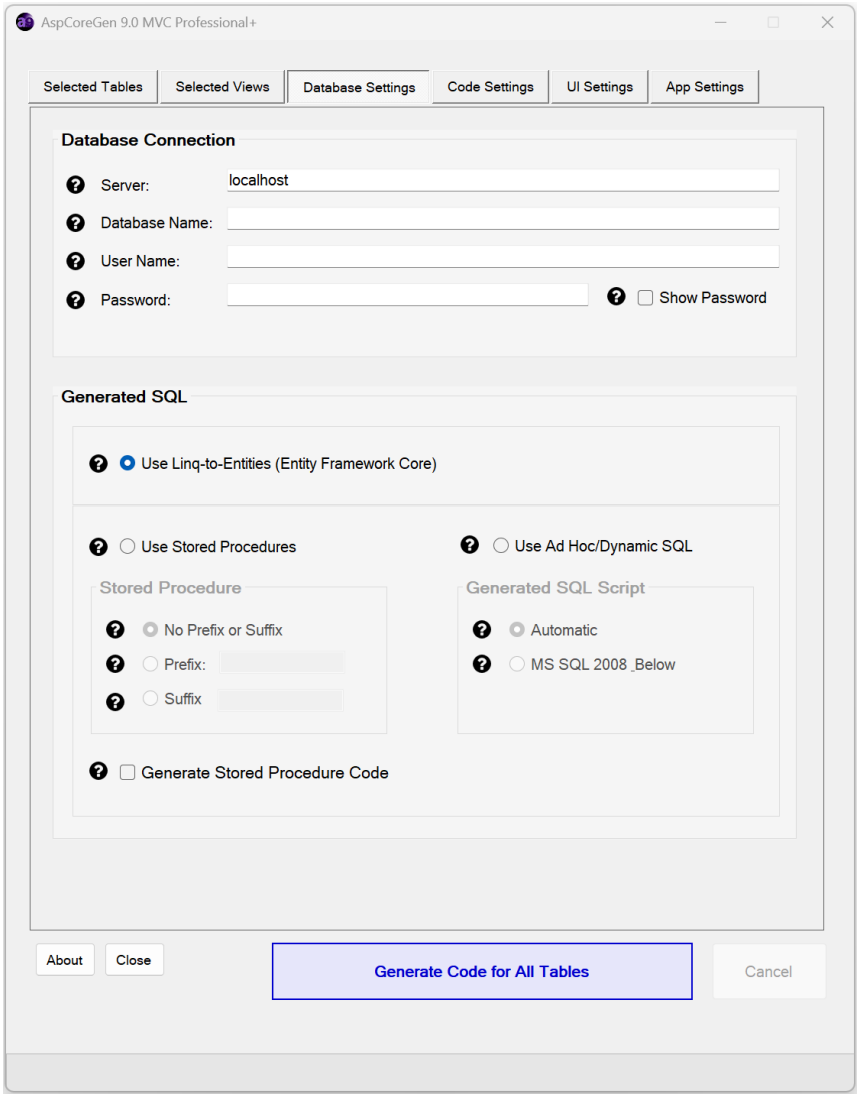
Database Settings Tab

1 CONTENTS

Database Settings Tab

The Database Settings Tab is where we set database-specific information such as the database connection properties and the type of SQL script to generate.



A. Database Connection

This is the MS SQL Server Database connection settings to the database. All of these items are required and cannot be blank.

1. **Server:** Server Name. If the MS SQL Server Database is local, you can just enter "*localhost*".
2. **Database Name:** The database you're trying to access.
3. **User Name:** An admin user name login that has access to the database.
4. **Password:** The password associated with the user name.
5. **Show Password:** Checking the *Show Password* in Figure 9 will remember the password you enter here so you don't have to enter it again the next time you open AspCoreGen 9.0 MVC.

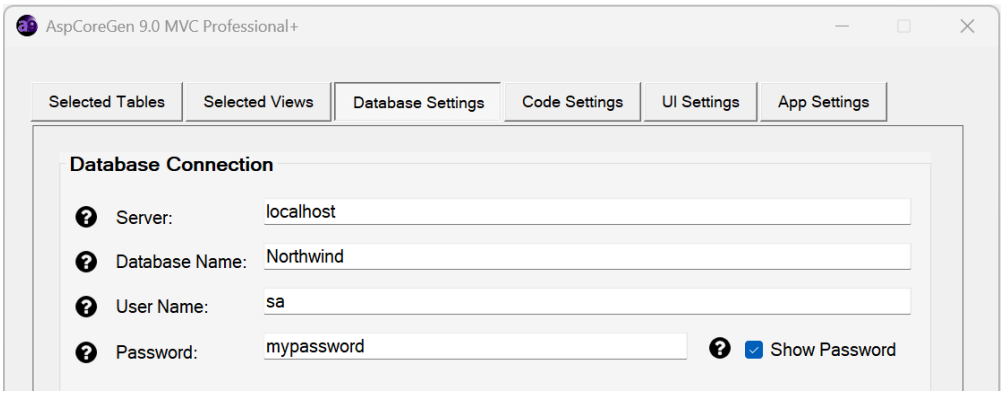
B. GENERATED SQL SCRIPT

The type of SQL script that will be generated.

1. **USE LINQ-TO-ENTITIES (ENTITY FRAMEWORK CORE):** Entity Framework.
2. **USE STORED PROCEDURES:** SQL Scripts generated straight in your MS SQL database.
    a. **No Prefix or Suffix*:*** Names the generated script with no prefix or suffix. E.g. *Products_SelectAll*.
    b. **Prefix:** Names the generated script with a prefix. For example, when you enter *"sp_"* on the Prefix text box, the generated stored procs will start with an *"sp_"*. E.g. ***sp_****Products_SelectAll*.
    c. **Suffix:** Names the generated script with a suffix. For example, when you enter *"_sp"* on the Suffix text box, the generated stored procs will end with an *"_sp"*. E.g. *Products_SelectAll****_sp***.
3. **USE AD-HOC/DYNAMIC SQL:** SQL Scripts generated in your C# code.
    a. **Automatic:** Automatically determines your MS SQL Server version before generating scripts.
    b. **MS SQL 2008 Below:** Generates scripts for MS SQL Server versions 2008 and below. There's a slight difference with the generated script for older versions because some keywords are not available during this time.

1 DATABASE CONNECTION

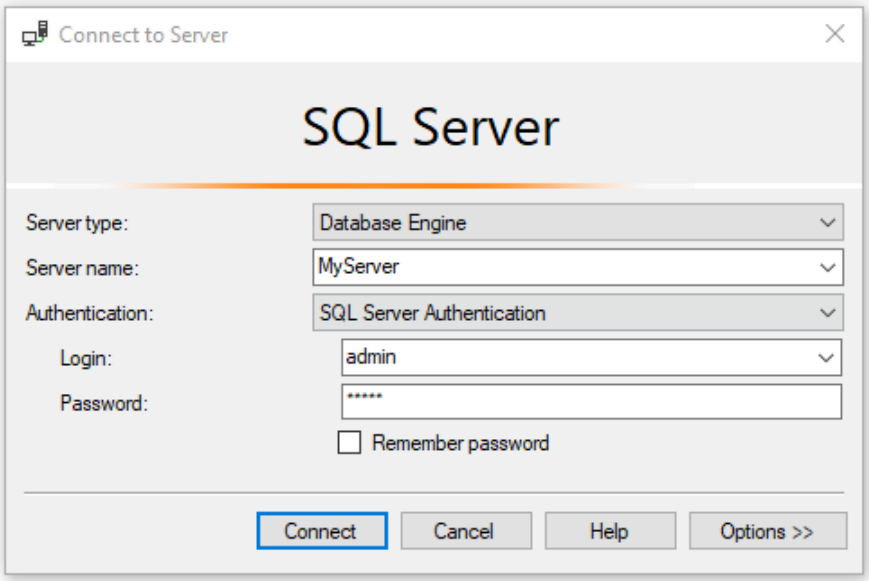
We use these fields/properties to connect to the MS SQL Server database, both to generate code and to perform CRUD (create, retrieve, update, delete) operations when you run the generated code.



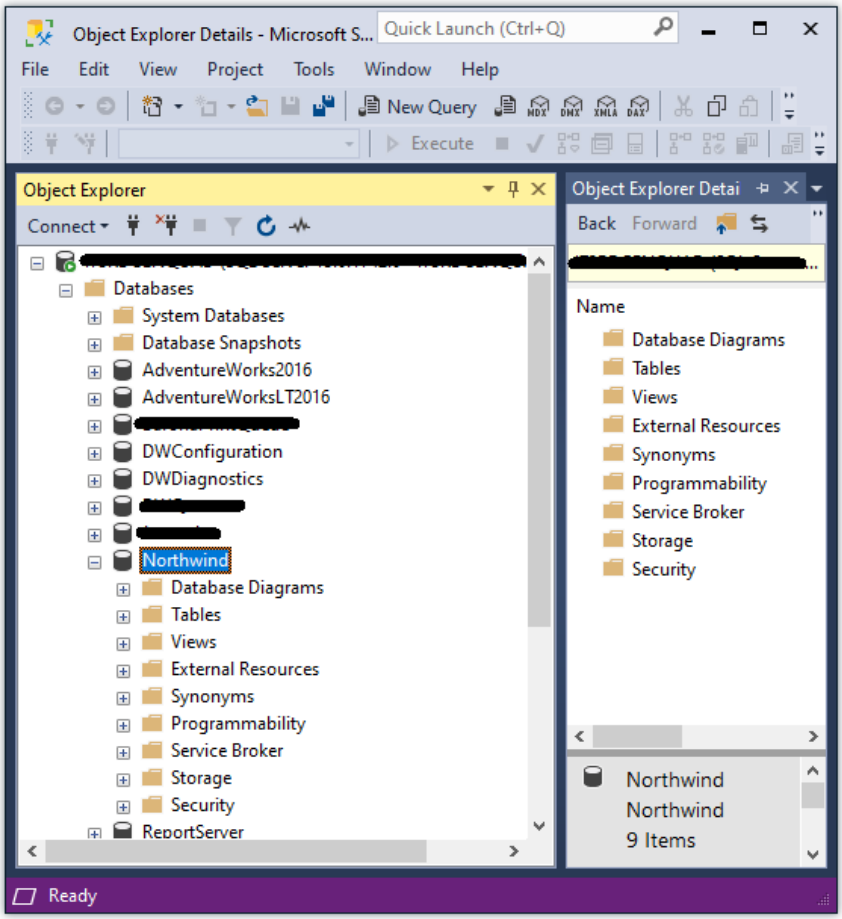
1.1 USING DATABASE CONNECTION TO CONNECT TO MS SQL SERVER DATABASE TO GENERATE CODE

When you open your MS SQL Server Database it displays the *Connect to Server* dialog. Please see the direct relationship between the Database Connection fields you enter in AspCoreGen 9.0 MVC and your MS SQL Server Database below.

AspCoreGen 9.0 MVC	MS SQL Server Database
Server: *localhost* (when local) or *MyServer*	**Server Name:** *localhost* (when local) or *MyServer*
Database Name: *Northwind*	**Databases:** *Northwind*
User Name: *admin*	**Login:** *admin*
Password: *admin*	**Password:** *admin*
Show Password: *true*	n/a



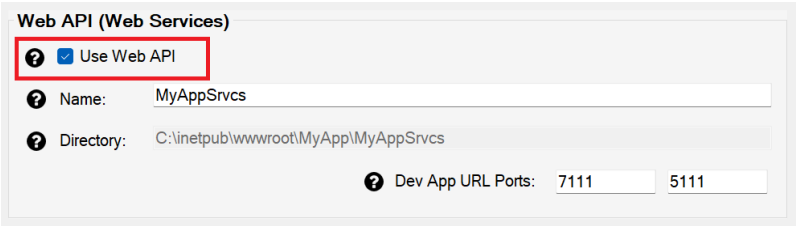
Once you're connected to MS SQL Server Database, you will see a list of *Databases* in the *Object Explorer*. The *Northwind* database is what we have in the example above (Database Name), the database we want to generate code from.



1.2 Using Database Connection to Connect to MS SQL Server Database to Perform CRUD

Once the code is generated, a part of the code is used to connect to the MS SQL Server Database you've chosen (*Northwind*) to perform CRUD operations on. In the generated code you will find the *Connection String* in the *appsettings.json* file.

When you choose *Use Web API* under the *Code Settings* tab, the *Connection String* will be in the *appsettings.json* file in the Web API project, otherwise it will be in the *appsettings.json* file in the Web Project.



Use Web API	appsettings.json Location
Checked	Web API project
Not Checked	Web Application project

appsettings.json in Web Project

appsettings.json in Web API Project

Note: The *appsettings.json* file will not be overwritten by AspCoreGen 9.0 MVC the next time you re-generate code for the same project. So it is safe to add your own settings in this file. You can change the *Connection String* values once you push your web application to your production environment. Make sure to point it to your Production Database instead.

1.2.1 Using The Connection String

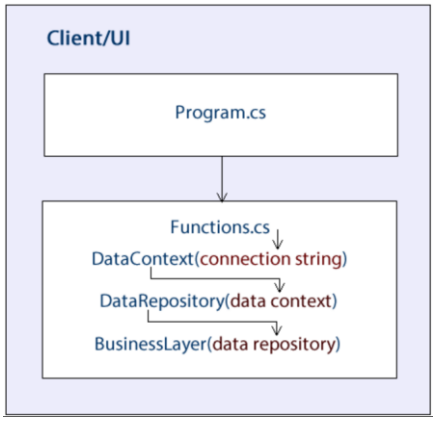
Note: In an n-tier or layered design, the *Data Repository Class* (data layer/tier) is ultimately **the only tier that calls to and from the database**.

We inject (dependency injection) the *Connection String* from the *Program.cs* (client) by calling a helper method in the *Functions.cs*.

AspCoreGen 9.0 MVC can generate SQL Script in 3 ways: Use Linq-To-Entities (Entity Framework), Ad-Hoc SQL, or Stored Procedures. Please see *#2 Generated SQL* below. Based on what you choose under the *Generated SQL*, the *Connection String* can either be injected to the *Data Repository* or to the *Data Context*.

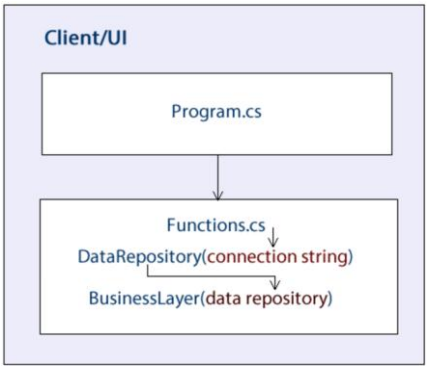
1.2.2 Using The Database Context Class (Entity Framework)

When you generate code with *Linq-To-Entities (Entity Framework)*, the *Connection String* is injected to the *Database Context Class* from the *Program.cs* (client). The *Database Context Class* is then injected to the *Data Repository* from the *Business Layer*. The *Data Repository* can then use the *Database Context Class* to connect to the database.



1.2.3 Using The Connection String (Ad-Hoc or Stored Procedures)

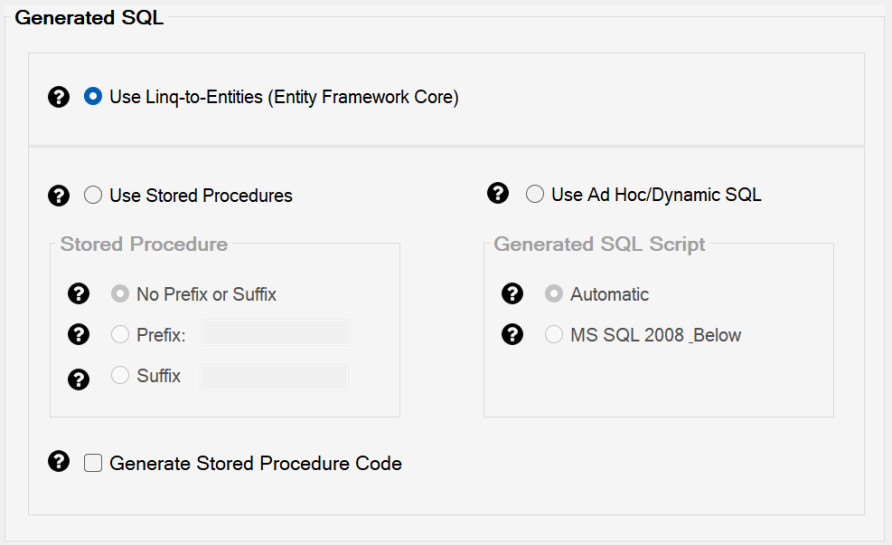
When you generate code with *Ad-Hoc SQL* or *Stored Procedures*, the *Connection String* is injected to the *Data Repository* from the *Program.cs* (client). The *Data Repository* is then injected to the *Business Layer*. The *Data Repository* can then use the *Connection String* to connect to the database.



2 GENERATED SQL

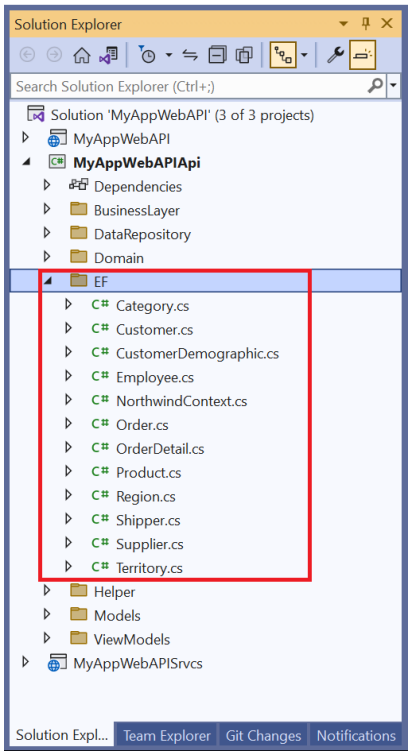
We use these fields/properties to determine the type of SQL Script that will be generated.

2.1 USE LINQ-TO-ENTITIES (ENTITY FRAMEWORK CORE)



When you select *Use Linq-to-Entities (Entity Framework Core)* (default), the generated *Data Repository* code will be in an *Entity Framework Core Linq-to-Entities* format. The code will be generated in the: *Business Layer and Data Repository API Project* (Class Library Project).

- EF Directory



Generated Entity Classes

Each table selected for code generation will also have their respective class generated in the EF folder. E.g. *Categories* table will generate *Category.cs* (singular) class, etc.

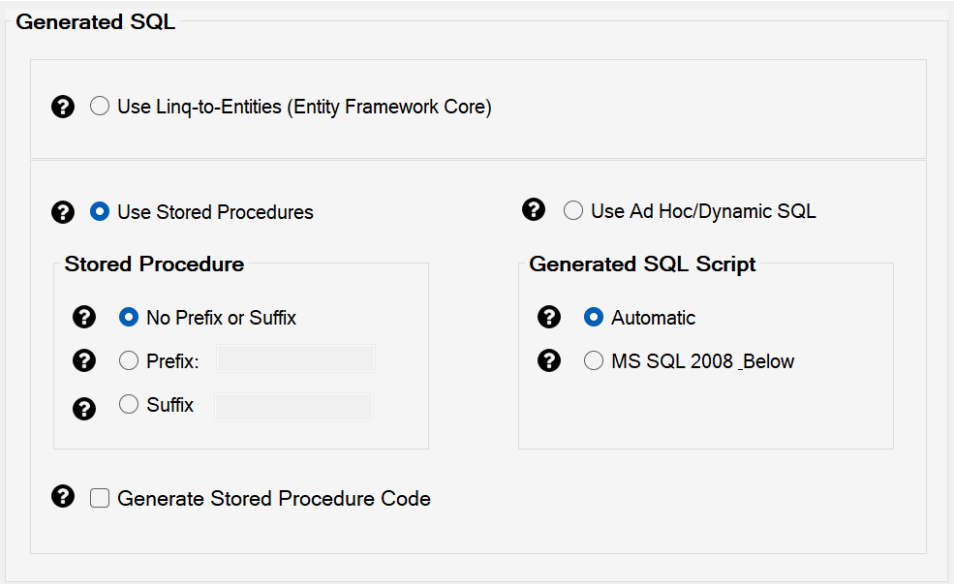
Category.cs

MyAppWebAPIApi | MyAppWebAPIApi.EF.Category

```
1   using System;
2   using System.Collections.Generic;
3
4   namespace MyAppWebAPIApi.EF
5   {
        31 references
6       internal class Category
7       {
            2 references
8           internal Category()
9           {
10              Products = new HashSet<Product>();
11          }
12
            22 references
13          internal int CategoryID { get; set; }
            20 references
14          internal string CategoryName { get; set; }
            9 references
15          internal string Description { get; set; }
16
            2 references
17          internal virtual ICollection<Product> Products { get; set; }
18      }
19  }
20
```

Generated Categoy.cs Entity Class

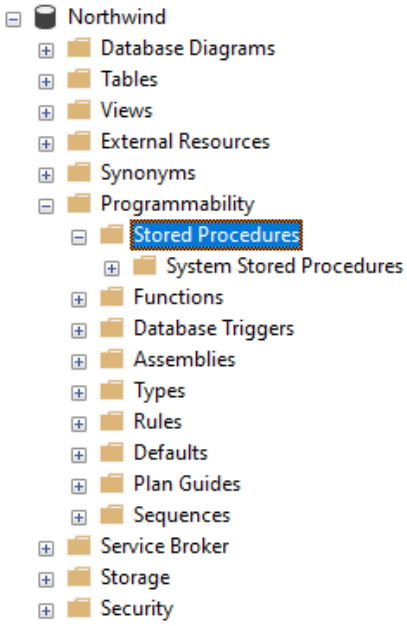
2.2 USE STORED PROCEDURES



When you select *Use Stored Procedures*, the generated *SQL Script* code will be in a *Stored Procedure* form. The code will be generated in the MS SQL Server Database, in our example, SQL Scripts will be generated in the *Northwind* database. You will find the generated Stored Procedures:

Under the *Northwind* database

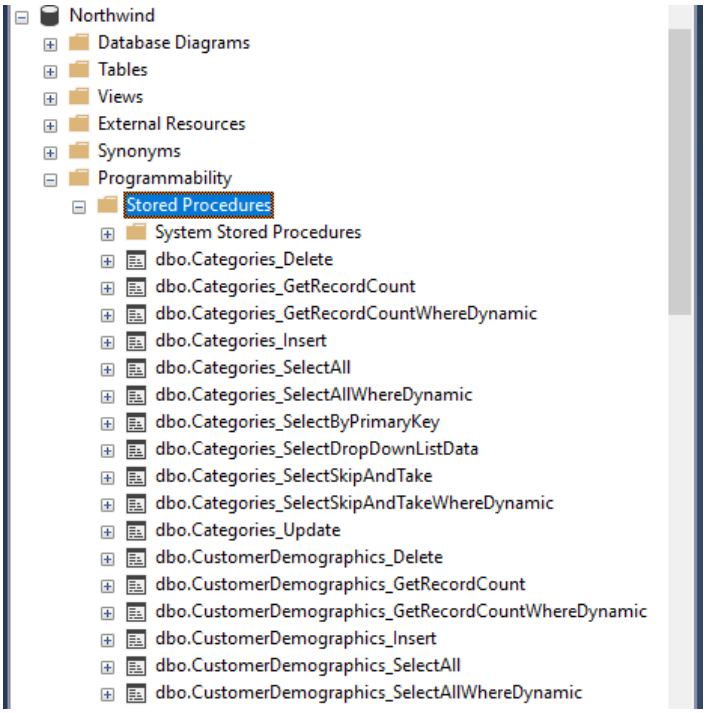
- *Programability* folder
    - *Stored Procedures* folder



Generated Stored Procedures will be under Stored Procedures folder

2.2.1 Use Stored Procedures – No Prefix or Suffix

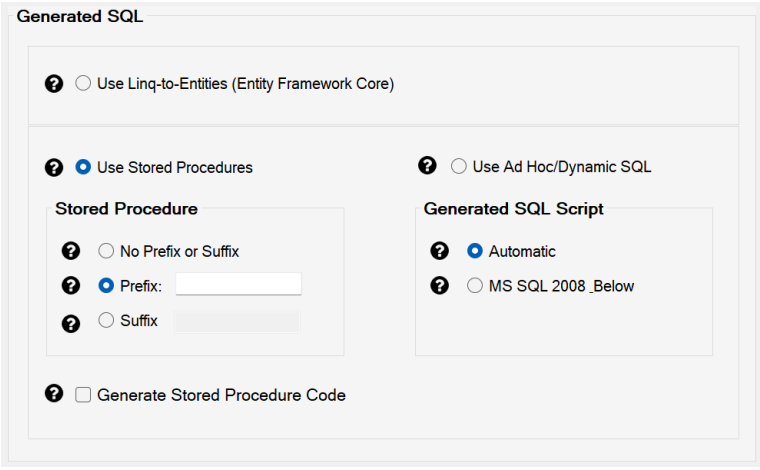
When you select *No Prefix or Suffix* (default), the generated *Stored Procedure Names* will have no prefixes or suffixes. E.g. *Categories_Delete, Categories_GetRecordCount*



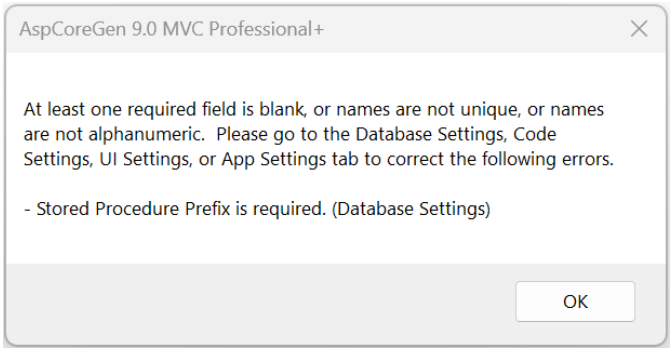
Stored Procedure Names with No Prefix or Suffix

2.2.2 Use Stored Procedures – Prefix

When you select *Prefix*, the generated *Stored Procedure Names* will have a prefix. The *Prefix* box is **required** when *Prefix* under *Stored Procedure* is selected.

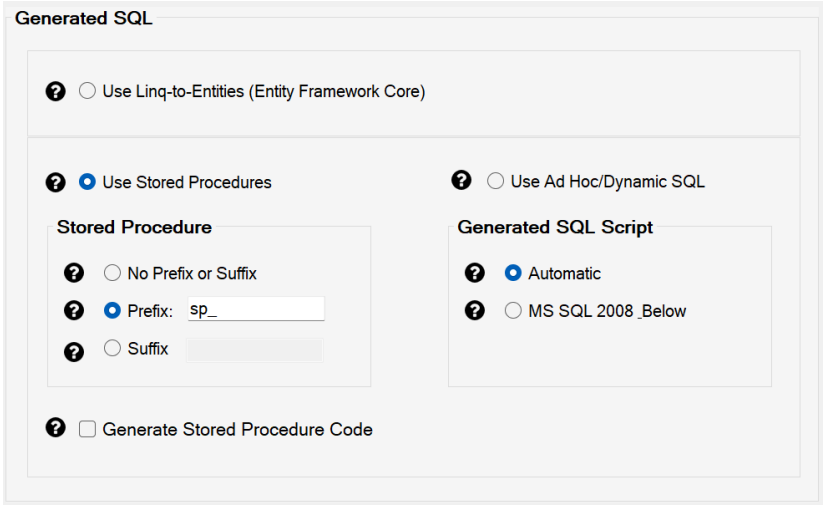


Prefix with Blank Box

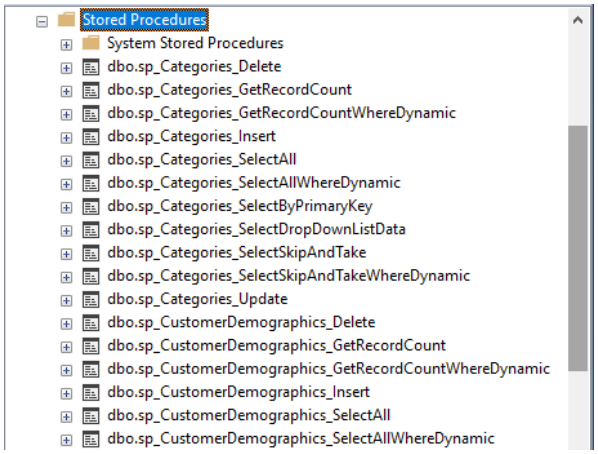


Prefix is required

The *Prefix* you enter will be used as the *Prefix* for the generated *Stored Procedure Names*. E.g. ***sp_****Categories_Delete,* ***sp_****Categories_GetRecordCount*.



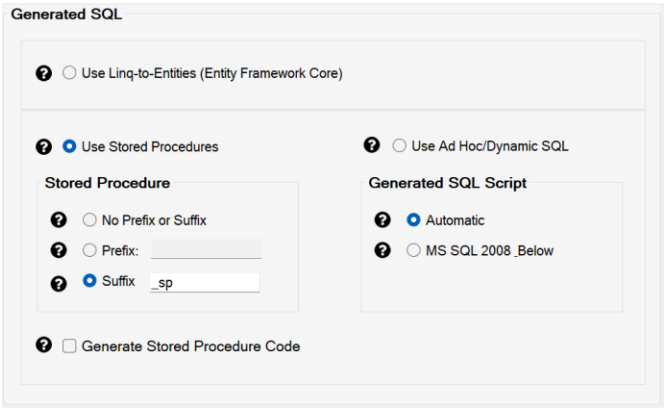
Prefix (filled)



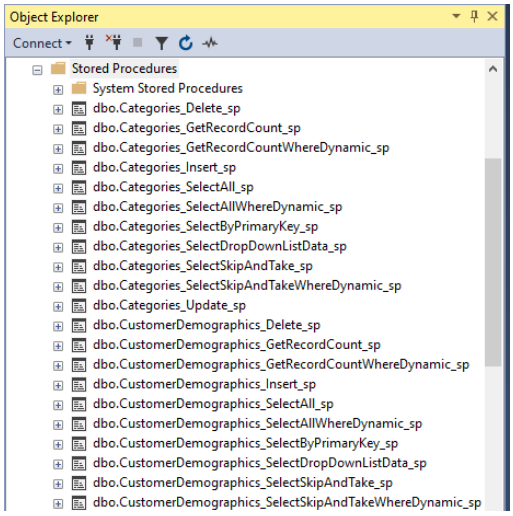
Stored Procedure Names with Prefix

2.2.3 Use Stored Procedures – Suffix

When you select *Suffix*, the generated *Stored Procedure Names* will have a prefix. The *Suffix* box is **required** when *Suffix* under *Stored Procedure* is selected. The *Suffix* you enter will be used as the *Suffix* for the generated *Stored Procedure Names*. E.g. *Categories_Delete_**sp**, Categories_GetRecordCount_**sp***.



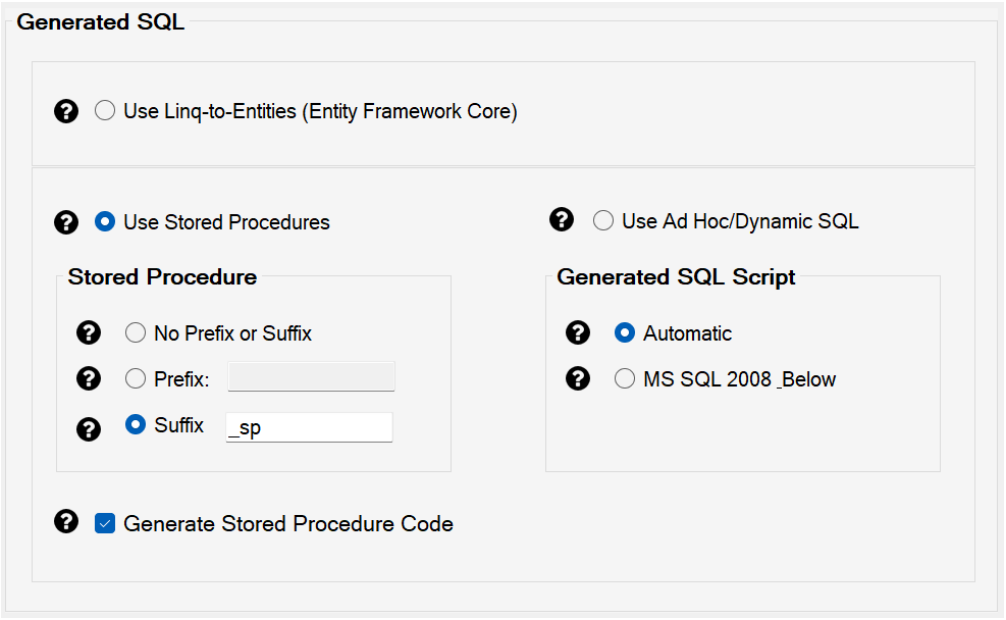
Suffix (filled)



Stored Procedure Names with Prefix

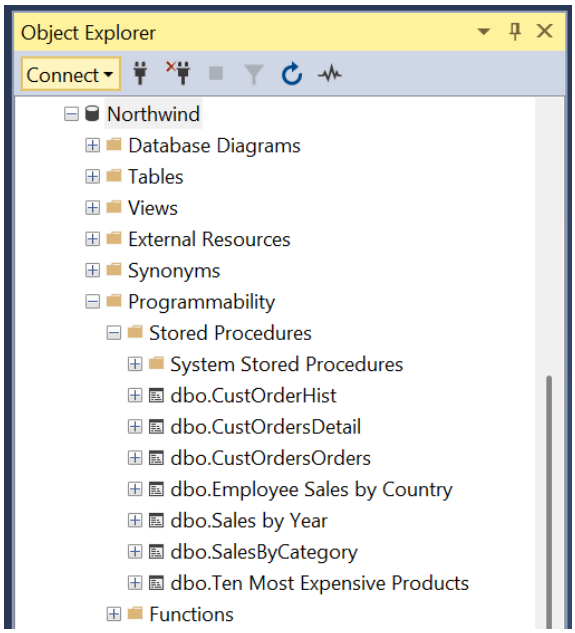
2.2.4 Generate Stored Procedure Code

When the *Generate Stored Procedure Code* is checked, Business Layer code for each of the existing stored procedure in your database is generated.



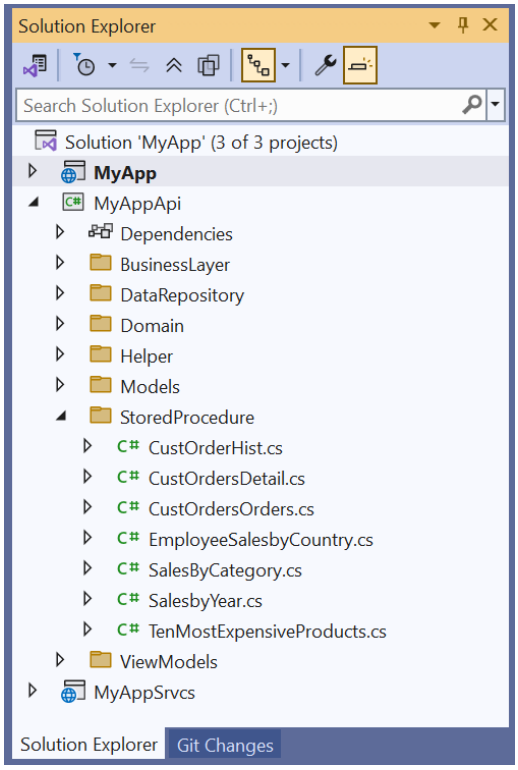
Generated Stored Procedure Code

For the example above, code for the existing stored procedures are generated except for the ones that have a Suffix of *_sp*, because in this example the stored procedures with an *_sp* suffix does not exist yet.



Existing Stored Procedures in the Database

In image snapshot above, it shows that you have 7 existing stored procedures in the database. A Business Layer class will be generated for each of these stored procedures.



Business Layer Classes Generated for Existing Stored Procedures

The Business Layer classes generated for the existing stored procedures will be in the *Stored Procedure* directory, under the Middle Layer project (Shared API libraries).

Here's an example of the generated Business Layer class for *CustOrderHist* stored procedure in Visual Studio. The class will have the same (or similar) name as the stored procedure's name.

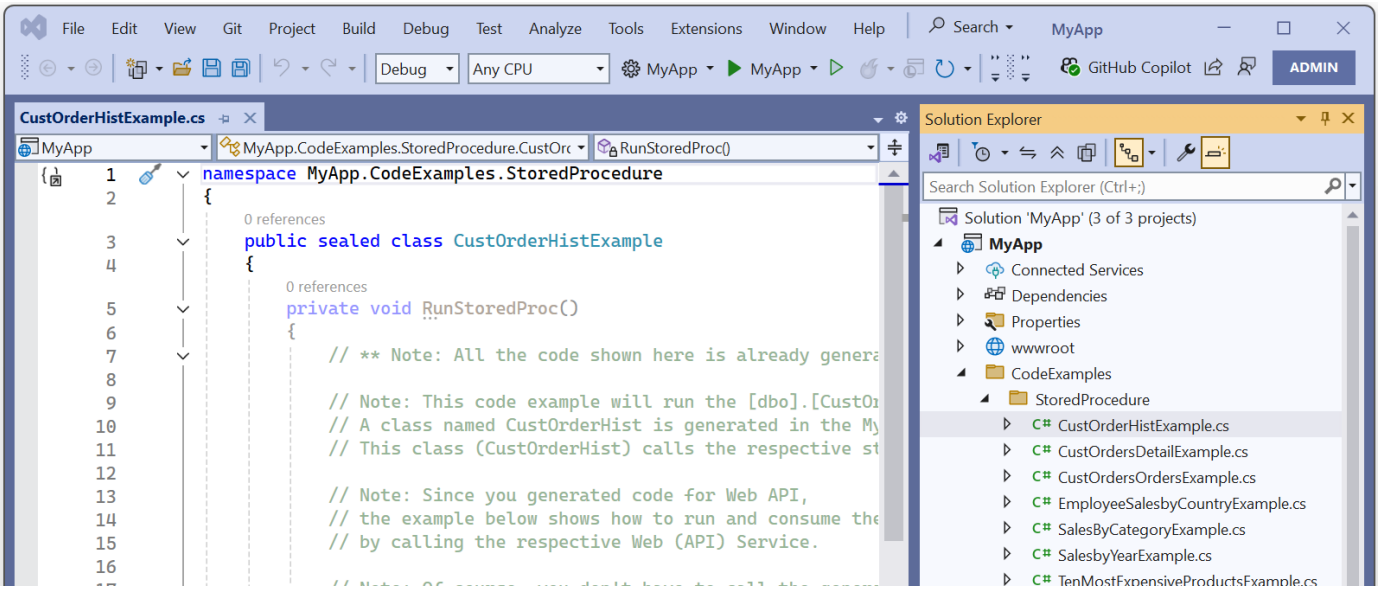
CustOrderHist.cs

MyAppApi — MyAppApi.StoredProcedure.CustOrderHist

```
40      /// <summary>
41      /// Runs the stored procedure dbo.CustOrderHist
42      /// </summary>
        0 references
43      public async Task<List<CustOrderHist>> Run(string customerID)
44      {
45          List<SqlParameter> sqlParamList = new();
46          List<CustOrderHist> objCustOrderHistList = null;
47
48          // add stored procedure parameters
49          DatabaseFunctions.AddSqlParameter(sqlParamList, "@CustomerID", customerID);
50
51          // get the data
52          DataTable dt = await DatabaseFunctions.GetDataTableAsync(_connectionString, "[dbo].[CustOrderHist]", sqlParamList, _commandType);
53
54          // add items to the list
55          if (dt != null && dt.Rows.Count > 0)
56          {
57              objCustOrderHistList = new List<CustOrderHist>();
58
59              foreach (DataRow dr in dt.Rows)
60              {
61                  // instantiate the class
62                  CustOrderHist objCustOrderHist = new();
63
64                  // assign values
65                  objCustOrderHist.ProductName = dr["ProductName"].ToString();
66
67                  if (dr["Total"] != System.DBNull.Value)
68                      objCustOrderHist.Total = (int)dr["Total"];
69                  else
70                      objCustOrderHist.Total = null;
71
72                  // add item to the list
73                  objCustOrderHistList.Add(objCustOrderHist);
74              }
75          }
76
77          return objCustOrderHistList;
```

Business Layer Class Generated for the Stored Procedure CustOrderHist

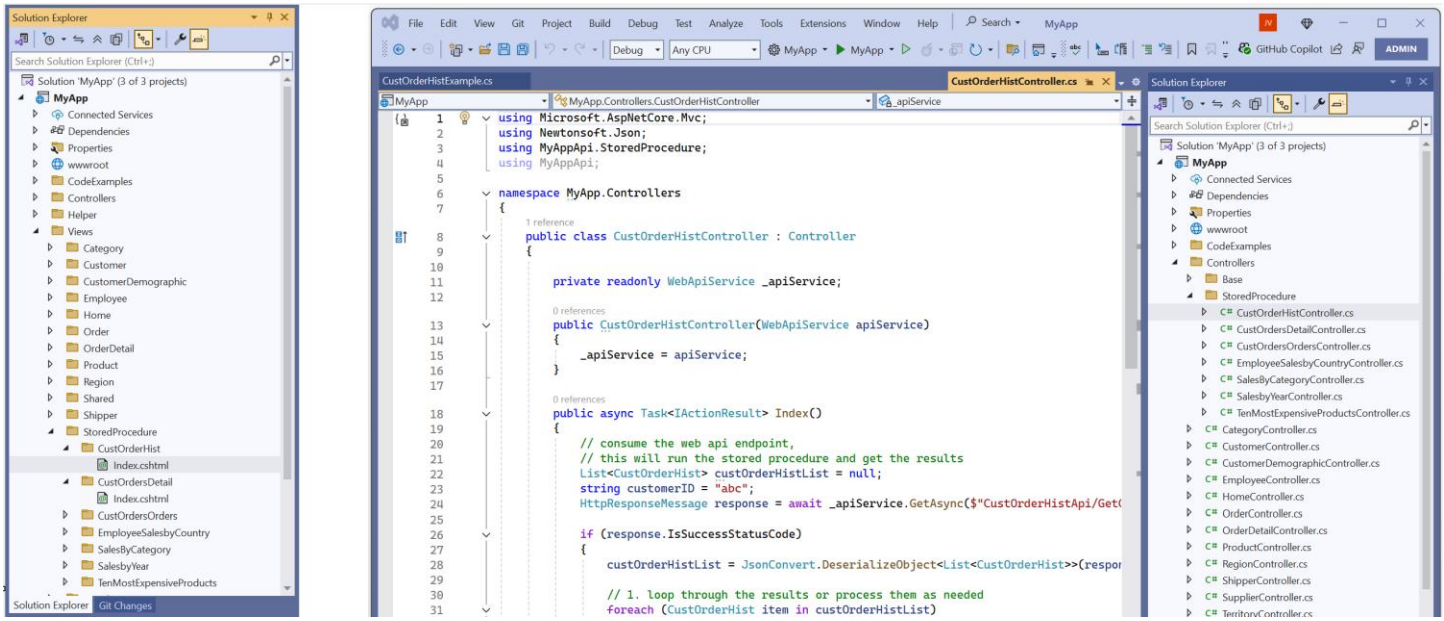
If the *Generate Code Examples* is checked under *Code Settings*, an example class is also generated for each of these existing stored procedures. Each of the classes shows how to run the respective stored procedure.



Example Class Generated for the Stored Procedure CustOrderHist

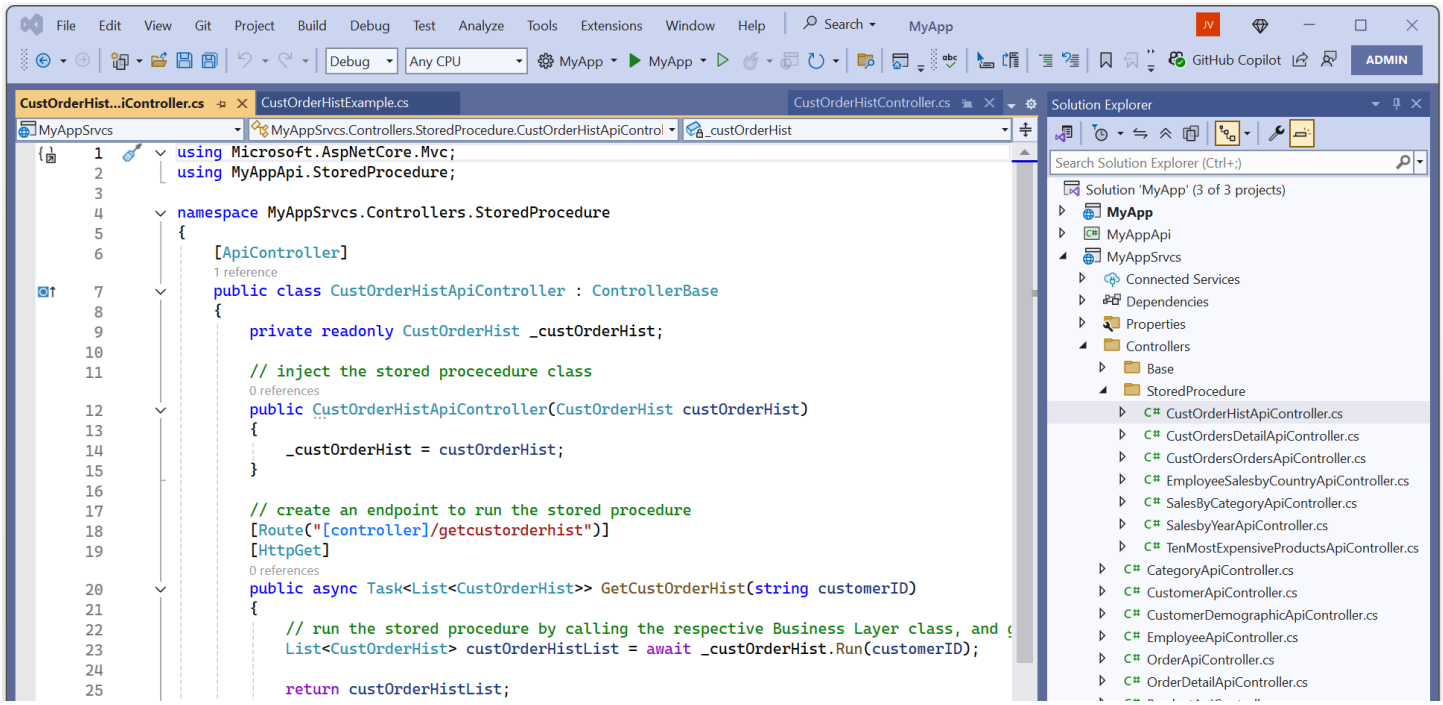
If the code example walk-through is not enough, a *View* (and *Controller*) is also generated for each one of the existing stored procedure. The *View* (and *Controller*) in this example simply shows how to Run the respective stored procedure. Both the *View* (and *Controller*) are generated in their respective *Stored Procedure* folders.

The generated *Controller* shows how to call/run the respective stored procedure, simply go to the Index page of the respective view. Since we generated code for *Web API (Web Services)*, the *Controller* shows how to consume the *Web API* service which in turn calls the respective *Business Layer* class. A few examples are shown on how to use the data retrieved (get) from the web service, or if the stored procedure is a non-query– simply how to run the respective stored procedure.



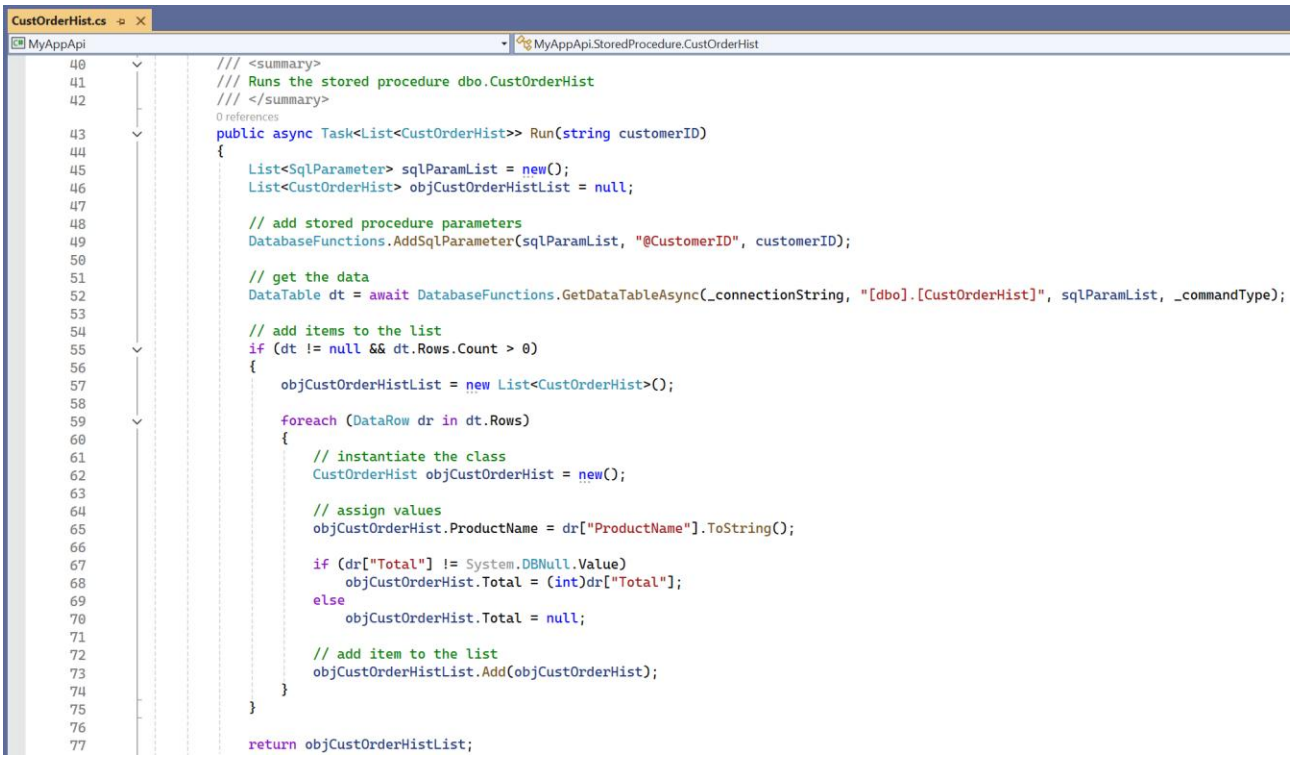
View and Controller for the CustOrderHist Stored Procedure

A *Web API (Service) Controller* is also generated for each of the respective existing stored procedure. As mentioned above, this is called by the Web App's controller. The respective *Web API (Service) Controller* calls the respective *Business Layer* class which in turn runs the respective stored procedure. You can see the Business Layer call in *line 23* in this example.



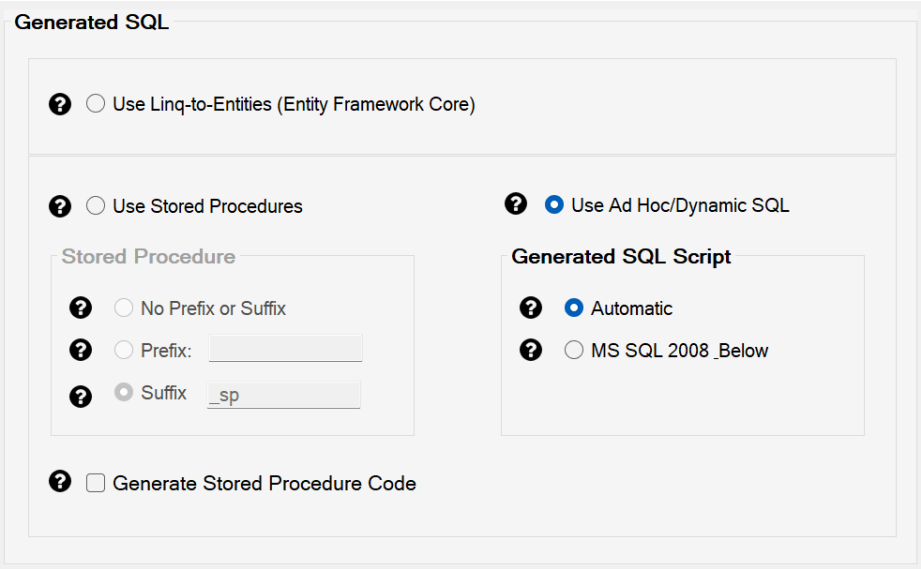
Web API Controller for the CustOrderHist Stored Procedure

Here's the Business Layer class called by the Web API above.



Business Layer Class Generated for the Stored Procedure CustOrderHist

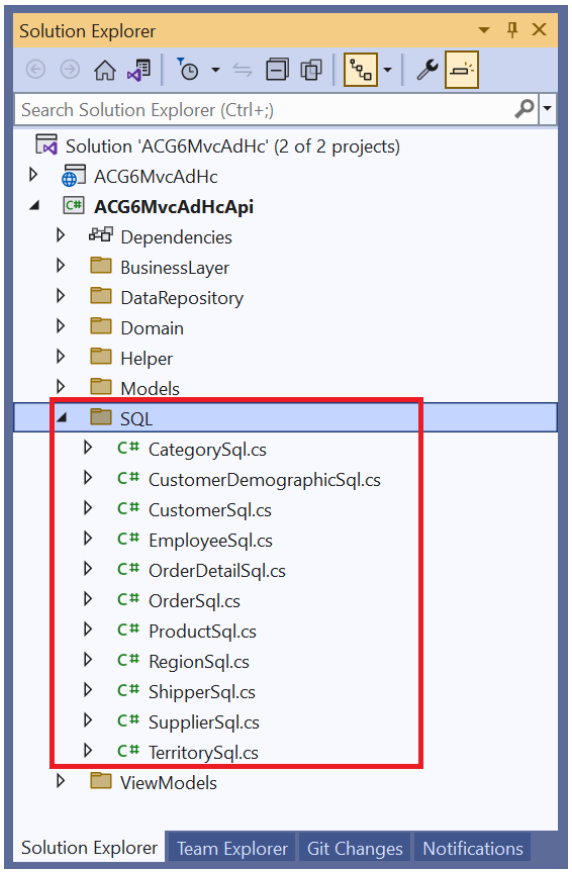
2.3 USE AD HOC/DYNAMIC SQL



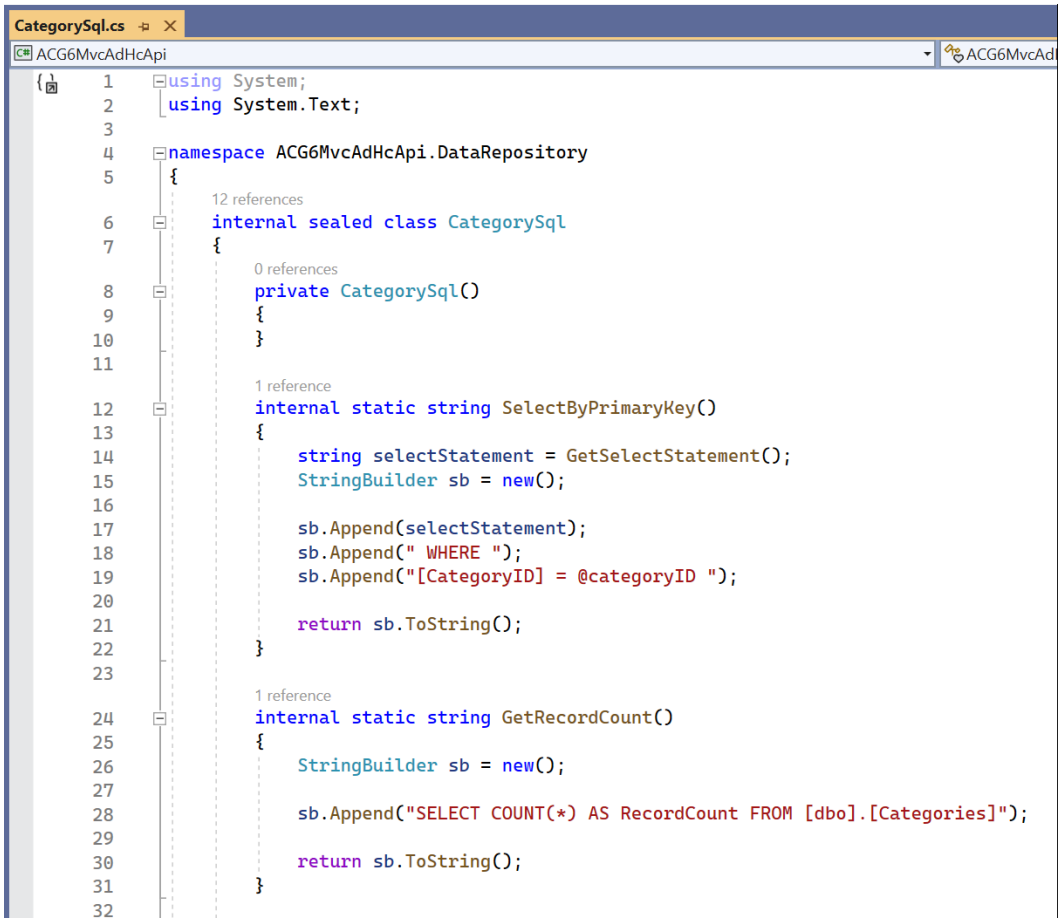
When you select *Use Ad Hoc/Dynamic SQL*, the generated *SQL Script* code will be in generated in a class file. The code will be generated in the:

Business Layer and Data Repository API Project (Class Library Project).

- SQL Directory



Generated Ad Hoc SQL Classes



CategorySql.cs – Generated SQL – Ad Hoc

2.3.1 Automatic

Automatic is selected by default. This will automatically determine the MS SQL Server you're using and generate scripts based on the version. Newer generated SQL script keywords are used for MS SQL Server versions that is 2008 and above.

2.3.2 MS SQL 2008 Below

When selected, SQL script keywords used in older (older than 2008) versions are used in the generated scripts. Although these keywords may still run using the newer versions of MS SQL Server, they may not be as fast.

You can read end-to-end tutorials on more subjects on using AspCoreGen 9.0 MVC Professional Plus that came with your purchase. These tutorials are available to customers and are included in a link on your invoice when you purchase AspCoreGen 9.0 MVC Professional.

Note: Some features shown here are not available in the Express Edition.

End of tutorial.