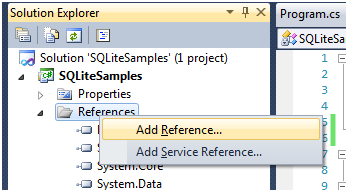
**Development Documentation**

**Creating a standard C# console project using SQLite**

Since we’re working in C# we’ll be using the System.Data.SQLite library. This library is not a standard library (packaged with .NET for example) so we’ll need to download it. It is being developed by the people who’re also working on the (original) SQLite.

All you’ll need are two files, a .dll and a .xml file for some documentation. Put these files in the folder of your project and add an assembly reference to the .dll. Just browse to System.Data.SQLite.dll and select it.

[](http://blog.tigrangasparian.com/wp-content/uploads/2012/02/Add-Assembly.png)

Now add using System.Data.SQLite; to the usings and you’re done. You’ve successfully added the SQLite library to you project!

**Creating a database file:**

You usually don’t need to create a new database file, you work with an existing one, but for those cases where you do need to create a brand new one, here’s the code:

|  |  |
| --- | --- |
| 1 | SQLiteConnection.CreateFile("MyDatabase.sqlite"); |

**Connecting to a database:**

Before you can use the database, you’ll need to connect to it. This connection is stored inside a connection object. Every time you interact with the database, you’ll need to provide the connection object. Therefore, we’ll declare the connection object as a member variable.

|  |  |
| --- | --- |
| 1 | SQLiteConnection m\_dbConnection; |

When creating a connection, we’ll need to provide a “connection string” this string can contain information about the… connection. Things like the filename of the database, the version, but can also contain things like a password, if it’s required.

The first one is good enough to get our connection up and running, so we get:

|  |  |
| --- | --- |
| 1  2  3 | m\_dbConnection =  new SQLiteConnection("Data Source=MyDatabase.sqlite;Version=3;");  m\_dbConnection.Open(); |

After we create the connection object, we’ll have to open it. And with every Open() there comes a Close(), so don’t forget to call that after you’re done with your connection.

Now we’ll need to create an SQL command in order to execute it. Luckily, we’ve got the SQLiteCommand class for this. We create a command by entering the sql query along with the connection object.

|  |  |
| --- | --- |
| 1 | SQLiteCommand command = new SQLiteCommand(sql, m\_dbConnection); |

Afterwards, we execute the command. But before we execute our command, i’d like to mention that not all commands are the same, some commands return results (like SELECT etc.) and others don’t (like the one we just wrote) That’s why there are two execute methods (actually, there are three) One returns the actual results (the rows of the table) the other returns an integer indicating the number of rows that have been modified. We’ll use the last one now.

|  |  |
| --- | --- |
| 1 | command.ExecuteNonQuery(); |

At this time, we’re not interested in the number of rows that have been modified (it’s 0) But you could imagine that it might be interesting to know this information in UPDATE queries.

**Entity Framework**

* For developing data-oriented software applications we use EF.
* The application interacts with a SQLite Engine.
* Functions that support this type of interactions is provided by the Entity Framework.

**Microsoft.Office.Interop.Excel**

* To add references
* In Solution Explorer, right-click your project's name and then click Add Reference. The Add Reference dialog box appears.
* On the Assemblies page, select Microsoft.Office.Interop.Word in the Component Name list, and then hold down the CTRL key and select Microsoft.Office.Interop.Excel. If you do not see the assemblies, you may need to ensure they are installed and displayed
* Click OK

**System.Data.Sqlite**

System.Data.SQLite is an ADO.NET provider for SQLite. System.Data.SQLite was started by Robert Simpson. Robert still has commit privileges on this repository but is no longer an active contributor. Development and maintenance work is now mostly performed by the SQLite Development Team. The SQLite team is committed to supporting System.Data.SQLite long-term.

**OleDbConnection**

* Object Linking and Embedding, Database (OLEDB).
* Used to convert the tables as object of a class and the columns as class variables.
* This was done since the size of each table changes based on the user requirement.
* The user adds/deletes columns based on the requirement which is to be mapped and linked to the database.
* The application allows the user to add columns to the excel sheet dynamically based on the user requirements
* This in turn adds a new column to the database table, thus making the ER diagram dynamic, which changes based on every sheet the user modifies
* This requirement allow us to indicate only one table in the database

**NuGet package**

* The application is developed in Visual Studio 2015
* To reference any package for the solution we are required to import NuGet packages
* The components mentioned in the earlier module were included using NuGet package manager
* This will be shown in the demo along with test cases and the actual application