MSSQLTips.com
brought to you by Edgewood Solutions

Your daily source for SQL Server Tips

Writing to an operating system file using the SQL Server SQLCLR

Written By: Andy Novick -- 1/9/2009

Problem

Reading files from the operating system can be done with T-SQL as I showed in the tip Using OPENROWSET to read large files into SQL Server. What if you want to write to an operating system file? For example, writing to a text file. There's no T-SQL that supports writing to a file.

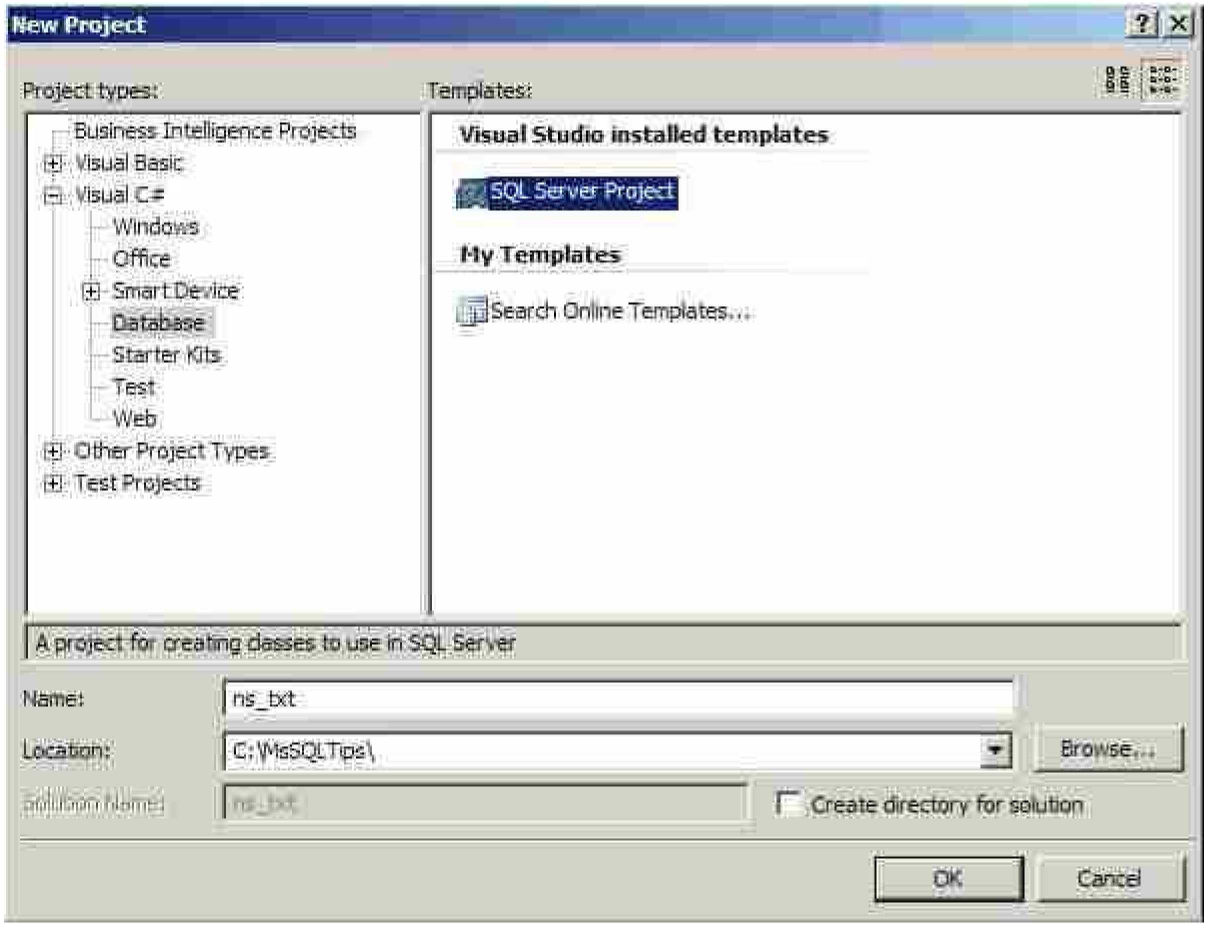
Solution

The solution is to create a stored procedure that is implemented in the SQLCLR, which allows writing code in .Net languages and running them within SQL Server. Stored procedures can be written in C#, VB.Net or C++ and the compiler produces an assembly, which is the code compiled into .Net Intermediate Language (IL). The assembly is then loaded into SQL Server and a stored procedure is defined to call one of the static methods in the assembly. When the stored procedure is invoked by a T-SQL EXECUTE statement the .Net assembly is loaded, Just-in-Time (JIT) compiled into machine code and the machine code is loaded into SQL Server's memory. SQLCLR code is similar to extended stored procedures and it is intended to replace extended stored procedures when that feature is phased out of SQL Server in a future release.

The easiest way to create SQLCLR objects is to use the Database project template in Visual Studio 2005 Professional Edition or Team Edition. Visual Studio 2008 with Service Pack 1 also supports database projects for SQL Server 2008. Before I walk through that process you're going to need a database. The example files use a connection to the database ns_lib, which you can create with this T-SQL:

```
CREATE DATABASE ns_lib
```

Once you have a database, you'll need to pick a programming language. This tip is written in C# but you can also use VB.Net or C++. Then in Visual Studio use the File/New/Project menu command and specify the project name and folder. Here I've named the project ns_txt:



A project for creating classes to use in SQL Server

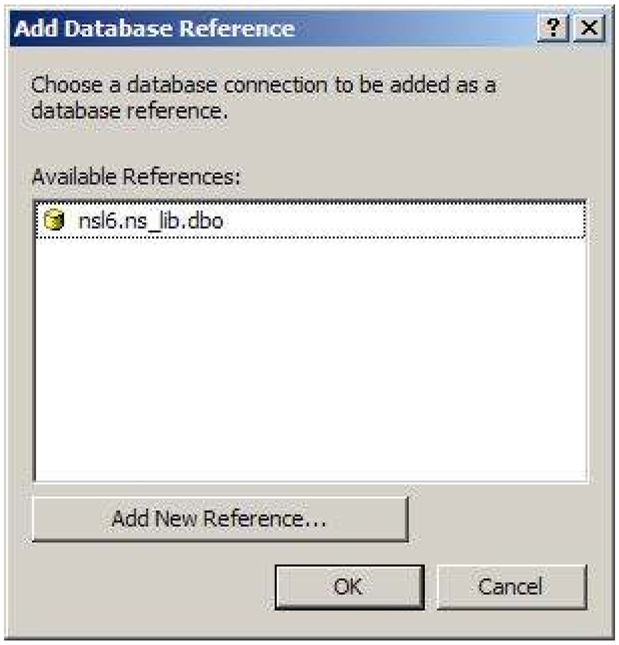
Name: ns_txt

Location: C:\MsSQLTips\

Solution Name: ns_txt

Create directory for solution

If you have existing database connections defined in Visual Studio, the "Add Database Reference" dialog box will open and you'll be given the chance to pick one of the Available References as depicted here:

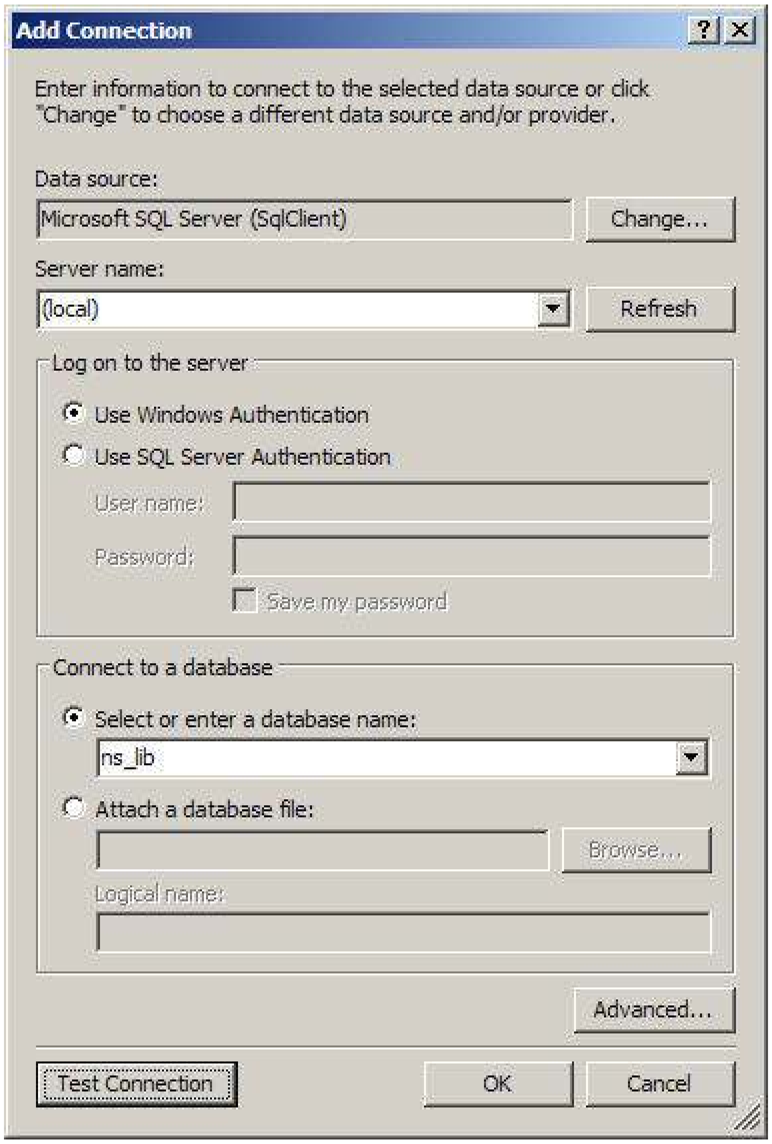


Choose a database connection to be added as a database reference.

Available References:

nsl6.ns_lib.dbo

You can pick a database reference or press the "Add New Reference..." button. If you don't have existing connections Visual Studio goes right to the "New Database Reference" screen and allows you to create a connection to the database where you want the SQLCLR assembly to be loaded. The following picture shows adding a connection to the ns_lib database on the local machine's default instance:



Add Connection

Enter information to connect to the selected data source or click "Change" to choose a different data source and/or provider.

Data source: Microsoft SQL Server (SqlClient)

Server name: (local)

Log on to the server

(●) Use Windows Authentication

( ) Use SQL Server Authentication

User name:

Password:

[ ] Save my password

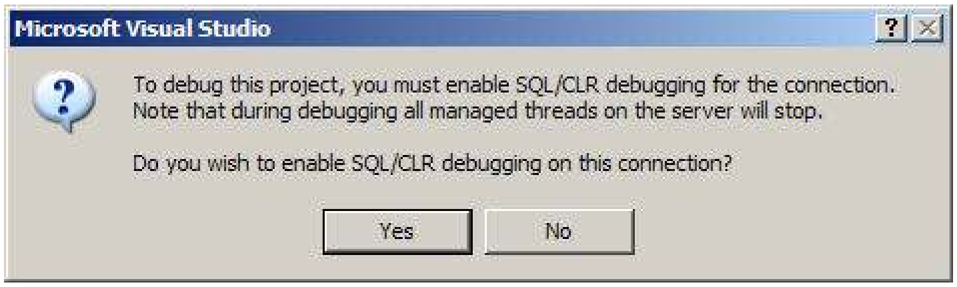
Connect to a database

(●) Select or enter a database name: ns_lib

( ) Attach a database file:

Logical name:

Once a connection is selected Visual Studio will ask to enable debugging with this dialog box:



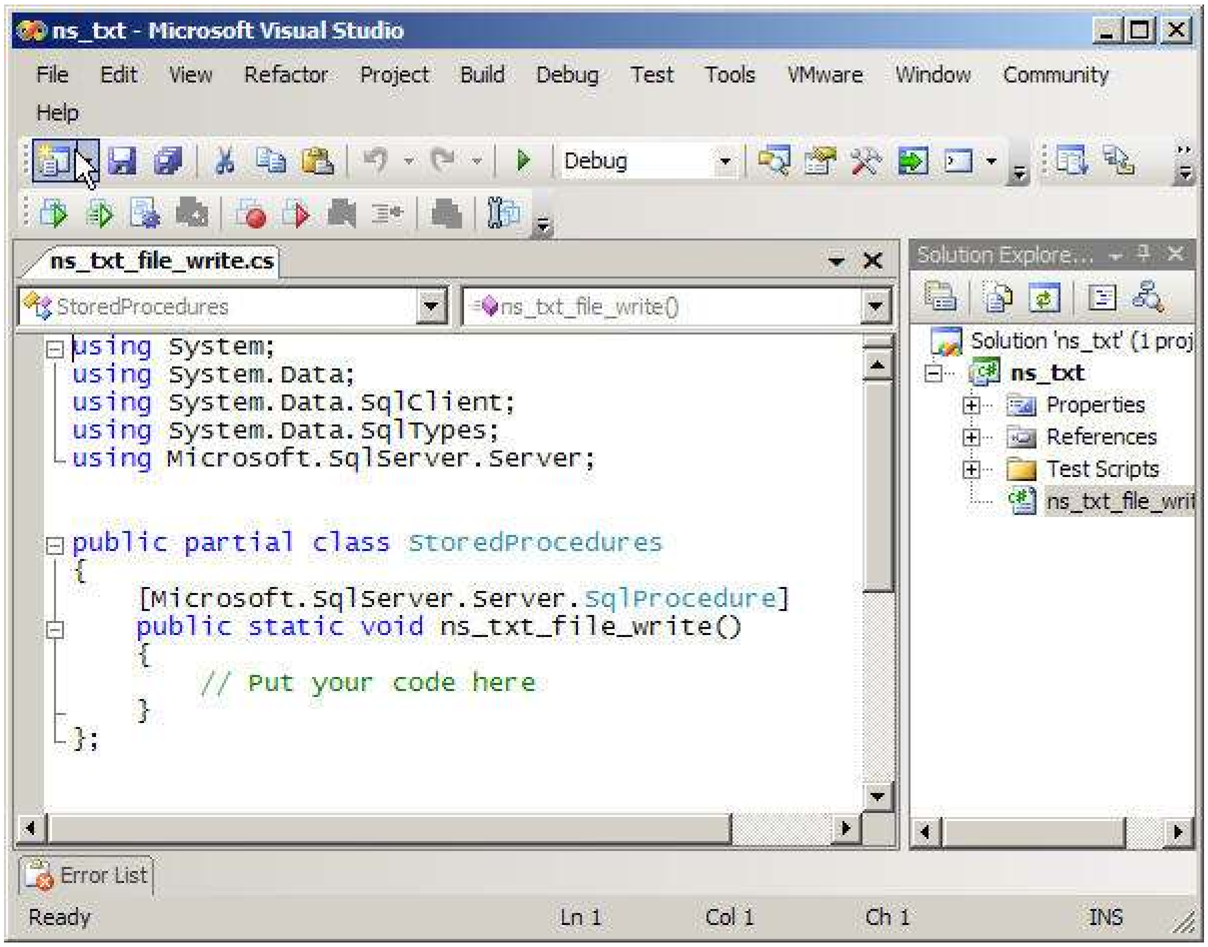
Microsoft Visual Studio

To debug this project, you must enable SQL/CLR debugging for the connection. Note that during debugging all managed threads on the server will stop.

Do you wish to enable SQL/CLR debugging on this connection?

The facility to debug the CLR code while it's being called from within SQL Server is a fantastic aid to productivity. However, debugging stops all managed threads, which is the SQLCLR code, from running while debugging. Therefore, I only debug CLR code running on my local development machine. Never debug SQLCLR code on a production server. You can learn more about debugging in this tip Debugging SQL Server CLR functions, triggers and stored procedures. Pick Yes to enable debugging, or No if you don't want it.

Visual Studio creates the project files and you can start work. The next step is to add the stored procedure to the project. Do this by selecting the ns_txt project in the Solution Explorer window and using the menu command Project/Add Stored Procedure. The Add New Item dialog box pops up and you can name the file for your stored procedure code: Here's the dialog with the name that I've chosen, ns_txt_file_write.

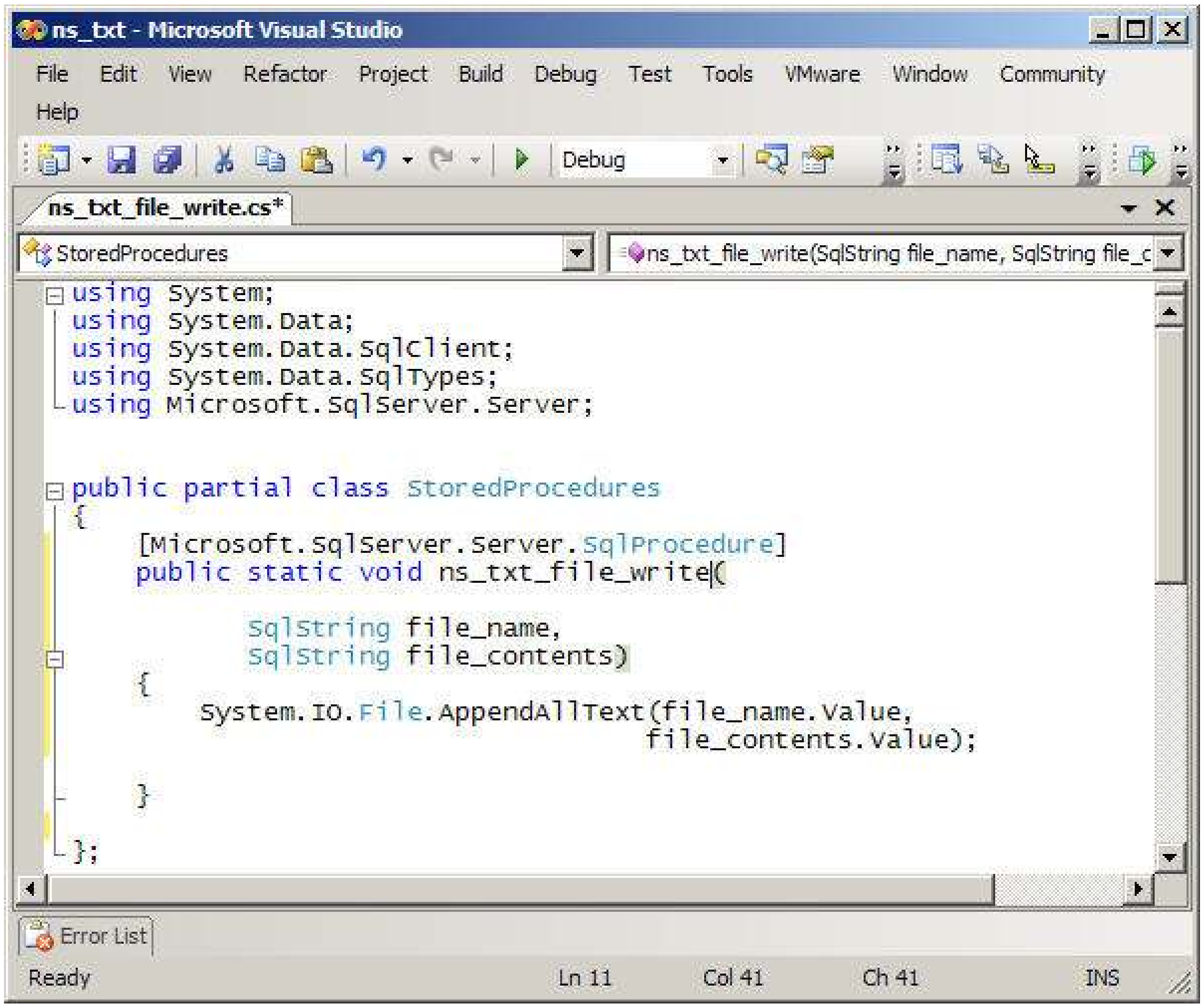

ns_txt - Microsoft Visual Studio
File Edit View Refactor Project Build Debug Test Tools VMware Window Community Help
Debug
ns_txt_file_write.cs
StoredProcedures
ns_txt_file_write()
Solution Explore...
Solution 'ns_txt' (1 proj
ns_txt
Properties
References
Test Scripts
ns_txt_file_wri
Error List
Ready Ln 1 Col 1 Ch 1 INS

```
using System;
using System.Data;
using System.Data.SqlClient;
using System.Data.SqlTypes;
using Microsoft.SqlServer.Server;


public partial class StoredProcedures
{
    [Microsoft.SqlServer.Server.SqlProcedure]
    public static void ns_txt_file_write()
    {
        // Put your code here
    }
};
```

Visual Studio creates the file with a partial class named StoredProcedures. You can change that name, if you like. It also creates a "public static void" method for your procedure with the name you gave it and no parameters. Above the procedure declaration is the line "[Microsoft.SqlServer.Server.SqlProcedure]". This line is an attribute that tells Visual Studio how to deploy the procedure into SQL Server.

Now that the project and code file are created, it's time to write the code that implements the stored procedure. ns_txt_file_write has two parameters, file_name and file_contents. Both are strings declared with the SqlString type. The Microsoft.Data.SqlTypes namespace, which is referenced in the third using statement, provides types that allow SQL Server to convey the data precisely and include properties such as IsNull, to communicate that a value is null as well as conversion functions. While it's possible to use CLR types, such as string, for the parameters, I prefer to stick to the SqlTypes to avoid conversion overhead. However, the SqlTypes don't match up with that the CLR expects for strings or integers so in the C# it's necessary to refer to the Value property of the parameter in order to use the variable.

Here's the function all done:

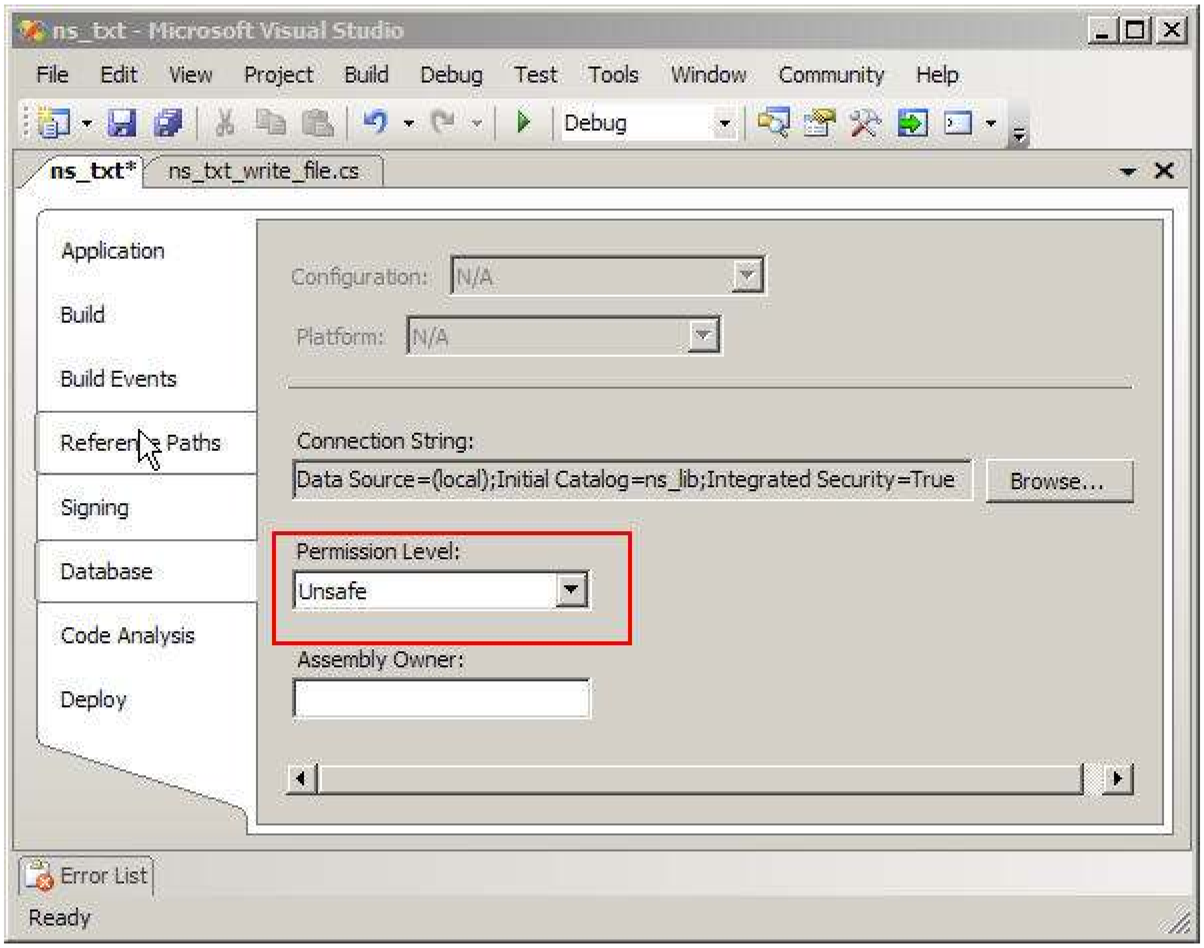

ns_txt - Microsoft Visual Studio
File Edit View Refactor Project Build Debug Test Tools VMware Window Community Help
Debug
ns_txt_file_write.cs*
StoredProcedures
ns_txt_file_write(SqlString file_name, SqlString file_c
Error List
Ready Ln 11 Col 41 Ch 41 INS

```
using System;
using System.Data;
using System.Data.SqlClient;
using System.Data.SqlTypes;
using Microsoft.SqlServer.Server;


public partial class StoredProcedures
{
    [Microsoft.SqlServer.Server.SqlProcedure]
    public static void ns_txt_file_write(

            SqlString file_name,
            SqlString file_contents)
    {
        System.IO.File.AppendAllText(file_name.Value,
                                 file_contents.Value);

    }

};
```

That's it? All this activity for just one line? It's kind of disappointing but it illustrates the power of the SQLCLR. The CLR has the built-in method AppendAllText that takes care of the job and all that ns_txt_file_write has to do is expose it to SQL Server.

There's one more step required before testing the procedure. Because ns_txt_file_write reaches outside of SQL Server into the file system, the permission level given to this code must be raised. By default the permission level of Visual Studio Database projects is Safe, which doesn't allow access outside of SQL Server. The permission level External lets the code access the file system and other external resources, such as Active Directory or web services. There is a third permission level, Unsafe, which allows the use of unmanaged code and calls to Windows API functions. While it's called Unsafe, it's no worse then the permissions given to external stored procedures.

The permission level is set on the Database tab of the projects properties. The following screen shot shows the Permission Level highlighted in red:



Configuration: N/A
Platform: N/A
Connection String: Data Source=(local);Initial Catalog=ns_lib;Integrated Security=True
Permission Level: Unsafe
Assembly Owner:

Now that everything is ready the project can be compiled and loaded into SQL Server using the menu command Build/Deploy Solution.... That almost works. Instead of deploying to SQL Server I got this message:

```
&5($7( ▮$66(0%/< ▮IRU▮DVVHPEO\▮QVBW[W▮▮IDLOHG▮EHFDXVH▮DVVHPEO\▮QVBW[W▮▮LV▮QRW
DXWKRUL]HG▮IRU▮3(50,66,21B6(7▮ ▮(;7(51$/B$&&(66▮▮7KH▮DVVHPEO\▮LV▮DXWKRUL]HG
ZKHQ▮HLWKHU▮RI▮WKH▮IROORZLQJ▮LV▮WUXH▮▮WKH▮GDWDEDVH▮RZQHU▮▮'%2▮▮KDV▮(;7(51$/▮$&&(66
$66(0%/<▮SHUPLVVLRQ▮DQG▮WKH▮GDWDEDVH▮KDV▮WKH▮75867:257+<▮GDWDEDVH▮SURSHUW\▮RQ▮
RU▮WKH▮DVVHPEO\▮LV▮VLJQHG▮ZLWK▮D▮FHUWLILFDWH▮RU▮DQ▮DV\PPHWULF▮NH\▮WKDW▮KDV▮D▮FRUUHVSRQGLQJ
ORJLQ▮ZLWK▮(;7(51$/▮$&&(66▮$66(0%/<▮SHUPLVVLRQ▮▮,I▮\RX▮KDYH▮UHVWRUHG▮RU▮DWWDFKHG
WKLV▮GDWDEDVH▮▮PDNH▮VXUH▮WKH▮GDWDEDVH▮RZQHU▮LV▮PDSSHG▮WR▮WKH▮FRUUHFW▮ORJLQ▮RQ▮WKLV▮VHUYHU▮
,I▮QRW▮▮XVH▮VSBFKDQJHGERZQHU▮WR▮ÀL[▮WKH▮SUREOHP▮▮
▮
```

As the message explains, SQLCLR code with the EXTERNAL_ACCESS permission set must either be signed or the database must be given the TRUSTWORTHY attribute. Code signing is a subject for another tip so here let's use the easier alternative of making the database TRUSTWORTHY with this ALTER DATABASE statement:

```
ALTER DATABASE ns_lib SET TRUSTWORTHY ON
```

Another problem that you may run into is database ownership. As the message indicates the owner of the database must have EXTERNAL ACCESS ASSEMBLY permission for assemblies that require external access to work. I prefer to have databases owned by sa so if your database isn't, it can be changed with this exec statement:

```
EXEC sp_changedbowner sa
```

My next try with the menu command Build/Deploy Solution was successful. Behind the scenes Visual Studio is telling SQL Server to create an assembly from the compiled code and to create the stored procedures with these SQL Statements:

```
CREATE ASSEMBLY [ns_txt]
FROM 0x4D5A90000300000004000000FFFF0000B800.... more binary omitted.
                WITH PERMISSION_SET = EXTERNAL_ACCESS



CREATE PROCEDURE [ns_txt_file_write]
@file_name nvarchar(4000),
@file_contents nvarchar(4000)
AS
EXTERNAL NAME [ns_txt].[StoredProcedures].[ns_txt_file_write]
```

Using a Visual Studio Database project makes creating SQLCLR procedures easy but you could use a text editor to create the cs file and handle the compiling, assembly loading, and stored procedure creation yourself. To read more about how to do that see the tip CLR function to delete older backup and log files in SQL Server, which shows the individual steps.

Once these security issues are managed and the deployment is successful, it's time to test out the stored procedure. From an SSMS query window it is now possible to execute the procedure. You may have to modify the @file_name parameter of the following query to point to a directory that exists on your server. Once it's set go ahead and execute it:

```
exec ns_lib.dbo.ns_txt_file_write
          @file_name    ='c:\temp\test_file.txt'
        , @file_contents='Now that''s what I call a message!'
```

There are additional details, such as error handling, that can be handled more robustly but this simple example illustrates how to create a stored procedure that takes advantage of the power of the CLR from within SQL Server. The procedure is also limited to working with @file_contents of only 4000 characters because of the data type given to the stored procedure by Visual Studio.

The SQLCLR was introduced in SQL Server 2005 and has been enhanced for SQL Server 2008. In earlier versions similar functionality can be achieved by writing extended stored procedures or in SQL Server 2000 using the sp_OA_* extended stored procedures to invoke COM objects running outside of SQL Server. Those solutions are more dangerous to SQL Server and in the case of Ole Automation much slower. The SQLCLR is a significant improvement in the capability of SQL Server and it can be used to create database objects other than stored procedures such as functions, triggers, user-defined aggregates, user-defined types and triggers.

Next Steps

- Download and use the ns_txt_file_write stored procedure in your own code to write to a text file
- Create your own stored procedures in C# or VB.Net and incorporate useful .Net code
- Review related MSSQLTips on the SQLCLR such as these:
    - CLR function to delete older backup and log files in SQL Server
    - CLR String Sort Function in SQL Server 2005