PDSC Developer Utilities (8)

The PDSC Developer Utilities (8) as shown in 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 and .NET are great development environments for creating applications quickly. However, they tend to leave a lot of miscellaneous folders and files all over your hard drive. This utility recycles these folder and files to free up hard drive space.

Utility	Description
Project Cleaner	This tool goes through Visual Studio or VS Code project folders and recycles several folders that are not needed and can be regenerated automatically next time you build your application. You can optionally have it look in .SLN, VBProj, CSProj files and eliminate any references to source control. It can also remove any read-only attributes from the files. This utility is configurable so you can choose what folders and files you wish to recycle.
Property Generator	This utility generates C# or Visual Basic property statements. There are several templates (like the snippets in the Visual Studio editor) from which you can choose. You can also create your own templates to generate any type of property you want.
JSON Generator	This utility allows you to choose a table or view and generates a JSON file of the data.
XML Generator	This utility allows you to choose a table or view and generates an XML file of the data. Optionally, an XSD file of the schema of the table or view can also be generated.
C# Data Repo Generator	This utility generates a repository class that returns hard-coded data that you select from a table in one of your database tables. 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.
Code Generator	This utility generates Entity, Repository, View Model and Search classes from a table. It can also generate a complete set of CRUD .NET MAUI pages and MVC Web API controllers.
SQL Server Schema Compare	This utility compares two SQL Server databases to determine any tables, constraints, stored procedures, views, etc. that are missing between the two databases.

Table 1. List of PDSC Developer Utilities

Computer Cleaner

Visual Studio, Visual Studio Code, the .NET Framework, and .NET Core are great development environments for creating applications quickly. However, they sometimes 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, the .NET Framework, and .NET Core you have installed on your machine.

Disclaimer Tab

When you first come into the Computer Cleaner utility, a disclaimer tab (Figure 2) is displayed. We provide you with the warning that this tool is going to recycle files on your computer into the Recycle Bin. It puts them into the Recycle Bin so you can retrieve them if necessary. As there is always the potential for things to go wrong as Microsoft changes their operating systems frequently, it is good to be able to recover these recycled files. Please note that PDSC does not take any responsibility for the use of this tool on your machine.

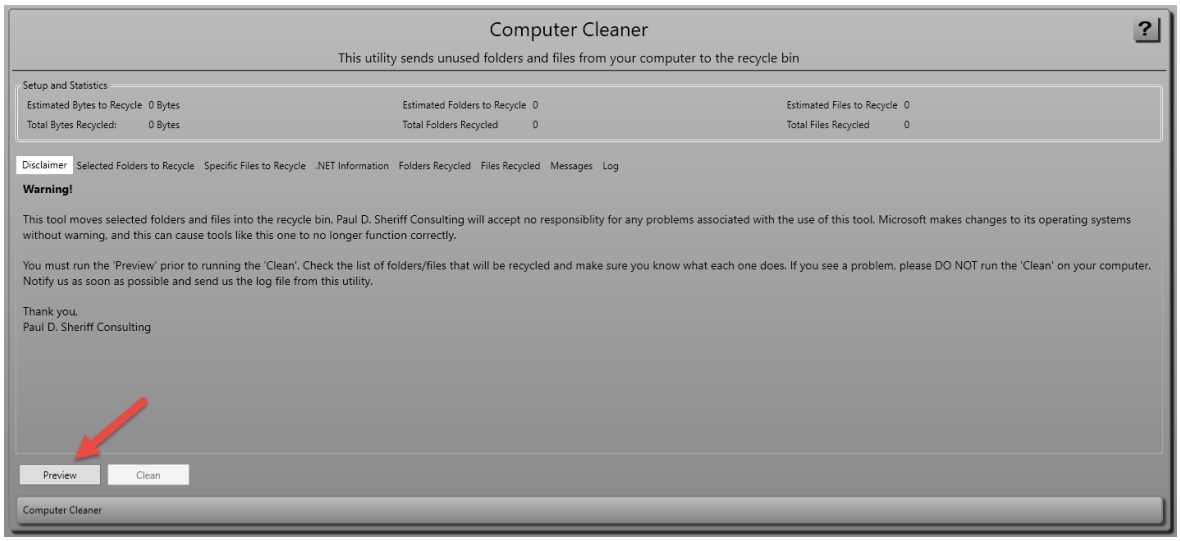

Figure 2: Disclaimer screen

The first thing you need to do is to click on the **Preview** button in the lower left-hand corner. This will provide you with a preview of the folders and/or files that are going to be recycled. The list of folders that are going to be recycled is contained in the **ComputerCleaner-FoldersToRecycle.xml** file located in the **PDSCDeveloperUtilities** folder in the **My Documents** folder on your computer.

> **NOTE**: Clicking on the Preview button can take a few minutes depending on how many folders and files are on your hard drive.

Selected Folders to Recycle Tab

After clicking on the **Preview** button, the list of folders that will be recycled (Figure 3). Be sure to review this list carefully. You may unselect any folders that you do not wish to recycle by unchecking the check box under the "Recycle?" Column next to the folder you don't want to recycle.

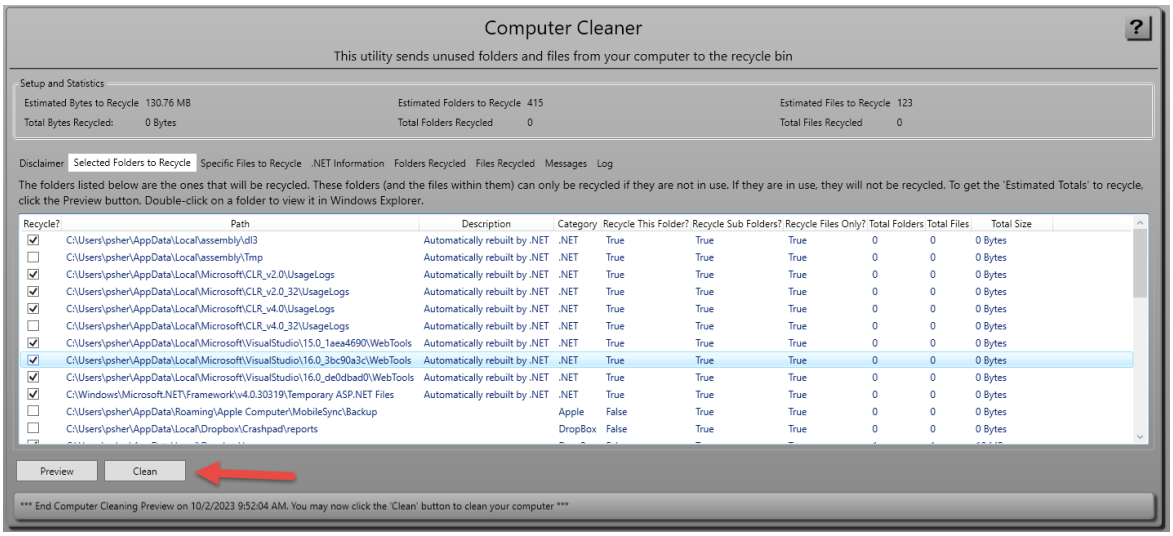

Figure 3: A list of the specified folders to recycle

Column	Description
Recyle?	Check to recycle this folder and/or the files within this folder.
Path	The actual path to the folder/files to recycle
Description	A description of the folder.
Category	What type of files, or the application, that created the files in this folder.
Recyle this Folder?	If set to true, this folder and all subfolders and files within it will be deleted. If the Description field reads "Automatically rebuilt by …" then this folder is safe to have deleted.
Recycle SubFolders?	If set to true, then any subfolders within this folder will be deleted.
Recycle Files Only?	If set to true, then any files within this folder and subfolders will be deleted.
Total Folders	After clicking the Preview button, this column displays the total number of folders found within this folder.

Total Files	After clicking the Preview button, this column displays the total number of files found within this folder.
Total Size	After clicking the Preview button, this column displays the total number of bytes of all files/folder found within this folder.

> **NOTE**: The count of folders and files, and the total bytes, is just an estimate of what could potentially be recycled. If the folder/file is in use, then it can't be recycled.

Selected Files to Recycle Tab

On this tab (Figure 4) is a list of specific files to recycle. The list of files that are going to be recycled is contained in the file **[MyDocuments]\PDSCDeveloperUtilities\Xml\ComputerCleaner-FilesToRecycle.xml**. You can add as many files to this XML file as you wish.

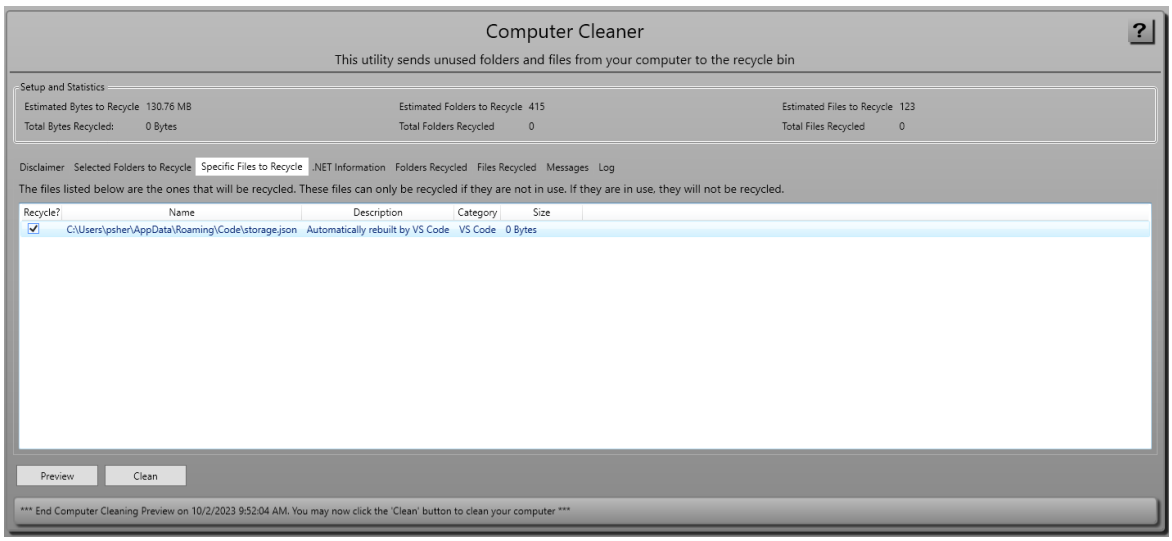

Figure 4: You can add additional files to recyle

.NET Information Tab

On this tab (Figure 5) is a list of .NET Framework and .NET Core versions located on your computer. The list of Visual Studio versions is also listed here.

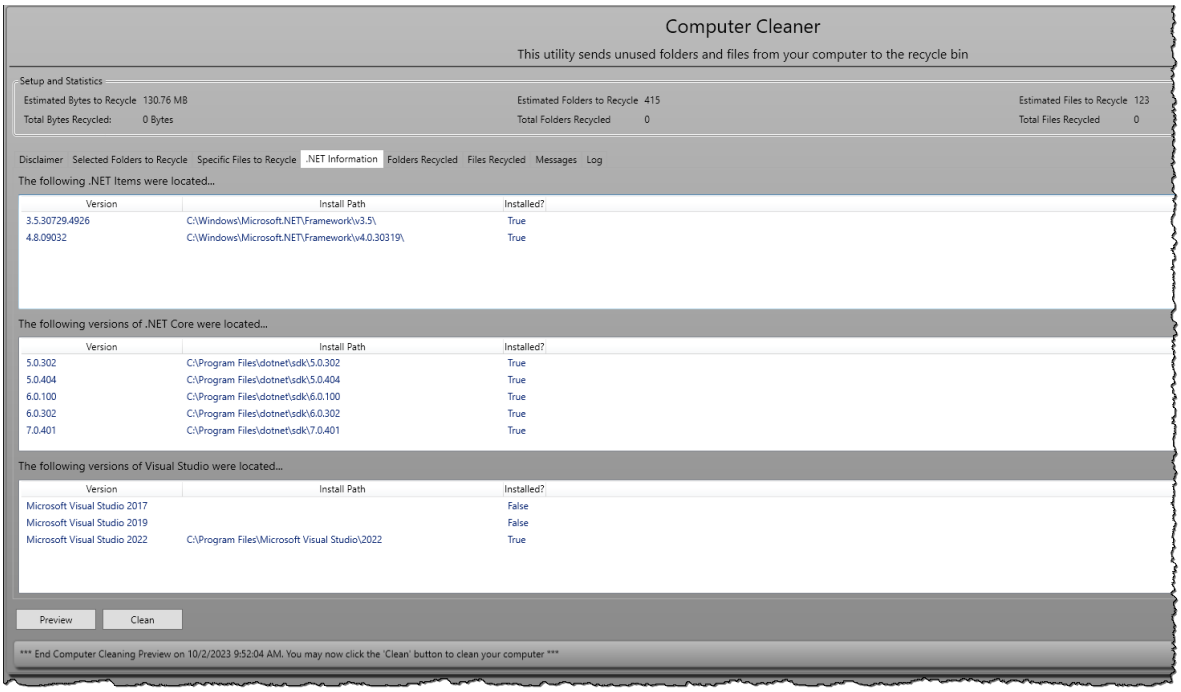

Figure 5: The list of .NET Frameworks, .NET Core and Visual Studio versions on your computer

Clean

Once you are satisfied with the list of folders and files to recycle from your hard-drive, click on the **Clean** button. This process can take several minutes depending on how many folders and files are on your hard drive. After this process is complete, the complete list of folders and files recycled is listed in the **Folders Recycled** and **Files Recycled** tabs.

You can view the list of everything that happened on the **Messages** tab, and all these messages are written into a log file that is in your [**My Documents]\PDSCDeveloperUtilities\Log** folder.

It is perfectly normal to have some Error Messages display as well, as some folders/files may be in use and not able to be accessed. Or because of security constraints, they also may not be able to be accessed.

NOTE:	If after cleaning, something does not work correctly, go to your Recycle Bin and restore the folders/files that were recycled during this cleaning process.

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 6.

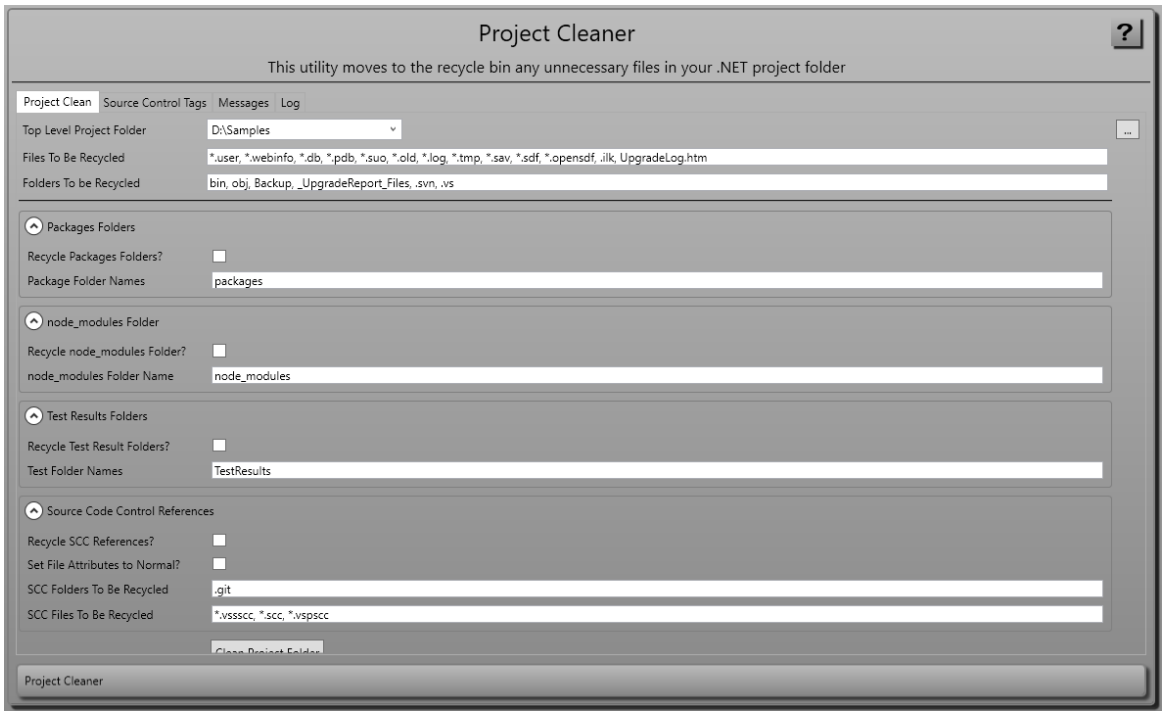

Figure 6: Clean up files using the Project Cleaner utility

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 node_modules Folder?	Check to remove any node_modules folders.
node_modules Folder Name	Fill in the names of the node_modules 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 2: 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.

Property Generator

Visual Studio has code snippets that will let you create properties (Figure 7). 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 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 are also able to add your own templates to control how you generate the properties.

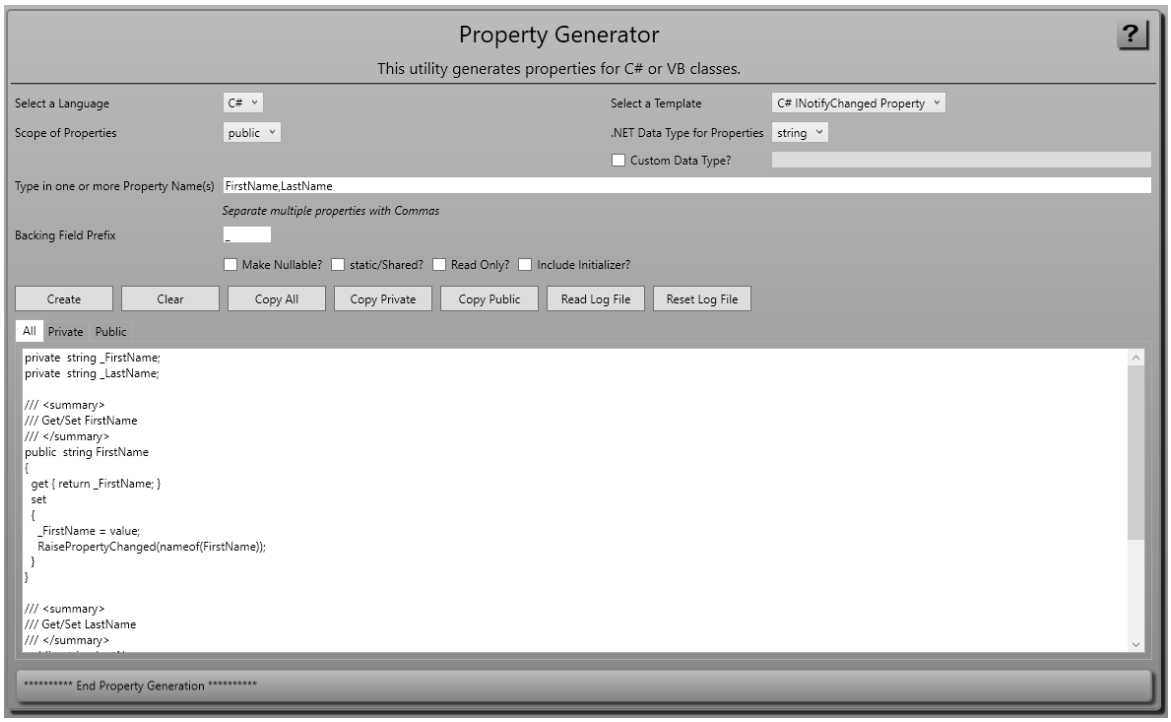

Figure 7: Property generator helps you create properties in many different styles

Adding Your Own Templates

Under the **[My Documents]\PDSCDeveloperUtilities\Xml** folder is a file named **PropertyGen-Templates.xml**. This file contains the list of template files you can use to generate properties. There is also a folder named **\Templates\PropertyGenerator** (Figure 8) in which there are several .txt files that hold the snippets for each of the types of properties that you can generate.

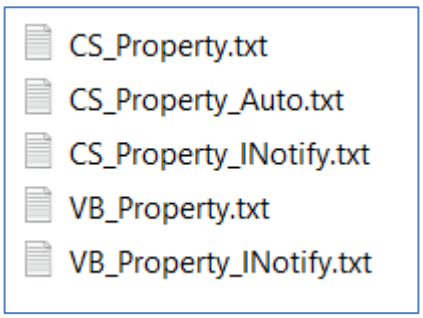

Figure 8: All the property snippets are just contained in .txt files

You will find one XML node in the **PropertyGen-Templates.xml** file for each .txt file located in the **\Templates\PropertyGenerator** folder. To add a new template, you 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 can create a 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:

```
/// <summary>
/// Get/Set <|PROPERTY_NAME|>
/// </summary>
<|SCOPE|> <|STATIC|> <|LANGUAGE_DATA_TYPE|> <|PROPERTY_NAME|>
{
   get { return <|PRIVATE_FIELD_PREFIX|><|PROPERTY_NAME|>; }
   {|READ_ONLY|}set { <|PRIVATE_FIELD_PREFIX|><|PROPERTY_NAME|> =
value; }{/|READ_ONLY|}
}
```

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

```
/// <summary>
/// Get/Set <|PROPERTY_NAME|>
/// </summary>
<|SCOPE|> <|STATIC|> <|LANGUAGE_DATA_TYPE|> <|PROPERTY_NAME|>
{
   get { return <|PRIVATE_FIELD_PREFIX|><|PROPERTY_NAME|>; }
   {|READ_ONLY|}set
   {
     <|PRIVATE_FIELD_PREFIX|><|PROPERTY_NAME|> = value;
     MyMethod("<|PUBLICNAME|>");
   }{/|READ_ONLY|}
}
```

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 **PropertyGen-Template.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 for the property generator will 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 a few tokens that are recognized by our property generator that are different from the Code Generator tokens. Table 3 contains the list of the tokens that you can use in your templates.

Token	Description
{|READ_ONLY|} and {|/READ_ONLY|}	Wrap these tokens around your "set" property to remove the "set" if you choose "Read only" in the Property Genreator tool.
<|READ_ONLY|>	Used for Visual Basic to insert "ReadOnly" into a property name
<|SCOPE|>	Returns the scope specified in the property generator tool.
<|STATIC|>	Returns "static"
<|SHARED|>	Returns "Shared"
<|NO_PRIVATE_VARIABLES|>	Do not generate the private variables

Table 3. 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 files are located in the **[My Documents]\PDSCDeveloperUtilities\Xml** folder under the location where you installed the Developer Utilities.

Xml File name	Description
LanguageDataTypes	A list of data types for C# and Visual Basic.
PropertyGen-DotNetLanguages	The list of .NET languages.
PropertyGen-LanguageScope	A list of scopes for C# and Visual Basic.
PropertyGen-Templates.xml	The list of templates that can be generated

Table 4. List of XML files for the Property Generator

JSON Generator

JSON files are very handy for a lot of things. The PDSC JSON Generator utility builds a JSON file from the data within 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 9). You can optionally expand the "What to Load…" area and 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 the Class Name (Singular), Class Name (Plural), Description (Singular), and Description (Plural) in the Grid for the class to generate. You can also modify the SQL statement prior to moving to step 2 if you wish to generate a different set of columns, add a SELECT TOP n to generate a limited set of rows, or use aliases for your columns to generate different names for your element/attribute names.

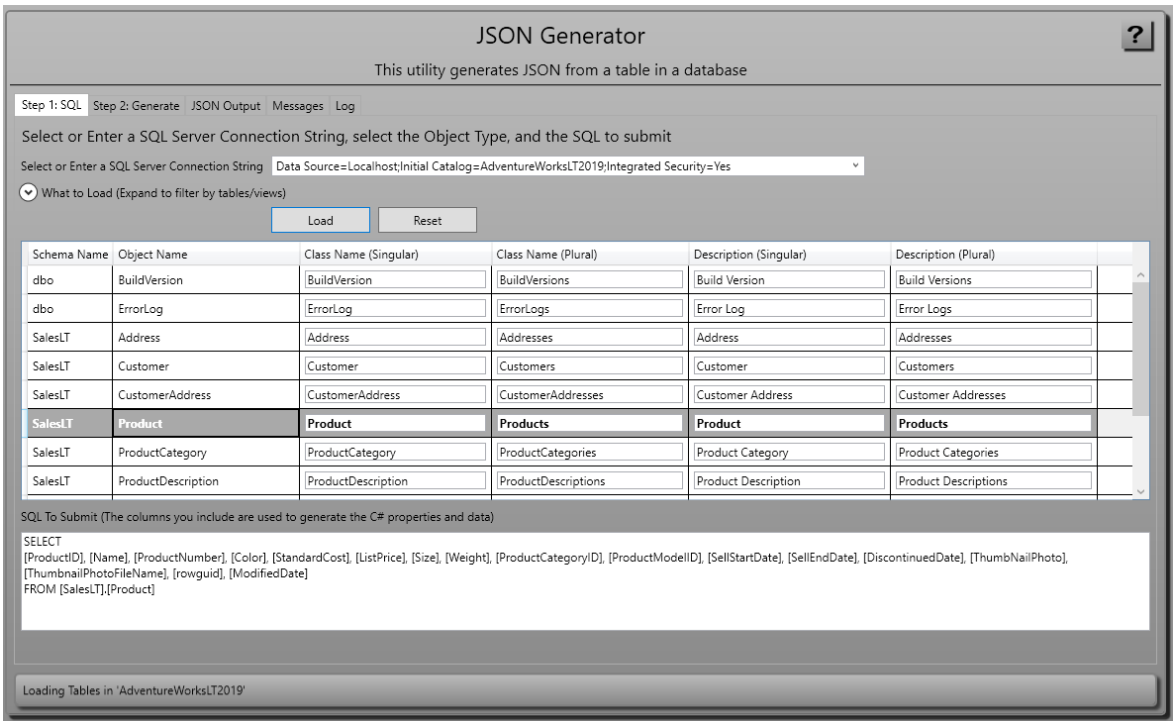

Figure 9: Step 1: JSON Generator SQL Tab

Step 2: Generate

Click on **Step 2: Generate** (Figure 10) to fill in information on how you wish to generate the JSON. You can either write to a file or not. If you write to a file, 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.

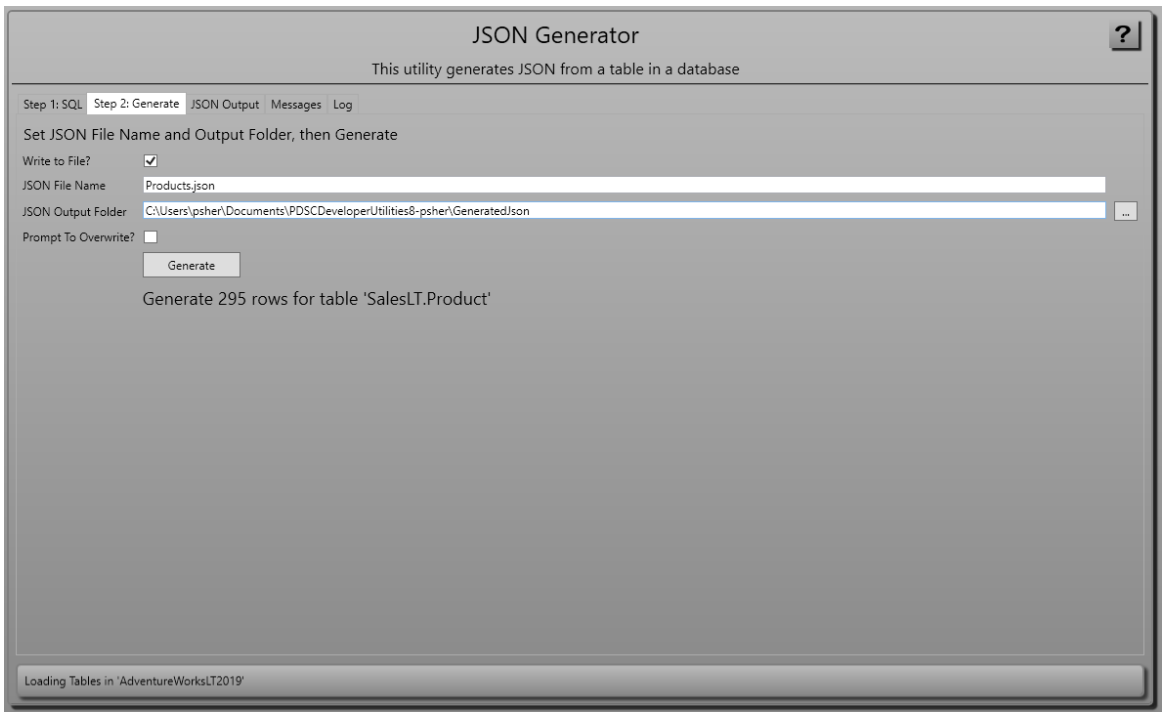

Figure 10: Step 2: JSON Generator Generate Tab

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

View the JSON Output

After you click on the **Generate** (Figure 11) button, you are presented with the screen shown in Figure 11. This screen shows you where the generated .json file is located and the JSON output.

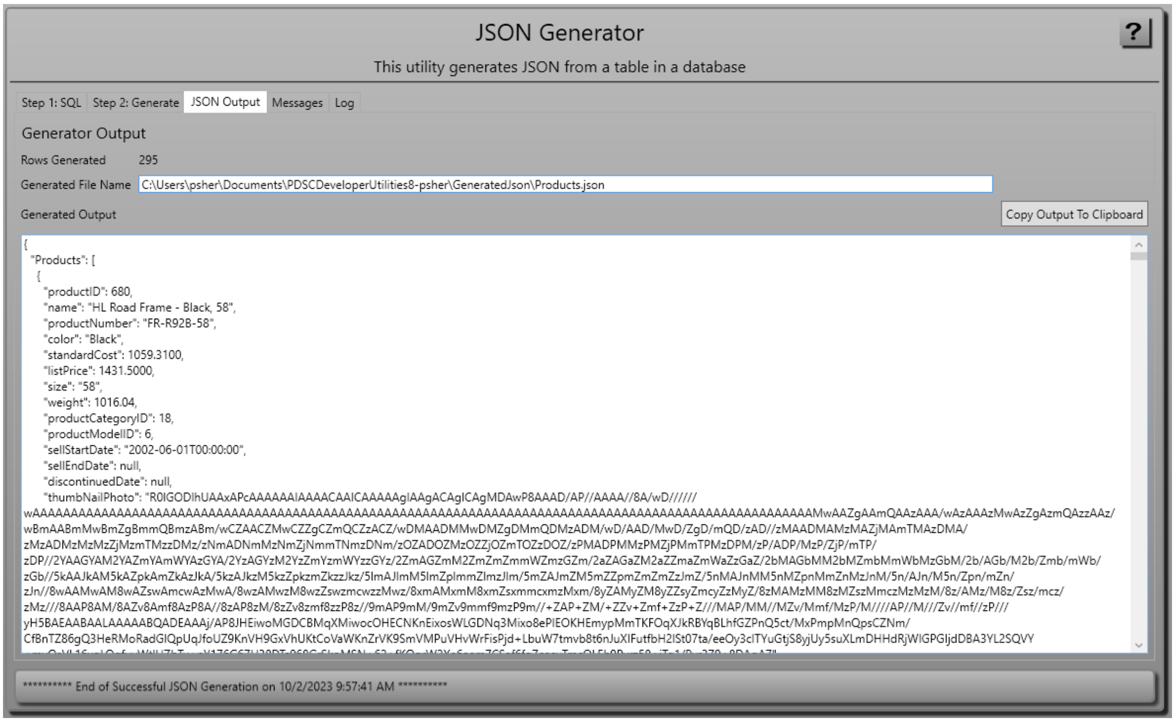

Figure 11: JSON Generator Output tab

XML Generator

XML files are very handy for a lot of things. The PDSC XML Generator utility builds XML and XSD files from any table or view in your SQL Server 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 12). You can optionally expand the "What to Load…" area and 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 the Class Name (Singular), Class Name (Plural), Description (Singular), and Description (Plural) in the Grid for the class to generate. You can also modify the SQL statement prior to moving to step 2 if you wish to generate a different set of columns, add a SELECT TOP n to generate a limited set of rows, or use aliases for your columns to generate different names for your element/attribute names.

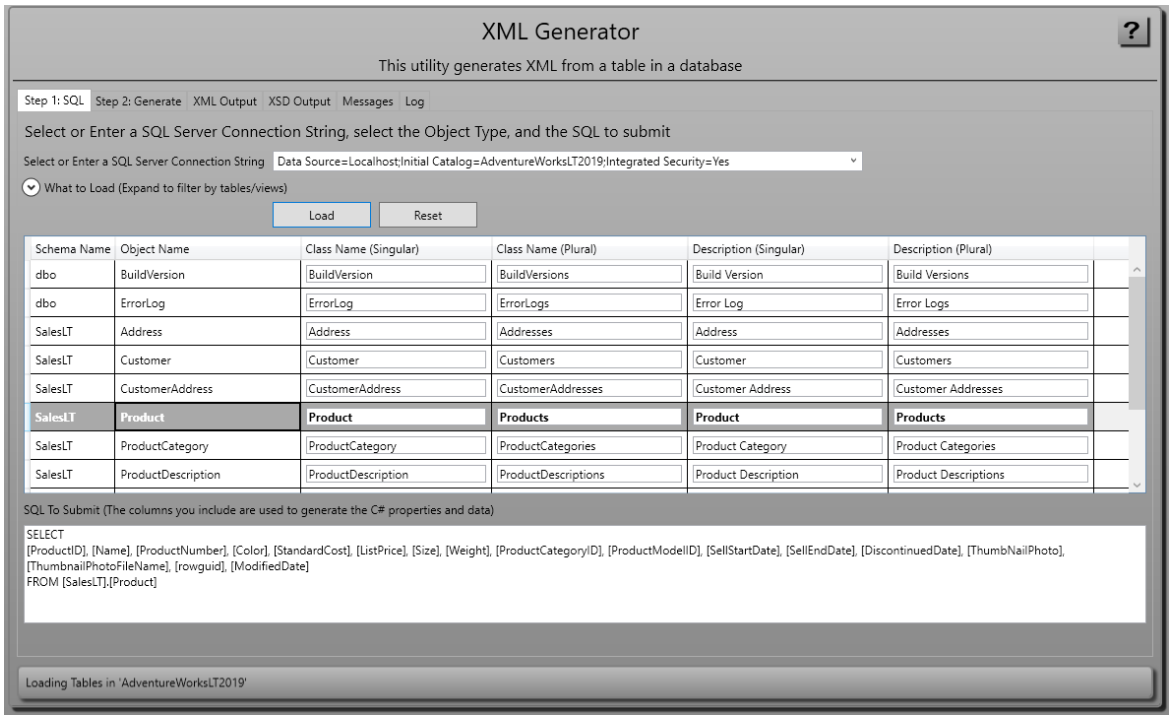

Figure 12: Step 1: XML Generator SQL Tab

Step 2: Generate

Click on **Step 2: Generate** (Figure 13) 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.

You can either write to a file or not. 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(s)** button to start the generation process.

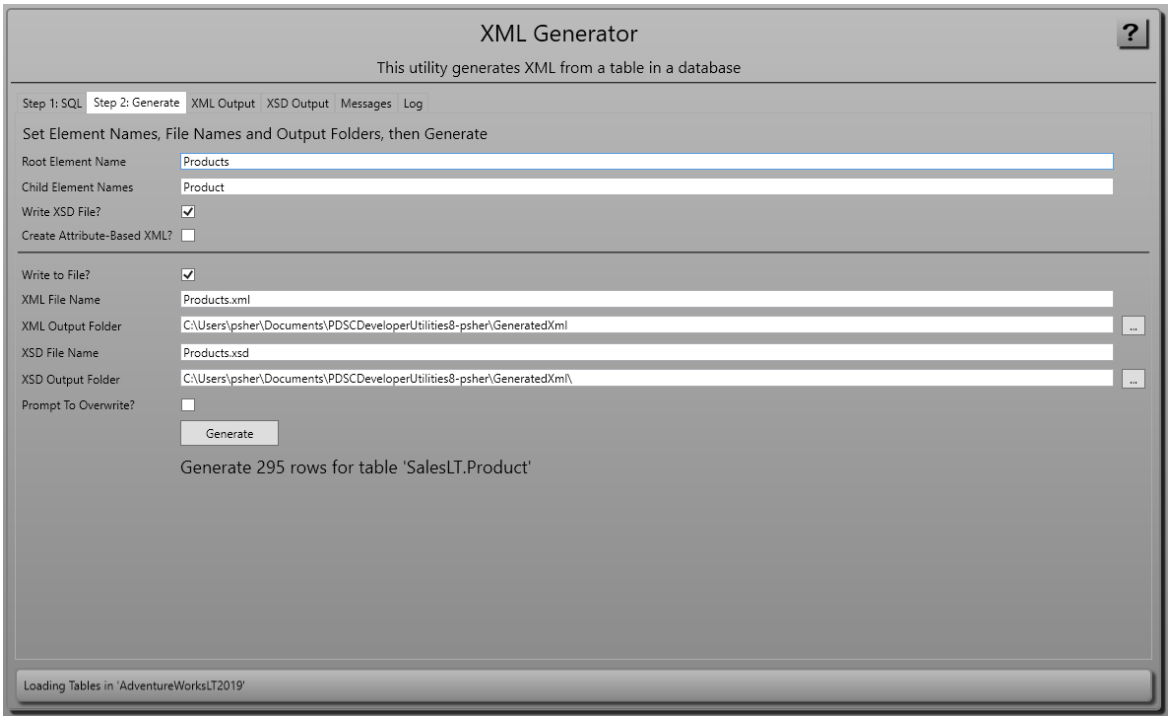

Figure 13: Step 2: XML Generator Generate Tab

XML Output

After you click on the **Generates** button, you will be presented with the screen shown in Figure 14.

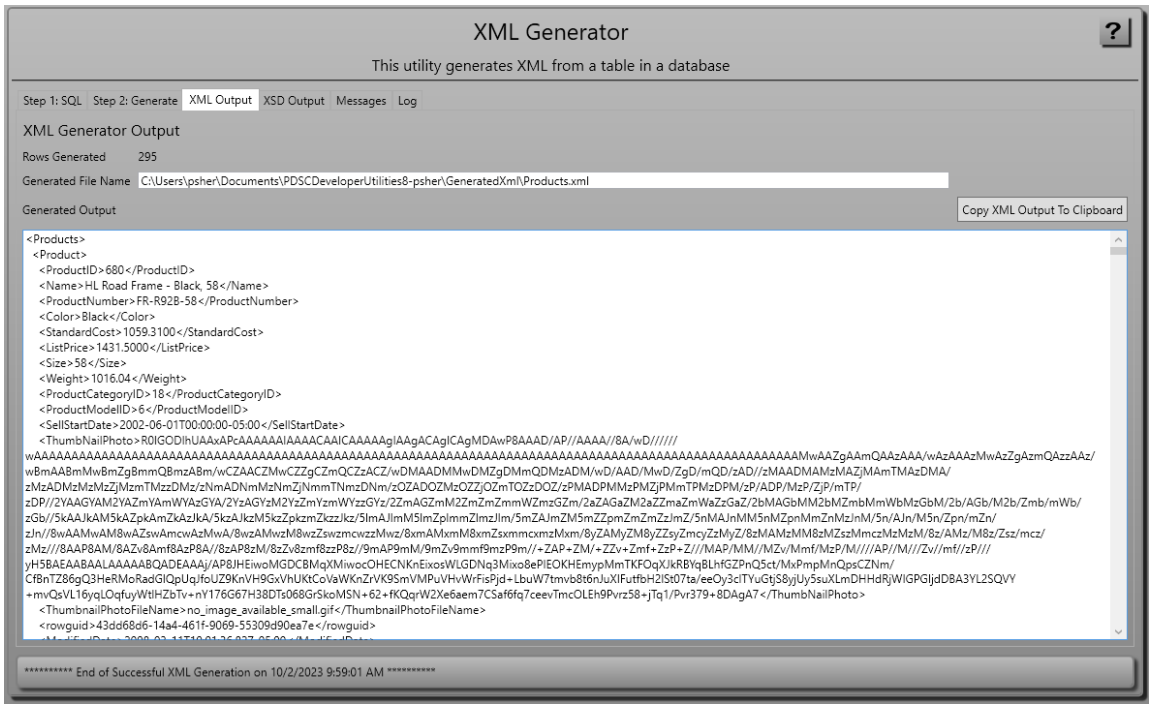

Figure 14: XML Output Tab

XSD Output

If you generated an XSD, you can view the XSD on the screen show in Figure 15.

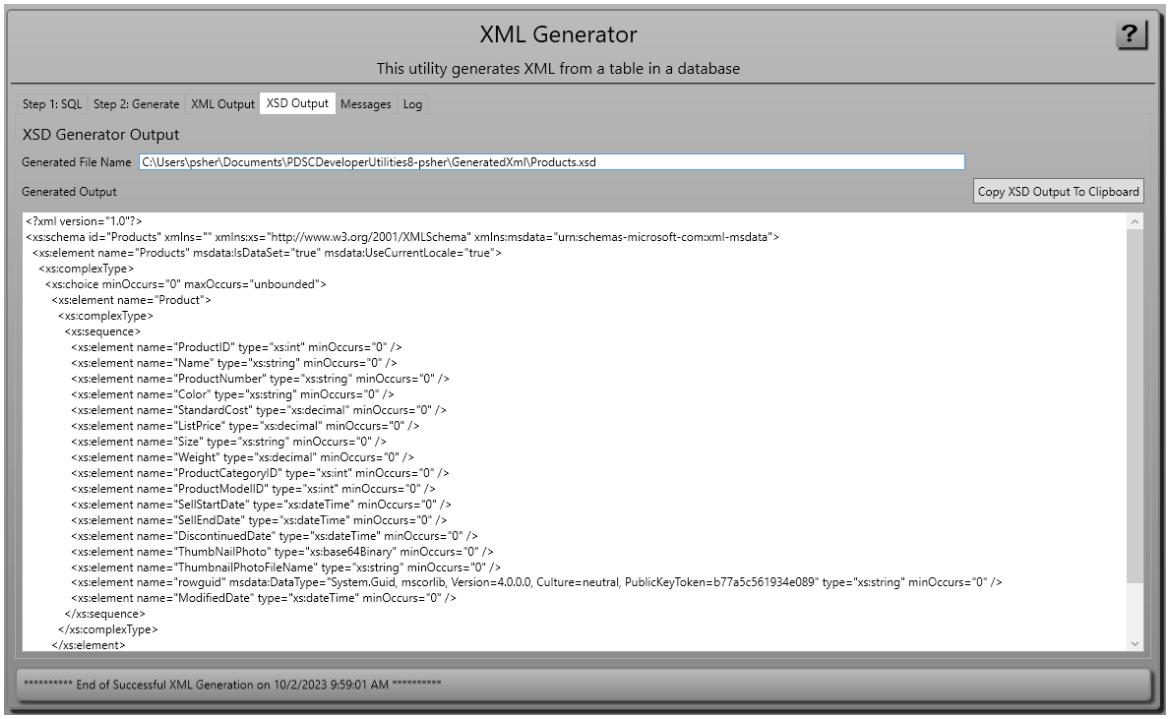

Figure 15: XSD Output tab

C# Data 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 16). 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.

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 the Class Name (Singular), Class Name (Plural), Description (Singular), and Description (Plural) in the Grid for the class to generate. You can also modify the SQL statement prior to moving to step 2 if you wish to generate a different set of columns, add a SELECT TOP n to generate a limited set of rows, or use aliases for your columns to generate different names for your element/attribute names.

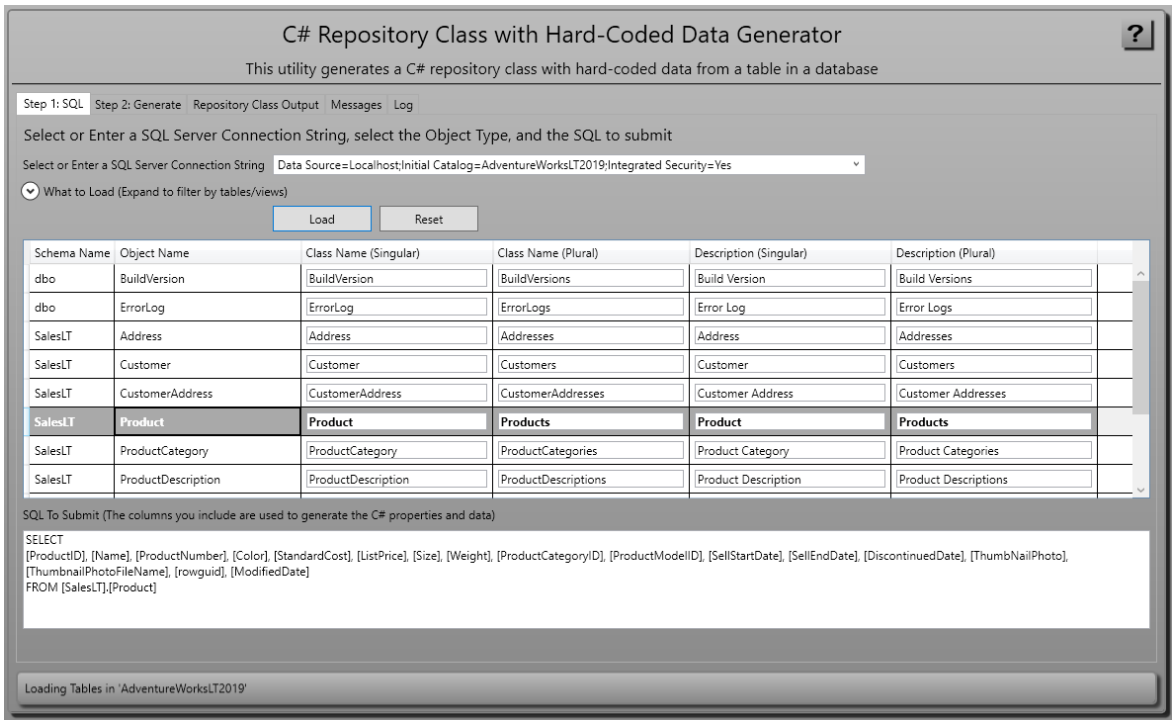

Figure 16: Step 1: C# Repository Class Generator SQL Tab

Step 2: Generate

Click on **Step 2: Generate** (Figure 17) 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. You can either write to a file or not. Fill in 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** button to start the generation process.

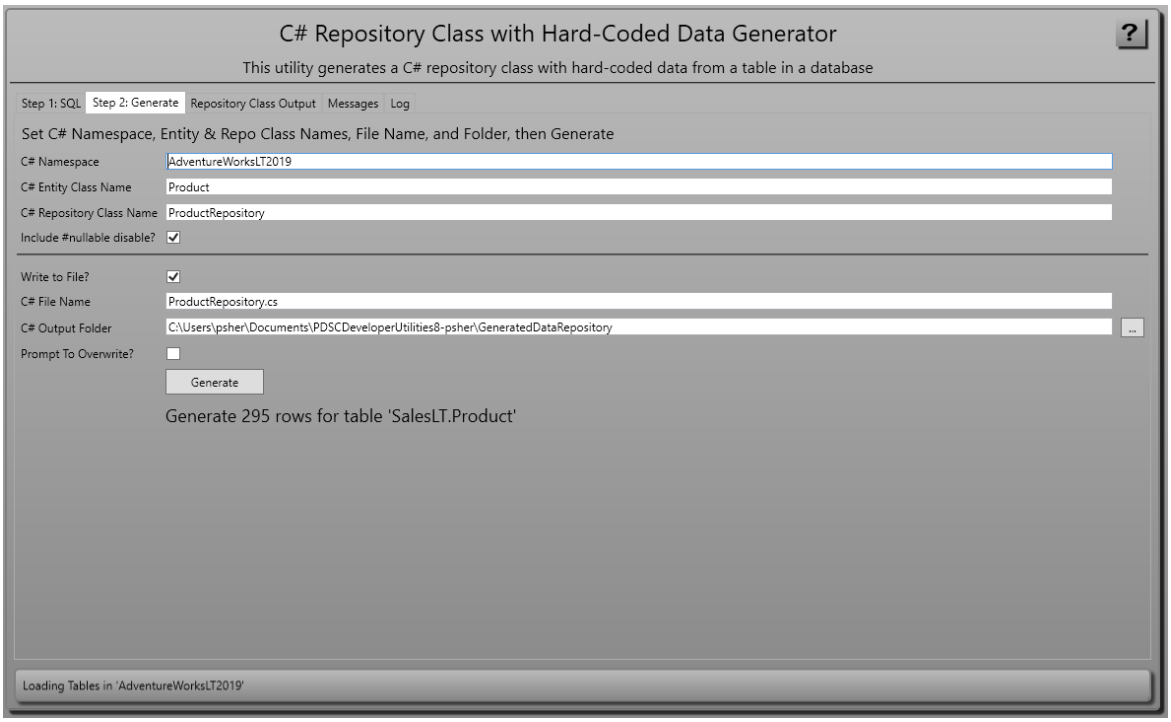

Figure 17: Step 2: C# Repository Class Generator Generate Tab

Repository Class Output

After you click on the **Generate** button, you will be presented with the screen shown in Figure 18. This screen tells you where the C# repository file is located and allows you to copy the entity class to the clipboard.

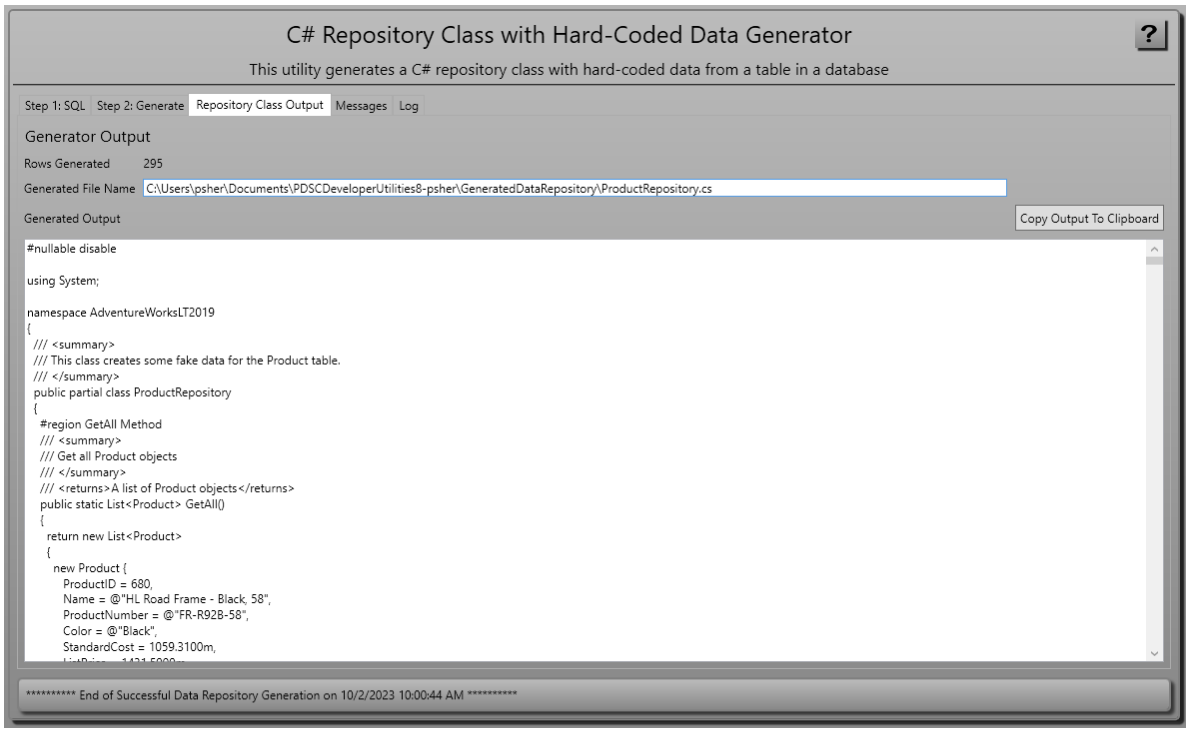

Figure 18: Repository Class Output tab

Data Repository Generator Tokens

In the .txt files that represent the code to generate for the Hard Coded Data Repository class you find a few tokens in the format <|TOKEN_NAME|>. There are a few tokens that are recognized by our data repository generator that are different from the Code Generator tokens. Table 5 contains the list of the tokens that you can use in your templates.

Token	Description
<|HARD_CODED_DATA|>	Returns the generated values from all rows and columns and inserts them the generated output
<|UPDATE_COLUMNS|>	Returns the columns to update
<|CLASS_NAME_REPOSITORY|>	Returns the name of the repository class as specified in the Repository Generator tool

Table 5. List of Tokens in Data Repository generator

Code Generator

The PDSC Code Generator generates a set of classes and pages for one or more tables in your SQL Server database. For example, with the supplied templates, you

can generate Entity, Repository, Search and View Model classes for accessing data in a SQL Server database. Other technologies are added periodically to this code generator.

Step 1: Select Template(s)

On the first tab (Figure 19) from the Select a Template Group drop-down, choose a template group you wish to generate. In the Grid, select one or more of the templates to generate from within that group. Most typically you will leave them all checked.

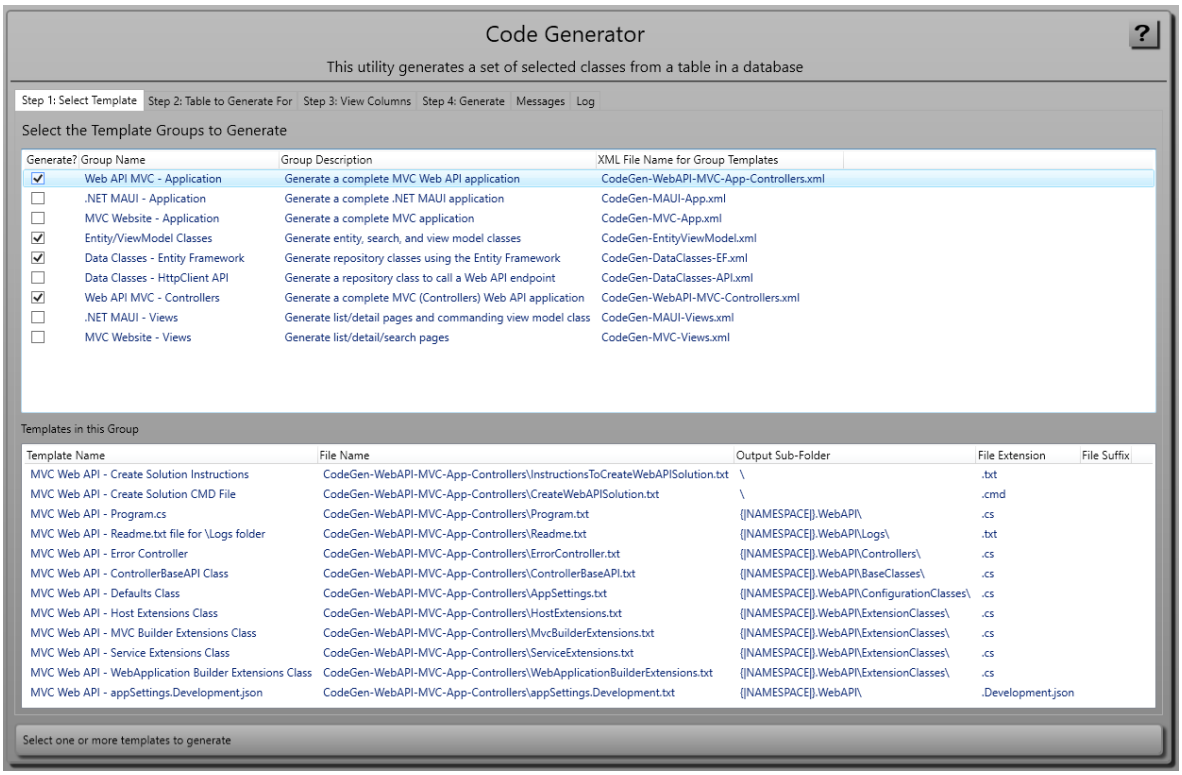

Figure 19: Step 1: Code Generator Select Template Tab

The following table explains what you can currently generate with the Code Generator.

Template	Description
Web API MVC – Application	Generates all the files necessary to build a complete MVC Web API application using controllers. A .cmd file is also created to make it easy to create the Web API application once generated.
.NET MAUI - Application	Generates all the files necessary to build a complete .NET MAUI application. A .cmd file is also created to make it easy to create the .NET MAUI application once generated.

MVC Website - Application	Generates an MVC controller class, a home page, Program.cs, and an appsettings.Development.json file to replace the default ones generated by the **dotnet new mvc** command.
Entity/ViewModel Classes	Generates an Entity class with properties that matches each column in each selected table. Generates a Search class. Generates a ViewModel class that accepts an IRepository<TEntity, TSEarch>object.
Data Classes - Entity Framework	Generates a Repository class that uses the Entity Framework to communicate with the database. Generates a DbContext class to which you can add additional generated code for more tables.
Data Classes - HttpClient API	Generates a Repository class that uses the .NET HttpClient to make Web API calls to a web service with API endpoints to communicate with the database.
Web API MVC - Controllers	Generates all the CRUD controllers with endpoints for each table selected.
.NET MAUI - Views	Generates a list and detail page for use in the .NET MAUI application. Generates the commanding View Model.
MVC Website - Views	Generates partial pages for listing, searching, adding, editing, and deleting data from each table selected.

Table 6. List of Standard Template in the Code Generator

Step 2: Table(s) to Generate For

Either type in or select from the drop-down a connection string that will connect you to your database (Figure 20). You can optionally expand the "What to Load…" area and 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 you can modify the Class Name (Singular), Class Name (Plural), Description (Singular), and Description (Plural) in the Grid for the class to generate.

You are allowed to generate one or more tables by making sure the Generate check box is checked next to the table for which you wish to generate code. When a table in this grid is highlighted, you can view the columns for that table in the **Step 3: View Columns** tab.

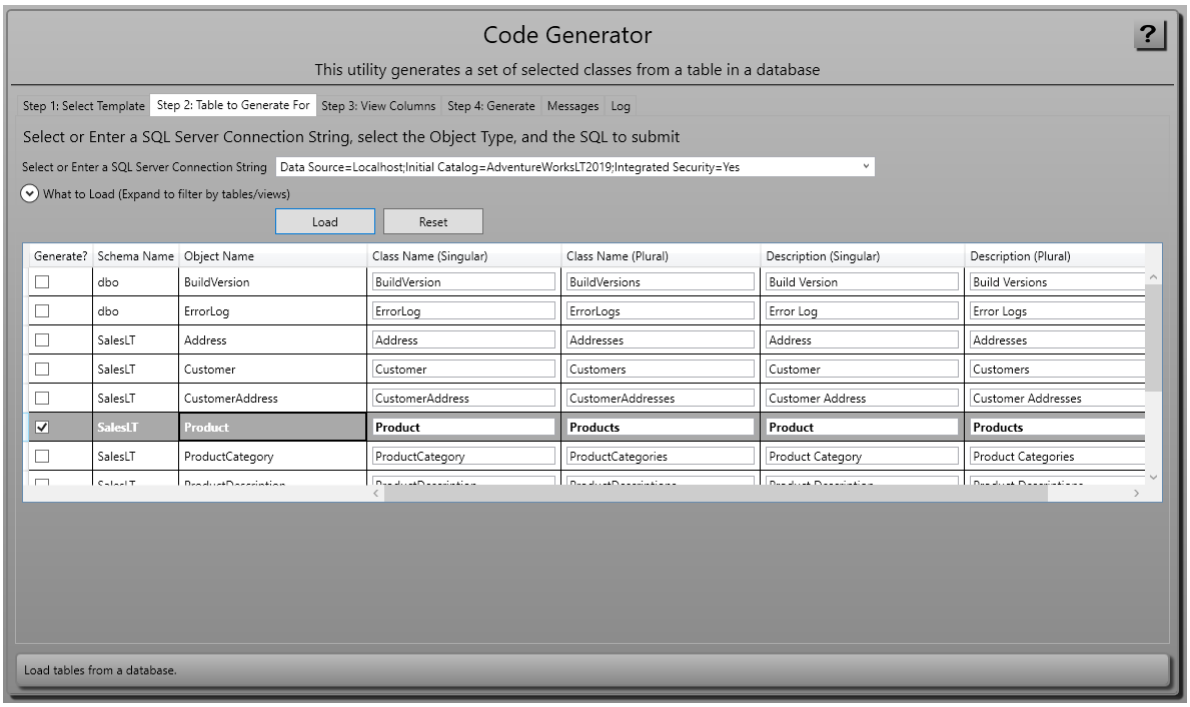

Figure 20: Step 2: Code Generator Table to Generate For Tab

Step 3: View Columns

On this tab (Figure 21) you can view all the columns for a selected table on the previous tab. If you have not clicked on a table, this list will be blank. For the selected table you may check or uncheck the appropriate usage of each of the columns. You may also modify the Property Name, Label and C# data type. If you change any of these values, they are saved after the generation occurs and will be available the next time you generate.

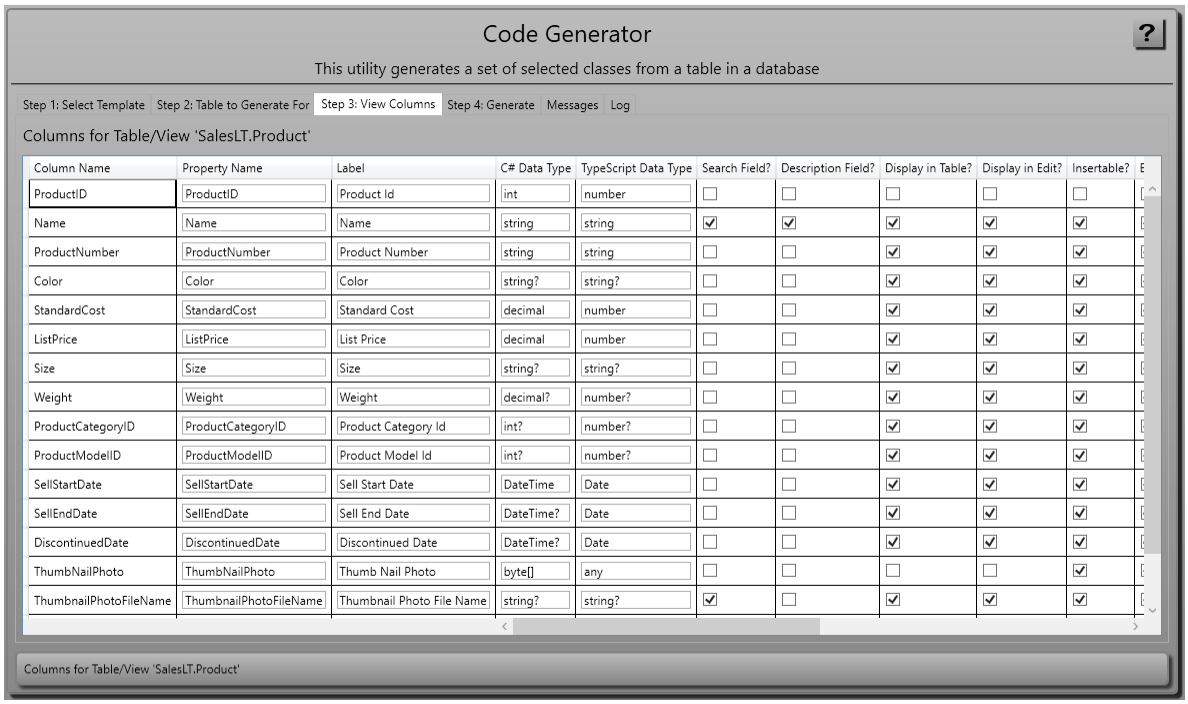

Figure 21: Step 3: Code Generator View Columns Tab

Step 4: Generate

Click on **Step 4: Generate** (Figure 22) to fill in information on how you wish to generate the classes as shown in Table 7.

Field	Description
Application Name	The application name
Namespace	The namespace to use in your generated classes
DbContext Class Name	The name of the Entity Framework DbContext class to generate
Generation Output Folder	The location to which you want all the files to be generated
Delete Output Folder Before Generating?	If you previously generated code, and you wish to delete that old code prior to generating new code, check this option
Open Windows Explorer After Generating?	If you want Windows Explorer to open to the Generation Output Folder after generating, check this option
Backing Field Prefix	The prefix to use before all private fields for your properties.
Standard Fields in All Tables?	Only check this option if all tables selected have the same set of standard fields as listed in the CodeGen-StandardFields.xml file.

Include Property Comments?	Check this if you wish property comments to be generated
Include Data Annotations?	Check this is you wish to include data annotations on the properties in your Entity classes
Use Full Property Get/Set?	Check this if you wish to use a backing field and a full property get/set, otherwise, auto-properties are generated
Use PropertyChanged Event?	If you are generating code to be used with WPF or other XAML applications, check this to have a RaisePropertyChanged event called within each property set

Table 7: Code Generation Parameters

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

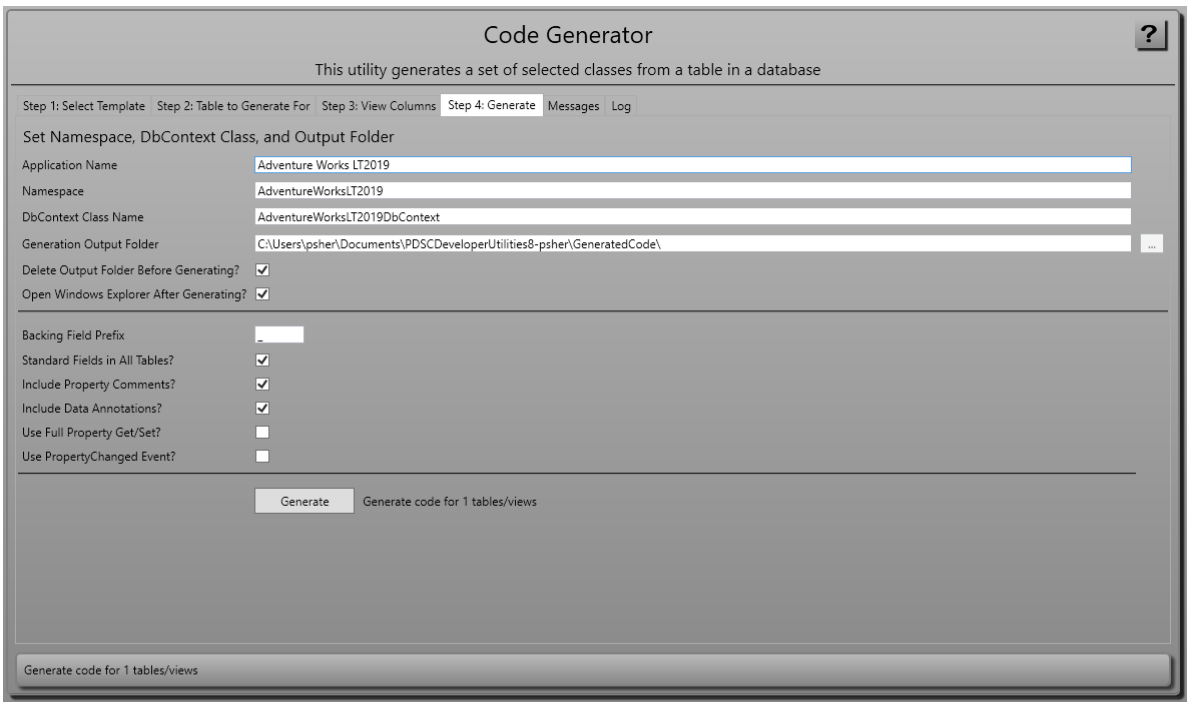

Figure 22: Step 4: Code Generator Generate Tab

After you have clicked on the Generate button a File Explorer window is opened to the folder in which you chose to generate the code. Depending on which template you generated from, refer to one of the next sections to learn how to add the generated code to a .NET application and see it in action.

What's in the _DoNotAddToProject Folder

In the **_DoNotAddToProject-Includes** folder (Figure 23) is where you will find some .txt files that have code you can manually add to some of your previously generated files in your project. For example, you will find a **DbContextDbSets.txt**

file that contains the public DbSet<T> property for any tables you have generated. Open this .txt file and copy the code in there and put it into your [NAMESPACE]DbContext.cs file in your application.

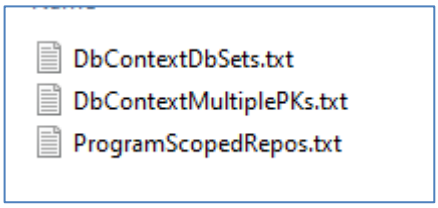

Figure 23: Examples of additional code to add to previously generated files

You will find a **DbContextMultiplePKs.txt**. If you have any tables that have more than one field for the primary key, you need to open this file and copy the code into your [NAMESPACE]DbContext.cs file.

You will find a **MauiProgramDI.txt** file. This file contains the code to add to the **MauiProgram.cs** file your newly generated repositories as a service to be used for dependency injection.

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 24) and a complete list of missing or changed objects will appear in the messages tabs at the bottom of the screen.

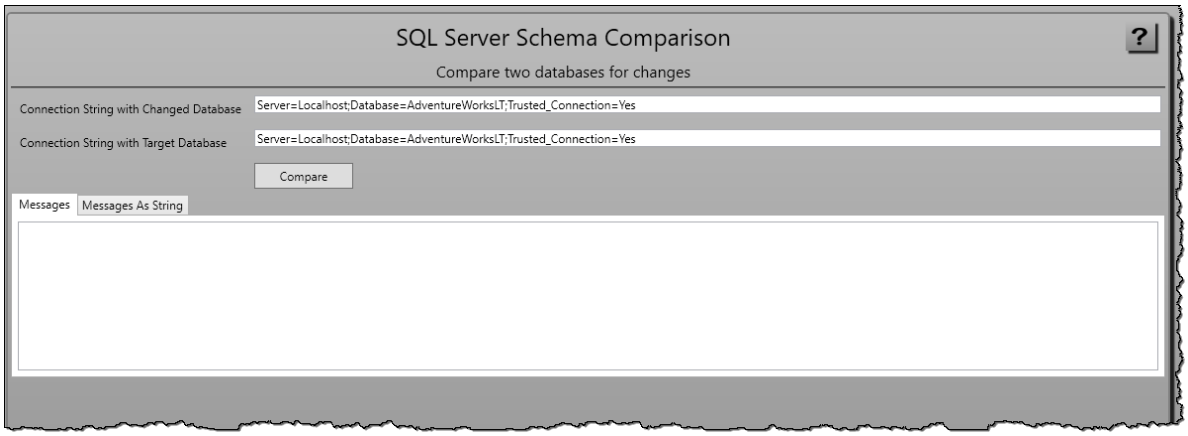

Figure 24: Get missing objects from one database to another via the SQL Compare Utility.

Generate a .NET MAUI Application

Generate a Web API Application (MVC Controllers)

Code Generator: Template Tokens

This section explains the various tokens and how they are used throughout the various template files.

Looping Tokens

The following tokens are used to loop through columns and tables.

Token	Description
{|FOR EACH COLUMN|}	Loop through all columns for the table
{|FOR EACH COLUMN:IsEditable|}	Loop through all columns marked for Editing
{|FOR EACH COLUMN:IsInsertable|}	Loop through all columns marked for Inserting
{|FOR EACH COLUMN:IsPrimaryKey|}	Loop through all primary key columns
{|FOR EACH COLUMN:IsNotPrimaryKey|}	Loop through all columns that are not primary keys
{|FOR EACH COLUMN:IsDescriptionField|}	Loop through all columns marked as Description
{|FOR EACH COLUMN:IsSearchField|}	Loop through all columns marked as Search fields
{|FOR EACH COLUMN:DisplayInTable|}	Loop through all columns marked for displaying in table
{|FOR EACH COLUMN:DisplayInEdit|}	Loop through all columns marked for displaying in edit page

{|FOR EACH COLUMN:NoStandardFields|}	Loop through all columns that are NOT standard fields
{|FOR EACH COLUMN:StandardFields|}	Loop through all columns that are standard fields
{|FOR EACH COLUMN: StandardFieldIsInsertable|}	Loop through all columns that are standard fields and are marked as insertable
{|FOR EACH COLUMN: StandardFieldIsEditable|}	Loop through all columns that are standard fields and are marked as editable
{|FOR EACH TABLE|}	Loop through all selected tables
{|END_LOOP|}	Each of the above must be terminated with this. Note, you can NOT nest {|FOR EACH |} tokens

Table 8: Looping Tokens

Remove Tokens

The following tokens can be used to remove blocks of code if a certain condition is not matched.

Token	Description
{|REMOVE_WHEN:NoStandardFields|}	Removes the block of code when there are no standard fields in the table
{|REMOVE_WHEN:IsAutoIncrement|}	Removes the block of code when the Primary key is and IDENTITY property
{|REMOVE_WHEN:IsNotAutoIncrement|}	Removes the block of code when the Primary key is NOT an IDENTITY property
{|REMOVE_WHEN:IsPrimaryKeyInteger|}	Removes the block of code when the Primary key is an Integer
{|REMOVE_WHEN:IsPrimaryKeyNotInteger|}	Removes the block of code when the Primary key is NOT an Integer
{|REMOVE_WHEN:IsPrimaryKeyString|}	Removes the block of code when the Primary key is a string
{|REMOVE_WHEN:IsPrimaryKeyNotString|}	Removes the block of code when the Primary key is NOT a string
{|REMOVE_WHEN:IsPrimaryKeyGuid|}	Removes the block of code when the Primary key is a unique identifier
{|REMOVE_WHEN:IsPrimaryKeyNotGuid|}	Removes the block of code when the Primary key is NOT a unique identifier
{|REMOVE_WHEN:IsColumnBoolean|}	Removes the block of code when the current column is a Boolean data type

{|REMOVE_WHEN:IsColumnNotBoolean|}	Removes the block of code when the current column is NOT a Boolean data type
{|REMOVE_WHEN:IsColumnDateTime|}	Removes the block of code when the current column is a date/time data type
{|REMOVE_WHEN:IsColumnNotDateTime|}	Removes the block of code when the current column is NOT a date/time data type
{|REMOVE_WHEN:IsColumnString|}	Removes the block of code when the current column is a string data type
{|REMOVE_WHEN:IsColumnNotString|}	Removes the block of code when the current column is NOT a string data type
{|REMOVE_WHEN:IsColumnUniqueIdentifier|}	Removes the block of code when the current column is a unique identifier (Guid) data type
{|REMOVE_WHEN: IsColumnNotUniqueIdentifier |}	Removes the block of code when the current column is NOT a unique identifier (Guid) data type
{|REMOVE_WHEN:OnlyOnePrimaryKey|}	Removes the block of code when there is only 1 primary key column
{|END_REMOVE|}	Each of the above must be terminated with this. **NOTE**: you can NOT nest {|REMOVE WHEN |} tokens **NOTE**: you can NOT place {|REMOVE WHEN |} blocks within any of the loop tokens

Table 9: Remove Tokens

Column Tokens

The following tokens are used as you are looping through the list of columns. All tokens are enclosed within <|TOKEN|>, and sometimes can be enclosed with {|TOKEN|}.

Token	Description
<|MAX_LENGTH|> and {|MAX_LENGTH|}	Returns the maximum length marked for a string column in SQL Server
<|PRECISION|> and {|PRECISION|}	Returns the numeric precision for a column in SQL Server
<|SCALE|> and {|SCALE|}	Returns the numeric scale for a column in SQL Server
<|DATETIME_PRECISION|> and {|DATETIME_PRECISION|}	Returns the datetime precision for a column in SQL Server
<|DATA_TYPE|> and {|DATA_TYPE}}	Returns the data type in SQL Server

<|COLUMN_NAME|> and {|COLUMN_NAME|}	Returns the column name in SQL Server
<|PRIMARY_KEY_FIELD|> and {|PRIMARY_KEY_FIELD|}	Returns the primary key column name in SQL Server
<|PRIMARY_KEY_FIELD_LABEL|>	Returns the label for the primary key column name
<|DESCRIPTION_FIELD|>	Returns the description property for the current column
<|DESCRIPTION_FIELD_LOWER_FIRST_LETTER|>	Returns the description property for the current column with the first letter as lower-case
<|LANGUAGE_DATA_TYPE|>	Returns the data type for the current column for the selected template language and it could have the symbol for a nullable data type if the column is marked as nullable in the database.
<|LANGUAGE_DATA_TYPE_NON_NULLABLE|>	Returns the data type for the current column for the selected template language without any nullable symbol
<|LANGUAGE_DATA_TYPE_NULLABLE|>	Returns the data type for the current column for the selected template language with the nullable symbol
<|NULLABLE_CHARACTER_IF_NULLABLE|>	Returns the nullable character for the selected template language unless the column is a string data type.
<|NULLABLE_CHARACTER|>	Returns the nullable character if the column is NOT a string data type.
<|HTML_INPUT_TYPE|>	Returns the text value to set on the <input type="INPUT TYPE". Valid values are based on the Is* properties of the column. IsDateTime="datetime-local" IsTime="time" IsNumeric="number" IsEmail="email" IsTelephone="tel" IsPassword="password" Default="text"
<|PK_PROPERTY_NAME|>	Returns the property name for the primary key field for a table
<|PK_LANGUAGE_DATA_TYPE|>	Returns the data type (and possibly nullable type) of the primary key field for a table for the selected template language
<|PK_LANGUAGE_DATA_TYPE_NEVER_NULLABLE|>	Returns just the data type of the primary key field for a table for the selected template language

<|PRIVATE_FIELD_PREFIX|>	Returns the specified backing field prefix such as an underscore
<|PROPERTY_NAME|> or {|PROPERTY_NAME|}	Returns the property name for the current column
<|PROPERTY_NAME_LOWER_FIRSTCHAR|>	Returns the property name with the first character as lower-case for the current column
<|PROPERTY_NAME_ALL_LOWER|>	Returns the lower-case version of the property name for the current column
<|PROPERTY_LABEL|>	Returns the Label for the current column
<|PROPERTY_SEARCH_PATTERN|>	Returns the Search Pattern used in the View Model templates. Comes from LanguageDataType XML File.
<|PROPERTY_SEARCH_IF|>	Used in the Repository template to help with building the Where() clause.
<|PROPERTY_INITIALIZER|>	Returns the Property Initializer that can be used for initializing property values for the data type of the current column. Comes from LanguageDataType.xml File
<|PROPERTY_INITIALIZER_STATEMENT|>	Returns the Property Initializer Statement that can be used for initializing property values for the data type of the current column. Comes from LanguageDataType.xml File
<|PROPERTY_SETVALUE_STATEMENT|>	Returns the Property SetValue Statement that can be used in the SetValue() method of a view model for setting the changes to the data from the database. Comes from LanguageDataType.xml File
<|STD_PROPERTY_INITIALIZER|>	Returns the Initializer that can be used for initializing property values for the data type of a standard field column. Comes from CodeGen-StandardFields.xml File
<|STD_PROPERTY_MODIFY|>	Returns the Modifier that can be used to set the value of a standard field for initialization. Comes from CodeGen-StandardFields.xml File
<|SEARCH_METHOD_PARAMS|>	Used in controllers to return a comma-delimited list of those parameters in the Search() method that match to the properties in the Search class.
{|DATA_ANNOTATION:[AnnotationName]|}	Returns a single Data Annotation by looking at the Name and Language elements in the CodeGen-DataAnnotations.xml file

Table 10: Column Tokens

Table Tokens

The following tokens are used for table information, or as you are looping through the list of tables.

Token	Description
<|TABLE_DATA_ANNOTATION|>	Returns the data annotation to apply to an entity class for use with the Entity Framework
<|NAMESPACE|> and {|NAMESPACE|}	Returns the Namespace specified in the code generator
<|CLASS_NAME|> and {|CLASS_NAME|}	Returns the Class Name property for the current table
<|CLASS_NAME_SINGULAR|>	Returns the Singular version of the Class Name for the current table
<|CLASS_NAME_PLURAL|>	Returns the Pluralized version of the Class Name for the current table
<|CLASS_DESC_SINGULAR|>	Returns the Singular version of the Description for the current table
<|CLASS_DESC_PLURAL|>	Returns the Pluralized version of the Description for the current table
<|CATALOG_NAME|>	Returns the Catalog for the current table
<|REPO_INTERFACE|>	Returns the definition for the IRepository<> interface for those tables that use an int primary key, or returns IRepositoryOtherPK<> for those that don't.
<|SCHEMA_NAME|>	Returns the Schema for the current table
<|TABLE_NAME|>	Returns the Table name for the current table
<|SELECT_SQL|>	Returns the SQL used to select all columns for the current table

Table 11: Table Tokens

Code Generation Template Tokens

The following tokens are used for information about the current template.

Token	Description
<|BASE_CLASS_NAME|>	Returns the value from the BaseClassName element from current template XML file. These files generally follow the pattern "CodeGen-?.xml" file

<|OUTPUT_PREFIX|>	Returns the value from the OutputFilePrefix element from current template XML file. These files generally follow the pattern "CodeGen-?.xml" file
<|OUTPUT_SUFFIX|>	Returns the value from the OutputFileSuffix element from current template XML file. These files generally follow the pattern "CodeGen-?.xml" file
<|OUTPUT_FILE_EXTENSION|>	Returns the value from the OutputFileExtension element from current template XML file. These files generally follow the pattern "CodeGen-?.xml" file
<|PROPERTIES|>	Generates all properties from a tables' columns for use with the Entity Framework entity class. These properties are generated using all the appropriate data annotations
<|PROPERTIES_STD_FIELDS_ONLY|>	Generates all standard field properties from a tables' columns for use with the Entity Framework entity class. These properties are generated using all the appropriate data annotations
<|PROPERTIES_NO_STD_FIELDS|>	Generates all properties from a tables' columns, except for the standard fields, for use with the Entity Framework entity class. These properties are generated using all the appropriate data annotations
<|DB_CONTEXT|>	The DbContext class name as specified in the code generator
<|CONNECTION_STRING|>	Returns the Connection string as specified in the code generator
<|GEN_PATH|>	Returns the path the code is being generated to
<|APPLICATION_NAME|>	Returns the application name input by the user
<|APPLICATION_NAME_NO_SPACES|>	Returns the application name input by the user with all spaces stripped out
<|DATABASE_NAME|>	Returns the database name that is extracted from the connection string

Table 12: Code Generation Tokens

Screen Generation Tokens

The following tokens are used for screens while generating code.

Token	Description
<|GRID_ROW_AUTO|>	Returns "Auto,Auto,.." for however many rows are needed for a detail screen in .NET MAUI
<|GRID_ROW|>	Returns the grid row number and increments the row number
<|GRID_ROW_FIRST|>	Returns the grid row number, but does not increment the row number

<|GRID_ROW_SECOND|>	Returns the grid row number and increments the row number
<|GRID_ROW_INCREMENT|>	Just increments the row number
<|GRID_ROW_RESET|>	Resets the row number to 0
<|GRID_ROW_RESET:*n*|>	Resets the row number to the value *n* after the colon.

Miscellaneous Generation Tokens

The following tokens are used for various functionality while generating code.

Token	Description
<|CRLF|> and {|CRLF|}	Returns a carriage return, line feed
<|LOGICAL_AND|>	Returns single ampersand (&) after the first time through a loop, and following a <|LOGICAL_AND_RESET|>
<|LOGICAL_AND_RESET|>	Resets "<|LOGICAL_AND|>" token to an empty string
<|LOGICAL_OR|>	Returns single vertical bar (|) after the first time through a loop, and following a <|LOGICAL_OR_RESET|>
<|LOGICAL_OR_RESET|>	Resets "<|LOGICAL_OR|>" token to an empty string
<|CONDITIONAL_AND|>	Returns double ampersands (&&) after the first time through a loop, and following a <|CONDITIONAL_AND_RESET|>
<|CONDITIONAL_AND_RESET|>	Resets "<|CONDITIONAL_AND|>" token to an empty string
<|CONDITIONAL_OR|>	Returns double vertical bars (||) after the first time through a loop, and following a <|CONDITIONAL_OR_RESET|>
<|CONDITIONAL_OR_RESET|>	Resets "<|CONDITIONAL_OR|>" token to an empty string
<|COMMA|>	Returns a comma (,) after the first time through a loop, and following a <|COMMA_RESET|>
<|COMMA_RESET|>	Resets "<|COMMA|>" token to an empty string
<|AND|>	Returns the word "And" after the first time through a loop, and following a <|AND_RESET|>
<|AND_RESET|>	Resets "<|AND|>" token to an empty string
<|OR|>	Returns the word "Or" after the first time through a loop, and following a <|OR_RESET|>
<|OR_RESET|>	Resets "<|OR|>" token to an empty string
<|REMOVE_LINE|>	Do not add the current line to the output

<|SOLUTION_GUID|>	Returns a new Guid
<|VS_VERSION|>	Returns the version of Visual Studio running on the machine
{|INCLUDE:FILENAME|}	Allows you to include one template file into the specified location of another template file. This included template file is inserted into the location after all other templates have been generated.

Table 13: Miscellaneous Generation Tokens

Summary

The PDSC Developer Utilities v8 helps increase your productivity while developing your applications. We hope you enjoy using this product.