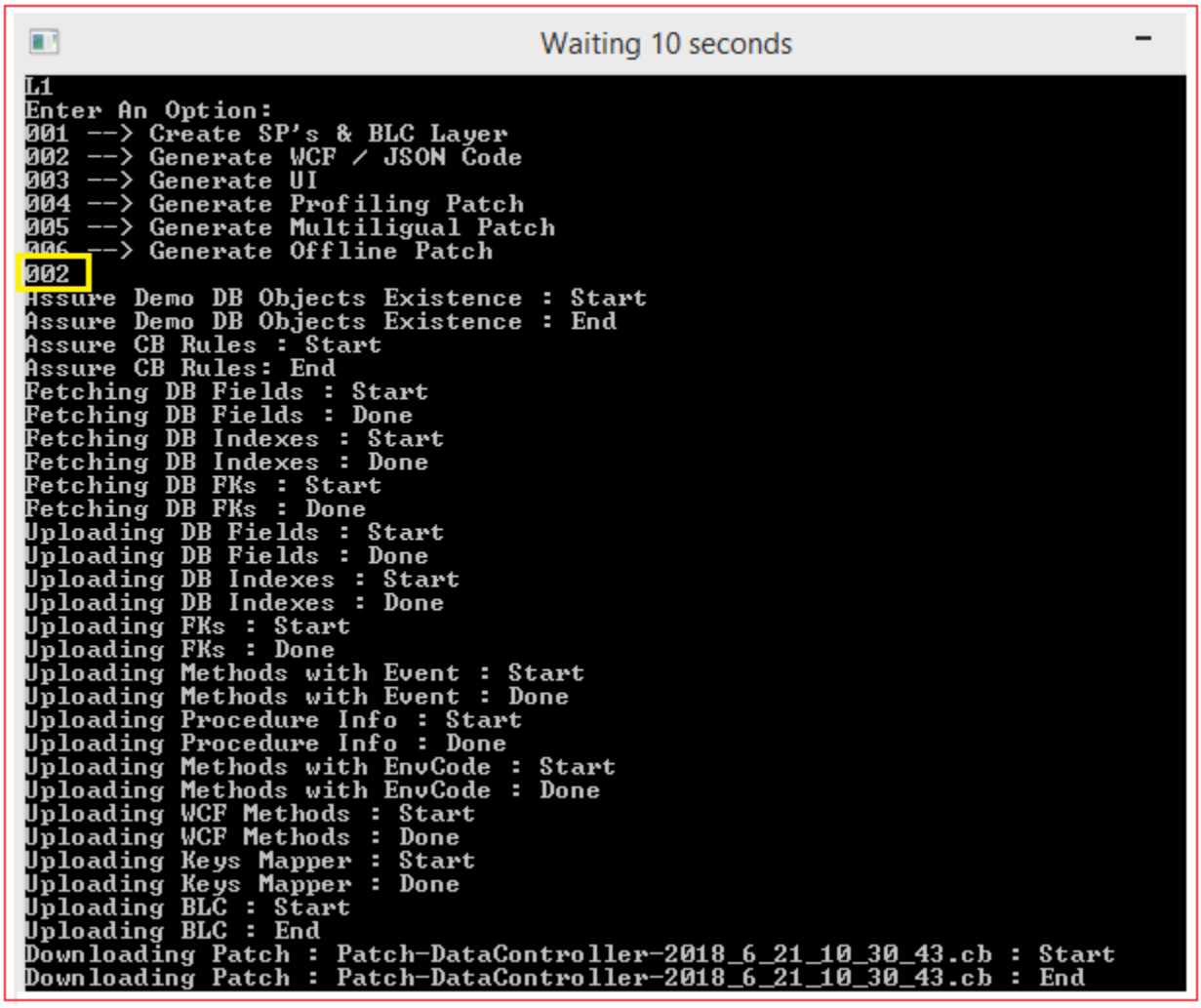
1. Apply T-Sql on database
2. run Sql Metal to generate DALC.cs file (and override it under the DALC Project)
3. Override BLC Files under the BLC Project

*. Now, in order to be able to create a screen to manage the TBL_CURRENCY table in our database, you must expose the following BLC Methods as Web API Methods

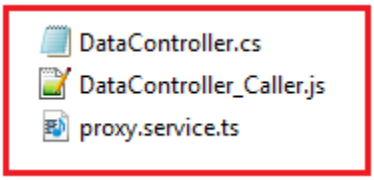
N.B: This the following code should reside in the "region API" in the AppGenerator / program.cs file.

```
oCodeBooster.API.MethodsSelection.Add("Edit_Currency");
oCodeBooster.API.MethodsSelection.Add("Delete_Currency");
oCodeBooster.API.MethodsSelection.Add("Get_Currency_By_OWNER_ID");
```

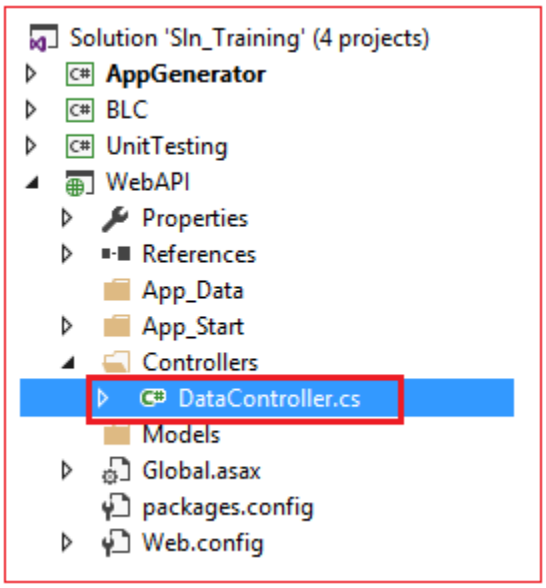
*. Run the AppGenerator console application and select option "002" to generate the Web API (Data Controller) and the corresponding Proxy.service.ts file.



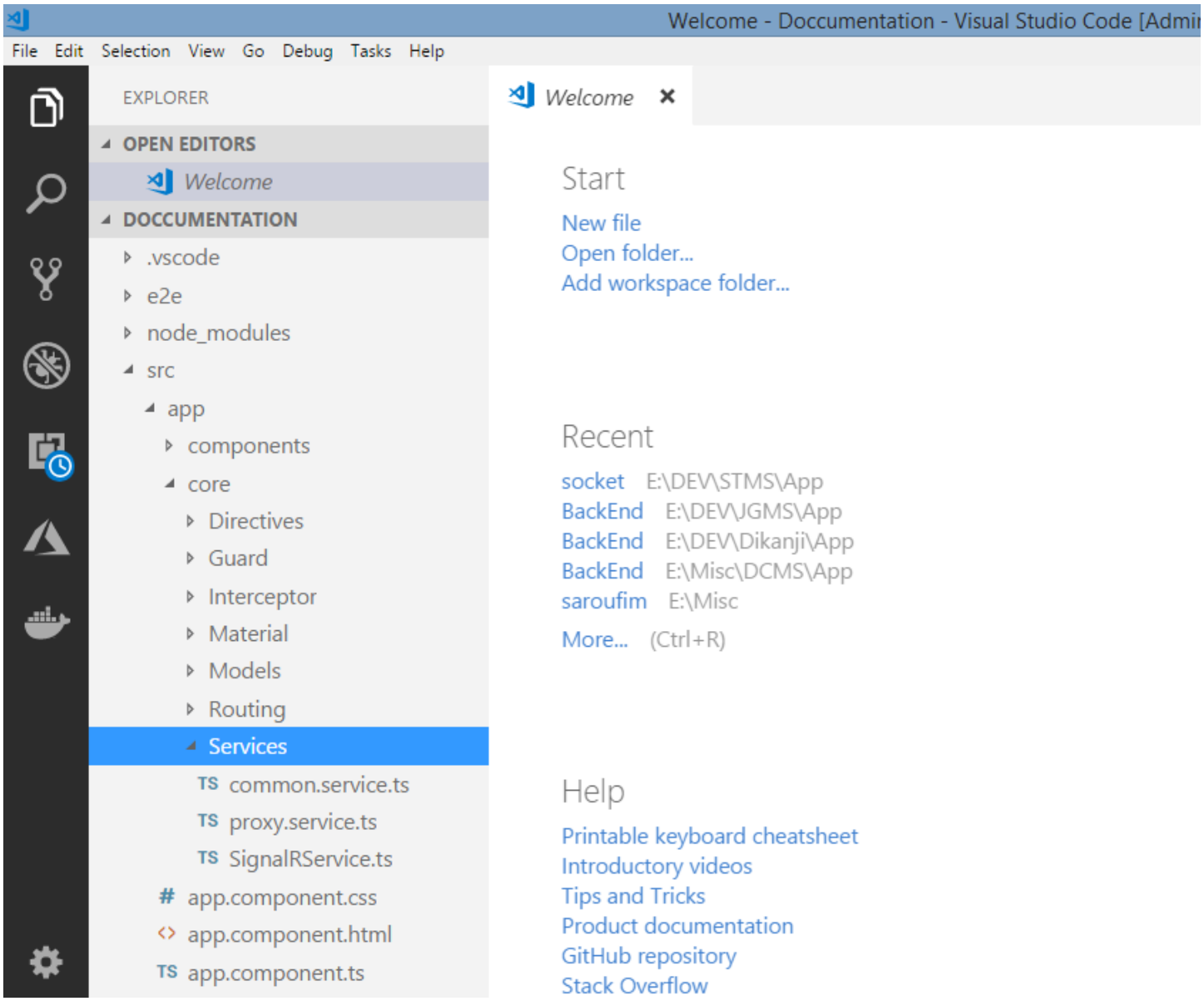
*. Once the patch is downloaded, extract the zip file and you will find the following file.



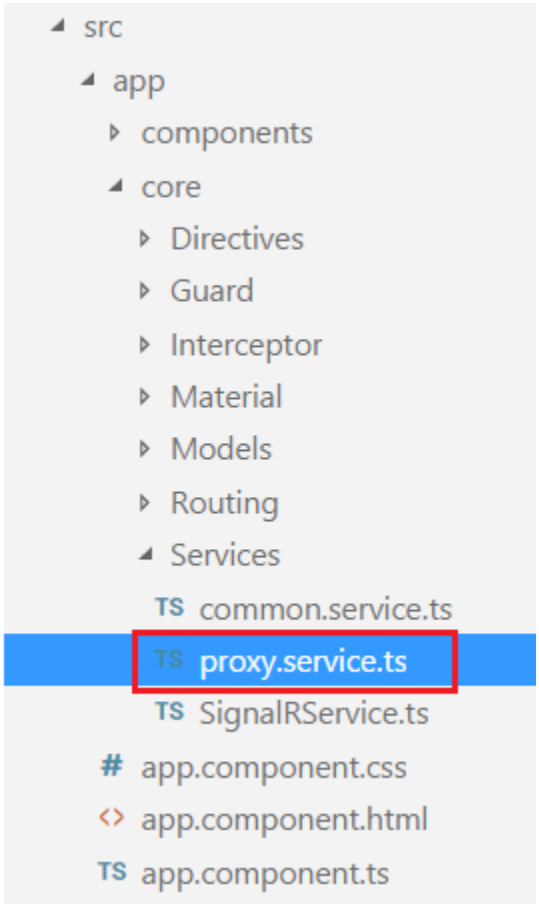
*. Override the DataController.cs file under the "Controllers" folder of the WebAPI project.



*. Open the Web Application source code using Visual Studio Code application (You can download it for free)



*. Now, override the proxy.service.ts file (included in the patch) under **src\app\core\services** folder as the following

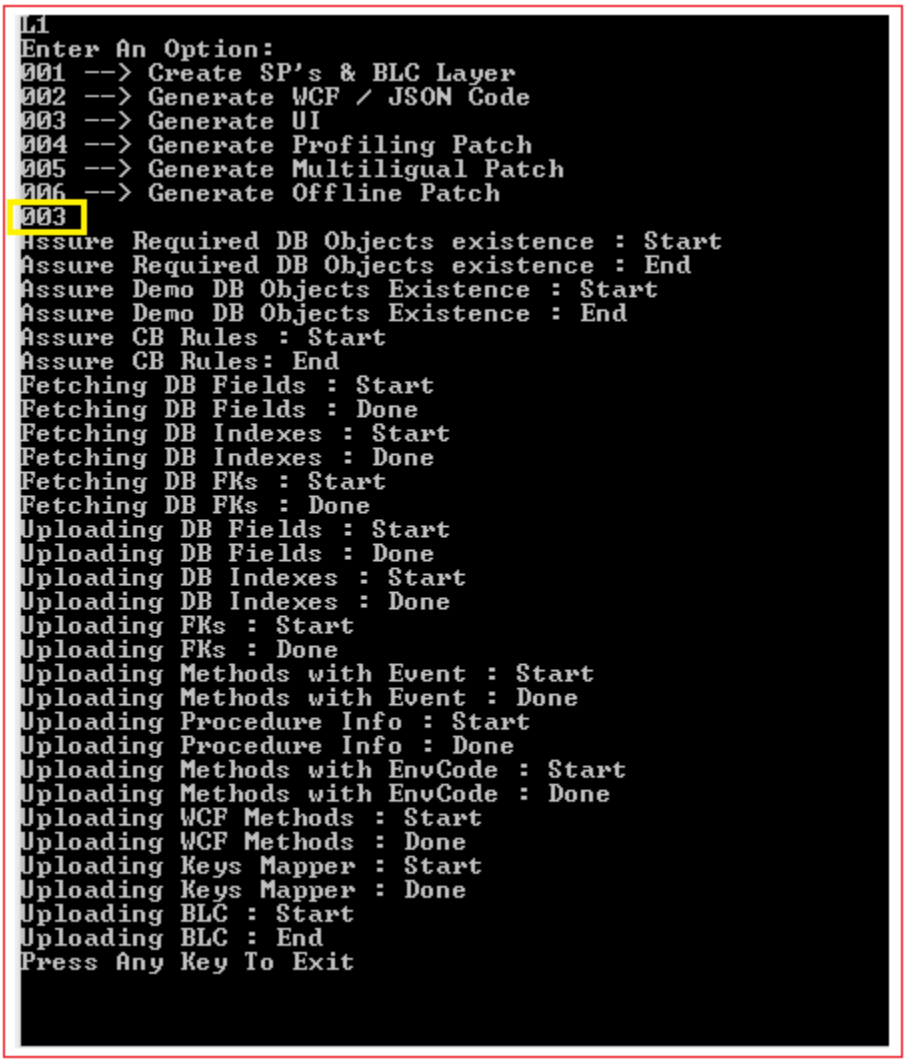


*. Open the common.service.ts file under **src\app\core\services** folder , and change the Web API URL to point to your API

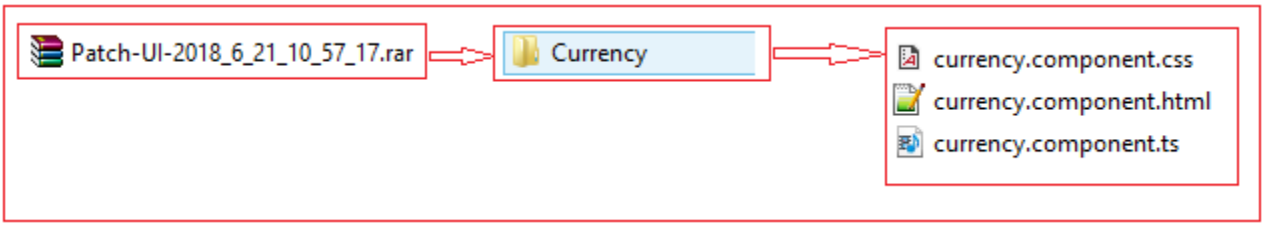
*. Now, let's do something new, by asking code booster engine to generate for us an angular component to handle (create, update, delete and list TBL_CURRENCY entries) by writing the following in option "003" section in AppGenerator/program.cs file

```
#region Currency
#region Search Screen
oUIFields_Criteria = new UIFields();
oUIFields_Criteria.MainTableName = "[TBL_CURRENCY]";
oUIFields_Criteria.Based_On_Type = "BLC.Params_Get_Currency_By_OWNER_ID";

oUIFields_Result = new UIFields();
oUIFields_Result.MainTableName = "[TBL_CURRENCY]";
oUIFields_Result.Based_On_Type = "BLC.Currency";
oUIFields_Result.GetMethodName = "Get_Currency_By_OWNER_ID";
oUIFields_Result.GridFields = new List<GridField>();


oSearch_AdvancedProp = new Search_AdvancedProp();
oSearch_AdvancedProp.ContainerMargins = "0,5,0,5";
oCodeBooster.Entity_FriendlyName = "Currency";
oCodeBoosterClient.Generate_ListUI(Enum_SearchMethod.Without_Criteria_Section, oUIFields_Criteria, oUIFields_Result, oSearch_AdvancedProp);
#endregion
#endregion
```

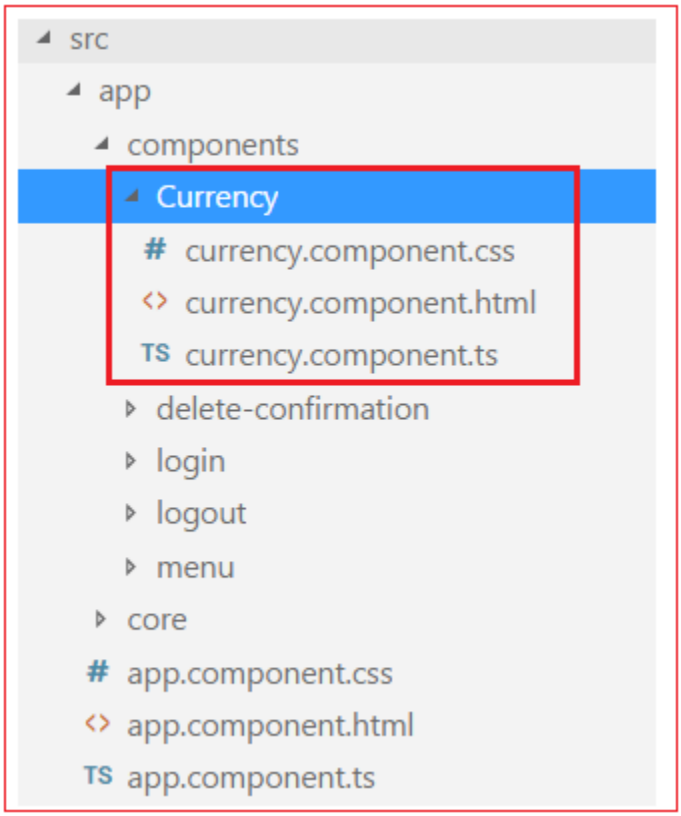
*. Run AppGenerator console application and select option "003"



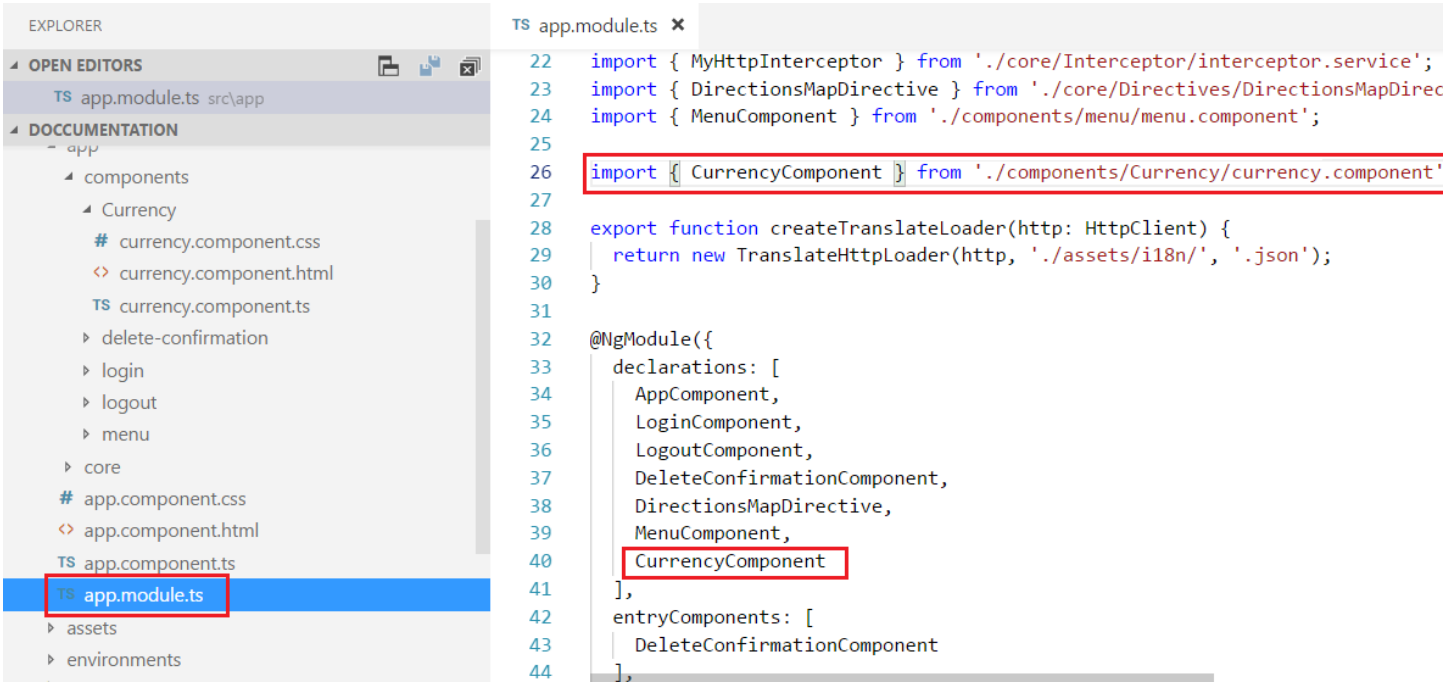
and the following patch will be download containing the Currency angular component (.ts, .css & .html files)



*. All What you have to do is to copy "Currency" folder under **src\app\components** folder as the following.

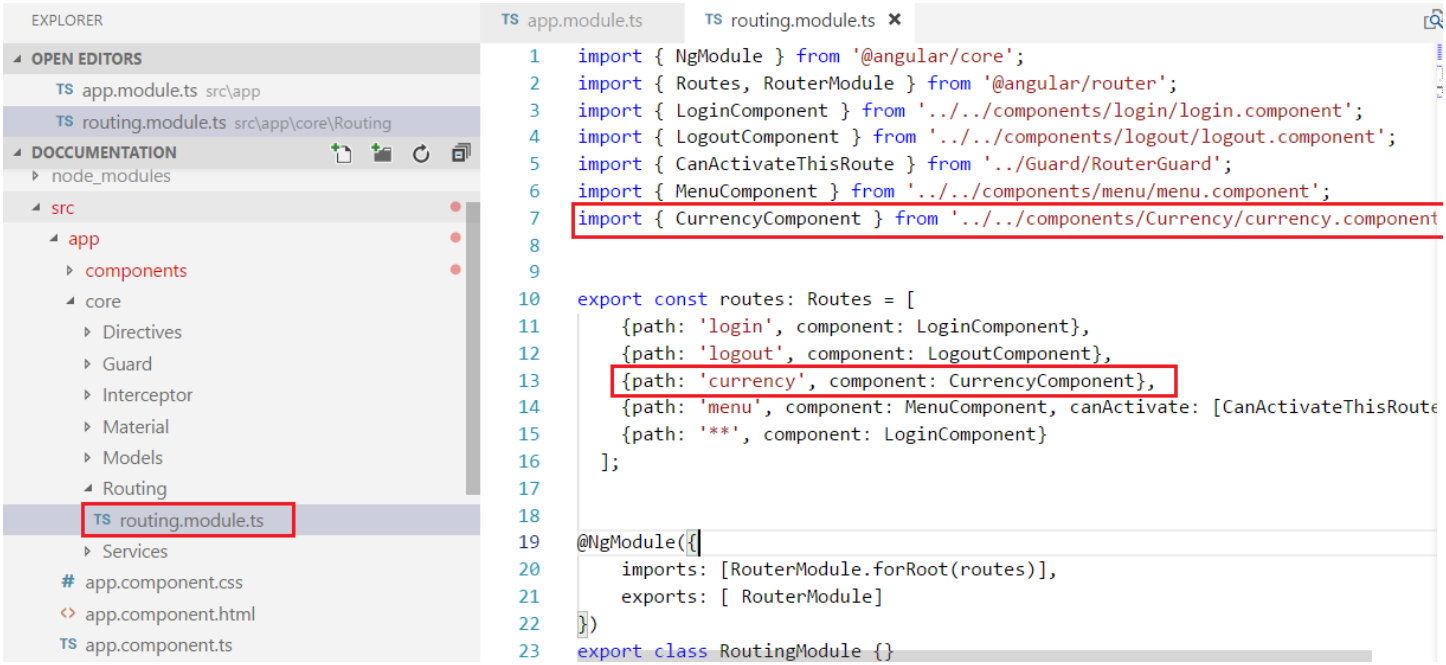


*. As Like any other angular component, we have to add it to declarations array in the main module, so open the App.module.ts file and assure that Currency Component is listed in the declarations array as the following

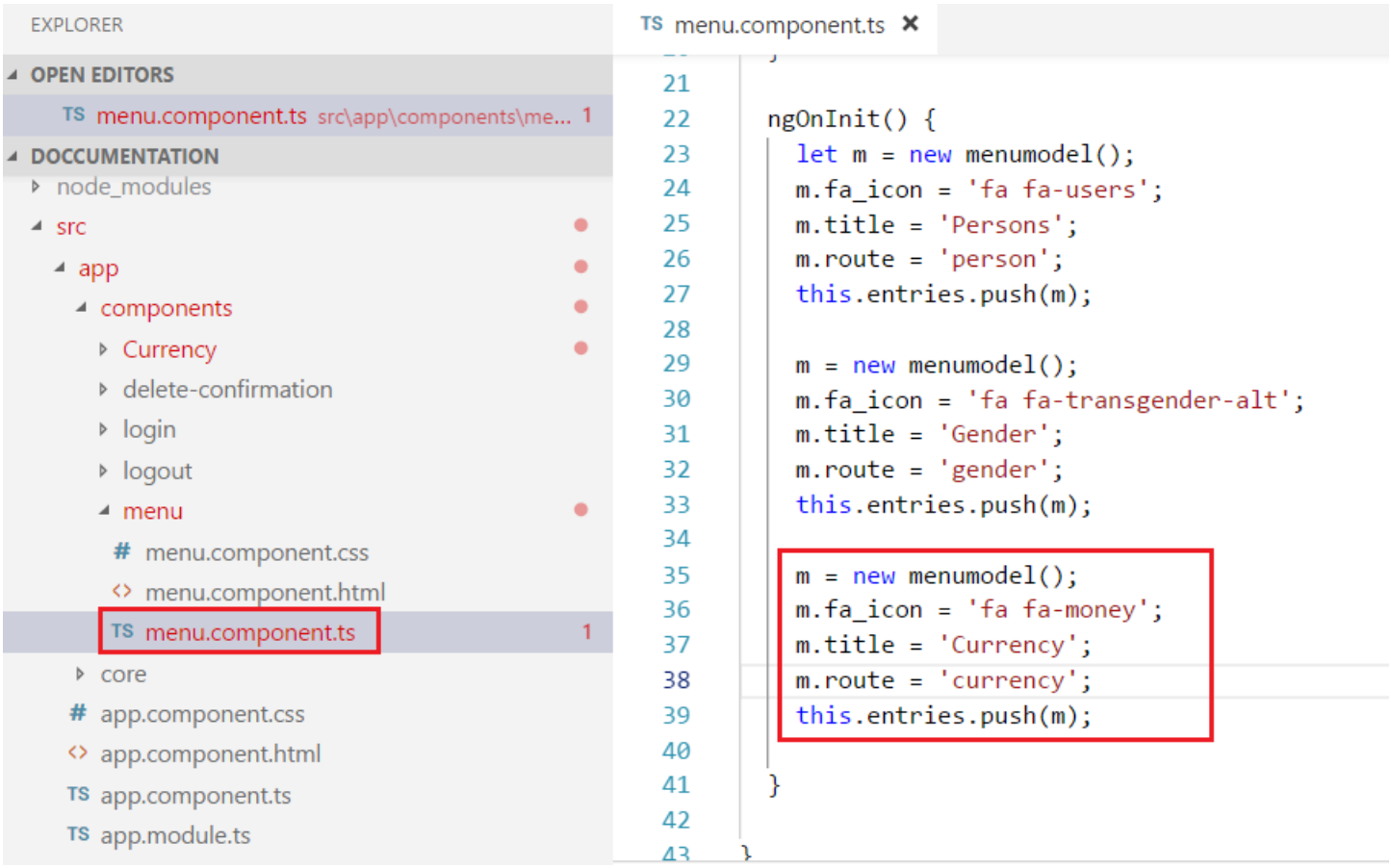


*. Now, let's add the Currency component to the routing mechanism by doing the following.

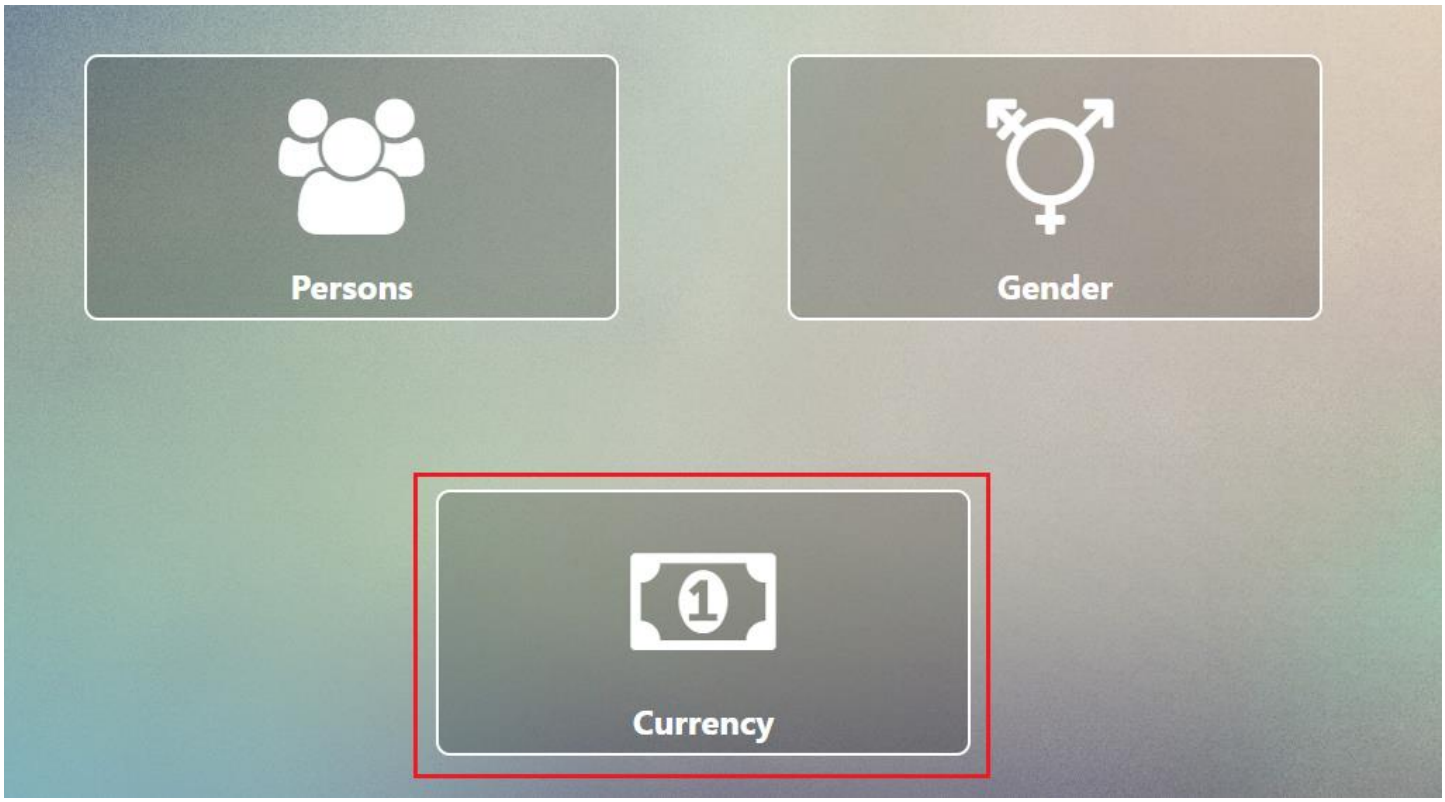
(Open the routing.module.ts file and add in routes array the currency entry)



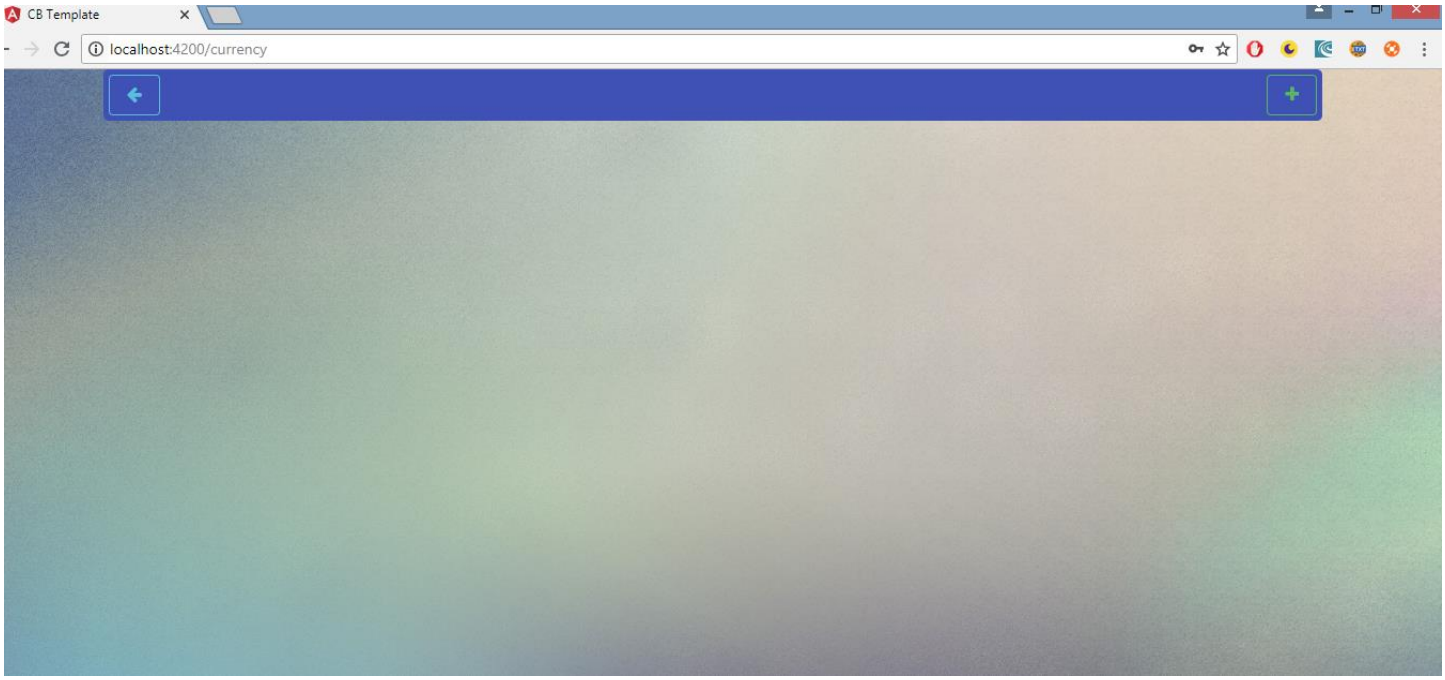
*. The final step, and in order to show a new menu entry in the web application, open the menu.component.ts file (under **src\app\components** folder) and add the following:



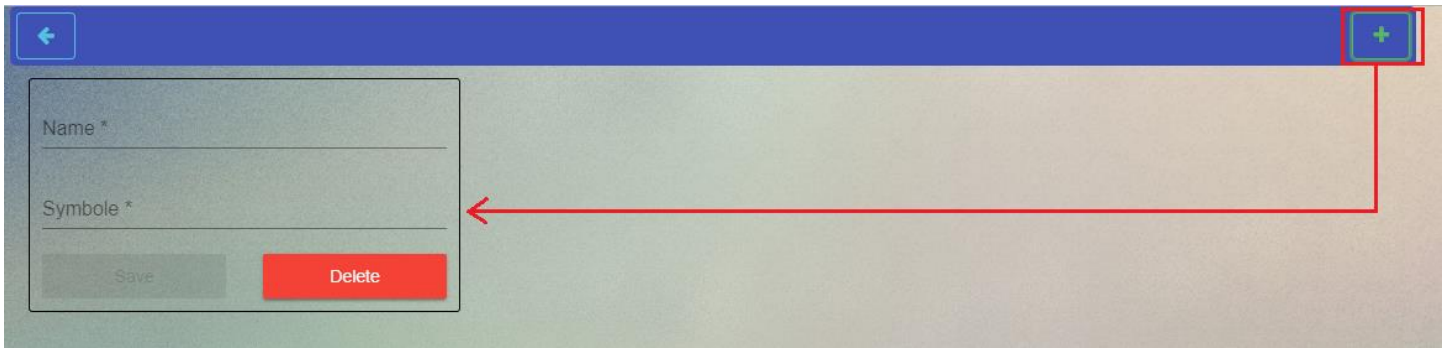
*. Now, visit the following url (**http://localhost:4200**) and after entering whatever you want in user name & password, press login and the following screen (menu) will be showing a new entry named "Currency"



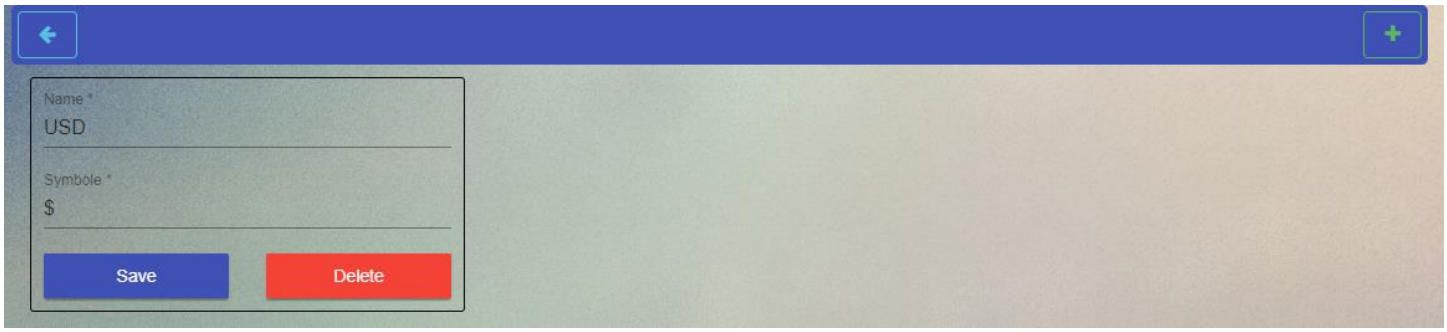
*. Press on "Currency" and the following page will appear.



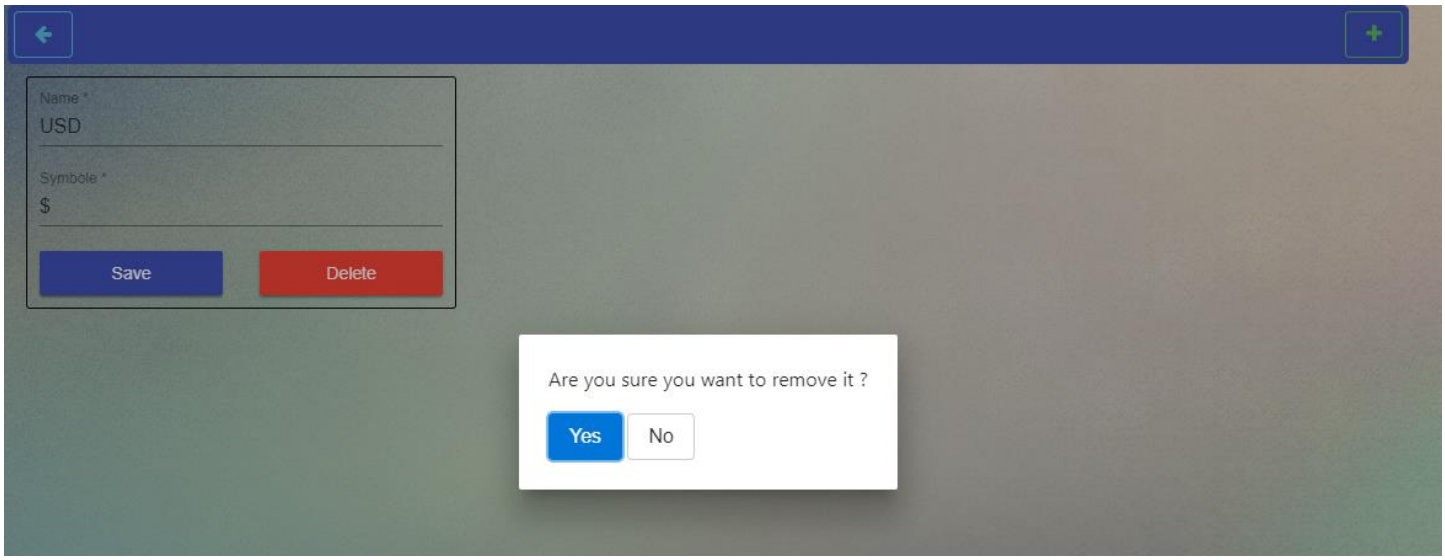
*. Sure, no currencies will be shown by now: In order to add a new Currency just press on the "+" icon (top right corner) and the following will be shown.



*. Just enter a value for "Name" & "Symbol" files and press Save button and data will be saved in database



*. In order to delete this new currency, just press on delete button (confirmation message will appear)



*. Let's add a new table to your database named TBL_PRODUCT with the following structure.

TBL_PRODUCT

Column Name	Data Type	Allow Nulls
PRODUCT_ID	int	☐
NAME	nvarchar(50)	☐
PRICE	decimal(18, 3)	☐
CURRENCY_ID	int	☐
DESCRIPTION	nvarchar(MAX)	☐
		☐

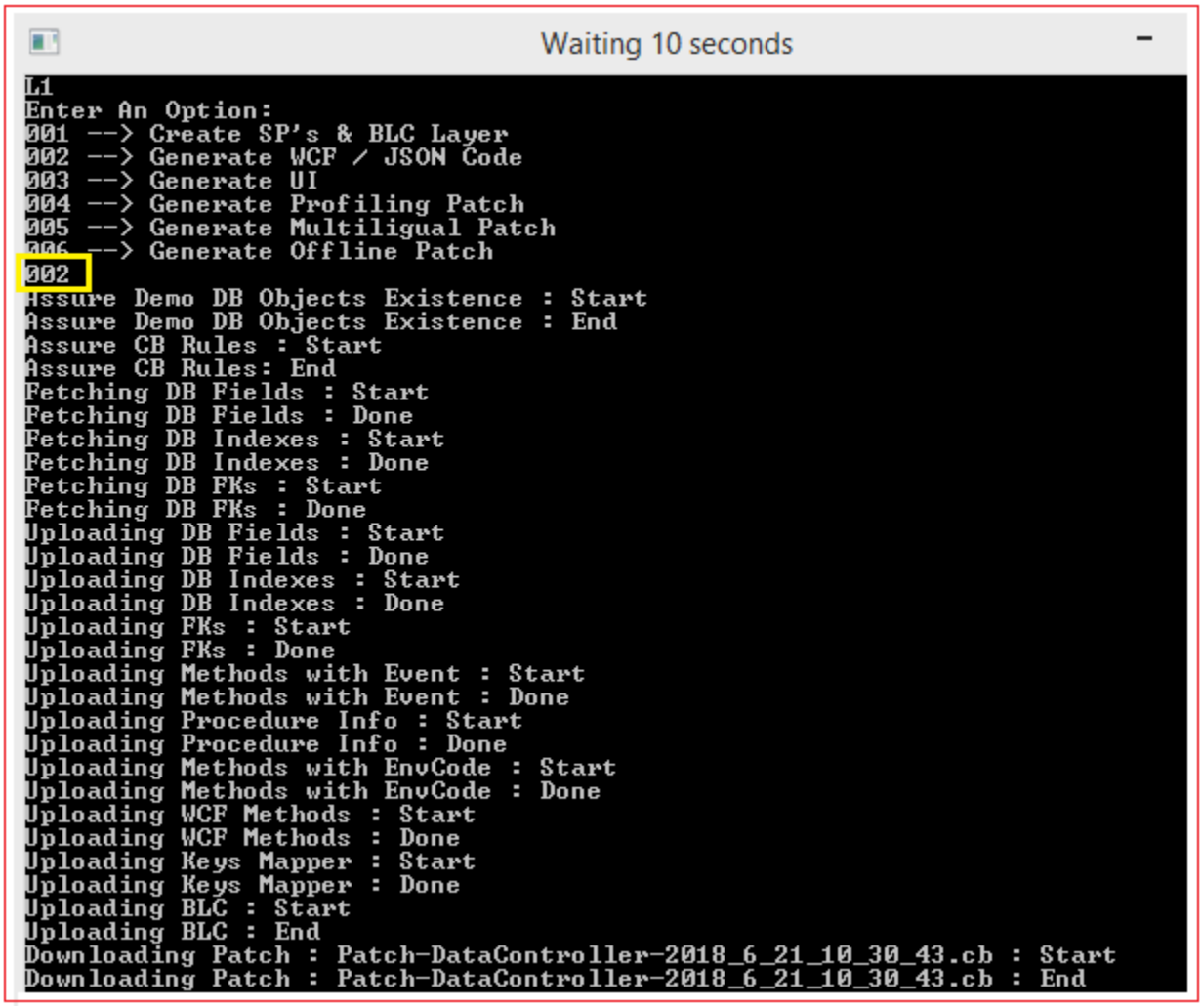
*. As we did for TBL_CURRENCY, we have to run AppGenerator console application with option "001" and the following steps should be applied.

1. Apply T-Sql Script
2. Override the files that exist under the DALC folder in the DALC Project
3. Override BLC Files under the BLC Project

*. Let's expose the following BLC Methods as Web API Methods, by adding them to program.cs file of AppGenerator Console Application

```
// Currency
//----------------------------
oCodeBooster.APIMethodsSelection.Add("Edit_Currency");
oCodeBooster.APIMethodsSelection.Add("Delete_Currency");
oCodeBooster.APIMethodsSelection.Add("Get_Currency_By_OWNERE_ID");
//----------------------------

// Product
//----------------------------
oCodeBooster.APIMethodsSelection.Add("Edit_Product");
oCodeBooster.APIMethodsSelection.Add("Delete_Product");
oCodeBooster.APIMethodsSelection.Add("Get_Product_By_Where");
//----------------------------
```

*. Run the AppGenerator console application and select option "002" to generate the Web API (Data Controller) and the corresponding Proxy.service.ts file.



*. As we did previously, you have to override DataController.cs file under Web API Project & proxy.service.ts file in the Angular Web Application.

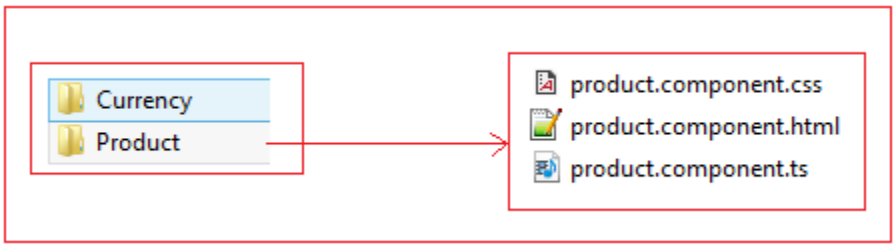
*. Now, Let's generate angular component to manage TBL_PRODUCT table

N.B: Since we might have a lot of products, we must be able to fetch by name or any field in the TBL_PRODUCT table.

Add the following to AppGenerator/program.cs file, in the region titled "003" as we did for TBL_CURRENCY

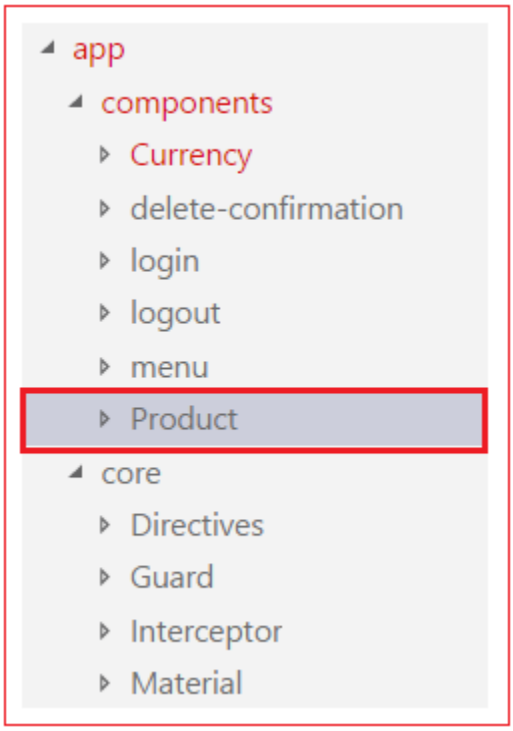
```
#region Product
#region Search Screen
oUIFields_Criteria = new UIFields();
oUIFields_Criteria.MainTableName = "[TBL_PRODUCT]";
oUIFields_Criteria.Based_On_Type = "BLC.Params_Get_Product_By_Where";

oUIFields_Result = new UIFields();
oUIFields_Result.MainTableName = "[TBL_PRODUCT]";
oUIFields_Result.Based_On_Type = "BLC.Product";
oUIFields_Result.GetMethodName = "Get_Product_By_Where";
oUIFields_Result.GridFields = new List<GridField>();


oSearch_AdvancedProp = new Search_AdvancedProp();
oSearch_AdvancedProp.ContainerMargins = "0,5,0,5";
oCodeBooster.Entity_FriendlyName = "Product";
oCodeBoosterClient.Generate_ListUI(Enum_SearchMethod.With_Criteria_Section, oUIFields_Criteria, oUIFields_Result, oSearch_AdvancedProp);
#endregion
#endregion
```

*. Run AppGenerator and choose option "003", and a patch will be downloaded containing the Product angular component as the following.

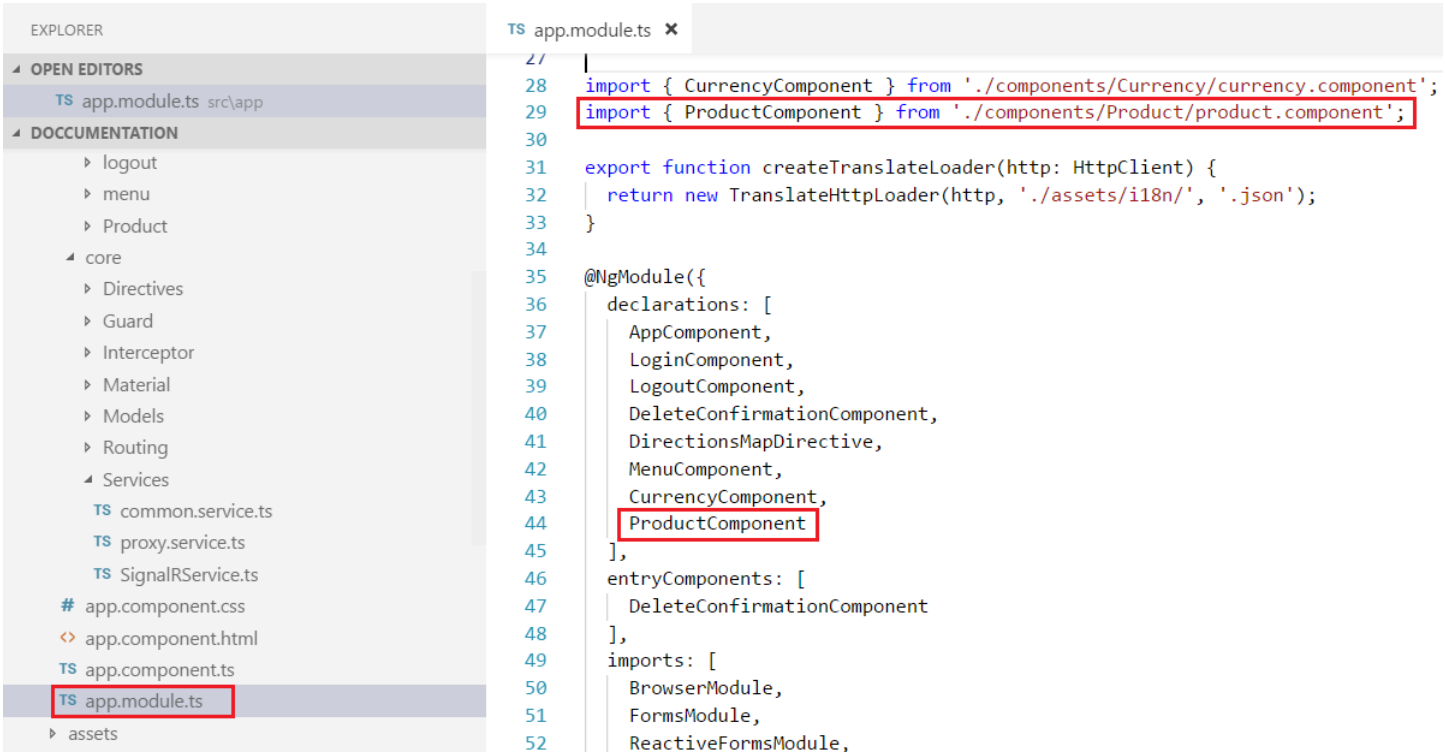


*. Now, we must repeat the same steps to integrate the Product angular component in the web application:

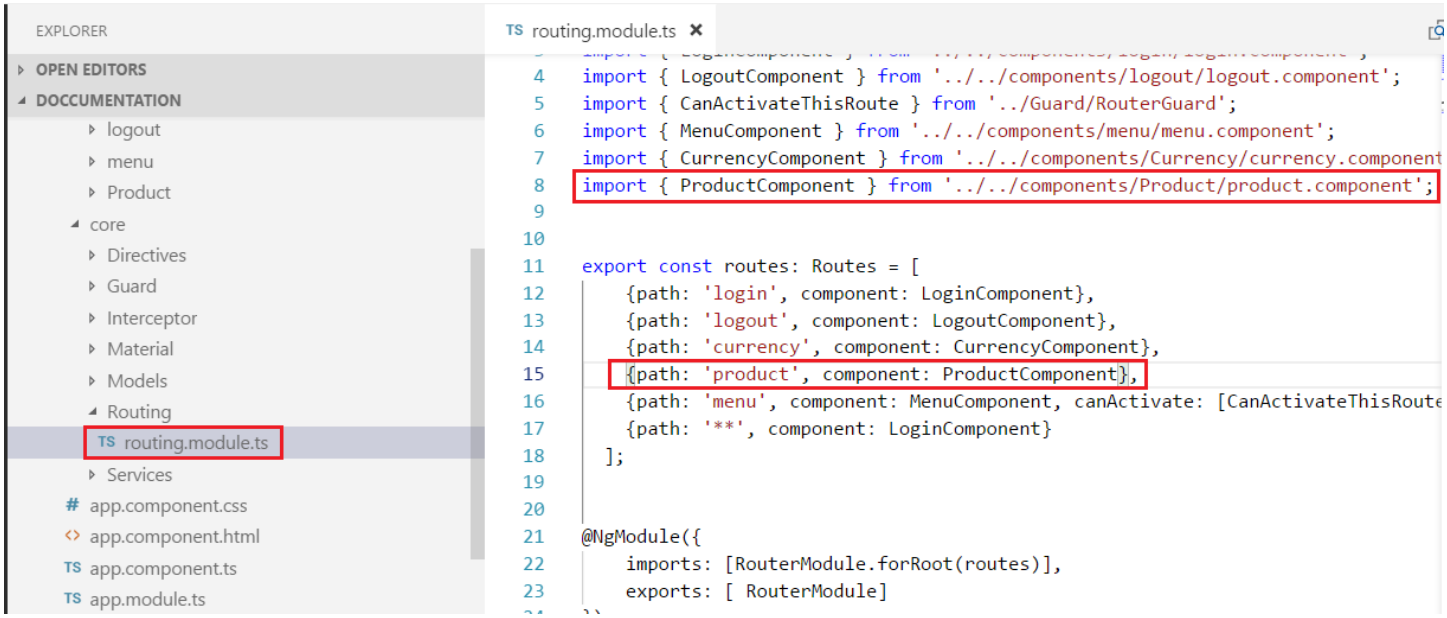
1. Copy the Product folder (containing the .ts, .css & .html files) under the components folder.



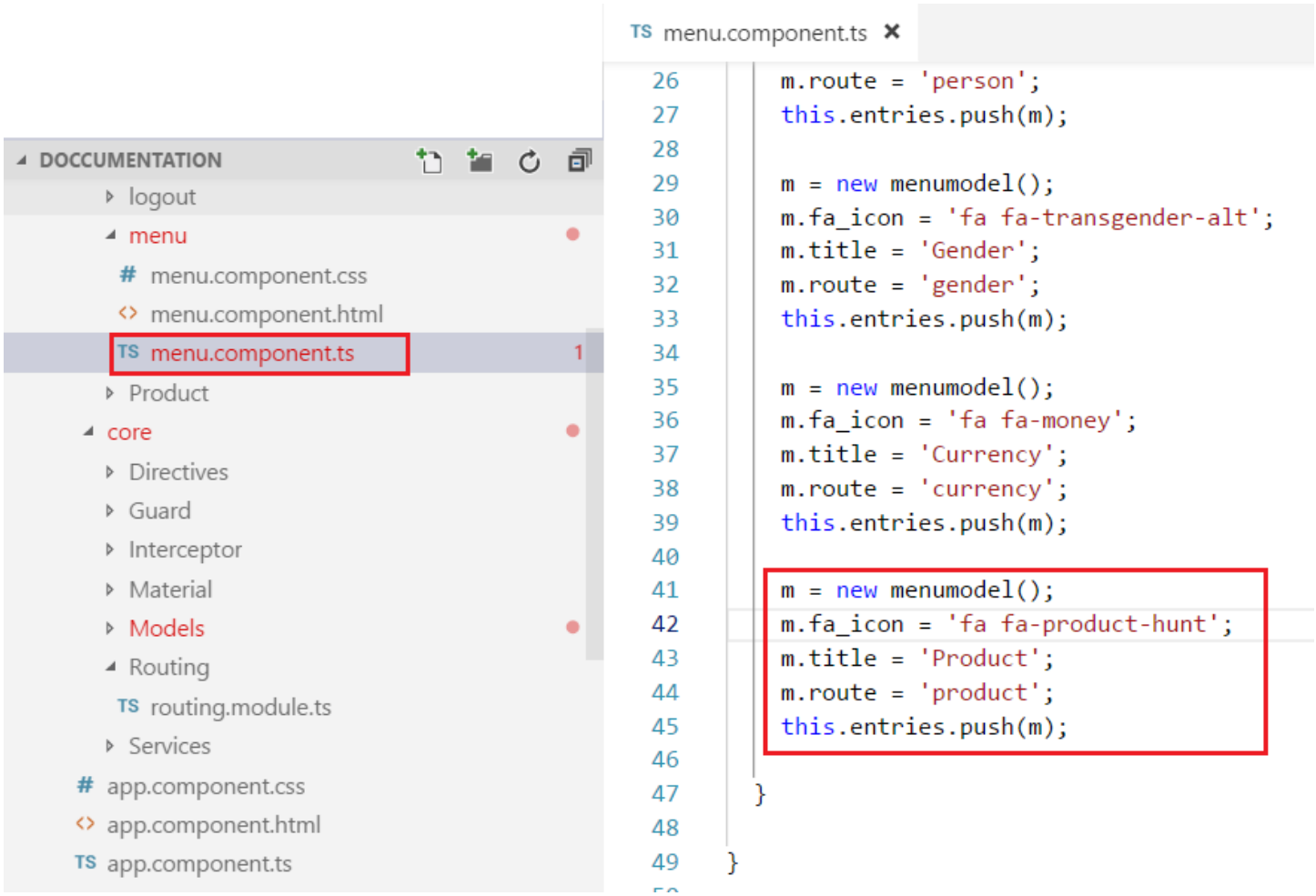
2. Update the app.module.ts file : Add the Product Component to the declarations array.



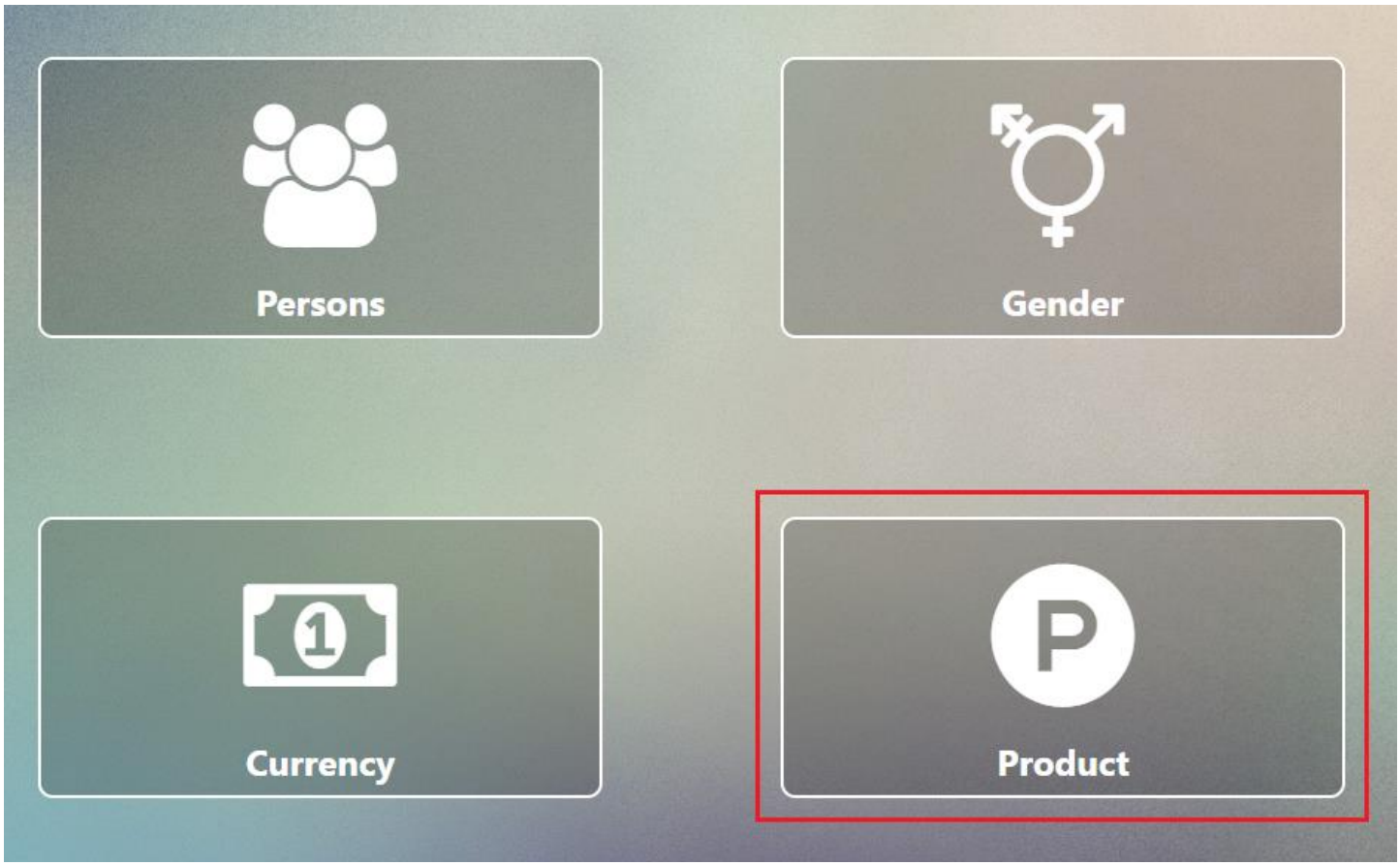
3. Add routing entry for the newly created Product Component in routing.module.ts file



4. Add to the menu component a new entry, so we can access the Product component from the menu screen

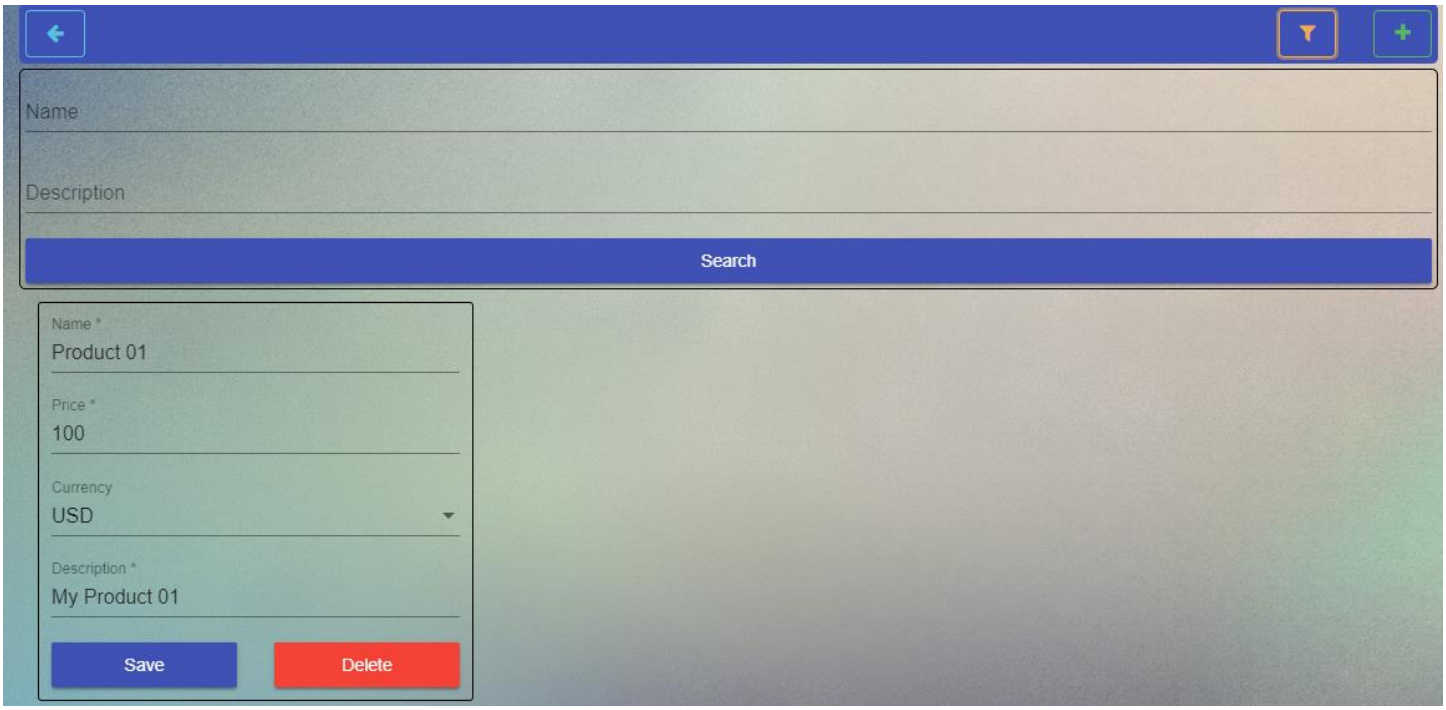


*. Now if you login (again , any user name / password) you will see a new menu entry named "Product"



*. By clicking on the Product menu, you will be able (as for the Currency) to create products, update products and delete them.

*. The main difference between the Currency & Product angular Components is that the Product component allows you to fetch for a certain product by clicking on the "Filter" icon (exists beside the "+" icon)



*. Let us add new table to your database named TBL_PURCHASE with the following structure.

TBL_PURCHASE

Column Name	Data Type	Allow Nulls
PURCHASE_ID	int	☐
DATE	date	☐
PRODUCT_ID	int	☐
QUANTITY	int	☐
DESCRIPTION	nvarchar(50)	☐

*. As we did for TBL_CURRENCY, we have to run AppGenerator console application with option "001" and the following steps should be applied.

1. Apply T-Sql Script
2. Override the files that exist under the DALC folder in the DALC Project
3. Override BLC Files under the BLC Project

*. One new extra step should be done here, and it is adding a custom property to the Product Class in the BLC Project

So open BLC_CustomBehavior.cs file and in the Business Entities region, add the following.

```
#region Business Entities
Setup
public partial class Product
{
    0 references | 0 changes | 0 authors, 0 changes
    public List<Purchase> My_Purchases { get; set; }
}
```

*. Let us expose the following BLC Methods as Web API Methods, by adding them to program.cs file of AppGenerator Console Application

```
oCodeBooster.WCFMethodsSelection.Add("Edit_Currency");
oCodeBooster.WCFMethodsSelection.Add("Delete_Currency");
oCodeBooster.WCFMethodsSelection.Add("Get_Currency_By_OWNER_ID");

oCodeBooster.WCFMethodsSelection.Add("Edit_Product");
oCodeBooster.WCFMethodsSelection.Add("Delete_Product");
oCodeBooster.WCFMethodsSelection.Add("Get_Product_By_Where");

oCodeBooster.WCFMethodsSelection.Add("Edit_Purchase");
oCodeBooster.WCFMethodsSelection.Add("Delete_Purchase");
```

*. After running the option "002" (Web API Generation), and the corresponding patch is downloaded.

*. As we did previously, you have to override DataController.cs file under Web API Project & proxy.service.ts file in the Angular Web Application.

*. Now, regarding the user interface, it would be great for the end user to be able to see for a certain product.

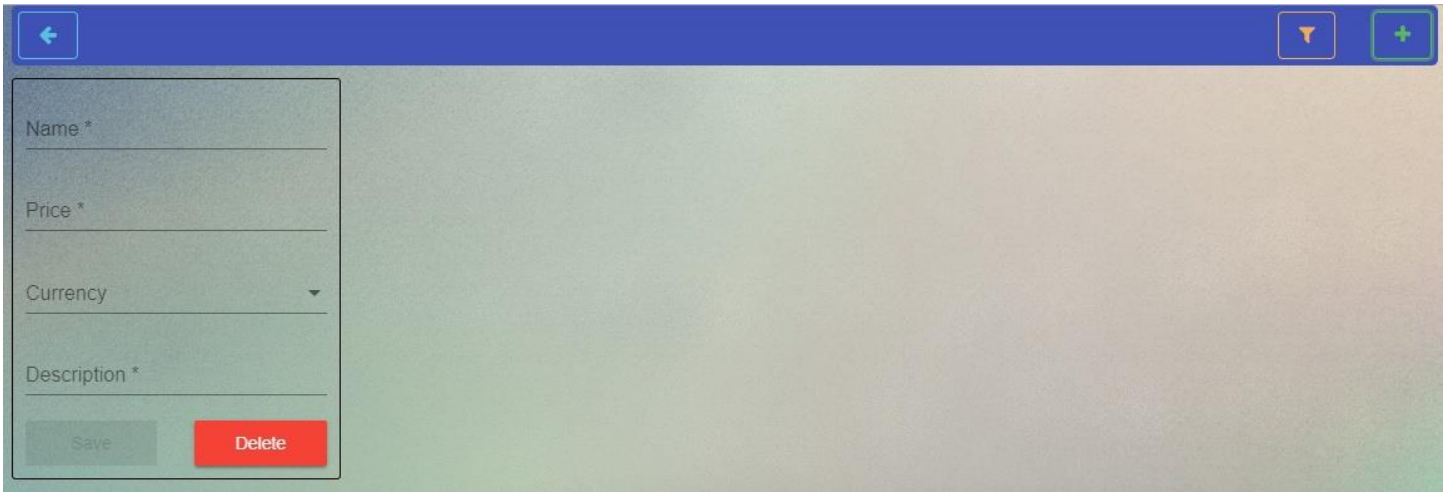
the corresponding purchases done till now.

All what we have to do is to inform code booster that, when generating the Product angular component, that the Purchases should be shown in the same screen for each product as the following

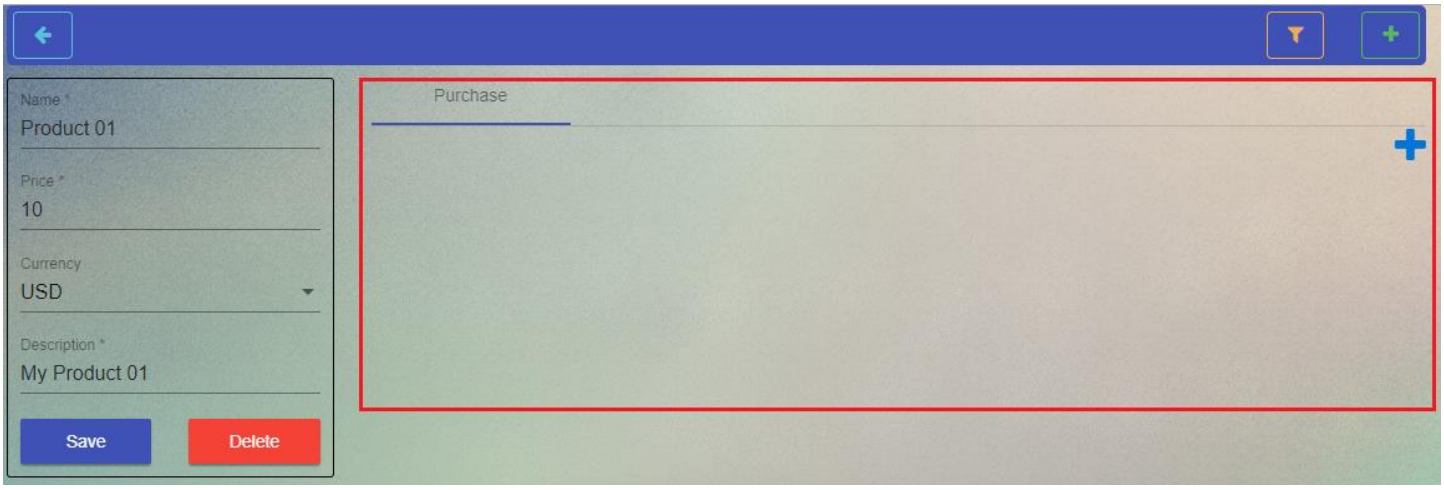
```
#region Product
#region Search Screen
oUIFields_Criteria = new UIFields();
oUIFields_Criteria.MainTableName = "[TBL_PRODUCT]";
oUIFields_Criteria.Based_On_Type = "BLC.Params_Get_Product_By_Where";

oUIFields_Result = new UIFields();
oUIFields_Result.MainTableName = "[TBL_PRODUCT]";
oUIFields_Result.Based_On_Type = "BLC.Product";
oUIFields_Result.GetMethodName = "Get_Product_By_Where";
oUIFields_Result.GridFields = new List<GridField>();

oUIFields_Result.Has_Related_Data = true;
oUIFields_Result.Related_Tables = new List<string>();
oUIFields_Result.Related_Tables.Add("[TBL_PURCHASE]");

oSearch_AdvancedProp = new Search_AdvancedProp();
oSearch_AdvancedProp.ContainerMargins = "0,5,0,5";
oCodeBooster.Entity_FriendlyName = "Product";
oCodeBoosterClient.Generate_ListUI(Enum_SearchMethod.With_Criteria_Section, oUIFields_Criteria, oUIFields_Result, oSearch_AdvancedProp);
#endregion
#endregion
```

*. Run AppGenerator and selection option "003" and once the patch is downloaded containing the Product Angular Component just override it (the Product Angular component) under the src\app\components folder.

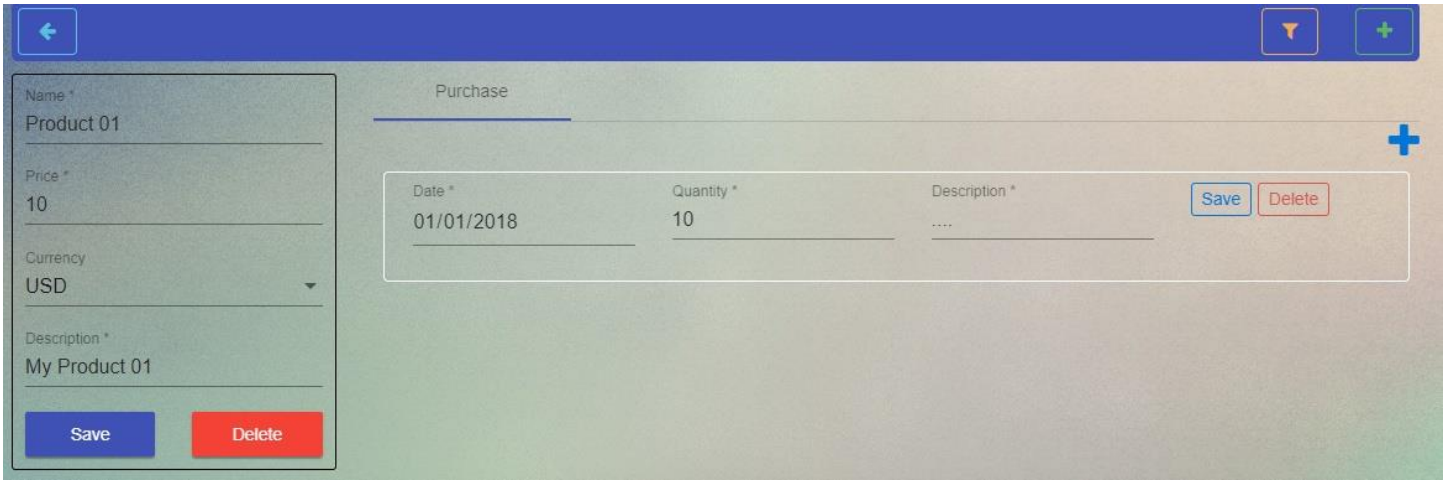
So now, if you visit the product screen and click on "+" icon you will see the following.



*. Fill required fields (name, price, currency & description) , and you will see a tab section will open on the right side allowing you to add purchases related to this product.



*. By Click on the blue "+" icon, you can add now (then update or delete) purchases related to this product as the following.



*. Uploading and Managing files related to a certain entity.

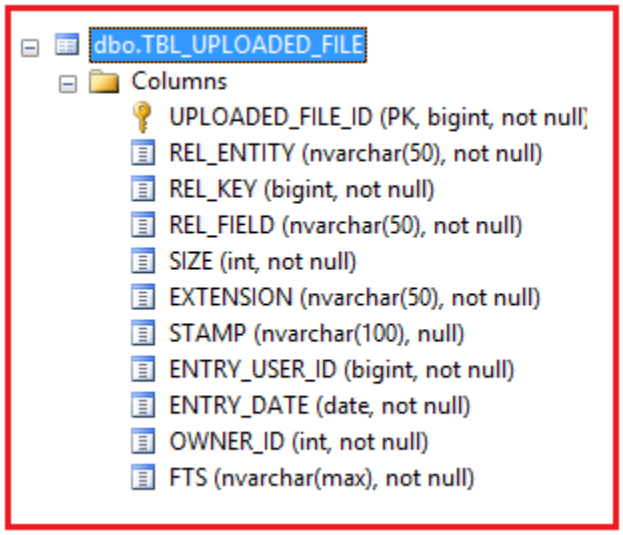
In Most of mobile & applications you have to manage images ,and by "manage" we mean upload image(s), delete image (s) & retrieve image(s) related to a certain entity (ex: Product table)

*. Do the following steps in order to integrate the image uploading feature to your application:

*. In AppGenerator and especially in program.cs file, assure you have the following added.

```
#region Uploaded_files
oCodeBooster.Is_Uploaded_File_Feature = true;
#endregion
```

*. Run Option "001" and apply the corresponding patch (T-Sql, DALC Layer, BLC layer) and you will notice the following table in your database



*. In AppGenerator and especially in program.cs file, assure you have the following added.

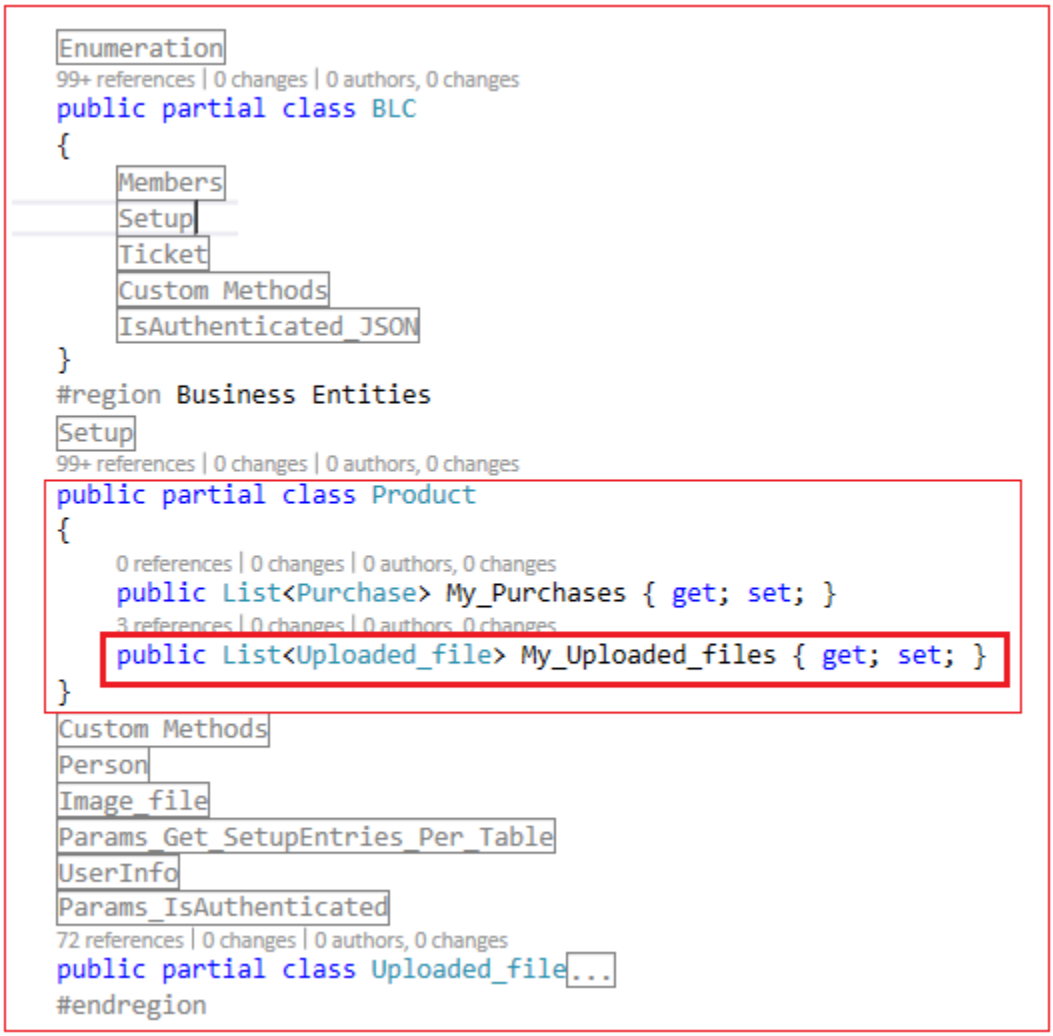
```
#region Uploaded_files
oCodeBooster.Is_Uploaded_File_Feature = true;
oCodeBooster.Uploaded_Files_BackEnd_Events = new List<Uploaded_File_BackEnd_Event>();

oCodeBooster.Uploaded_Files_BackEnd_Events.Add
            (
                new Uploaded_File_BackEnd_Event()
                {
                    TBL_NAME = "[TBL_PRODUCT]",
                    UI_METHOD_NAME = "Get_Product_By_Where", Mode = 0
                }
            );
#endregion
```

*. Run Option "001" and apply the corresponding patch (T-Sql, DALC Layer, BLC layer)

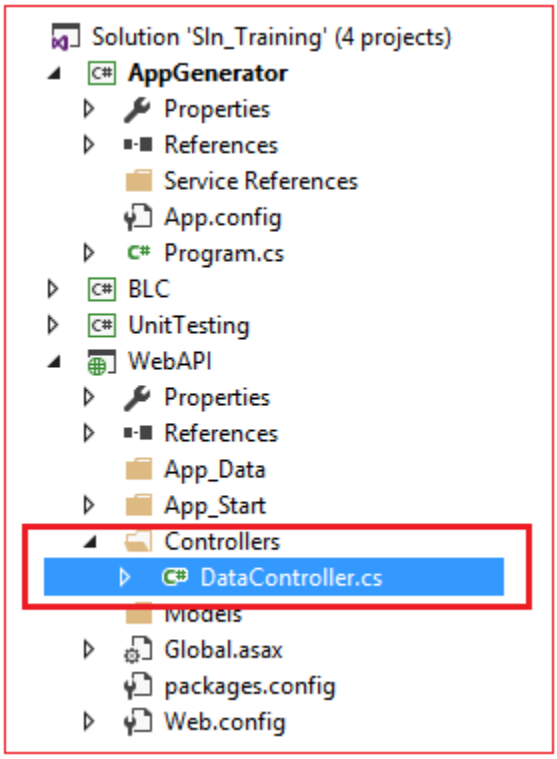
*. In BLC_Initial.cs file and especially in SubscribeToEvents method, add the following line of code to automatically register all required events for the newly created table TBL_UPLOADED_FILE

```
#region Subscribe To Events

public void SubscribeToEvents()
{
    Declaration And Initialization Section.
    #region Body Section.

    Register_Uploaded_Events_Handlers();

    #endregion
}
#endregion
```

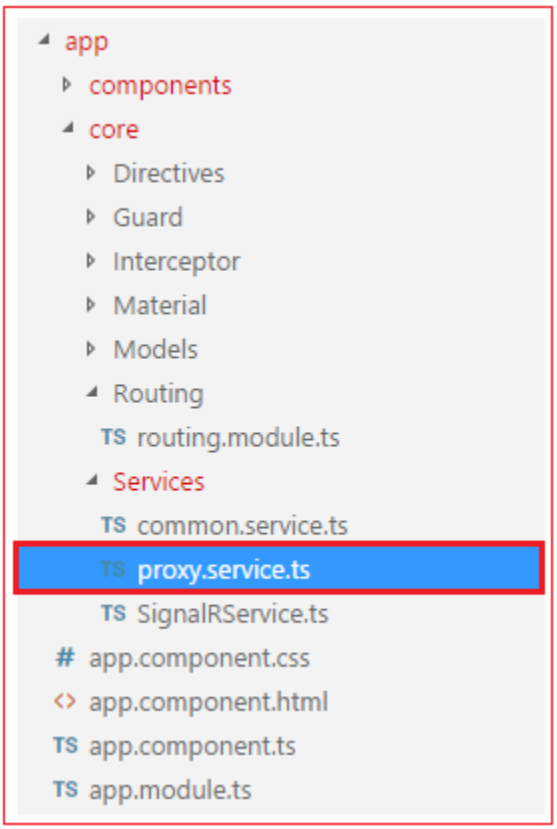
*. In BLC_CustomBehavior.cs file and especially in BLC Entities region, add the following property to the Product class



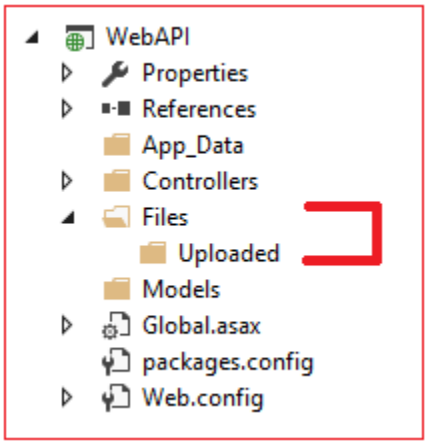
*. Run Option "002" to generate the API related files and override the DataController.cs file under "Controllers"



*. Override the proxy.service.ts file (generated with the patch) in the angular application in its place as the following



*. In the Web API Project, you have to create the following hierarchy of folders.



*. Add the following entries to the app.config file of the API Project.

```
<?xml version="1.0"?>
<configuration>
  <appSettings>
    <add key="CONN_STR" value="Data Source=roni-mobile\INSTANCE_2K8R2_1;Database=MMS;User ID=sa;Password=sapassword"/>
    <add key="BLC_MESSAGES" value="E:\DEV\MMS\App\BLC\Messages\Messages.xml"/>
    <add key="ENABLE_TICKET" value="0"/>
    <add key="UPLOAD_DOCUMENT_LOCATION" value="E:\DEV\MMSCore\App\WebAPI\Files\Uploaded\"/>
    <add key="WEB_PATH" value="http://localhost:5000/api/Data"/>
    <add key="PRINTING_FOLDER_TEMPLATE" value="E:\DEV\MMSCore\App\WebAPI\Printing\Template" />
    <add key="PRINTING_FOLDER_GENERATED" value="E:\DEV\MMSCore\App\WebAPI\Printing\Generated" />
  </appSettings>
</configuration>
```

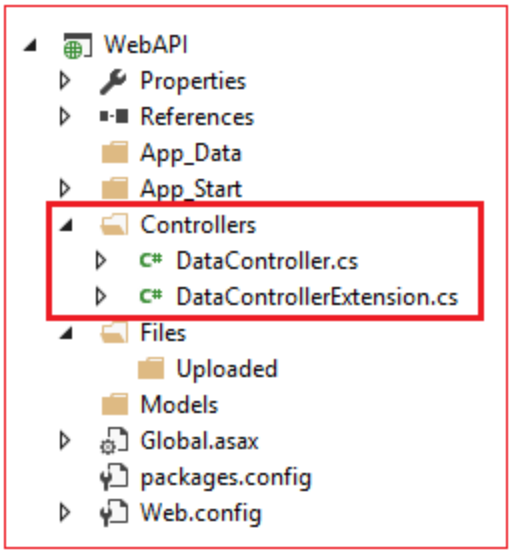
In case you have Linux Based Deployment you can the following sample of Configuration entries

```
<?xml version="1.0"?>
<configuration>
  <appSettings>
    <add key="CONN_STR" value="Data Source=192.168.56.101:Database=MMS;User ID=sa;Password=yourStrong(!)Password"/>
    <add key="BLC_MESSAGES" value="./Messages.xml"/>
    <add key="ENABLE_TICKET" value="0"/>
    <add key="UPLOAD_DOCUMENT_LOCATION" value="./Files/Uploaded/"/>
    <add key="WEB_PATH" value="http://192.168.56.101:3000/api/Data"/>
    <add key="PRINTING_FOLDER_TEMPLATE" value="./Printing/Template" />
    <add key="PRINTING_FOLDER_GENERATED" value="./Printing/Generated" />
  </appSettings>
</configuration>
```

N.B: Sure you have to change the corresponding values to match your environment

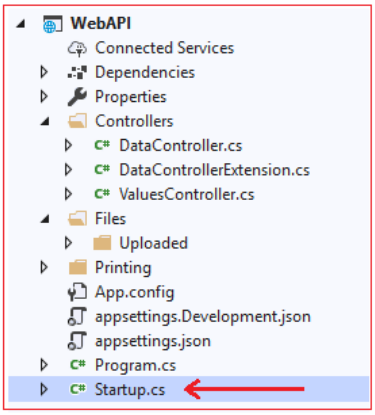
*. Assure that you are referencing the CoreTools.dll library from the API Project.

*. In the "CB\Core" folder you will find a file named **DataControllerExtension.cs ,** just copy & past it under the **Controllers** folder as the following



*. In the Startup.cs file assure you have added the following section to be able to access the uploaded file (static files)

```
public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    app.UseStaticFiles(new StaticFileOptions
    {
        FileProvider = new PhysicalFileProvider(
        Path.Combine(Directory.GetCurrentDirectory(), "./Files/Uploaded/")),
        RequestPath = "/api/Data/Files/Uploaded"
    });

    app.UseStaticFiles(new StaticFileOptions
    {
        FileProvider = new PhysicalFileProvider(
       Path.Combine(Directory.GetCurrentDirectory(), "./Printing/Generated/")),
        RequestPath = "/Printing/Generated"
    });

    app.UseCors(options => options.AllowAnyOrigin().AllowAnyHeader().AllowAnyMethod());
    app.UseMvc();
}
```

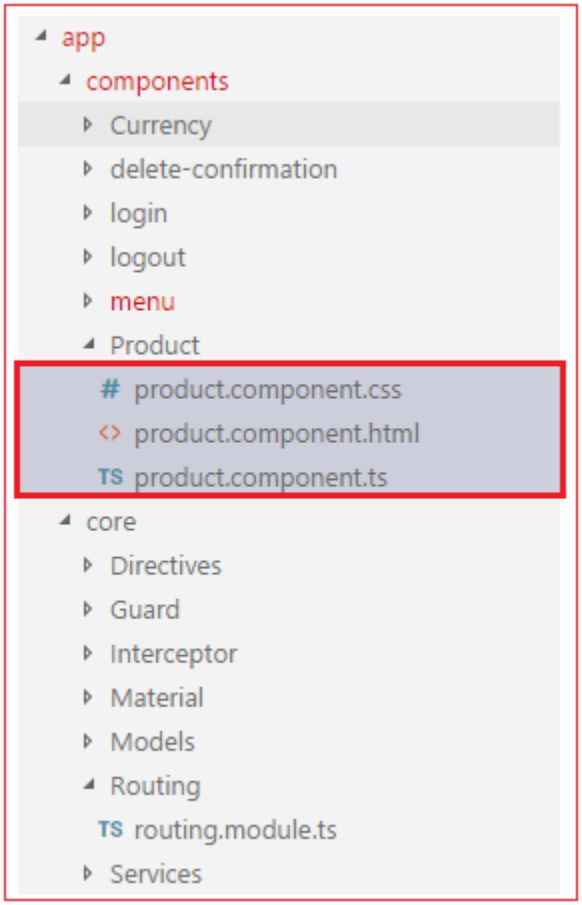


*. Build the API Project to assure no error existence.

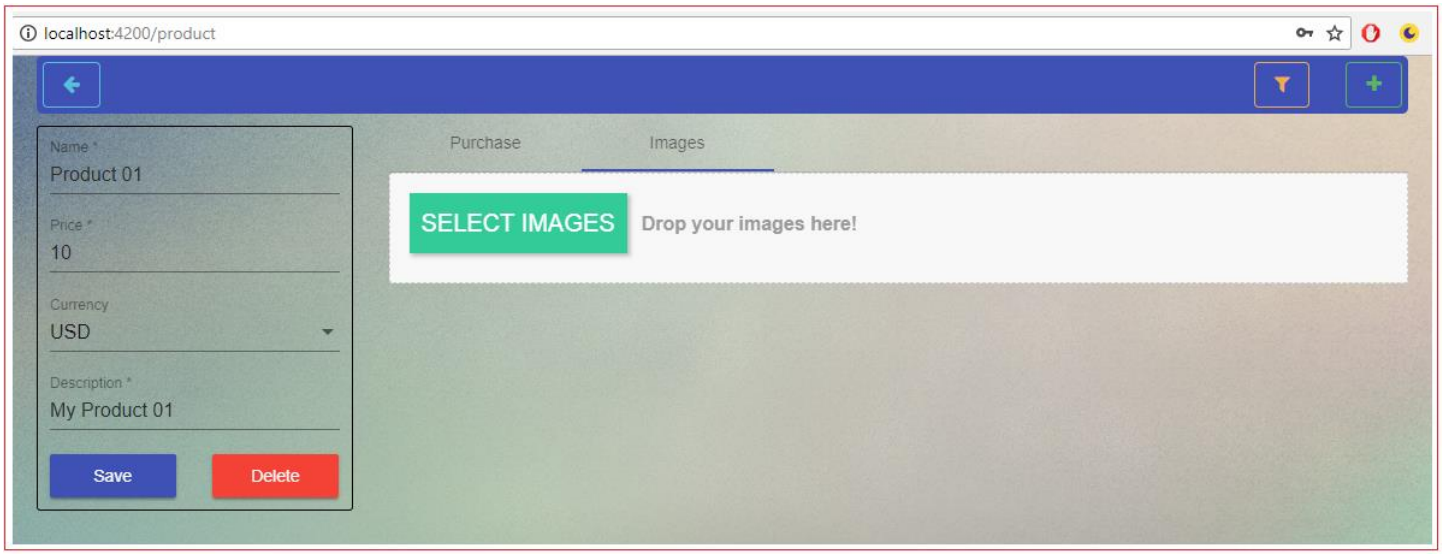
*. Now, in order to inform Code Booster engine that you want to upload images for each product , add the following flag

```
#region Product
#region Search Screen
oUIFields_Criteria = new UIFields();
oUIFields_Criteria.MainTableName = "[TBL_PRODUCT]";
oUIFields_Criteria.Based_On_Type = "BLC.Params_Get_Product_By_Where";

oUIFields_Result = new UIFields();
oUIFields_Result.MainTableName = "[TBL_PRODUCT]";
oUIFields_Result.Based_On_Type = "BLC.Product";
oUIFields_Result.GetMethodName = "Get_Product_By_Where";
oUIFields_Result.GridFields = new List<GridField>();

oUIFields_Result.Has_Related_Data = true;
oUIFields_Result.Has_Related_Files = true;

oUIFields_Result.Related_Tables = new List<string>();
oUIFields_Result.Related_Tables.Add("[TBL_PURCHASE]");



oSearch_AdvancedProp = new Search_AdvancedProp();
oSearch_AdvancedProp.ContainerMargins = "0,5,0,5";
oCodeBooster.Entity_FriendlyName = "Product";
oCodeBoosterClient.Generate_ListUI(Enum_SearchMethod.With_Criteria_Section, oUIFields_Criteria, oUIFields_Result, oSearch_AdvancedProp);
#endregion
```

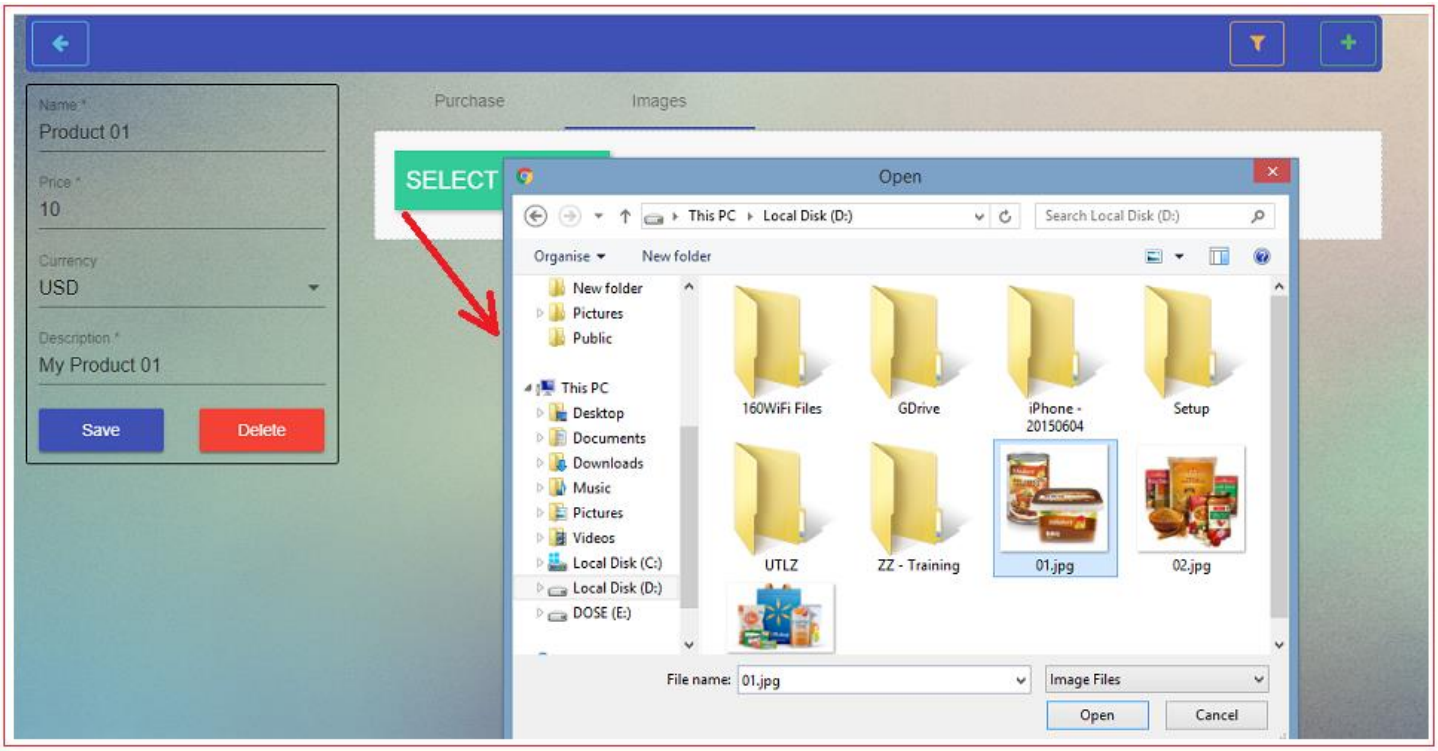
*. Generate the UI by issuing option "003" and override the corresponding component files (.ts, .html & .css) under the product folder as the following.



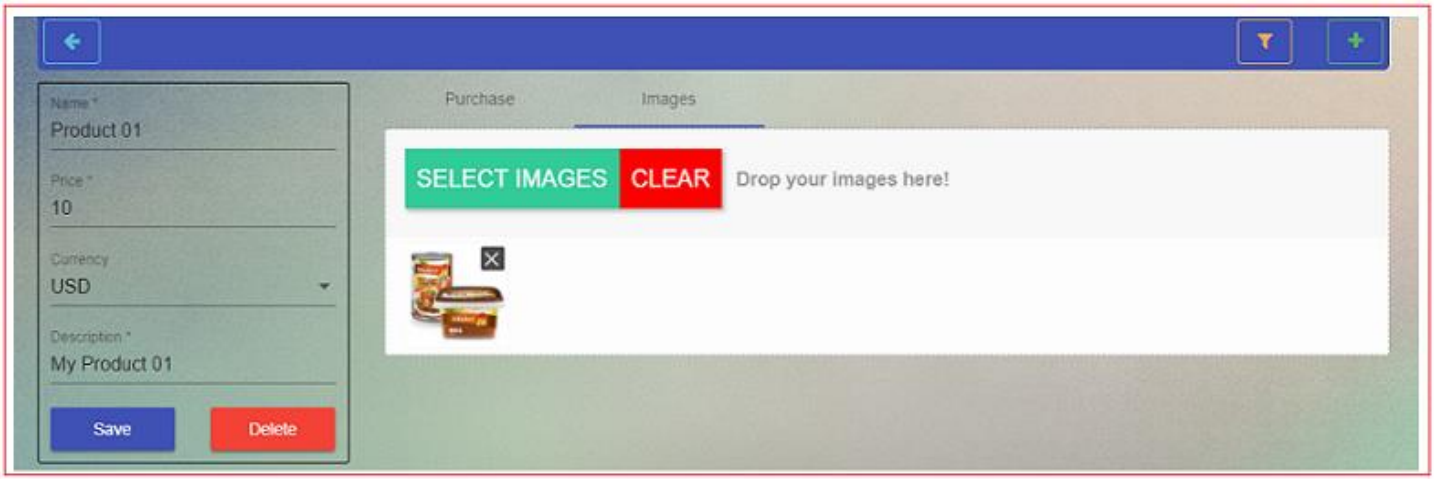
*. If you check the "Product" screen you will notice that a new tab has been created named "images" as the following



*. If you click on "SELECT IMAGES" , you will be able to upload images related to the selected product as the following



*. Once you select an image you get the following screen (sure you can select as many as you want images to be linked to a certain product)



*. If you check the "Files\Uploaded" folder you just created in the API application, you will notice that the image has been uploaded to the server (in your case you Laptop itself)