SQL Server Connection Strings for ASP.NET Web Applications

.NET Framework 4.5

This topic provides examples of SQL Server connection strings for typical ASP.NET web application scenarios. It also includes instructions for converting connection strings between SQL Server Express and LocalDB, and explanations of some common connection string settings. The topic contains the following sections:

Connection String Examples

The examples show the connection strings as they appear in the **ConnectionStrings** element of the Web.config file, where the **connectionString** attribute must be accompanied by a **providerName** attribute.

In all of the examples, if you are using Entity Framework Code First, *ConnectionStringName* is typically the name of the context class. For more information about Entity Framework connection strings, see Entity Framework Connections and Models.

You can also construct connection strings in code by using the SqlConnectionStringBuilder API for SQL Server Express, LocalDB, SQL Server, or SQL Database. There is no corresponding API for SQL Server Compact.

LocalDB Connection String Example

The following example specifies the default automatic instance name for LocalDB.

Xml

```
<add name="ConnectionStringName"
    providerName="System.Data.SqlClient"
    connectionString="Data Source=
(LocalDB)\v11.0;AttachDbFileName=|DataDirectory|\DatabaseFileName.mdf;InitialCatalog=DatabaseName;Integrated
Security=True;MultipleActiveResultSets=True" />
```

For more information about connection string settings, see Data Source, AttachDbFileName, Initial Catalog, Integrated Security, and MultipleActiveResultSets later in this topic.

SQL Server Express Connection String Examples

The following example is for a SQL Server Express database that is defined in the local SQL Server Express instance. The example assumes that the SQL Server Express instance is named SQLEXPRESS, which is the default.

Xml

```
<add name="ConnectionStringName"
    providerName="System.Data.SqlClient"
    connectionString="Data Source=.\SQLEXPRESS;Initial Catalog=DatabaseName;Integrated Security=True;MultipleActiveResultSets=True"/>
```

The following example is for a SQL Server Express database in an .mdf file in the App_Data folder.

Xml

```
<add name="ConnectionStringName"
    providerName="System.Data.SqlClient"
    connectionString="Data Source=.\SQLEXPRESS;AttachDbFileName=|DataDirectory|\DatabaseFileName.mdf;Integrated Security=True;User
Instance=True;MultipleActiveResultSets=True" />
```

You can also use SQL Server security with a SQL Server Express database: specify the name and password and specify `Integrated Security=false`, as shown in the first SQL Server full edition example.

For more information about connection string settings, see Data Source, AttachDbFileName, Initial Catalog, Integrated Security, and MultipleActiveResultSets later in this topic.

SQL Server (Full Editions) Connection String Examples

The following example is for a SQL Server database using SQL Server security (log on to the server by using user credentials in the connection string). The example assumes that you are connecting to the default SQL Server instance on the server.

Xml

```
<add name="ConnectionStringName"
    providerName="System.Data.SqlClient"
    connectionString="Data Source=ServerName;Initial Catalog=DatabaseName;Integrated Security=False;User
Id=userid;Password=password;MultipleActiveResultSets=True" />
```

The following example is for a SQL Server database using integrated security (log on to the server using the credentials of the Windows user account). The example specifies a named instance of SQL Server.

Xml

```
<add name="ConnectionStringName"
    providerName="System.Data.SqlClient"
    connectionString="Data Source=ServerName\InstanceName;Initial Catalog=DatabaseName;Integrated
Security=True;MultipleActiveResultSets=True" />
```

For more information about connection string settings, see Data Source, Initial Catalog, Integrated Security, and MultipleActiveResultSets later in this topic.

Windows Azure SQL Database (formerly SQL Azure) Connection String Example

The following example shows a connection string for SQL Database (formerly SQL Azure).

Xml

```
<add name="ConnectionStringName"
    providerName="System.Data.SqlClient"
    connectionString="Data Source=tcp:ServerName.database.windows.net,1433;Initial Catalog=DatabaseName;Integrated Security=False;User
Id=username@servername;Password=password;Encrypt=True;TrustServerCertificate=False;MultipleActiveResultSets=True" />
```

The `TrustServerCertificate=False` setting is recommended to help protect against man-in-the-middle attacks. For more information, see How to: Connect to Windows Azure SQL Database Using ADO.NET.

For more information about connection string settings, see Data Source, Initial Catalog, Integrated Security, and MultipleActiveResultSets later in this topic.

Entity Framework Database First or Model First Connection String Example

The following example is for a full edition of SQL Server with Entity Framework Database First or Model First.

Xml

```
<add name="ConnectionStringName"
    providerName="System.Data.EntityClient"
    connectionString="metadata=res://*/ ContextClass.csdl|res://*/ ContextClass.ssdl|res://*/
ContextClass.msl;provider=System.Data.SqlClient;provider connection string=&amp;quot;Data Source=ServerName;Integrated Security=False;User
Id=userid;Password=password;MultipleActiveResultSets=True&amp;quot;" />
```

The part before the first **"** symbol specifies the conceptual model, data schema, and mapping information that is stored in the .edmx file. The part between the two **"** symbols is the database connection string. In this example, the database connection string is the same as the example for SQL Server using SQL Server security.

ContextClass in this example represents the fully qualified context class name (for example, *namespace.classname*). For more information about Entity Framework Database First or Model First connection strings, see Data Developer Center - Entity Framework - Connections and Models and ADO.NET Entity Framework Connection Strings.

SQL Server Compact Connection String Example

The following example is for a SQL Server Compact database located in the App_Data folder.

Xml

```
<add name="ConnectionStringName"
    providerName="System.Data.SqlServerCe.4.0"
    connectionString="Data Source=|DataDirectory|\DatabaseFileName.sdf" />
```

For more information about the **Data Source** setting, see Data Source later in this topic.

Converting Connection Strings between LocalDB and SQL Server Express

How to Convert a SQL Server Express Connection String to LocalDB

Project templates for Visual Studio 2010 and Visual Web Developer 2010 Express create connection strings that specify SQL Server Express databases. To convert one of these connection strings to LocalDB, make the following changes:

- Change "`Data Source=.\SQLEXPRESS`" to "`Data Source=(LocalDB\v11.0)`".

  This change assumes that you installed LocalDB with the default instance name. For more information, see Data Source later in this topic.

- Remove "`User Instance=True`" if it is present. Also remove the preceding or following semicolon (;).

How to Convert a LocalDB Connection String to SQL Server Express

Project templates for Visual Studio 2012 and Visual Studio Express 2012 for Web create connection strings that specify LocalDB databases. To convert one of these connection strings to SQL Server Express, make the following changes:

- Change "`Data Source=(LocalDB)\v11.0`" to "`Data Source=.\SQLEXPRESS`".

  This change assumes that you installed SQL Server Express with the default instance name. For more information, see Data Source later in this topic.

- If the connection string contains **AttachDBFileName**, add at the end "`;User Instance=True`". Omit the semicolon (;) if the connection string already ends with one.

Common Connection String Settings

In many cases, you can use different names in the connection string to configure the same setting; for example, the following two connection strings are equivalent:

Xml

```
<add name="ConnectionStringName"
    providerName="System.Data.SqlClient"
    connectionString="Data Source=
(LocalDB)\v11.0;AttachDbFileName=|DataDirectory|\DatabaseFileName.mdf;InitialCatalog=DatabaseName;Integrated
Security=True;MultipleActiveResultSets=True" />
```

Xml

```
<add name="ConnectionStringName"
    providerName="System.Data.SqlClient"
    connectionString="Server=(LocalDB)\v11.0;Initial File
Name=|DataDirectory|\DatabaseFileName.mdf;Database=DatabaseName;Trusted_Connection=True;MultipleActiveResultSets=True" />
```

In the following sections, the section title specifies the version of the setting name most commonly used in ASP.NET project templates. For settings that have alternative names, those are added in parentheses.)

Data Source (synonyms: Server, Addr, Address, Network Address)

For SQL Server Express, LocalDB, SQL Server, and SQL Database, this setting specifies the name of the server and the SQL Server instance on the server. For example, you can specify *ServerName**Instancename*. You can use ".", "(local)", or "localhost" in place of the server name to specify the local computer, and you can use an IP address instead of the server name. If you omit the instance name, the default instance is assumed. By default, SQL Server Express is installed without a default instance and with a named instance that is named SQLEXPRESS.

For LocalDB, use "(LocalDB)\v11.0" to refer to the automatic instance of LocalDB for SQL Server 2012. For named instances of LocalDB, replace "v11.0" with the name of the instance. For information about named instances of LocalDB, see SQL Server 2012 Express LocalDB.

For SQL Database the value of this setting typically has a "tcp:" prefix to indicate the protocol used and a ",1433" suffix to indicate the port number.

For SQL Server Compact, this setting specifies the path and name of the database file. You can use the **|DataDirectory|** variable in place of an absolute path; for information about this option, see the following section about the **AttachDBFileName** setting.

For more information, see the following resources:

- SqlConnection.ConnectionString
- SqlCeConnection.ConnectionString

AttachDbFileName (synonyms: Initial File Name, Extended Properties)

This setting specifies the path and name of the database file for SQL Server Express or LocalDB databases that are not defined in the local SQL Server Express instance.

This setting is typically used for database files that you keep in the App_Data folder. The App_Data folder is a relatively secure place to store database files that you want to keep in the web application folder structure. ASP.NET will not serve contents of the App_Data folder in response to Web requests, which helps to maintain the security of the data in database files kept in this folder. In place of the physical path to a database file that is stored in the App_Data folder, you can specify the **|DataDirectory|** variable in the **AttachDbFileName** setting. ASP.NET automatically substitutes the file path to the App_Data folder for the **|DataDirectory|** connection-string variable when it opens a connection to the database. This ensures that the path to your database remains current if the application is moved to a different directory. If you don't use the **|DataDirectory|** connection string variable, you have to provide the full physical path to the database file.

For information about the database name that is used when the file is attached by the SQL Server Express or LocalDB instance, see Initial Catalog later in this topic.

If you use this option with SQL Server Express, you also have to specify the User Instance option, which is deprecated. If you use this option with SQL Server Express, you won't find the database listed in the local SQL Server Express instance when you use SQL Server Management Studio or SQL Server Data Tools. For more information, see User Instance later in this topic.

The first time you connect to a database by using this option in the connection string, the SQL Server Express or LocalDB instance attaches the database, and it stays attached. When you want to connect to the same database in the future you could use **Initial Catalog** without **AttachDbFileName** if you prefer.

For more information, see SqlConnection.ConnectionString.

Initial Catalog (synonym: Database)

This setting specifies the name of the database in the SQL Server instance catalog.

> **Note**
>
> If the database will be used by the Entity Framework and your application targets the .NET Framework 4, you must include this setting for all connection strings except SQL Server Compact. You can omit this setting in applications that target the .NET Framework 4.5.

If you omit this setting, ADO.NET connects to the default database for the SQL Server instance specified in the **Data Source** setting.

If you omit this setting and include the **AttachDbFileName** setting for SQL Server Express or LocalDB, the full path to the database file is used as the database name. If the full path is longer than 128 characters, a default database name is constructed from the path by appending a hash of the full path to the file name.

In LocalDB connection strings, the Visual Studio web project templates add a unique number as a suffix to both the file name and the **Initial Catalog** setting, as shown in the following example:

Xml

```
<add name="DefaultConnection" connectionString="Data Source=(LocalDb)\v11.0;Initial Catalog=aspnet-ProjectName-
20120702083251;Integrated Security=SSPI;AttachDBFilename=|DataDirectory|\aspnet-ProjectName-20120702083251.mdf"
providerName="System.Data.SqlClient" />
```

The reason for this is to avoid database name collisions in the SQL Server Express LocalDB instance. Suppose the project specified the following values:

Xml

```
Initial Catalog=aspnet;AttachDBFilename=|DataDirectory|\aspnet.mdf"
```

When you create a project with these settings and run it, LocalDB attaches the aspnet.mdf file and names the database "aspnet" Then when you create another project and run that one, LocalDB tries to attach that project's aspnet.mdf file to the same database name "aspnet". The operation fails because that name is already attached to the first project's aspnet.mdf file.

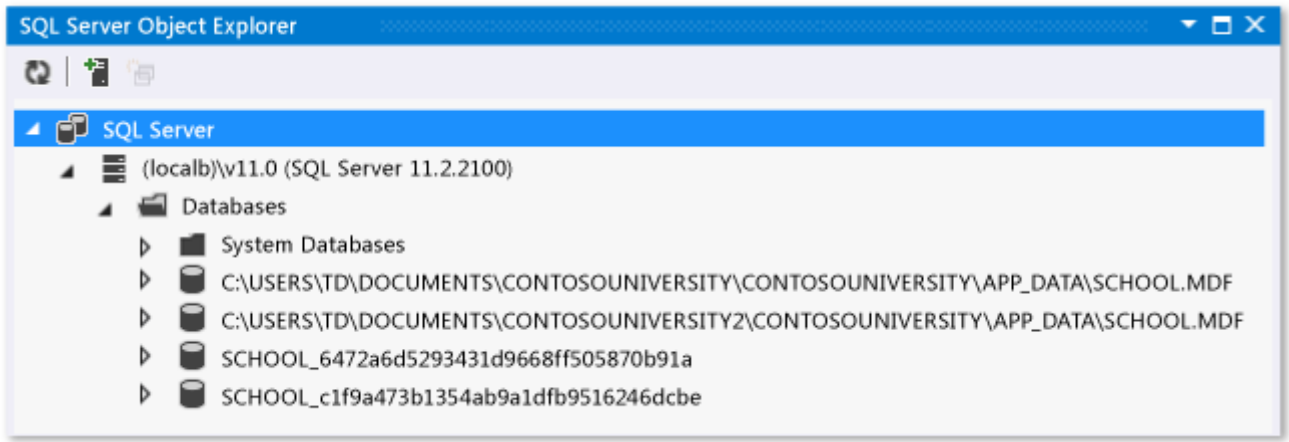
For the same reason, the suffixes added by the Visual Studio templates won't prevent name collisions if you create a copy of a project by copying the files instead of by creating a new project in Visual Studio, which generates a new unique number. If you want to create a new project by copying an existing project's files, change the suffix number manually to make it unique.

You can omit the **Initial Catalog** setting, as shown in this example:

Xml

```
<add name="DefaultConnection" connectionString="Data Source=(LocalDb)\v11.0;Integrated
Security=SSPI;AttachDBFilename=|DataDirectory|\aspnet.mdf" providerName="System.Data.SqlClient" />
```

With this connection string, what happens to the database name in the SQL Server instance is complicated. If the database is created by the ASP.NET membership system or by Entity Framework Code First, a database name is created by taking the file name without the .mdf extension, changing it to all caps, and appending a unique value similar to a GUID. If your own code creates the database, or if you detach the database and run the project again after the .mdf file is already created, the database name is the full path to the .mdf file, including the file name. If that name would be longer than 127 characters, the database name would be the file name with a hash of the full path appended to it. Some examples of these database names are shown in the following illustration:



For more information, see SqlConnection.ConnectionString.

Integrated Security (synonym: Trusted_Connection)

This setting specifies whether the connection should use the user ID and password in the connection string to log on to the SQL Server instance, or the current Windows account credentials should be used for authentication:

- **True** means use Windows integrated security to log in to SQL Server, even if the connection string includes **User ID** and **Password** settings.
- **False** means use SQL Server security to log in by using the **User ID** and **Password** values in the connection string, and raise an exception if they are not present.
- **SSPI** (Security Support Provider Interface) means use Windows Security if **User ID** and **Password** are absent, and use SQL Server security if they are present.

You have to use integrated security when you use LocalDB or the **AttachDBFileName** option. Otherwise,

For more information, see SqlConnection.ConnectionString.

MultipleActiveResultSets

The **MultipleActiveResultSets** (MARS) option makes it possible to execute multiple queries simultaneously. This is a common scenario when you use the Entity Framework, especially if you leave lazy loading enabled. For example, the second line of code in the following example will raise an exception if you don't enable MARS because the query that returns instructors is still active when the `ToList` method executes a new query that retrieves related courses:

```
foreach (var instructor in db.Instructors)
{
     var course = instructor.Courses.ToList();
}
```

It is often more efficient to load related data by using eager loading, which retrieves all of the data by using a single join query, but sometimes join queries are inefficient. In most real-world applications, there are times when the best choice for loading related data is lazy loading or explicit loading. Therefore, connection strings for databases that you access by using the Entity Framework typically specify this option. This option carries a slight performance penalty, but in most scenarios its benefits outweigh any performance loss even if you aren't using the Entity Framework.

For more information, see the following resources:

- Using Multiple Active Result Sets (MARS)
- Multiple Active Result Sets (MARS)

User Instance

This option is only used with SQL Server Express. User instances are not supported by LocalDB, full editions of SQL Server, or SQL Server Compact.

You use the **AttachDbFileName** setting to connect to a database in an .mdf file, typically located in the web application's App_Data folder. In order to connect to this database file, the SQL Server Express instance on your computer has to attach the file. But only SQL Server Express administrators can attach databases, so this operation will fail if you are not an administrator in the SQL Server Express instance. Also, even if you are an administrator and are able to attach the .mdf file, the default service account used by SQL Express might not have permissions to read and write the .mdf file. The service account typically does not have permissions for folders in your user profile, such as your My Documents folder, which is where Visual Studio projects are created by default. You use the **User Instance** option to solve these problems.

When you have enabled the **User Instance** option in the SQL Server Express instance and connect using a connection string that has **User Instance** set to **True**, a special instance of SQL Server Express is created for your user account. You are a SQL Server Express administrator in this user instance, so the instance is able to attach the database file. And when your web application runs under the Visual Studio Development Server (Cassini) or IIS Express, the user instance runs under your account, so it has access to folders in your user profile. For information about what identity is used for the user instance when the application runs in IIS, see ASP.NET Impersonation.

Databases that you have attached to a user instance are difficult to manage by using tools such as SQL Server Management Studio (SSMS). You won't see them in SSMS when you attach to the SQL Server Express instance on your computer.

Note
In SQL Server 2012, user instances are deprecated. If you need the **AttachDbFileName** functionality, we recommend that you use LocalDB instead of SQL Server Express.

For more information, see the following resources:

- SQL Server Express User Instances
- User Instances for Non-Administrators
- Connecting to SQL Express User Instances in Management Studio (SQL Server Express blog)
- SQL Server 2005 Express Edition User Instances

Connection Pool Settings

Connection pooling is enabled by default in ADO.NET. If you define multiple connection strings for the same database, they automatically use the same connection if the connection strings are identical. For example, you might have ASP.NET membership system data and your application's data in a single database, but the Web.config file might have two connection strings, one for the membership system and one for your application's Entity Framework context class. If these two connection strings are different in any way, even if they specify the same keywords but in a different order, they will use different connections. Several connection string settings control connection pooling behavior.

For example the default value for **Min Pool Size** is zero, which means all connections are closed after a period of inactivity. To make sure that connections are kept open, you can set **Min Pool Size** to a value greater than zero.

For more information, see the following resources:

- SQL Server Connection Pooling (ADO.NET)
- SqlConnection.ConnectionString

Additional Connection String Resources

For more information about connection strings, see the following resources:

- Connection Strings
- SqlConnection.ConnectionString
- SqlCeConnection.ConnectionString
- Data Developer Center - Entity Framework - Connections and Models
- Using Connection String Keywords with SQL Server Native Client
- ConnectionStrings.com

See Also

Concepts
ASP.NET Data Access Options
Other Resources
ASP.NET Data Access Content Map