FEBRUARY 6, 2017 2:21 PM

Using SQLite databases in UWP apps

By Gautam Kanumuru / Program Manager

SHARE TWEET SHARE SHARE SKYPE

For many developers, SQLite has become the preferred client-side technology for data storage. It is a server-less, embedded, open-source database engine that satisfies most local data access scenarios. There are numerous advantages that come with its use, many of which are explained in the SQLite about page.

Since the Windows 10 Anniversary Update (Build 14393), SQLite has also shipped as part of the Windows SDK. This means that when you are building your Universal Windows Platform (UWP) app that runs across the different Windows device form factors, you can take advantage of the SDK version of SQLite for local data storage. This comes with some advantages:

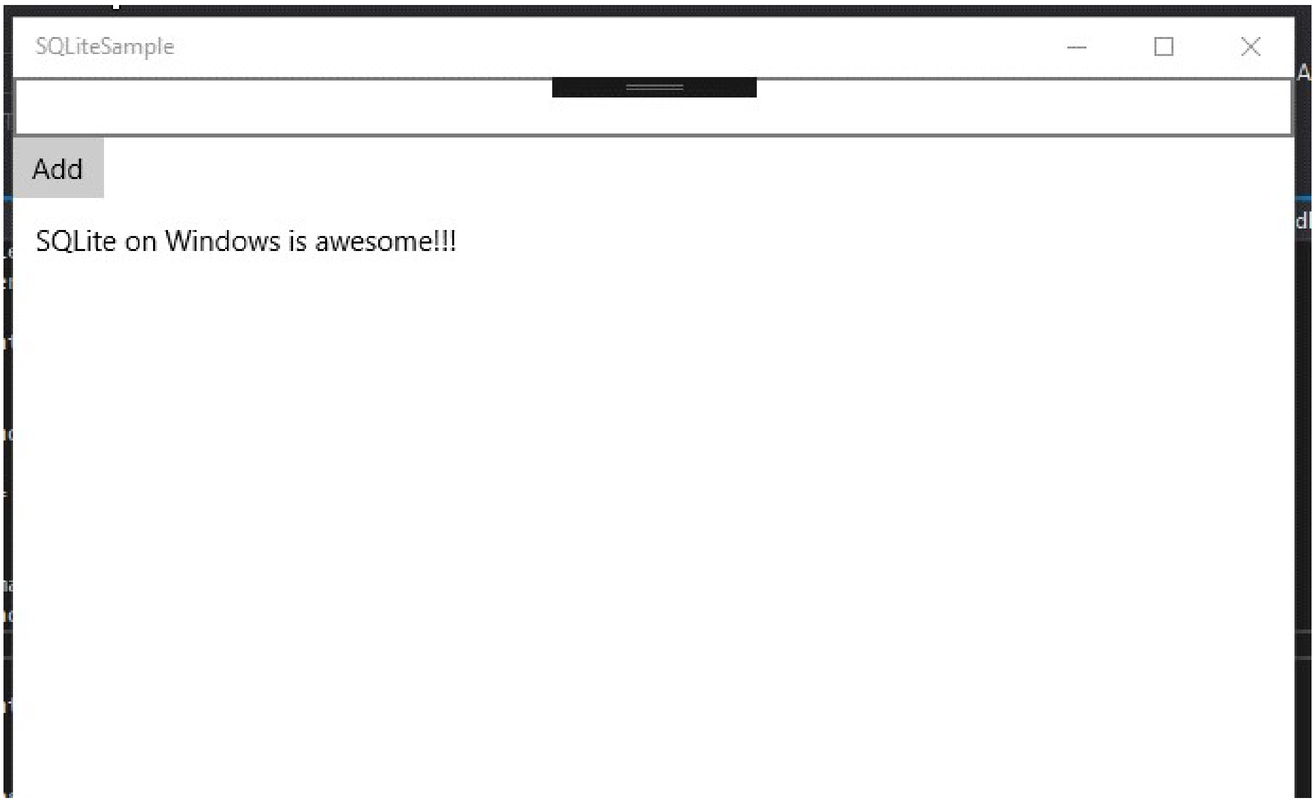
- Your application size reduces since you don't download your own SQLite binary and package it as part of your application
  - *Note: Microsoft.Data.SQLite (used in the example below) currently has an issue where both SQLite3.dll and WinSQLite.dll are loaded in memory whenever a .NET Native version of your application is run. This is a tracked issue that will be addressed in subsequent updates of the library.*
- You can depend on the Windows team to update the version of SQLite running on the operating system with every release of Windows.
- Application load time has the potential to be faster since the SDK version of SQLite will likely already be loaded in memory.

Below, we provided a quick coding example on how to consume the SDK version of SQLite in your C# application.

Note: Since the Windows SDK version of SQLite has only been available since the Windows 10 Anniversary Update, it can only be used for UWP apps targeting Build 14393 or higher.

shipped as part of the Windows SDK. Therefore this code sample is meant to be as simple as possible, so as to provide a foundation that can be further built upon.

An example of the end product is shown below:



SQLite on Windows is awesome!!!

SQLite C# API Wrappers

As mentioned in [the SQLite documentation](), the API provided by SQLite is fairly low-level and can add an additional level of complexity for the developer. Because of this, many open-source libraries have been produced to act as wrappers around the core SQLite API. These libraries abstract away a lot of the core details behind SQLite, allowing developers to more directly deal with executing SQL statements and parsing the results.

For SQLite consumption across Windows, we recommend the open-source [Microsoft.Data.Sqlite]() library built by the ASP.NET team. It is actively being maintained and provides an intuitive wrapper around the SQLite API. The rest of the example assumes use of the Microsoft.Data.Sqlite library.

Alternative SQLite wrappers are also linked in the “Additional Resources” section below.

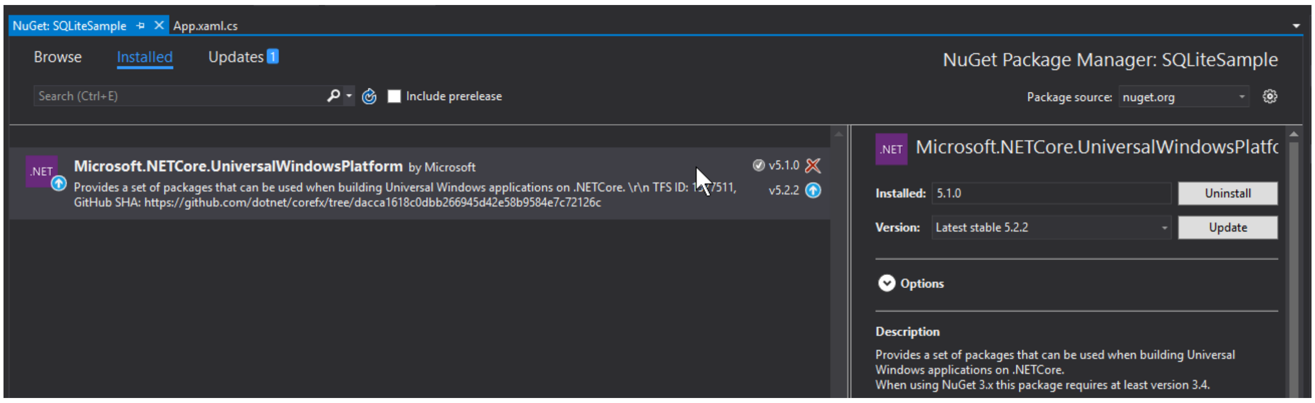
Products for NuGet Package Manager. You can go to the [NuGet download page]() and grab the version 3.5 VSIX update if you have a lower version.

Note: Visual Studio 2015 Update 3 is pre-installed with NuGet version 3.4, and will likely require an upgrade. Visual Studio 2017 RC is installed with NuGet version 4.0, which works fine for this sample.

Adding Microsoft.Data.Sqlite and upgrading the .NET Core template

The Microsoft.Data.Sqlite package relies on at least the 5.2.2 version of .NET Core for UWP, so we'll begin by upgrading this:

- Right click on *References ‣ Manage NuGet Packages*
- Under the *Installed* tab, look for the Microsoft.NETCore.UniversalWindowsPlatform package and check the version number on the right-hand side. If it's not up to date, you'll be able to update to version 5.2.2 or higher.



Microsoft.NETCore.UniversalWindowsPlatform by Microsoft
Provides a set of packages that can be used when building Universal Windows applications on .NETCore. \r\n TFS ID: 1[illegible]7511, GitHub SHA: https://github.com/dotnet/corefx/tree/dacca1618c0dbb266945d42e58b9584e7c72126c

Microsoft.NETCore.UniversalWindowsPlatfc

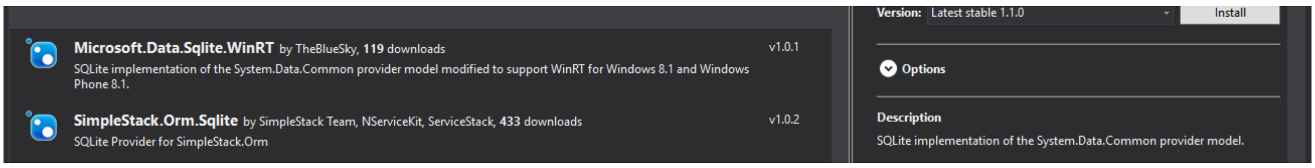
Installed: 5.1.0
Version: Latest stable 5.2.2

Description
Provides a set of packages that can be used when building Universal Windows applications on .NETCore.
When using NuGet 3.x this package requires at least version 3.4.

Note: Version 5.2.2 is the default for VS 2017 RC. Therefore, this step is not required if you are using this newest version of Visual Studio.

To add the Microsoft.Data.Sqlite NuGet package to your application, follow a similar pattern:

- Right-click on *References ‣ Manage NuGet Packages*
- Under the *Browse* tab, search for the Microsoft.Data.Sqlite package and install it.


Microsoft.Data.Sqlite.WinRT by TheBlueSky, 119 downloads v1.0.1
SQLite implementation of the System.Data.Common provider model modified to support WinRT for Windows 8.1 and Windows Phone 8.1.
SimpleStack.Orm.Sqlite by SimpleStack Team, NServiceKit, ServiceStack, 433 downloads v1.0.2
SQLite Provider for SimpleStack.Orm
Version: Latest stable 1.1.0 Install
Options
Description
SQLite implementation of the System.Data.Common provider model.

Code

Application User Interface

We'll start off by making a simple UI for our application so we can see how to add and retrieve entries from our SQLite database.

```
<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
    <StackPanel>
        <TextBox Name="Input_Box"></TextBox>
        <Button Click="Add_Text">Add</Button>
        <ListView Name="Output">
            <ListView.ItemTemplate>
                <DataTemplate>
                    <TextBlock Text="{Binding}"/>
                </DataTemplate>
            </ListView.ItemTemplate>
        </ListView>
    </StackPanel>
</Grid>
```

There are three important parts to our application's interface:

1. A text box that allows us to take text from the user.
2. A button linked to an event for pulling the text and placing it in the SQLite database.
3. An ItemTemplate to show previous entries in the database.

Code Behind for Application

In the App.xaml.cs and MainPage.xaml.cs files generated by Visual Studio, we'll start by importing the Microsoft.Data.Sqlite namespaces that we'll be using.

```
using Microsoft.Data.Sqlite;
using Microsoft.Data.Sqlite.Internal;
```

Then as part of the app constructor, we'll run a "CREATE TABLE IF NOT EXISTS" command to guarantee that the SQLite .db file and table are created the first time the application is launched.

```
    {
        db.Open();
        String tableCommand = "CREATE TABLE IF NOT EXISTS MyTable (Primary_Key
        SqliteCommand createTable = new SqliteCommand(tableCommand, db);
        try
        {
        createTable.ExecuteReader();
        }
        catch (SqliteException e)
        {
            //Do nothing
        }
    }
}
```

There are couple of points worth noting with this code:

1. We make a call to SqliteEngine.UseWinSqlite3() before making any other SQL calls, which guarantees that the Microsoft.Data.Sqlite framework will use the SDK version of SQLite as opposed to a local version.
2. We then open a connection to a SQLite .db file. The name of the file passed as a String is your choice, but should be consistent across all SqliteConnection objects. This file is created on the fly the first time it's called, and is stored in the application's local data store.
3. After establishing the connection to the database, we instantiate a SqliteCommand object passing in a String representing the specific command and the SqliteConnection instance, and call execute.
4. We place the ExecuteReader() call inside a try-catch block. This is because SQLite will always throw a SqliteException whenever it can't execute the SQL command. Not getting the error confirms that the command went through correctly.

Next, we'll add code in the View's code-behind file to handle the button-clicked event. This will take text from the text box and put it into our SQLite database.

```
private void Add_Text(object sender, RoutedEventArgs e)
{
    using (SqliteConnection db = new SqliteConnection("Filename=sqliteSample.d
    {
        db.Open();

        SqliteCommand insertCommand = new SqliteCommand();
        insertCommand.Connection = db;

        //Use parameterized query to prevent SQL injection attacks
        insertCommand.CommandText = "INSERT INTO MyTable VALUES (NULL, @Entry)
        insertCommand.Parameters.AddWithValue("@Entry", Input_Box.Text);

try
        {
            insertCommand.ExecuteReader();
        db.Close();
    }
    Output.ItemsSource = Grab_Entries();
}
```

As you can see, this isn't drastically different than the SQLite code explained in the app's constructor above. The only major deviation is the use of parameters in the query so as to prevent SQL injection attacks. You will find that commands that make changes to the database (i.e. creating tables, or inserting entries) will mostly follow the same logic.

Finally, we go to the implementation of the Grab_Entries() method, where we grab all the entries from the Text_Entry column and fill in the XAML template with this information.

```
private List<String> Grab_Entries()
{
    List<String> entries = new List<string>();
    using (SqliteConnection db = new SqliteConnection("Filename=sqliteSample.d
    {
        db.Open();
        SqliteCommand selectCommand = new SqliteCommand("SELECT Text_Entry fro
        SqliteDataReader query;
        try
        {
            query = selectCommand.ExecuteReader();
        }
        catch(SqliteException error)
        {
            //Handle error
            return entries;
        }
        while(query.Read())
        {
            entries.Add(query.GetString(0));
        }
        db.Close();
    }
    return entries;
}
```

Here, we take advantage of the SqliteDataReader object returned from the ExecuteReader() method to run through the results and add them to the List we eventually return. There are two methods worth pointing out:

1. The Read() method advances through the rows returned back from the executed SQLite command, and returns a boolean based on whether you've reached the end of the query or not (True if there

are dealing with.

1. The ordinal parameter isn't as relevant in this example since we are selecting all the entries in a single column. However, in the case where multiple columns are part of the query, the ordinal represents the column you are pulling from. So if we selected both Primary_Key and Text_Entry, then GetString(0) would return the value of Primary_Key String and GetString(1) would return the value of Text_Entry as a String.

And that's it! You can now build your application and add any text you like into your SQLite database. You can even close and open your application to see that the data persists.

A link to the full code can be found at: https://github.com/Microsoft/windows-developer-blog-samples/tree/master/Samples/SQLiteSample

Moving Forward

There are plenty of additions that you can make to tailor this sample to your needs:

- Adding more tables and more complicated queries.
- Providing more sanitation over the text entries to prevent faulty user input.
- Communicating with your database in the cloud to propagate information across devices.
- And so much more!

What about Entity Framework?

For those developers looking to abstract away particular database details, Microsoft's Entity Framework provides a great model that lets you work at the "Object" layer as opposed to the database access layer. You can create models for your database using code, or visually define your model in the EF designer. Then Entity Framework makes it super-easy to generate a database from your defined object model. It's also possible to map your models to existing databases you may have already created.

SQLite is one of many database back-ends that Entity Framework is configured to work with. This documentation provides an example to work from.

Conclusion

it ships as part of Windows 10, you can have peace of mind, knowing that you're always using an up-to-date version of the binary.

As always, please leave any questions in the comments section, and we'll try our best to answer them. Additional resources are also linked below.

Additional Resources

- SQLite Documentation from SQLite page
- Additional open source SQLite wrappers available via NuGet:
  - Net
  - raw
- Entity Framework Documentation

TAGS .NET core c# Entity Framework local data storage nuget open source SQLite SQLite API UWP UWP apps Visual Studio Windows 10 Anniversary windows sdk

SHARE TWEET SHARE SHARE SKYPE

Join the conversation

Please sign in to comment
Secured by OneAll Social Login

Guilherme Petenuci *1 year ago*

THANK YOU !!

Eugeny Ipatov *1 year ago*

How about improving ESENT (Jet Blue) support on .NET? Windows already have good built-in database engine, but it lack modern and easy-to-use C# API

It looks like you're very familiar with this GitHub issue (https://github.com/aspnet/EntityFramework/issues/4423) from the Entity Framework repository :). This is definitely on our radar, but I unfortunately can't give a specific timeline on when we can provide this. Its part of the greater set of database providers we are trying to build support for on UWP.

Hiro Kuo K. *1 year ago*

great work, nice job!

Dmitry Ponomarenko *1 year ago*

Are there any NuGet packages of SQLite-Net Extensions compatible with sqlite-net-pcl or should I build it myself? I found only a version depending on SQLite.Net-PCL.

Gautam Kanumuru *1 year ago*

I'm not as familiar with SQLite-Net and SQLite-Net Extensions unfortunately. However from what I can see with on the SQLite-Net Extension repository (https://bitbucket.org/twincoders/sqlite-net-extensions), they have a couple of NuGet packages to deal with the different SQLite-Net versions. Hopefully this helps.

Anders Ingemarsson *1 year ago*

Thanks for this. It is really good that SQLite is supported in UWP apps. But please note that this fantastic little db has some quirks… You should not really use autoincrement in normal cases as done here in "Primary_Key INTEGER PRIMARY KEY AUTOINCREMENT". Just use INTEGER PRIMARY KEY as you then get autoincrement for free. The use of key word AUTOINCREMENT will work but it will also imposes extra CPU, memory, disk space, and disk I/O overhead and is only to be used in some special cases. Please check out http://sqlite.org/autoinc.html for more info

Gautam Kanumuru *1 year ago*

Thanks Anders for pointing out this quirk! I'll make the changes in the GitHub repo soon.

This is exactly what I was looking for. Most other examples that I've been able to find gave overly complicated and outdated solutions. Great post!!

Ігор Фастнахт

1 year ago

Hello friends.

Could anybody help me. I'm using approach that described in this article, but
I've crash in next line:
– db.Open()
without any exception or errors.
Please help
Thanks

Gautam Kanumuru

1 year ago

Could you elaborate a little more on your issue? Are you hitting this on application launch (i.e. when creating the tables if they haven't been created before), or when you are trying to add entries to the database?

Adrian Johnson

1 year ago

Hello,
I get exactly the same issue. In my app, the db already exists and I'm running a query to insert data into the database. When I step into the line db.Open() the app crashes and there is no exception.
The only difference between my app and the example here, is that my db already existed so I don't use the CREATE TABLE command to create a table or db.
I'm using VS2015 on W10 Anniversary Update.
Thanks.

Tyler Hardy

1 year ago

I'm getting the same behavior over here too. In fact, when I set a breakpoint on db.open() in the sample you provided it throws the exception as well. But, if there isn't a breakpoint it runs fine.

Again, thanks for the article! Any help from anybody would be appreciated.

tulika baranwal

1 year ago

When you say this, "Since the Windows SDK version of SQLite has only been available since the Windows 10 Anniversary Update, it can only be used for UWP apps targeting Build 14393 or higher", does it mean that it would not work for apps targeting build versions lower than 14393 ?

Shibo Wei

1 year ago

I think YES. just only for SDK version is 14393+

Gautam Kanumuru

12 months ago

Yep, Shibo is correct.

Mateusz Piasecki

1 year ago

Is it normal that I can't debug this code? When I put breakpoint in using SqliteConnection part application closes (without exception ???)

Tony Palumbo

1 year ago

Interesting Article. Thank you.
I am curious. In creating Windows 10 apps I used the SQLite.Net-PCL reference and have used lambdas like such:

public static string dbName = "TM.db";
public static string DB_PATH = Path.Combine(Windows.Storage.ApplicationData.Current.LocalFolder.Path, dbName);
public static SQLitePlatformWinRT SQLITE_PLATFORM = new SQLitePlatformWinRT();
public static SQLiteConnection DB_CONNECTION = null;

Then I connect and run queries as such:
using (SQLiteConnection db = DbConnection)
{
foreach (var item in db.Table().OrderBy(c => c.LastName))
{
oc.Add(item);
}
}

If I try to do the same using the Microsoft.Data.SQLite the above is not applicable anymore?
Is there a way to do the same?

Carlo Mendoza *1 year ago*

How does this relate to SQLiteStore for offline/syncing apps?

Raiford Bonnell *1 year ago*

From Blog
"Under the Installed tab, look for the Microsoft.NETCore.UniversalWindowsPlatform package and check the version number on the right-hand side. If it's not up to date, you'll be able to update to version 5.2.2 or higher."

A version higher than 5.2.2 requires Visual Studio 2017.

Gautam Kanumuru *12 months ago*

I don't believe that is correct. Using NuGet, I was able to update to version 5.2.2. If you right-click on 'References' in your solution explorer and click 'Manage NuGet Packages…' you should be presented with a UI. From there, you can look in the 'Installed' tab to find the Microsoft.NETCore.UniversalWindowsPlatform package and use the pane on the right to update your version.

Paul Wainwright *12 months ago*

Gautam Kanumuru

12 months ago

There aren't any Microsoft.Data.SQLite specific implementations that allow for syncing to an Azure hosted DB. However having a cohesive story for syncing between local DBs and cloud DBs is something on our radar to support in the coming iterations.

Diego Vega

12 months ago

For those reporting a crash while debugging on connection open: We believe it is due to the issue detailed at https://github.com/aspnet/Microsoft.Data.Sqlite/issues/342, and we are aiming to fix it in an upcoming patch of Microsoft.Data.Sqlite.

Cody Wetzel

11 months ago

I am having a lot of trouble…

How do I use my own pre-existing database? I add the database file to my project as content, copy always but can't seem to access it.

Any help would be greatly appreciated.

家琦 冯

2 months ago

Hi there. I'm having same issue. Have you found any solution yet?

javier ramos

11 months ago

Hello:
In the reading part of the SQLite can be done on another page, SQLite-pcl.net or Microsoft Data SQLite

Podria poner un ejemplo utilizando clase.cs, despues la parte de lectura "Select" se puede hacer otra pagina.xaml

Roberto Carrer

10 months ago

Godo article!
I have a question: if my database is very large what is the best way for read/write data?
I opened the connection any time or open the connection when at starting app and close at the end?
You have a sample with the complex database?
Thank you!

Stefan Tucker

10 months ago

Might there be support for REGEXP (and, therefore, SQLiteFunction) soon?

Elias Pujols

9 months ago

Great!
Thank you!

Dima Maltsev

9 months ago

Make sure you use NuGet 3.5 with VS 2015!
I hope this will help anyone who is using VS 2015 and running into the 'sqlite3 not found' issue. I've spent hours trying to resolve it. The odd thing was that everything worked on one box, but the app would not run in the emulator or device if it was compiled on the other machine. After enabling the "Detailed" verbosity level for the build options, I confirmed that the wrong version of sqlite3 was pulled into the build. I was getting win7-x86 version instead of win10-x86. To cut a long story short, pay attention to what the article says about NuGet version 3.4. It turned out that my other machine had that version, which kind of worked in terms of pulling the package, resolving the references and compiling code (which even worked on my local machine), however, it would not pull the correct version for ARM processors and UWP. By the way, WACK tool would also give a ton of errors like this: "sqlite3.dll has failed the AppContainerCheck check" and "API AreFileApisANSI in kernel32.dll is not supported for this application type. sqlite3.dll calls this API."

Anyway, after upgrading NuGet to 3.5 everything now works fine and the App Package passes the Certification Test.
http://www.companova.com

How to use pre-designed SQLite in UWP project with Microsoft.Data.Sqlite??

Thanks.

Robert van Dam *7 months ago*

the two using directives have squiggles underneath .data

VS2017 says:

The type or namespace name 'Data' does not exist in the namespace 'Microsoft'

Under references I can clearly see Microsoft.Data.Sqlite

Bryan Eriksson *6 months ago*

Same here 🙁

Windows LiveUser1255 *6 months ago*

I have same issue and i have installed the below namespaces in my project
VS2015:
The type or namespace name 'Data' does not exist in the namespace 'Microsoft'

Microsoft.NETCore.UniversalWindowsPlatform 5.2.2

Microsoft.Data.Sqlite 2.0.0

Mike Tanis

4 months ago

I'm an entry-level programmer. I'm using Win10-64 version 1709 and Visual Studio 2017 (Version 15.4.3) with System.Data.SQLite version 1.0.106 installed via NuGet.

I couldn't get this code to work until I did 2 things.

1. I had file exception error on db.Open() until I just tested for the existence of the Database and created the file if it didn't exist.

```
if (! File.Exists(myDB))
{
SQLiteConnection.CreateFile(myDB);
}
```

2. I had to change the connection string to explicitly state the version number.

```
using (SQLiteConnection db = new SQLiteConnection("DataSource=" + myDB + ";Version=3;"))
```

Thank you for a nice article that got me started down the correct methods!

DAN POPA

2 months ago

Thank You for this article.
With this sample I can use one file type SQLite.db.
And write one information .
I want to use this file for using in an app for phone type MICROSOFT LUMIA 550 in which
i put informations about 4 persons max.
I wrote an app Meeting2_4 (in store for w10 , is free) , and my file will contain 4 rows (person) ,
and 3columns (about birthday: day , month, year).
But I want a routine for READ from file , and Delete file if I want to restart test.
I try very much ...(DROP or DELETE but I do not realize...°
Therefore, please give me indications for using CRUD oprations with SQLite3
Thank you again
danmagdapopa_603@hotmail.com

3. I have installed using NUGET:
3.1 Windows.Data.Sqlite V2.0.0
3.2 Windows.NETCore.Universal Windows Platform v6.0.7
3.3 SQLitePCLRaw.bundle_green V1.1.9
Unfortunately I'm failing at the first hurdle because Microsoft.Data.Sqlite isnt available.

Any idea what I could be doing wrong?

Thanks

Peter Kent

4 weeks ago

Please ignore this post, I have sorted the problem

RELATED POSTS

ICYMI – Your weekly TL;DR
Read more

Data Access in Universal Windows Platform (UWP) Apps
Read more

Using the SQLite database engine with Windows Phone 8 apps
Read more

SQLite WinRT wrapper for Windows Phone
Read more

Windows 10 SDK Build 10586 breakdown
Read more