http://www.sqlservercentral.com/articles/T-SQL/101154/
Printed 2013/08/14 07:29AM

How to Compile, Deploy and Consume a SQL Server CLR Assembly

By Stan Kulp, 2013/08/01

Databases that include Social Security or credit card numbers are vulnerable to identity theft and represent a liablity for any organization possessing either one. For that reason it makes sense to store Social Security and credit card numbers as encrypted values. Use of a CLR (Common Language Runtime) assembly for encryption can be useful in situations where you are trying to match the encryption algorithm being used by an extenal organization with which you exchange data.

Overview

This article will demonstrate how to:

1. Compile a CLR assembly named AES_EncryptDecrypt using Microsoft Visual C# 2010 Express
2. Deploy the assembly to SQL Server
3. Create SQL Server functions named Encrypt and Decrypt that reference corresponding public methods in the CLR assembly
4. Use the Encrypt SQL Server function to insert encrypted data into a SQL Server table
5. Use the Decrypt SQL Server function to extract decrypted data from the table

SQL Server CLR Assembly Tutorial

The C# console application code below encrypts and decrypts a string.

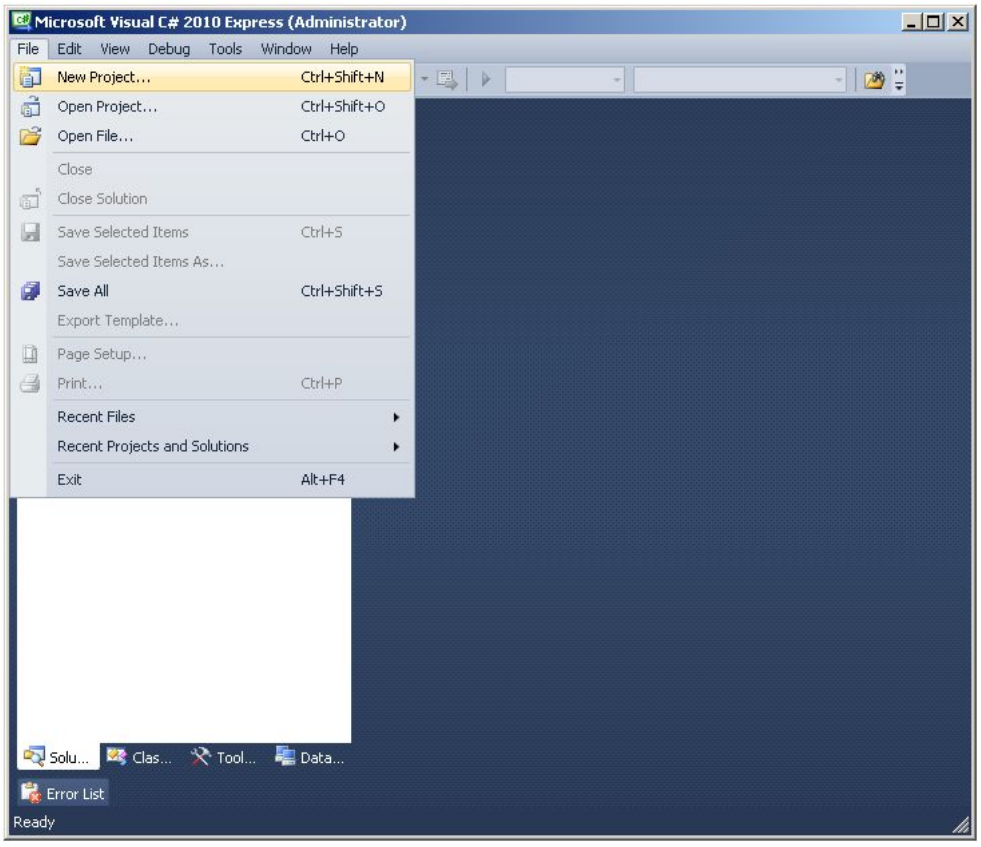
```
using System;
using System.IO;
using System.Security.Cryptography;

namespace AES_EncryptDecrypt
{
    sealed class AES_EncryptDecryptConsole
    {
        private static void Main()
        {
            string decryptionKey = "ayb&e#i&BWLGMe2V";

            Console.Write("    Enter String: ");
            string testString = Console.ReadLine();

            string encrypted = AES_EncryptDecryptLibrary.AES_Encrypt(testString, decryptionKey);
            Console.WriteLine("Encrypted String: " + encrypted);

            string decrypted = AES_EncryptDecryptLibrary.AES_Decrypt(encrypted, decryptionKey);
            Console.WriteLine("Decrypted String: " + decrypted);

            Console.ReadLine();

        }
    }

    public sealed class AES_EncryptDecryptLibrary
    {
        public static string AES_Encrypt(string input, string pass)
        {
            try
            {
                return Convert.ToBase64String(EncryptStringToBytes(input, System.Text.Encoding.Default.GetBytes(pass)));
            }
            catch (Exception)
            {
                return "";
            }
        }
        public static string AES_Decrypt(string input, string pass)
        {
            try
            {
                return DecryptStringFromBytes(Convert.FromBase64String(input), System.Text.Encoding.Default.GetBytes(pass));
            }
            catch (Exception)
            {
                return "";
            }
        }

        private static byte[] EncryptStringToBytes(string plainText, byte[] Key)
        {
            return EncryptStringToBytes(plainText, Key, null);
        }

        private static byte[] EncryptStringToBytes(string plainText, byte[] Key, byte[] IV)
        {
            // Check arguments.
            if ((plainText == null) || (plainText.Length <= 0))
            {
                throw (new ArgumentNullException("plainText"));
            }
            if ((Key == null) || (Key.Length <= 0))
            {
```

```
            throw (new ArgumentNullException("Key"));
        }
        // Create an RijndaelManaged object
        // with the specified key and IV.
        RijndaelManaged rijAlg = new RijndaelManaged();
        rijAlg.Key = Key;
        if (!(IV == null))
        {
            if (IV.Length > 0)
            {
                rijAlg.IV = IV;
            }
            else
            {
                rijAlg.Mode = CipherMode.ECB;
            }
        }
        else
        {
            rijAlg.Mode = CipherMode.ECB;
        }
        byte[] encrypted = null;
        // Create a decrytor to perform the stream transform.
        ICryptoTransform encryptor = rijAlg.CreateEncryptor();
        // Create the streams used for encryption.
        encrypted = encryptor.TransformFinalBlock(System.Text.Encoding.Default.GetBytes(plainText), 0, plainText.Length); //msEncr
        // Return the encrypted bytes from the memory stream.
        return encrypted;
    }

    private static byte[] EncryptBytesToBytes(byte[] Input, byte[] Key)
    {
        return EncryptBytesToBytes(Input, Key, null);
    }

    private static byte[] EncryptBytesToBytes(byte[] Input, byte[] Key, byte[] IV)
    {
        // Check arguments.
        if ((Input == null) || (Input.Length <= 0))
        {
            throw (new ArgumentNullException("plainText"));
        }
        if ((Key == null) || (Key.Length <= 0))
        {
            throw (new ArgumentNullException("Key"));
        }
        // Create an RijndaelManaged object
        // with the specified key and IV.
        RijndaelManaged rijAlg = new RijndaelManaged();
        rijAlg.Key = Key;
        if (!(IV == null))
        {
            if (IV.Length > 0)
            {
                rijAlg.IV = IV;
            }
            else
            {
                rijAlg.Mode = CipherMode.ECB;
            }
        }
        else
        {
            rijAlg.Mode = CipherMode.ECB;
        }
        byte[] encrypted = null;
        // Create a decrytor to perform the stream transform.
        ICryptoTransform encryptor = rijAlg.CreateEncryptor(rijAlg.Key, rijAlg.IV);

        encrypted = encryptor.TransformFinalBlock(Input, 0, Input.Length);
        // Return the encrypted bytes from the memory stream.
        return encrypted;
    }

    private static string DecryptStringFromBytes(byte[] cipherText, byte[] Key)
    {
        return DecryptStringFromBytes(cipherText, Key, null);
    }

    private static string DecryptStringFromBytes(byte[] cipherText, byte[] Key, byte[] IV)
    {
        // Check arguments.
        if ((cipherText == null) || (cipherText.Length <= 0))
        {
            throw (new ArgumentNullException("cipherText"));
        }
        if ((Key == null) || (Key.Length <= 0))
        {
            throw (new ArgumentNullException("Key"));
        }
        // Create an RijndaelManaged object
        // with the specified key and IV.
        RijndaelManaged rijAlg = new RijndaelManaged();
        rijAlg.Key = Key;
        rijAlg.Mode = CipherMode.CBC;
        if (!(IV == null))
        {
```

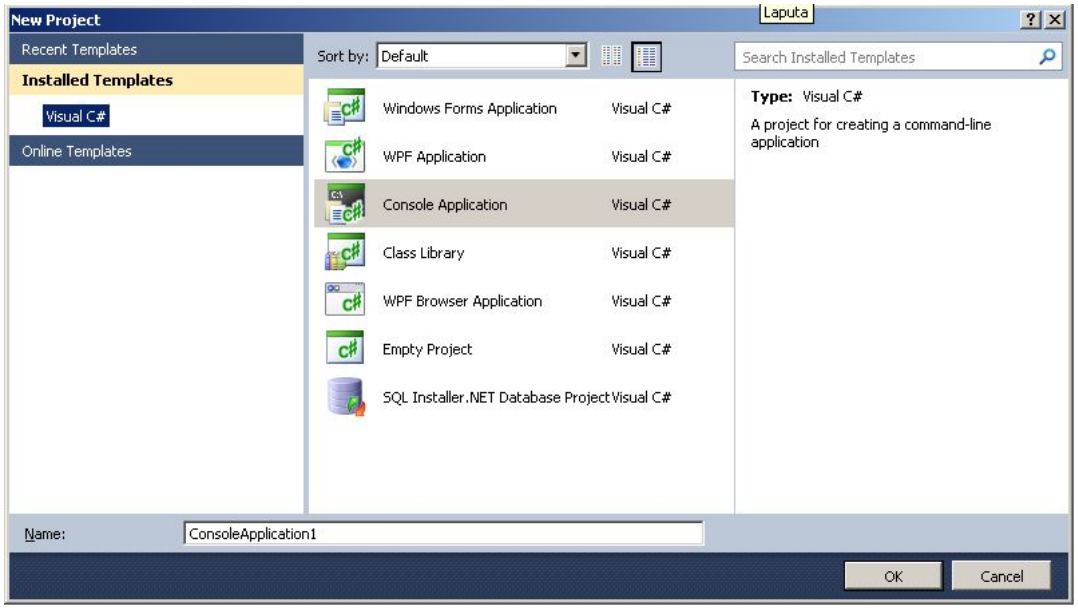
```
                if (IV.Length > 0)
                {
                    rijAlg.IV = IV;
                }
                else
                {
                    rijAlg.Mode = CipherMode.ECB;
                }
            }
            else
            {
                rijAlg.Mode = CipherMode.ECB;
            }
            string plaintext = null;
            // Create a decrytor to perform the stream transform.
            ICryptoTransform decryptor = rijAlg.CreateDecryptor(rijAlg.Key, rijAlg.IV);

            // Create the streams used for decryption.
            MemoryStream msDecrypt = new MemoryStream(cipherText);
            CryptoStream csDecrypt = new CryptoStream(msDecrypt, decryptor, CryptoStreamMode.Read);
            StreamReader srDecrypt = new StreamReader(csDecrypt);
            // Read the decrypted bytes from the decrypting stream
            // and place them in a string.
            plaintext = srDecrypt.ReadToEnd();
            return plaintext;
        }

        private static byte[] DecryptBytesFromBytes(byte[] cipherText, byte[] Key)
        {
            return DecryptBytesFromBytes(cipherText, Key, null);
        }

        private static byte[] DecryptBytesFromBytes(byte[] cipherText, byte[] Key, byte[] IV)
        {
            // Check arguments.
            if ((cipherText == null) || (cipherText.Length <= 0))
            {
                throw (new ArgumentNullException("cipherText"));
            }
            if ((Key == null) || (Key.Length <= 0))
            {
                throw (new ArgumentNullException("Key"));
            }
            // Create an RijndaelManaged object
            // with the specified key and IV.
            RijndaelManaged rijAlg = new RijndaelManaged();
            rijAlg.Key = Key;
            if (!(IV == null))
            {
                if (IV.Length > 0)
                {
                    rijAlg.IV = IV;
                }
                else
                {
                    rijAlg.Mode = CipherMode.ECB;
                }
            }
            else
            {
                rijAlg.Mode = CipherMode.ECB;
            }
            byte[] output = null;
            // Create a decrytor to perform the stream transform.
            ICryptoTransform decryptor = rijAlg.CreateDecryptor(rijAlg.Key, rijAlg.IV);
            // Create the streams used for decryption.
            MemoryStream msDecrypt = new MemoryStream(cipherText);
            CryptoStream csDecrypt = new CryptoStream(msDecrypt, decryptor, CryptoStreamMode.Read);

            StreamReader srDecrypt = new StreamReader(csDecrypt);
            // Read the decrypted bytes from the decrypting stream
            // and place them in a string.
            MemoryStream ms = new MemoryStream();
            while (!srDecrypt.EndOfStream)
            {
                ms.WriteByte((byte)(srDecrypt.Read()));
            }
            ms.Position = 0;
            output = ms.ToArray();
            return output;
        }
    }
}
```

(Attached to this article is the same console application written in VB.NET. If you prefer VB.NET, download the code and adapt the following tutorial to it.)

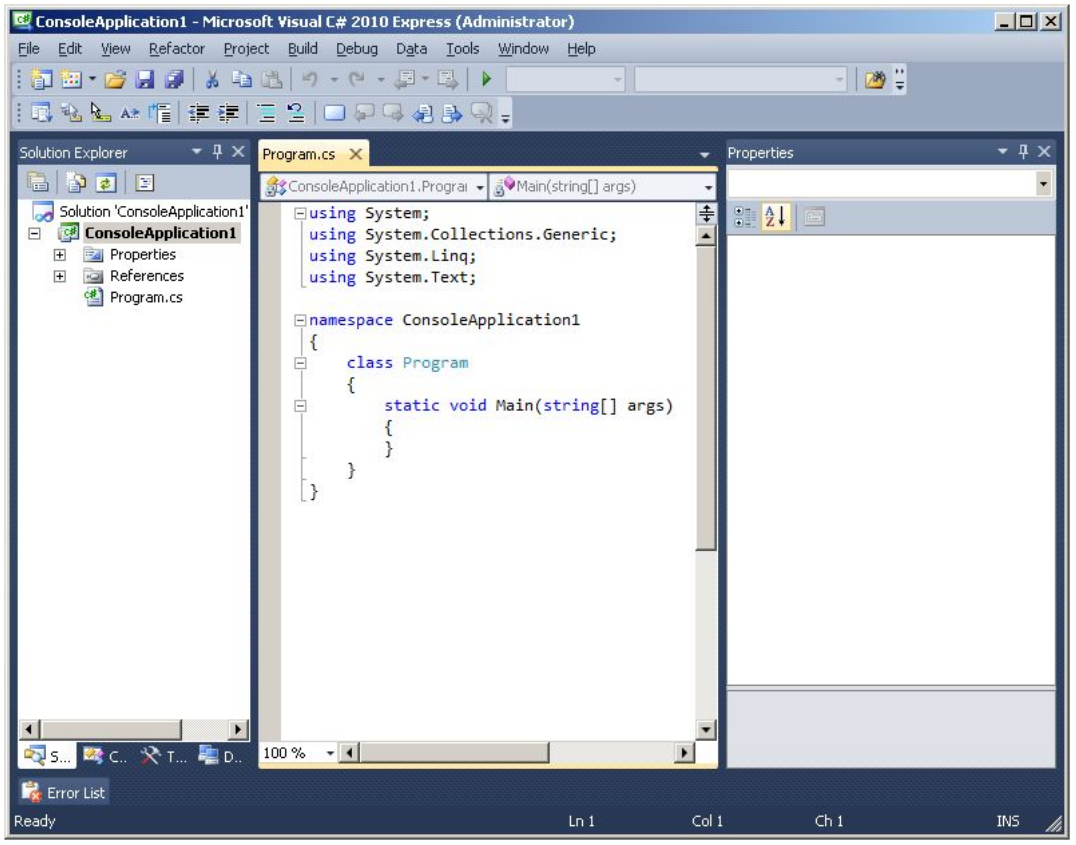
To compile and execute the code, create a new project in Microsoft Visual C# 2010 Express (downloadable at http://go.microsoft.com/?linkid=9709939).



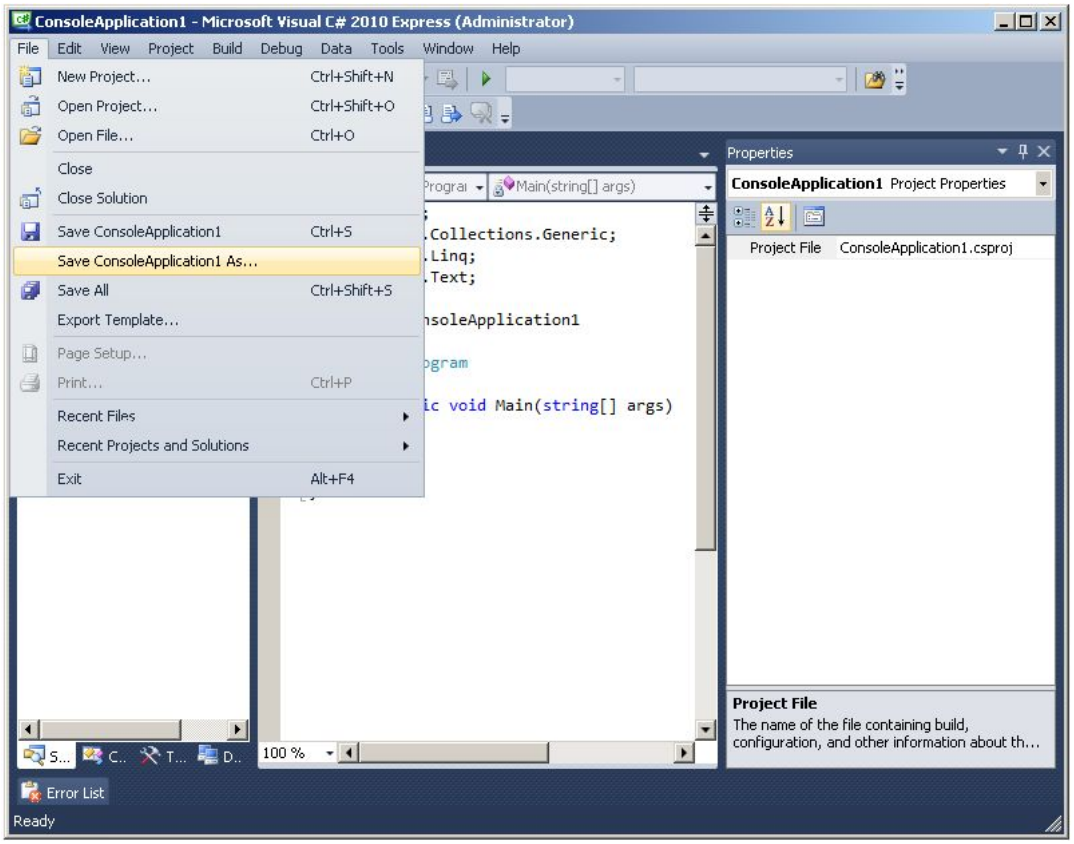
Select the Console Application template and click OK.



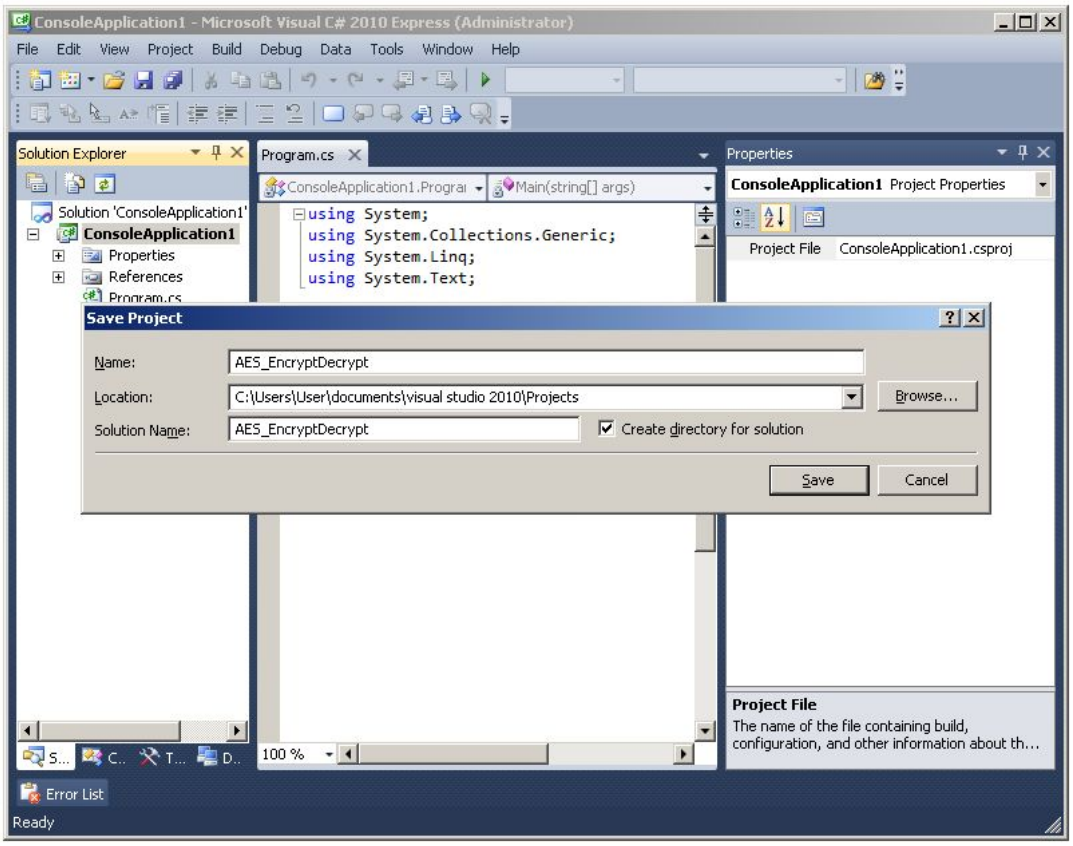
A default C# program file named Programs.cs is created.



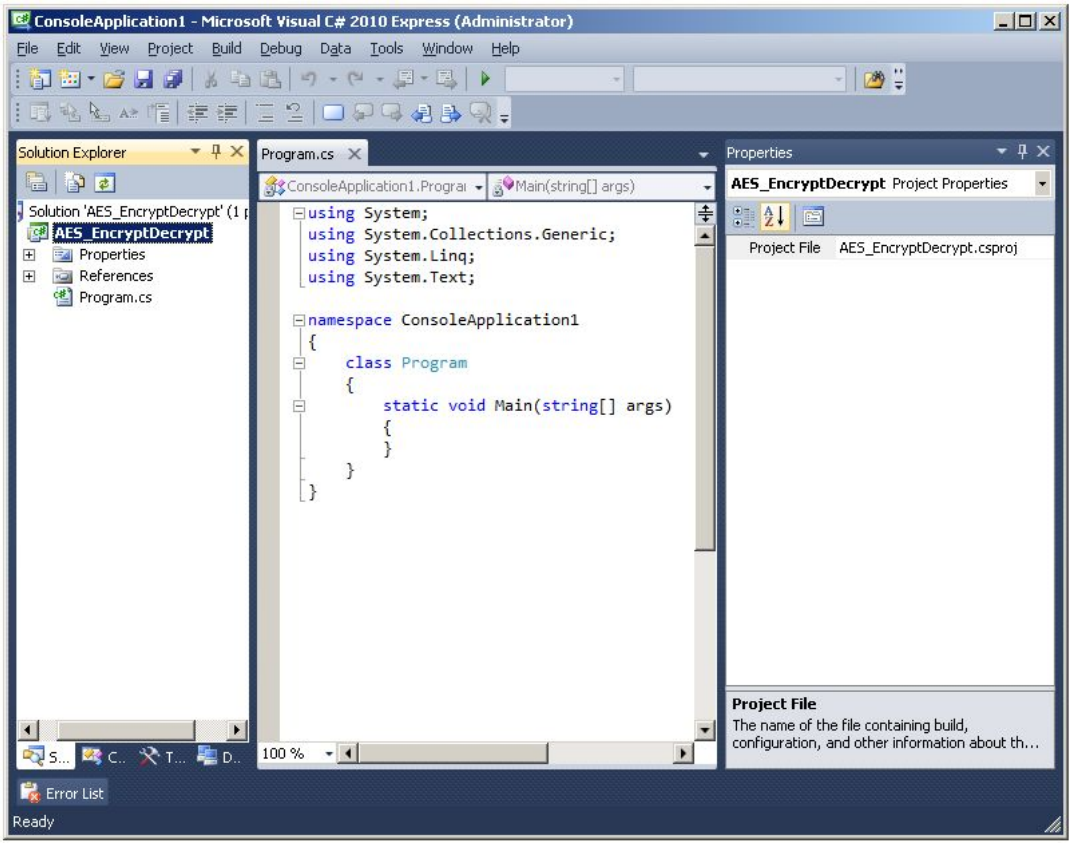
To give the project a meaningful name, click on File-SaveConsoleApplication1 As...



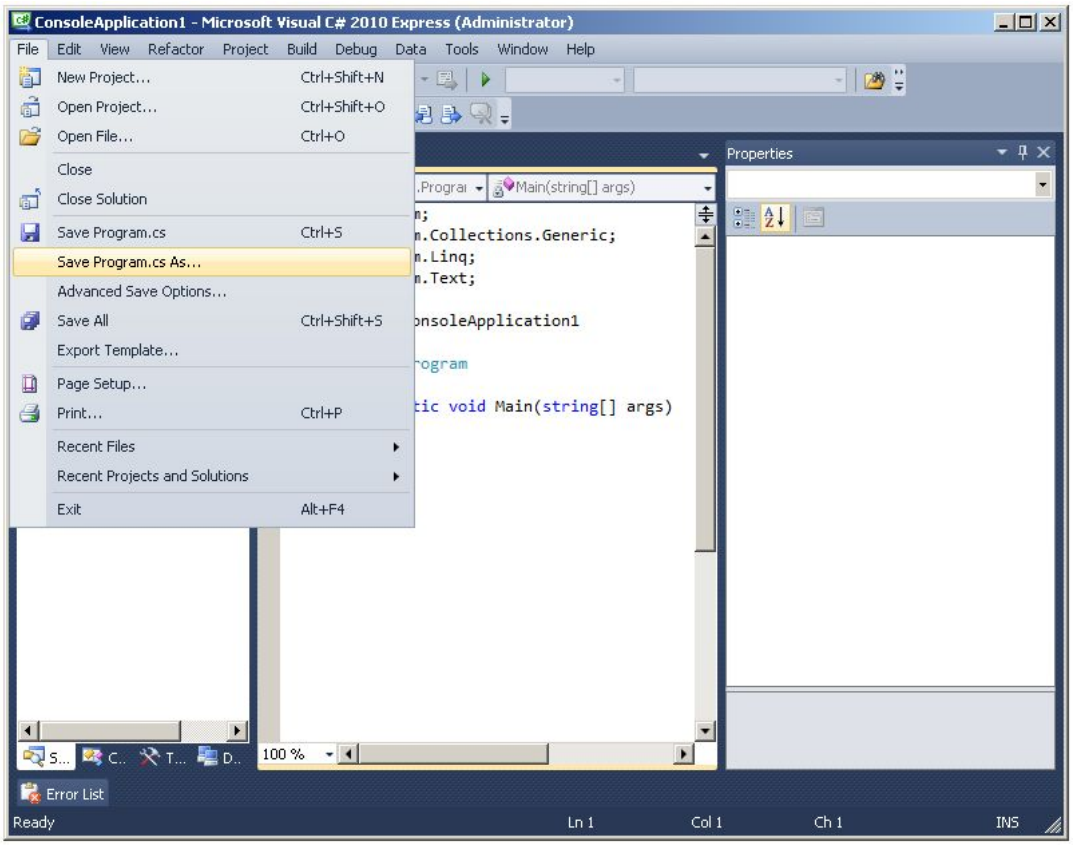
Enter AES_EncryptDecrypt into the project name field.



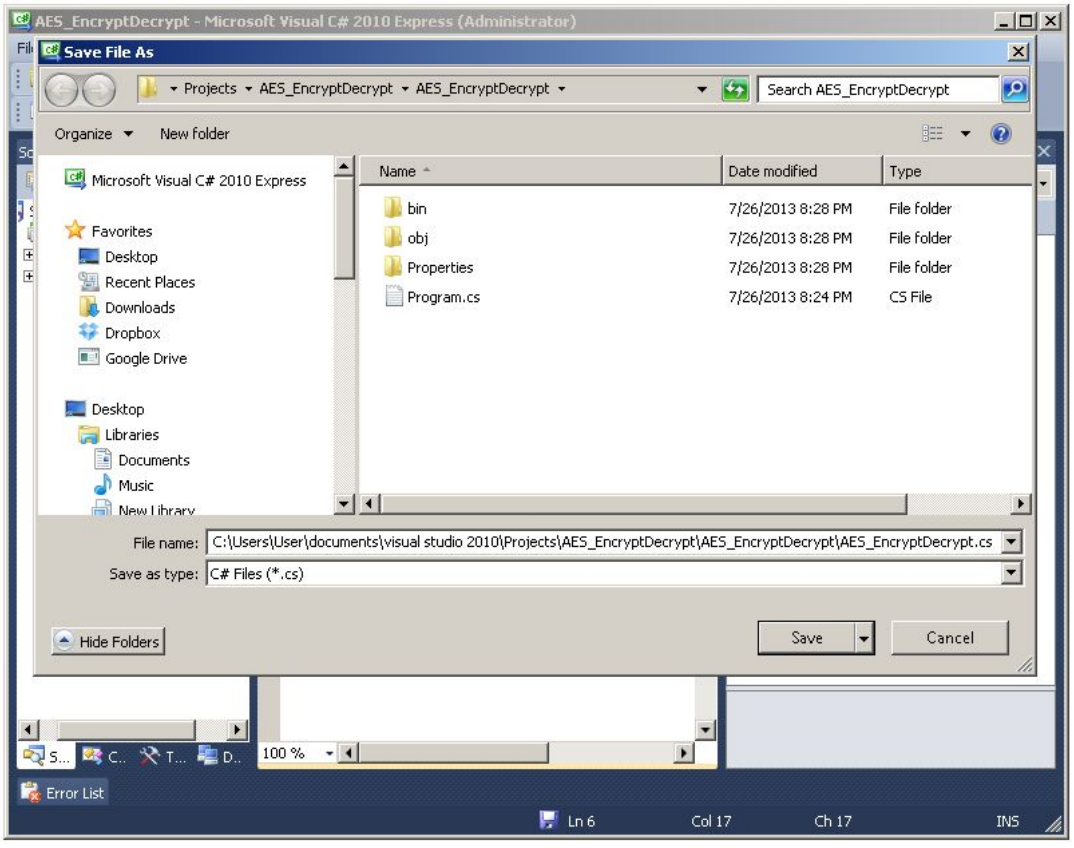
The name of the project has now been changed. Next, change the default name of the C# program file Program.cs...



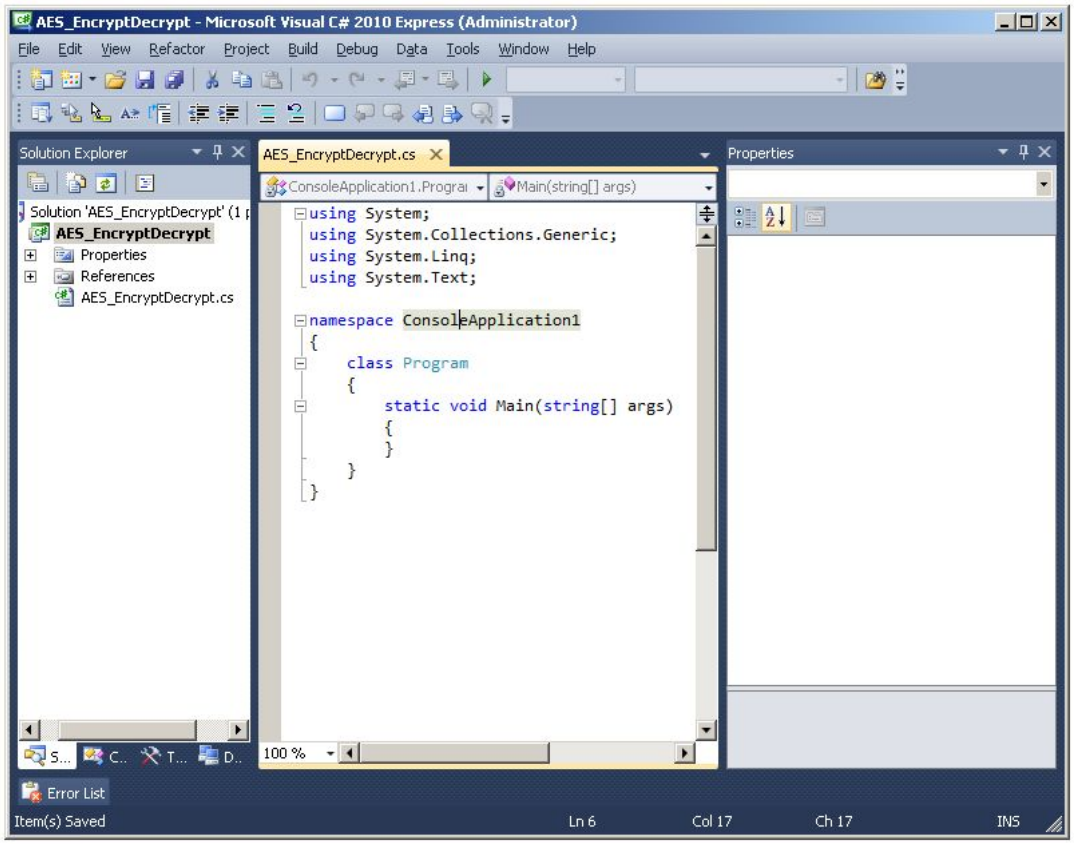
...by clicking on File-Save Program.cs As...



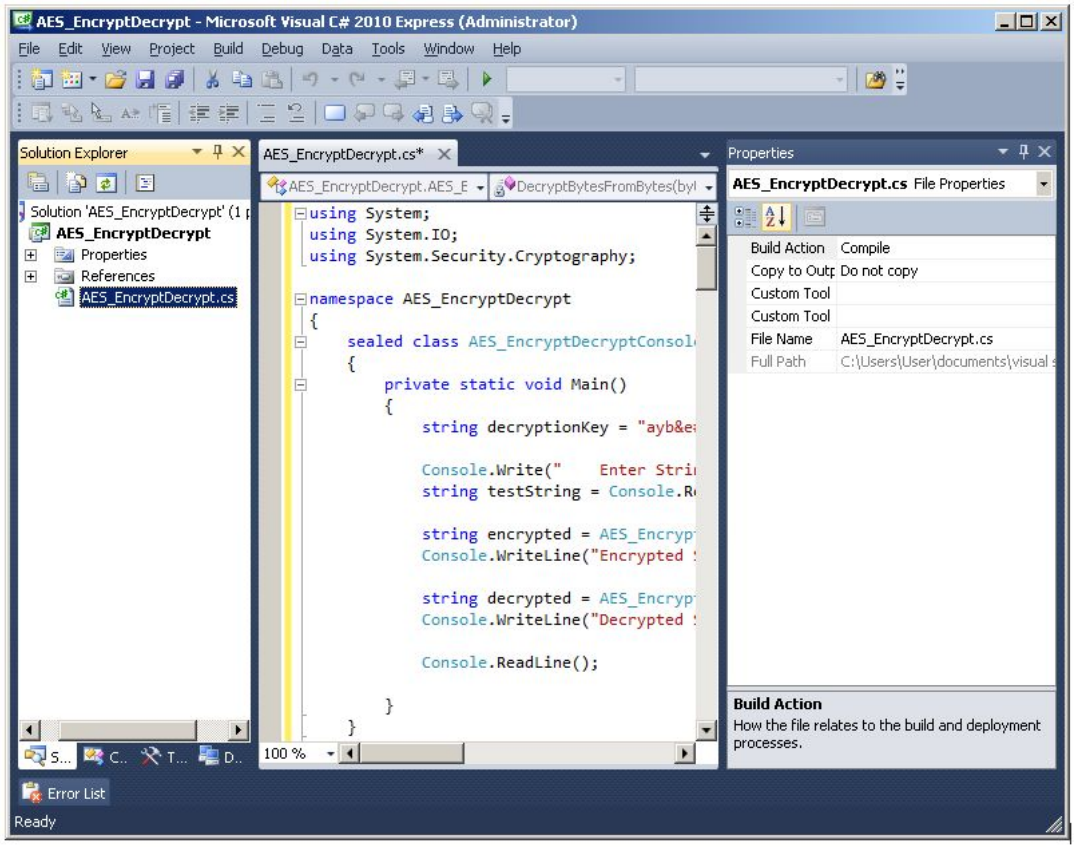
Change the file name to AES_EncryptDecrypt.cs.



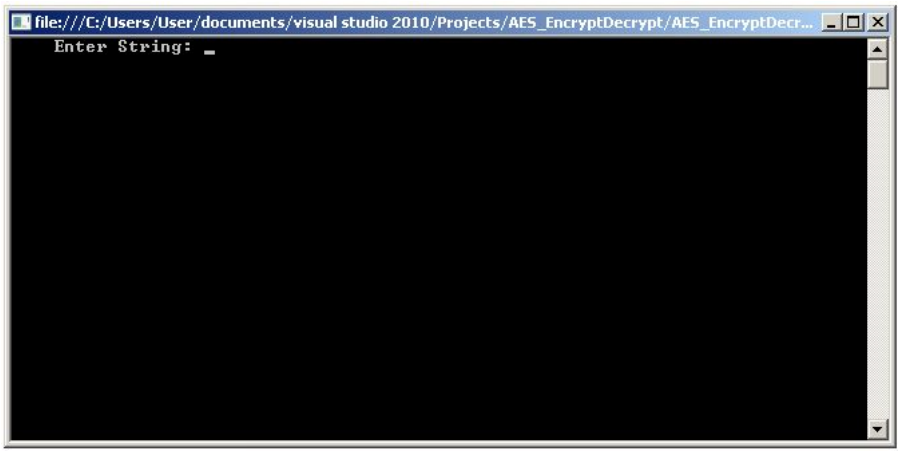
The name of the C# program file has been changed, but it still contains the template code.



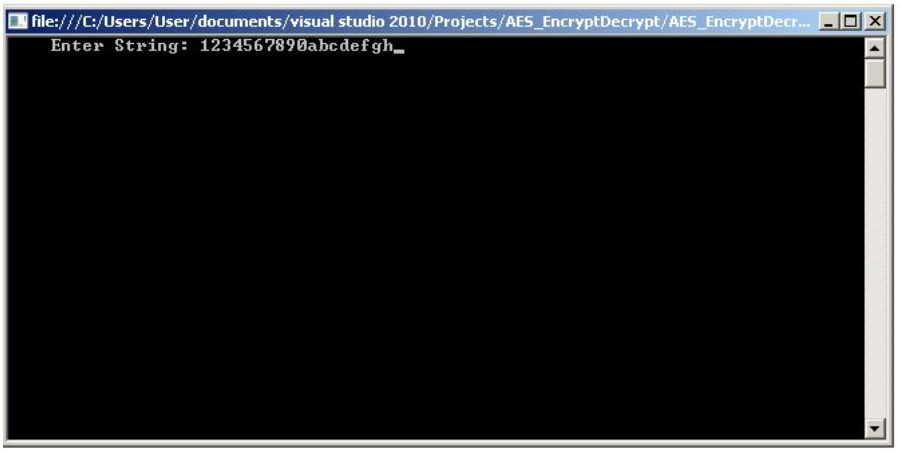
Copy the C# console application code in the text box at the top of this article and paste it over the template code in the AES_EncryptDecrypt.cs file.



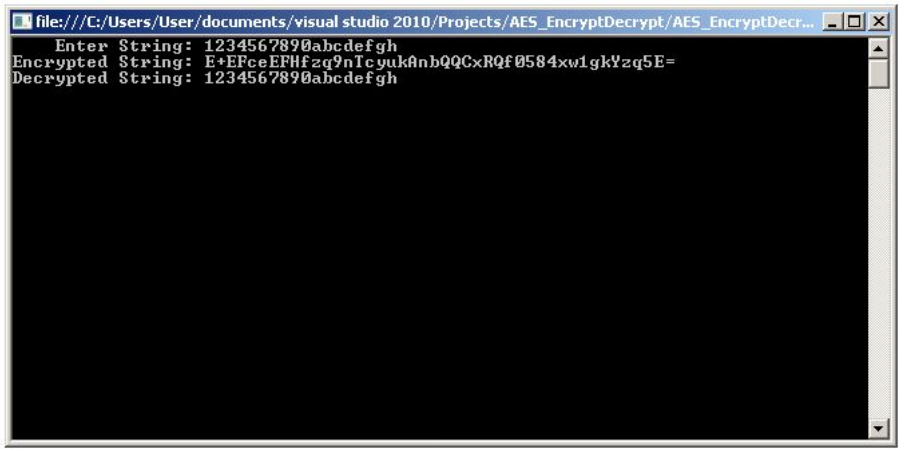
Click the "Start Debugging" button  to execute the project. Upon execution, the application will generate a command console.



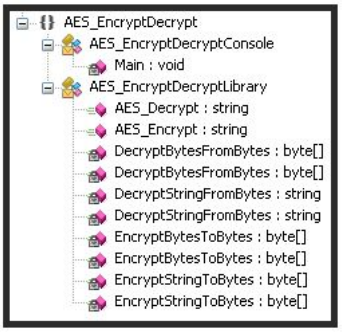
Enter a string of characters at the prompt and press the Enter key.



The application prints the encrypted string and the result of decrypting the encrypted string, which should match the string you entered.



There are two classes in the C# code, AES_EncryptDecryptConsole and AES_EncryptDecryptLibrary.



The AES_EncryptDecryptConsole class contains only the Main() method, which handles console IO and calls public methods in the AES_EncryptDecryptLibrary class.

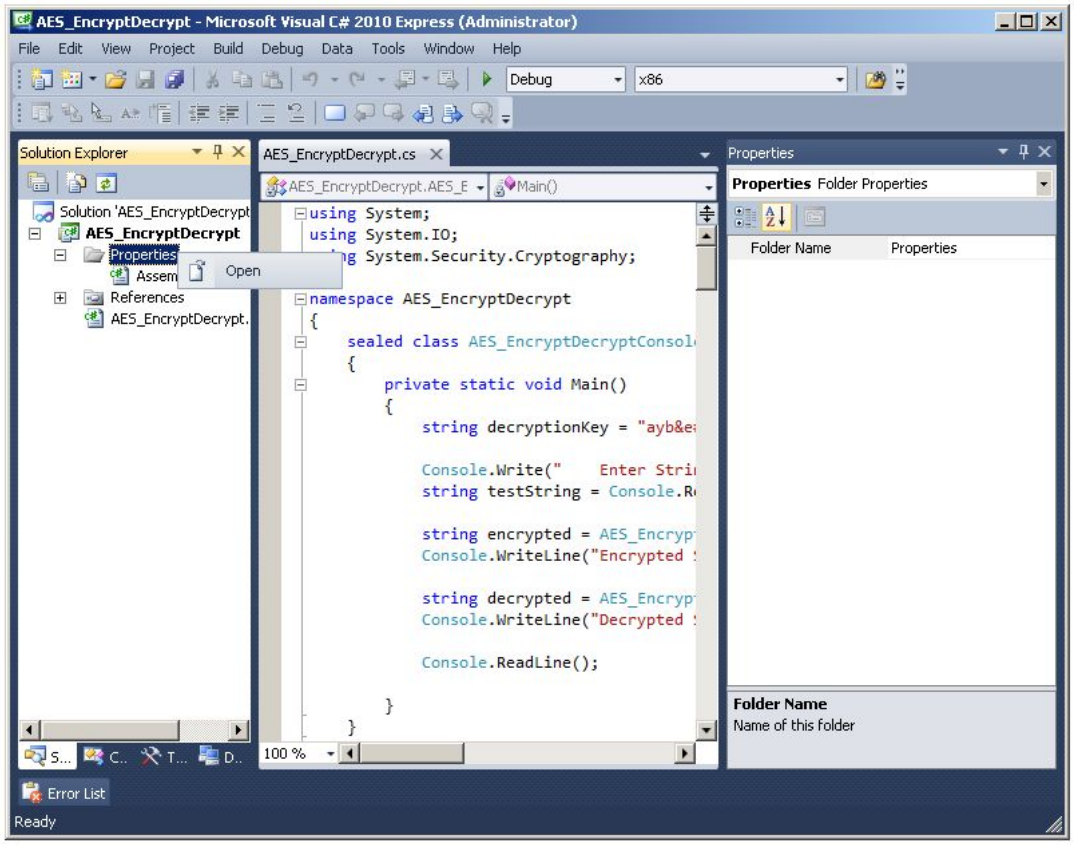
```
sealed class AES_EncryptDecryptConsole
{

    private static void Main()
    {

        string decryptionKey = "ayb&e#i&BWLGMe2V";

        Console.Write("    Enter String: ");
        string testString = Console.ReadLine();

        string encrypted = AES_EncryptDecryptLibrary.AES_Encrypt(testString, decryptionKey);
        Console.WriteLine("Encrypted String: " + encrypted);

        string decrypted = AES_EncryptDecryptLibrary.AES_Decrypt(encrypted, decryptionKey);
        Console.WriteLine("Decrypted String: " + decrypted);

        Console.ReadLine();

    }
}
```

The AES_EncryptDecryptLibrary class contains two public methods, AES_Encrypt and AES_Decrypt. A string and the encryption key are passed to each method, and an encrypted or decrypted string is returned.

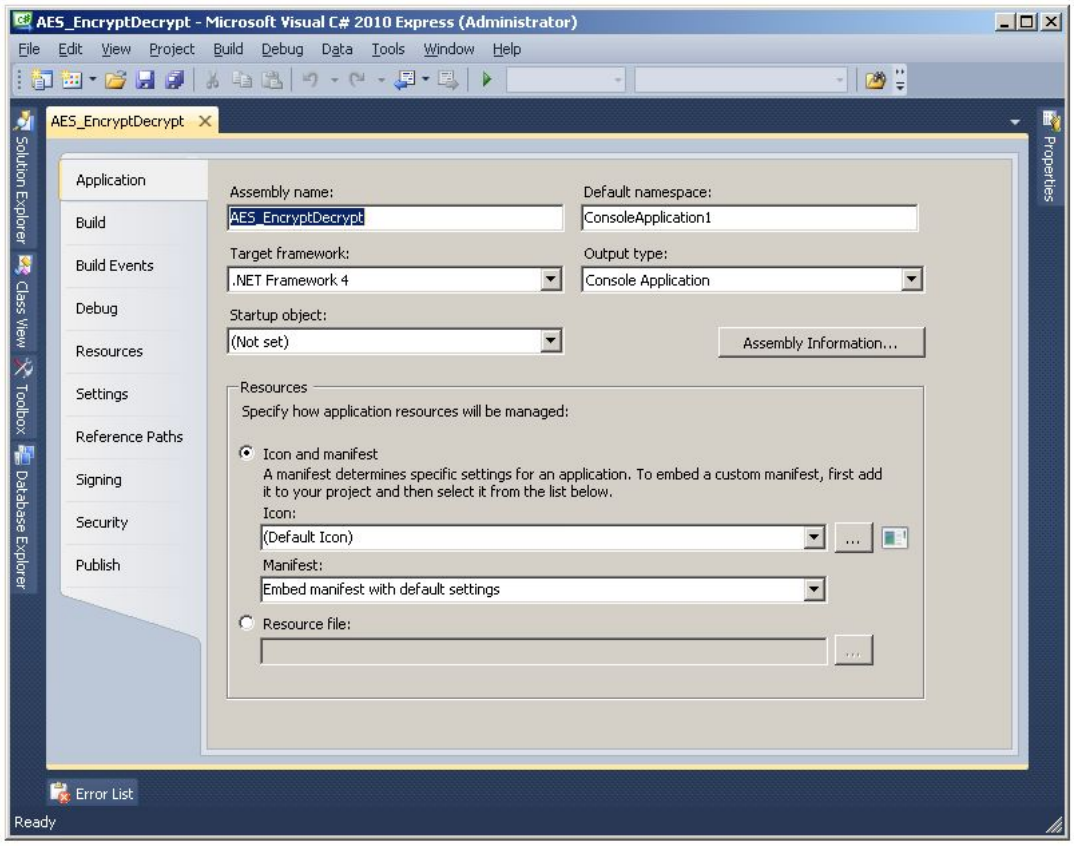
```
sealed class AES_EncryptDecryptLibrary
{
    public static string AES_Encrypt(string input, string pass)
    {
        try
        {
            return Convert.ToBase64String(EncryptStringToBytes(input, System.Text.Encoding.Default.GetBytes(pass)));
        }
        catch (Exception)
        {
            return "";
        }
    }
    public static string AES_Decrypt(string input, string pass)
    {
        try
        {
            return DecryptStringFromBytes(Convert.FromBase64String(input), System.Text.Encoding.Default.GetBytes(pass));
        }
        catch (Exception)
        {
            return "";
        }
    }
```

In order to create a CLR assembly suitable for deployment to SQL Server, we will recompile the console application to a class library.

Right-click on the Properties node and select the Open menu item, or simply double-click the Properties node.

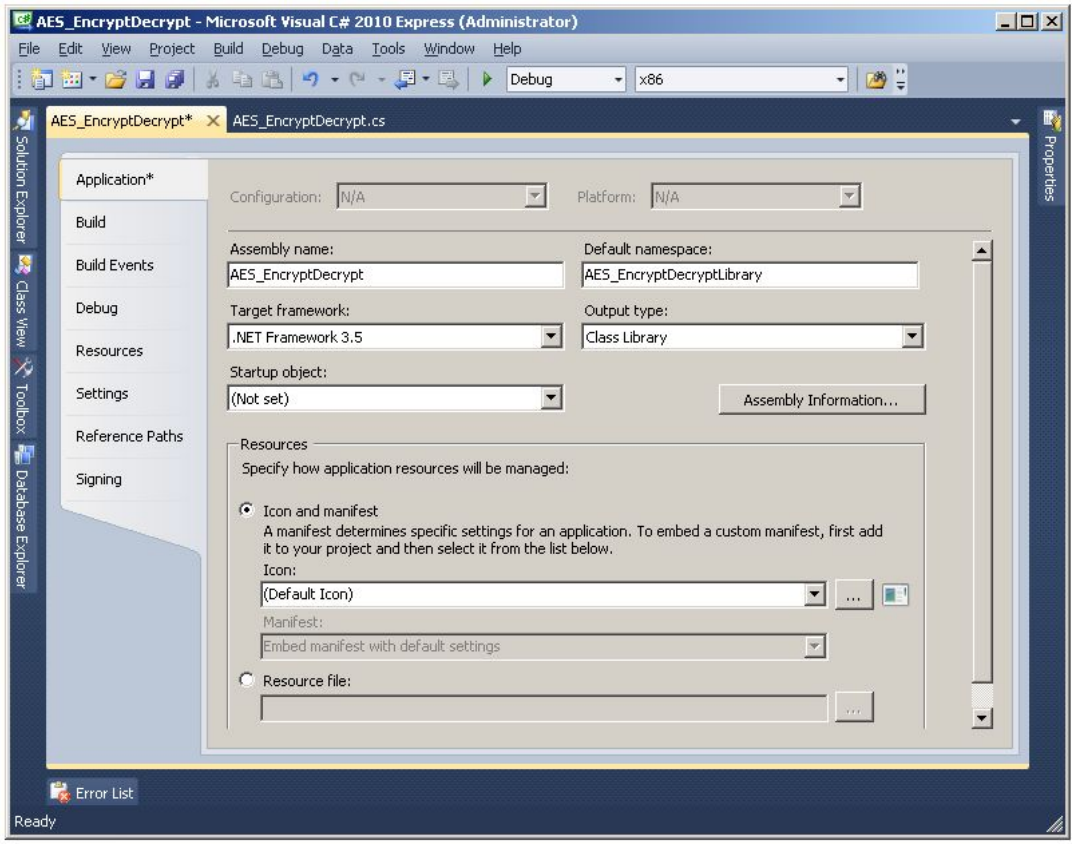


This is what the Properties-Application tab looks like by default.

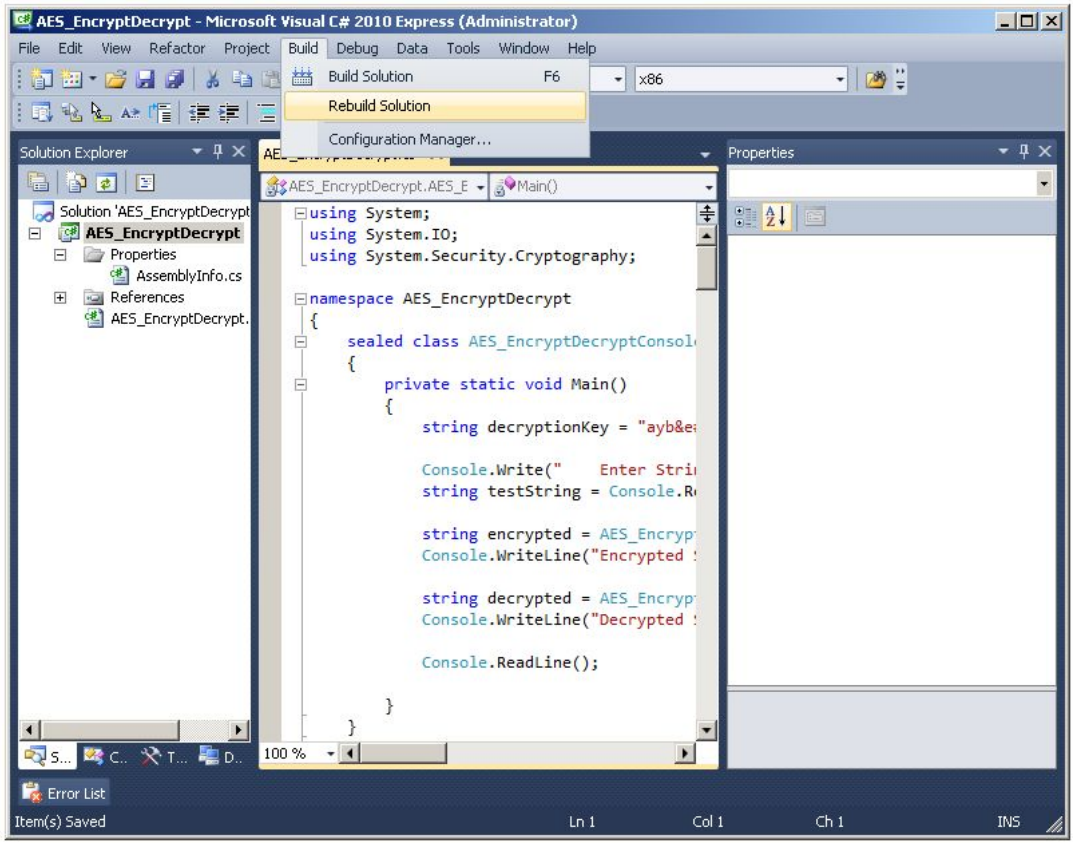


Change the target framework to .NET Framework 3.5, the default namespace to AES_EncryptDecryptLibrary and the output type to Class Library.

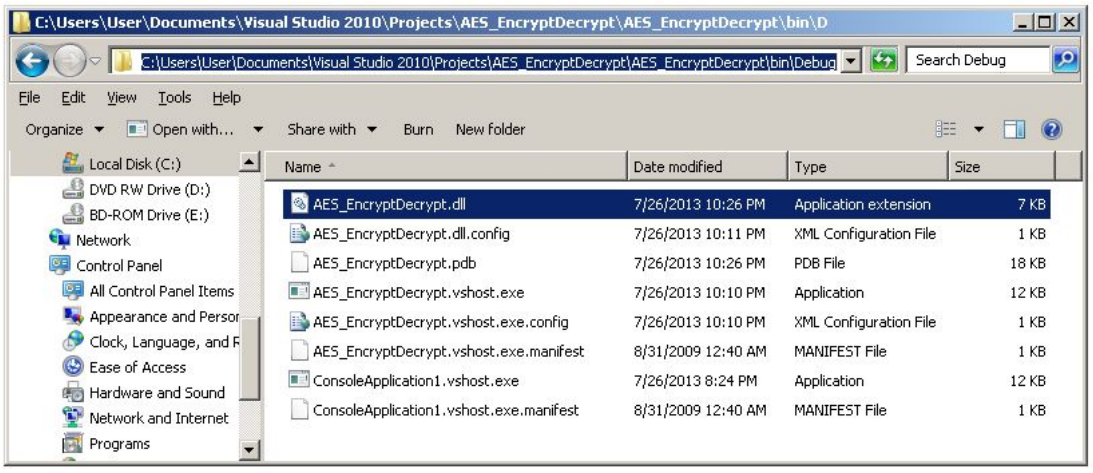
(The target framework does not need to be changed from .NET Framework 4 if you are using SQL Server 2012 or later.)



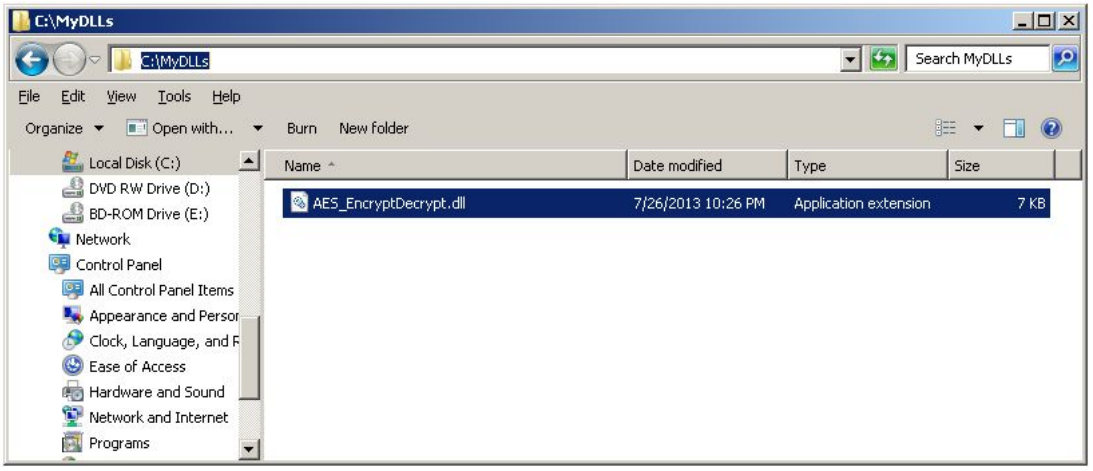
Rebuild the project by selecting Build-Rebuild Solution from the top menu bar.



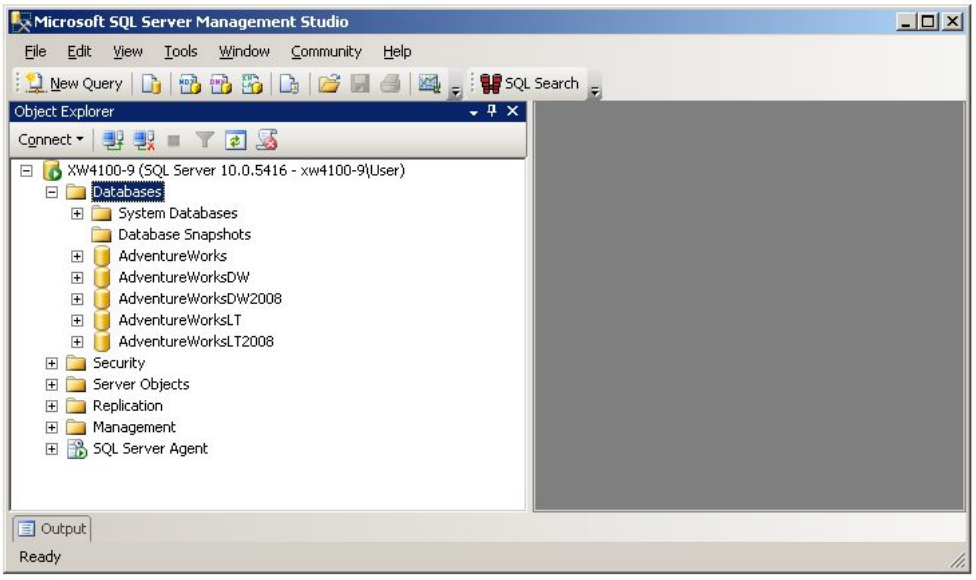
We have now built the CLR assembly AES_EncryptDecrypt.dll as shown in Windows Explorer.



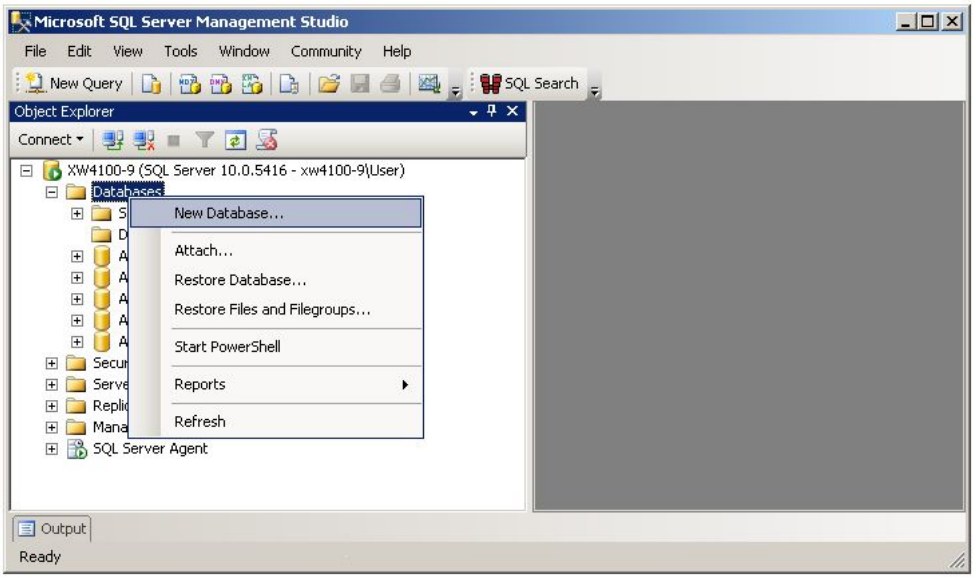
Create the directory C:\MyDLLs\ and copy AES_EncryptDecrypt.dll to it to make it easier to find when we deploy it to SQL Server.



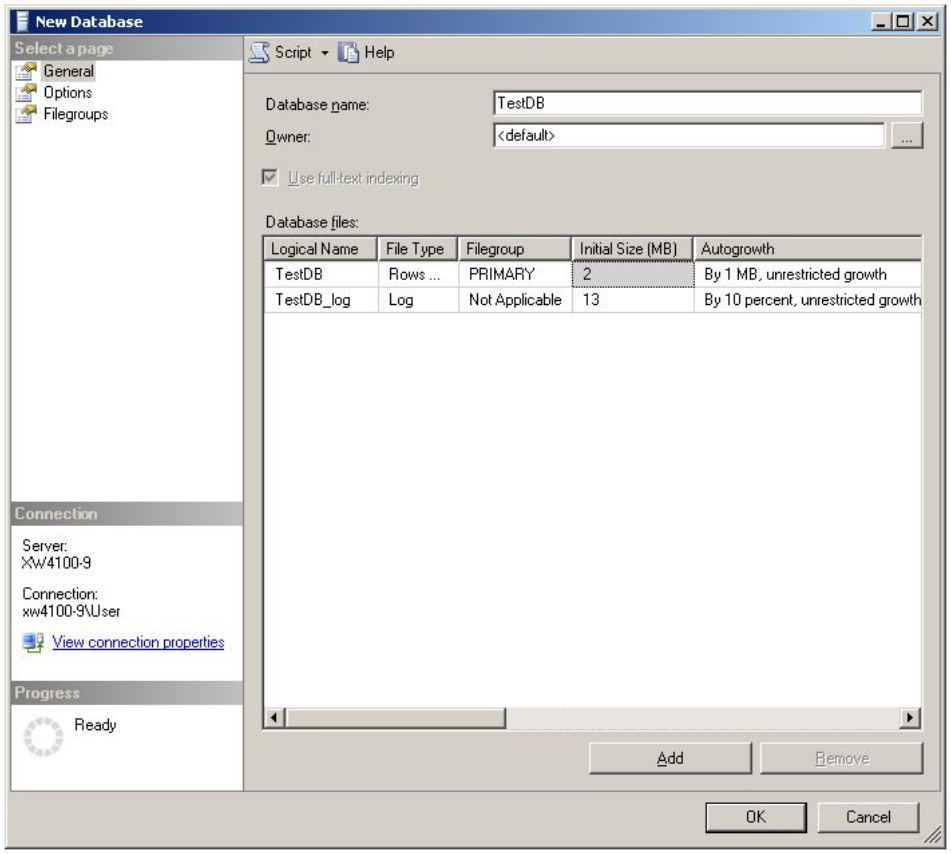
Now start SQL Server Managment Studio.



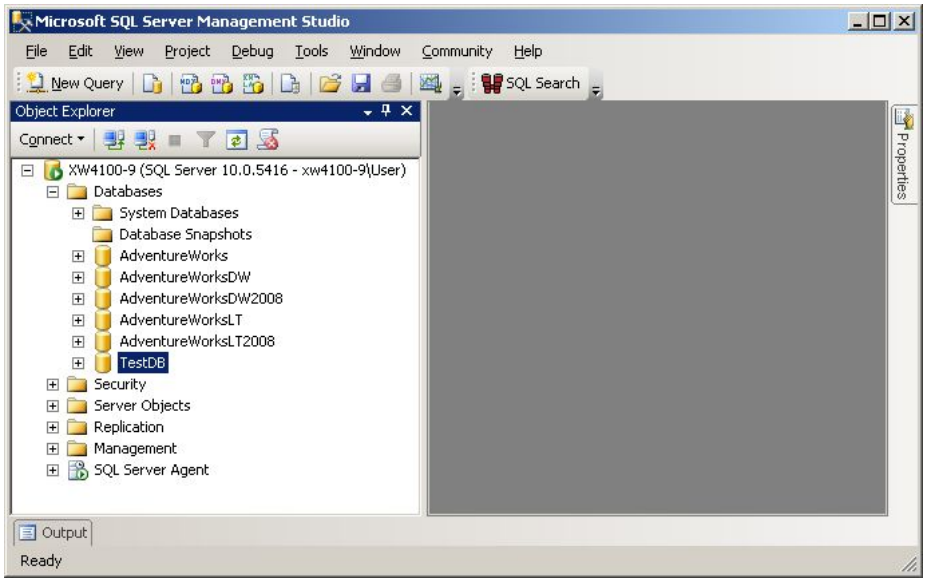
Create a new database...



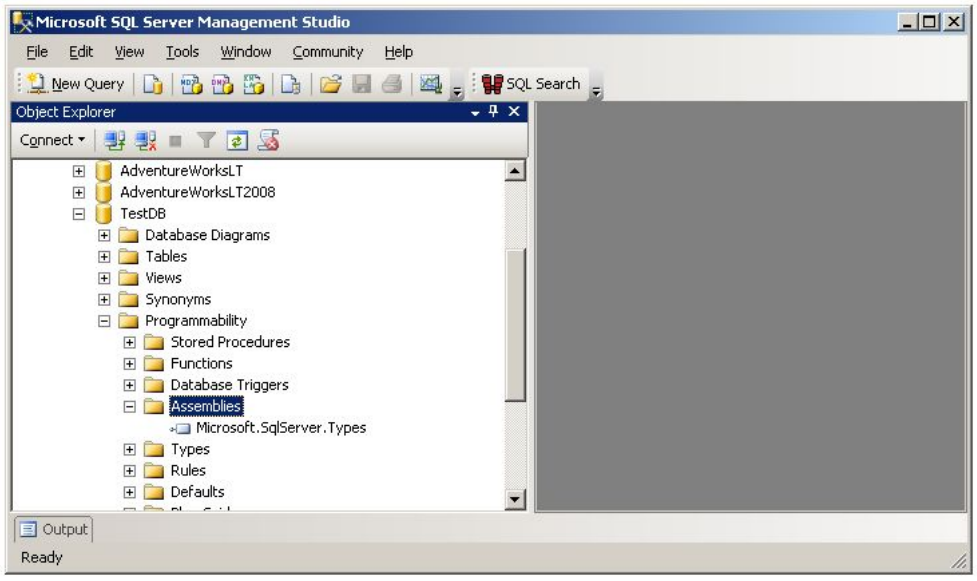
...named TestDB.



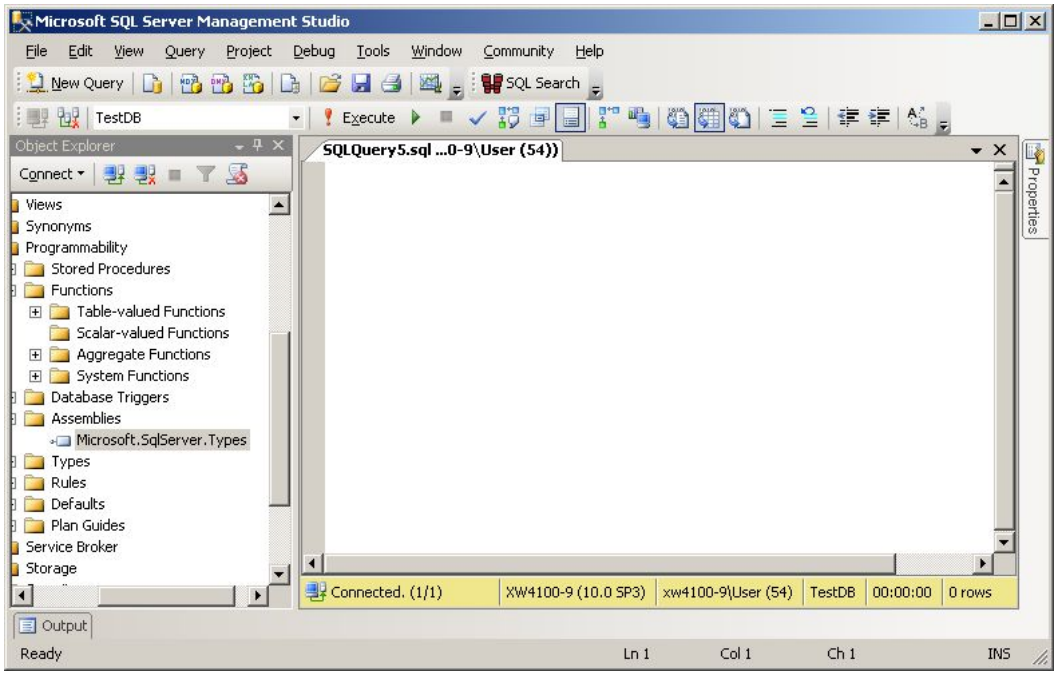
Expand the TestDB node...



Browse to the TestDB-Programmability-Assemblies node and note that the only assembly available by default is Microsoft.SqlServer.Types.



Click on the New Query menu item to create a query editor window.

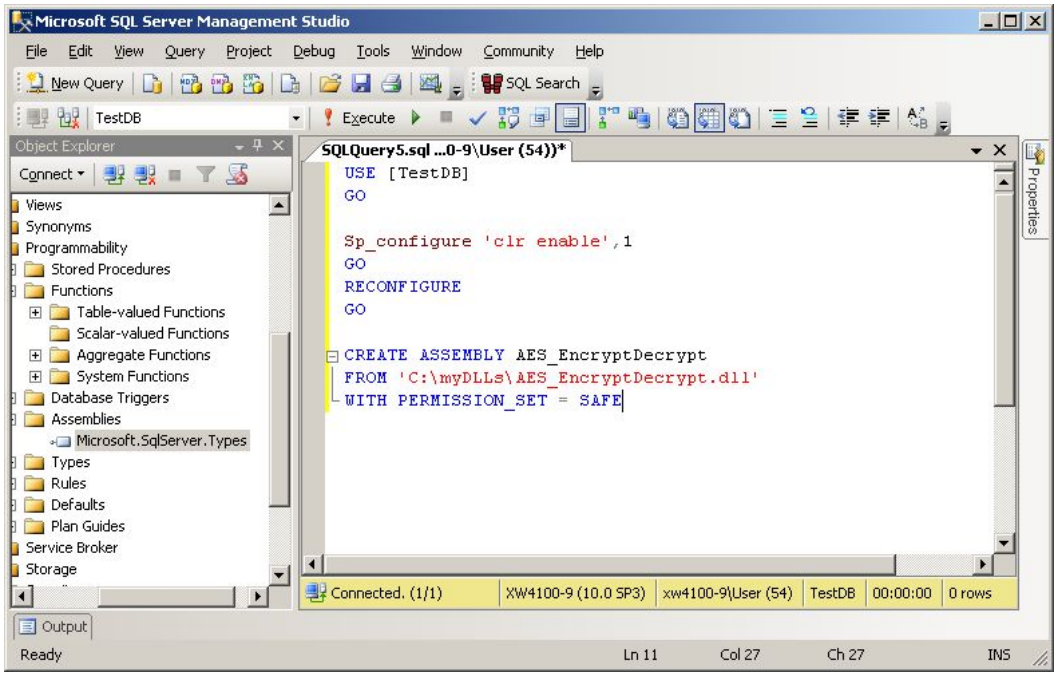


Paste the following code...

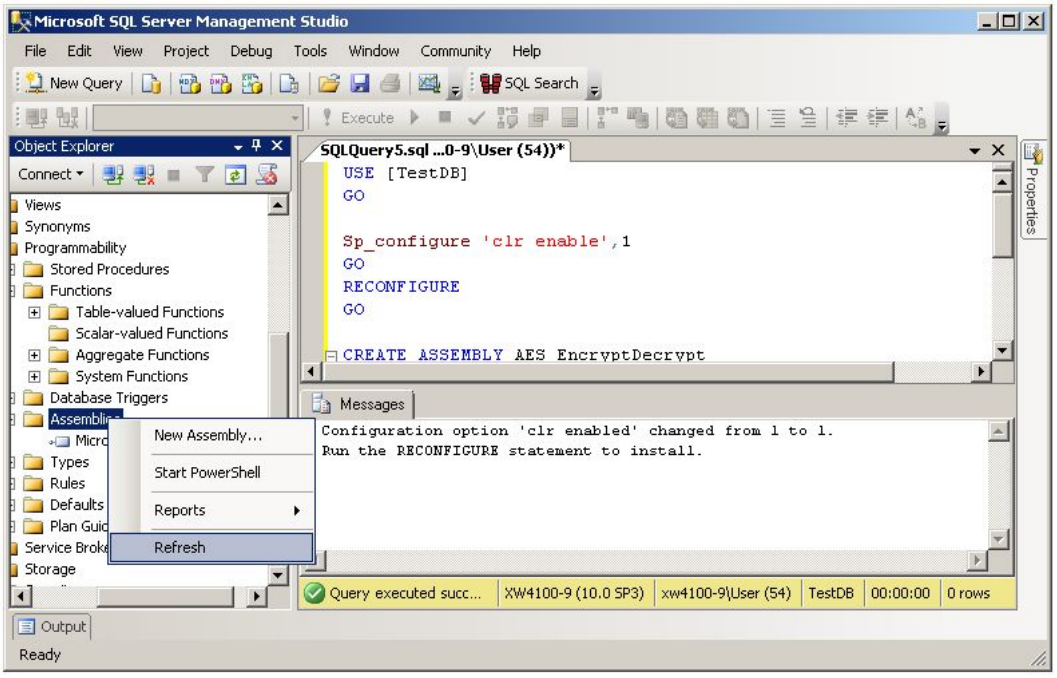
```
USE [TestDB]
GO
```

```
Sp_configure 'clr enable',1
GO
RECONFIGURE
GO

CREATE ASSEMBLY AES_EncryptDecrypt
FROM 'C:\myDLLs\AES_EncryptDecrypt.dll'
WITH PERMISSION_SET = SAFE
```

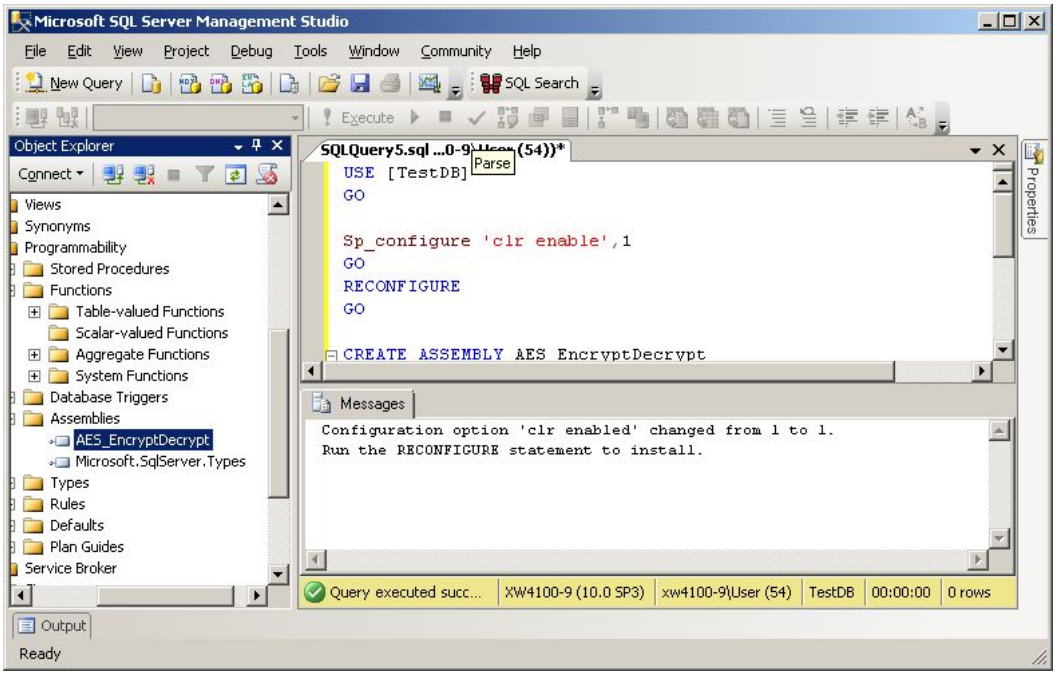
...into the query editor window.



Execute the T-SQL statements and refresh the Object Explorer.



Note that the AES_EncryptDecrypt assembly has been added to the Assemblies node.

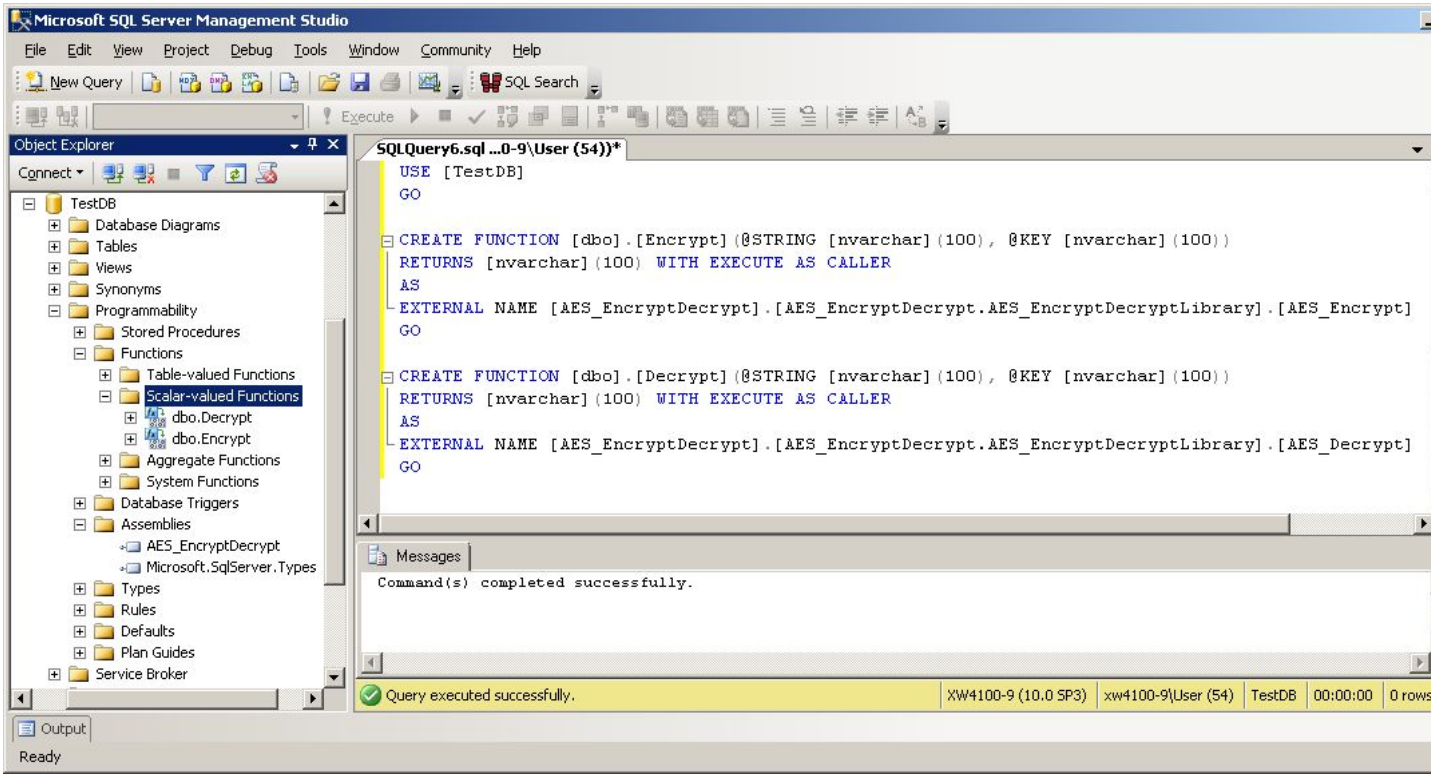


Paste the following T-SQL code...

```
USE [TestDB]
GO

CREATE FUNCTION [dbo].[Encrypt](@STRING [nvarchar](100), @KEY [nvarchar](100))
RETURNS [nvarchar](100) WITH EXECUTE AS CALLER
AS
EXTERNAL NAME [AES_EncryptDecrypt].[AES_EncryptDecrypt.AES_EncryptDecryptLibrary].[AES_Encrypt]
GO

CREATE FUNCTION [dbo].[Decrypt](@STRING [nvarchar](100), @KEY [nvarchar](100))
RETURNS [nvarchar](100) WITH EXECUTE AS CALLER
AS
EXTERNAL NAME [AES_EncryptDecrypt].[AES_EncryptDecrypt.AES_EncryptDecryptLibrary].[AES_Decrypt]
GO
```

...into a new query editor window and execute it, then expand the Scalar-valued Functions node to reveal the two new functions it created.

Inspect the code to confirm that both these functions reference public methods in the AES_EncryptDecryptLibrary class of the AES_EncryptDecrypt assembly.

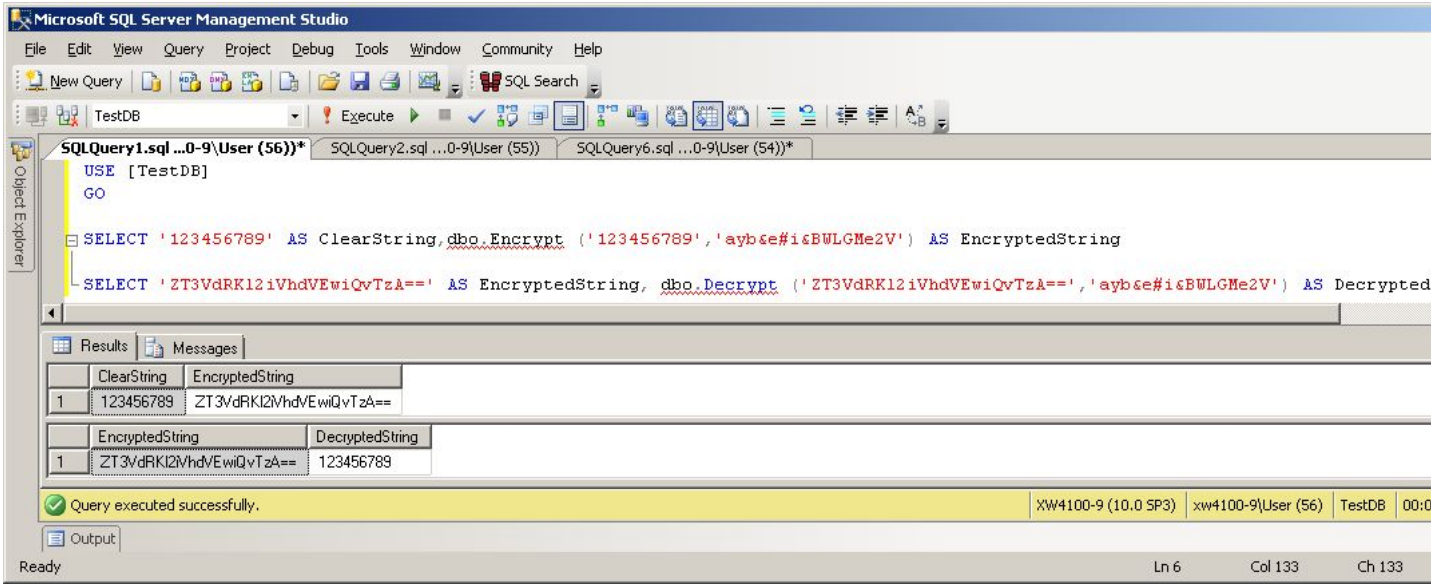


Now paste the following code...

```
USE [TestDB]
GO
SELECT '123456789' AS ClearString,dbo.Encrypt ('123456789','ayb&e#i&BWLGMe2V') AS EncryptedString
```

```
SELECT 'ZT3VdRKl2iVhdVEwiQvTzA==' AS EncryptedString, dbo.Decrypt ('ZT3VdRKl2iVhdVEwiQvTzA==','ayb&e#i&BWLGMe2V') AS DecryptedString
```

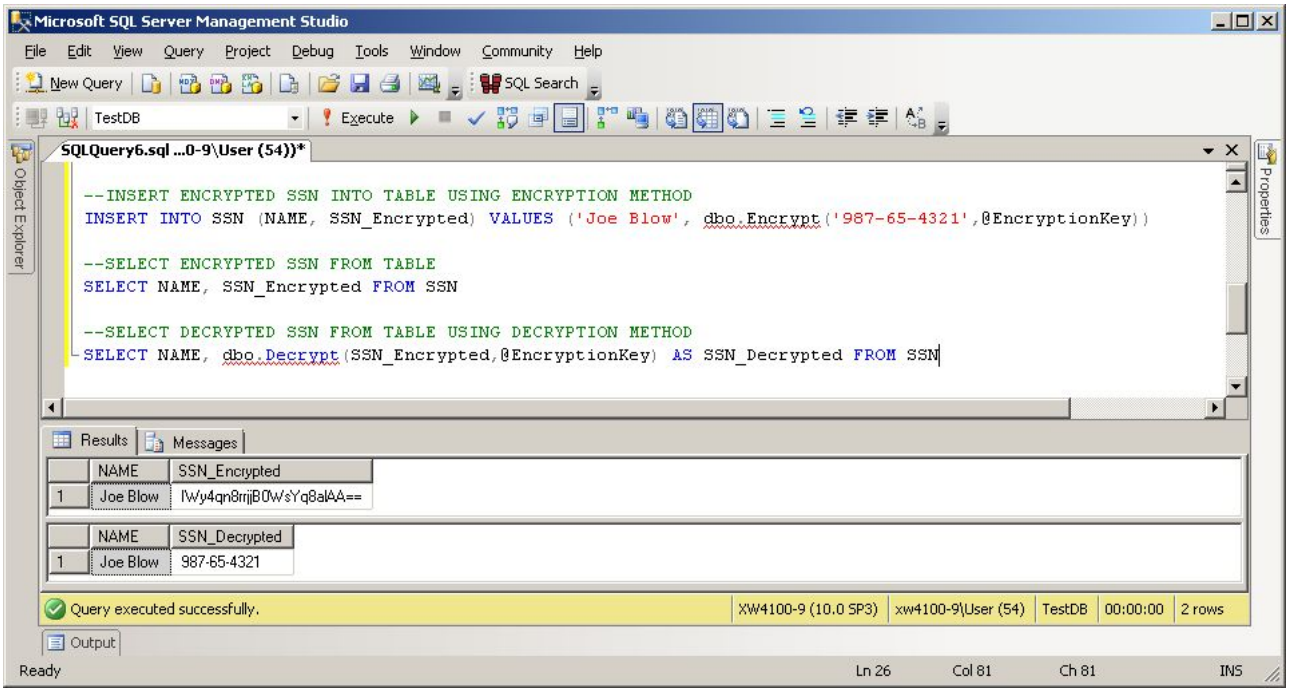
...into a new query editor window and execute it. As you can confirm by inspection, the code uses the two new functions to successfully encrypt and decrypt a string.



Now paste the following code...

```
USE TestDB
GO

DECLARE @EncryptionKey NVARCHAR(100)
SET @EncryptionKey = 'ayb&e#i&BWLGMe2V'

BEGIN TRY
    DROP TABLE SSN
END TRY
BEGIN CATCH
END CATCH

CREATE TABLE SSN
(
    NAME NVARCHAR(20),
    SSN_Encrypted NVARCHAR(100)
)

--INSERT ENCRYPTED SSN INTO TABLE USING ENCRYPTION METHOD
INSERT INTO SSN (NAME, SSN_Encrypted) VALUES ('Joe Blow', dbo.Encrypt('987-65-4321',@EncryptionKey))

--SELECT ENCRYPTED SSN FROM TABLE
SELECT NAME, SSN_Encrypted FROM SSN

--SELECT DECRYPTED SSN FROM TABLE USING DECRYPTION METHOD
SELECT NAME, dbo.Decrypt(SSN_Encrypted,@EncryptionKey) AS SSN_Decrypted FROM SSN
```

...into a new query editor window and execute it.



The above code creates a table named SSN, inserts a name and an encrypted SSN into the table, selects from the table to show that it contains an encrypted SSN, then selects the name and the decrypted SSN from the table.

Conclusion

This article has shown how to create a CLR assembly named AES_EncryptDecrypt using Microsoft Visual C# 2010 Express, how to deploy that assembly to SQL Server, how to create functions named Encrypt and Decrypt that reference corresponding methods in the CLR assembly, how to use the Encrypt function to insert encrypted data into a SQL Server table, and how to extract decrypted data from the table using the Decrypt function.
