# Chapter 2: PDSC Developer Utilities Usage

The PDSC Developer Utilities (Figure 1) is a set of tools to help you develop your .NET applications and keep your development environment clean and working as efficient as it can. This chapter gives you an overview of the various utilities and describes the installation of the tool.


Figure 1: Screen shot of the PDSC Developer Utilities

# Overview of the Developer Utilities

After installing the PDSC Developer Utilities you will have the following programs that you can run.

| Utility | Description |
|---|---|
| Computer Cleaner | Visual Studio is a great development environment for creating applications quickly. However, it will leave a lot of miscellaneous files all over your hard drive. There are a few locations on your hard drive that you should be checking to see if there are left-over folders or files that you can delete. This utility will help you get rid of all of these left-over files. |

| Utility | Description |
|---|---|
| Property Generator | This utility allows you to generate Property statements for your C# or Visual Basic classes. There are several templates (similar to the snippets in the Visual Studio editor) that you can choose from. You can also create your own templates to generate any type of property you want. |
| Project Cleaner | This tool goes through a folder and all sub-folders and delete any \bin and \obj folders. It will also delete any .suo, .webinfo and .user files. You can optionally have it look in .SLN, VBProj, CSProj files and eliminate any references to source control. It will also remove any read-only attributes from the files. This utility is configurable so you can choose what folders and files you wish to delete. |
| JSON Generator | This utility allows you to choose a table, view or a SELECT stored procedure and will generate a JSON file of the data in the table or view. |
| XML Generator | This utility allows you to choose a table, view or a SELECT stored procedure and will generate an XML file of the data, or an XSD file of the schema of the table or view. |
| SQL Compare | This utility compares two SQL Server databases to determine what objects are missing, or have been changed between them. |
| C# Application Creator | This utility copies all the files and folders from the where you installed the PDSC Framework template project to a new folder and name that you specify. It then renames the appropriate files to the new application name you specify. |
| C# Entity Generator | This utility generates a C# entity class from a table in a database. |
| C# Repository Generator | This utility generates a C# hard-coded repository class from the data contained in table in a database. |

Table 1. List of PDSC Developer Utilities

# Computer Cleaner

Visual Studio and Visual Studio Code are two great development environments for creating applications quickly. However, both leave a lot of miscellaneous files all over your hard drive. There are a few locations on your hard drive that you should

check to see if there are left-over folders or files that you can delete. I have attempted to gather as much data as I can about the various versions of .NET and operating systems. Of course, your mileage may vary on the folders and files listed here. This utility attempts to find the various folders depending on which version(s) of Visual Studio/VS Code you have installed on your machine.

## Preview

You will first have to click on the **Preview** button. This will then display a list of files and folders that will be deleted (Figure 2). Be sure to review this list carefully. Then you can click on the **Clean** button.

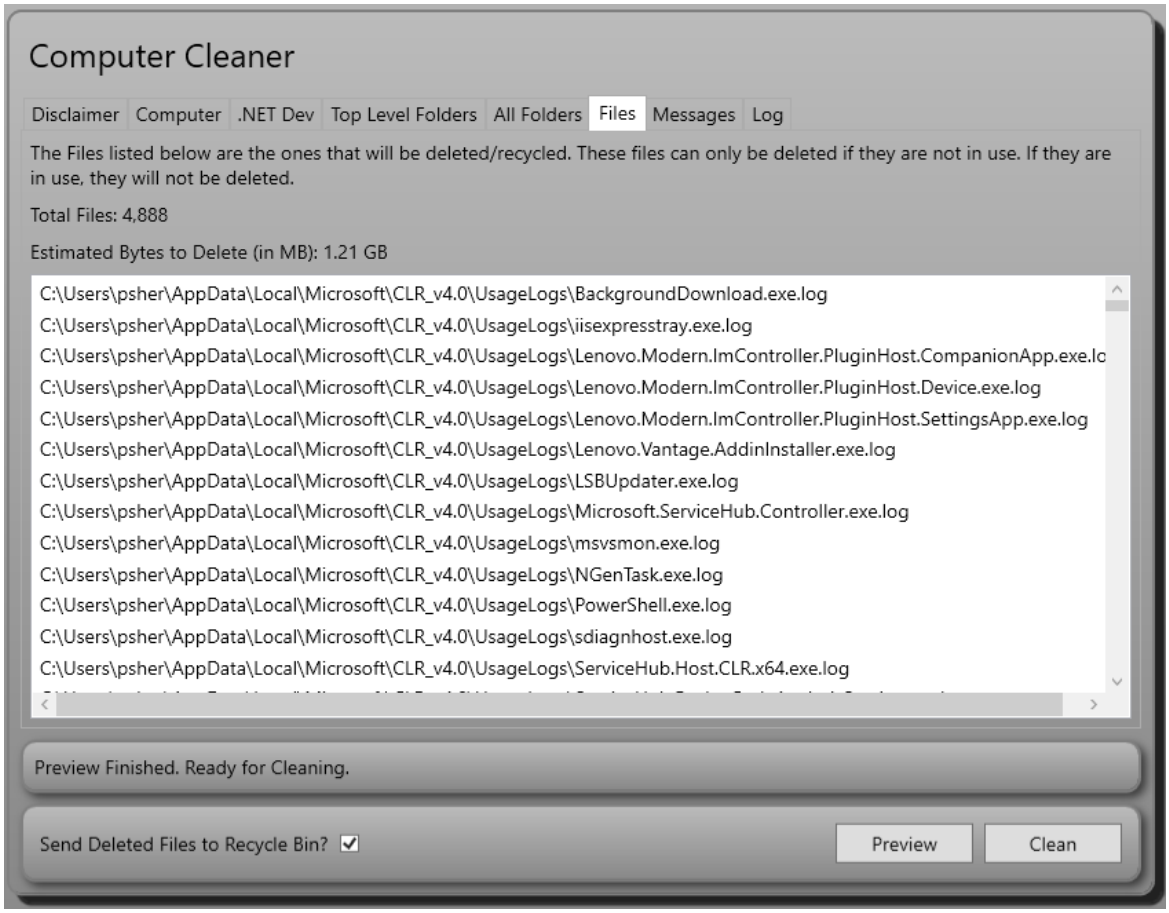


Figure 2: The Computer Cleaner utility will help you clean up your hard drive

Each version of VS.NET will create “temporary” files in different folders. The problem is that the files created are not always “temporary”. Most of these files do not get cleaned up like they should.

Click on the **Top Level Folders** tab to view the many folders where I have identified Visual Studio and some other utilities leave files that are no longer needed.

# Property Generator

Visual Studio has code snippets that will let you create properties (Figure 3). These snippets such as **prop** and **propfull** are great for normal one-at-a-time properties. However, when you wish to create a lot of properties, or you need other types of properties, this is where the PDSC Property generator can help you out.

This tool will allow you to put in a comma-delimited list of property names, choose a scope and a data type and will then generate all of the appropriate private variables and public property names in C# or Visual Basic. You will have a set of different templates to choose from that will allow you to create automatic properties, properties that raise the NotifyPropertyChanged event. You will also be able to add your own templates to control how you generate the properties.

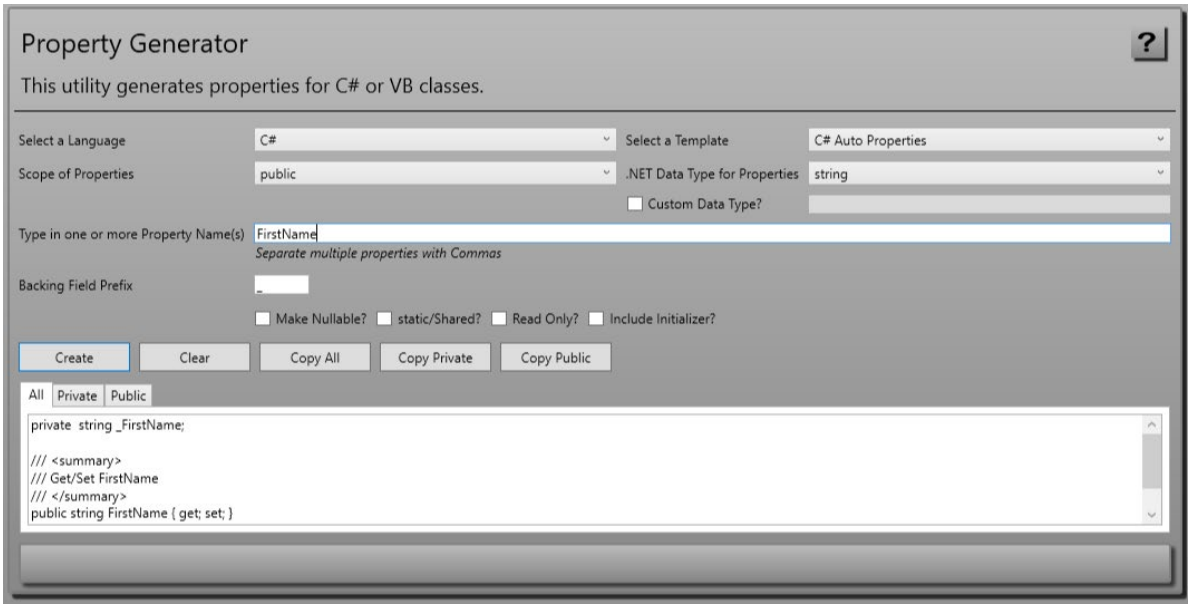


Figure 3: Property generator helps you create properties in many different styles

## Adding Your Own Templates

Under the folder where you installed the Developer Utilities you will find an \Xml folder (Figure 4). In that folder is a file named PropertyTemplates.xml. This contains the list of template files that you can use to generate properties. There is also a folder named \Resources where all of the .txt files that hold the snippets for each of the types of properties that you can generate.

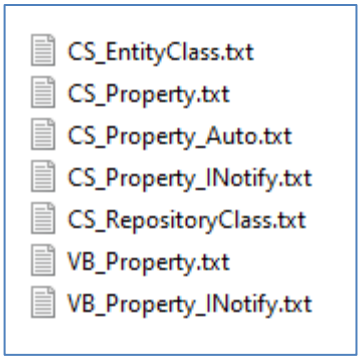


Figure 4: All the property snippets are just contained in .txt files

You will find one XML node in the PropertyTemplates.xml file for each .txt file located in the \Resources folder. To add a new template, you should just copy one of the existing .txt files and give it a new name.

As an example, let's say you wanted to add a method call from every property "setter". You could copy the CS_Property.txt and call it CS_Test.txt. Open the CS_Test.txt in Notepad. It should look something like the following:

```
<|SCOPE|><|STATIC|> <|DATATYPE|> <|PUBLICNAME|>
{
   get { return <|PRIVATENAME|>; }
   <READONLY>set { <|PRIVATENAME|> = value; }</READONLY>
}
```

You can now expand the "set" portion and add your own method call by changing this code to look something like the following:

```
<|SCOPE|><|STATIC|> <|DATATYPE|> <|PUBLICNAME|>
{
   get { return <|PRIVATENAME|>; }
   <READONLY>set
   {
     <|PRIVATENAME|> = value;
     MyMethod("<|PUBLICNAME|>");
   }</READONLY>
}
```

In the above template you broke up the "set" onto separate lines and then added a call to a method called MyMethod and you pass in as a string the public property name.

Next you need to add a new node to the PropertyTemplate.xml file. Copy an existing node and paste it immediately after one of the descendant nodes. Modify the Description element to something you will recognize and the FileName element to the name of your new .txt file.

```
<PropertyTemplate>
  <Description>C# My Method Get/Set</Description>
  <FileName>CS_Test.txt</FileName>
  <Language>CSharp</Language>
  <GenPrivateVars>True</GenPrivateVars>
  <GenPublic>True</GenPublic>
</PropertyTemplate>
```

Now, restart the PDSC Developer Utilities and your new template will now appear.

# Property Generator Tokens

In the .txt files that represent the code to generate for the properties you find a set of tokens in the format <|TOKEN_NAME|>. There are just a few tokens that are recognized by our property generator. Table 2 contains the list of the tokens that you can use in your templates.

| Token | Description |
|---|---|
| CONVERSIONMETHOD | Based on the Data Type you choose for the variable, you can wrap the private or public property into a Convert.<\|CONVERSIONMETHOD\|>. For example, if you are generating a string property and you store it into a session variable, you will want to convert it to a string when you bring it back from the session variable. You would then write code like the following:<br>*get { return Convert.<\|CONVERSIONMETHOD\|>*<br><br>*(HttpContext.Current.Session["<\|PUBLICNAME\|>"]); }* |
| DATATYPE | The data type you choose for this property. This type comes from the DataTypes??.xml file. |
| PRIVATENAME | The private variable name for this property. |
| PUBLICNAME | The public property name. |
| <READONLY></READONLY> | Wrap these tokens around your "set" property to remove the "set" if you choose "read only" on the interface. |
| SCOPE | The scope you choose in the user interface for your property. These scopes come from the Scope??.xml files. |
| SHARED | Will generate "Shared" or "static" on your property if you choose this option on the interface. |

Table 2. List of Tokens in Property generator

## Other XML files for the Property Generator

There are a few other XML files that the property generator uses to assist with the generation. These are located in the \Xml folder under the location where you installed the Developer Utilities.

| Xml File name | Description |
|---|---|
| DotNetLanguages | The list of .NET languages. |
| LanguageDataTypes | A list of data types for C# and Visual Basic. |
| LanguageScope | A list of scopes for C# and Visual Basic. |

Table 3. List of XML files for the Property Generator

# Project Cleaner

When you create a project in Visual Studio, compile in different modes, and add the project to source control; a set of files and folders are created under your original project folder. Sometimes you might want to delete all these folders and files. For example, if you wish to give your project to someone else that is not on your network, does not have access to your source control, or you just want to clean up the folders under your project prior to adding your project for the first time to source control, you will want to eliminate all these extra files and folders using the Project Cleaner shown in Figure 5.

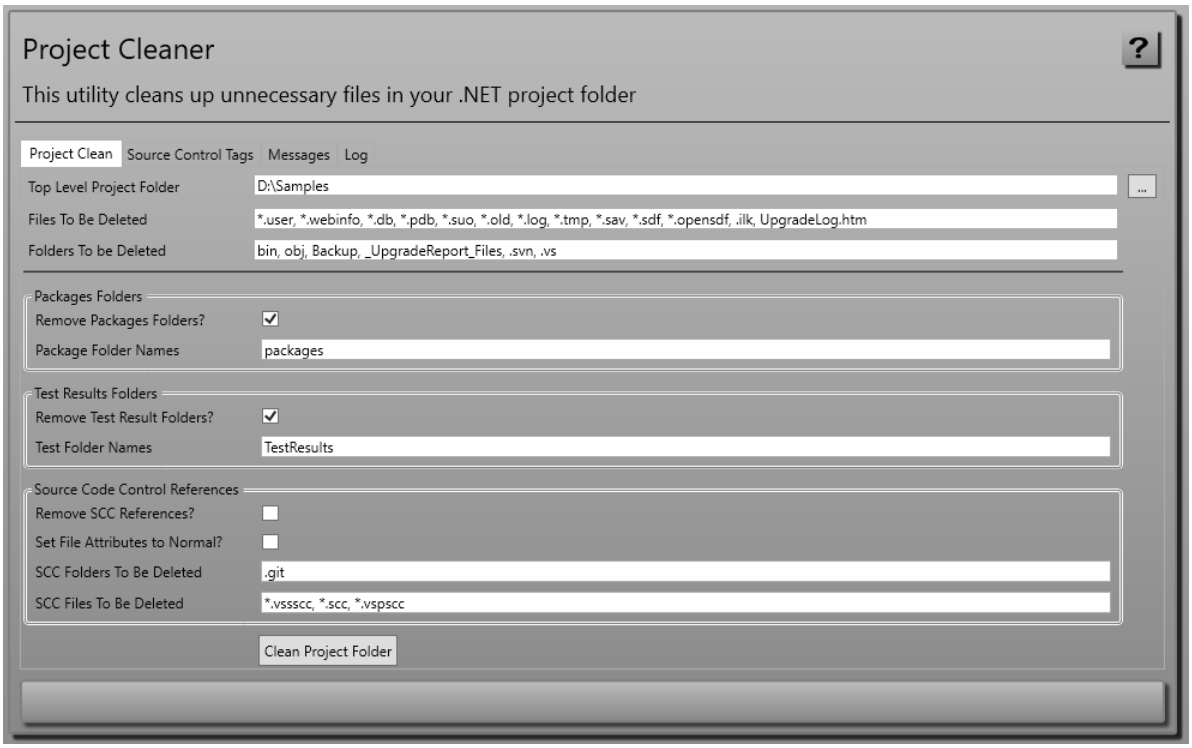


Figure 5: Clean up files using the Project Cleaner utility

You will first enter a top-level folder and the Project Cleaner utility will iterate through all the lower level folders and files underneath this folder and perform a series of operations. The operations performed will depend on what you fill in on the form in the following fields:

| Field | Description |
|---|---|
| Top Level Project Folder | Enter the top level folder you wish to iterate through |
| Files to be Deleted | A list of file extensions that should be removed. |
| Folders to be Deleted | A list of folder names that should be deleted. |
| | |
| Remove Packages Folders? | Remove the "packages" folder. |
| Package Folder Names | Fill in the names of the packages folders to remove. |
| | |
| Remove Test Result Folders? | Check to remove any test result folders. |
| Test Folder Names | Fill in the names of the test result folders to remove. |
| | |

| | |
|---|---|
| Remove SCC References? | Check this is you wish this utility to remove the folders and files listed and to also open your .SLN and any .csproj or .vbproj files and remove the source control tags from these files. |
| Set File Attributes to Normal? | Check this to set the attribute of all files under the Top Level Folder to normal. |
| SCC Folders to be Deleted | A list of source control folders that should be removed. |
| SCC Files to be Deleted | A list of source control file extensions that should be removed. |

Table 4: Fields to fill in for cleaning projects.

> **NOTE**: This utility only goes thru the folder and sub-folders specified in the **Top Level Folder** field. If the solution in the top-level folder points to another project in another folder structure, that other project will NOT have any of its attributes reset, or its source control references removed.

# JSON Generator

JSON files are very handy for a lot of things. If you have data in a database you might want to generate some JSON files from that data. The PDSC JSON Generator utility will build a JSON file from any table or view in your SQL Server database.

## Step 1: SQL / Select Object to Generate

To start the JSON generation process, put in the appropriate connection string that will connect you to your database (Figure 6). Select the type of objects to load (Tables or Views). If you have a large collection of objects in your database, you may wish to fill in a Schema name (or partial schema name), and/or an object name (or partial object name) prior to clicking on the Load button.

Click on the Load button to load all objects in the database. Click on one of the objects in the list and the appropriate SQL statement will be generated in the text box below the object list. You can modify this SQL prior to moving to step 3 if you wish to generate different names for your element or attribute names.

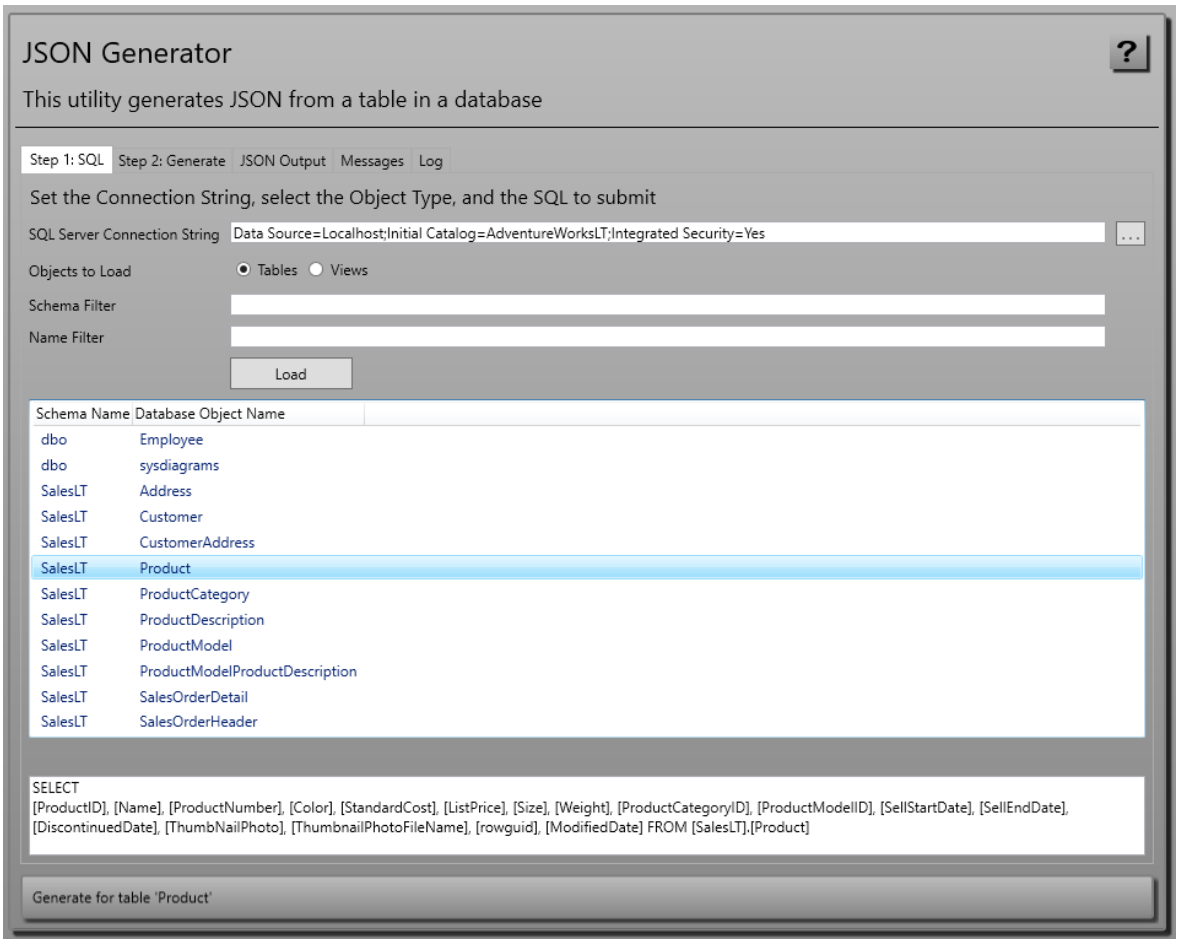


Figure 6: Step 1: JSON Generator SQL Tab

# Step 2: Generate

Click on **Step 2: Generate** (Figure 7) to fill in information on how you wish to generate the JSON file. You can specify the name of the file and the folder for the JSON file. A ".json" file extension will automatically be added to the file name. You will be prompted to overwrite this file if you check the **Prompt to Overwrite?** check box.

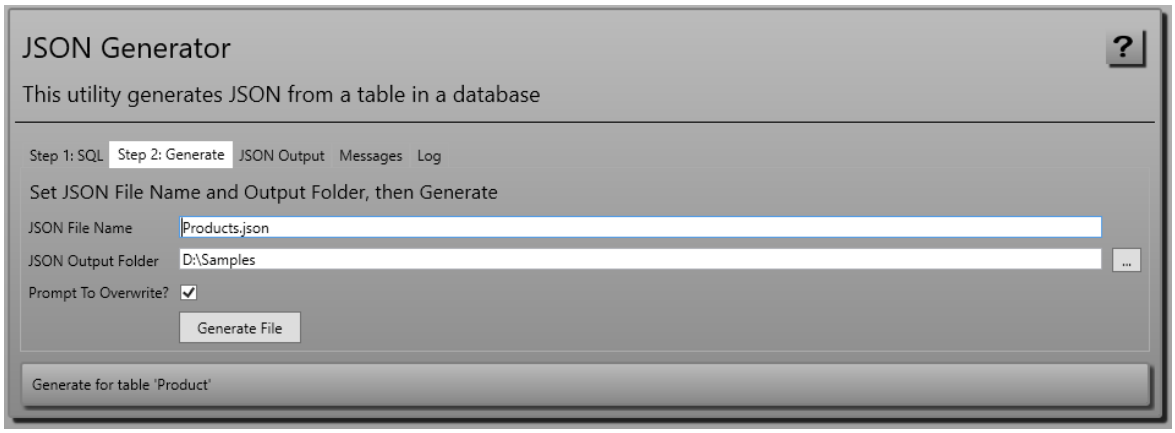


Figure 7: Step 2: JSON Generator Generate Tab

Click the **Generate File** button to start the generation process.

## View the JSON Output

After you click on the **Generate File** (Figure 8) button, you are presented with the screen shown in Figure 8. This screen shows you where the generated .json file is located and the JSON output.

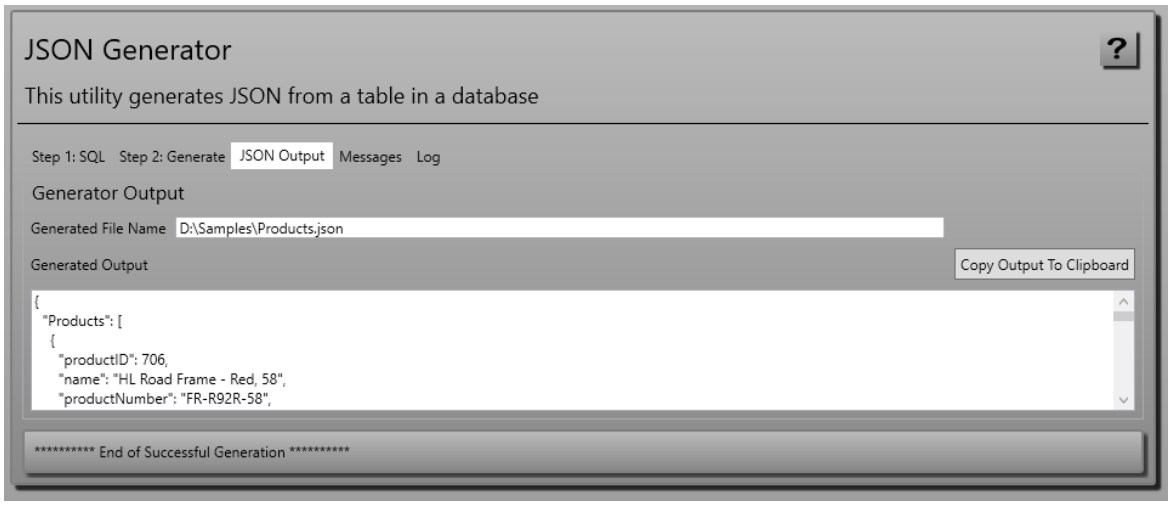


Figure 8: JSON Generator Output tab

# XML Generator

XML files are very handy for a lot of things. If you have data in a database you might want to generate some XML files from that data. The PDSC XML Generator utility will build XML and XSD files from any table or view in your SQL Server or Oracle database.

# Step 1: SQL / Select Object to Generate

To start the XML generation process, put in the appropriate connection string that will connect you to your database (Figure 9). Select the type of objects to load (Tables or Views). If you have a large collection of objects in your database, you may wish to fill in a Schema name (or partial schema name), and/or an object name (or partial object name) prior to clicking on the Load button.

Click on the Load button to load all objects in the database. Click on one of the objects in the list and the appropriate SQL statement will be generated in the text box below the object list. You can modify this SQL prior to moving to step 3 if you wish to generate different names for your element or attribute names.

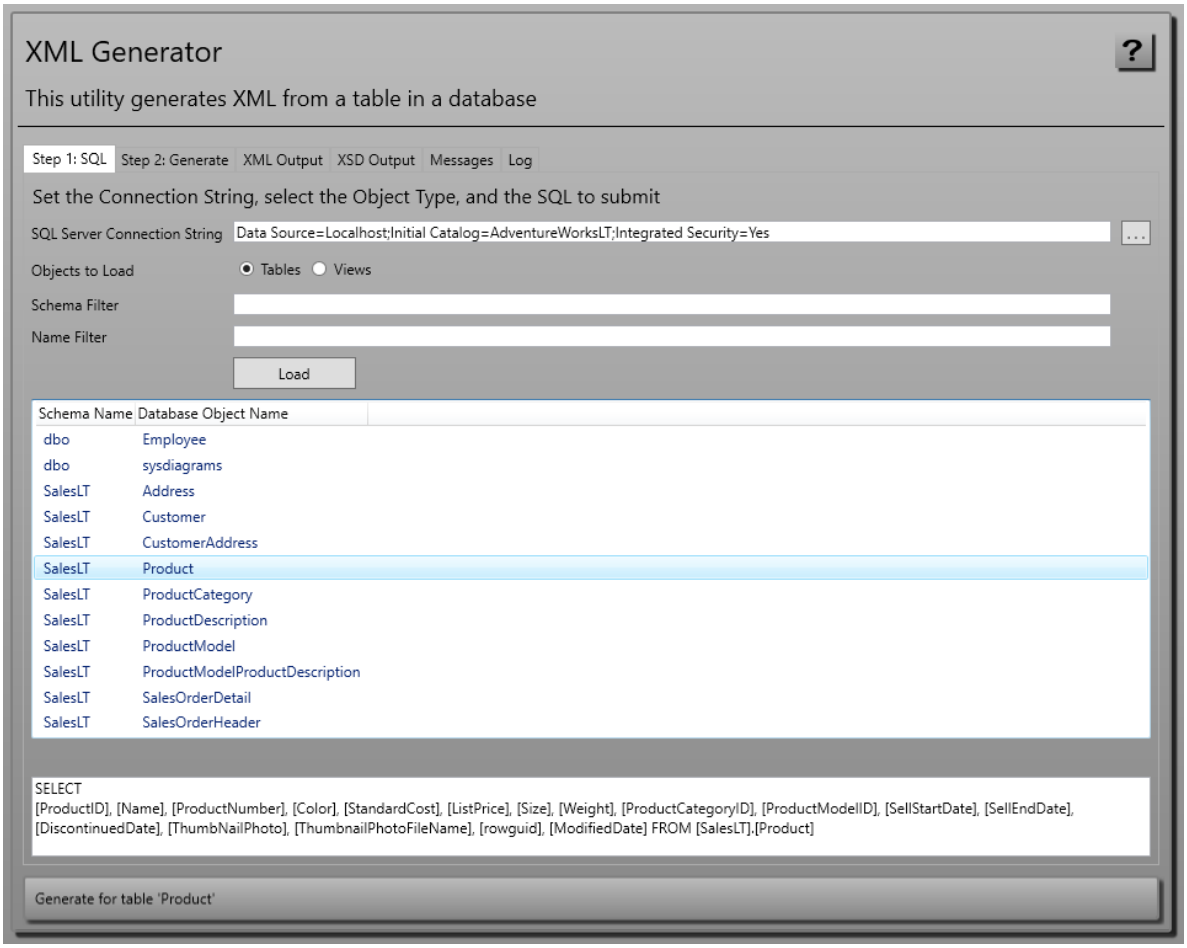


Figure 9: Step 1: XML Generator SQL Tab

# Step 2: Generate

Click on **Step 2: Generate** (Figure 10) to fill in information on how you wish to generate the XML/XSD files. You can specify your Root Element Name, and for each row the Child Element Name to use. Check the **Write XSD File?** to generate

an XSD file. Check the **Create Attribute-Based XML?** to generate attribute-based XML file.

Fill in the name of the XML file name and XML Output folder. Fill in the XSD file name and XSD output folder. You will be prompted to overwrite this file if you check the **Prompt To Overwrite?** check box.

Click the **Generate File(s)** button to start the generation process.

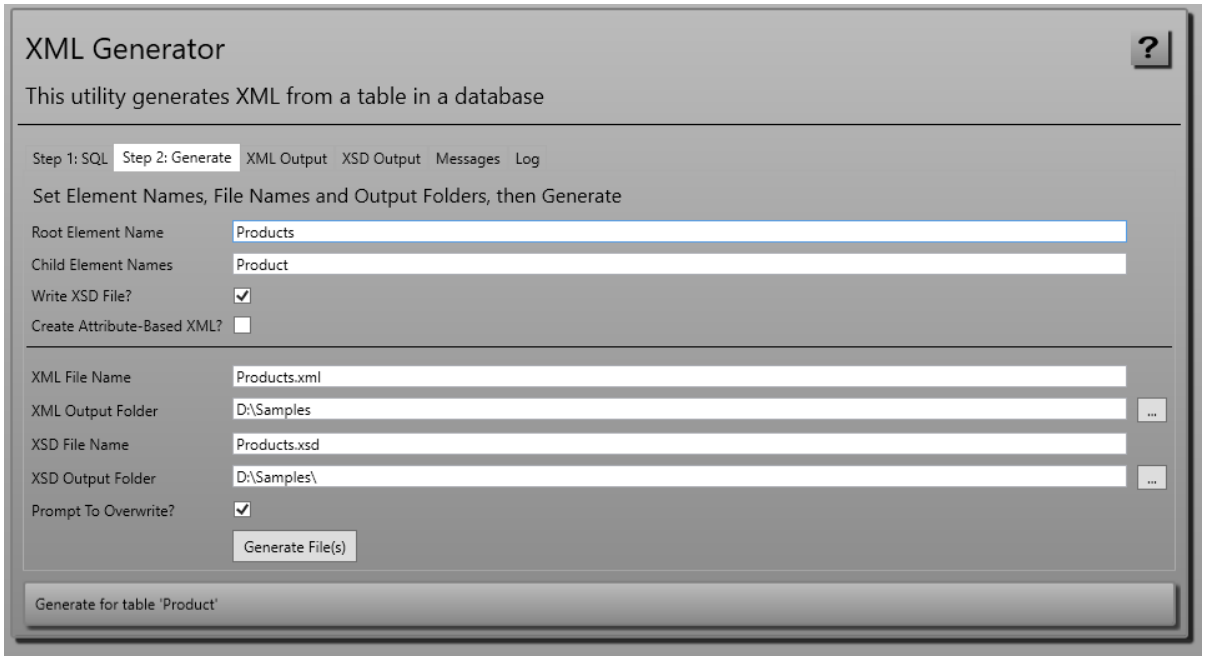


Figure 10: Step 2: XML Generator Generate Tab

# XML Output

After you click on the **Generate Files** button, you will be presented with the screen shown in Figure 11.

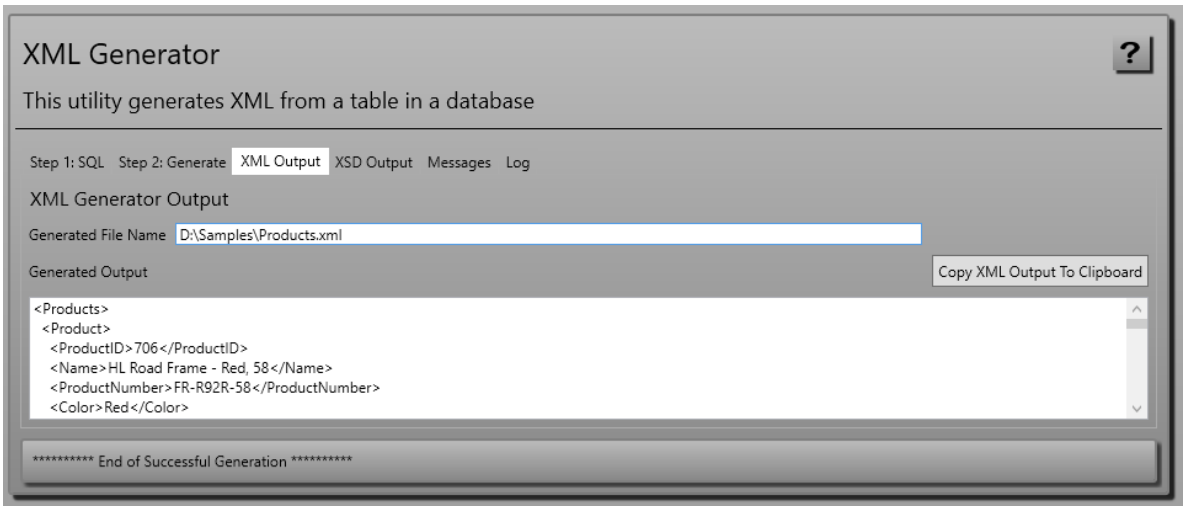


Figure 11: XML Output Tab

## XSD Output

If you generated an XSD, you can view the XSD on the screen show in Figure 12.

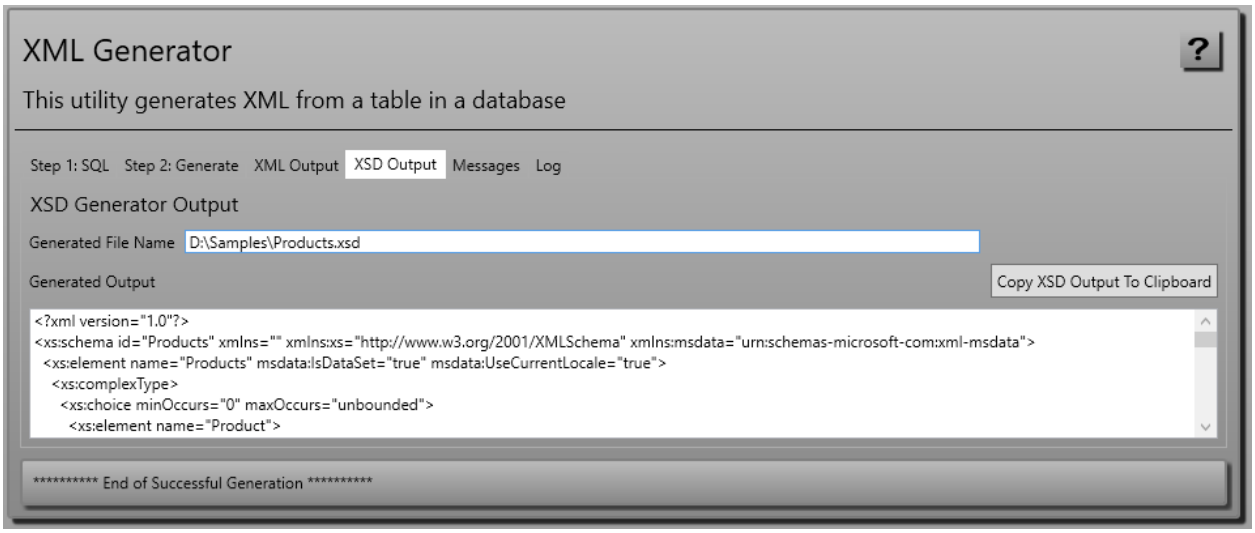


Figure 12: XSD Output tab

# SQL Compare

When you run the SQL Compare utility, you put in two different connections string that point to similar databases. For example, maybe you need to find out what you changed in your QA database compared to your Production database. Click the Compare button (shown in Figure 13) and a complete list of missing or changed objects will appear in the messages tabs at the bottom of the screen.

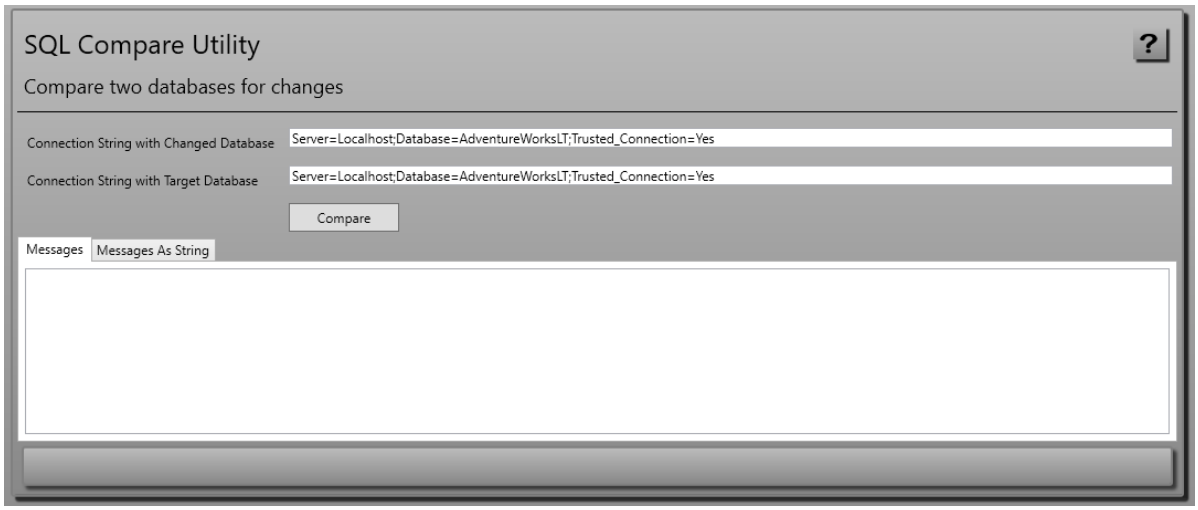


Figure 13: Get missing objects from one database to another via the SQL Compare Utility.

# C# Application Creator

Click on the C# App Creator menu to see a screen that looks like Figure 14. Modify the "New Application Location" to a valid hard drive, and folder, on your system and click the **Create** button. If you get an error that one of the paths is incorrect, fix it up, then click the Create button again. In just a few seconds you should receive a message that the process is complete, and you will see a bunch of messages as seen at the bottom of the screen.

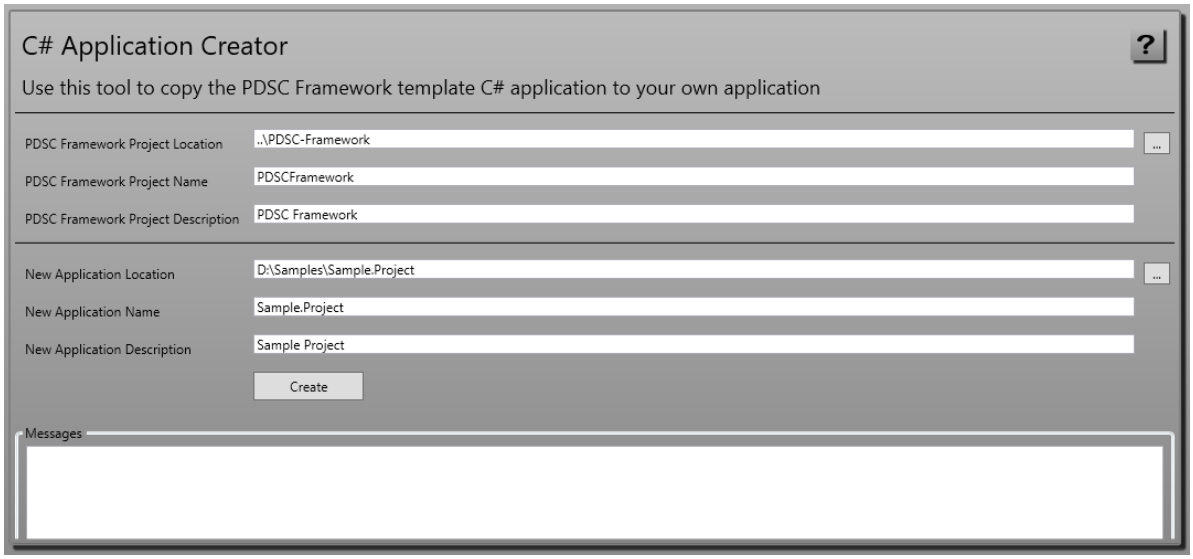


Figure 14: The PDSC C# Application Creator helps you build a new MVC project.

You can now go to the "New Application Location" folder and view the results of running this tool as shown in Figure 15.

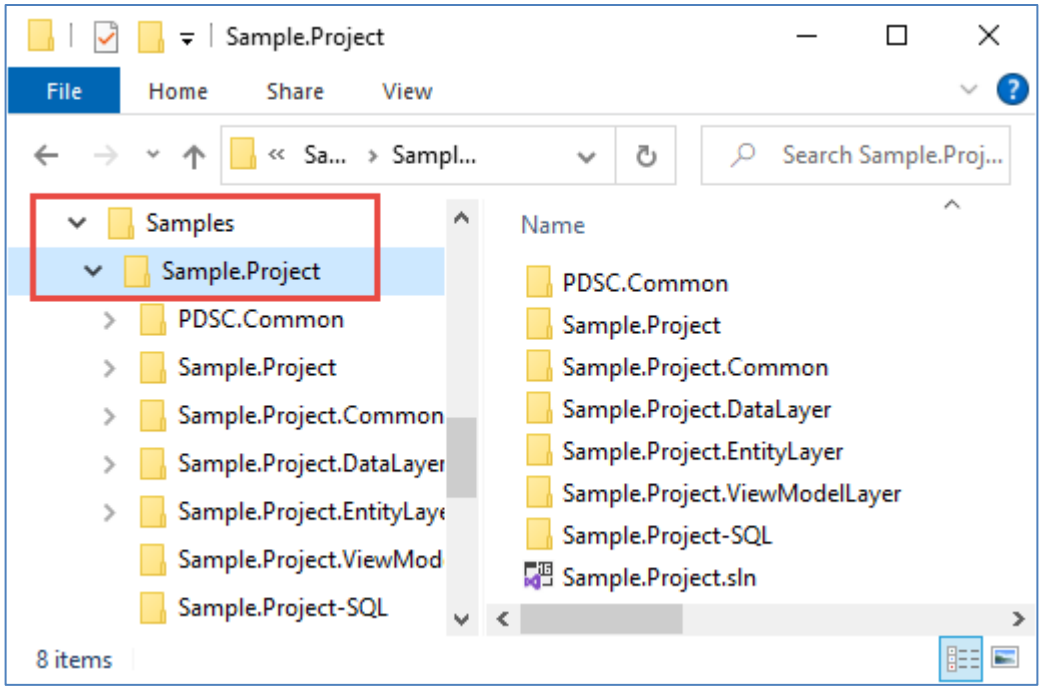


Figure 15: In the Application Location folder you find a folder structure like that of the PDSC Framework template.

# Create a Sample Database

Open the SQL Server Management Studio and create a new database named **Sample.Project**. Open the "Sample.Project-SQL" folder and locate the **Sample.Project.sql** file (Figure 16) and load that file into SQL Server Management Studio. Run this script to create the database objects. Open the **Sample.Project-Data.sql** file in SQL Server Management Studio and run this script to add data to the database objects.

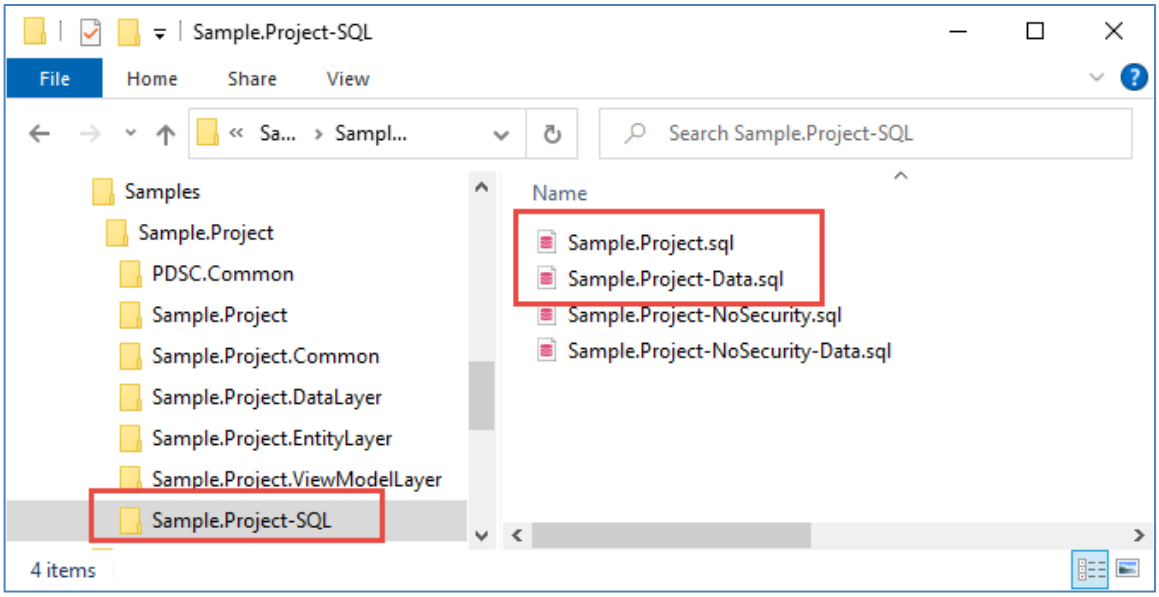


Figure 16: Locate the .SQL files to install in your new application folder.

## Run the Sample Project

Go to the **\Sample.Project** folder and double-click on the **Sample.Project.sln** file. Run this project. When you get to the home page, click on the Login link and login with 'bill@microsoft.com' and the password 'P@ssw0rd'. If you have done everything correctly, you should now be logged into your sample application.

## Next Steps

Please read the chapter on the **Haystack Code Generator** for information on how to generate add/edit/delete pages for your new project.

# C# Entity Generator

A typical entity class in C# is one that has a one-to-one correlation between the properties of the class and a table in a database. If you have a Product table in a database, this tool generates a Product class with one property for each column in your Product table.

## Step 1: SQL and Object Selection

To start the C# entity class generation process, put in the appropriate connection string that will connect you to your database (Figure 17). Choose whether you wish to load Tables or Views by selecting the appropriate radio button. If you have a

large collection of objects in your database, you may wish to fill in a Schema Filter (partial schema name), and/or a Name Filter (partial object name) prior to clicking on the Load button.

After clicking on the Load button, you will be presented with a list of database objects that match your specific filter. Click on one of the objects in the list and the appropriate SQL statement will be generated in the text box below the object list. You can modify this SQL prior to moving to step 3 if you wish to generate different names for your element or attribute names.

If you type in your own SQL, add on "WHERE 1 = 0" at the end so no records are generated. This tool only requires an empty result set to get the list of columns from which to generate the entity class.

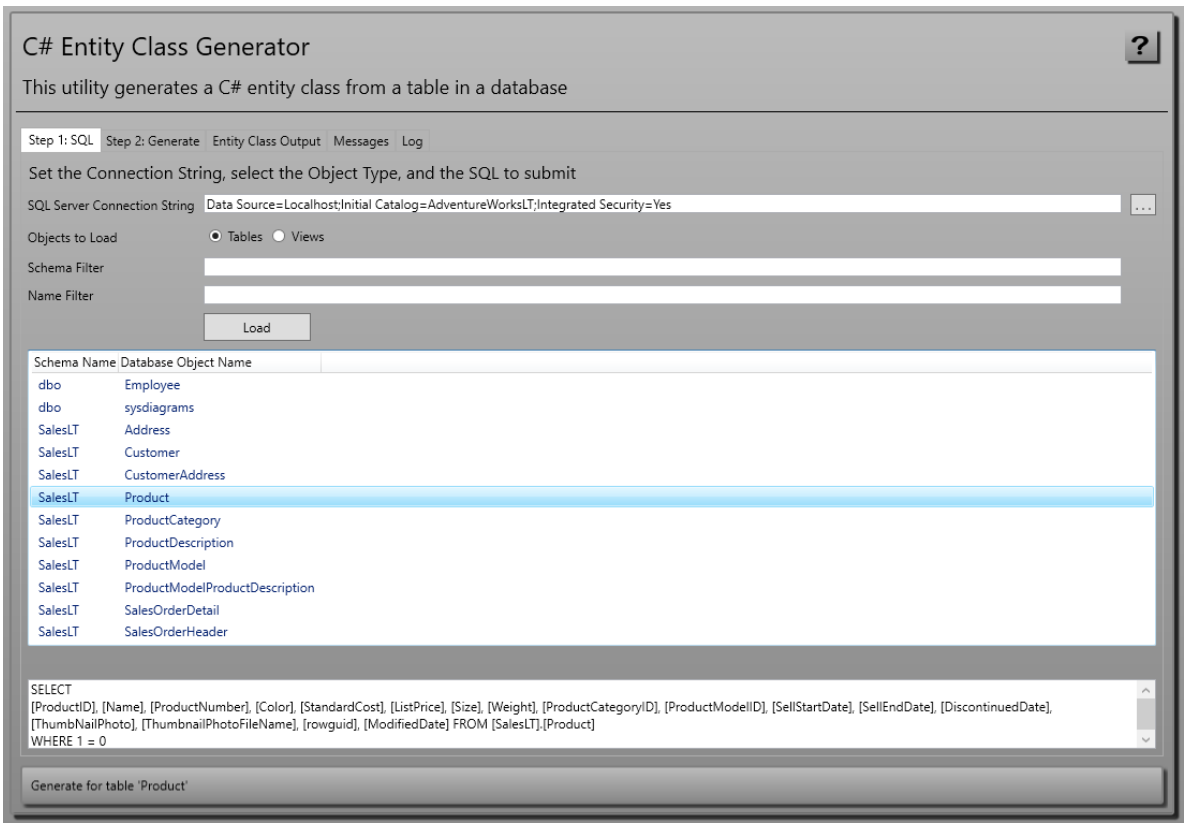


Figure 17: Step 1: C# Entity Class Generator Connection Tab

# Step 2: Generate

Click on **Step 2: Generate** (Figure 18) to fill in information on how you wish to generate the C# entity class. Fill in the Namespace to use, the entity class name, and if you wish to generate Data Annotations for each property and if you wish to include the #nullable disable statement at the top of the file. Next, fill in the C# File Name, the C# Output Folder and check the **Prompt to Overwrite?** check box if you want to be prompted before overwriting a previously written file. Click the **Generate File** button to start the generation process.

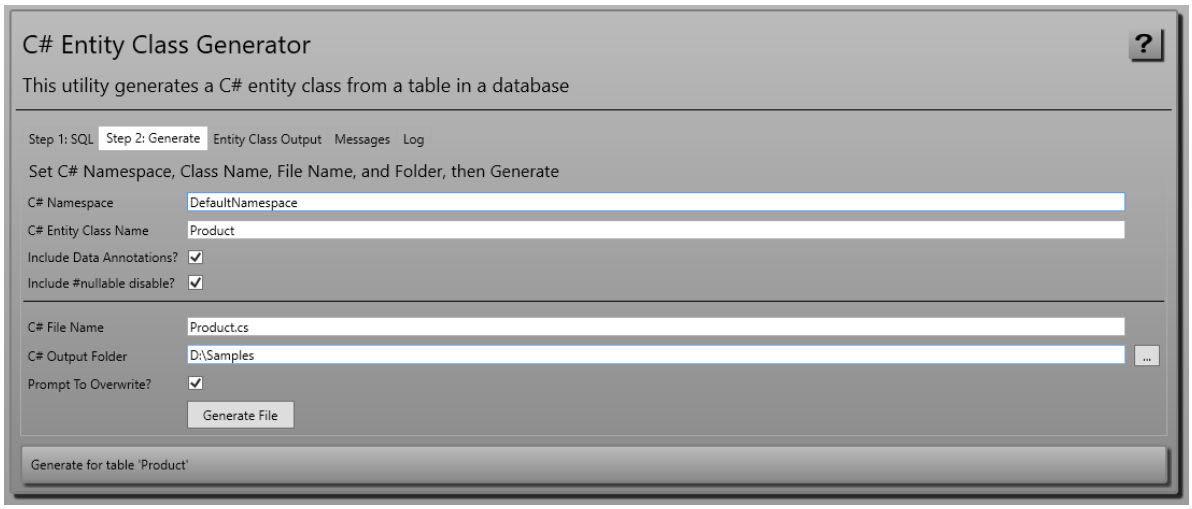


Figure 18: Step 2: C# Entity Class Generator Generate Tab

## Step 3: Output

After you click on the **Generate File** button, you will be presented with the screen shown in Figure 19. This screen tells you where the C# entity file is located and allows you to copy the entity class to the clipboard.

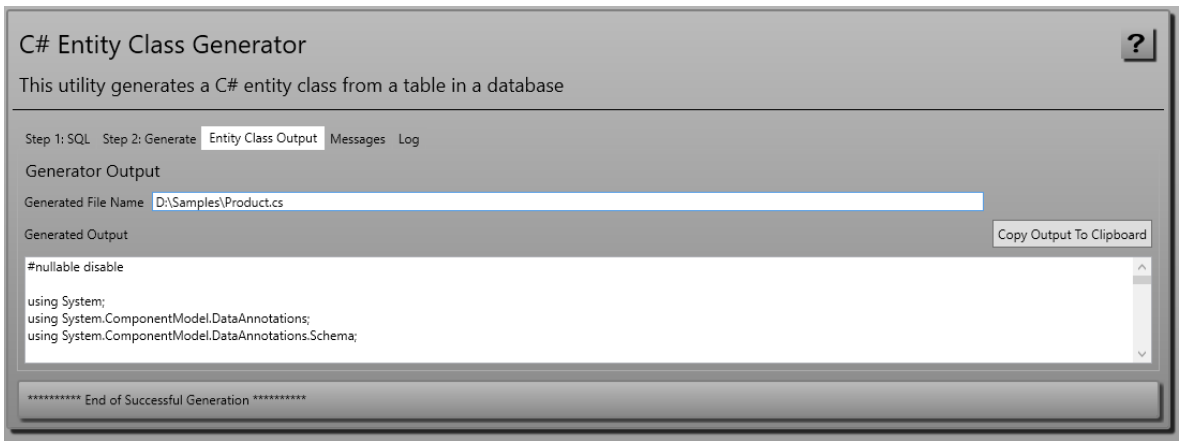


Figure 19: Step 3: C# Entity Class Generator Output tab

# C# Repository Generator

A repository class is one that has methods to return data from a data source. When creating exercises for a training class, or to test some functionality, but you don't want to have to connect to a database, it is nice to have a collection of hard-coded data that can be returned. Instead of you having to create all this data by hand, if

you have a table with data, you can use this generator to select some data and have it hard-coded into a repository class.

# Step 1: SQL and Object Selection

To start the C# repository class generation process, put in the appropriate connection string that will connect you to your database (Figure 17). Choose whether you wish to load Tables or Views by selecting the appropriate radio button. If you have a large collection of objects in your database, you may wish to fill in a Schema Filter (partial schema name), and/or a Name Filter (partial object name) prior to clicking on the Load button.

After clicking on the Load button, you will be presented with a list of database objects that match your specific filter. Click on one of the objects in the list and the appropriate SQL statement will be generated in the text box below the object list. You can modify this SQL prior to moving to step 3 if you wish to generate different names for your element or attribute names.

If you type in your own SQL, add you may add on "*SELECT **TOP 100***" or some other number if you don't want to generate all the records in the table/view.

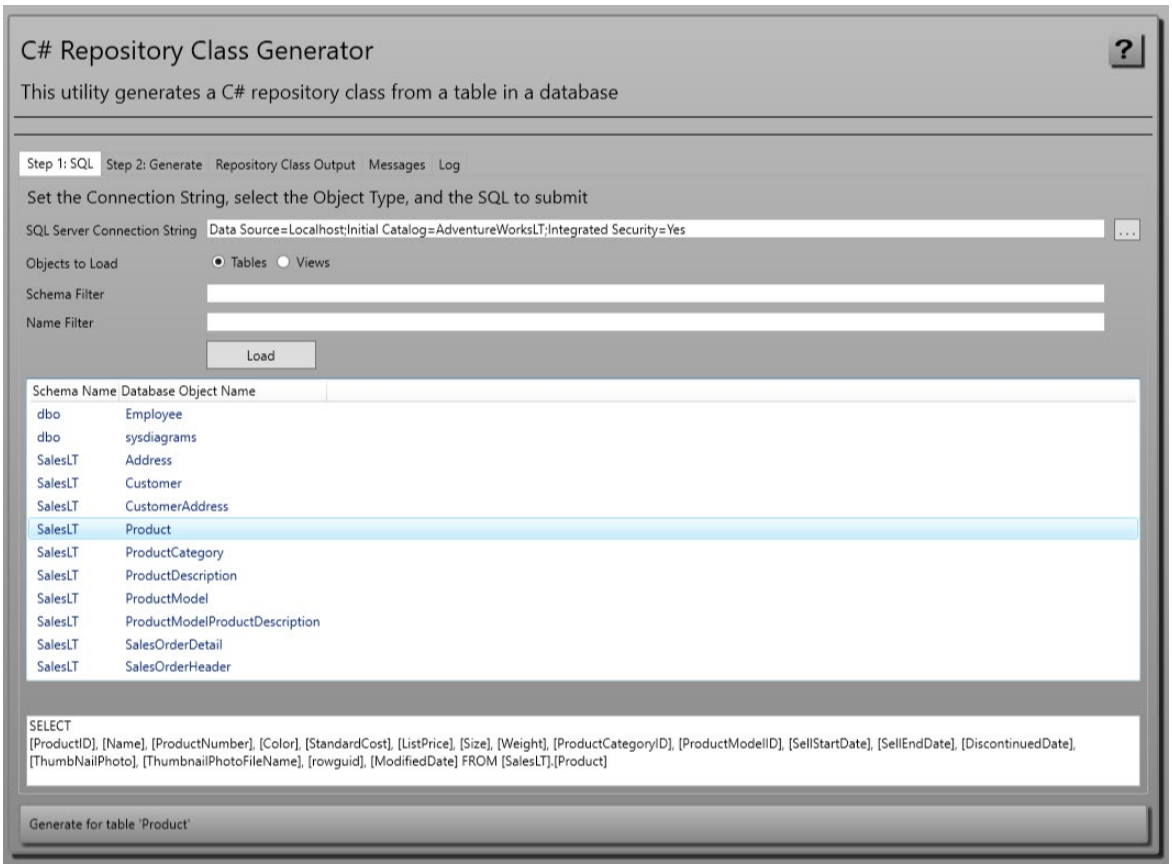


Figure 20: Step 1: C# Repository Class Generator SQL Tab

# Step 2: Generate

Click on **Step 3: Generate** (Figure 21) to fill in information on how you wish to generate the C# repository class. Fill in the Namespace to use, the Entity Class Name, the Repository Class Name and if you wish to include the #nullable disable statement at the top of the file the file. name. and the output folder. Check the **Prompt to Overwrite?** check box if you want to be prompted before overwriting a previously written file. Click the **Generate File** button to start the generation process.

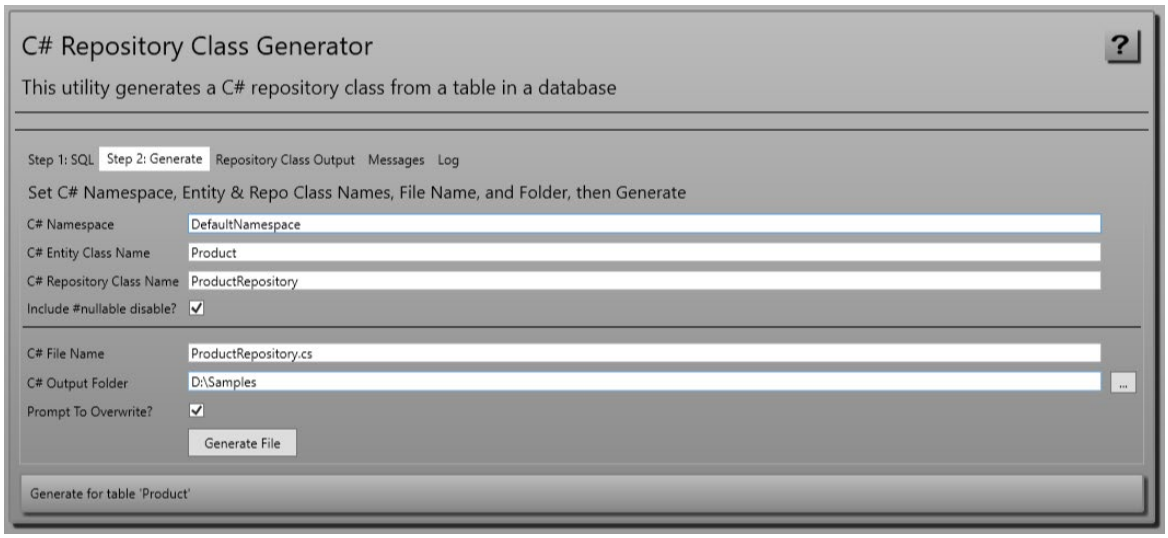


Figure 21: Step 2: C# Repository Class Generator Generate Tab

# Step 3: Output

After you click on the **Generate File** button, you will be presented with the screen shown in Figure 22. This screen tells you where the C# repository file is located and allows you to copy the entity class to the clipboard.

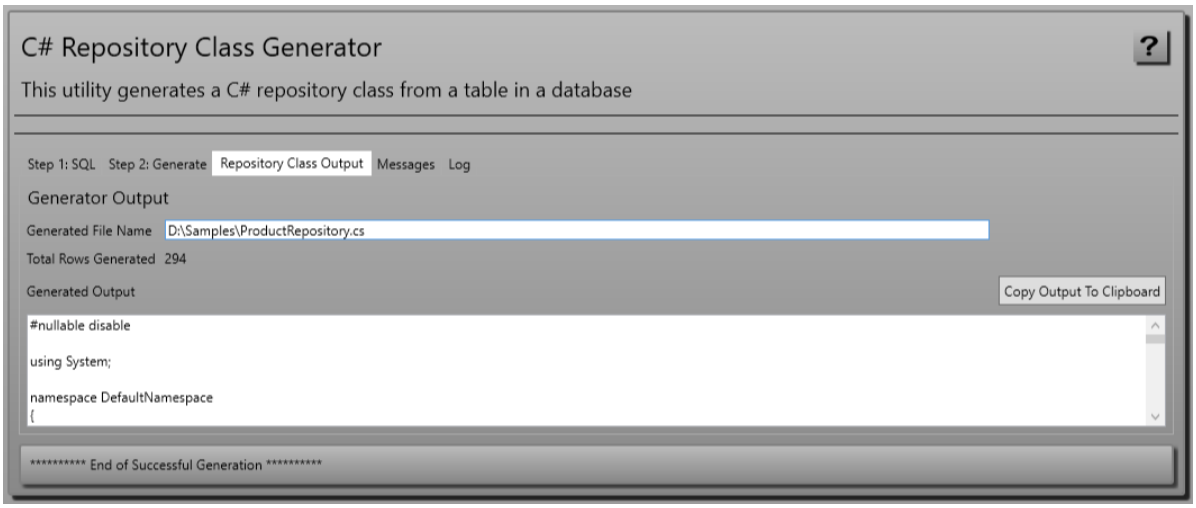


Figure 22: Step 3: C# Repository Class Generator Output tab

# Summary

The PDSC Developer Utilities will help increase your productivity while developing your applications. We hope you enjoy using this product.