SQLiteKit

Abstract

The SQLiteKit is cross-platform pure C# library which replaces native sqlite library plus gives true multi-threaded disconnection between database working thread and unity main render thread. Provide few obstruction layers(wrappers) on top of `native/original` sqlite functions and promises to run on all unity supported platforms with ease. The package is based on http://code.google.com/p/csharp-sqlite which was adopted, optimized and fixed for Unity.

Environment

On your developer machine it would require to install Unity 2020.3.0.f1 or latest. There is no external libraries or platform dependencies.

Package content

As mentioned earlier there are no externals, so let’s look at what you get. The package contains ported code from csharp-sqlite project located at `Assets/SQLiteKit/src` folder. Two test database files ‘db.sqlite’ and ‘test.db’ at StreamingAssets as well as in sqlitekit/CopyToStreamingAssets just to keep copies if you need to restore them.
The rest are the list files below which I will explain one-by-one.

Editor/SQLiteKitEditor.cs – google spreadsheet import example
DemoObject1.cs – basic test & examples (demo1_basic scene)
DemoObject2.cs – shows how to run multi-threading (demo2_run_async_db scene)
DemoObject3.cs – database encryption (demo3_encryption scene)
DemoObject4.cs – example using yield functionality (demo4_yield_banchmark scene)
SQLiteAsync.cs – Glue between ThreadQueue & SQLiteDB.
SQLiteDB.cs – hold and manage a single sqlite database file.
SQLiteExtension.cs – C# extension which provides yield database functions.
SQLiteQuery.cs – hold and manage a single database request.
ThreadQueue.cs – utility code which allows communication and control over execution code between different threads.

Pure SQLite.ORG (C style) programming coding style

Make it short – SQLiteKit is SQLite.ORG with nice wrappers over “native” code. It allows to work with full stack of functions from the list: https://www.sqlite.org/c3ref/funclist.html but in C# ported variants. Small example:

```
using Sqlite3 = Community.CsharpSqlite.Sqlite3;
using sqlite3 = Community.CsharpSqlite.Sqlite3.sqlite3;
using vdbe = Community.CsharpSqlite.Sqlite3.Vdbe;

sqlite3 db;
vdbe vm;
if ( Sqlite3.sqlite3_open( ":memory:", out db ) == Sqlite3.SQLITE_OK )
{
    if( Sqlite3.sqlite3_prepare16_v2( db, "DROP TABLE IF EXISTS test_values;", -1, ref
vm, 0) == Sqlite3.SQLITE_OK )
    {
        Sqlite3.sqlite3_step( vm );
        Sqlite3.sqlite3_reset( vm );
        Sqlite3.sqlite3_finalize( vm );
    };
    Sqlite3.sqlite3_close( db );
}
```

So, you are free to do so. But if you want the easy way, I will show SQLiteKit wrappers next...

SQLiteDB & SQLiteQuery coding style

It is the most favorable way to code with SQLiteKit and the easiest one. Two classes SQLiteDB and SQLiteQuery are just wrappers over “native” functions with a lot of sugar.
Let repeat the same code above:

```
SQLiteDB db = new SQLiteDB();
db.OpenInMemory();

SQLiteQuery qr = new SQLiteQuery(db, "DROP TABLE IF EXISTS test_values;");
qr.Step();
qr.Release();

db.Close();
```

And some little bit more complex example - reading table with conditions:

```
SQLiteDB db = new SQLiteDB();
db.Open(filename);

SQLiteQuery qr = new SQLiteQuery(db, "SELECT * FROM test_values WHERE str_field=?;");
qr.Bind("some string...");
while(qr.Step()) {
    string aString = qr.GetString("str_field");
    //... do something
}
qr.Step();
qr.Release();

db.Close();
```

Mode examples and explanations you will find in code and most importantly in DemoObject1.cs files.

Asynchronies callbacks coding style

There we go, this code style is really painstaking – it's required to pay attention to sequence of callbacks but in exchange your database operations will be isolated from the main unity thread and has no performance penalties in FPS of your game or project. That functionality is implemented in SQLiteAsync class and it is wrapper over (incapsulate) SQLiteDB and SQLiteQuery and uses SQLiteManager to hold active open databases plus glued all together with ThreadQueue (thread messaging util).

In DemoObject2.cs file show all possible use-cases, so I will show only one case here:

```
  private SQLiteAsync asyncDB = null;
  private string queryCreate = "CREATE TABLE IF NOT EXISTS test_values (id INTEGER PRIMARY KEY, str_field TEXT, blob_field BLOB);";
  private string queryInsert = "INSERT INTO test_values (str_field,blob_field) VALUES(?,?);";
  private string testString = "1231 \n\r \t weqw";
  private byte[] testBlob = new byte[] {2,3,5,78,98,21,32,255};
  private int recordsNum = 1000;
  private int frameCount = 0;
  private TestCallbackData demoData = null;

  // Use this for initialization
  void Start () {
    asyncDB = new SQLiteAsync();
    // open database
    asyncDB.Open(Application.persistentDataPath + "/demo_for_async_db.db",null,null);
    // create test table.
    asyncDB.Query(queryCreate, null, CreateQueryCreated, null);
  }
```

Last line of Start() function will request creation of query which eventually end-up with creation "test_values" table. It's wait CreateQueryCreated callback:

```
//
//   CREATE CALLBACKS
//

void CreateQueryCreated(SQLiteQuery qr, object state)
{
   // call step asynchronously
   asyncDB.Step(qr, null, CreateStepInvoked, state);
}


void CreateStepInvoked(SQLiteQuery qr, object state)
{
   // call release asynchronously
   asyncDB.Release(qr, CreateQueryReleased, state);
}

void CreateQueryReleased(object state)
{
   // nothing to do here
}
```

On CreateQueryCreated call it's request asyncDB.Step(…) with CreateStepInvoked
On CreateStepInvoked call it's request asyncDB.Release(…) with CreateQueryReleased
On CreateQueryReleased everything done, but you could put something next to do if you will.

As it shown it's really "not common" way to work with a database, but if you need to record something huge in real-time (track player or NPC position, for example) it will help a lot.

Yield functions coding style

Don't be scared, there is a wrapper for callbacks from chapter above and it is implemented in SQLiteExtension.cs file. What it does - it wraps the callbacks by Unity yield compatible function set, which now you could use with ease. So, this is the top wrapper which is multi-threaded and has exactly the same benefits as the previous one and improves programmers experience and reduces the chance of mistakes. But I will still recommend to use "SQLiteDB & SQLiteQuery" code style in case you do not need a thousand requests in real-time as long as there is a price in memory footprint and penalties due to lag in thread synchronization (messaging).

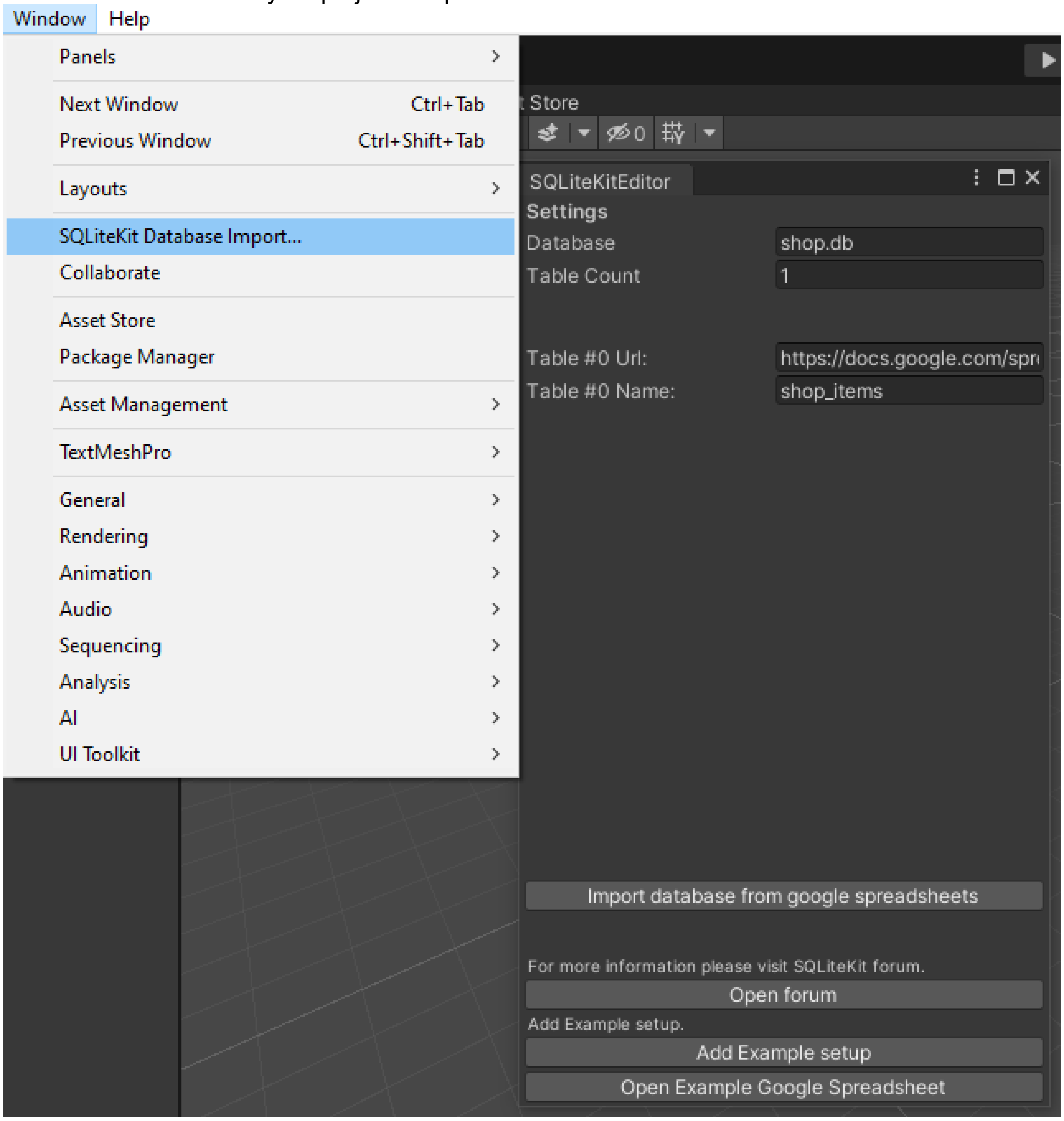
Let’s have a look (DemoObject4.cs):

```
IEnumerator AsyncPerformanceTestCoroutine()
{

   // ... copy db.sqlite in to Application.persistentDataPath

   SQLiteExt.Handle handle = new SQLiteExt.Handle();
   // Open Database
   yield return this.SQLiteOpenDatabase(Application.persistentDataPath + "/db.sqlite", handle);
   yield return this.SQLiteQuery("SELECT * FROM en WHERE word like 'a%' LIMIT 1", null, handle);
   yield return this.SQLiteStep(null, handle);
   if(handle.Success)
   {
      UnityEngine.Debug.Log(handle.Query.GetString("word"));
   }
   yield return this.SQLiteRelease(handle);
   // Close database
   yield return this.SQLiteCloseDatabase(handle);
}
```

Google Spreadsheets Importer

It’s very common for game development pipe-line then in-game data stored/shared through google spreadsheets to be easily modifiable by game designers and programmers. So, the task to reintegrate google spreadsheet back into the client in sqlite database representation is a really common task.

Editor/SQLiteKitEditor.cs – is a simple Unity Editor Window google spreadsheet importer, which could be used as the basis for your project. It’s provided a window and menu item:



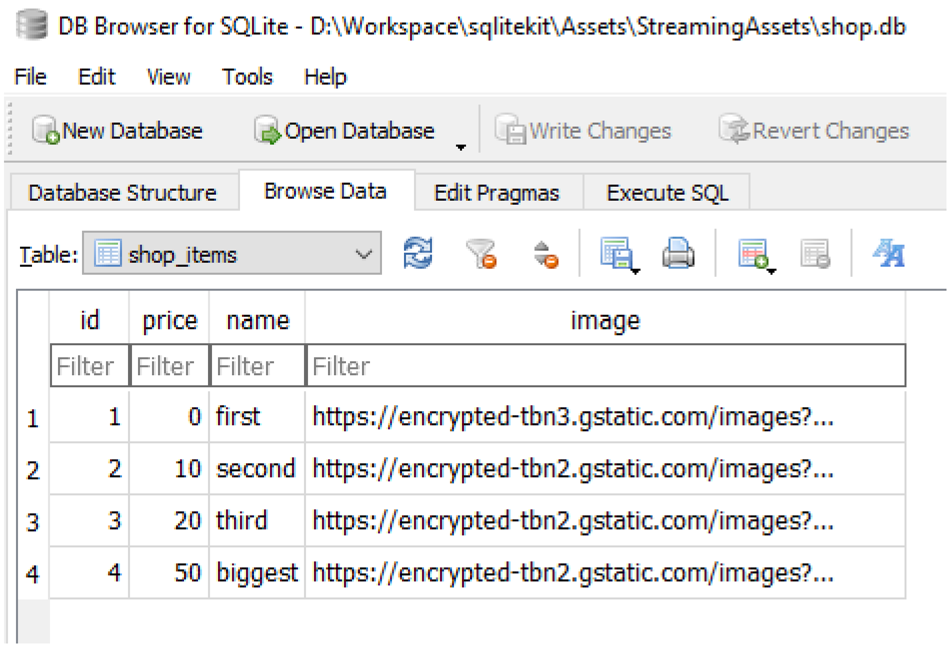
Settings

Database	shop.db
Table Count	1
Table #0 Url:	https://docs.google.com/spr
Table #0 Name:	shop_items

For more information please visit SQLiteKit forum.

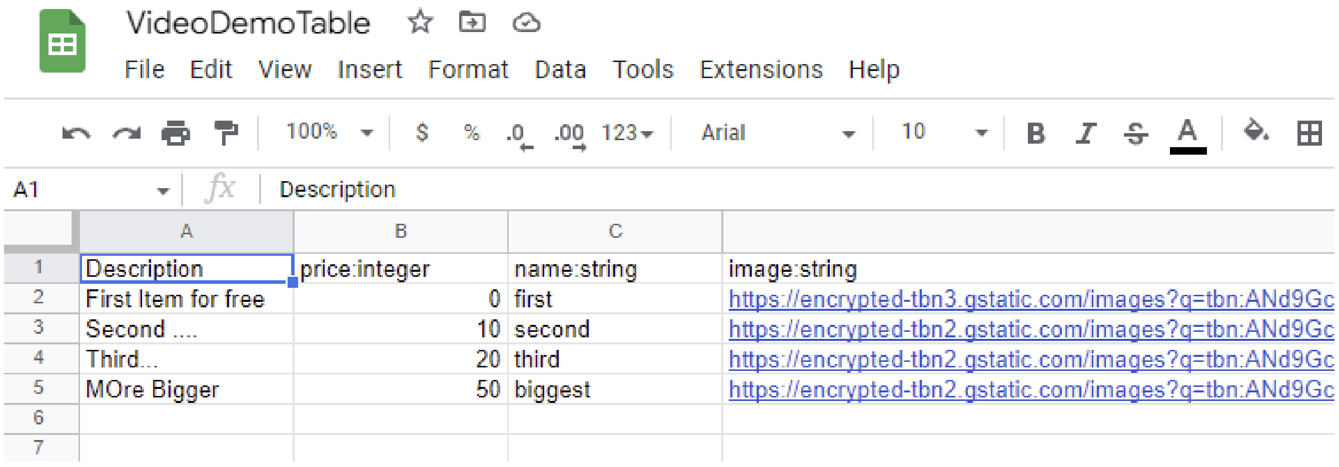
Add Example setup.

By clicking on “Import database from google spreadsheet” the script will import google document into the “shop.db” database, which is stored in StreamingAssets.



	id	price	name	image
1	1	0	first	https://encrypted-tbn3.gstatic.com/images?...
2	2	10	second	https://encrypted-tbn2.gstatic.com/images?...
3	3	20	third	https://encrypted-tbn2.gstatic.com/images?...
4	4	50	biggest	https://encrypted-tbn2.gstatic.com/images?...

It’s few missing steps which are creation of the spreadsheet itself and sharing setup.
At first you need to create spreadsheet like this:

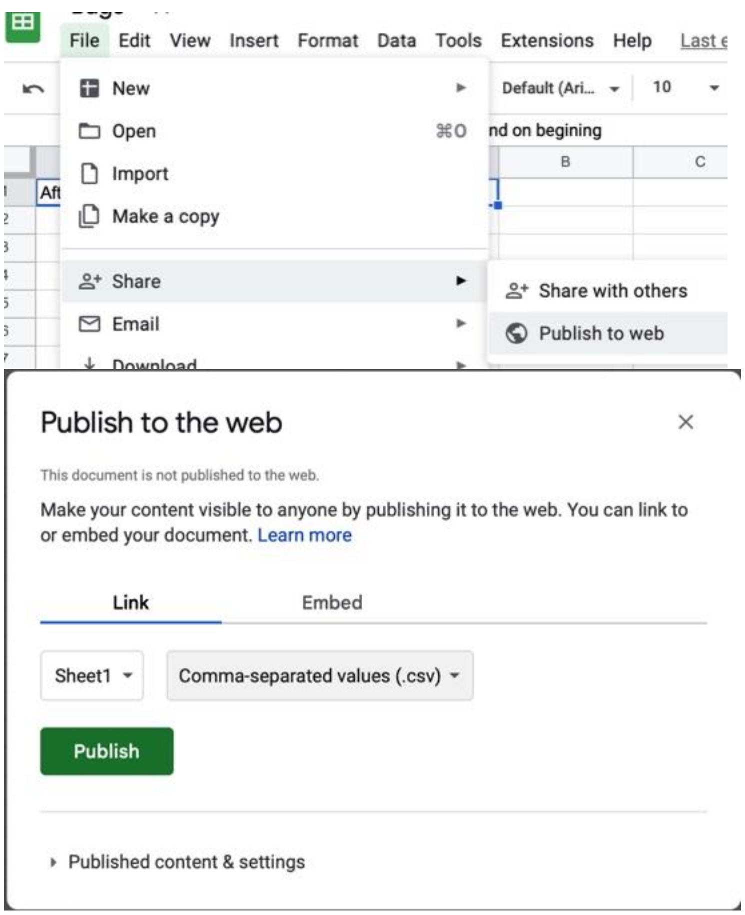


	A	B	C	
1	Description	price:integer	name:string	image:string
2	First Item for free	0	first	https://encrypted-tbn3.gstatic.com/images?q=tbn:ANd9Gc
3	Second	10	second	https://encrypted-tbn2.gstatic.com/images?q=tbn:ANd9Gc
4	Third...	20	third	https://encrypted-tbn2.gstatic.com/images?q=tbn:ANd9Gc
5	MOre Bigger	50	biggest	https://encrypted-tbn2.gstatic.com/images?q=tbn:ANd9Gc
6				
7				

Please note that the header line has a pairs - <name>:<type> , it’s really important because it tells which column type it will be in the final sqlite database. In this example code I have only two types: integer or string. The column name has to be without spaces or ANY special characters ONLY a-zA-Z0-9 as well. As you could see, that Description column wasn’t imported into the database due to a header mismatch.

Once the document is ready – it’s time to setup share settings. Make sure that document is shared for everyone, it’s really important as soon as the importer script will download that document as a guest, so the document has to be accessible to anyone who knows the link.

Next, it's required to publish on the web in CSV format link.



Publish to the web

This document is not published to the web.

Make your content visible to anyone by publishing it to the web. You can link to or embed your document. Learn more

Link | Embed

Sheet1 | Comma-separated values (.csv)

Publish

Published content & settings

The final web csv link is the link which is used in SQLiteKitEditor.cs to download data from Google Spreadsheet.

Hints

- https://sqlitebrowser.org this is the best standalone PC and Mac sqlite browser, I guess.
- If you broke an example database from StreamingAssets you could find a copy which I left for this case at the Asset/sqlitekit/CopyToStreamingAssets folder.

Contacts

Fill free email us: orangetree.developers@gmail.com or visit our website https://orangetree.tk