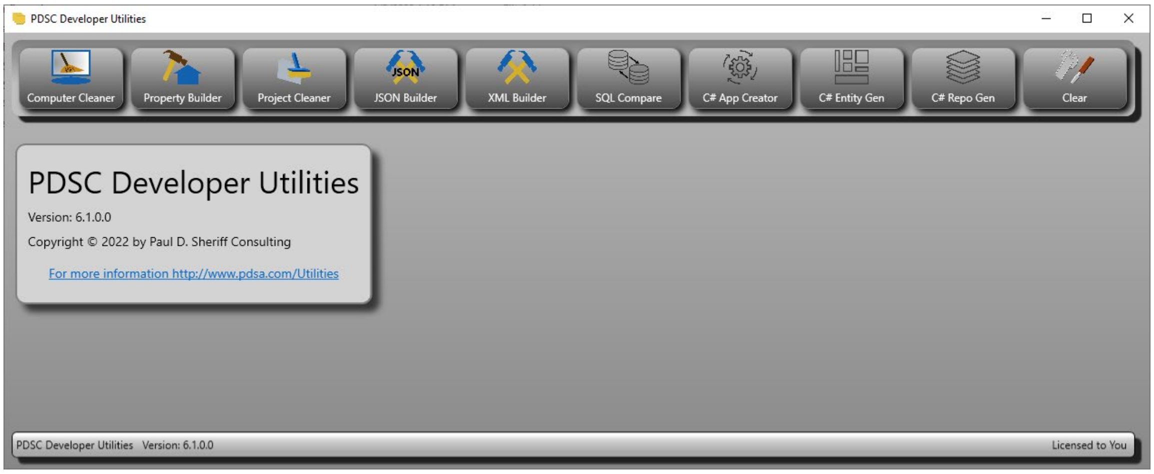
Chapter 2: PDSC Developer Utilities Usage

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



PDSC Developer Utilities

Version: 6.1.0.0

Copyright © 2022 by Paul D. Sheriff Consulting

For more information http://www.pdsa.com/Utilities

PDSC Developer Utilities Version: 6.1.0.0 Licensed to You

Figure 1: Screen shot of the PDSC Developer Utilities

Overview of the Developer Utilities

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

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

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

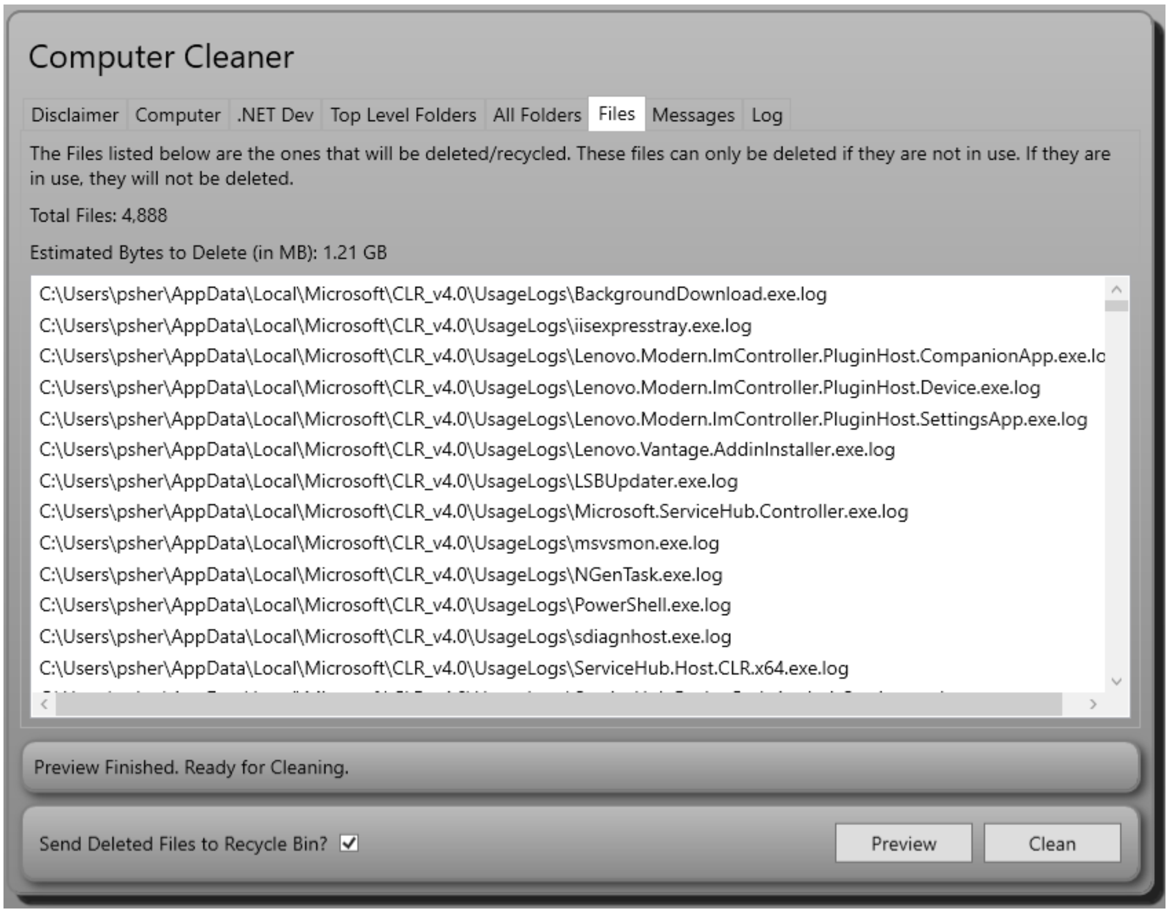
Table 1. List of PDSC Developer Utilities

Computer Cleaner

Visual Studio and Visual Studio Code are two great development environments for creating applications quickly. However, both leave a lot of miscellaneous files all over your hard drive. There are a few locations on your hard drive that you should check to see if there are left-over folders or files that you can delete. I have attempted to gather as much data as I can about the various versions of .NET and operating systems. Of course, your mileage may vary on the folders and files listed here. This utility attempts to find the various folders depending on which version(s) of Visual Studio/VS Code you have installed on your machine.

Preview

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



The Files listed below are the ones that will be deleted/recycled. These files can only be deleted if they are not in use. If they are in use, they will not be deleted.

Total Files: 4,888

Estimated Bytes to Delete (in MB): 1.21 GB

```
C:\Users\psher\AppData\Local\Microsoft\CLR_v4.0\UsageLogs\BackgroundDownload.exe.log
C:\Users\psher\AppData\Local\Microsoft\CLR_v4.0\UsageLogs\iisexpresstray.exe.log
C:\Users\psher\AppData\Local\Microsoft\CLR_v4.0\UsageLogs\Lenovo.Modern.ImController.PluginHost.CompanionApp.exe.lo
C:\Users\psher\AppData\Local\Microsoft\CLR_v4.0\UsageLogs\Lenovo.Modern.ImController.PluginHost.Device.exe.log
C:\Users\psher\AppData\Local\Microsoft\CLR_v4.0\UsageLogs\Lenovo.Modern.ImController.PluginHost.SettingsApp.exe.log
C:\Users\psher\AppData\Local\Microsoft\CLR_v4.0\UsageLogs\Lenovo.Vantage.AddinInstaller.exe.log
C:\Users\psher\AppData\Local\Microsoft\CLR_v4.0\UsageLogs\LSBUpdater.exe.log
C:\Users\psher\AppData\Local\Microsoft\CLR_v4.0\UsageLogs\Microsoft.ServiceHub.Controller.exe.log
C:\Users\psher\AppData\Local\Microsoft\CLR_v4.0\UsageLogs\msvsmon.exe.log
C:\Users\psher\AppData\Local\Microsoft\CLR_v4.0\UsageLogs\NGenTask.exe.log
C:\Users\psher\AppData\Local\Microsoft\CLR_v4.0\UsageLogs\PowerShell.exe.log
C:\Users\psher\AppData\Local\Microsoft\CLR_v4.0\UsageLogs\sdiagnhost.exe.log
C:\Users\psher\AppData\Local\Microsoft\CLR_v4.0\UsageLogs\ServiceHub.Host.CLR.x64.exe.log
```

Preview Finished. Ready for Cleaning.

Send Deleted Files to Recycle Bin? ✓

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

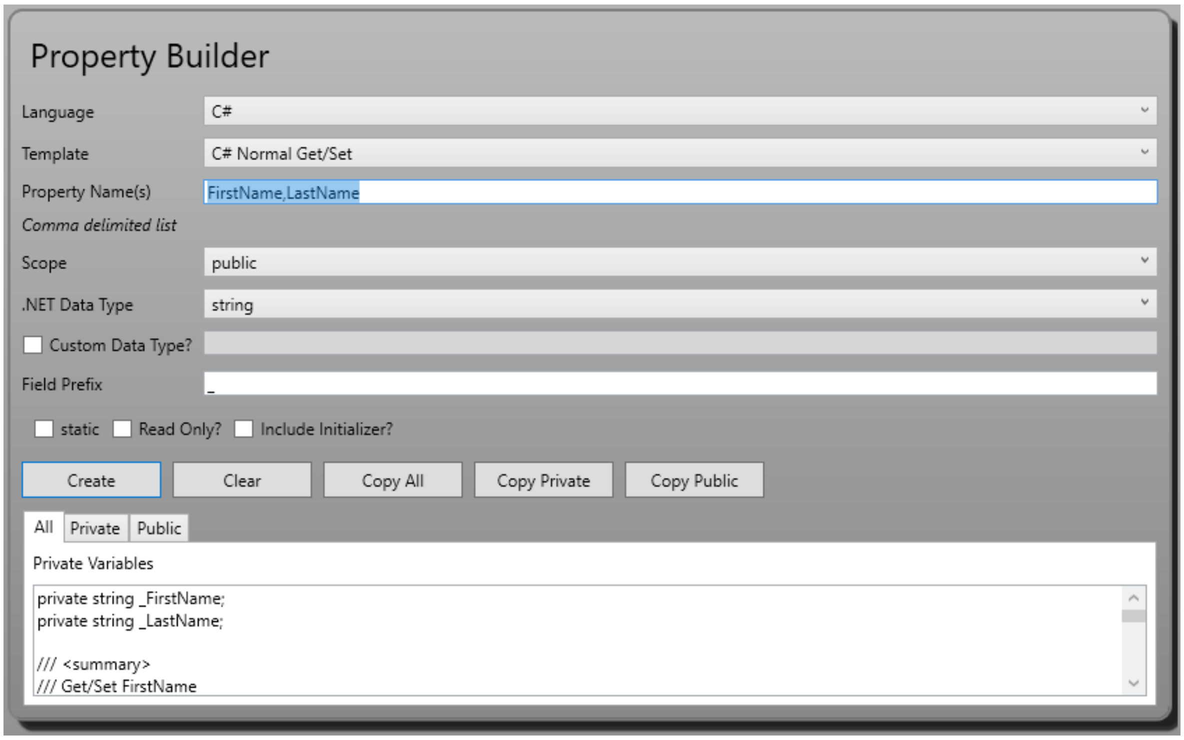
Each version of VS.NET will create "temporary" files in different folders. The problem is that the files created are not always "temporary". Most of these files do not get cleaned up like they should.

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

Property Builder

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

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


Property Builder

Language: C#
Template: C# Normal Get/Set
Property Name(s): FirstName,LastName
Comma delimited list
Scope: public
.NET Data Type: string
☐ Custom Data Type?
Field Prefix: _
☐ static ☐ Read Only? ☐ Include Initializer?

Private Variables

```
private string _FirstName;
private string _LastName;

/// <summary>
/// Get/Set FirstName
```

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

Adding Your Own Templates

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

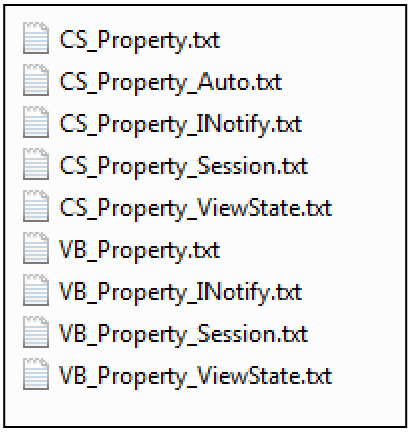

CS_Property.txt
CS_Property_Auto.txt
CS_Property_INotify.txt
CS_Property_Session.txt
CS_Property_ViewState.txt
VB_Property.txt
VB_Property_INotify.txt
VB_Property_Session.txt
VB_Property_ViewState.txt

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

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

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

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

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

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

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

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

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

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

Property Builder Tokens

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

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

Table 2. List of Tokens in Property Builder

Other XML files for the Property Builder

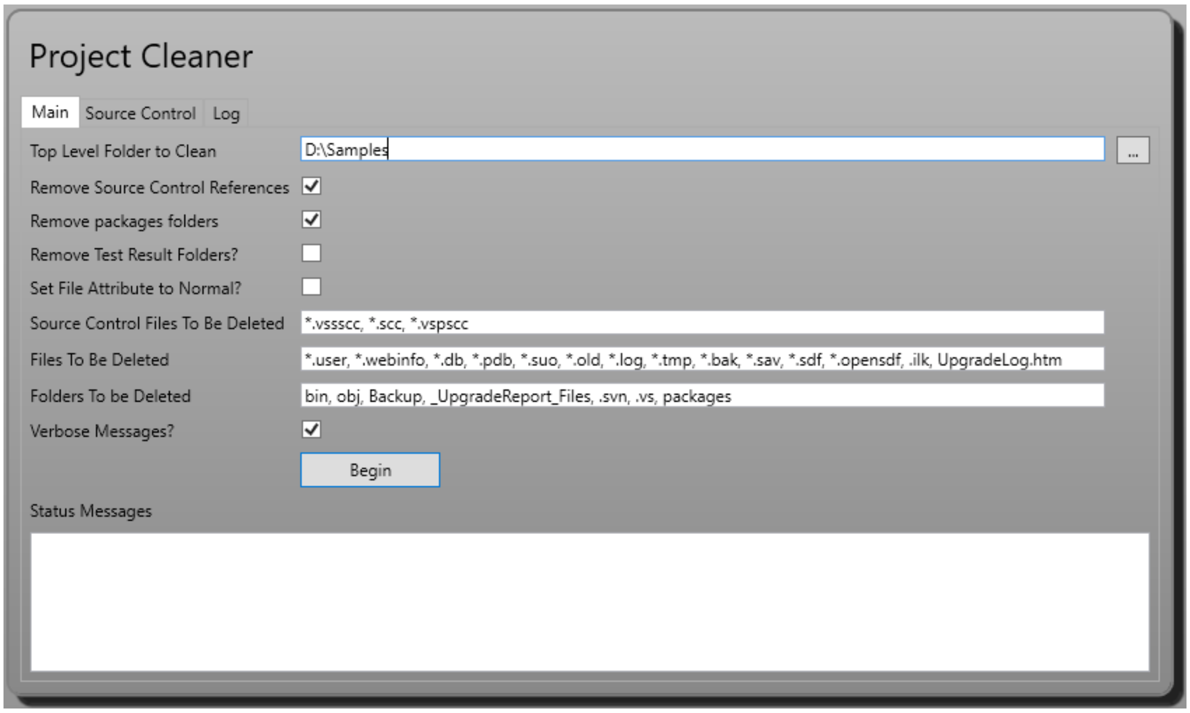
There are a few other XML files that the property builder uses to assist with the generation. These are located in the \Xml folder under the location where you installed the Developer Utilities.

Xml File name	Description
DataTypesCS	A list of data types for C#.
DataTypesVB	A list of data types for Visual Basic.
DotNetLanguages	The list of .NET languages.
ScopeCS	A list of scopes for C#.
ScopeVB	A list of scopes for Visual Basic.

Table 3. List of XML files for the Property Builder

Project Cleaner

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


Project Cleaner

Top Level Folder to Clean: D:\Samples
Remove Source Control References: ✓
Remove packages folders: ✓
Remove Test Result Folders?: ☐
Set File Attribute to Normal?: ☐
Source Control Files To Be Deleted: *.vssscc, *.scc, *.vspscc
Files To Be Deleted: *.user, *.webinfo, *.db, *.pdb, *.suo, *.old, *.log, *.tmp, *.bak, *.sav, *.sdf, *.opensdf, .ilk, UpgradeLog.htm
Folders To be Deleted: bin, obj, Backup, _UpgradeReport_Files, .svn, .vs, packages
Verbose Messages?: ✓
Status Messages

Figure 5: Clean up files using the Project Cleaner utility

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

Field	**Description**
Top Level Folder to Clean	Enter the top level folder you wish to iterate through
Remove Source Control References	Check this is you wish this utility to open your .SLN and any .csproj or .vbproj files and remove the source control tags from these files.
Remove packages folder	Remove the "packages" folder.
Remove Test Result folders	Remove any test result folders.
Source Files to be Deleted	A list of source control file extensions that should be removed. The files with these extensions will only be deleted if the Remove Source Control References is checked.
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.
Verbose Messages?	Display verbose messages while cleaning. NOTE: The utility will run much faster with this unchecked.

Table 4: Fields to fill in for cleaning projects.

In addition to the above items, this utility will also remove read-only attributes from all files.

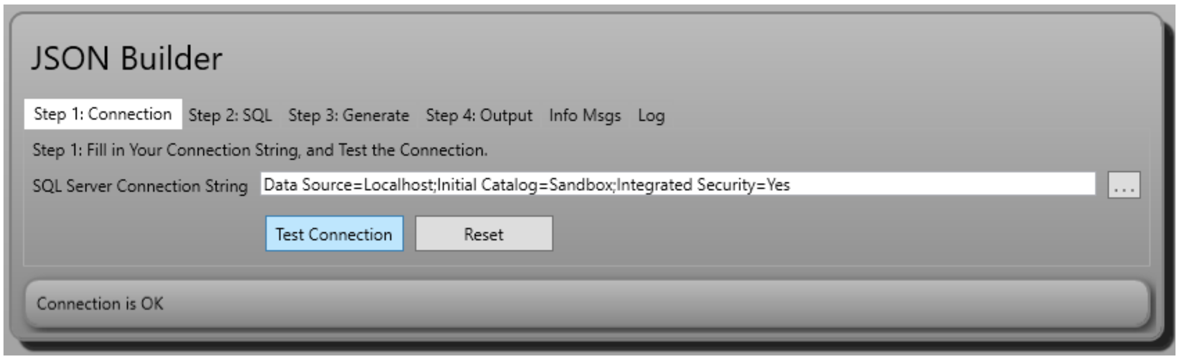
NOTE:	This utility only goes thru the folder and sub-folders specified in the **Top Level Folder** field. If the solution in the top-level folder points to another project in another folder structure, that other project will NOT have any of its attributes reset, or its source control references removed.

JSON Builder

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

Step 1: Connection

To start the JSON generation process, put in the appropriate connection string that will connect you to your database (Figure 6). Click on the Test Connection button to ensure that you can reach the database.



Step 1: Fill in Your Connection String, and Test the Connection.

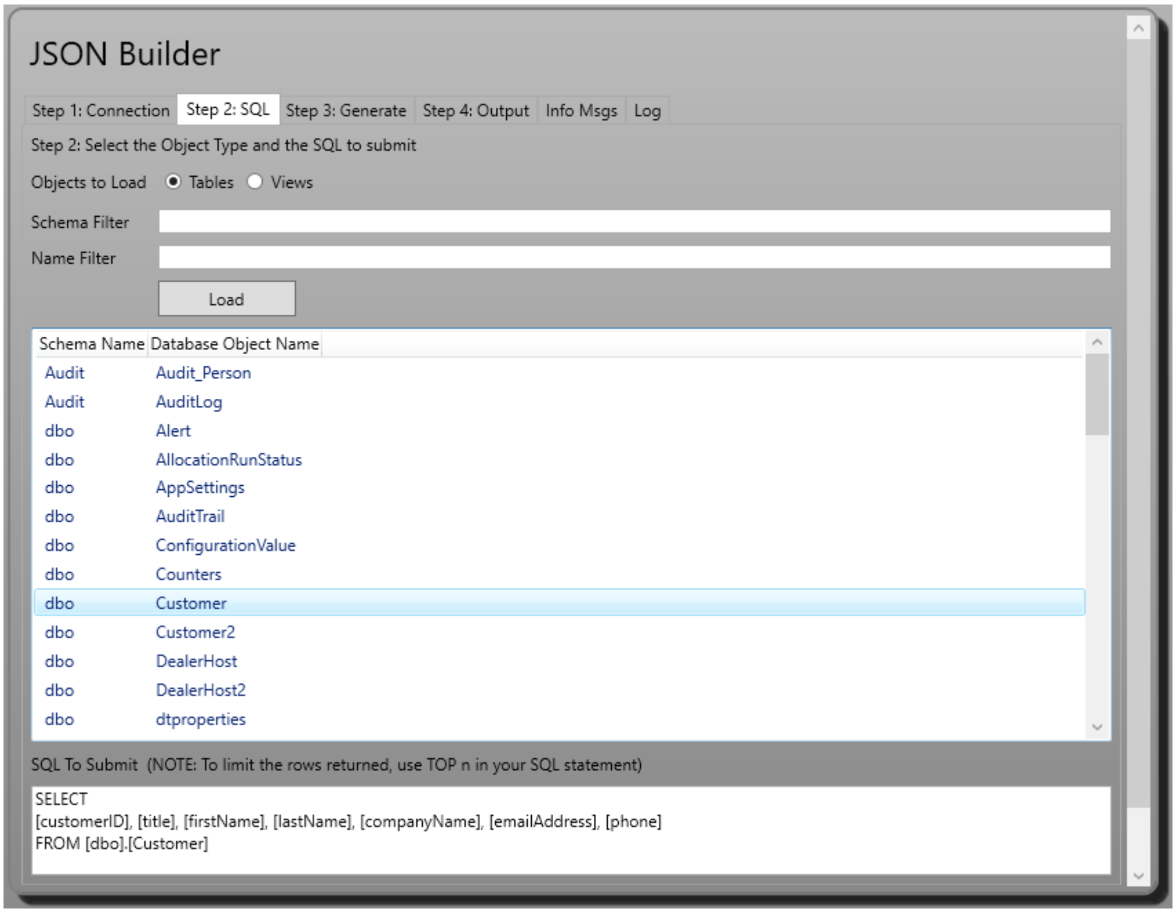
SQL Server Connection String: Data Source=Localhost;Initial Catalog=Sandbox;Integrated Security=Yes

Connection is OK

Figure 6: Step 1: JSON Builder Connection Tab

Step 2: SQL

Click on the **Step 2: SQL** tab (Figure 7) to select a table or view to generate JSON from. First 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 name (or partial schema name), and/or an object name (or partial object name) prior to clicking on the Load button.



Step 2: Select the Object Type and the SQL to submit

Objects to Load [✓ Tables] Views

Schema Filter

Name Filter

Schema Name	Database Object Name
Audit	Audit_Person
Audit	AuditLog
dbo	Alert
dbo	AllocationRunStatus
dbo	AppSettings
dbo	AuditTrail
dbo	ConfigurationValue
dbo	Counters
dbo	Customer
dbo	Customer2
dbo	DealerHost
dbo	DealerHost2
dbo	dtproperties

SQL To Submit (NOTE: To limit the rows returned, use TOP n in your SQL statement)

```
SELECT
[customerID], [title], [firstName], [lastName], [companyName], [emailAddress], [phone]
FROM [dbo].[Customer]
```

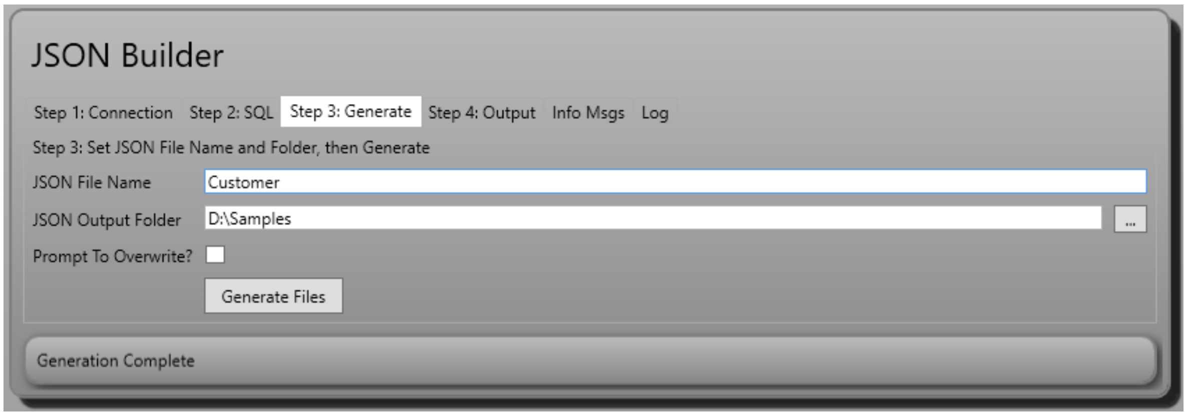
Figure 7: Step 2: JSON Builder SQL Tab

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

Step 3: Generate

Click on **Step 3: Generate** (Figure 8) to fill in information on how you wish to generate the JSON file. You can specify the name of the file and the folder for the JSON file.

A ".json" file extension will automatically be added to the file name. You will be prompted to overwrite this file if you check the **Prompt to Overwrite?** check box.



JSON Builder

Step 3: Set JSON File Name and Folder, then Generate

JSON File Name: Customer

JSON Output Folder: D:\Samples

Prompt To Overwrite? ☐

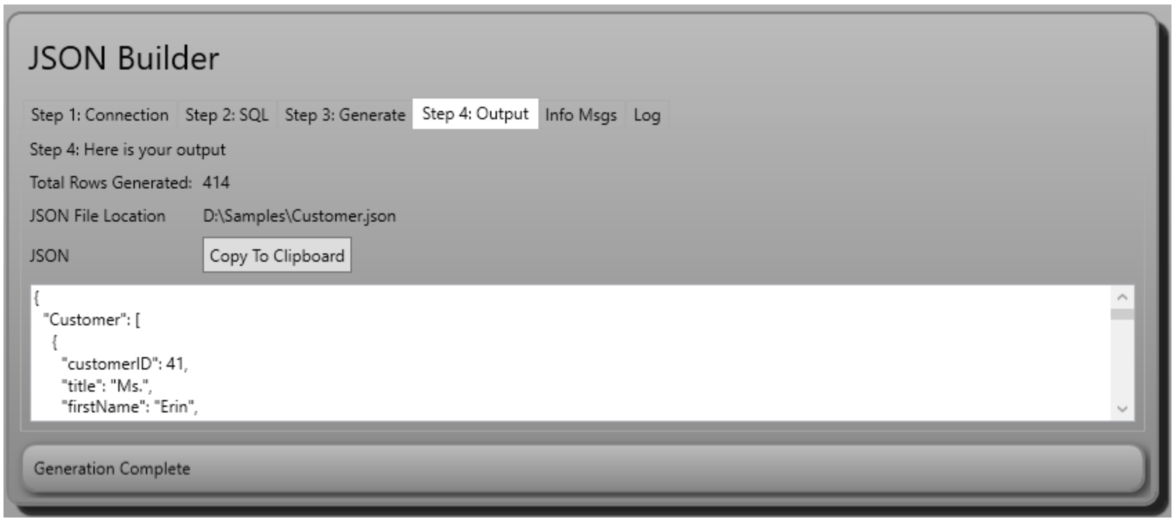
Generation Complete

Figure 8: Step 3: JSON Builder Generate Tab

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

Step 4: Output

After you click on the **Generate Files** (Figure 9) button, you are presented with the screen shown in Figure 9. This screen tells you how many rows were generated, where the JSON file is located.



JSON Builder

Step 4: Here is your output

Total Rows Generated: 414

JSON File Location: D:\Samples\Customer.json

JSON

```
{
  "Customer": [
    {
      "customerID": 41,
      "title": "Ms.",
      "firstName": "Erin",
```

Generation Complete

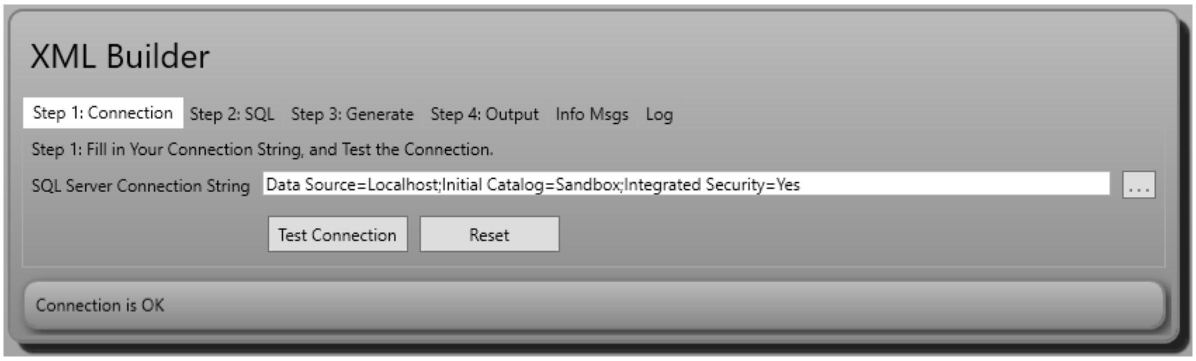
Figure 9: Step 4 JSON Builder Output tab

XML Builder

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

Step 1: Connection

To start the XML generation process, put in the appropriate connection string that will connect you to your database (Figure 10). Click on the Test Connection button to ensure that you can reach the database.



Step 1: Fill in Your Connection String, and Test the Connection.

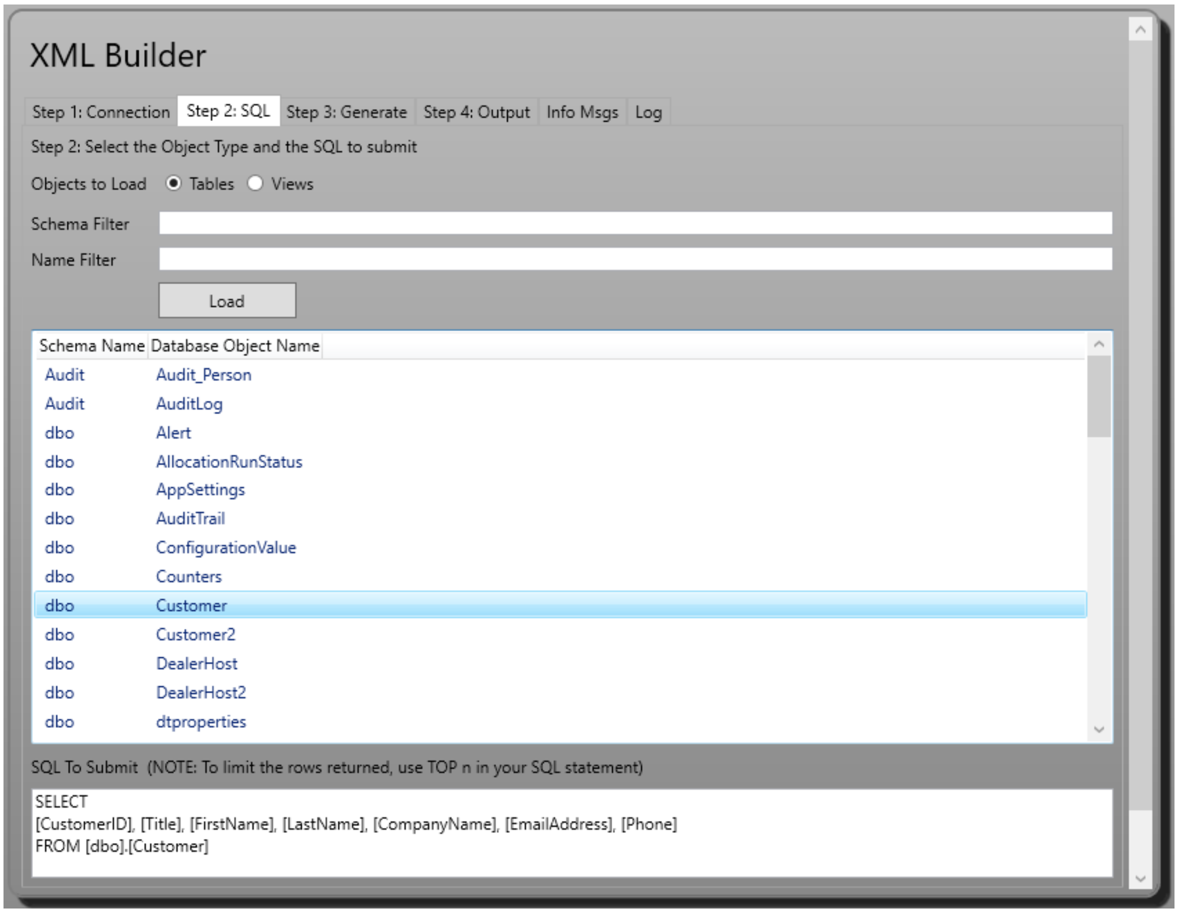
SQL Server Connection String: Data Source=Localhost;Initial Catalog=Sandbox;Integrated Security=Yes

Connection is OK

Figure 10: Step 1: XML Builder Connection Tab

Step 2: SQL

Click on the **Step 2: SQL** (Figure 11) tab to select a table or view to generate XML from. First 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 name (or partial schema name), and/or an object name (or partial object name) prior to clicking on the Load button.



Step 2: Select the Object Type and the SQL to submit

Objects to Load [✓ Tables] Views

Schema Filter

Name Filter

Schema Name	Database Object Name
Audit	Audit_Person
Audit	AuditLog
dbo	Alert
dbo	AllocationRunStatus
dbo	AppSettings
dbo	AuditTrail
dbo	ConfigurationValue
dbo	Counters
dbo	Customer
dbo	Customer2
dbo	DealerHost
dbo	DealerHost2
dbo	dtproperties

SQL To Submit (NOTE: To limit the rows returned, use TOP n in your SQL statement)

```
SELECT
[CustomerID], [Title], [FirstName], [LastName], [CompanyName], [EmailAddress], [Phone]
FROM [dbo].[Customer]
```

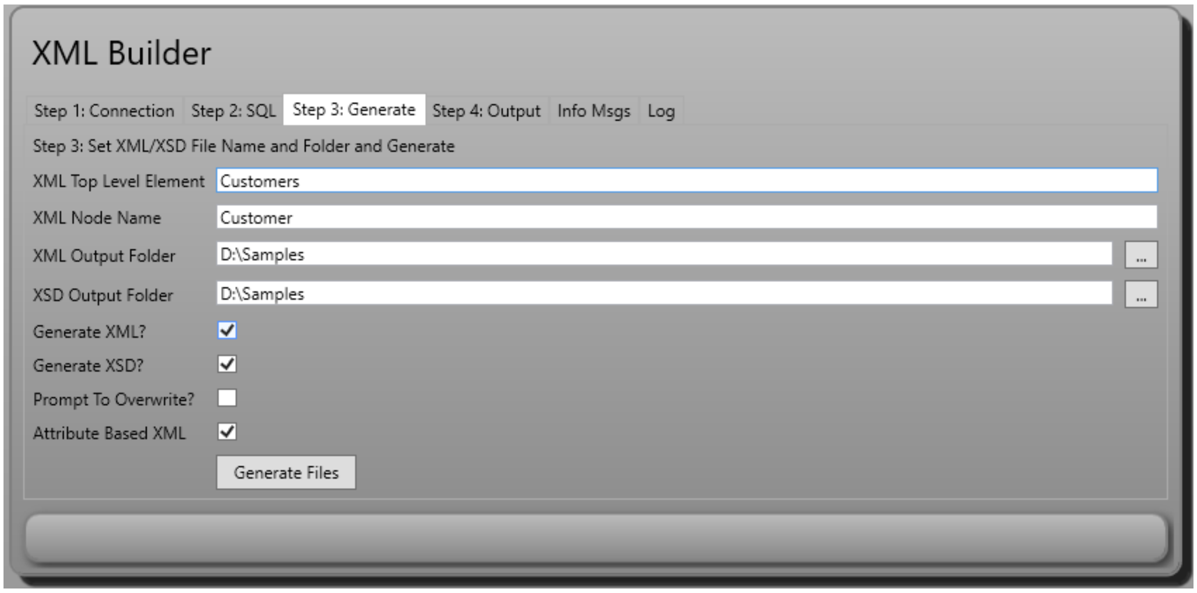
Figure 11: Step 2: XML Builder SQL Tab

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

Step 3: Generate

Click on **Step 3: Generate** (Figure 12) to fill in information on how you wish to generate the XML file. You can specify your top level element name, and for each row the XML node name to use. You then specify the folder for the XML file and, optionally, the XSD file.

Check the **Generate XML?** to generate an XML file and check the **Generate XSD?** to generate an XSD file for the query you will be submitting. The file name generated will be the same as the XML Top Level Element name. You will be prompted to overwrite this file if you check the **Prompt To Overwrite?** check box. The last option you can specify is whether or not you want element or attribute-based XML to be generated.



Step 3: Set XML/XSD File Name and Folder and Generate

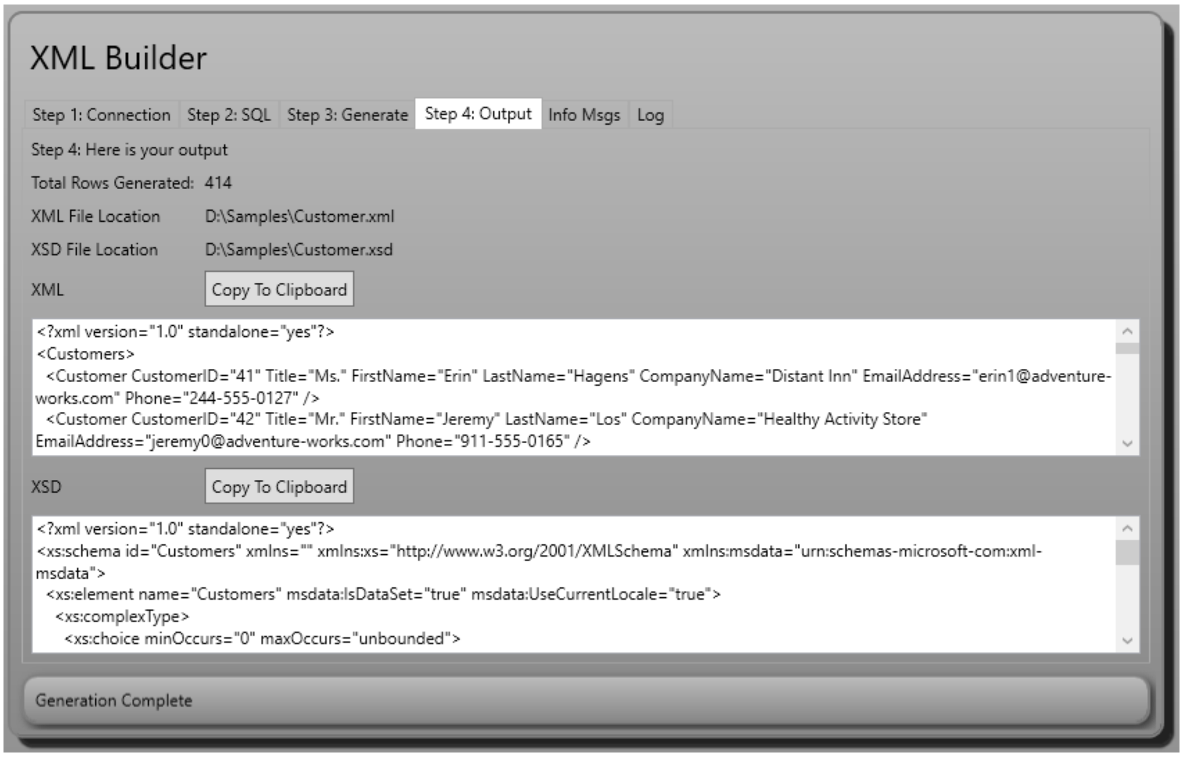
XML Top Level Element: Customers
XML Node Name: Customer
XML Output Folder: D:\Samples
XSD Output Folder: D:\Samples
Generate XML? ✓
Generate XSD? ✓
Prompt To Overwrite? ☐
Attribute Based XML ✓

Figure 12: Step 3: XML Builder Generate Tab

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

Step 4: Output

After you click on the **Generate Files** button, you will be presented with the screen shown in Figure 13. This screen tells you how many rows were generated, where the files are located and allows you to copy the XML and/or the XSD information to the clipboard.



Step 4: Here is your output

Total Rows Generated: 414

XML File Location D:\Samples\Customer.xml

XSD File Location D:\Samples\Customer.xsd

XML

```
<?xml version="1.0" standalone="yes"?>
<Customers>
  <Customer CustomerID="41" Title="Ms." FirstName="Erin" LastName="Hagens" CompanyName="Distant Inn" EmailAddress="erin1@adventure-
works.com" Phone="244-555-0127" />
  <Customer CustomerID="42" Title="Mr." FirstName="Jeremy" LastName="Los" CompanyName="Healthy Activity Store"
EmailAddress="jeremy0@adventure-works.com" Phone="911-555-0165" />
```

XSD

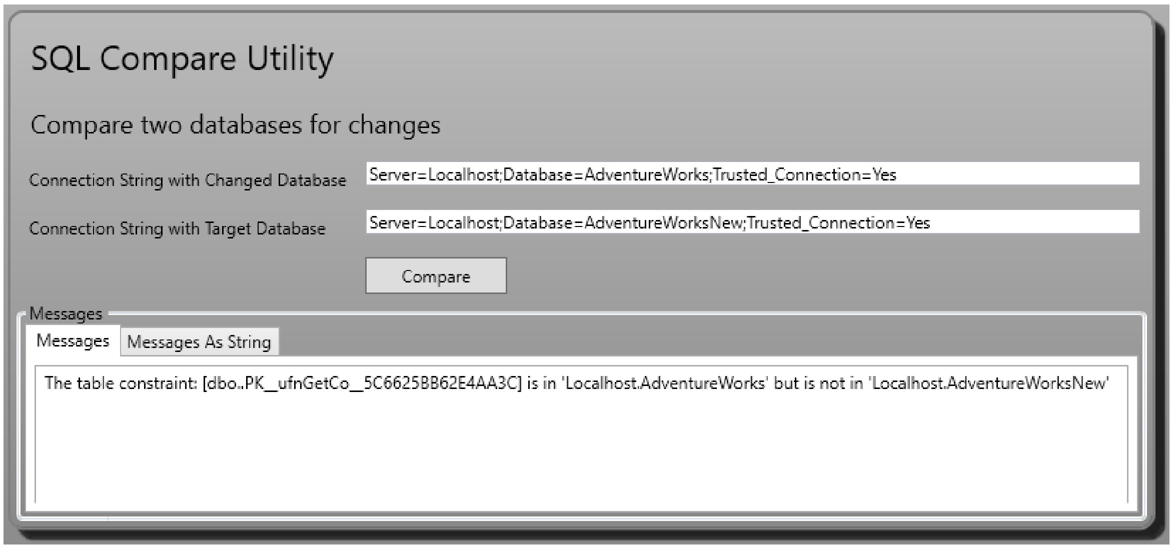
```
<?xml version="1.0" standalone="yes"?>
<xs:schema id="Customers" xmlns="" xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:msdata="urn:schemas-microsoft-com:xml-
msdata">
  <xs:element name="Customers" msdata:IsDataSet="true" msdata:UseCurrentLocale="true">
    <xs:complexType>
      <xs:choice minOccurs="0" maxOccurs="unbounded">
```

Generation Complete

Figure 13: Step 4: XML Builder: Output tab

SQL Compare

When you run the SQL Compare utility, you put in two different connections string that point to similar databases. For example, maybe you need to find out what you changed in your QA database compared to your Production database. Click the Compare button (shown in Figure 14) and a complete list of missing or changed objects will appear in the messages tabs at the bottom of the screen.



SQL Compare Utility

Compare two databases for changes

Connection String with Changed Database: Server=Localhost;Database=AdventureWorks;Trusted_Connection=Yes

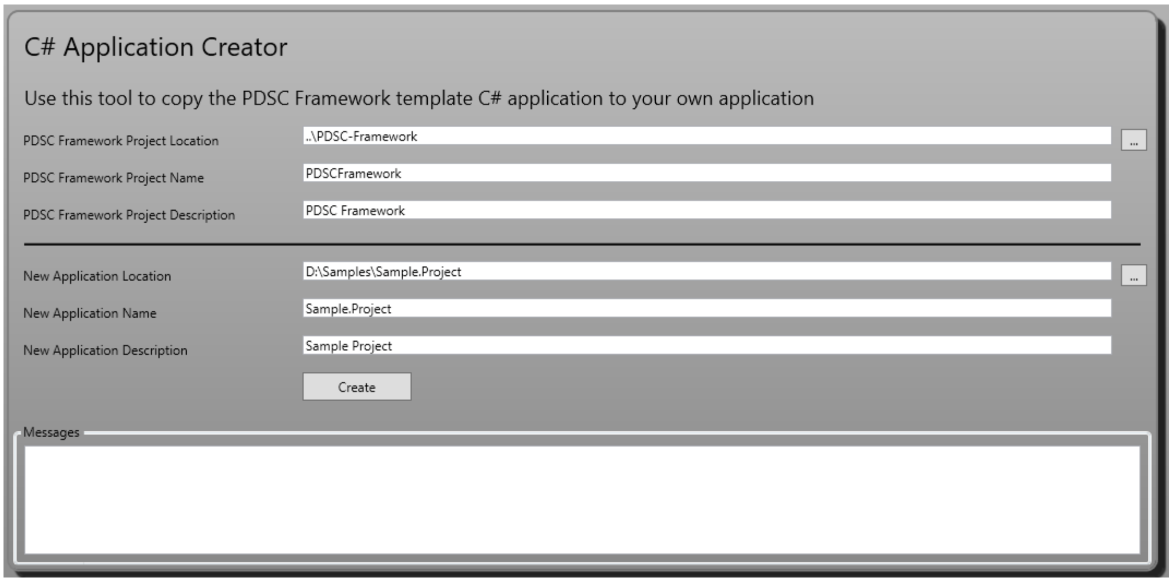
Connection String with Target Database: Server=Localhost;Database=AdventureWorksNew;Trusted_Connection=Yes

```
The table constraint: [dbo..PK__ufnGetCo__5C6625BB62E4AA3C] is in 'Localhost.AdventureWorks' but is not in 'Localhost.AdventureWorksNew'
```

Figure 14: Get missing objects from one database to another via the SQL Compare Utility.

C# Application Creator

Click on the C# App Creator menu to see a screen that looks like Figure 15. Modify the "New Application Location" to a valid hard drive, and folder, on your system and click the **Create** button. If you get an error that one of the paths is incorrect, fix it up, then click the Create button again. In just a few seconds you should receive a message that the process is complete, and you will see a bunch of messages as seen at the bottom of the screen.



C# Application Creator

Use this tool to copy the PDSC Framework template C# application to your own application

PDSC Framework Project Location	..\PDSC-Framework
PDSC Framework Project Name	PDSCFramework
PDSC Framework Project Description	PDSC Framework
New Application Location	D:\Samples\Sample.Project
New Application Name	Sample.Project
New Application Description	Sample Project

Figure 15: The PDSC C# Application Creator helps you build a new MVC project.

You can now go to the "New Application Location" folder and view the results of running this tool as shown in Figure 16.

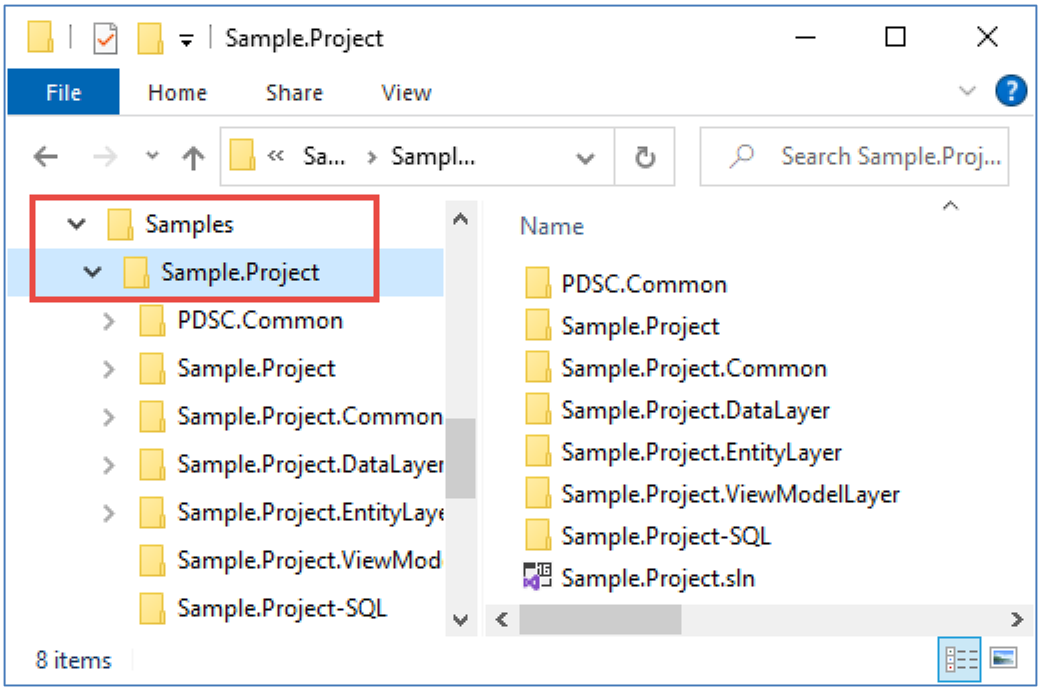


Figure 16: In the Application Location folder you find a folder structure like that of the PDSC Framework template.

Create a Sample Database

Open the SQL Server Management Studio and create a new database named **Sample.Project**. Open the "Sample.Project-SQL" folder and locate the **Sample.Project.sql** file (Figure 17) and load that file into SQL Server Management Studio. Run this script to create the database objects. Open the **Sample.Project-Data.sql** file in SQL Server Management Studio and run this script to add data to the database objects.

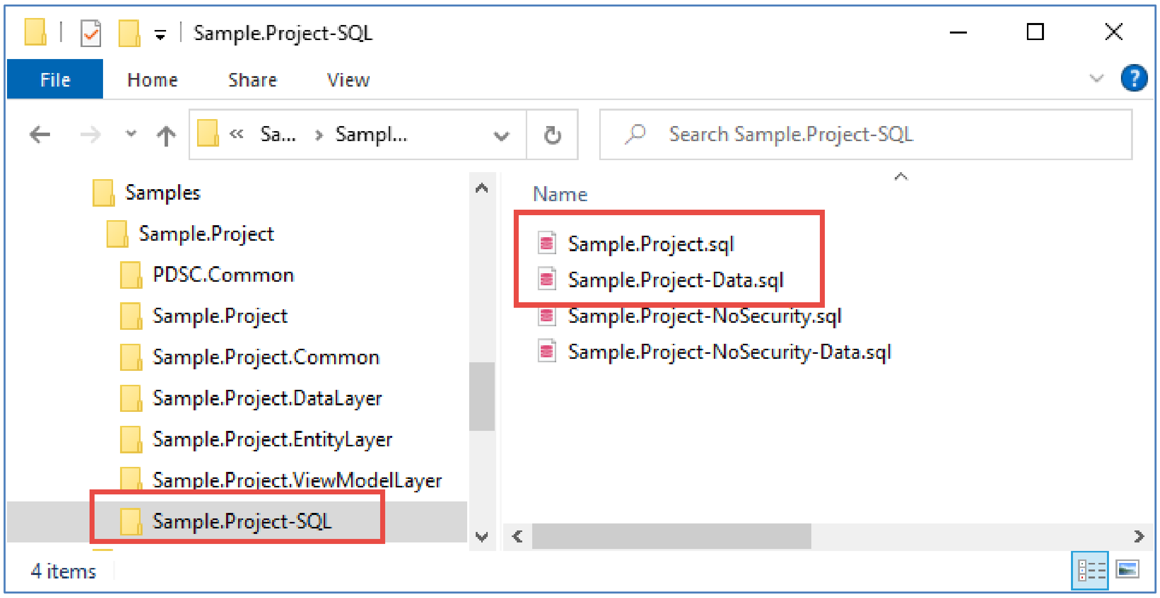


Figure 17: Locate the .SQL files to install in your new application folder.

Run the Sample Project

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

Next Steps

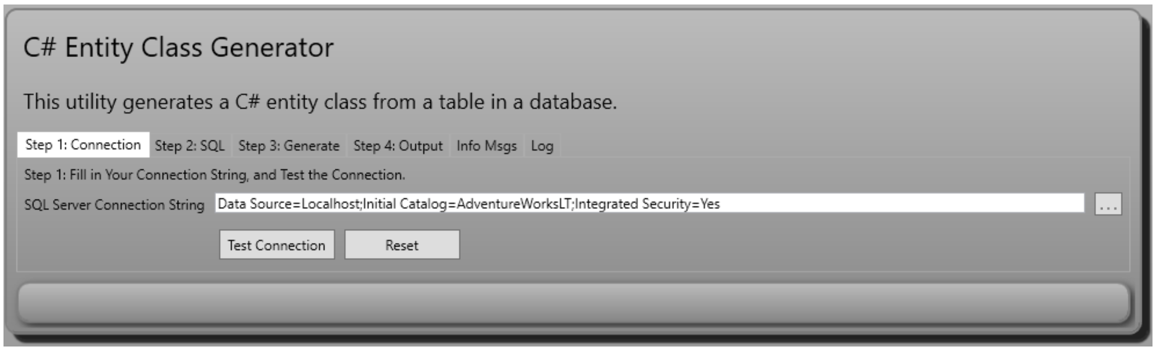
Please read the chapter on the **Haystack Code Generator** for information on how to generate add/edit/delete pages for your new project.

C# Entity Generator

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

Step 1: Connection

To start the C# entity class generation process, put in the appropriate connection string that will connect you to your database (Figure 18). Click on the Test Connection button to ensure that you can reach the database.



C# Entity Class Generator

This utility generates a C# entity class from a table in a database.

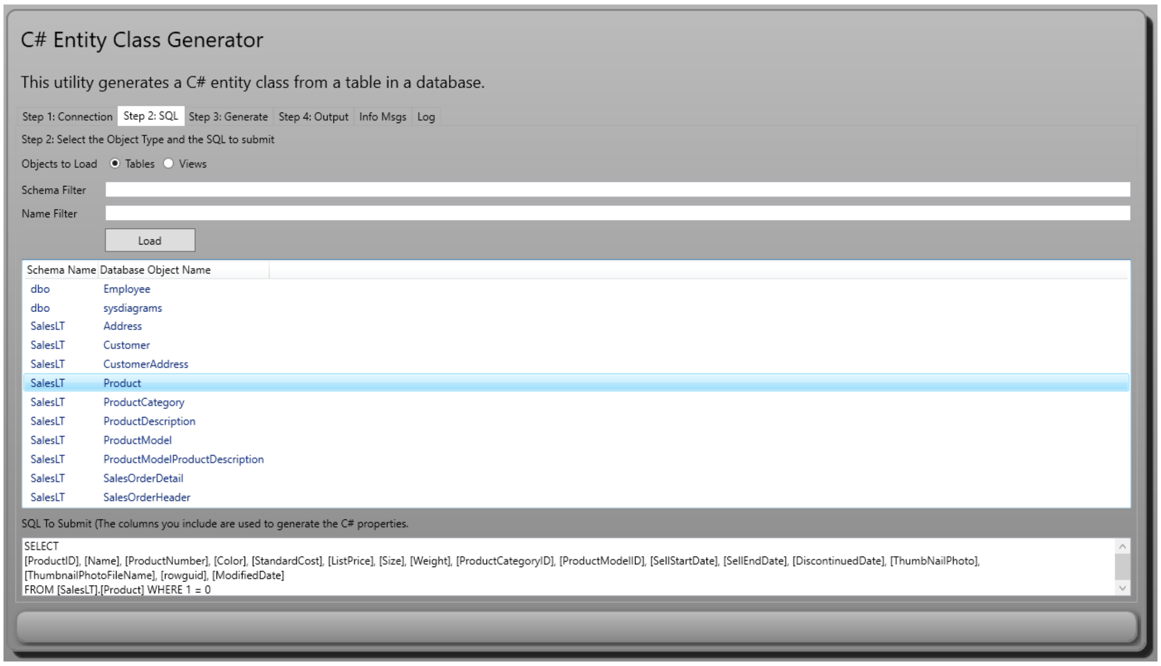
Step 1: Fill in Your Connection String, and Test the Connection.

SQL Server Connection String: Data Source=Localhost;Initial Catalog=AdventureWorksLT;Integrated Security=Yes

Figure 18: Step 1: C# Entity Class Generator Connection Tab

Step 2: SQL

Click on the **Step 2: SQL** (Figure 19) tab to select a table or view to generate an entity class from. First 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 name (or partial schema name), and/or an object name (or partial object name) prior to clicking on the Load button.



C# Entity Class Generator

This utility generates a C# entity class from a table in a database.

Step 2: Select the Object Type and the SQL to submit

Objects to Load [✓ Tables] Views

Schema Filter

Name Filter

Schema Name	Database Object Name
dbo	Employee
dbo	sysdiagrams
SalesLT	Address
SalesLT	Customer
SalesLT	CustomerAddress
SalesLT	Product
SalesLT	ProductCategory
SalesLT	ProductDescription
SalesLT	ProductModel
SalesLT	ProductModelProductDescription
SalesLT	SalesOrderDetail
SalesLT	SalesOrderHeader

SQL To Submit (The columns you include are used to generate the C# properties.

```
SELECT
[ProductID], [Name], [ProductNumber], [Color], [StandardCost], [ListPrice], [Size], [Weight], [ProductCategoryID], [ProductModelID], [SellStartDate], [SellEndDate], [DiscontinuedDate], [ThumbNailPhoto],
[ThumbnailPhotoFileName], [rowguid], [ModifiedDate]
FROM [SalesLT].[Product] WHERE 1 = 0
```

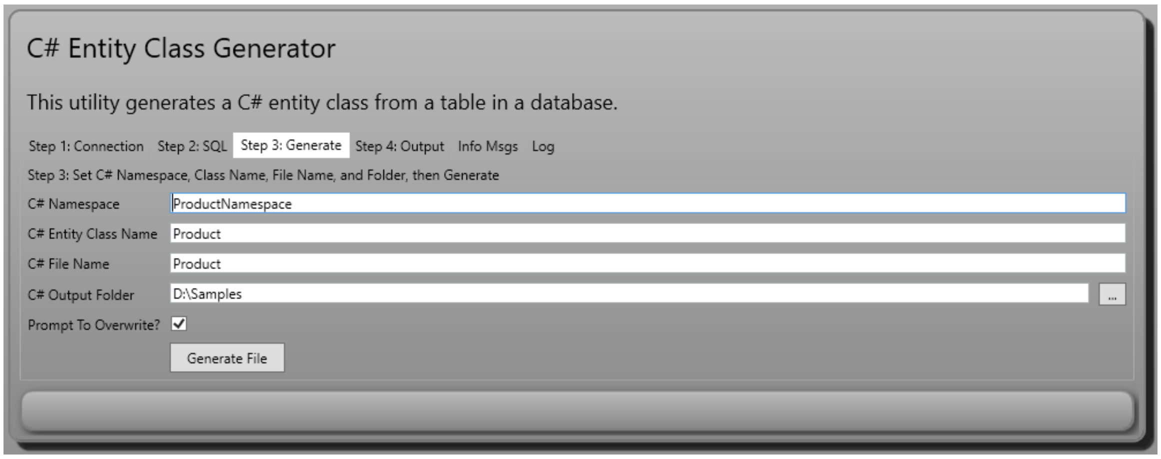
Figure 19: Step 2: C# Entity Class Generator SQL Tab

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

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

Step 3: Generate

Click on **Step 3: Generate** (Figure 20) to fill in information on how you wish to generate the C# entity class. Fill in the Namespace to use, the entity class name, the file name, and the output folder. Check the **Prompt to Overwrite?** check box if you want to be prompted before overwriting a previously written file. Click the **Generate File** button to start the generation process.



This utility generates a C# entity class from a table in a database.

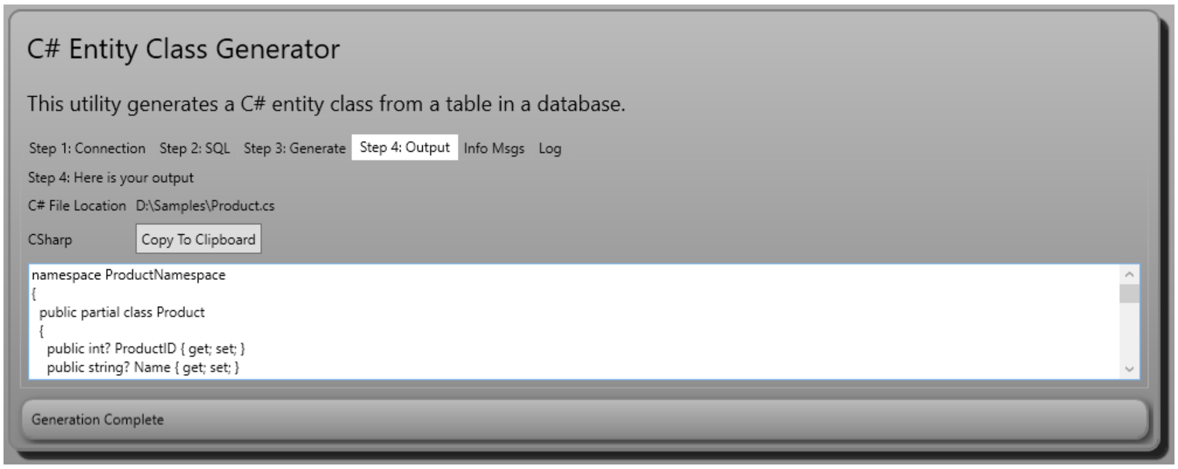
Step 3: Set C# Namespace, Class Name, File Name, and Folder, then Generate

C# Namespace	ProductNamespace
C# Entity Class Name	Product
C# File Name	Product
C# Output Folder	D:\Samples
Prompt To Overwrite?	✓

Figure 20: Step 3: C# Entity Class Generator Generate Tab

Step 4: Output

After you click on the **Generate File** button, you will be presented with the screen shown in Figure 21. This screen tells you where the C# entity file is located and allows you to copy the entity class to the clipboard.



C# Entity Class Generator

This utility generates a C# entity class from a table in a database.

Step 4: Here is your output

C# File Location D:\Samples\Product.cs

CSharp

```
namespace ProductNamespace
{
  public partial class Product
  {
    public int? ProductID { get; set; }
    public string? Name { get; set; }
```

Generation Complete

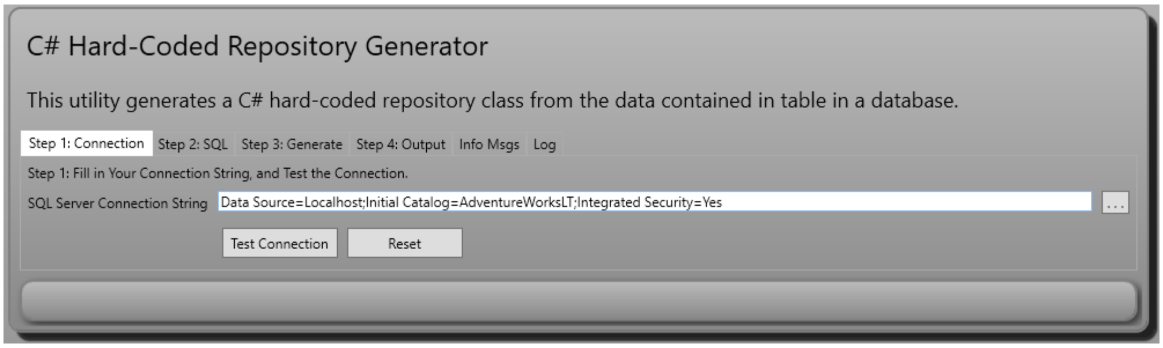
Figure 21: Step 4: C# Entity Class Generator Output tab

C# Repository Generator

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

Step 1: Connection

To start the C# hard-coded repository class generation process, put in the appropriate connection string that will connect you to your database (Figure 22). Click on the Test Connection button to ensure that you can reach the database.



C# Hard-Coded Repository Generator

This utility generates a C# hard-coded repository class from the data contained in table in a database.

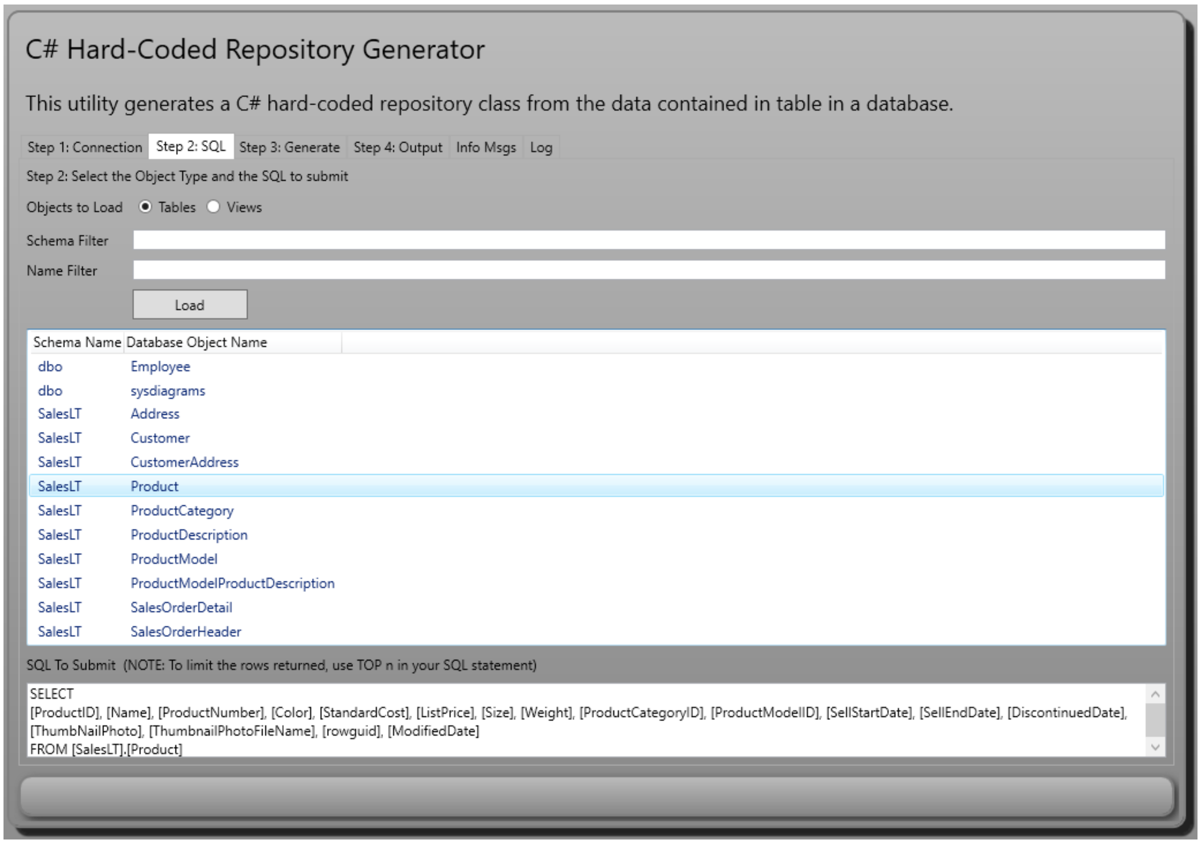
Step 1: Fill in Your Connection String, and Test the Connection.

SQL Server Connection String: Data Source=Localhost;Initial Catalog=AdventureWorksLT;Integrated Security=Yes

Figure 22: Step 1: C# Repository Class Generator Connection Tab

Step 2: SQL

Click on the **Step 2: SQL** (Figure 23) tab to select a table or view to generate a repositoryy class from. First 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 name (or partial schema name), and/or an object name (or partial object name) prior to clicking on the Load button.


C# Hard-Coded Repository Generator

This utility generates a C# hard-coded repository class from the data contained in table in a database.

Step 2: Select the Object Type and the SQL to submit

Objects to Load [✓ Tables] Views

Schema Filter

Name Filter

Schema Name	Database Object Name
dbo	Employee
dbo	sysdiagrams
SalesLT	Address
SalesLT	Customer
SalesLT	CustomerAddress
SalesLT	Product
SalesLT	ProductCategory
SalesLT	ProductDescription
SalesLT	ProductModel
SalesLT	ProductModelProductDescription
SalesLT	SalesOrderDetail
SalesLT	SalesOrderHeader

SQL To Submit (NOTE: To limit the rows returned, use TOP n in your SQL statement)

```
SELECT
[ProductID], [Name], [ProductNumber], [Color], [StandardCost], [ListPrice], [Size], [Weight], [ProductCategoryID], [ProductModelID], [SellStartDate], [SellEndDate], [DiscontinuedDate],
[ThumbNailPhoto], [ThumbnailPhotoFileName], [rowguid], [ModifiedDate]
FROM [SalesLT].[Product]
```

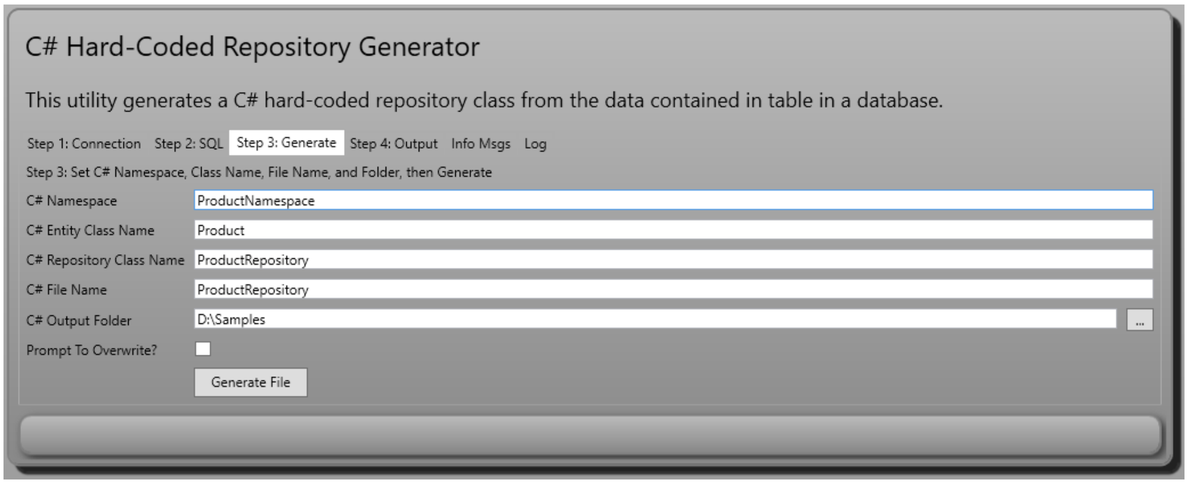
Figure 23: Step 2: C# Repository Class Generator SQL Tab

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

You might also consider adding a *SELECT TOP 50* instead of generating for all rows in your table.

Step 3: Generate

Click on **Step 3: Generate** (Figure 24) 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, the file name, and the output folder. Check the **Prompt to Overwrite?** check box if you want to be prompted before overwriting a previously written file. Click the **Generate File** button to start the generation process.



This utility generates a C# hard-coded repository class from the data contained in table in a database.

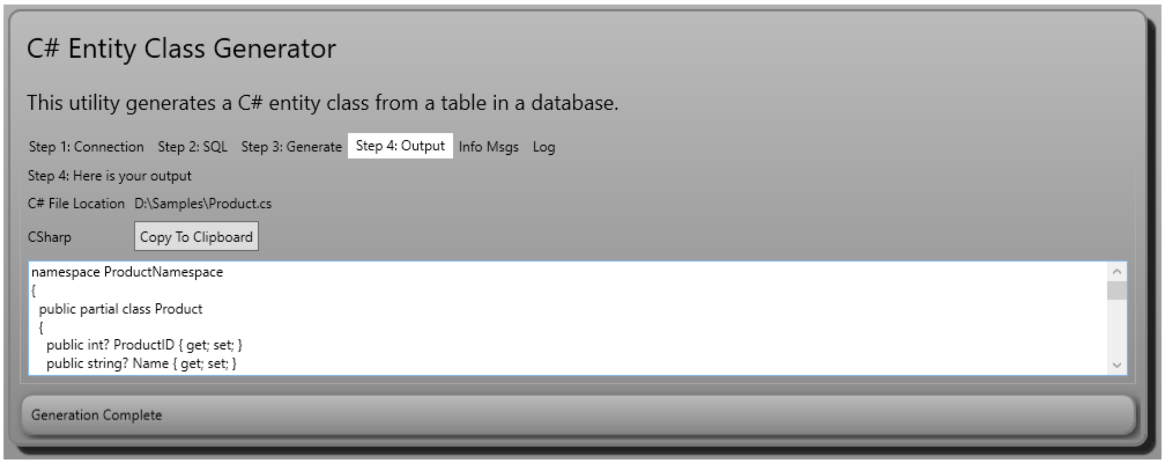
Step 3: Set C# Namespace, Class Name, File Name, and Folder, then Generate

C# Namespace	ProductNamespace
C# Entity Class Name	Product
C# Repository Class Name	ProductRepository
C# File Name	ProductRepository
C# Output Folder	D:\Samples
Prompt To Overwrite?	☐

Figure 24: Step 3: C# Repository Class Generator Generate Tab

Step 4: Output

After you click on the **Generate File** button, you will be presented with the screen shown in Figure 25. This screen tells you where the C# repository file is located and allows you to copy the entity class to the clipboard.



This utility generates a C# entity class from a table in a database.

Step 4: Here is your output

C# File Location D:\Samples\Product.cs

```
namespace ProductNamespace
{
  public partial class Product
  {
    public int? ProductID { get; set; }
    public string? Name { get; set; }
```

Generation Complete

Figure 25: Step 4: C# Repository Class Generator Output tab

Summary

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