PDSC Developer Utilities - Version 9

The PDSC Developer Utilities (9) as shown in Figure 1 is a set of tools to help you develop your .NET applications, and to 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.

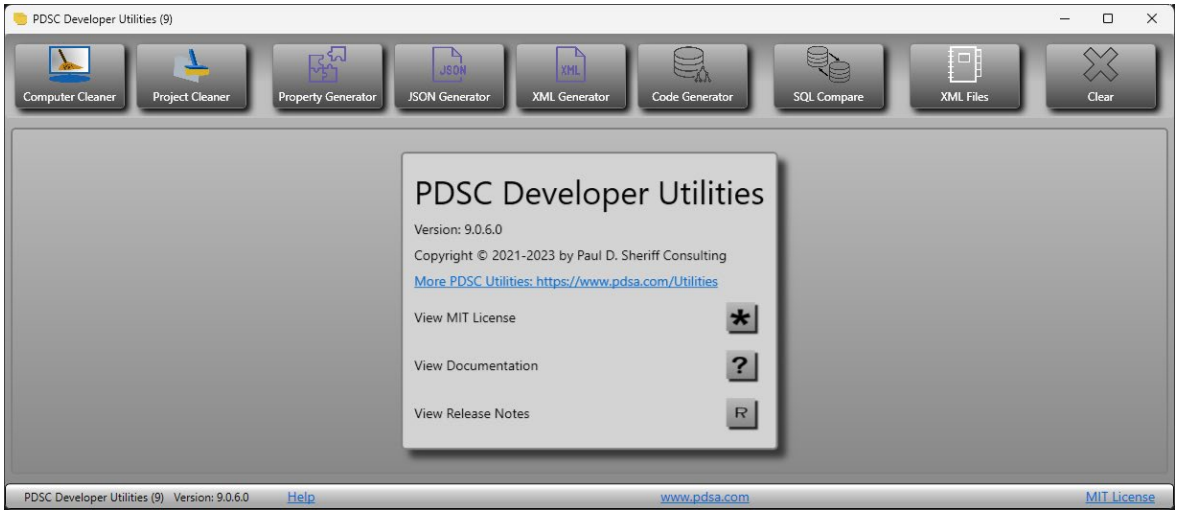

Figure 1: Screen shot of the PDSC Developer Utilities Startup Screen.

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.
Code Generator	This utility generates Entity, Repository, View Model and Search classes from a table. It can also generate CRUD WPF, MVC, .NET MAUI, Web API (MVC), and Minimal Web API applications.
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.
XML Files	A list and a description of each XML file used in the PDSC Developer Utilities.

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.

NOTE:	You may need to run this utility as an **Administrator** on your computer to have it clean up all files and folders.

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.

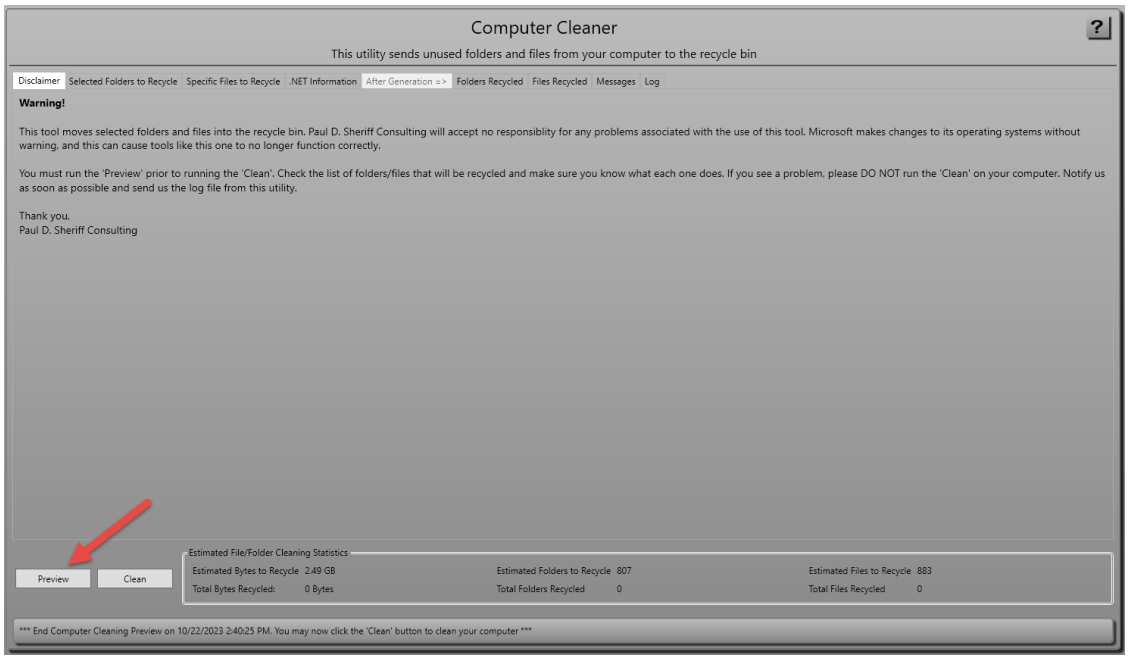

Figure 2: Disclaimer screen.

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.

> **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 is displayed (Figure 3). The list of folders that are going to be recycled is contained in the file **[MyDocuments]\PDSCDeveloperUtilities9\Xml\ComputerCleaner-FoldersToRecycle.xml**. You can also create a **ComputerCleaner-FoldersToRecycle.xml** in the **[MyDocuments]\PDSCDeveloperUtilities9-[YourLoginId]\Xml** to add as many additional folders to recycle as you wish.

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.

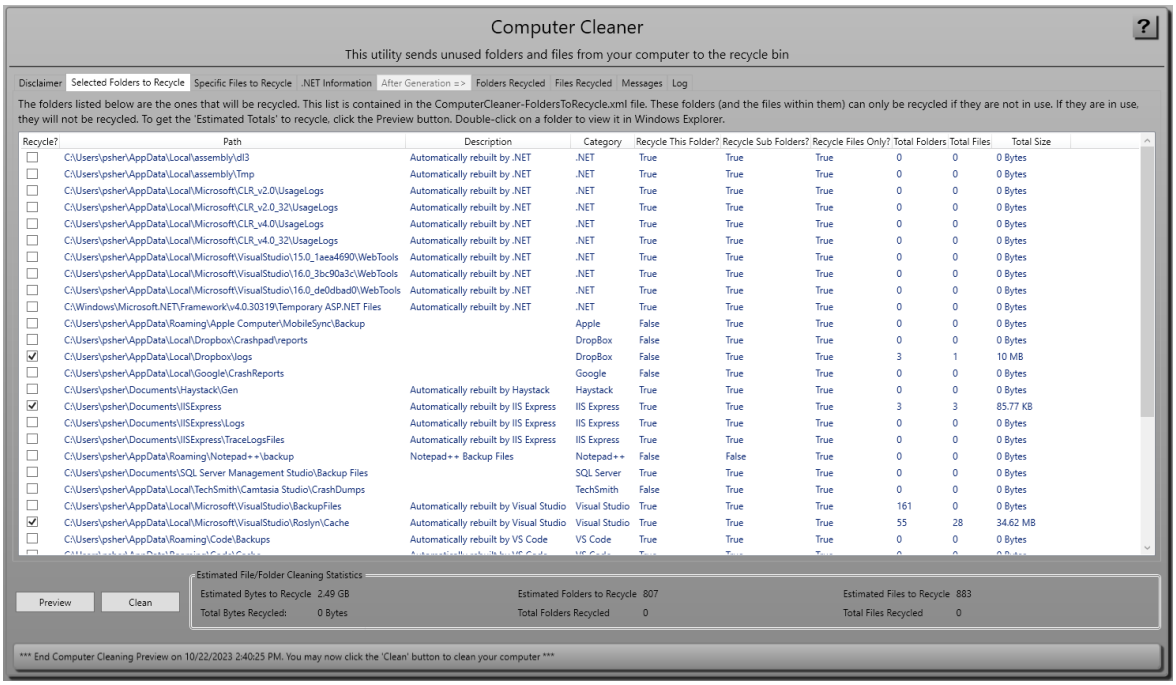

Figure 3: A list of the specified folders to recycle.

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 estimated 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.

Specific 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]\PDSCDeveloperUtilities9\Xml\ComputerCleaner-FilesToRecycle.xml**. You can also create a **ComputerCleaner-FilesToRecycle.xml** in the **[MyDocuments]\PDSCDeveloperUtilities9-[YourLoginId]\Xml** to add as many additional files to recycle as you wish.

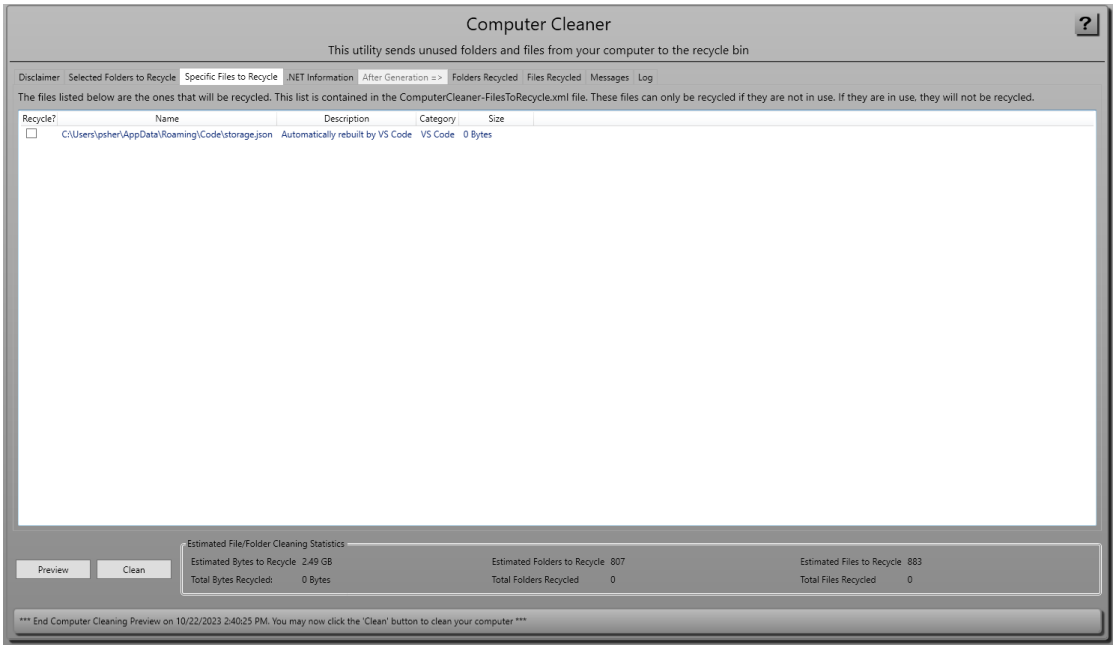

Figure 4: You can add additional files to recycle.

.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.

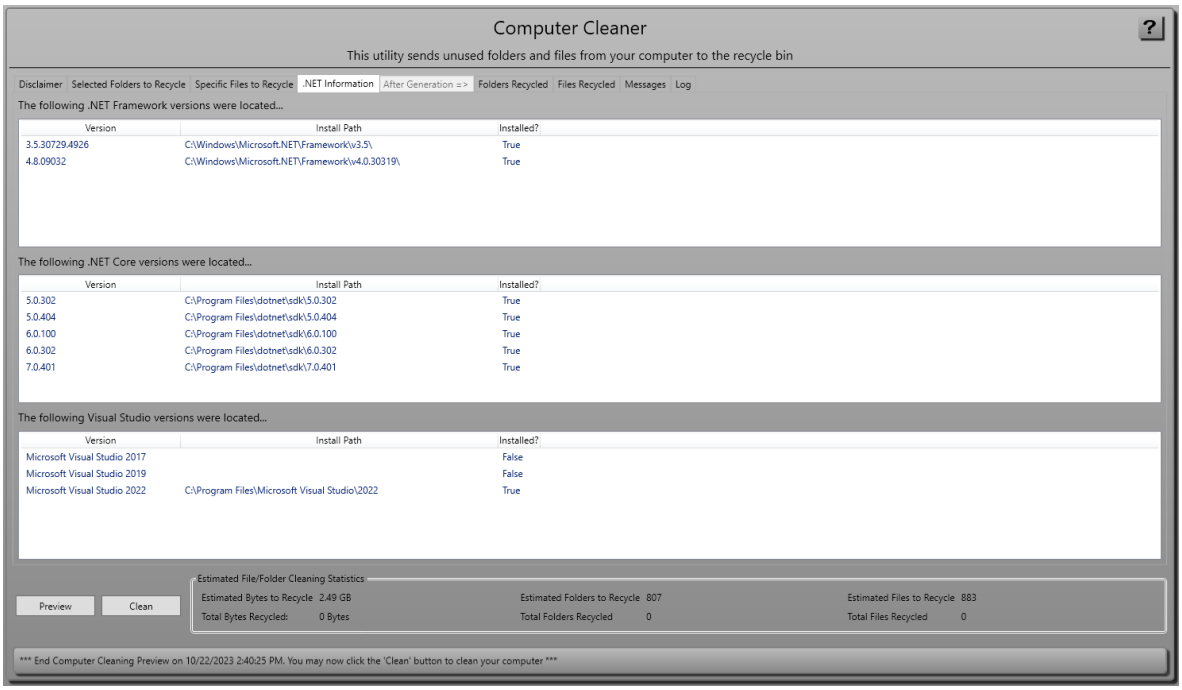

Figure 5: The list of .NET Frameworks, .NET Core and Visual Studio versions on your computer.

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]\PDSCDeveloperUtilities9-[YourLoginId]\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 recycled. Or because of security constraints, they also may not be able to be recycled.

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 may want to eliminate all these extra files and folders using the Project Cleaner utility.

Folder to Clean

The first tab, shown in Figure 6, allows you to choose the folder to clean up.

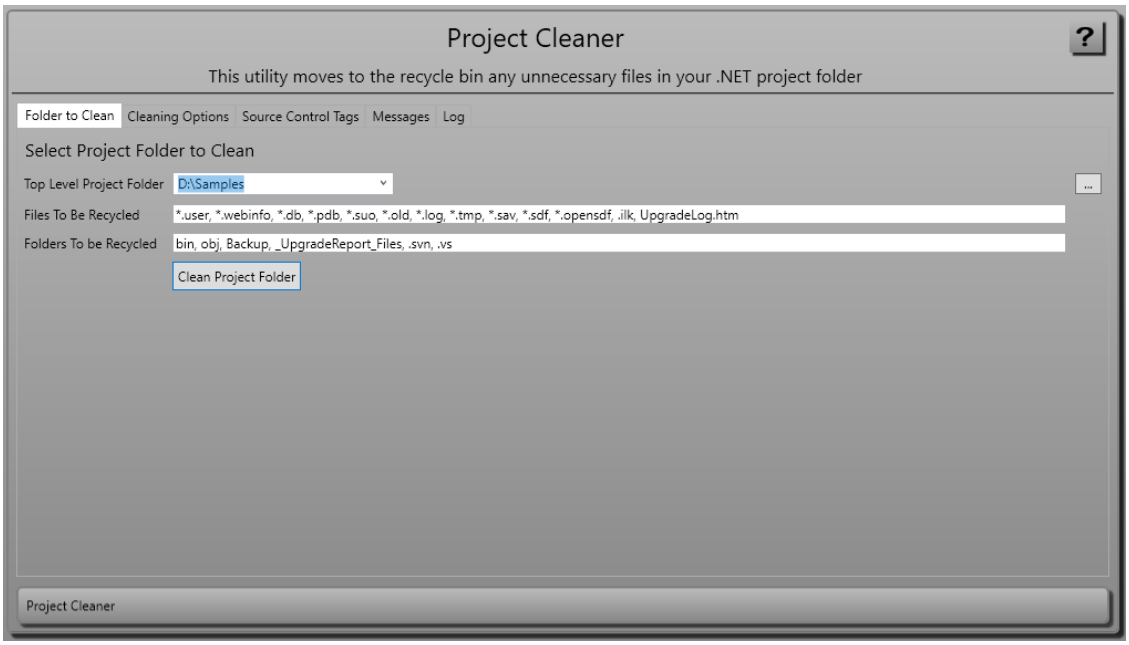

Figure 6: Tab: Folder to Clean - Clean up files using the Project Cleaner utility.

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 fields on this tab are described in Table 2.

Field	Description
Top Level Project Folder	Enter the top-level folder you wish to iterate through
Files to be Recycled	A list of file extensions that should be recycled.
Folders to be Recycled	A list of folder names that should be recycled.

Table 2: Fields to fill in for cleaning projects.

NOTE: This utility only goes through 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.

Cleaning Options

On the Cleaning Options, shown in Figure 7, you can select which folders and files to delete when cleaning up the top-level folder.

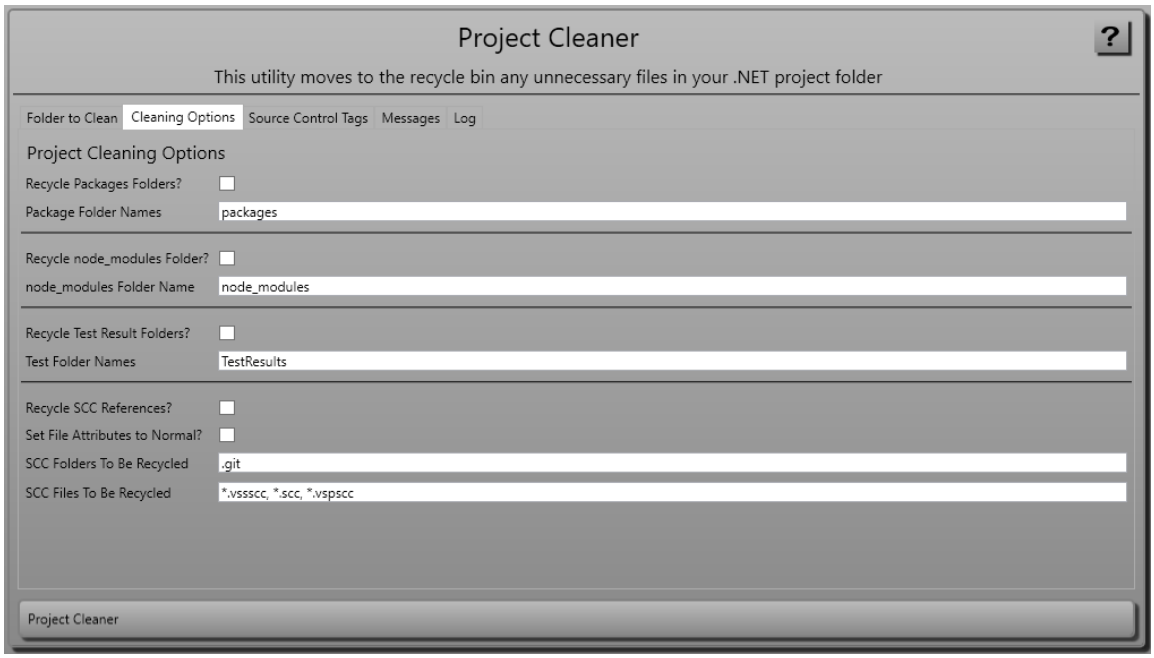

Figure 7: Tab: Cleaning Options – Choose which items to remove from the top-level folder.

The fields on this tab are described in Table 3.

Field	Description
Recycle Packages Folders?	recycle the "packages" folder.
Package Folder Names	Fill in the names of the packages folders to recycle.
Recycle node_modules Folder?	Check to recycle any node_modules folders.
node_modules Folder Name	Fill in the names of the node_modules folders to recycle.
Recycle Test Result Folders?	Check to recycle any test result folders.
Test Folder Names	Fill in the names of the test result folders to recycle.

Recycle SCC References?	Check this is you wish this utility to recycle 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 Recycled	A list of source control folders that should be recycle.
SCC Files to be Recycled	A list of source control file extensions that should be recycle.

Table 3: Options for files/folders to remove when cleaning up a project.

Source Control Tags

The list of SCC tags (Figure 8) that are going to be removed from your .SLN or .CSPROJ file comes from the **[MyDocuments]\PDSCDeveloperUtilities9\Xml\ ProjectCleaner-SourceControlTags.xml**. You can also create a **ProjectCleaner-SourceControlTags.xml** in the **[MyDocuments]\PDSCDeveloperUtilities9-[YourLoginId]\Xml** to add as many additional SCC tags to recycle as you wish.

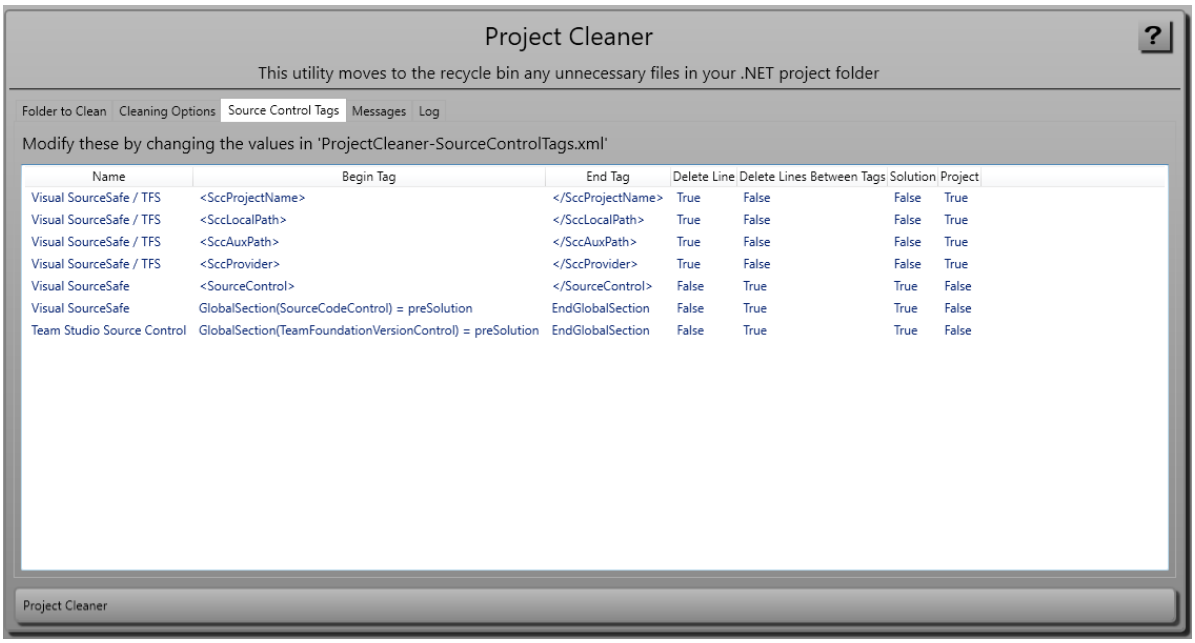

Figure 8: Tab: Cleaning Options – Choose which items to remove from the top-level folder.

Property Generator

Visual Studio has code snippets that will let you create properties (Figure 9). 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.

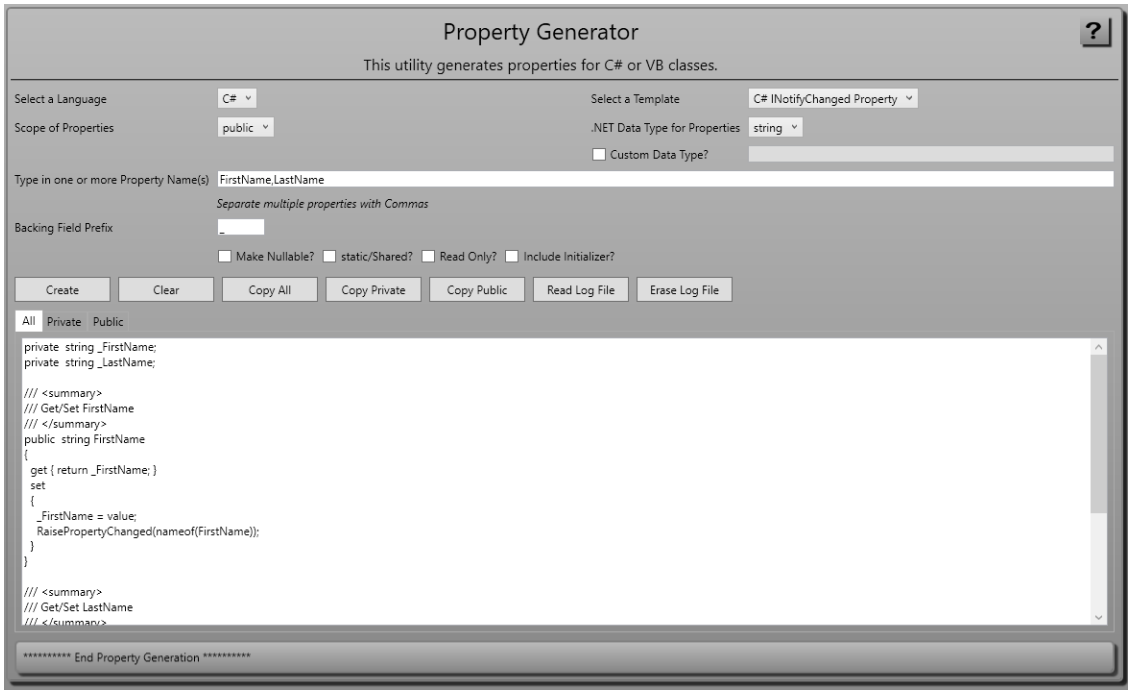

Figure 9: Property generator helps you create properties in many different styles.

The Supplied Templates

Under the **[My Documents]\PDSCDeveloperUtilities9\Xml** folder is a file named **PropertyGen-Templates.xml**. This file contains the list of template files supplied with the PDSC Developer Utilities used to generate properties for C# and Visual Basic. There is also a folder named **\Templates\PropertyGenerator** (Figure 10) in which there are several .txt files that hold the snippets for each of the types of properties that you can generate.

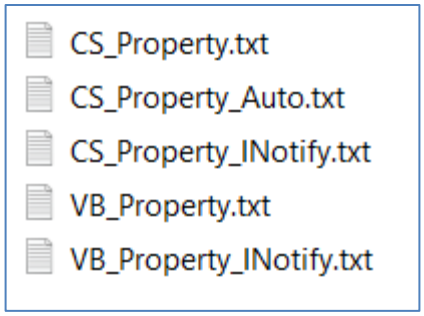

Figure 10: All the property snippets are just contained in .txt files

In this **PropertyGen-Templates.xml** file is one node for each .txt file located in the **[My Documents]\PDSCDeveloperUtilities9\Templates\PropertyGenerator** folder.

Add Your Own Property Generator Templates

To add a new template to generate properties the way you want them, copy the **[My Documents]\PDSCDeveloperUtilities9\Xml\PropertyGen-Templates.xml** file to **[My Documents]\PDSCDeveloperUtilities9-[YourLoginId]\Xml\PropertyGen-Templates.xml**. Open this new file and delete all but the first node.

Modify the **Description** element to something you will recognize and the **FileName** element to the name of your new .txt file. Set the **Language** element to either **CSharp** or **VBNET**. Specify a true or false value in the GenPrivateVars and GenPublic elements as appropriate for your template.

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

Create a copy of the **[My Documents]\PDSCDeveloperUtilities9\Templates\CS_Property.txt** and paste that copy into the **[My Documents]\PDSCDeveloperUtilities9-[YourLoginId]\Templates\PropertyGenerator** folder. Rename the file to **CS_Test.txt**. Open the **CS_Test.txt** in Notepad and modify the template to however you want it to generate. For example, in the following code a method named MyMethod() within the set statement.

```
/// <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|}
}
```

Close and restart the PDSC Developer Utilities and your new template for the property generator now appears as a C# property generation option.

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 4 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 4. List of Tokens in Property generator

Other XML files for the Property Generator

There are a few other XML files (Table 5) that the property generator uses to assist with the generation. These files are located in the **[My Documents]\PDSCDeveloperUtilities9\Xml** folder under the location where you installed the Developer Utilities.

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

Table 5. List of XML files used by 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 11). You can select the type of objects to load (Tables and/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.

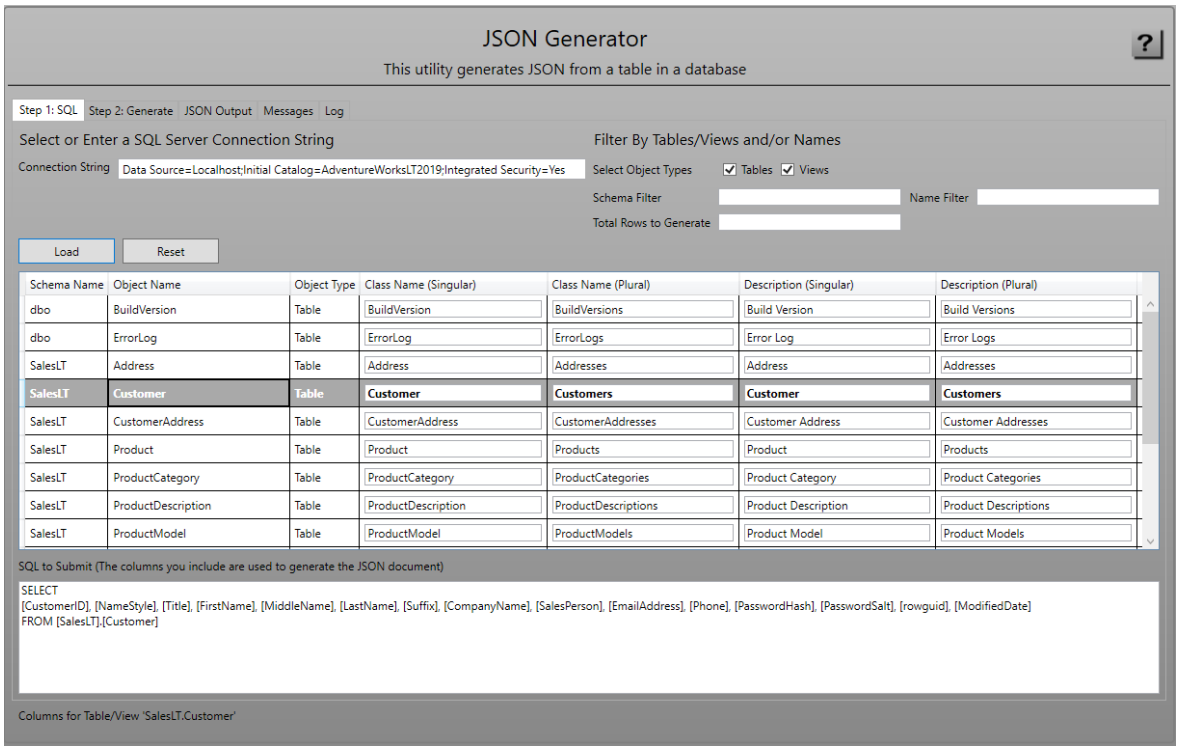

Figure 11: Step 1: JSON Generator SQL Tab.

Step 2: Generate

Click on **Step 2: Generate** (Figure 12) 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.

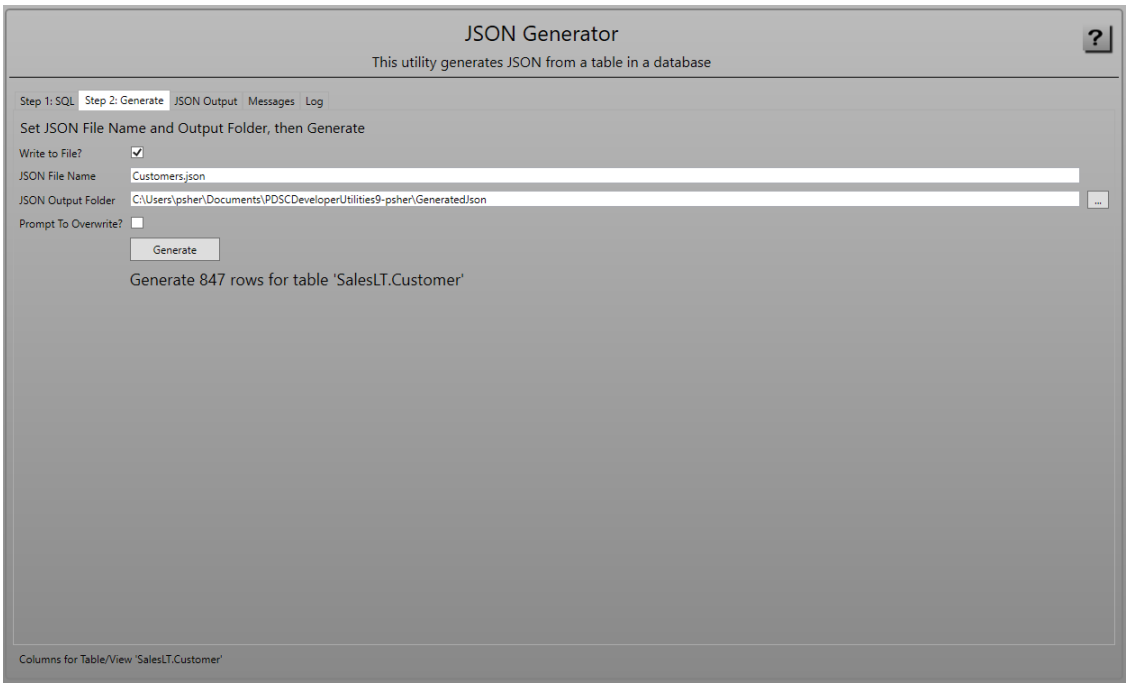

Figure 12: Step 2: JSON Generator Generate Tab.

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

View the JSON Output

After you click on the **Generate** (Figure 13) button, you are presented with the screen shown in Figure 13. This screen shows you where the generated .json file is located and the JSON output.

Figure 13: 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 14). You can select the type of objects to load (Tables and/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.

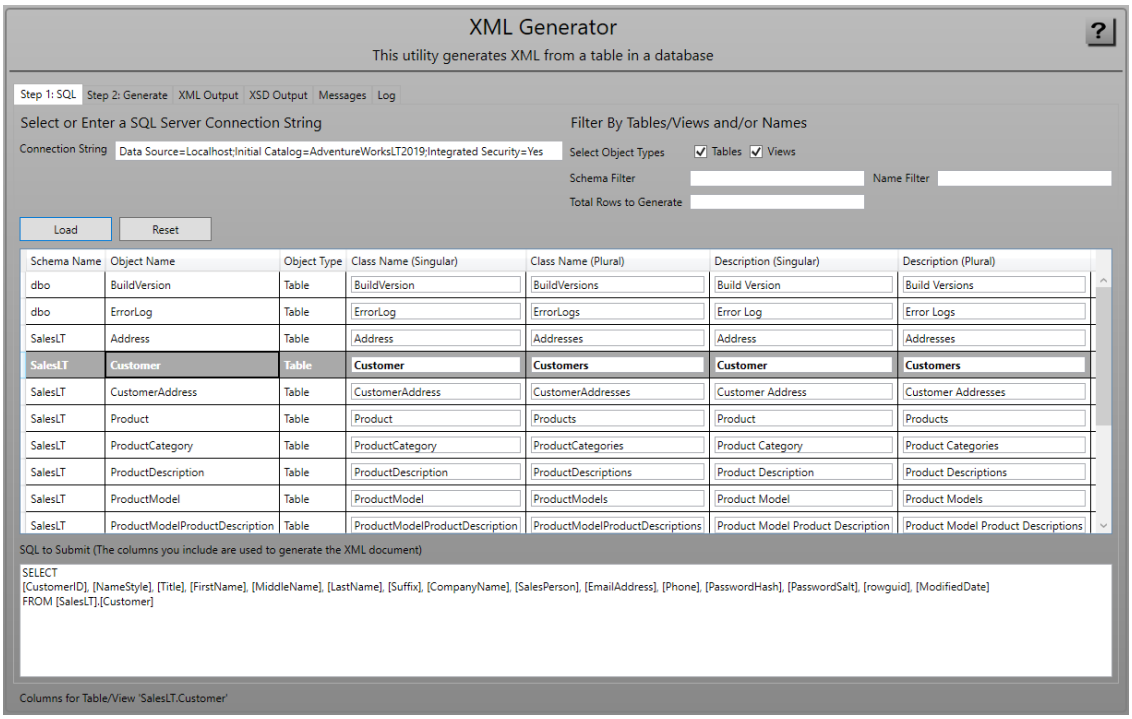

Figure 14: Step 1: XML Generator SQL Tab.

Step 2: Generate

Click on **Step 2: Generate** (Figure 15) 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** button to start the generation process.

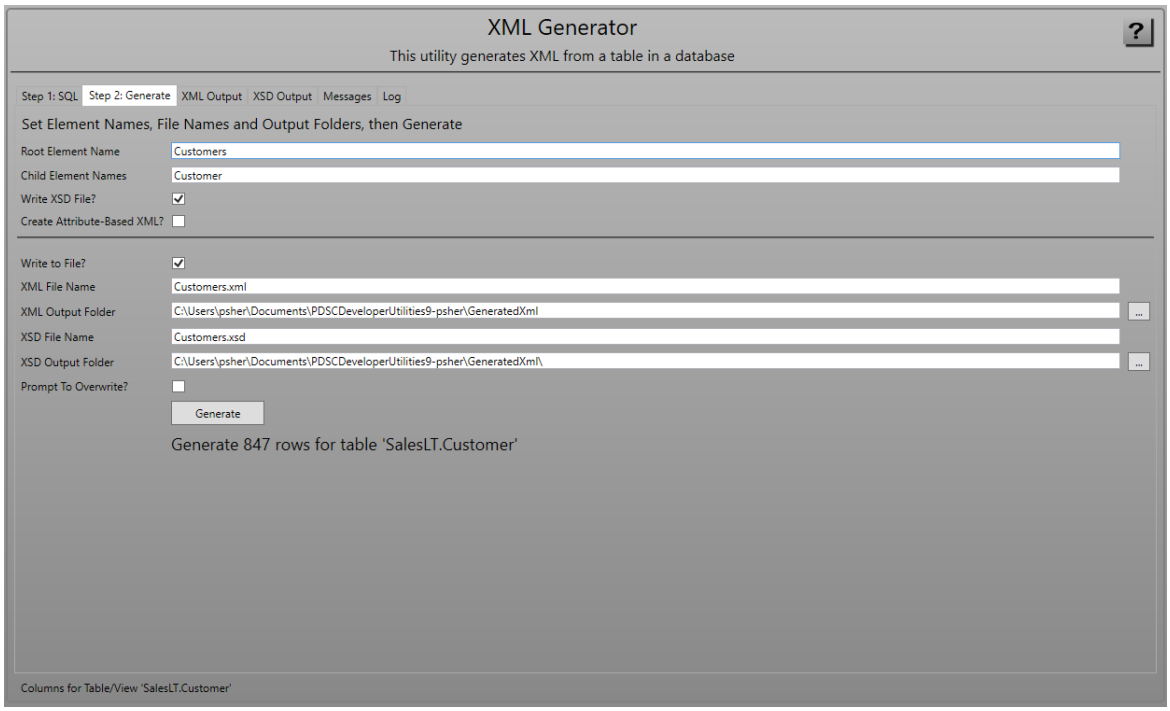

Figure 15: Step 2: XML Generator Generate Tab.

XML Output

After you click on the **Generate** button, you will be presented with the screen shown in Figure 16.

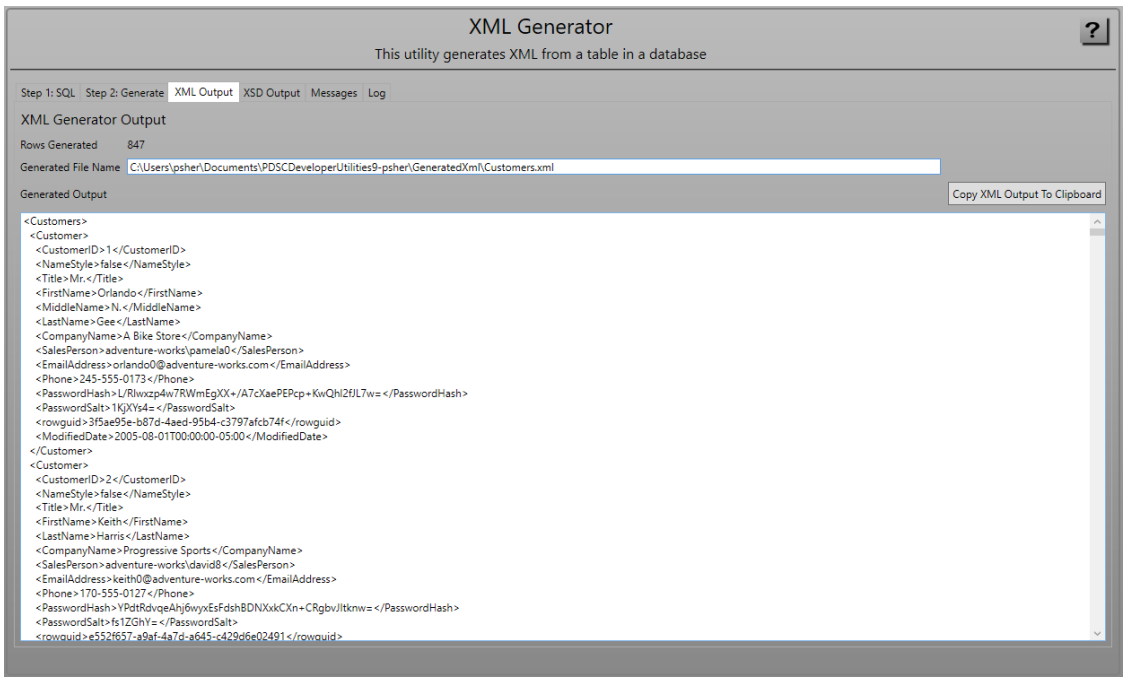

Figure 16: XML Output Tab.

XSD Output

If you generated an XSD, you view the XSD on the screen show in Figure 17.

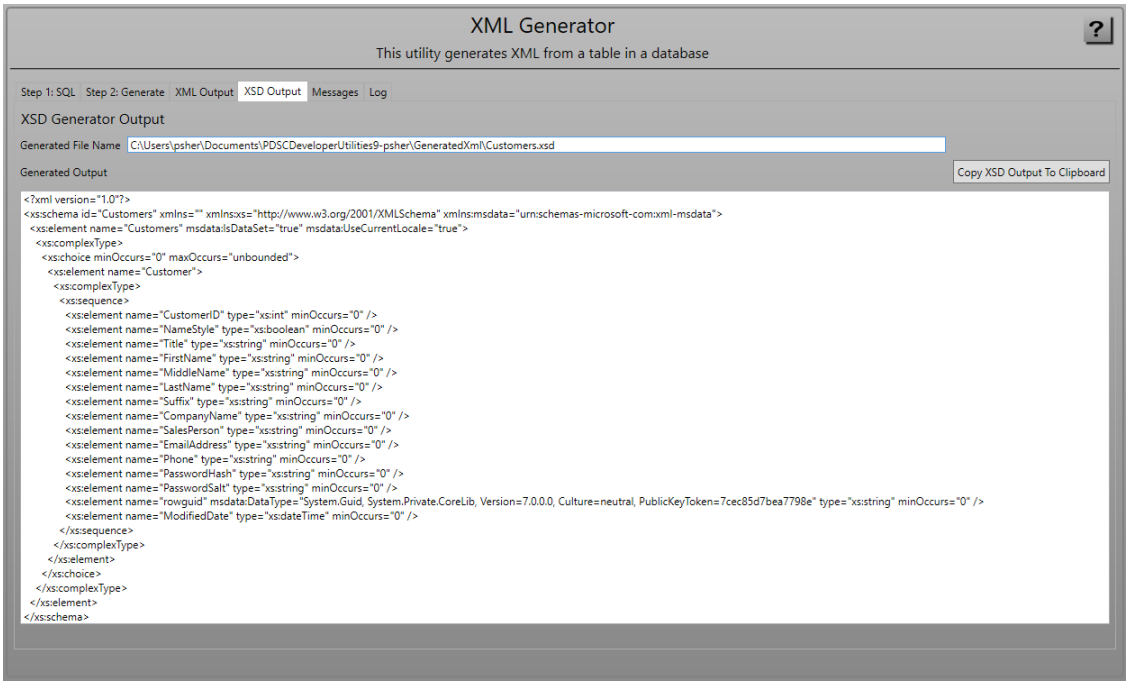

Figure 17: XSD Output tab.

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. You can generate a Web API application using MVC, a .NET MAUI application, and an MVC website application. Other technologies are added periodically to this code generator.

Code Generator Limits

The code generator is intended to provide you with a good head start on the standard CRUD logic for your tables, but it can't handle every situation. Below are some scenarios that are not currently handled by the code generator.

- Multiple columns in the primary key.
- Two fields having a foreign key relationship to the same foreign key table.
- A field in one table referencing a foreign key field in the same table.

- When there is no auto-incrementing primary key on a table (or a string or GUID primary key), the edit control on the page can be edited. You need to mark this edit control as read-only after generating.

The above scenarios will require you to do some manual coding to get it to work correctly.

Step 1: Select Template(s)

On the first tab (Figure 18) 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.

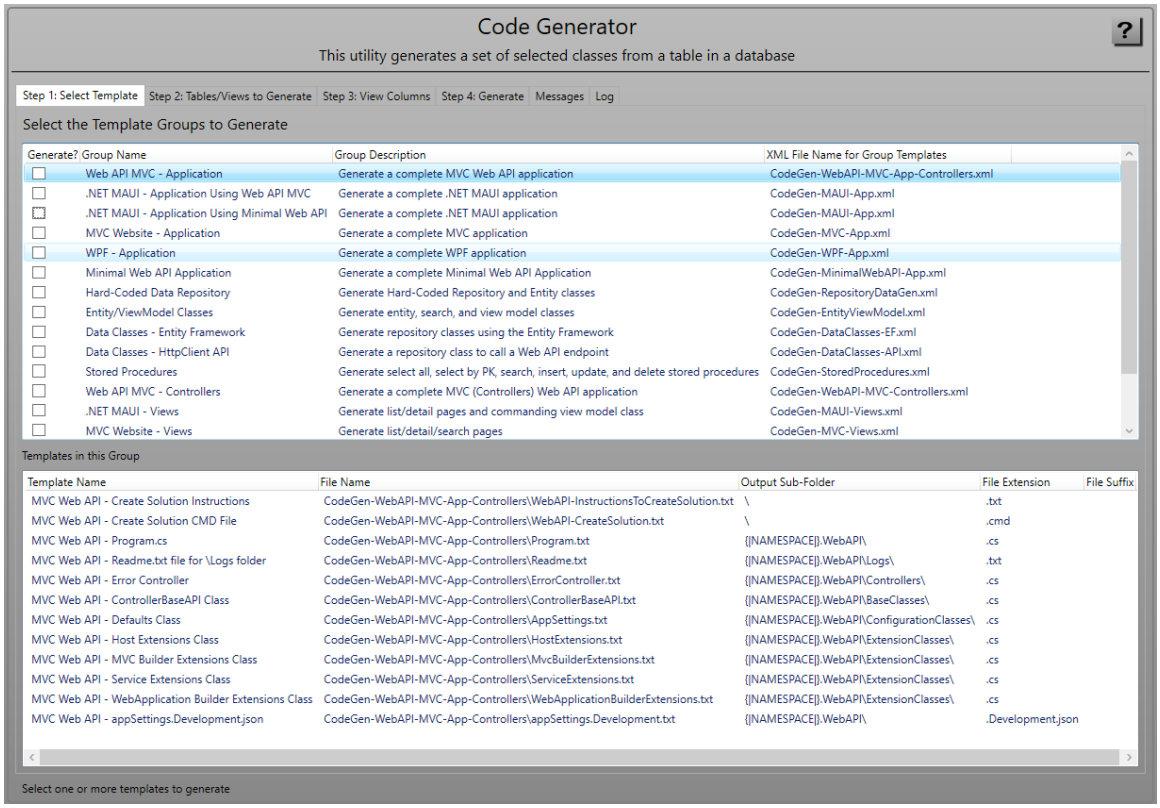

Figure 18: Step 1: Code Generator Select Template Tab.

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 Using Web API MVC	Generates all the files necessary to build a complete .NET MAUI application using the Web API MVC for data access. A .cmd file is also created to make it easy to create the .NET MAUI application once generated.
.NET MAUI – Application Using Minimal Web API	Generates all the files necessary to build a complete .NET MAUI application using the Minimal API MVC for data access. 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. A .cmd file is also created to make it easy to create the MVC Website application once generated.
WPF - Application	Generates a WPF application and all the list and view screens for all tables selected. A .cmd file is also created to make it easy to create the WPF application once generated.
Minimal Web API Application	Generates all the files necessary to build a complete Minimal Web API application using router classes. A .cmd file is also created to make it easy to create the Minimal Web API application once generated.
Hard-Coded Data Repository	Generates Entity and Repository classes. The Repository class returns hard-coded list of Entity data that you select from one of your database tables. The fields generated are the primary key field(s) and any tables marked as **Display in Table**. 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.
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.
Stored Procedures	Generates a set of CRUD stored procedures for each table.
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.
WPF – Views	Generates list, detail, and search screens to be used in a WPF application.
Minimal Web API Routers and Search Classes	Generates router classes and search classes that work with Minimal Web API applications.

Table 6. List of Standard Template in the Code Generator

Step 2: Tables/Views to Generate

Either type in or select from the drop-down a connection string that will connect you to your database (Figure 19). You can select the type of objects to load (Tables and/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 Tables/Views** button.

Click on the **Load Tables/Views** 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.

NOTE:	Tables that have multiple primary key fields cannot be generated. Tables with no primary keys will be generated as a View.

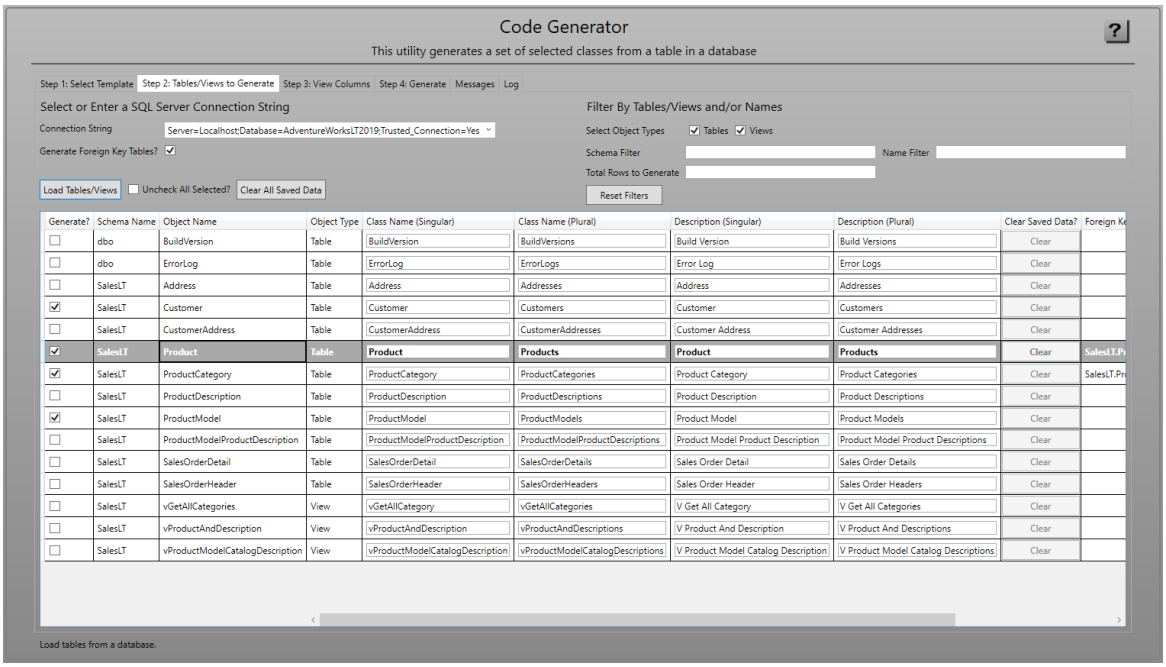

Figure 19: Step 2: Code Generator Tables/Views to Generate Tab.

Table 8 describes each of the various columns in the Columns grid.

Column	Description
Generate?	Check this to generate code for this table/view.
Schema Name	The SQL Server schema name that contains this database object.
Object Name	The SQL Server database object name.
Object Type	The SQL Server database object type (Table or View).
Class Name (Singular)	The class name to be generated for this database object as a "singular" noun.
Class Name (Plural)	The class name to be generated for this database object as a "plural" noun.
Description (Singular)	A description that can be used in code generation as a "singular" verb.
Description (Plural)	A description that can be used in code generation as a "plural" verb.

Clear Saved Data	When changes are made to the previous columns, this data and the column data are stored in a **[MyDocuments]\PDSCDeveloperUtilities9-[YourLoginId]\CodeGeneration\[DatabaseName]\[TableName].json** file. Clicking on this button deletes that file so you can reset all previously saved changes.
Foreign Keys	The list of related tables to the current table.
Can't Generate Reason	This is a message that describes why a table can't be generated, or if a table will be generated as a view.

Table 7: A description of the columns in the code generation table/views grid.

Step 3: View Columns

On this tab (Figure 20) 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.

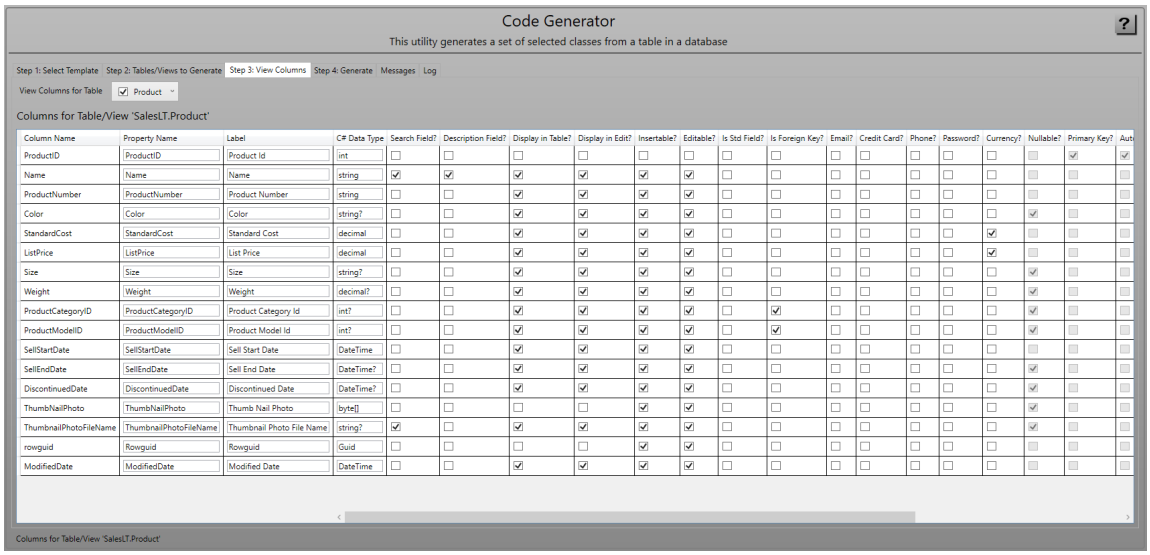

Figure 20: Step 3: Code Generator View Columns Tab.

Table 8 describes each of the various columns in the Columns grid.

Column	Description
Column Name	This is the actual column name in the table. This value cannot be changed.

Property Name	The name of the C# property that will be mapped to this field.
Label	The label to display next to the data entry field for this column.
C# Data Type	The C# data type this column is mapped to.
Search Field?	Do you want this column to appear in a screen used for searching? This field will also be included in the AddWhereClause() method in the Repository class.
Description Field?	Typically, you only specify one column as the description column. This column may be used in a list box or a combo box in the drop-down portion of the control.
Display in Table?	Do you want this column to appear in a table or list displayed on a screen?
Display in Edit?	Do you want this column to appear as an editable field in a form displayed for the user.
Insertable?	Can the value for this column be in an INSERT statement, or is it automatically generated like an IDENTITY property.
Editable?	Can the value for this column be in an UPDATE statement, or is it automatically generated.
Is Std. Field?	Is this column one of the standard fields listed in the **[MyDocuments]\PDSCDeveloperUtilities9-[YourLoginId]\Xml\CodeGen-StandardFields.xml** file.
Is Foreign key?	Is this column a Foreign Key?
Email?	Does this column hold an email address?
Credit Card?	Does this column hold a credit card number?
Phone?	Does this column hold a phone number?
Password?	Does this column hold a password?
Currency?	Does this column hold a currency value?
	The rest of the columns are gathered directly from the SQL Server table and are not changeable.

Table 8: A description of the various column properties you may set.

Step 4: Generate

Click on **Step 4: Generate** (Figure 21) to fill in information on how you wish to generate the classes as shown in Table 9.

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.
.NET Version	Select which version of .NET you wish to generate.
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.
Stored Procedure Prefix	The prefix to use before all generated stored procedures.
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.
Use HTTPS for Web API Project?	Check this if you wish to use HTTPS for the Web API project.

Table 9: Code Generation Parameters.

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

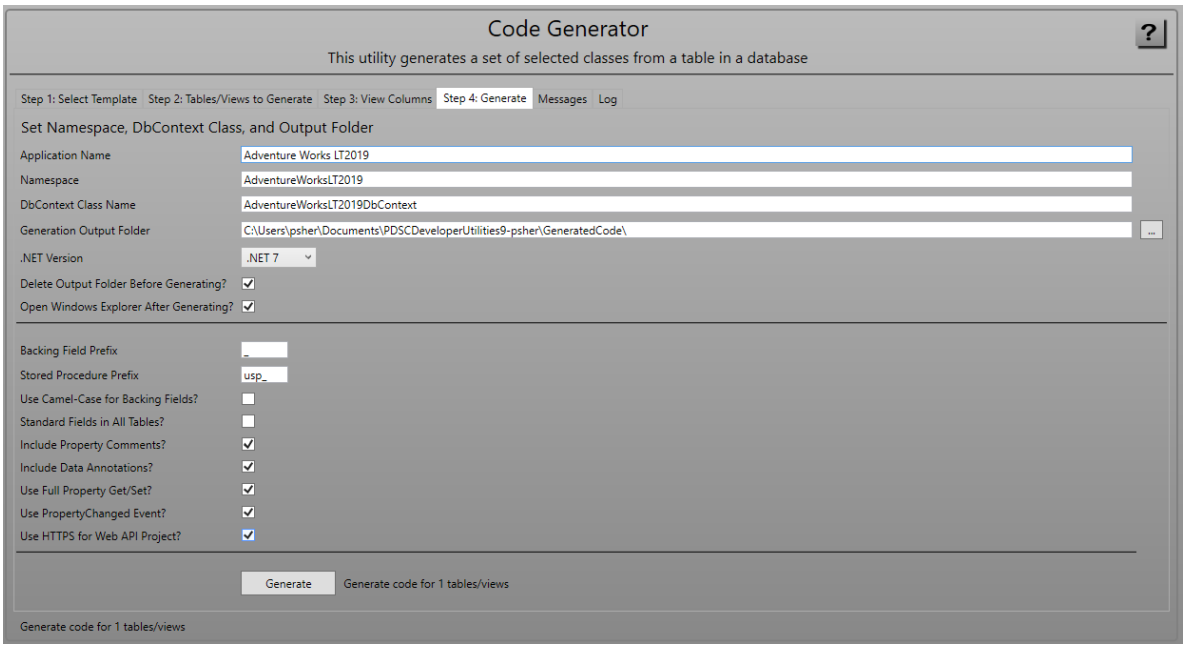

Figure 21: Step 4: Code Generator Generate Tab.

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.

Adding a New Generated Table to Your Project

In the **_DoNotAddToProject-Includes** folder (Figure 22) is where you will find files that have code you may need to manually add to some of your previously generated files in your project.

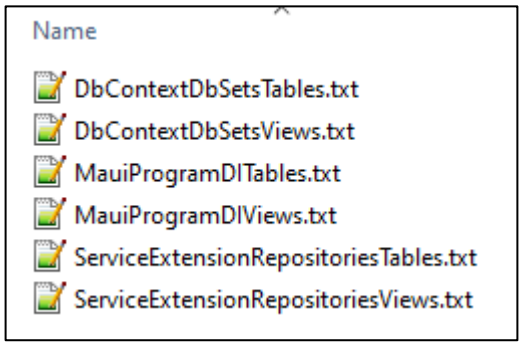

Figure 22: Examples of additional code to add to previously generated files

After you have generated your project for the first time, and you then go back into the Code Generator to add another table, there are a few steps you take to add the classes required to work with the new table to your Visual Studio project.

All Applications

Open the GeneratedCode_DoNotAddToProject-Includes\DbContextDbSetsTables.txt file and add any lines of code to the PROJECT.DataLayer\DbContextClasses\PROJECTDbContext.cs class in your VS project.

Open the GeneratedCode_DoNotAddToProject-Includes\DbContextDbSetsViews.txt file and add any lines of code to the OnModelCreating() method in the PROJECT.DataLayer\DbContextClasses\PROJECTDbContext.cs class in your VS project.

Copy the GeneratedCode\PROJECT.DataLayer\RepositoryClasses*.cs file(s) to the PROJECT.DataLayer\RepositoryClasses folder in your VS project.

Copy the GeneratedCode\PROJECT.EntityLayer\EntityClasses*.cs file(s) to the PROJECT.EntityLayer\EntityClasses folder in your VS project.

Copy the GeneratedCode\PROJECT.EntityLayer\SearchClasses*.cs file(s) to the PROJECT.EntityLayer\SearchClasses folder in your VS project.

Copy the GeneratedCode\PROJECT.ViewModelLayer\ViewModelClasses*.cs file(s) to the PROJECT.ViewModelLayer\ViewModelClasses folder in your VS project.

For .NET MAUI Applications

Open the GeneratedCode_DoNotAddToProject-Includes\MauiProgramDITable.txt file and add any lines of code to the appropriate methods within the MauiProgram.cs file in your VS project.

Open the GeneratedCode_DoNotAddToProject-Includes\MauiProgramDIViews.txt file and add any lines of code to the appropriate methods within the MauiProgram.cs file in your VS project.

Copy the GeneratedCode\PROJECT.DataLayer.API\RepositoryClasses*.cs file(s) to the PROJECT.DataLayer.API\RepositoryClasses folder in your VS project if this folder exists in your project.

Copy the GeneratedCode\PROJECT.MAUI\Views*.* file(s) to the PROJECT.MAUI\Views folder in your VS project.

For Web API (MVC) Projects

Open the Open the GeneratedCode_DoNotAddToProject-Includes\ServiceExtensionRepositoriesTables.txt file and add any lines of code to the appropriate methods within the \ExtensionClasses\ServiceExtensions.cs file in your VS project.

Open the Open the GeneratedCode_DoNotAddToProject-Includes\ServiceExtensionRepositoriesViews.txt file and add any lines of code to

the appropriate methods within the \ExtensionClasses\ServiceExtensions.cs file in your VS project.

Copy the GeneratedCode\PROJECT.WebAPI\Controllers*.cs file(s) to the PROJECT.WebAPI\Controllers folder in your VS project.

DO **NOT** copy the **ErrorController.cs** file.

For Minimal Web API Projects

Open the Open the GeneratedCode_DoNotAddToProject-Includes\ServiceExtensionRepositoriesTables.txt file and add any lines of code to the appropriate methods within the \ExtensionClasses\ServiceExtensions.cs file in your VS project.

Open the Open the GeneratedCode_DoNotAddToProject-Includes\ServiceExtensionRepositoriesViews.txt file and add any lines of code to the appropriate methods within the \ExtensionClasses\ServiceExtensions.cs file in your VS project.

Copy the GeneratedCode\PROJECT.MinWebAPI\RouterClasses*.cs file(s) to the PROJECT.MinWebAPI\RouterClasses folder in your VS project.

Copy the GeneratedCode\PROJECT.MinWebAPI\SearchClasses*.cs file(s) to the PROJECT.MinWebAPI\SearchClasses folder in your VS project.

For MVC Website Projects

Open the Open the GeneratedCode_DoNotAddToProject-Includes\ProgramDITables.txt file and add any lines of code to the appropriate methods within the Program.cs file in your VS project.

Open the Open the GeneratedCode_DoNotAddToProject-Includes\ProgramDIViews.txt file and add any lines of code to the appropriate methods within the Program.cs file in your VS project.

Copy the GeneratedCode\PROJECT.MVC\Controllers*.cs file(s) to the PROJECT.MVC\Controllers folder in your VS project.

Copy the GeneratedCode\PROJECT.MVC\Views\OBJECTNAME folder(s) to the PROJECT.MVC\Views folder in your VS project.

For WPF Projects

Open the Open the GeneratedCode_DoNotAddToProject-Includes\MainWindowTables.txt file and add any lines of code to the RouteTo() method within the MainWindows.xaml.cs file in your VS project.

Open the Open the GeneratedCode_DoNotAddToProject-Includes\MainWindowViews.txt file and add any lines of code to the RouteTo() method within the MainWindows.xaml.cs file in your VS project.

Copy the GeneratedCode\PROJECT.WPF\Views*.* file(s) to the PROJECT.WPF\Views folder in your VS project.

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 23) and a complete list of missing or changed objects will appear in the message's tabs at the bottom of the screen.

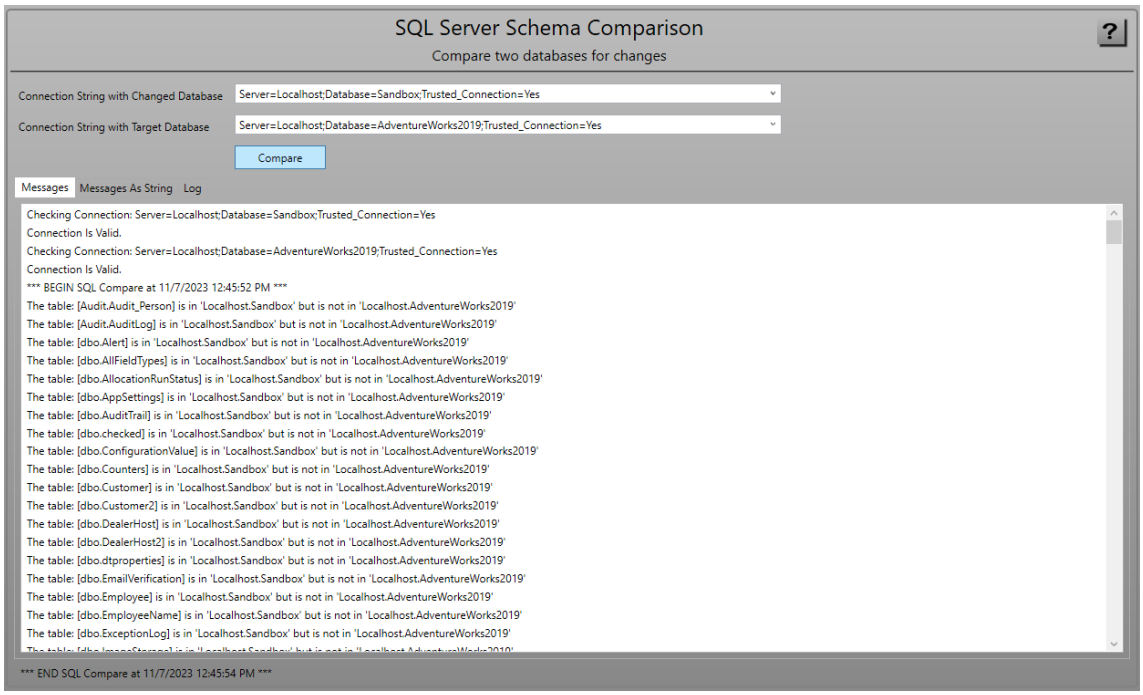

Figure 23: Get missing objects from one database to another via the SQL Compare Utility.

XML Files

This screen is a list and a description of each XML file used in the PDSC Developer Utilities.

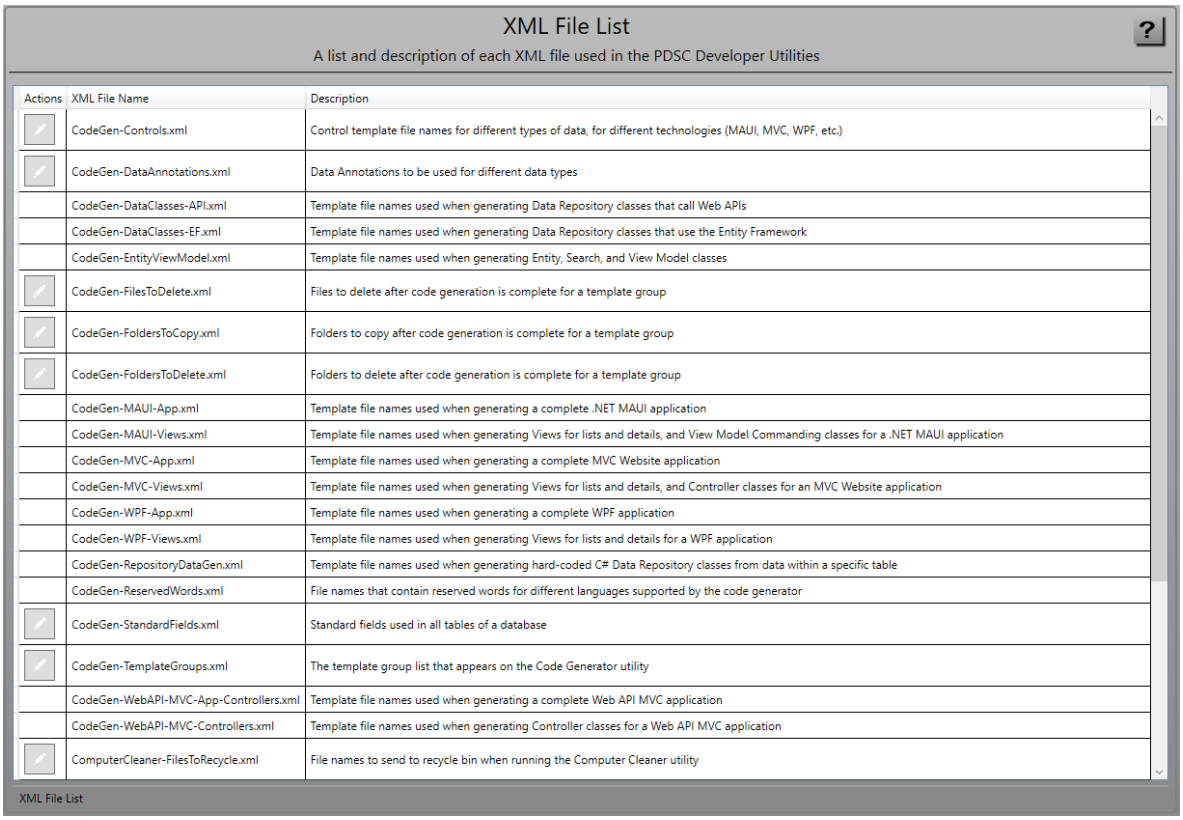

Figure 24: A list of all XML files used in the PDSC Developer Utilities.

XML Files used by the Code Generator

Table 10 is a list of those XML files used by the Code Generation tool.

XML File	Description
CodeGen-Controls	Control template file names for different types of data, for different technologies (MAUI, MVC, WPF, etc.)
CodeGen-DataAnnotations	Data Annotations to be used for different data types
CodeGen-FilesToDelete	Files to delete after code generation is complete for a template group
CodeGen-FolderToCopy	Folders to copy after code generation is complete for a template group
CodeGen-FoldersToDelete	Folders to delete after code generation is complete for a template group
CodeGen-ReservedWords	File names that contain reserved words for different languages supported by the code generator
CodeGen-StandardFields	The template group list that appears on the Code Generator utility

CodeGen-TemplateGroups	Points to a CodeGen-TECH-*.xml file, other than the ones listed above. These XML root within these XML files start is named <CodeGenTemplates>.

Table 10: XML files used by the Code Generator utility.

CodeGen-TemplateGroups.xml File

The CodeGen-TemplateGroups.xml file has the following structure. A description of each element is shown below.

```
<CodeGenTemplateGroups>
  <TemplateGroup>
    <GroupId>
      A Unique ID
    </GroupId>
    <DependentGroupIds>
      Other Groups This one Depends Upon
    </DependentGroupIds>
    <DisplayOrder>
      The display order for displaying on the screen
    </DisplayOrder>
    <GroupName>
      A Group Name
    </GroupName>
    <Description>
      A description of this group
    </Description>
    <LanguageCode>
      CSharp or VB.NET
    </LanguageCode>
    <TemplatesXmlFileName>
      The XML File Name that contains the templates to generate
    </TemplatesXmlFileName>
    <RequirePropertyChangedEvent>
      True or False
    </RequirePropertyChangedEvent>
  </TemplateGroup>
</CodeGenTemplateGroups>
```

XML Files for UI Technologies

All other **CodeGen-*.xml** files not described in the tables above are Code Generation Template XML files. Below is the structure of one of the code generation template files.

```
<CodeGenTemplates>
  <Template>
    <Name>
      A short description of what will be generated
    </Name>
    <TemplateFileName>
      A .txt file with the \Templates folder
    </TemplateFileName>
    <BaseClassName>
      Any base class name for the class in the template file name
    </BaseClassName>
    <OutputSubFolder>
      A sub-folder within the generation folder to place the output
    </OutputSubFolder>
    <OutputFileExtension>
      The extension for the generated file
    </OutputFileExtension>
    <OutputFilePrefix>
      Any prefix to add to the generated file
    </OutputFilePrefix>
    <OutputFileSuffix>
      Any suffix to add to the generated file
    </OutputFileSuffix>
    <UseClassNameAsFileName>
      If set to true, the currently generated class name is used as
the file name, plus any prefix and/or suffixes
    </UseClassNameAsFileName>
    <AppendToFile>
      If set to true, any tokens within the template are added to
the generated file.
    </AppendToFile>
  </Template>
</CodeGenTemplates>
```

Note that the <OutputSubFolder>, <OutputFilePrefix>, <OutputFileSuffix> elements may contain the following special tokens:

{|NAMESPACE|} = Uses the namespace

{|NAMESPACE_CLEAN|} = Uses the namespace cleaned of all characters that would make this an invalid file name

{|CLASS_NAME|} = Uses the class name

Other XML Files

Table 11 describes the XML files used by various tools within the PDSC Developer Utilities.

XML File	Description
ComputerCleaner-FilesToRecycle	A list of files to recycle.

ComputerCleaner-FoldersToRecycle	A list of folders to recycle.
LanguageDataTypes	A list of data types for each language.
Languages	A list of programming languages supported by the PDSC Developer Utilities.
LanguageScopes	A list of variable scopes used by each language.
ProjectCleaner-SourceControlTags	A list of source control tags to look for within .SLN or .CSPROJ files to remove during the Project Cleaning process.
PropertyGen-Templates	A list of templates that can be used for Property Generation.
XmlFileList	A list of all XML files within the PDSC Developer Utilities.

Table 11: Other XML files used by various PDSA Developer utilities.

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:IsForeignKey|}	Loop through all foreign key columns
{|FOR EACH COLUMN: IsNotForeignKey |}	Loop through all columns that are not foreign 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
{|FOR EACH VIEW|}	Loop through all selected views
{|END_LOOP|}	Each of the above must be terminated with this. Note, you can NOT nest {|FOR EACH |} tokens

Table 12: 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:IsTable|}	Removes the block of code when the current object being generated is a table
{|REMOVE_WHEN:IsView|}	Removes the block of code when the current object being generated is a view
{|REMOVE_WHEN:NoStandardFields|}	Removes the block of code when there are no standard fields in the table
{|REMOVE_WHEN:NoForeignKeys|}	Removes the block of code when there are no foreign keys 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: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
{|REMOVE_WHEN:Https|}	Removes the block of code when there the Use HTTPS for Web API project is true.
{|REMOVE_WHEN:NoHttps|}	Removes the block of code when there the Use HTTPS for Web API project is false.
<|REMOVE_WHEN:net7.0|>	Removes the block of code when the selected .NET Version is 7.0
<|REMOVE_WHEN:net8.0|>	Removes the block of code when the selected .NET Version is 8.0
{|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 13: 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
<|PROPERTY_NAME_WITH_VALUE_IF_NULL|>	Adds a ".Value" after the property name if the column is nullable
<|FIRST_SORT_FIELD|>	Returns the first sort field property name in the table
<|DESCRIPTION_FIELD|>	Returns the first description field property name in the table
<|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_INITIALIZER_STATEMENT|>	Returns the Initializer Statement that can be used after the {get;set;} statement to initialize the property. Comes from CodeGen-StandardFields.xml File
<|STD_PROPERTY_MODIFY|>	Returns the Modifier that can be used to set the value of a standard field in the Update() method. 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 14: Column Tokens.

Table/View Tokens

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

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
<|NAMESPACE_CLEAN|>	Returns the Namespace stripped of all commas, periods, or any other characters that would be an invalid name.
<|CLASS_NAME|> and {|CLASS_NAME|}	Returns the Class Name property for the current table
<|CLASS_NAME_ALL_LOWER|>	Returns the Class Name property for the current table as all lower-case letters

<|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_NAME_SINGULAR_ALL_LOWER|>	Returns the Singular version of the Description for the current table as all lower-case letters
<|CLASS_DESC_PLURAL|>	Returns the Pluralized version of the Description for the current table
<|CATALOG_NAME|>	Returns the Catalog for the current table
<|SCHEMA_NAME|>	Returns the Schema for the current table
<|TABLE_NAME|> or <|VIEW_NAME|>	Returns the Table or View name for the current table/view
<|SELECT_SQL|>	Returns the SQL used to select all columns for the current table

Table 15: Table Tokens.

Foreign Key Tokens

The following tokens are used for foreign key table information.

Token	Description
<|FK_REPOSITORIES_CONSTRUCTOR|>	Returns the FK table repository list for DI
<|PK_TABLE_CLASS_NAME_SINGULAR|>	Returns the Class Name for the FK table as a singular.
<|PK_TABLE_CLASS_NAME_PLURAL|>	Returns the Class Name for the FK table as a plural.
<|PK_TABLE_PK_PROPERTY_NAME|>	Returns the primary key property name for the FK table. This token is used in a loop.
<|FK_TABLE_DESC_FIELD_PROPERTY_NAME|>	Returns the description field property name for the FK table.
<|FK_TABLE_PK_FIELD_PROPERTY_NAME|>	Returns the primary key field property name for the FK table for the currently selected table and column.

Table 16: Foreign Key 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 17: 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.
<|GRID_ROW_AUTO_TABLE|>	Returns "Auto,Auto,.." for however many rows are needed for a display in a table.
<|GRID_ROW_AUTO_SEARCH|>	Returns "Auto,Auto,.." for however many rows are needed for the search area in the list view.
<|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.

Table 18: Screen Generation Tokens.

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
<|DOT_NET_VERSION|>	Inserts the moniker for the .NET version selected for generation
<|DOT_NET_VERSION:NuGetPackage|>	Looks for an <element> that matches the 'NuGetPackage' after the colon and returns the version number to use for that package when creating the projects
<|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_UPPER|>	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|>	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
<|SP_PREFIX|>	Returns the stored procedure prefix
<|SOLUTION_GUID|>	Returns a new Guid
{|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.
<|HARD_CODED_DATA|>	Iterates over all rows in a table and generates a List<T> of all data.
<|NO_HTTPS|>	If Use HTTPS flag is set to false, "--no-https" is added to the .cmd file that generates the Web API project.

Table 19: Miscellaneous Generation Tokens.

Generated Solutions

When the PDSC Developer Utilities generates code, you will find the solution is made up of several assemblies as shown in Figure 25.

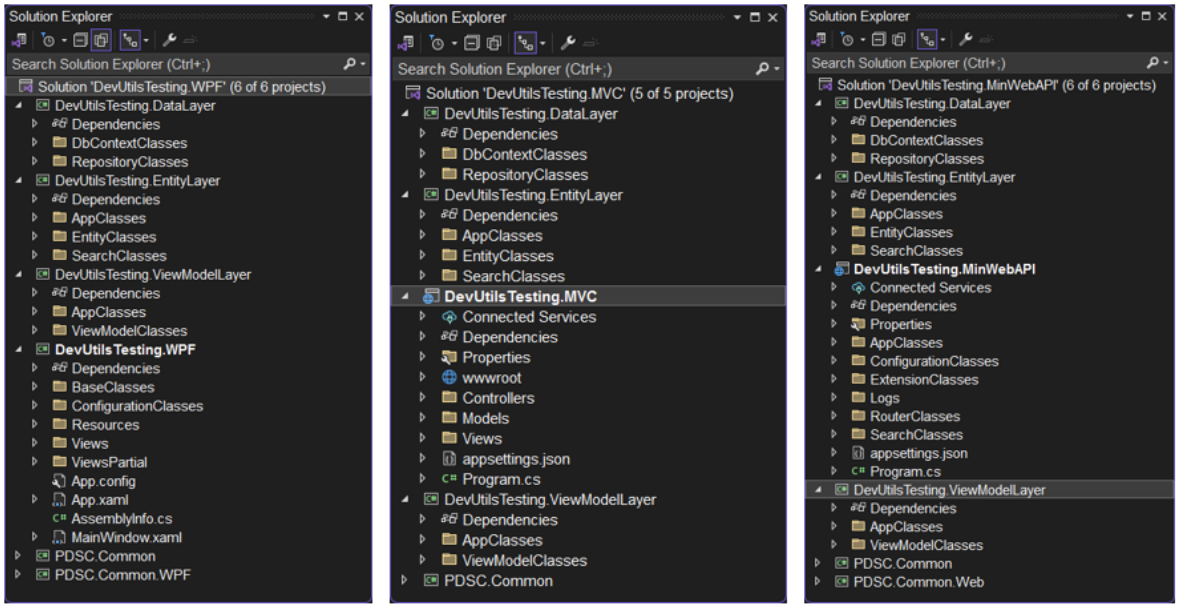

Figure 25: Sample of some generated solutions.

Table 20 describes each of the generated projects within the solution.

Project	Description
.DataLayer	Contains the Entity Framework DbContext class, and all repository classes.
.EntityLayer	Contains the entity classes that map to each table in your data store.
.ViewModelLayer	Contains the view model classes that use the repository and entity classes to build the views for your UI.
.WPF / .MVC / .MinWebAPI / .MAUI	These are the specific UI classes and views needed to create your application. These classes/views in this application should only use the View Model and Entity Classes and not rely on data layer except to inject the appropriate concrete implementation IRepository class into the View Model.
PDSC.Common	This assembly contains base classes that your repository, entity, router, controllers, and view model classes inherit from. It contains interfaces that many of these classes implement. In addition, there are some generic classes that help you with application development.
PDSC.MAUI	This class library is for any common classes that can be used within any .NET MAUI application.
PDSC.Web	This class library is for any common classes that can be used within any ASP.NET web application.
PDSC.WPF	This class library is for any common classes that can be used within any WPF application.

Table 20: A list of projects in the generated solutions.

Reusable Libraries

The ViewModelLayer, DataLayer, and EntityLayer are designed to be class libraries that can be reused with any UI layer you wish (Figure 26). You should not put references to any UI-specific classes or controls within any of these layers to ensure they can be reused over and over.

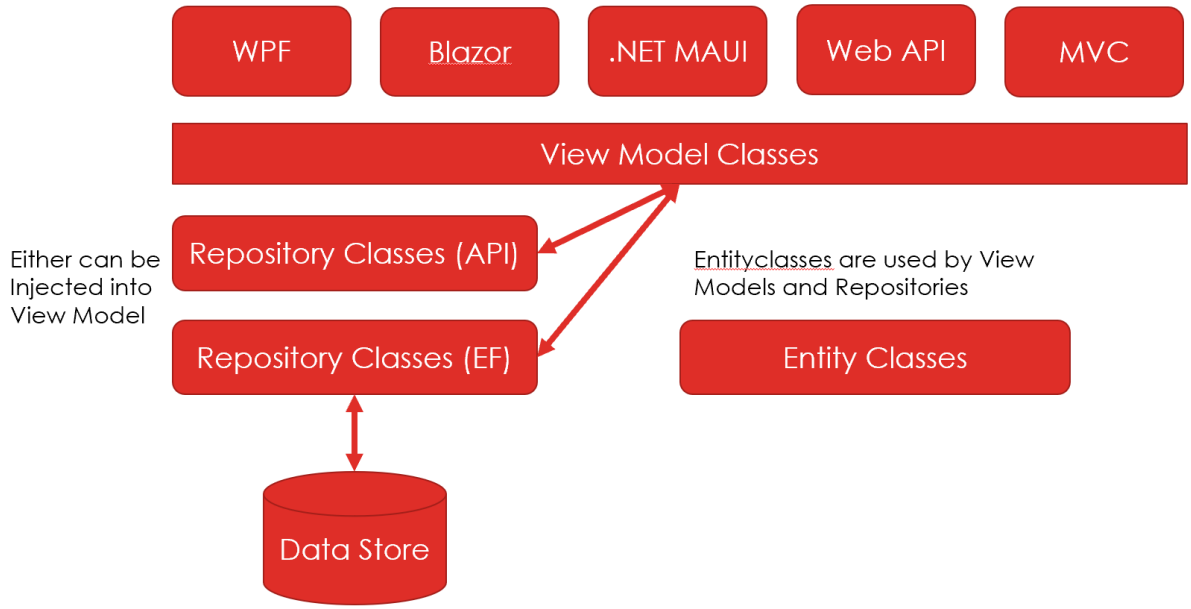

Figure 26: The ViewModelLayer, DataLayer, and EntityLayer assemblies are designed to be reusable.

Dependency Injection

Wherever possible, take advantage of Dependency Injection (DI) to inject concrete implementations of functionality (Figure 27). This ensures ultimate reusability of view models and repositories.

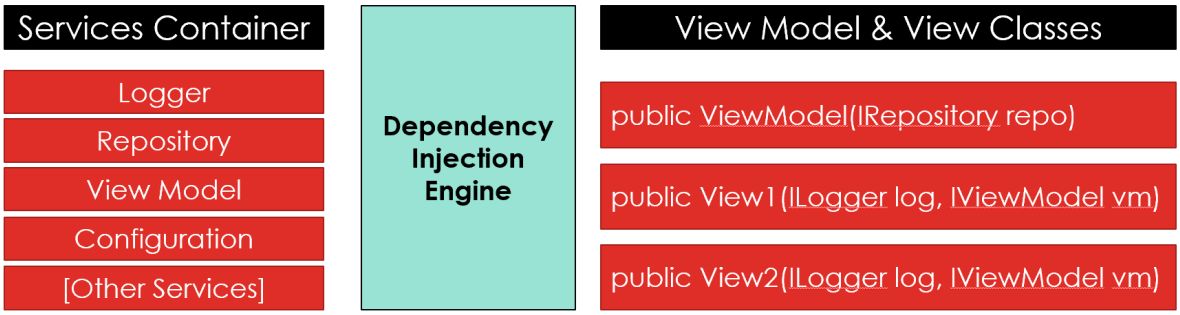

Figure 27: Use DI to inject services.

As an example, a .NET MAUI project the Repository classes that use API calls to a Web Server are injected into View Model classes. In the Web API project, the Repository classes that use Entity Framework are injected into the same View Model classes as shown in Figure 28.

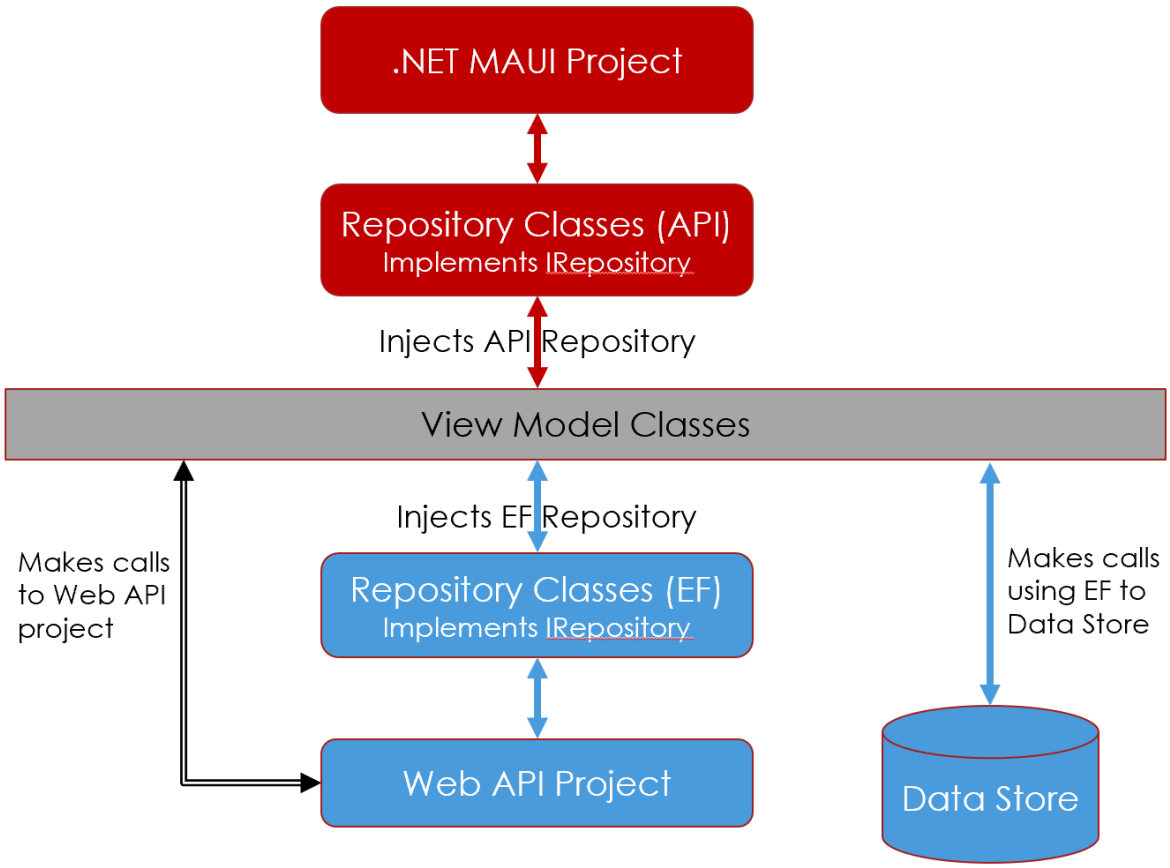

Figure 28: A .NET MAUI application injects Repository classes that use API calls, while a Web API project uses Repository classes that use EF to manipulate data.

PDSC Libraries

The PDSC Developer Utilities generates some common libraries that are meant to be used in any of your applications. The **PDSC.Common** library is a UI-agnostic library with classes designed to be used in any type of application. The **PDSC.Common.MAUI** library contains classes that can be utilized in any .NET MAUI application. The **PDSC.Common.Web** library contains classes that can be utilized in any ASP.NET application. The **PDSC.Common.WPF** library contains classes that can be utilized in any WPF application. The following sections describe each of these libraries.

PDSC Common Library

This assembly contains base classes that your repository, entity, router, controllers, and view model classes inherit from. It contains interfaces that many of these classes implement. In addition, there are some generic classes that help you with application development.

Interfaces

Table 21 describes the interfaces defined in this class library and what they are used for.

Class Name	Description
IRepository	All repository classes should inherit from this class.
IViewRepository	All repository classes that target Views in a database should inherit from this class.

Table 21: Various interfaces used in applications.

Miscellaneous Classes

Table 22 describes many of the classes you can use when writing applications.

Class Name	Description
DataResponseBase	A base class for the DataResponse class. Contains properties such as StatusCode, StatusMessage, RowsAffected, ResultMessage, LastException, and LastErrorMessage.
DataResponse	This class inherits from the DataResponseBase class and adds an additional property, Data. This Data property is where the payload is placed into when returning data from a Web API call. This Data, plus the other properties from the DataResponseBase class provide all the information needed to determine if a call was successful or not.
GenericsHelpers	Contains a static method to generically change one type into another.
HttpClientRepositoryBase	This class is a wrapper around the HttpClient class and is used to make Web API calls easier. All your RepositoryAPI classes should inherit from this class.
JsonHelper	Contains a static method to serialize any object into a string.
JWTSettings	This class holds the settings necessary for working with JSON Web Tokens (JWT).
PDSCException	This class holds properties related to any type of exception that can happen. If the exception is a database exception, there are many additional properties just related to the database exception.
PDSCExceptionManager	This class helps populate the PDSCException object with data around an exception. This is useful for logging exceptions.

ValidationError	Class used to convert a Dictionary<string, string[]> that comes from an MVC validation into a collection of ValidationMessage objects.
ValidationException	Class that inherits from the Exception class and it used in a catch block when a validation exception is thrown.
ValidationMessage	A class to hold the property name and validation message for why the property value fails its validation.

Table 22: Various classes used throughout almost all applications.

Base Classes

Table 23 lists the base classes you should ultimately be inheriting from within the classes in your application.

Class Name	Description
CommonBase	Almost all classes in your application will eventually inherit from this class. It contains properties that are very useful such as LastException, LastErrorMessage, UserName and many others.
RepositoryBase	Contains a RowsAffected (int) and ResponseObject (DataResponseBase). All your Repository classes should inherit from this class.
SearchBase	All search classes should inherit from this class.
SettingsBase	This class holds the common properties that are normally read in from the appsettings.json file of your application. Each of your application should have a settings class that inherits from this class. You can then add on additional properties as needed for your application.
ViewModelBase	All view models should inherit from this class, unless the view model classes work with Repository classes that use the Entity Framework. Then those view model classes should inherit from the ViewModelEFBase class.

Table 23: The set of base classes that other classes in your application inherit from.

PDSC Common MAUI Library

This class library is for any common classes that can be used within any .NET MAUI application. Table 24 lists the classes that you will find useful when developing .NET MAUI applications.

Class Name	Description
InvertedBoolConverter	A converter to change a true value to a false, or vice-versa.

Table 24: The set of classes that can be used in any .NET MAUI application.

PDSC Common Web Library

This class library is for any common classes that can be used within any ASP.NET web application. Table 25 lists the classes that you will find useful when developing ASP.NET Web applications.

Class Name	Description
RouterBase	All Minimal Web API router classes should inherit from this class.

Table 25: The set of classes that can be used in any ASP.NET web application.

PDSC Common WPF Library

This class library is for any common classes that can be used within any WPF application. Table 26 lists the classes that you will find useful when developing WPF applications.

Class Name	Description
BooleanToVisibilityConverter	Call this converter to change a true value to Visible and a false value to Collapsed.
BooleanToVisibilityHiddenConverter	Call this converter to change a true value to Visible and a false value to Hidden.
InvertedBoolConverter	A converter to change a true value to a false, or vice-versa.

Table 26: The set of classes that can be used in any WPF application.

This library also contains a set of standard colors and styles as described in Table 27.

XAML File	Description
Colors.xaml	A set of standard colors you can use in your WPF applications.
Styles	A set of standard styles you can use in your WPF applications.

Table 27: A standard set of resources to be used in any WPF application.

Generated Application Classes

When you use the PDSC Developer Utilities code generator, there are many classes generated for you. The following section describes the set of classes generated and what they inherit from.

Entity Classes and Inheritance Chain

An entity class is used to supply a mapping between a table in a database with a property in a C# class (Table 28).

Class Name	Description
Person, Customer, Etc.	Your entity class that contains a property for each column in a Person, Customer or other table in your data store. This class inherits from the **AppEntity** class.
AppEntity	A class in your EntityLayer that all your entity classes inherit from. This class inherits from the **CommonBase** class. This makes it easy to add properties or methods that you might need for just this specific application. A good example would be if you have a set of common fields on each table such as InsertDate, InsertName, UpdateDate, and UpdateName. Create the corresponding properties in this class so all your entity classes do not need to implement each of these.

Table 28: The inheritance chain for entity classes.

EF Repository Classes and Inheritance Chain

The Entity Framework Repository classes generated are what you use to read from any data store supported by the Entity Framework. The inheritance chain is described in Table 29.

Class Name	Description
PersonRespository, CustomerRespository, Etc.	This class contains methods to get all records, get a single record, and search for records. It also contains methods to count all records, or count records based on search criteria. This class also has methods for inserting, updating and deleting records. All methods in this class use the Entity Framework to access a data store. This class inherits from the **RepositoryBase** class and implements the **IRepository** interface.
RepositoryBase	This base class provides properties and methods to support all the functionality in your repository classes. The RepositoryBase class inherits from the **CommonBase** class.
IRepository<datatype, entity, entitySearch>	This interface ensures all repository classes adhere to the same set of properties and methods.

Table 29: The inheritance chain for Entity Framework repository classes.

API Repository Classes and Inheritance Chain

If your front-end UI (WPF, .NET MAUI, Blazor, etc.) requires you to interact with data via Web API calls, generate the API Repository classes. Table 30 describes the classes generated and the inheritance chain.

Class Name	Description
PersonRespositoryAPI, CustomerRespositoryAPI, Etc.	This class contains methods to get all records, get a single record, and search for records. It also contains methods to count all records, or count records based on search criteria. This class also has methods for inserting, updating and deleting records. All methods in this class use the HttpClient class to call a Web API to perform all the functionality required to work with the data. This class inherits from the **HttpClientRepositoryBase** class and implements the **IRepository** interface.
HttpClientRepositoryBase	This class is a wrapper around the .NET HttpClient class to help your RepositoryAPI classes make the Web API calls.
IRepository<datatype, entity, entitySearch>	This interface ensures all repository classes adhere to the same set of properties and methods.

Table 30: The inheritance chain for Web API repository classes.

View Model Classes and Inheritance Chain

All communication with your data store, whether through EF or API calls, should be done through View Model classes. The view model classes are designed to accept an IRepository interface through Dependency Injection. Table 31 describes the view model classes and the inheritance chain.

Class Name	Description
PersonViewModel, CustomerViewModel, etc.	The view model class is used to wrap up all functionality for working with pages/screens in your application. The data supplied to the view model comes from whichever Repository class is injected into this class. This class inherits from the **AppViewModel** class, and implements the **IViewModel** interface.
AppViewModel	This class is used in case you need to add specific functionality just for this one application. This class inherits from the **ViewModelBase** class. The ViewModelBase class inherits from the **CommonBase** class.
IViewModel	This interface ensures all view model classes adhere to the same set of properties and methods.

Table 31: The inheritance chain for view model classes.

Summary

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