Dynamic Connection String

Nick Sagriotis, 8 Apr 2014 CPOL

Rate this:

4.84 (20 votes)

You want to customize the database connection string at run-time? Here is how.

Download sources (C# and VB.NET) - 81.1 KB

Introduction

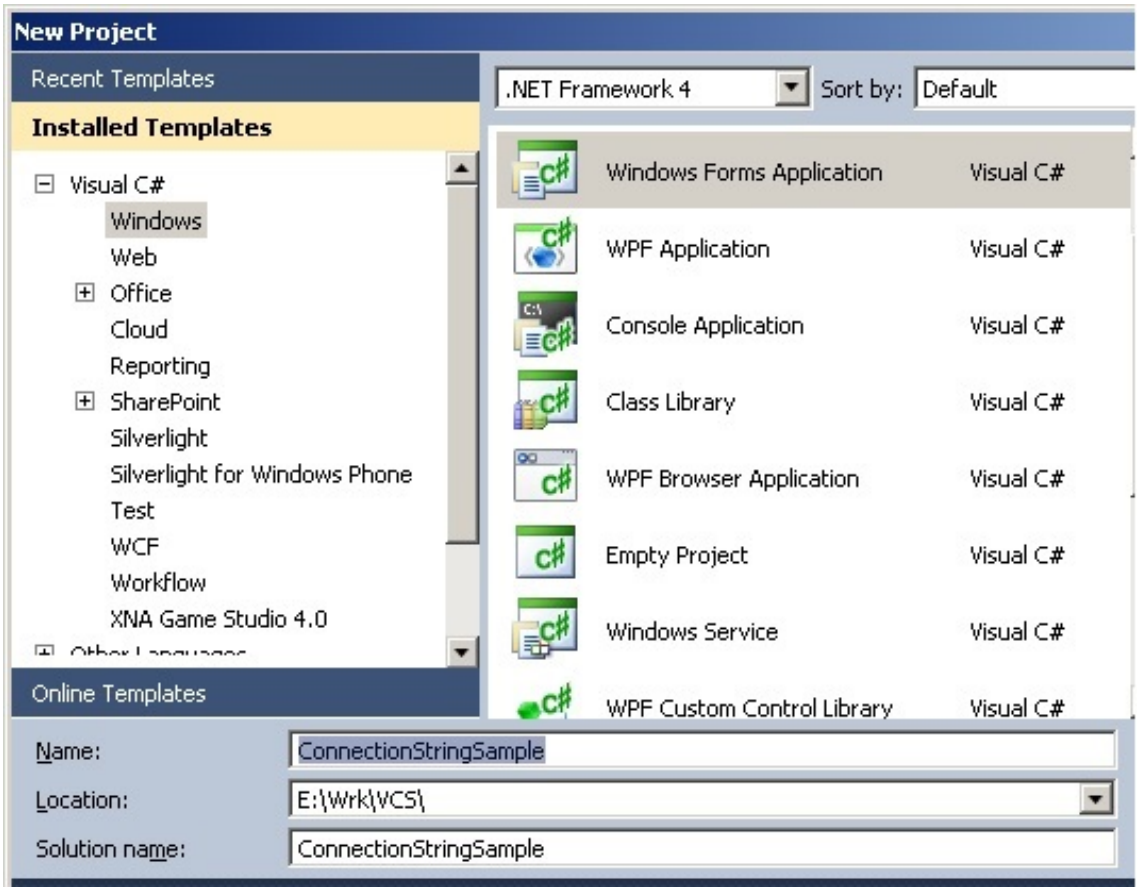
This is a "show by example" solution to the need, for dynamically defined database connection string. For the example, I use Visual Studio 2010, Microsoft SQL Server and C#, but other combinations could be possible.

Microsoft suggests that you should provide user credentials, to connect to a database from your application. Another approach is to supply SQL Server authorization, but this information will be held within your code or "*.config*" files (which has to be encrypted to hide your sensitive data, and so on . . .). What if you want to work on the development stage with your own credentials and the release product to get its values with some user intervention method, is this possible? Yes it is and this article will tell you how to do it!

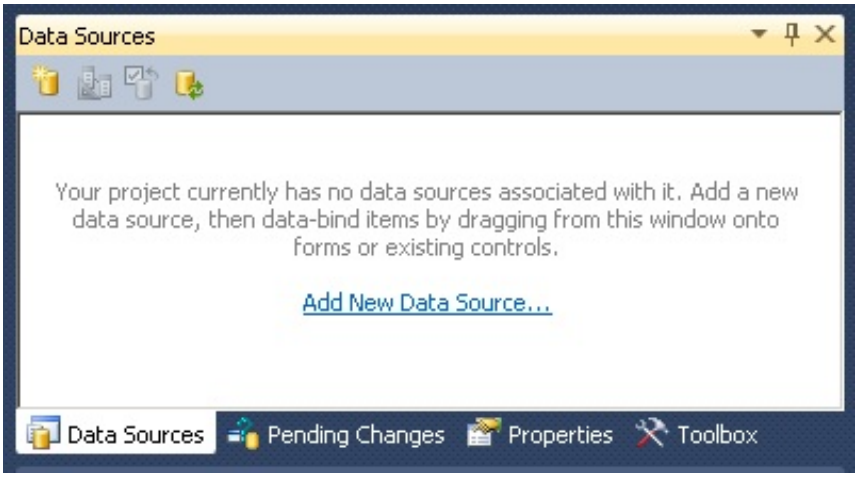
I believe the example is simple and anyone with basic knowledge of Visual Studio and C#, will have no problem to follow and understand it.

Begin Here

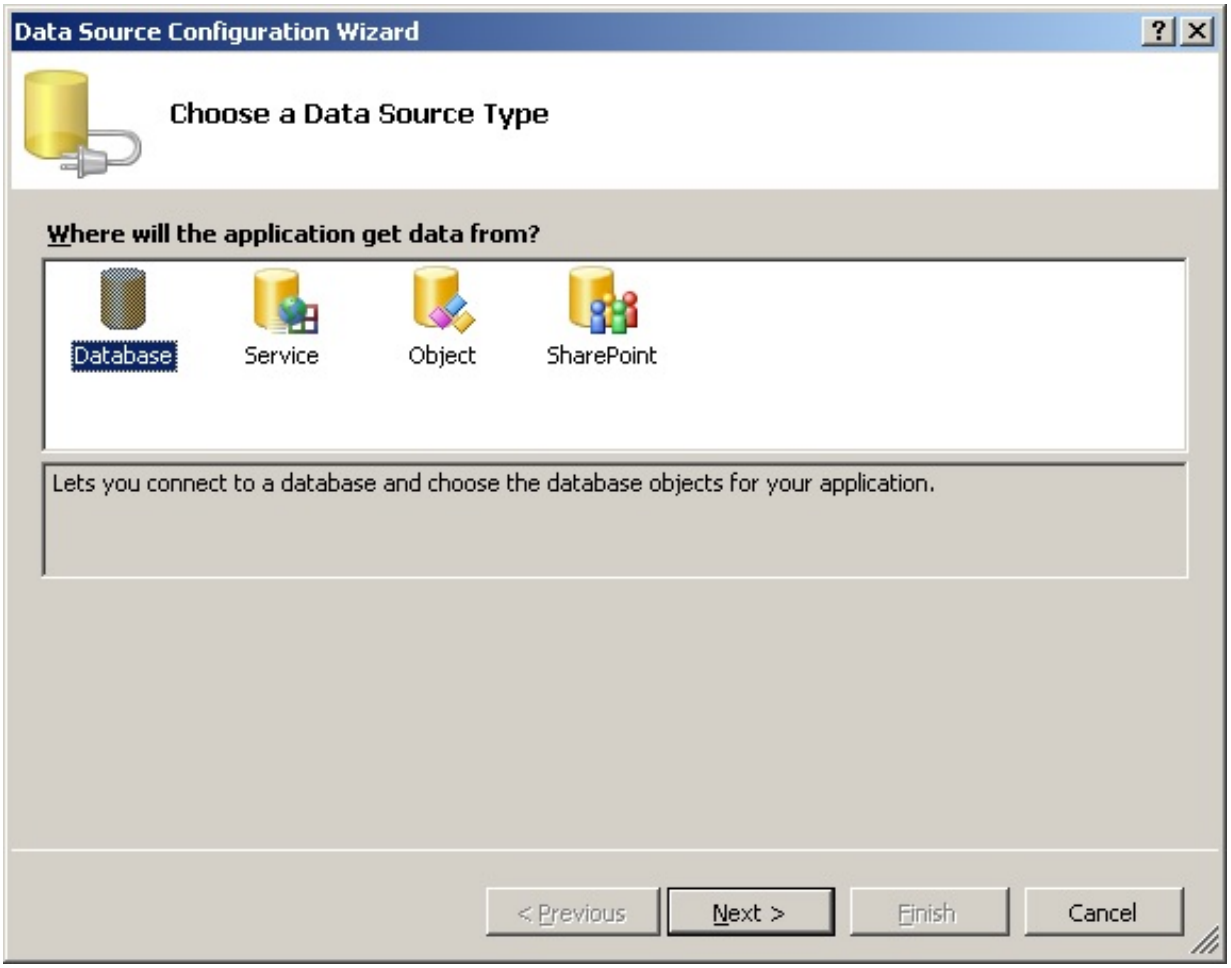
Start by creating a new project in Visual Studio 2010, for C#. Name the project `ConnectionStringSample`.



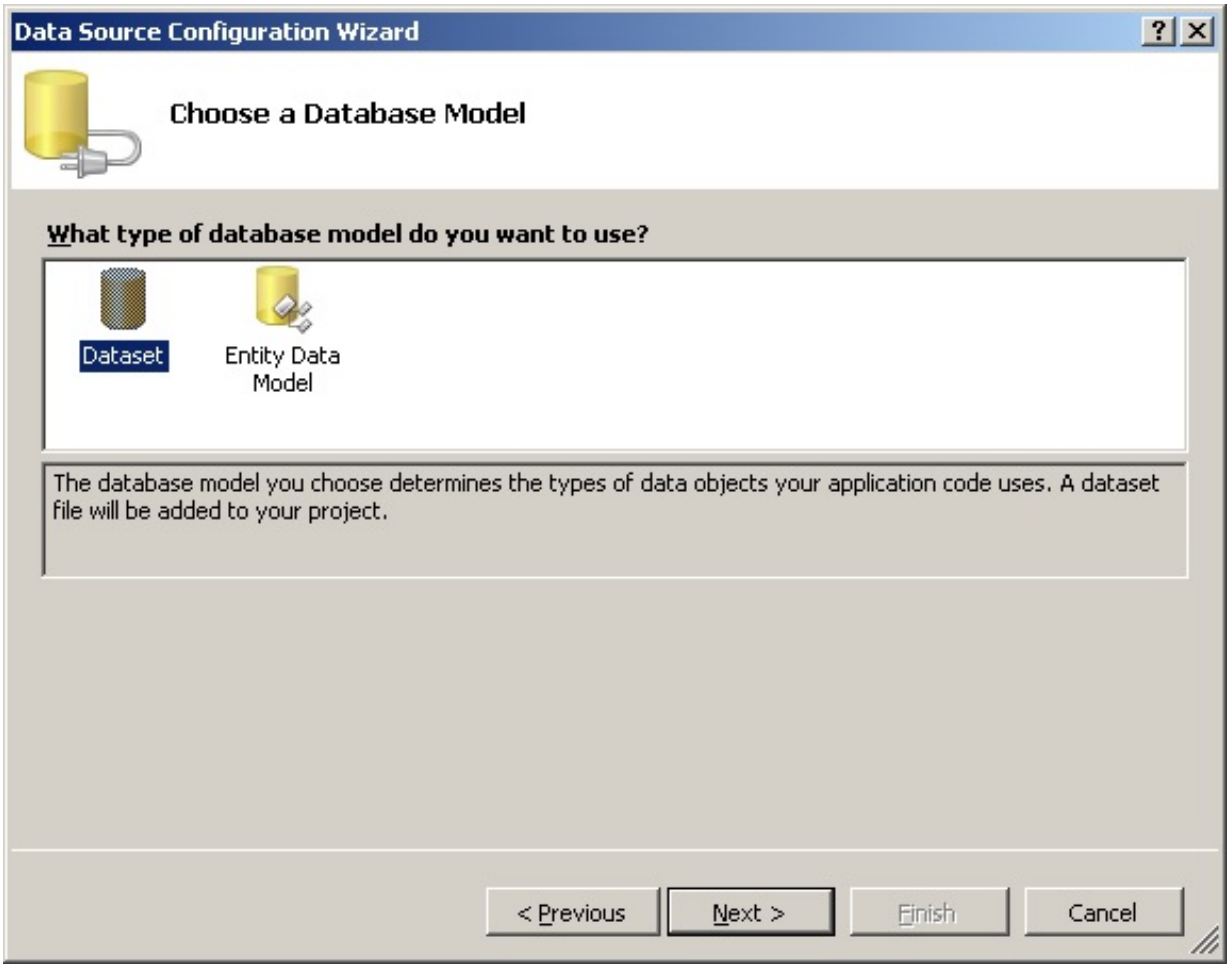
Then we will add the datasource to the application. For our example application, we will work with `NorthWind` sample database for SQL Server, but for any other database the procedure will be the same. Just click on the **Add New Data Source** within the **Data Sources** window.



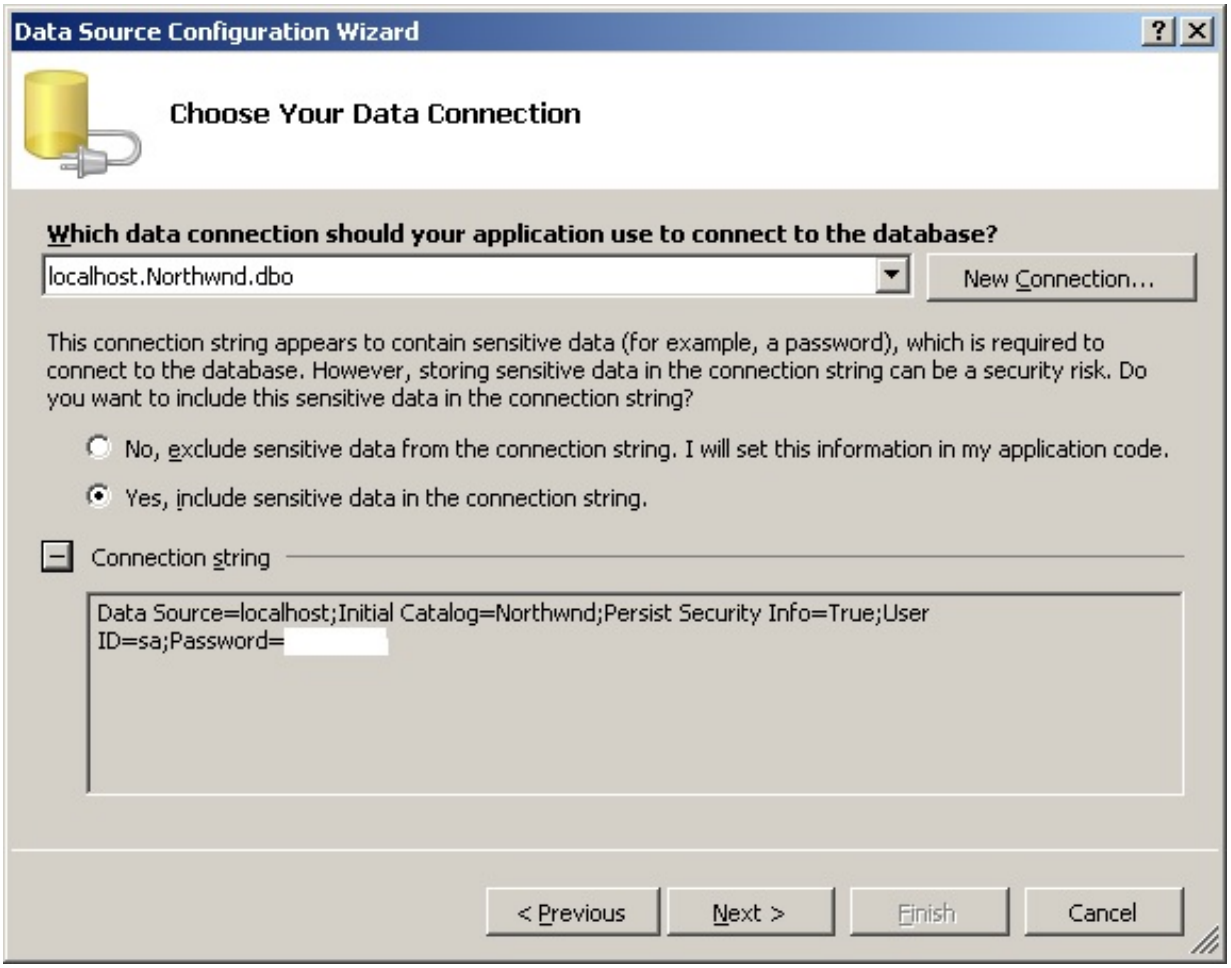
This will get us to the **Data Source Configuration Wizard**. Select and continue, as appears at the images below:


Data Source Configuration Wizard
Choose a Data Source Type
Where will the application get data from?
Database
Service
Object
SharePoint
Lets you connect to a database and choose the database objects for your application.
< Previous
Next >
Finish
Cancel

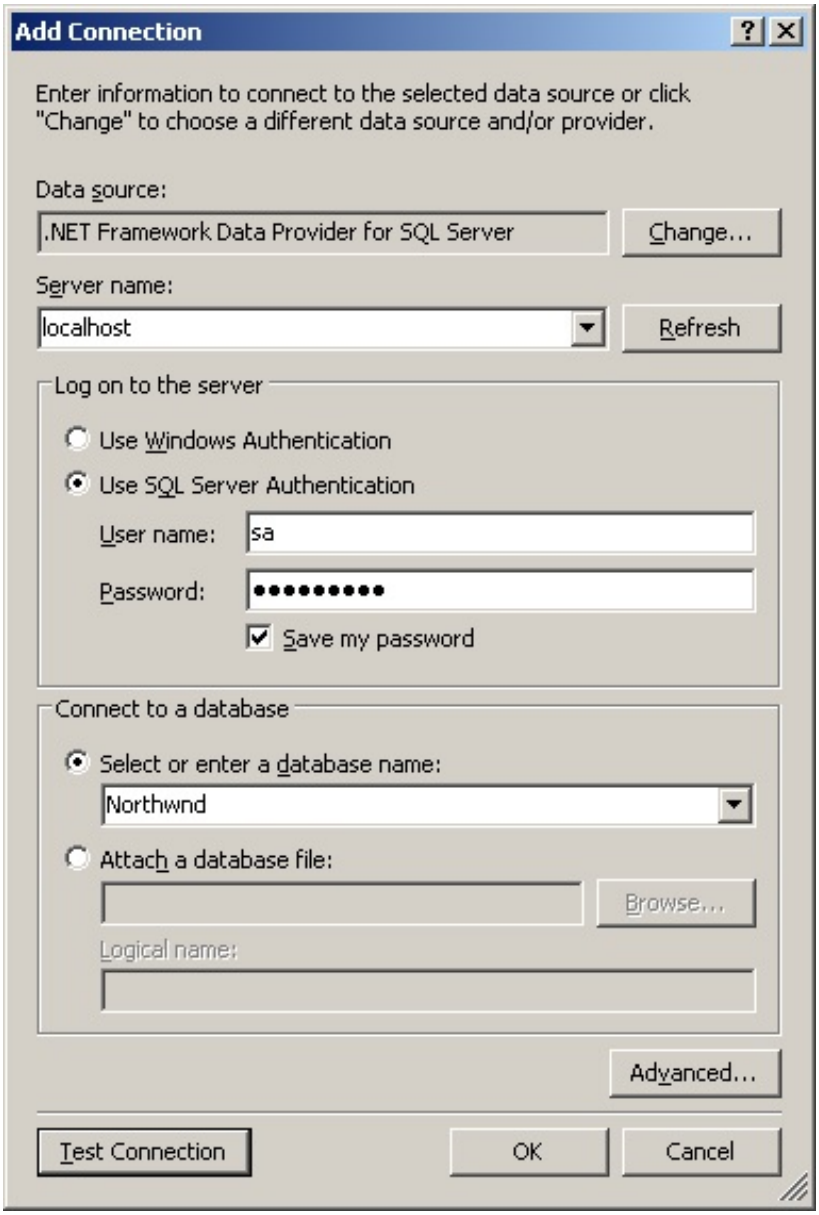
Step 1


Data Source Configuration Wizard
Choose a Database Model
What type of database model do you want to use?
Dataset
Entity Data Model
The database model you choose determines the types of data objects your application code uses. A dataset file will be added to your project.
< Previous
Next >
Finish
Cancel

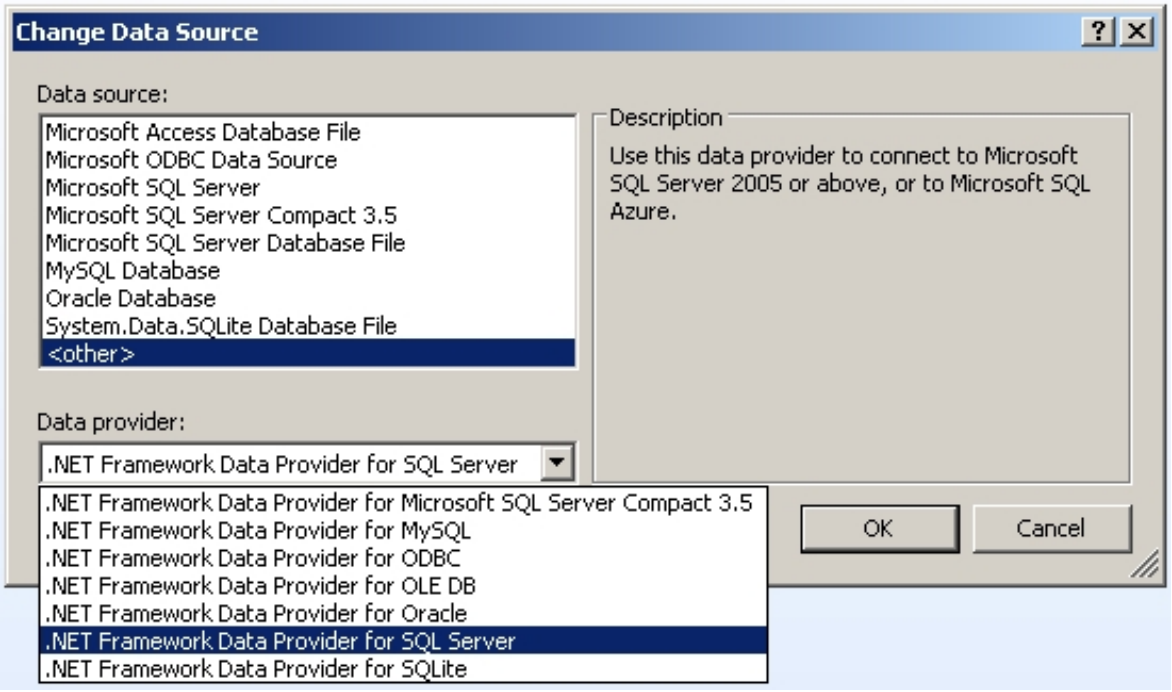
Step 2



Step 3 has a few more details, so for the first time we click on the **New Connection** button and we get the **Add Connection** window:

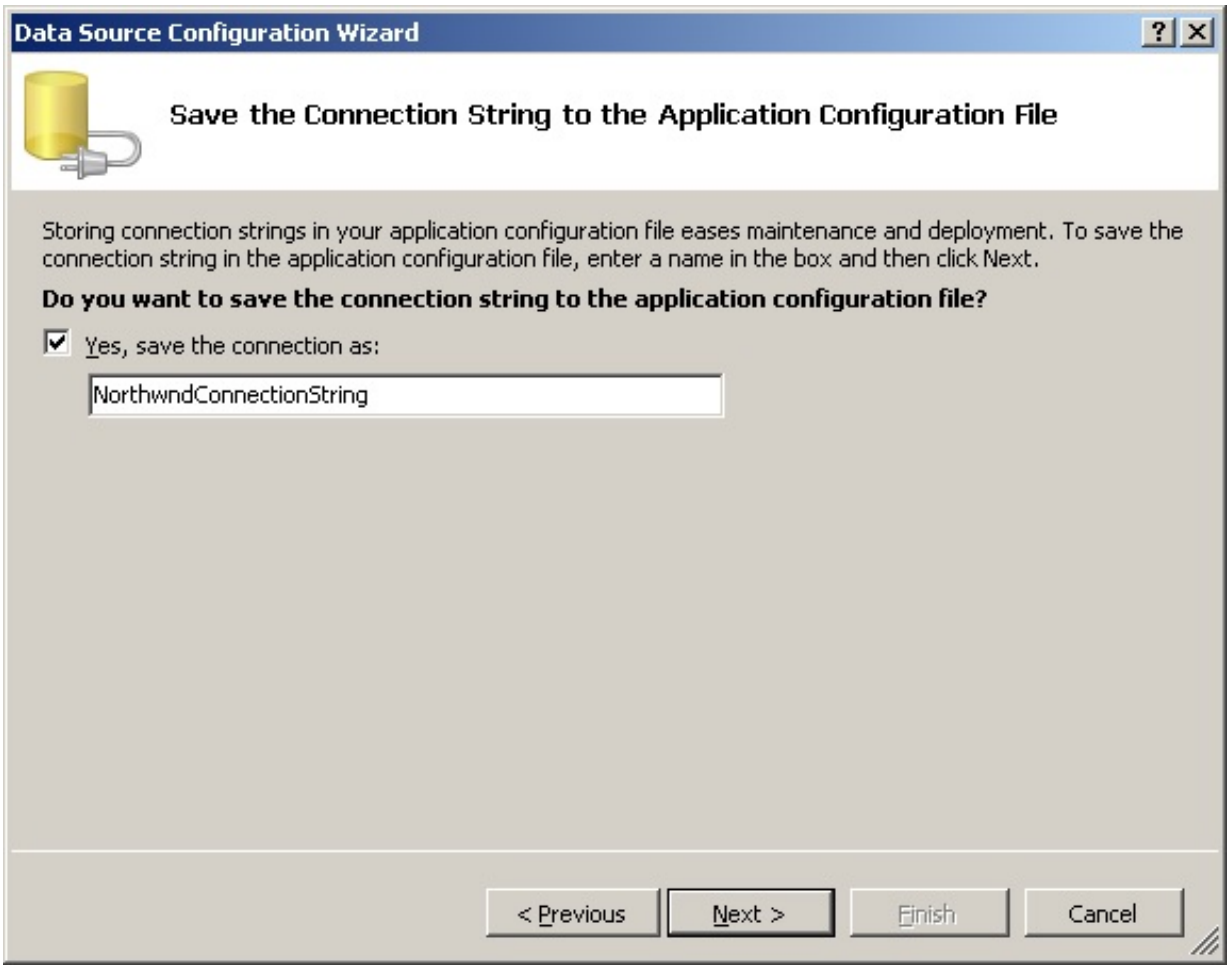


Here we will setup our SQL Server, the database and the necessary credentials. The first button at the top ([Data source:] **Change**), is the entry for the server type selection. By clicking it, we get the following window:

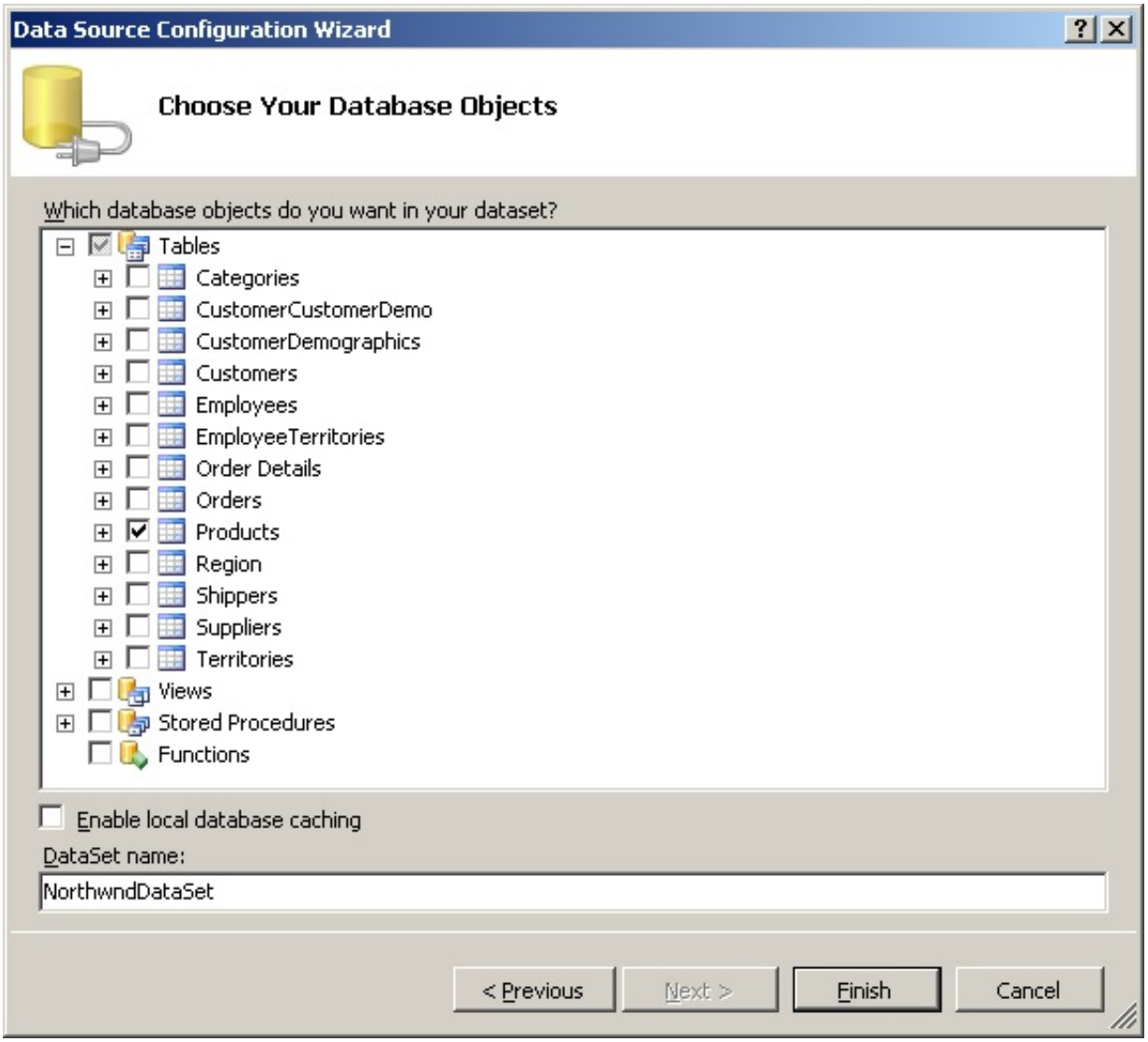


All the available **.NET Framework Data Providers** are possible and capable to work with our example!

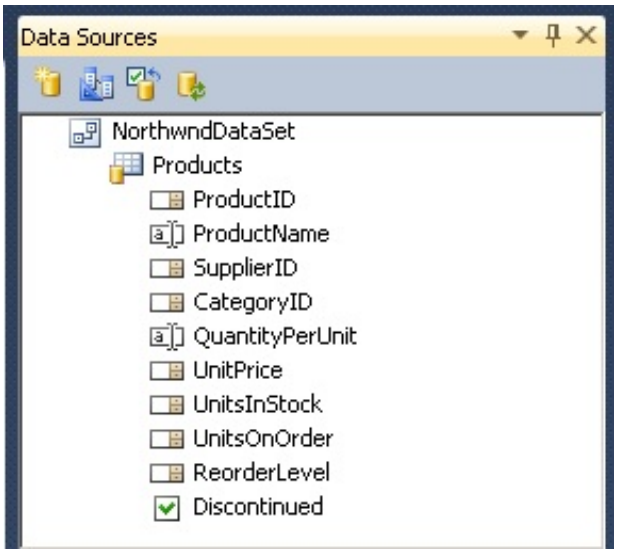
Finally, if all selections and choices are done right, we will get the window shown at the **Step 3**, with similar appearance. Pressing next at it, we get:



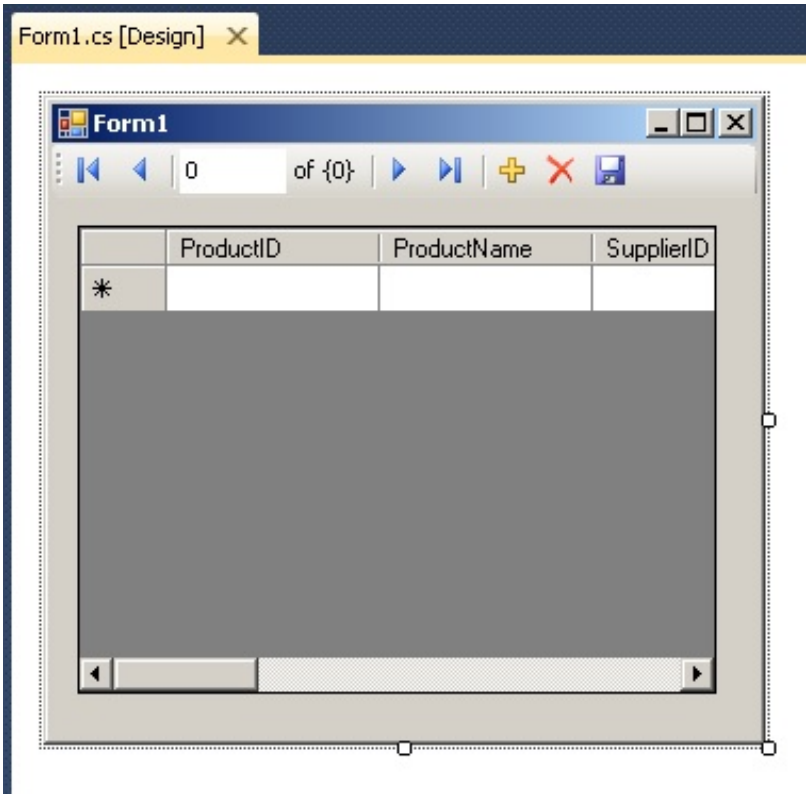
Step 4. Now, this is a name we should remember for a while, so make a short note of it!



Step 5. We are almost done with the dummy procedure. We select only `Products` datatable for our example, and click Finish button. Now the **Data Sources** window should appear something like this:

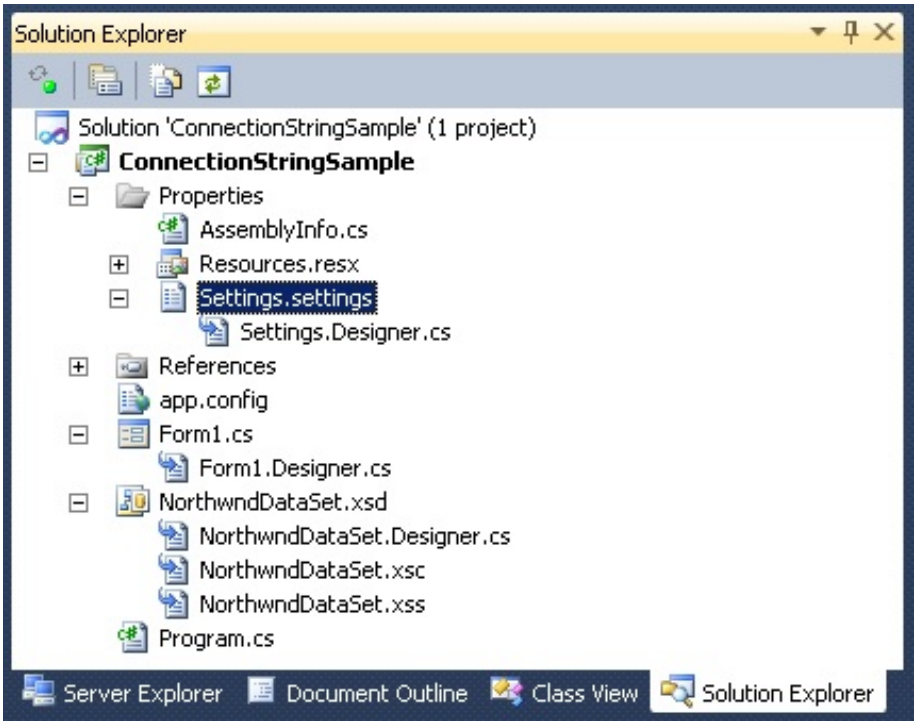


We now select our form designer window and then the word **Products** within the **Data Sources** window. We drag and drop the **Products** to our form and we get something similar to the next image:

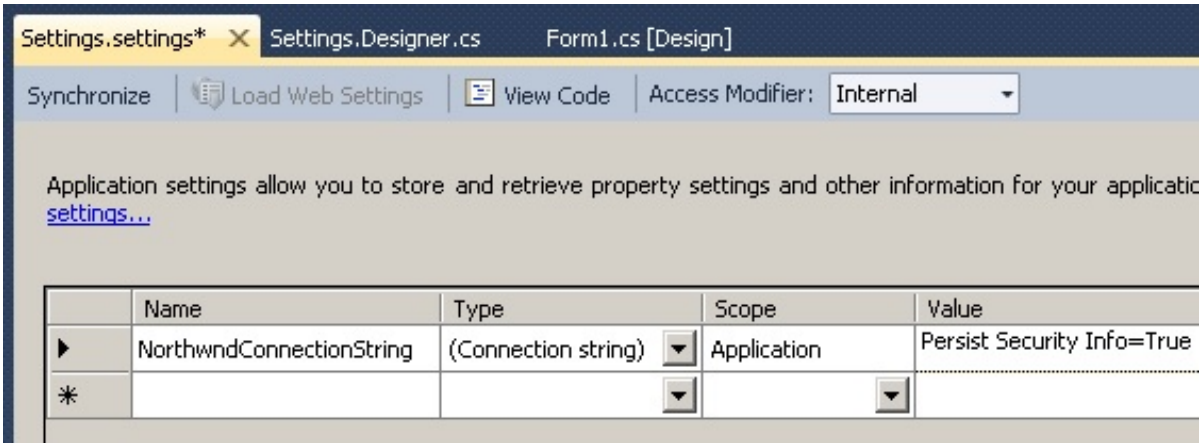


This is one of the miracles of the Visual Studio, or how to create an application with data management, without writing a single line of code! Yes, this is true, this is a fully working application!

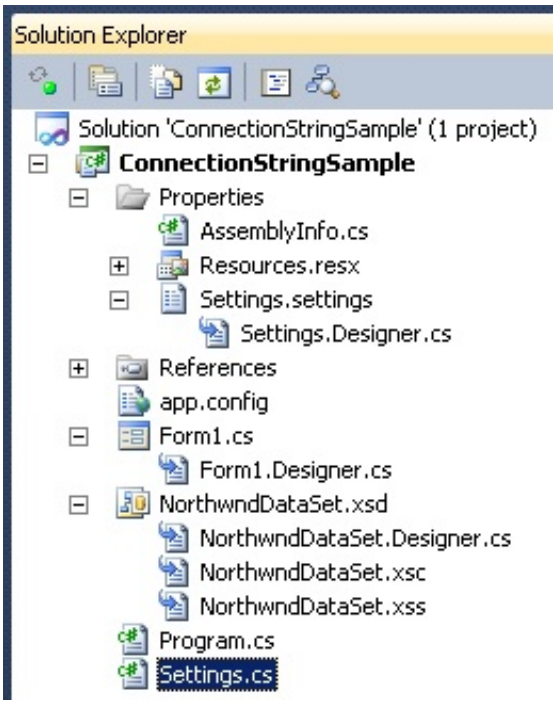
Let's go on now with the connection string. The **Solution Explorer** now appears as the following image:



Double click on the `Settings.settings` object to get the following image:



Here, we will remove almost all the information (sensitive or not) from the connection string and make it have a minimum content, as it appears at the `Value` column. Now save changes and at the toolbar of `Settings.settings` editor, click on **View Code**. This will create a new file with name *Settings.cs*.



This is a very sensitive and key role file. This is something like a base for the complete project! Every form to get opened within the designer, is using this file. If this file isn't correct, you cannot edit your forms! For now, do not change it!

Now let's get some information from the auto generated code of Visual Studio. We will search for the connection string within code automatically created. We've made a short note earlier, for the connection `string` variable, used by the **Data Source Wizard**, it is time to use it. Search within *NorthwndDataSet.Designer.cs* for the string "`NorthwndConnectionString`".

Hide Copy Code

```
[global::System.Diagnostics.DebuggerNonUserCodeAttribute()]
[global::System.CodeDom.Compiler.GeneratedCodeAttribute("System.Data.Design.TypedDataSetGenerator",
"4.0.0.0")]
private void InitConnection()
 {
 this._connection = new global::System.Data.SqlClient.SqlConnection();
 this._connection.ConnectionString
  = global::ConnectionStringSample.Properties.Settings.Default.NorthwndConnectionString;
 }
```

We will possibly find it at several places, but the one that appears at the code above, is the one we want. Placing the cursor at

the found `string "NorthwndConnectionString"` and pressing **F12** (context -> Go To Definition), will take us to the file *Settings.Designer.cs*, where we can see something similar to the following:

Hide Copy Code

```
[global::System.Configuration.ApplicationScopedSettingAttribute()]
[global::System.Diagnostics.DebuggerNonUserCodeAttribute()]
[global::System.Configuration.SpecialSettingAttribute(
global::System.Configuration.SpecialSetting.ConnectionString )]
[global::System.Configuration.DefaultSettingValueAttribute( "Persist Security Info=True" )]
public string NorthwndConnectionString
  {
  get
     {
     return ( ( string ) ( this["NorthwndConnectionString"] ) );
     }
  }
```

Now we are getting somewhere! Our connection string (`NorthwndConnectionString`), is a READ-ONLY property! Ok, behind a property, there is always (almost?) a variable. Which is our variable? The variable we need is:

Hide Copy Code

```
this["NorthwndConnectionString"]
```

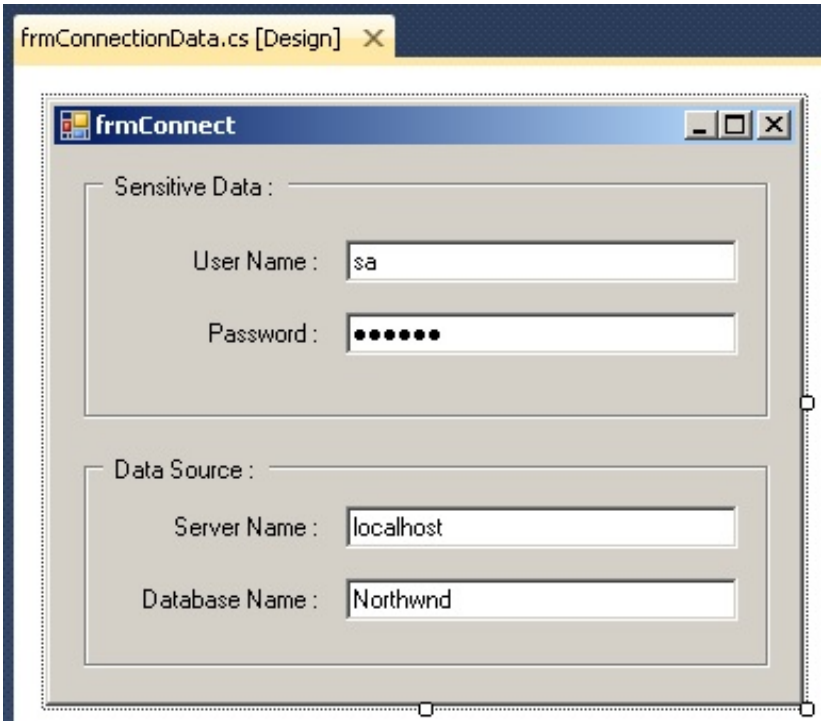
Great, we have it and we want to put it somewhere we can manipulate it! And the place for this is the file *Settings.cs*. Placing code within the constructor of the *Settings.cs*, will be efficient but it has to be self-sufficient, i.e., must not use code from other classes or functions, mainly can be only variable initialization with constant values. So we need something else to help us achieve the "Dynamic" promise we've made. Looking at the comments at the top of the class [from the file *Settings.cs*, **internal sealed partial class Settings**], we can see that we can use a few events, writing our event handlers. Most 'handy' of all appears to be the `SettingsLoaded` event. So we add a line at the bottom of the empty constructor  , to introduce our event handler.

Hide Copy Code

```
this.SettingsLoaded += [press TAB]
```

Visual Studio automation will do even most of the typing for us. Write the code up to the equal character and press TAB key twice. The code is entered for both event and event handler!

Now a little touch of design, again, to accomplish the purpose of all this document. We need another form, to let the user enter her/his sensitive data, like the one in the next image:



This way, the user or someone in charge of installation, will have the ability to setup the important data to let the application connect to database server. Of course, these data have to be saved somewhere (some kind of *.ini* file, or the registry, etc.).

We also need a way to give the connection data from one form to the other. So we add a new class, `public` and `static` to carry our data, as follows:

Hide Copy Code

```
using System;

namespace ConnectionStringSample
   {
   public static class GlobalData
      {
      public static String strConnectionUserName = "sa";
      public static String strConnectionPassword = "123456";
      public static String strConnectionServerName = "localhost";
      public static String strConnectionDatabaseName = "Northwnd";
      }
   }
```

This file we call *GlobalData.cs*. You should notice that we have default values for all variables, even if they are invalid (values).

Now the key role file *Settings.cs*. The complete file is as follows:

Hide Shrink Copy Code

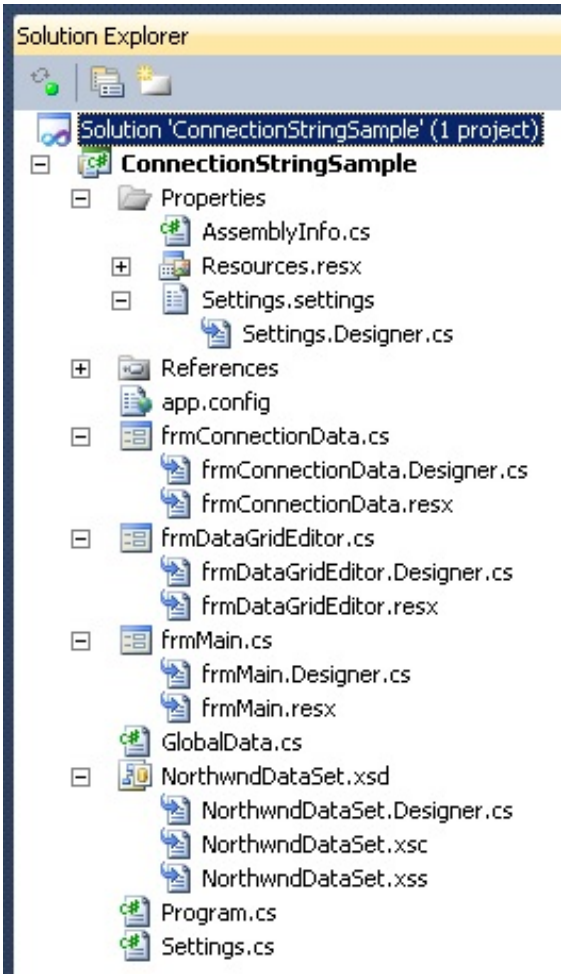
```
namespace ConnectionStringSample.Properties
   {

   // This class allows you to handle specific events on the settings class:
   //  The SettingChanging event is raised before a setting's value is changed.
   //  The PropertyChanged event is raised after a setting's value is changed.
   //  The SettingsLoaded event is raised after the setting values are loaded.
   //  The SettingsSaving event is raised before the setting values are saved.
   internal sealed partial class Settings
      {
      public Settings()
         {
         // // To add event handlers for saving and changing settings, uncomment the lines below:
```

```
        // this.SettingChanging += this.SettingChangingEventHandler;
        // this.SettingsSaving += this.SettingsSavingEventHandler;

        this.SettingsLoaded += new System.Configuration.SettingsLoadedEventHandler(
Settings_SettingsLoaded );
        }

     void Settings_SettingsLoaded( object sender, System.Configuration.SettingsLoadedEventArgs e )
        {
        this["NorthwndConnectionString"] = "Data Source=" + GlobalData.strConnectionServerName +
";"
                                           + "Initial Catalog=" +
GlobalData.strConnectionDatabaseName + ";"
                                           + "Persist Security Info=True;"
                                           + "User ID=" + GlobalData.strConnectionUserName + ";"
                                           + "Password=" + GlobalData.strConnectionPassword + ";";
        }

     private void SettingChangingEventHandler( object sender,
System.Configuration.SettingChangingEventArgs e )
        {
        // Add code to handle the SettingChangingEvent event here.
        }

     private void SettingsSavingEventHandler( object sender, System.ComponentModel.CancelEventArgs
e )
        {
        // Add code to handle the SettingsSaving event here.
        }
     }
  }
```

The important and new lines are these in bold typeface. This will do the job!

For the sake of completeness, we need another form to act as a dispatcher. At the beginning, we will call the `ConnectionData` form (*frmConnectionData.cs* - `frmConnect` form shown before) and when this form closes, it will call the form that allows the user to browse and edit the data from the database (*frmDataGridEditor.cs* formerly *Form1.cs*).

At this point, our solution appears as the following image:



A few minor changes have been done to complete the later points of the above text, such as starting form within *Program.cs* and a little code within *frmMain.cs* to switch forms. The complete project (source) is available for download (C# and VB.NET for Visual Studio 2010).

If the user entry for the connection is invalid data, she/he gets an empty grid.

Conclusion

Though the **Data Source Configuration Wizard** is a great tool that gives the programmer advanced database management abilities, the need for something more remains and that is the **Connection String** and how to change/update it at runtime. The powerful class `Settings` and the related source file *Settings.cs* are there to help us solve the problem. We also take a look at the auto-generated code from the **Data Source Configuration Wizard**. This way, we discovered the internally used variable for the connection string. The combination of the above is the solution we are searching for.

History

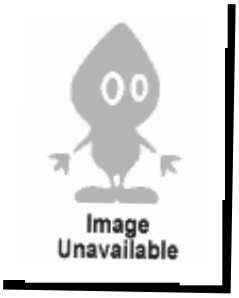
- 20140524: Conclusion paragraph added and single compressed file with sources available for download (including both VB.NET and C#), so when the CodeProject **Browse Code** feature is used, both projects will be available at the same time.
- 20140408: VB.NET source code added for download
- 20140406: First edition (including C# code for download)

License

Share

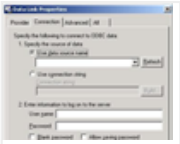
EMAIL TWITTER

About the Author



Nick Sagriotis No Biography provided
Software Developer
Greece

You may also be interested in...



ADO Connection Strings

Follow the Money: Big Data ROI and Inline Analytics

Dynamic connection string

SAPrefs - Netscape-like Preferences Dialog

Dynamic Expresso

Window Tabs (WndTabs) Add-In for DevStudio

Comments and Discussions

You must Sign In to use this message board.

Search Comments Go

First Prev Next

Excellent solution
Bassem Akl 20-Jan-16 14:07

Dynamic connection string tried in vb.net
padmaraj1983 21-Aug-14 20:20

Re: Dynamic connection string tried in vb.net
Nick Sagriotis 14-Sep-14 22:46

Port to MVC for Web App?
Terri Morgan 3-May-14 12:39

Re: Port to MVC for Web App?
Nick Sagriotis 4-May-14 21:45

Re: Port to MVC for Web App?
Terri Morgan 6-May-14 5:57

My vote of 5
CSharls 8-Apr-14 21:27

Refresh 1

General News Suggestion Question Bug Answer Joke Praise Rant Admin

Use Ctrl+Left/Right to switch messages, Ctrl+Up/Down to switch threads, Ctrl+Shift+Left/Right to switch pages.