Beginners guide to accessing SQL Server through C#

By **Matt Newman**, 22 Aug 2004

Introduction

In this article I plan to demonstrate how to insert and read data from a SQL Server or MSDE database. This code should work on both SQL Server , I am using 2000, and MSDE. I am using Visual Studio 2002, but this should work with Visual Studio 2003, Web Matrix, and the command line SDK. This code should work with both C# applications and C# web applications and webservices. This code does not compile on the FreeBSD with Rotor [^].

Background

Part of my current project required me too store and retrieve information from a database. I decided to use C# as my target language since I am currently reading Inside C# Second Edition [^] by Tom Archer [^], which by the way is a must have book. However I could not find any examples that were clear and just generic accesing SQL Server with C#.

Using the code

I did not include a sample application because the code provide within the article can really be dropped in and should work with no problem. Also through out the article I will refer to SQL Server, MSDE is a free version of SQL Server that does not have some of the GUI tools and has a few other limits such as database size. This code will work on both without problem.

Making the Love Connection

There is no real voodoo magic to creating a connection to a SQL Server assuming it is properly setup, which I am not going to go into in this article, in fact .NET has made working with SQL quite easy. First step is add the SQL Client namespace:

Collapse | Copy Code

```
using System.Data.SqlClient;
```

Then we create a `SqlConnection` and specifying the connection string.

Collapse | Copy Code

```
SqlConnection myConnection = new SqlConnection("user id=username;" +
                                    "password=password;server=serverurl;" +
                                    "Trusted_Connection=yes;" +
                                    "database=database; " +
                                    "connection timeout=30");
```

Note: line break in connection string is for formatting purposes only

SqlConnection.ConnectionString

The connection string is simply a compilation of options and values to specify how and what to connect to. Upon investigating the Visual Studio .NET help files I discovered that several fields had multiple names that worked the same, like `Password` and `Pwd` work interchangeably. I have not included all of the options for `SqlConnection.ConnectionString` at this time. As I get a chance to test and use these other options I will include them in the article.

User ID

About Article

A beginners guide to accessing a SQL or MSDE Server with C#

Type	Article
Licence	CPOL
First Posted	26 Jun 2003
Views	2,013,121
Bookmarked	345 times

.NET1.0 .NET1.1 C# Windows Visual-Studio ,

Print Email

Top News

News on the future of C# as a language

Get the Insider News free each

The `User ID` is used when you are using SQL Authentication. In my experience this is ignored when using a Trusted_Connection, or Windows Authentication. If the username is associated with a password `Password` or `Pwd` will be used.

```
"user id=userid;"
```

Password or Pwd

The password field is to be used with the User ID, it just wouldn't make sense to log in without a username, just a password. Both `Password` and `Pwd` are completely interchangeable.

```
"Password=validpassword;"-or-
"Pwd=validpassword;"
```

Data Source or Server or Address or Addr or Network Address

Upon looking in the MSDN documentation I found that there are several ways to specify the network address. The documentation mentions no differences between them and they appear to be interchangeable. The address is an valid network address, for brevity I am only using the `localhost` address in the examples.

```
"Data Source=localhost;"
-or-
"Server=localhost;"
-or-
"Address=localhost;"-or-"Addr=localhost;"
-or-"Network Address=localhost;"
```

Integrated Sercurity or Trusted_Connection

`Integrated Security` and `Trusted_Connection` are used to specify wheter the connnection is secure, such as Windows Authentication or SSPI. The recognized values are `true`, `false`, and `sspi`. According to the MSDN documentation `sspi` is equivalent to `true`. **Note:** I do not know how `SSPI` works, or affects the connection.

Connect Timeout or Connection Timeout

These specify the time, in seconds, to wait for the server to respond before generating an error. The default value is `15` (seconds).

```
"Connect Timeout=10;"-or-
"Connection Timeout=10;"
```

Initial Catalog or Database

`Initial Catalog` and `Database` are simply two ways of selecting the database associated with the connection.

```
"Inital Catalog=main;"
-or-
"Database=main;"
```

Network Library or Net

The Network Library option is essential if your are communicating with the server on a protocl other than TCP/IP. The default value for `Network Library` is `dbmssocn`, or TCP/IP. The following options are available: `dbnmpntw` (Named Pipes), `dbmsrpcn` (Multiprotocol), `dbmsadsn` (Apple Talk), `dbmsgnet` (VIA), `dbmsipcn` (Shared Memory), and `dbmsspxn` (IPX/SPX), and `dbmssocn` (TCP/IP). And as before `Network Library` and `Net` can be user interchangably. **Note:** The corresponding network protocol **must** be installed on the system to which you connect.

SqlConnection.Open()

This is the last part of getting connected and is simply executed by the following (remember to make sure your connection has a connection string first):

Collapse | Copy Code

```
try
{
    myConnection.Open();
}
catch(Exception e)
{
    Console.WriteLine(e.ToString());
}
```

`SqlConnection.Open()` is a void function and does not return an error but throws an exception so remember to put

it in a try/catch brace. rather than having the program explode in front of the user.

Command thee

SQL commands are probably the most difficult part of using an SQL database, but the .NET framework has wrapped up everything up nicely and takes most of the guess work out.

SqlCommand

Any guesses on what `SqlCommand` is used for? If you guessed for SQL commands then you are right on. An `SqlCommand` needs at least two things to operate. A command string, and a connection. First we'll look at the connection requirement. There are two ways to specify the connection, both are illustrated below:

Collapse | Copy Code

```
SqlCommand  myCommand = new SqlCommand("Command String", myConnection);

// - or -

myCommand.Connection = myConnection;
```

The connection string can also be specified both ways using the `SqlCommand.CommandText` property. Now lets look at our first `SqlCommand`. To keep it simple it will be a simple `INSERT` command.

Collapse | Copy Code

```
SqlCommand myCommand= new SqlCommand("INSERT INTO table (Column1, Column2) " +
                                     "Values ('string', 1)", myConnection);

// - or -

myCommand.CommandText = "INSERT INTO table (Column1, Column2) " +
                        "Values ('string', 1)";
```

Now we will take a look at the values . `table` is simply the table within the database. `Column1` and `Column2` are merely the names of the columns. Within the values section I demonstrated how to insert a `string` type and an `int` type value. The string value is placed in single quotes and as you can see an integer is just passed as is. The final step is to execute the command with:

Collapse | Copy Code

```
myCommand.ExecuteNonQuery();
```

SqlDataReader

Inserting data is good, but getting the data out is just as important. Thats when the `SqlDataReader` comes to the rescue. Not only do you need a data reader but you need a `SqlCommand`. The following code demonstrates how to set up and execute a simple reader:

Collapse | Copy Code

```
try
{
    SqlDataReader myReader = null;
    SqlCommand    myCommand = new SqlCommand("select * from table",
                                             myConnection);
    myReader = myCommand.ExecuteReader();
    while(myReader.Read())
    {
        Console.WriteLine(myReader["Column1"].ToString());
        Console.WriteLine(myReader["Column2"].ToString());
    }
}
catch (Exception e)
{
    Console.WriteLine(e.ToString());
}
```

As you can see the `SqlDataReader` does not access the database, it merely holds the data and provides an easy interface to use the data. The `SqlCommand` is fairly simple, table is the table your are going to read from. `Column1` and `Column2` are just the columns as in the table. Since there is a very high probability your will be reading more than one line a `while` loop is required to retrieve all of the records. And like always you want to `try` it and `catch` it so you don't break it.

SqlParameter

There is a small problem with using SqlCommand as I have demonstrated, it leaves a large security hole. For example, with the way previously demonstrated your command string would be constructed something like this if you were to get input from a user:

Collapse | Copy Code

```
SqlCommand myCommand = new SqlCommand(
 "SELECT * FROM table WHERE Column = " + input.Text, myConnection);
```

Its all fine and dandy if the user puts in correct syntax, however, what happens if the user puts `value1, DROP table`. Best case scenario it will cause an exception (I haven't checked to see what this example will do but it demonstrates a point), worst case you can kiss your table goodbye. You could parse all user input and strip out anything that could cause problems OR you could use an `SqlParameter`. Now the `SqlParameter` class is pretty big, but I will just show you a basic parameter usage. Basically you need three things to create a parameter. A name, data type, and size. (note for some data types you will want to leave off the size, such as `Text`).

Collapse | Copy Code

```
SqlParameter myParam = new SqlParameter("@Param1", SqlDbType.VarChar, 11);
myParam.Value = "Garden Hose";

SqlParameter myParam2 = new SqlParameter("@Param2", SqlDbType.Int, 4);
myParam2.Value = 42;

SqlParameter myParam3 = new SqlParameter("@Param3", SqlDbType.Text);
myParam.Value = "Note that I am not specifying size. " +
  "If I did that it would trunicate the text.";
```

It is naming convention, it might be required I'm not sure, to name all parameters starting with the `@` symbol. Now how do you use a parameter? Will its pretty easy as the following code shows.

Collapse | Copy Code

```
SqlCommand myCommand = new SqlCommand(
  "SELECT * FROM table WHERE Column = @Param2", myConnection);
myCommand.Parameters.Add(myParam2);
```

Now this keeps a rogue user from high-jacking your command string. This isn't all there is to parameters if you want to learn more advanced topics a good place to start is here[^].

Don't forget to close up when your done!

Closing a connection is just as easy as opening it. Just call`SqlConnection.Close()` but remember to put it in try/catch because like `SqlConnection.Open()` it does not return errors but throws an exception instead.

Collapse | Copy Code

```
try
{
    myConnection.Close();
}
catch(Exception e)
{
    Console.WriteLine(e.ToString());
}
```

When good connections go bad

The trusted connection had always been a mystery to me, I had never figured why IIS and SQL server never seemed to get along. Fortunately Pete (moredip) pointed out a helpful section of the documentation. To make it more simple I have decided to add it to this article. I am going to split this into 2 different sections. IIS 6, and other versions of IIS. To get started your going to want to make sure `osql.exe` is in your system path, or find it. It should be located wherever your SQL Server 2000 server/client tools directory. On my system it is something like this: `%Install Directory%\80\Tools\BINN\`.For simplicity I will use psuedo-variables in the examples so as not to create confusion. For example a psuedo-variable will look like this: `%VARIABLE%`. The server will be referred to as `%SERVER%\%INSTANCE%`. If you aren't using any instance names it can be just `%SERVER%`, `(local)` if the server is the local machine. If you are instance names it would be something like `ServerName\ServerInstance` etc etc. I will also be using `%DATABASE%` to refer to the database name.

IIS 6 on Windows 2003 Server

I know this will work on IIS 6 with Windows 2003 Server because I have done it and that is currently the only OS with IIS 6. On IIS 6 the ASP.NET process runs under the account '`NT AUTHORITY\NETWORK SERVICE`'.

Collapse | Copy Code

```
osql -E -S %SERVER%\%INSTANCE% -Q "sp_grantlogin 'NT AUTHORITY\NETWORK SERVICE'"
```

Now our ASP.NET application will be able to log into the server. Now all thats left is to grant access to the databases.

Collapse | Copy Code

```
osql -E -S %SERVER%\%INSTANCE% -d %DATABASE% -Q
  "sp_grantdbaccess 'NT AUTHORITY\NETWORK SERVICE'"
osql -E -S %SERVER%\%INSTANCE% -d %DATABASE% -Q
  "sp_addrolemember 'db_owner', 'NT AUTHORITY\NETWORK SERVICE'"
```

These 2 lines will add access to one of the databases. So if you want to add access to another database just change

%DATABASE% and run both lines.

IIS 5.1

This should work on all other IIS 5.1 (possibly other versions) combinations. The only difference between IIS 5.1 and IIS 6 is the account the ASP.NET process runs under. IIS 5.1 runs under a `%MACHINENAME%\ASPNET` where `%MACHINENAME%` is the machine name.

Collapse | Copy Code

```
osql -E -S %SERVER%\%INSTANCE% -Q "sp_grantlogin '%MACHINENAME%\ASPNET'"
```

Now our ASP.NET application will be able to log into the server. Now all thats left is to grant access to the databases.

Collapse | Copy Code

```
osql -E -S %SERVER%\%INSTANCE% -d %DATABASE%
  -Q "sp_grantdbaccess '%MACHINENAME%\ASPNET'"
osql -E -S %SERVER%\%INSTANCE% -d %DATABASE%
  -Q "sp_addrolemember 'db_owner', '%MACHINENAME%\ASPNET'"
```

These 2 lines will add access to one of the databases. So if you want to add access to another database just change %DATABASE% and run both lines.

Loose Ends

You now have the basics required to start using a SQL database in either webapplications or desktop applications.

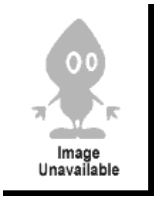
This article is by no means finished. I plan to expand the article and add a sample application as time allows. With information on stored procedures as well as an expanded connection options section. If you have any suggestions please leave them in the forum below

History

- **20 August 2004:** Added a section on SqlParameters
- **25 February 2004:** Added information on setting up permissions with IIS
- **2 July 2003:** Revised connection string section
- **28 June 2003**: Fixed a few typographical errors
- **27 June 2003:** Initial Release

License

About the Author



Matt Newman

United States

No Biography provided

Article Top

Comments and Discussions

You must Sign In to use this message board.

Search this forum Go

Profile popups Spacing Relaxed Noise Very High Layout Normal Per page 10 Update

First Prev Next

one of the best pages i've come across on the web for help with asp.net & sql server	Member 10463878	7-Jan-14 8:25
Connecting to sql server in c#	berkut2006	7-Nov-13 13:52
Here is the updated connection string	saiful_vonair	21-Oct-13 18:59
connection string for DB attached on user system	samira 2013	5-Oct-13 21:28
Quick Question on Parms	Michael Parzyck	14-Sep-13 16:13
Re: Quick Question on Parms [modified]	PIEBALDconsult	14-Sep-13 16:40
My vote of 5	Je7	7-Aug-13 10:40
Minh	hoangminh123	7-Aug-13 6:10
What does mean 'Love' in this context?	andyofmoscow	28-Jul-13 23:56
Re: What does mean 'Love' in this context?	david_dawkins	15-Dec-13 22:48

Last Visit: 31-Dec-99 18:00 Last Update: 6-Mar-14 2:13 Refresh 1 2 3 4 5 6 7 8 9 10 11 Next »

General News Suggestion Question Bug Answer Joke Rant Admin

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