Chapter 11

Haystack Code Generator Reference

Table of Contents

Code Generation

This chapter will describe many of the various features of the Haystack Code Generator for .NET (Figure 1). The first few pages describe the Project Information screen in detail as this has information that you will need to use Haystack effectively. After learning about the Project Information screen, a scenario based narrative will describe all the various ways you can generate code with Haystack.

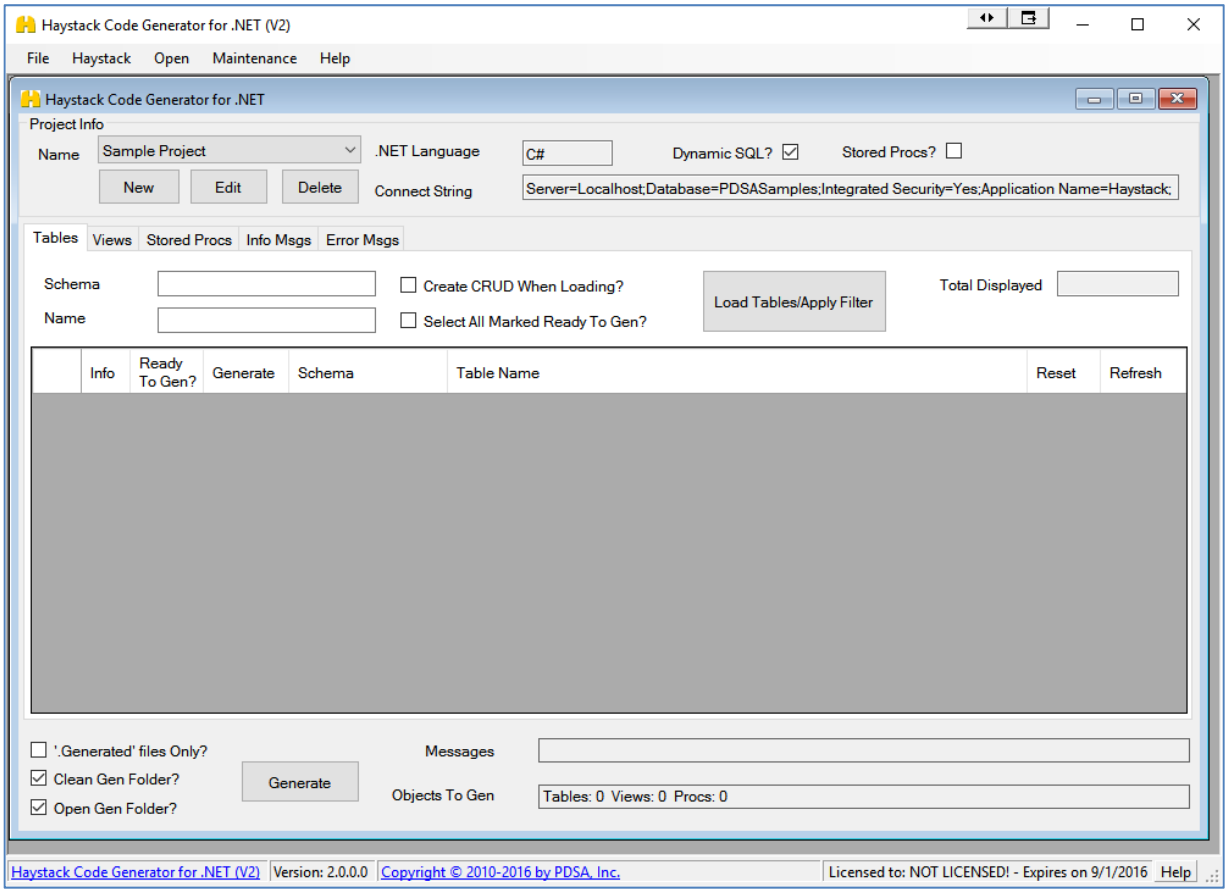

Figure 1: Haystack Main Screen

After starting Haystack you will need to create a new Project. A project points to one (and only one) database. A project is where you will setup a lot of defaults for all your generated classes. Please pay attention to the following pages as they are important for learning what you can generate.

Project Information Screen

The Project Information screen allows you to add new projects, or change the settings of current projects. After you have generated, or saved a table, view or stored procedure, you may be unable to modify certain project properties.

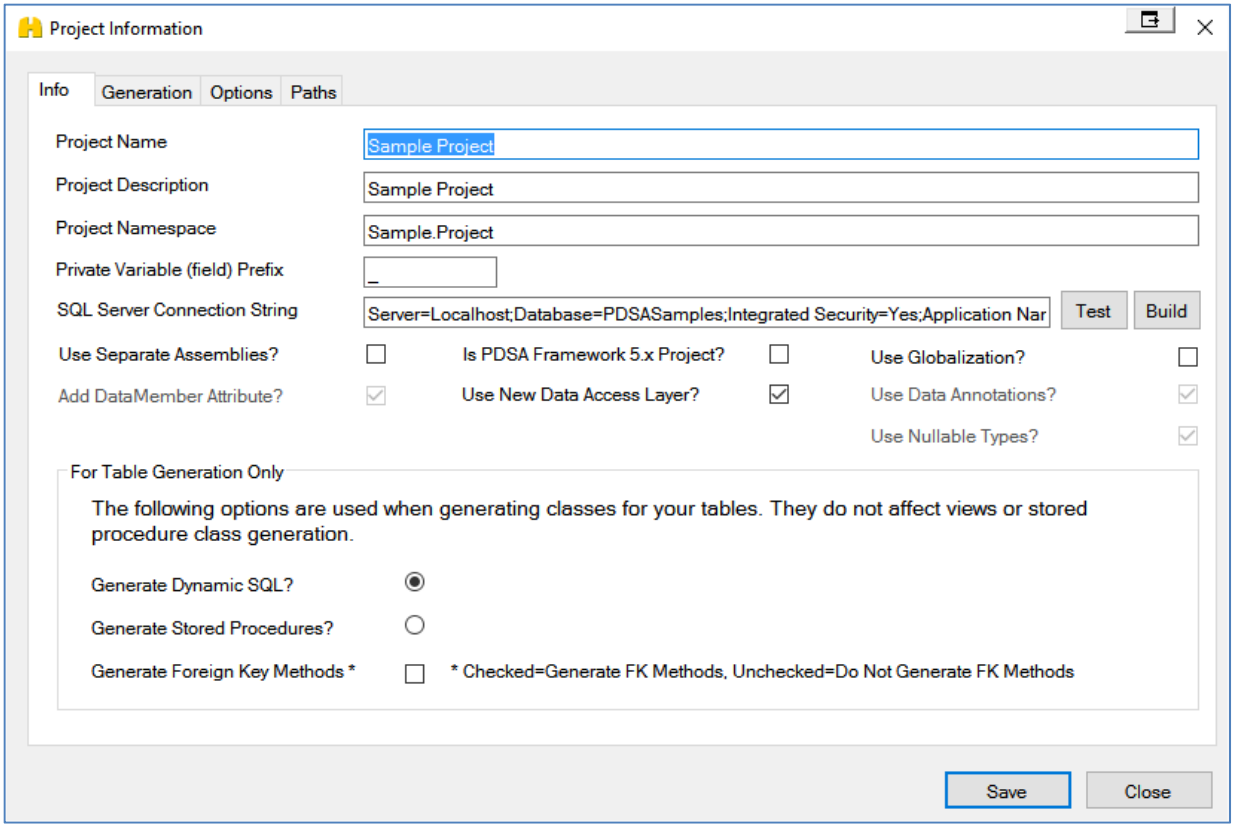

Figure 2: Project Information screen (Info Tab)

Field	Description
Project Name	This is the name of this project that will be presented in the Combo box drop down on the main Haystack screen. Each project must have a unique name in Haystack.
Project Description	A description of this project.
Project Namespace	The namespace you wish to use for the generated code.
Private Variable (field) Prefix	Any prefix you wish to put on fields that are generated in classes.
Project Connection String	The connection string that points to the database you will use for this project. **NOTE**: Once you set this property and you have read in the tables, views and/or stored procedures you will be unable to change this.
Use Separate	Check this is you are generating for a PDSA Framework template that separates

Assemblies?	Business, Data and Entity classes into separate DLLs.
Is PDSA Framework 5.x Project?	Check this if you are generating for a PDSA Framework v5.x project.
Use Globalization?	Check this to use DateTime.UtcNow to initialize date data types.
Add DataMember Attribute?	Check this to add the [DataMember] attribute to each property in your entity classes. This is generally used if you are using WCF and/or the Web API.
Use New Data Access Layer?	Check this if you wish to generate classes to target the new data layer contained in the PDSA.DataAccess namespace. If you check this you should be reading the chapters under the PDSA Data Access Layer documentation.
Use Data Annotations?	Add the appropriate .NET Data Annotations for handling validation on the properties of your Entity classes. **NOTE**: This only works when generating for the new PDSA.DataAccess namespace.
Use Nullable Types?	Check this if you want to use Nullable types. **NOTE**: This only works when generating for the new PDSA.DataAccess namespace.
For Table Generation	
Generate Dynamic SQL	Check this option if you wish to generate Dynamic SQL for your table's CRUD logic in your data classes. Even though Dynamic SQL is used, parameters are used for all submissions thus avoiding any issues with SQL Injection attacks.
Generate Stored Procedures	Check this option if you wish to generate and have your data classes for your table's CRUD logic use stored procedures. This means all select, insert, update and delete statements are called via stored procedures.
Generate Foreign Key Methods	Check this to automatically load any foreign key tables that are referenced from a table that you are viewing information for. If you select this option, you should make sure you open and generate code for any of the foreign key tables as well as the current table. A method is generated that uses the foreign key class that is generated. If you just generate the one table and don't generate the others then you will get compile-time errors.
Buttons	
Save Button	Click this button once you have filled in all of the project information to save the project data back to the Haystack database.
Close Button	Click this button to close this form without saving any of the project data.

Project Information Screen (Generation Tab)

On this tab (Figure 2) is where you can choose which items you wish to generate for this project. This screen may have different options than shown here based on which templates you have received when you purchased Haystack and which templates you have added on.

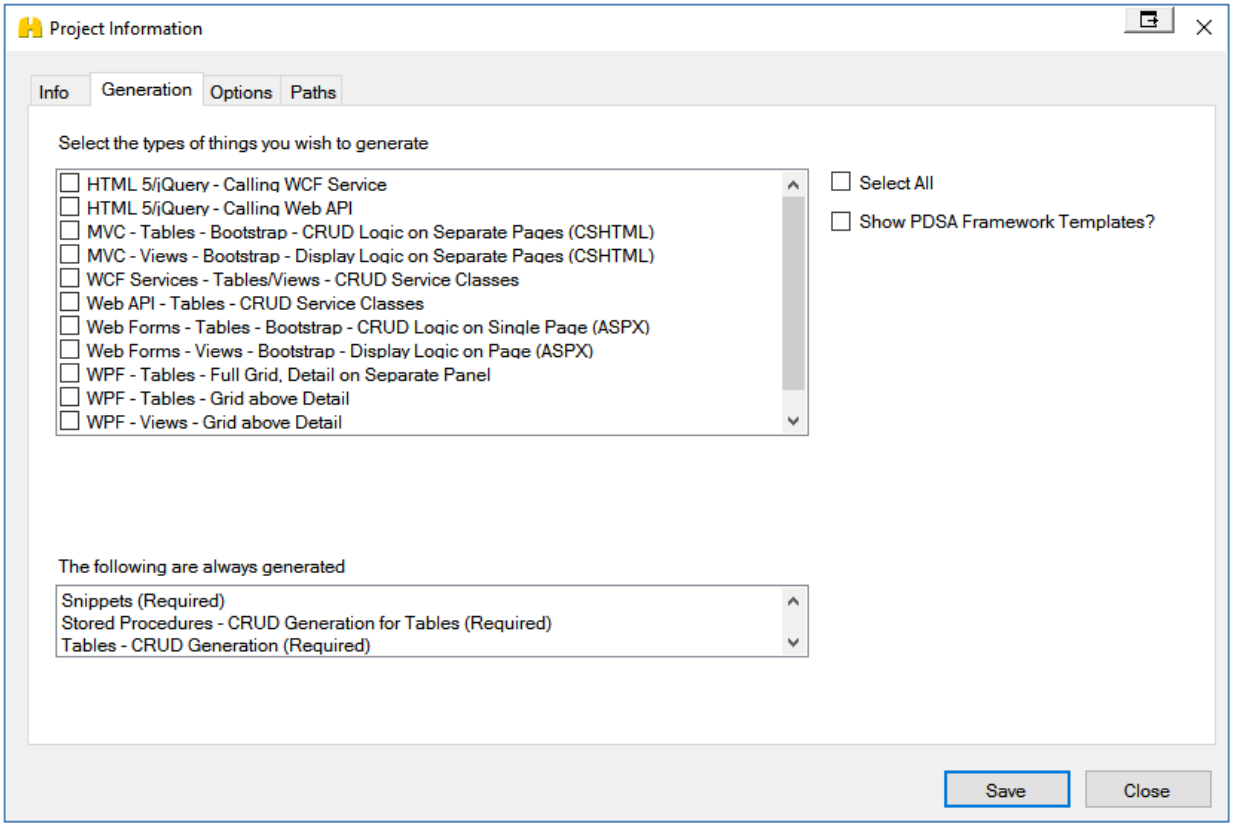

Figure 2: Project Information (Generation Tab)

Simply check and un-check which templates you wish to use when generating for this project.

Project Information (Options Tab)

On this tab (Figure 3) you can specify additional options that will control how the classes are generated in your project.

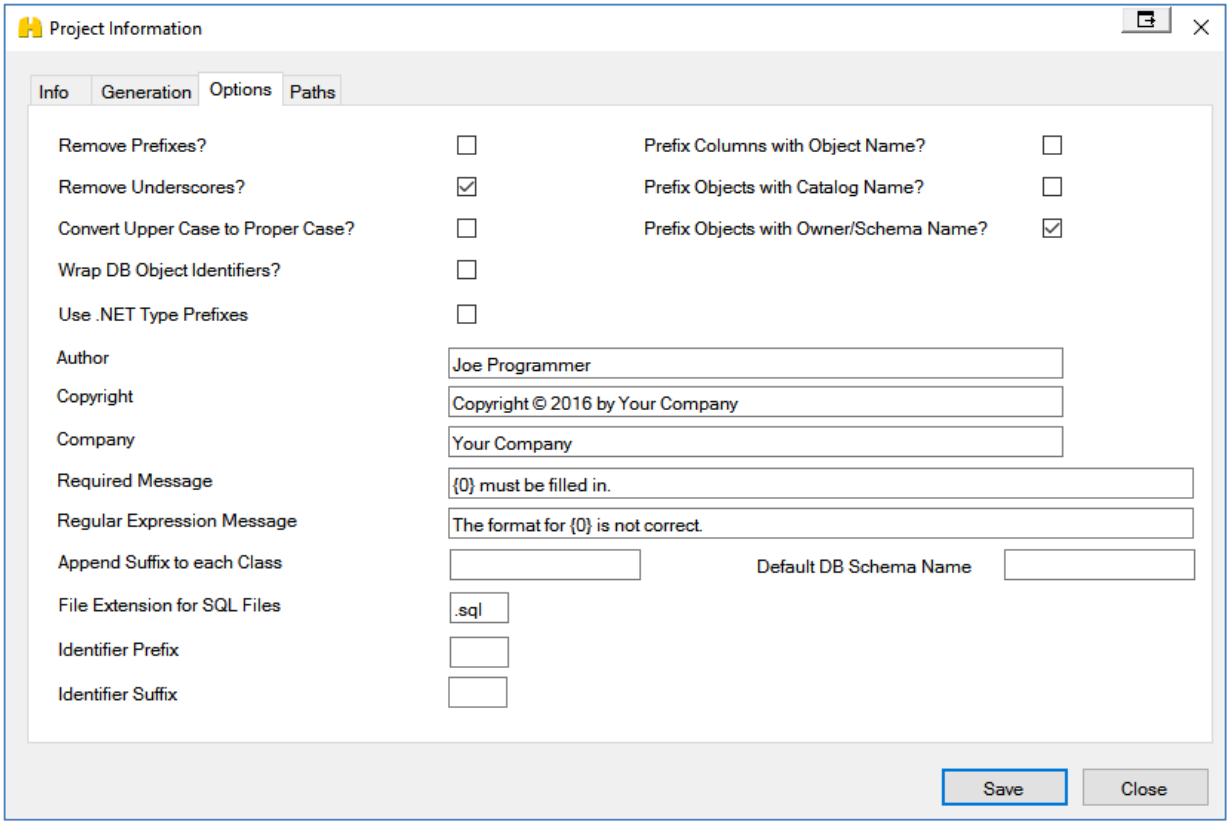

Figure 3: Project Information (Options Tab)

Field	Description
Remove Prefixes	If you prefix your column names with any 1 or 2 letters to denote type; ie. iProductId, sProductName, then choosing this option will remove the "i" and the "s". You can control the prefixes that are removed in the \Xml\Prefixes.xml file. **NOTE**: You only want to choose this option if you have used prefixes on all your tables and all columns.
Remove Underscores	If you use underscores in any table or column names, choosing this option will remove these prior to generating any names.
Convert Upper Case to Proper Case	If all objects in your database (table names, column names, etc.) are stored by the database engine in UPPER CASE characters and you want to convert them to proper case when you generate class names and property names, check this box. If you separate words with an underscore character then the underscore will be used as a delimiter to separate the words. If the table name or column name does not have an underscore just all one word, then the first letter will be capitalized and the rest of the word will be in lower case.
Wrap DB Object Identifiers	Check this option if you wish to put the "Identifier Prefix" and "Identifier Suffix" around all database object names such as tables, views, stored procedures and columns.
Use .NET Type Prefixes	If you wish to use "Hungarian Notation" for your generated fields (private variables), then you check this option. The prefix that is used can be found in the XML\xx_DotNetTypes.xml file. There is an element on each data type in this xml file called <Prefix> that you may change to whatever you wish.
Prefix Columns	Check this option if you wish to always prefix column names with the database

with Object Name	object name. For example, instead of just ProductName, it would be Product.ProductName.
Prefix Objects with Catalog Name	Check this option if you wish to prefix any database object name with the owner name/schema name and the Catalog Name. For example, instead of just a table name like Product, this will generate PDSASamples.dbo.Product. NOTE: If this option is checked, then the "Prefix Objects with Owner/Schema Name" will also be checked.
Prefix Objects with Owner/Schema Name	Check this option if you wish to prefix any database object name with the owner name/schema name. For example, instead of just a table name like Product, this will generate dbo.Product.
Author	Your name
Copyright	Your copyright
Company	Your company name
Required Message	Set this to the message that you wish to display when a field is required and must be filled in by the user. Use the .NET string format {0} to specify where you want the 'Header Text' for the column to be inserted. You will be able to override this on the Column Information screen.
Regular Expression Message	Set this to the message that you wish to display when a field does not meet the regular expression and must be fixed by the user. Use the .NET Format {0} to specify where you want the 'Header Text' for the column to be inserted. You will be able to override this on the Column Information screen.
Append Suffix to each Class	Set this to any special suffix you wish to add to all your generated classes.
Default DB Schema Name	The default DB Schema Name to use when filtering your tables, views and stored procedures.
File Extension for SQL Files	The file extension for all SQL files generated.
Identifier Prefix	The prefix to add before all generated database object names/identifiers. Use this if you have special characters such as a space in your field/column names.
Identifier Suffix	The suffix for add after all generated database object names/identifiers. Use this if you have special characters such as a space in your field/column names.

Project Information (Paths Tab)

On this tab (Figure 4) is where you can set various paths of where things are generated into or where templates, xml and xsd files for generation are located.

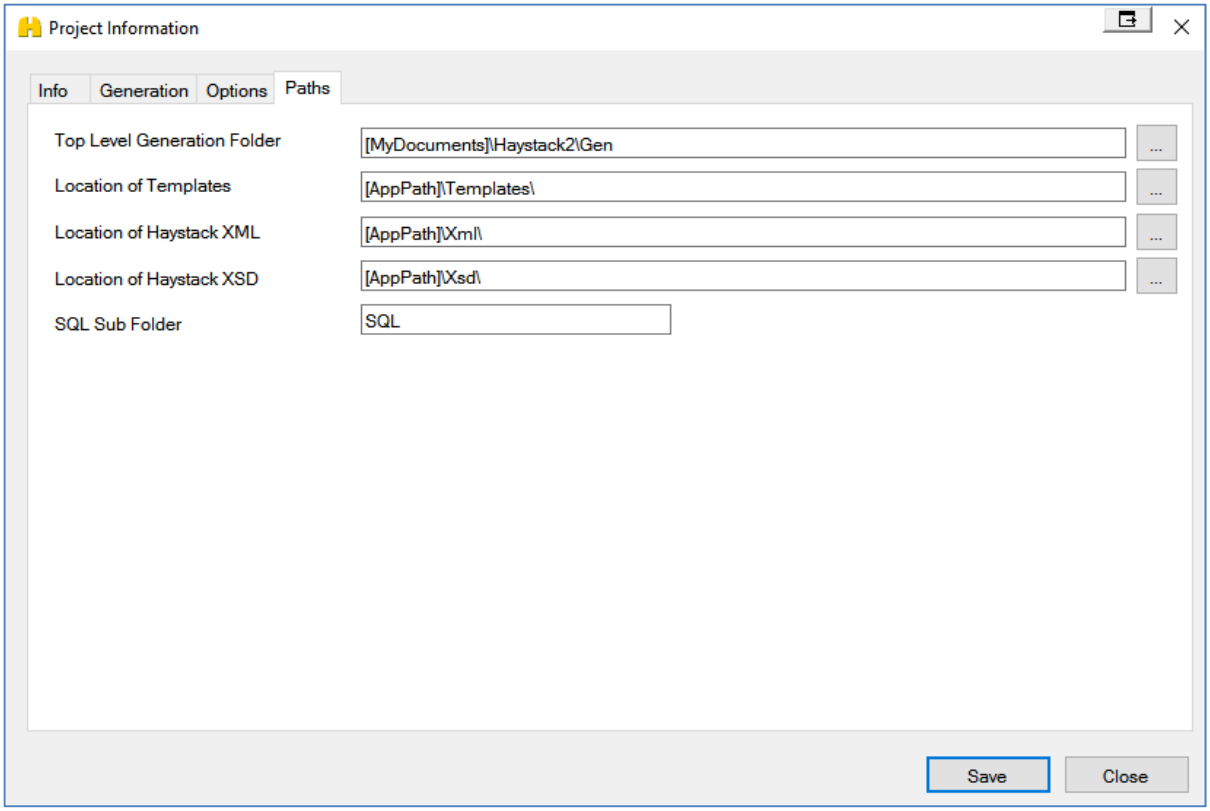

Figure 4: Project Information (Paths Tab)

Field	Description
Top Level Generation Folder	The top level folder into which to generate your classes. There can be subfolders that are generated into as well. The subfolders and where they are generated is controlled in the XML\xx_Templates.xml file. The default for this path is your personal [My Documents]\Haystack2 folder.
Location of Templates	The top level folder where all templates and code snippets are located. There are many other folders under this folder that contain the various templates. Again the location of where each template and snippet is located is controlled from the XML\xx_Templates.xml file.
Location of Haystack XML	The location of the XML files that control how Haystack works.
Location of Haystack XSD	The location of the XSD files that are the schema description of the XML files.
SQL Sub Folder	The location of where all SQL statements are generated into.

Step-by-Step Scenarios

The following are a series of step-by-step walkthroughs on how to perform some of the most common scenarios in Haystack. All of the Step-By-Step scenarios presented will assume you are using one of the template projects that ship with Haystack.

After each Step-By-Step you will then be shown how to generate code for a specific table and hook it up to one of the template projects and to your own project as well.

Be sure to look at the "**Quick Start**" chapters in this documentation for more step-by-step scenarios.

Table Data Classes using Dynamic SQL

If you are developing a service oriented application, using Web API, Angular, MVC, or Web Forms then using Dynamic SQL is perfectly acceptable as you will most likely be using a unique id and password to connect to your database that no one else can use or see since that connection string will be behind a server. Below are the steps you will take to create CRUD data classes that use dynamic SQL.

Create Dynamic SQL Project

Follow the steps below to create a project that will generated data classes that use Dynamic SQL for all database access.

1. Open up Haystack
2. Click **Add** on the Haystack main screen to add a new Project
3. Fill in the correct Project Connection String that points to the database where the tables are located that you wish to generate CRUD classes for
4. Check the **Generate Dynamic SQL?** check box only

Your screen should look similar to Figure 5.

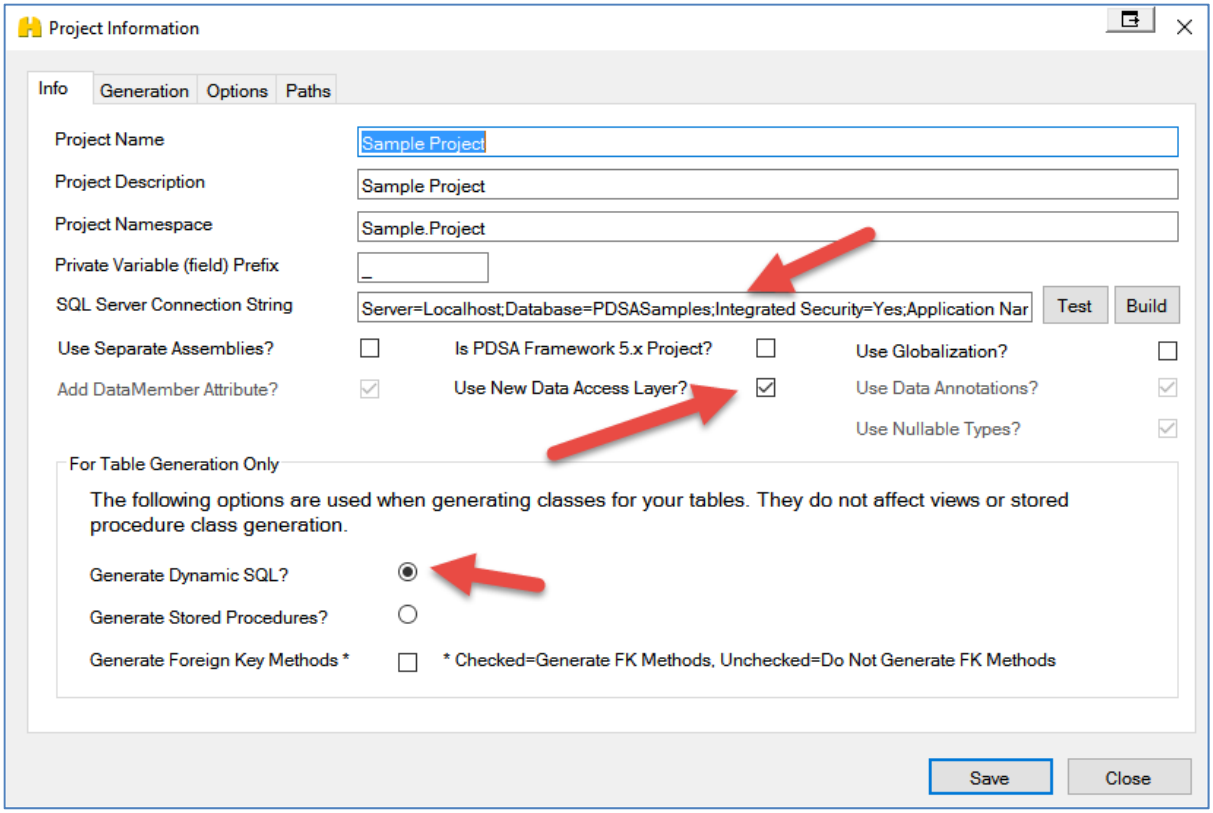

Figure 5: Project setup for Dynamic SQL

Click the **Save** button and you are then ready to read the tables in your selected database and generate CRUD classes for any table in that database.

Load Tables

After creating a project you now need to read in the tables you wish to generate CRUD classes for. Follow the steps below to read in your tables.

Click on the **Load Tables/Apply Filter** button (Figure 6) to read all of the tables located in the database specified in the connection string you setup in the Project Information screen.

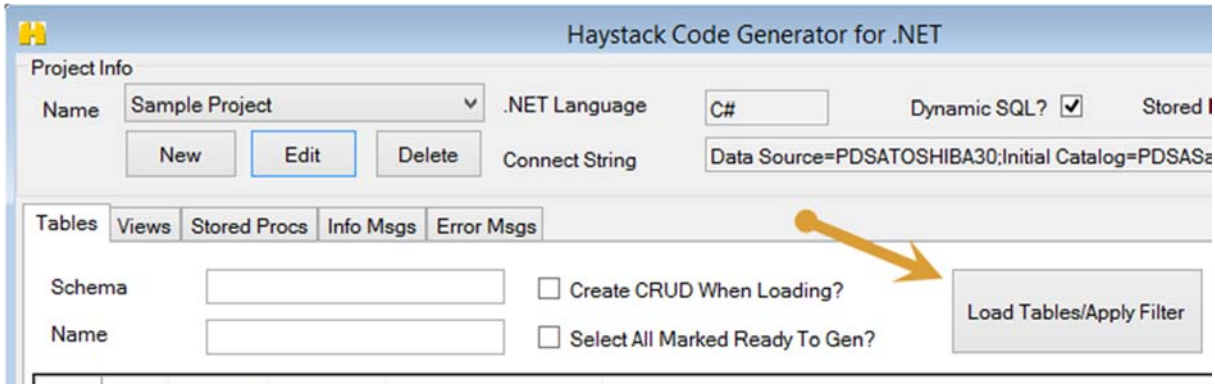

Figure 6: Read in Tables

After clicking on the Load Tables/Apply Filter button your screen will look similar to the following:

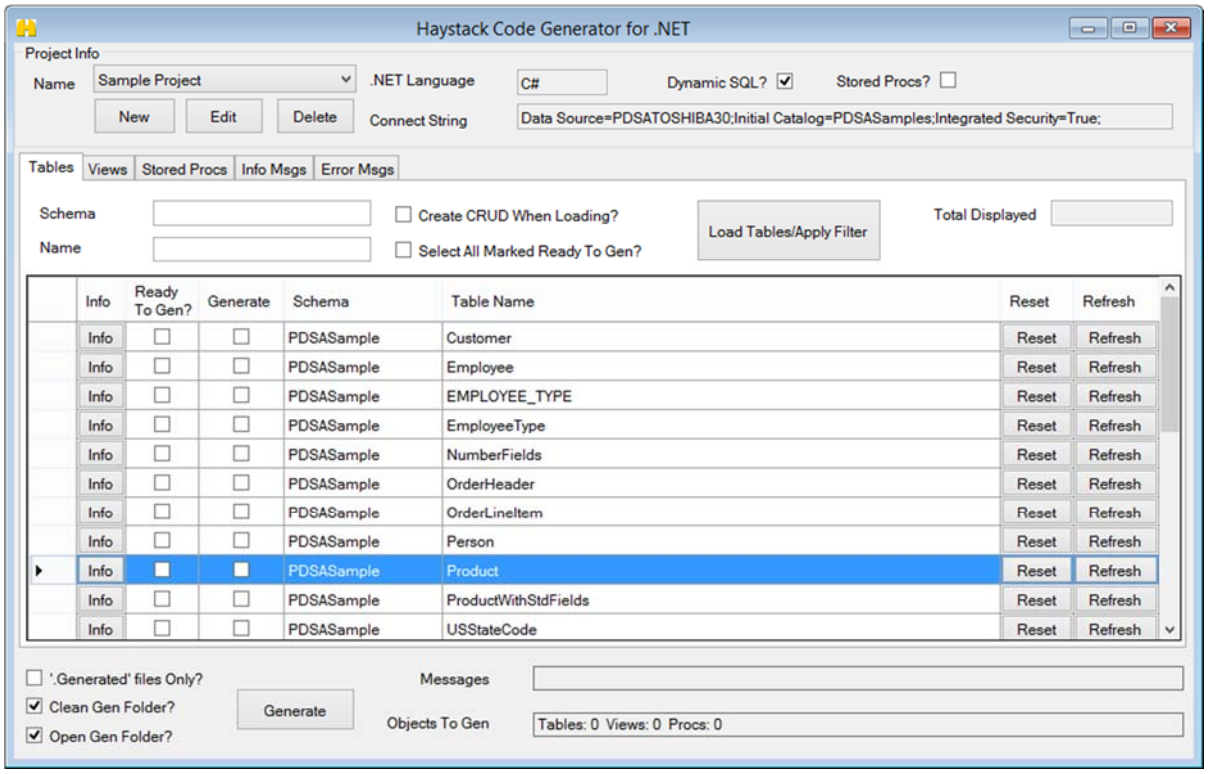

Figure 7: Tables loaded from your database

Create CRUD when Loading?

Prior to clicking on the “Load Tables/Apply Filter” you may click on the “Create CRUD When Loading?” This option will automatically read in all columns for all tables. This process will take longer to load all tables, but then your tables will be marked as ready to generate.

Select all Marked Ready to Gen?

After you have loaded tables and clicked on the Info button next to a table, or have created the CRUD logic for a table, then that table will have a check marked in the “Ready to Gen?” column. This means that there is enough meta-data read about the table that appropriate SELECT, INSERT, UPDATE and DELETE statements can be generated for that table.

Read Columns

Double click on one of your tables (or press the **Info** button) to display the Table Information screen (Figure 8).

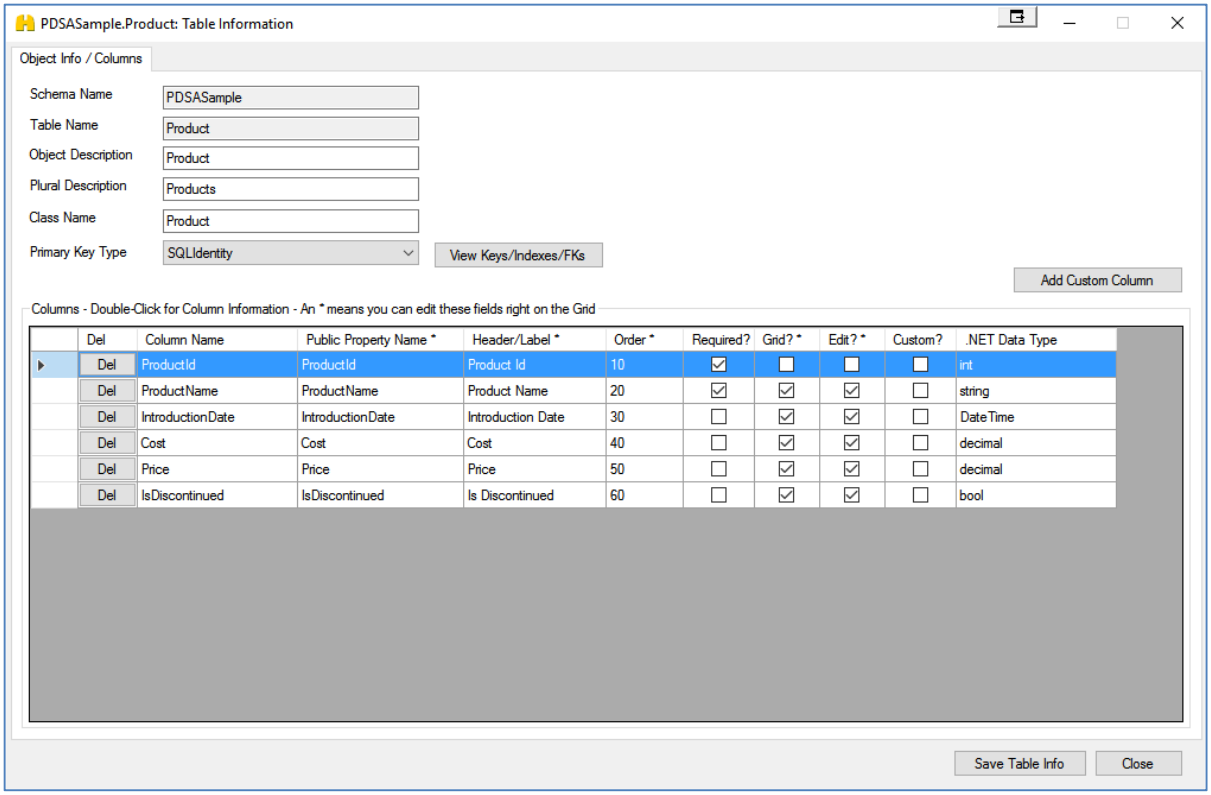

Figure 8: Table Information Screen (Object Info / Columns Tab)

When you display this screen Haystack has already loaded all of the columns for your table.

Generate Code for Dynamic SQL Classes

At this point you are ready to generate the code for your Dynamic SQL classes (Figure 9). When you go into the Table Information screen and you click the Save Table Info button, the table is marked as being ready for generation and the **Generate** check box is already checked.

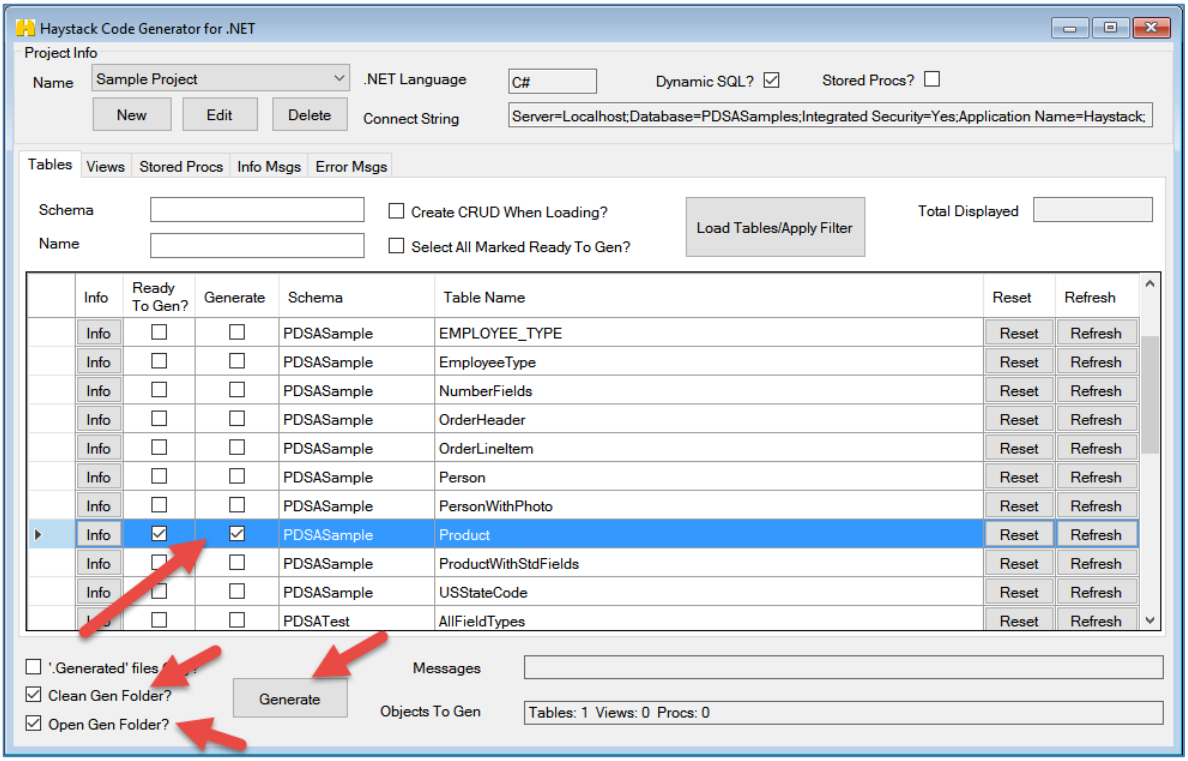

Figure 9: Generate Code for Dynamic SQL

If it is not already selected, choose the “Clean Gen Folder?” option and “Open Gen Folder?” These will delete all previously generated code prior to generating the new code and once completed will launch Windows Explorer to the generation folder.

Click on the **Generate** button to have the code for this table generated into the default code generation folder. Once the generation has completed Windows Explorer will open to the generation folder. For example, if you generated for the Product table, you would see the folders shown in Figure 10 located in the **[My Documents]\Haystack2\Gen\DataAccessLayer** folder.

NOTE: Your folders may be a little different depending on which templates you choose to generate.

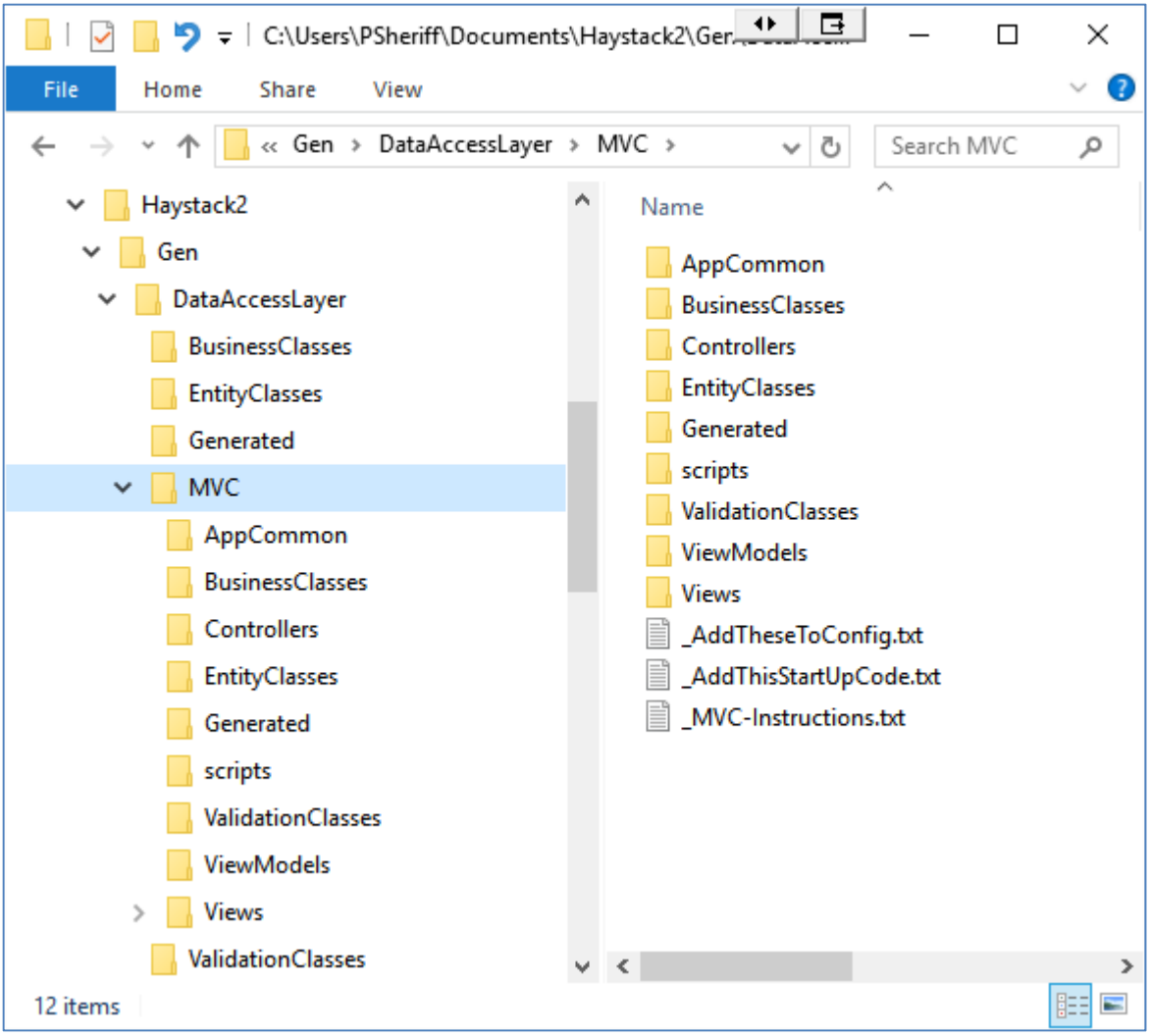

Figure 10: Generated Code Files for Dynamic SQL

The folders \BusinessClasses, \EntityClasses, \Generated and \ValidationClasses are where the Product data classes are generated to. Depending on what other templates you have chosen, you may see other folders like the MVC folder in the screen shot shown above.

NOTE:	Refer to one of the “Quick Start” chapter for information on how to add these generated files to your projects.

Table Data Classes using Stored Procedures

If you are developing a desktop application where the user will have direct access to the database server, or your DBA, or company specifies that you must use stored procedures, then Haystack can do that for you. Below are the steps you will take to create not only the CRUD data classes that call stored procedures for all SELECT, INSERT, UPDATE and DELETE statements, but will generate all of those stored procedures for you as well.

Create Stored Procedure Project

Follow the steps below to create a project that will generate data classes that use stored procedures for all access to your tables. This means NO dynamic SQL will be generated.

1. Open up Haystack
2. Click **Add** on the Haystack main screen to add a new Project
3. Fill in the correct Project Connection String that points to the database where the tables are located that you wish to generate CRUD classes for
4. Select the **Generate Stored Procedures?** radio button.

Your screen should look similar to Figure 11.

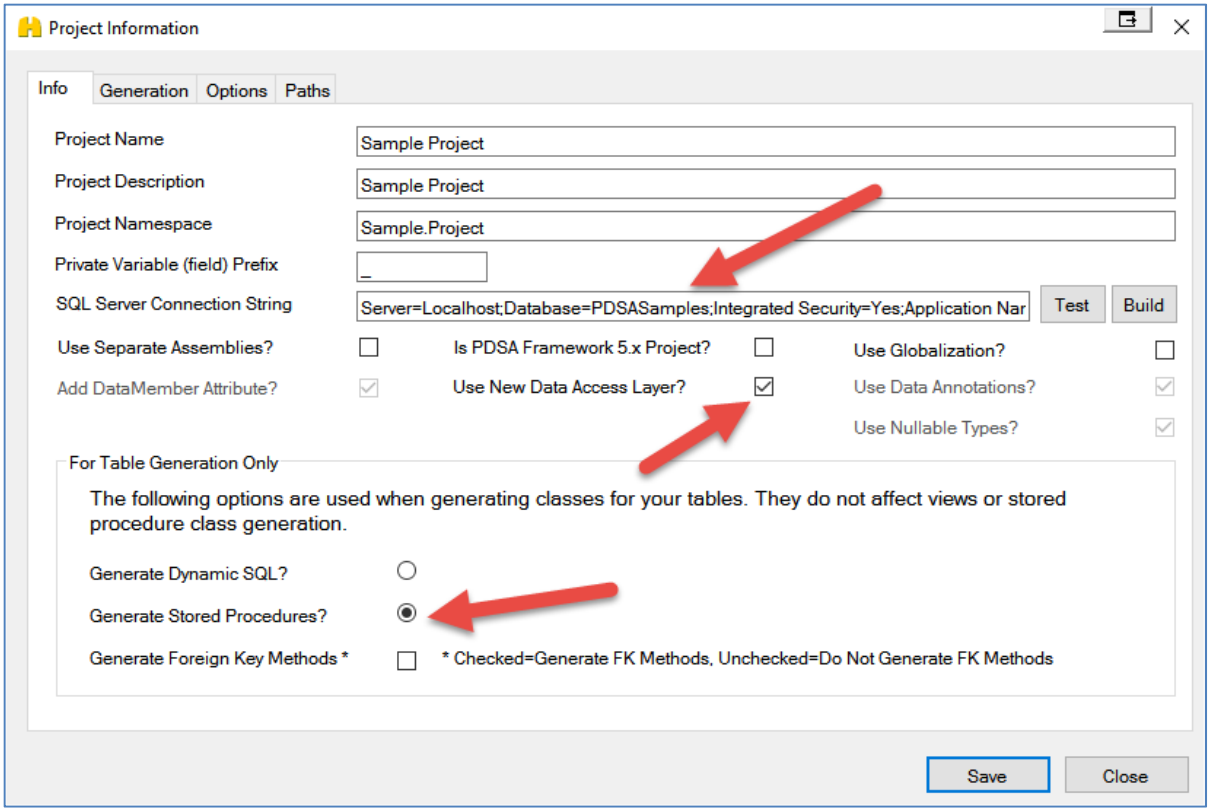

Figure 11: Project Information Screen for Generating Stored Procedures

Click the **Save** button and you are then ready to read the tables in your selected database and generate CRUD classes and stored procedures for any table in that database.

Load Tables to Generate Stored Procs for

After creating a project you now need to read in the tables you wish to generate CRUD classes and stored procedures for. Follow the steps below to read in your tables.

Click on the **Load Tables/Apply Filter** button (Figure 12) to read all of the tables located in the database specified in the connection string you setup in the Project Information screen.

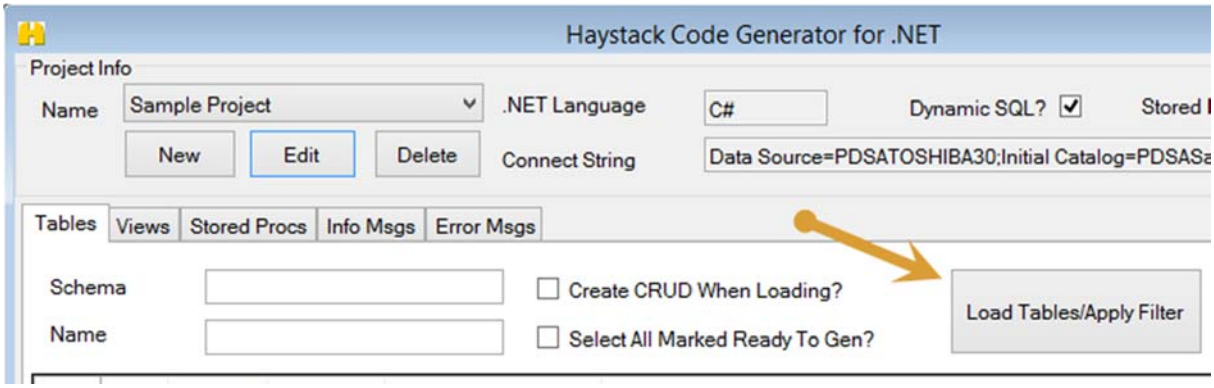

Figure 12: Read in Tables

After clicking on the Load Tables/Apply Filter button your screen will look similar to the following:

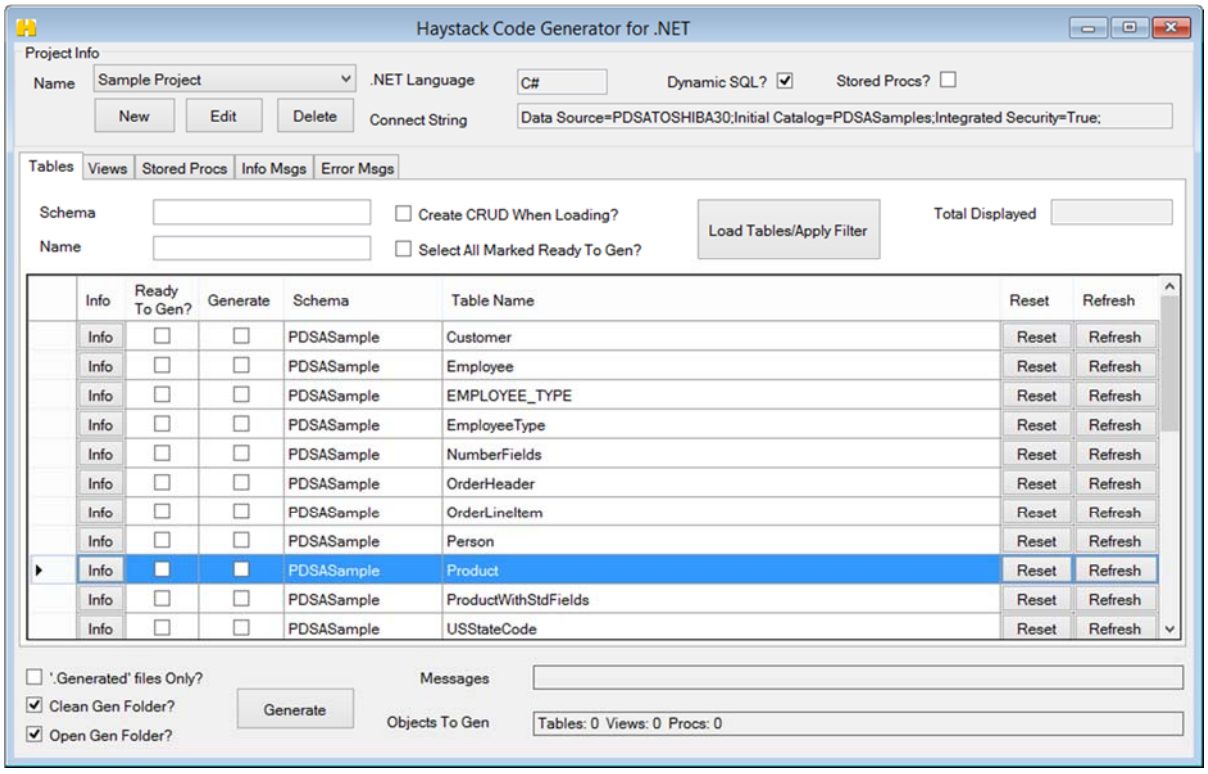

Figure 13: Tables loaded from your database

Read Columns and Generate Default CRUD Stored Procedures

Double click on one of your tables (or press the **Info** button) to display the Table Information screen (Figure 14).

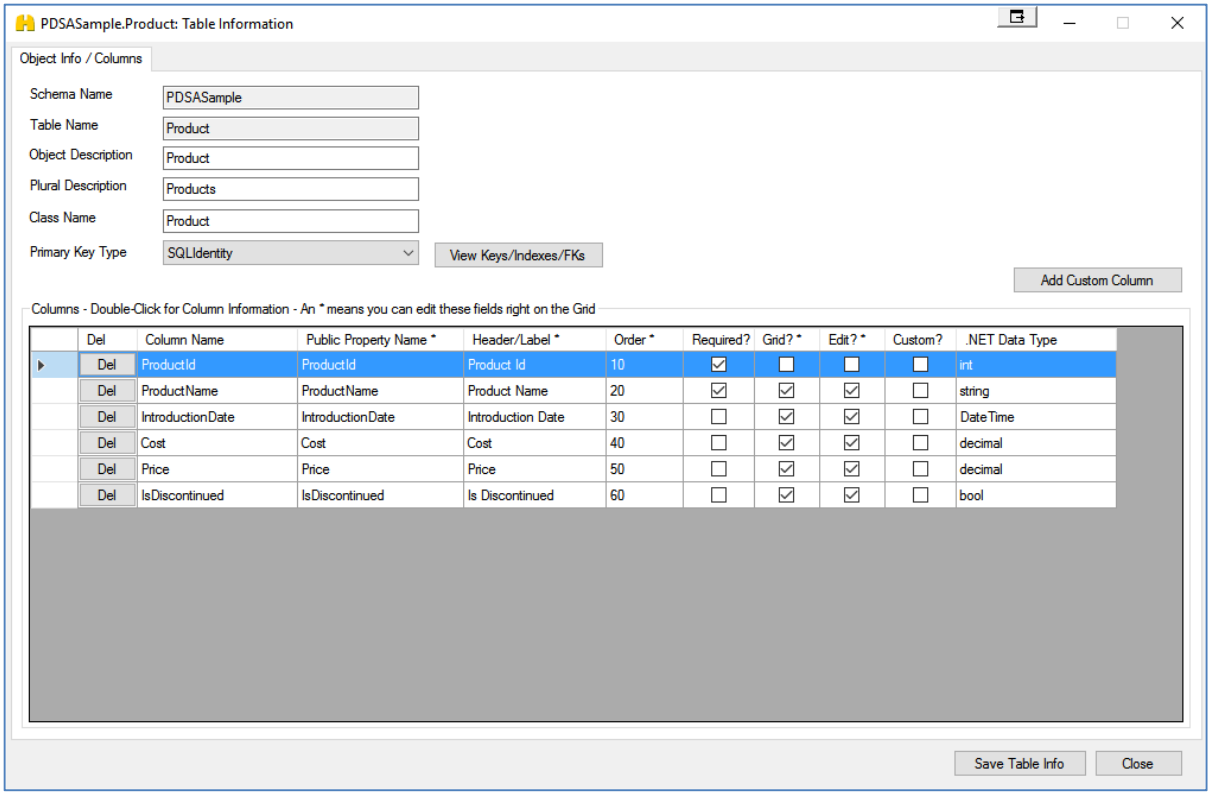

Figure 14: Table Information Screen (Object Info / Columns Tab)

When you display this screen Haystack has already loaded all of the columns for your table.

Generate Code for Table Stored Procedure Classes

At this point you are ready to generate the code for your table that uses stored procedures for all CRUD logic. When you go into the Table Information screen and you click the Save Table Info button, the table is marked as being ready for generation and the **Generate** check box is already checked (Figure 15).

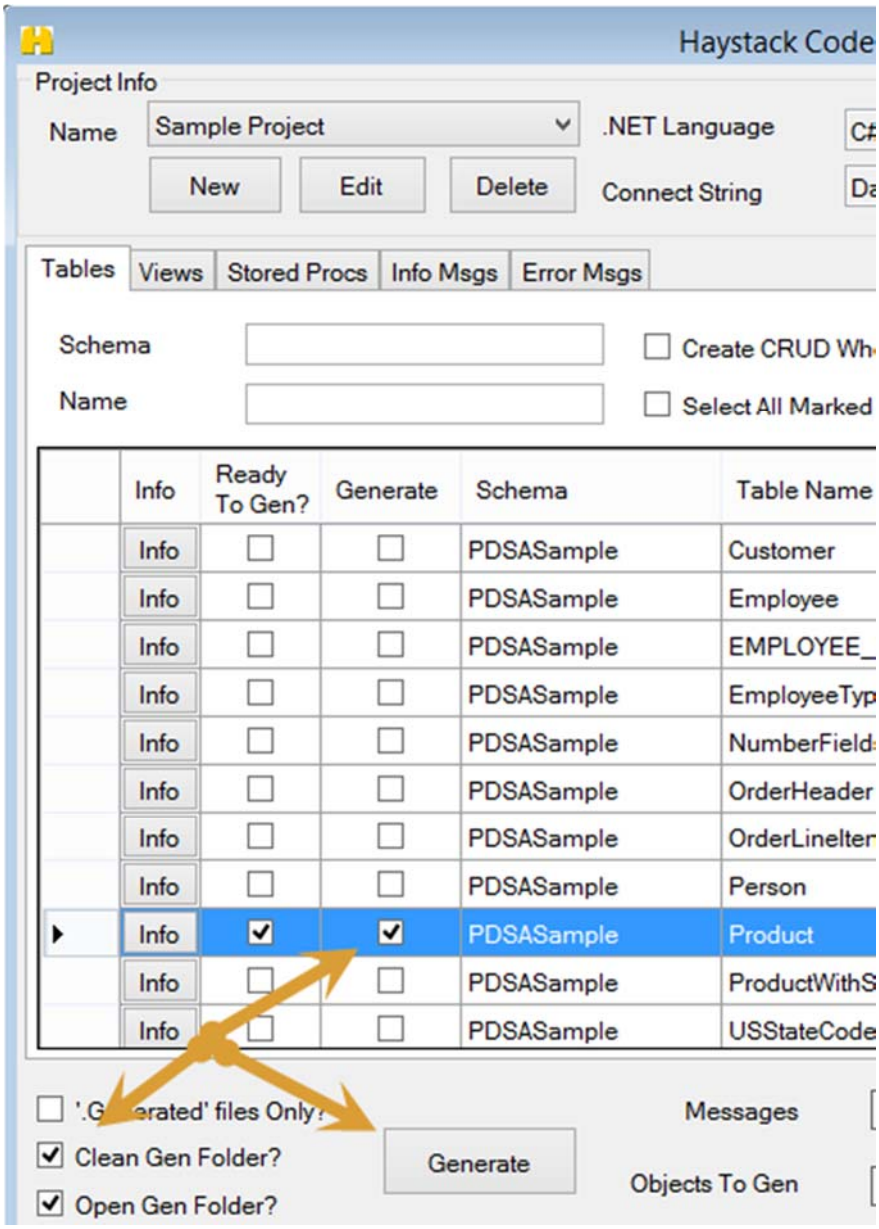

Figure 15: Generate Code for Table and Stored Procedures

If it is not already selected, choose the “Clean Gen Folder?” option and “Open Gen Folder?” These will delete all previously generated code prior to generating the new code and once completed will launch Windows Explorer to the generation folder.

Click on the **Generate** button to have the code for this table generated into the default code generation folder. Once the generation has completed Windows Explorer will open to the generation folder. For example, if you generated for the Product table, you would see the folders shown in Figure 16 located in the **[My Documents]\Haystack2\Gen\DataAccessLayer** folder.

NOTE: Your folders may be a little different depending on which templates you choose to generate.

For example, if you generated for the Product table, you would see the files shown in located in the generation folder.

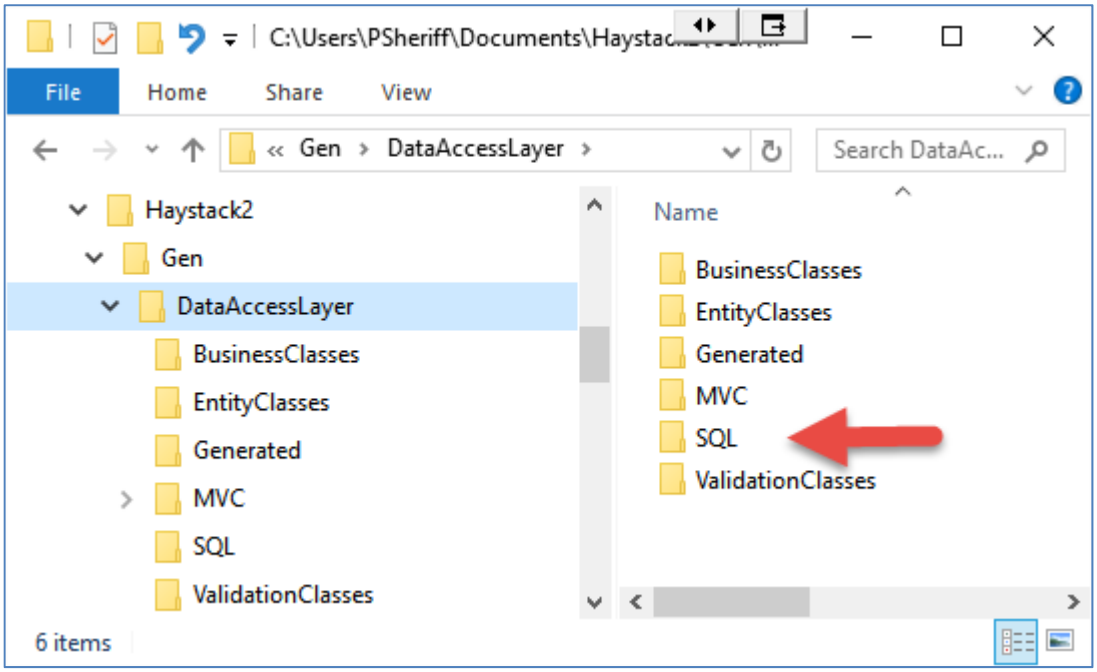

Figure 16: Generated Code Files for Stored Procedures

Notice the **\SQL** folder. This is where you will find the generated CRUD stored procedures that you will need to add manually to your database.

The folders \BusinessClasses, \EntityClasses, \Generated and \ValidationClasses are where the Product data classes are generated to. Depending on what other templates you have chosen, you may see other folders like the MVC folder in the screen shot shown above.

NOTE:	Refer to one of the "Quick Start" chapter for information on how to add these generated files to your projects.

View Data Classes

The steps are the same for generating views as for tables. Just click on the Views tab and load the views in the database. Of course, views are read-only, so you can't do any insert, updates or deletes. You may add any WHERE or ORDER BY clauses that you want however.

Stored Procedure Data Classes

If you have created stored procedures in your database to select or modify data, or perform any other routine, you may create classes to call those stored procedures. The classes that are generated create a property for every input and output parameter and for every column returned from the stored procedure. This makes calling your stored procedure as easy as calling the CRUD classes generated for tables.

To use the classes to call a stored procedure you simply set those properties that correspond to the parameters and call the appropriate method to execute the stored procedure. All of the parameters are automatically bundled into a parameters collection and submitted via an ADO.NET command object for you. After the statement executes, you can retrieve any output parameters via the corresponding property on your data class. It is that easy!

Haystack generates data classes that correspond to what the stored procedure does. Below are the different types of classes that can be generated for your stored procedures.

SP Type	Description
Execute	This is for stored procedures that just perform an action like an INSERT, UPDATE, or DELETE. Or maybe lots of various actions.
Read Only No Params	This class is generated for any stored procedures that just return data and there are no parameters to pass to it.
Read Only With Params	This class is generated for any stored procedures that return data and you have some input and/or output parameters.

Stored Procedures that Execute Data Modification Statements

To generate code for a stored procedure that executes data modification statements you simply need to load the stored procedures in the database then double click on the stored procedure that you wish to generate.

This will bring up the Stored Procedure Information form. On this form you will need to choose the type of stored procedure this is. Choose "Execute" from the combo box. Haystack will attempt to read the stored procedure and determine what type of stored procedure it is. You can always override it by changing the value in the drop-down box shown in Figure 17.

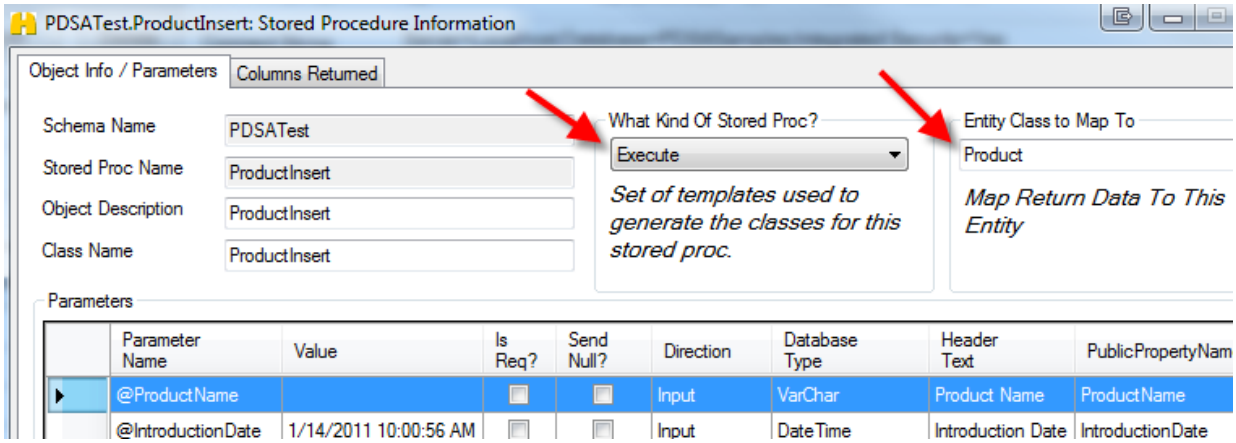

Figure 17: Choose the type of stored procedure

There is no need to go into the "Columns Returned" tab as no columns will be returned from these types of stored procedures. Any output parameters for this type of stored procedure will be displayed in the parameters list. These will be filled in after you execute the stored procedure in your application.

Just click the **Save Stored Proc Info** button to save the stored procedure information and make it ready to be generated. After it is generated you may add it to any of the template projects as described earlier in this chapter to test the generated class.

Entity Class to Map To

Notice in Figure 17 the field “Entity Class to Map To”. This is used when the stored procedure that you are calling has the exact same parameters as the columns in a table. For example, if you have a Product table and the stored procedure you are generating for does an Insert into the same columns as that table. Note that it does not have to be the exact same number, but should have the same names. In this case, put in the name of the generated Entity class of that table into this field. Haystack will then re-use that Entity class name in the generated code and will NOT generate a new entity class just for this stored procedure. This will cut down on the amount of classes that are generated.

Another option is if you are calling a stored procedure that has many of the same parameters as a particular table that you have generated, but maybe has some output parameters, you can still use this. In this particular case you would have to go into the Entity class that was generated that is for you to add properties to and simply add the appropriate properties.

Stored Procedures that Return Data

If you have a stored procedure that returns data to you, there are two different types. One type has no parameters that are passed in. The other has parameters that are passed in, and optionally passed back. From the Stored Procedure Information form you will need to choose the type from the combo box for the stored procedure selected (Figure 18).

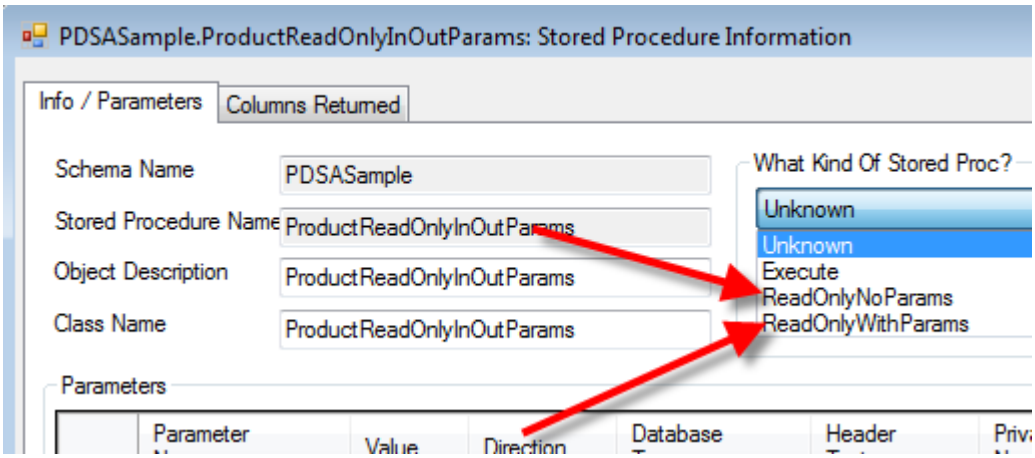

Figure 18: Two types of stored procedure classes can be generated for those stored procedures that return a result set.

If the stored procedure is going to be returning columns, you will need to go to the "Columns Returned" tab and execute the stored procedure so the DACGen can generate a property for each column that will be returned.

If the stored procedure needs input parameters, you will need to type in an appropriate value for that parameter. There is a column called “Value” next to the “Parameter Name” column. You can type right into the grid a string or numeric value as appropriate for that parameter (Figure 19).

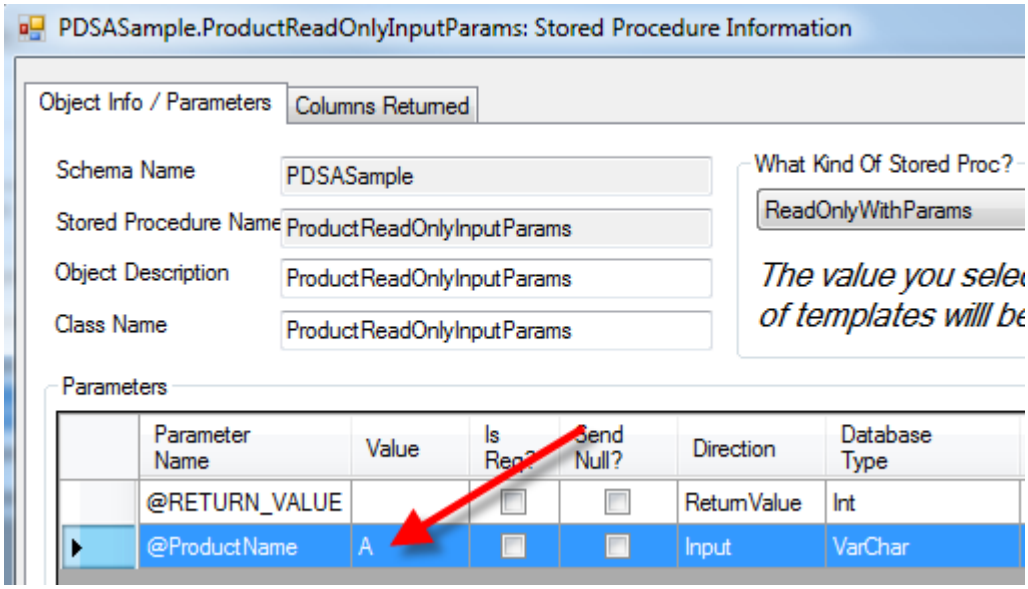

Figure 19: Stored Procedure Information Screen

After filling in all the required values you are now ready to click on the "Columns Returned" tab and click on the **Get Columns** (Figure 20) button.

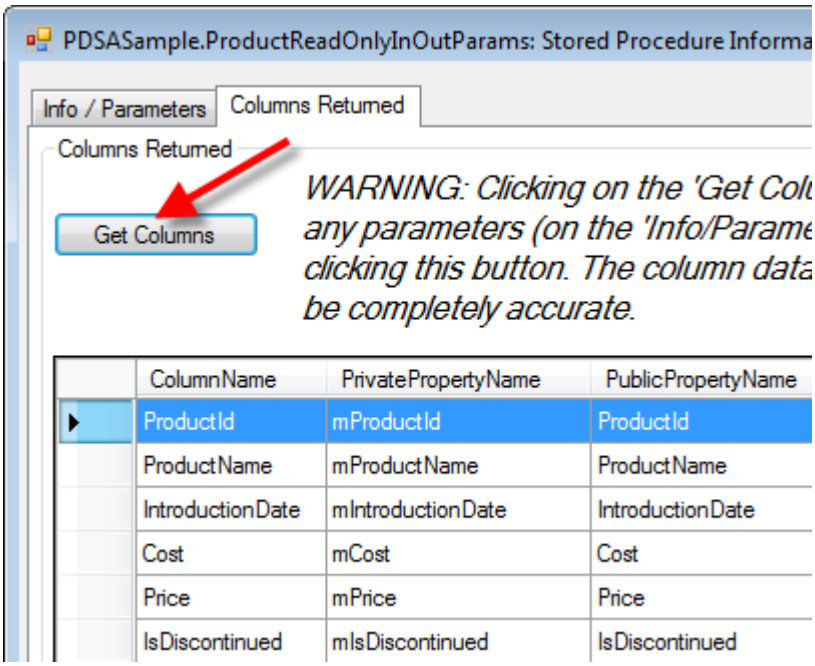

Figure 20: Click Get Columns to Run the Stored Procedure and have all columns returned brought into Haystack.

If you filled in the parameters correctly, you will get the list of columns that this stored procedure returns. If you get a message that no columns are returned, it typically means that you did not fill in one of the parameters correctly.

Summary

Generating code for all the various type of database objects is a fairly straight-forward process. This chapter presented an overview of how easy it is to generate code and put that code into a template project so you can start using it right away.

Chapter Index

R

S

T

U

V

W