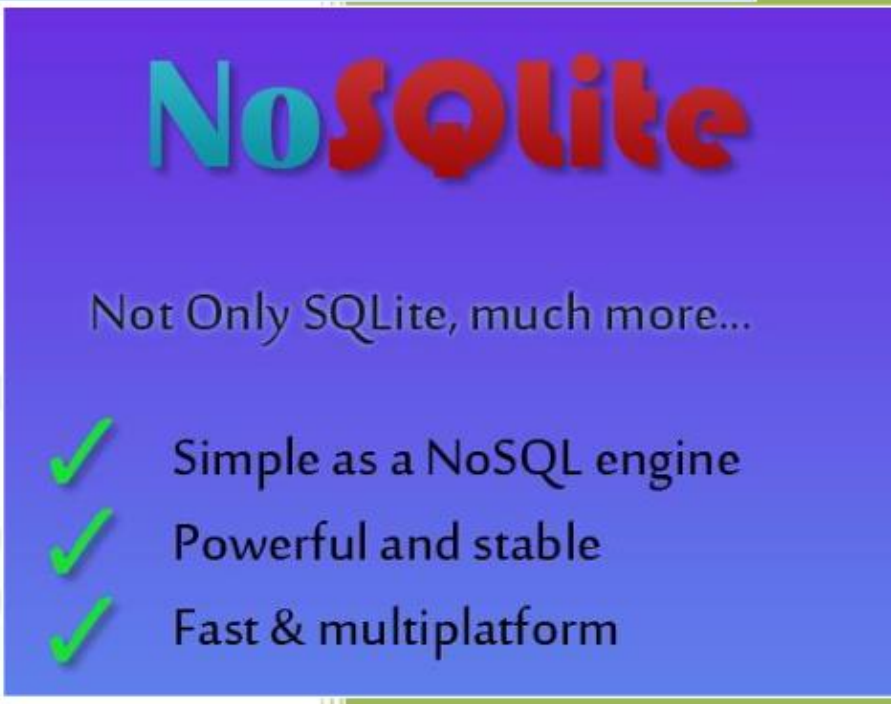
Not Only SQLite

User manual



Index

Introduction

1. Description

NoSQLite is a lightweight SQLite ORM designed for Unity. It simplifies database management, offering an experience similar to working with NoSQL or object-oriented databases. With just a few lines of code, you can easily store and retrieve entire objects.

```
using System.Collections;
using NoSqlite.Orm;
using UnityEngine;

public class Example : MonoBehaviour
{
    public int swordId = 0;

    SwordTable data = null;

    void Start()
    {
        StartCoroutine(RunExample());
    }

    private IEnumerator RunExample()
    {
        // Wait for load
        yield return NoSqliteScript.GetInstance().WaitForLoad();

        // Database handler instance
        var db = NoSqliteScript.Get();

        // Get the object with id == swordId from database
        data = db.Select(new SwordTable { id = swordId });
    }
}
```

2. Features

- Easily saves entire objects into an SQLite database.
- Automatically creates tables for your objects if they don't already exist.
- Supports multiple SQLite databases.
- Allows direct execution of SQL queries—no limitations.
- Uses SQLite 3 format, enabling database edits at any time.

3. Compatibility

- **Android & iOS**: Supported.
- **Windows, Linux, Mac**: Supported.
- **WebGL**: Supported (*).
- **IL2CPP**: Supported.
- **Windows Store & Phone**: Not supported.

(*) On WebGL, the database operates in memory only, so changes are not saved persistently.

4. Directories

- **Doc**: Includes the manual and API documentation in PDF format.
- **Examples:** Provides useful examples to help you get started quickly.
- **Scripts:** Contains scripts for implementing NoSQLite in your project.
- **Sources:** Includes extensions for NoSQLite.
- **Plugins**: Contains the core components of the package.

5. Why SQLite?

SQLite is fast, reliable, cross-platform, and open-source.

"SQLite is a software library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine. SQLite is the most widely deployed SQL database engine in the world. The source code for SQLite is in the public domain." - http://www.sqlite.org

Mapping

Mapping a table is simple—just create a class for your entity and add special attributes to its fields. You can map both simple objects and complex objects (nested objects).

1. Mapping simple objects

```
using System;
using NoSqlite.Attributes;
using NoSqlite.Orm;

public class UserEntity
{
    [PK(Autoincremental = true)]
    public int id;

    public string name;

    public DateTime birthDay;
}
```

Let's take a table called "UserEntity" with 3 columns: id, name, and birthday.

To create columns, simply define fields with the desired type and name. If you want to set a Primary Key, use the PK attribute.

2. Mapping complex objects

```
public class ComplexEntityExample
{
    [PK(true)]
    public int id;                      //<     Autoincremental

    [NotNull]
    public SColor color;                //<     Saving a nested object

    public SColor secundaryCollor;      //<     Saving a nested object
                                        //      (Can be null)

    public Status status;               //<     Enum
}
```

The entity "ComplexEntityExample" contains two objects (color, secundaryColor) serialized as strings, as well as an enum.

To store objects as strings, they must implement the ISerializable interface.

```
#region ISerializable implementation

public string Serialize()
{
    return string.Join(",", new string[] {
    ((int)Math.Round(color.r*255)).ToString(),
    ((int)Math.Round(color.g*255)).ToString(),
    ((int)Math.Round(color.b*255)).ToString(),
    ((int)Math.Round(color.a*255)).ToString()
});
}

public object Deserialize(string str)
{
    if (string.IsNullOrEmpty(str))
        return new SColor(Color.black);

    string[] c = str.Split(',');
    return new SColor(new Color(
        (int.Parse(c[0])) / 255f,
        (int.Parse(c[1])) / 255f,
        (int.Parse(c[2])) / 255f,
        (int.Parse(c[3])) / 255f
    ));
}

#endregion
```

In Serialize method, it should return the entity values represented as a text.

In Deserialize method, it should return an instance of its type using a serialized text.

```
public class SColor : ISerializable
{
    public Color color;

    public SColor(Color c)
    {
        color = c;
    }
    public SColor()
    {
    }

    ISerializable implementation
}
```

3. List of Special Attributes or Decorators

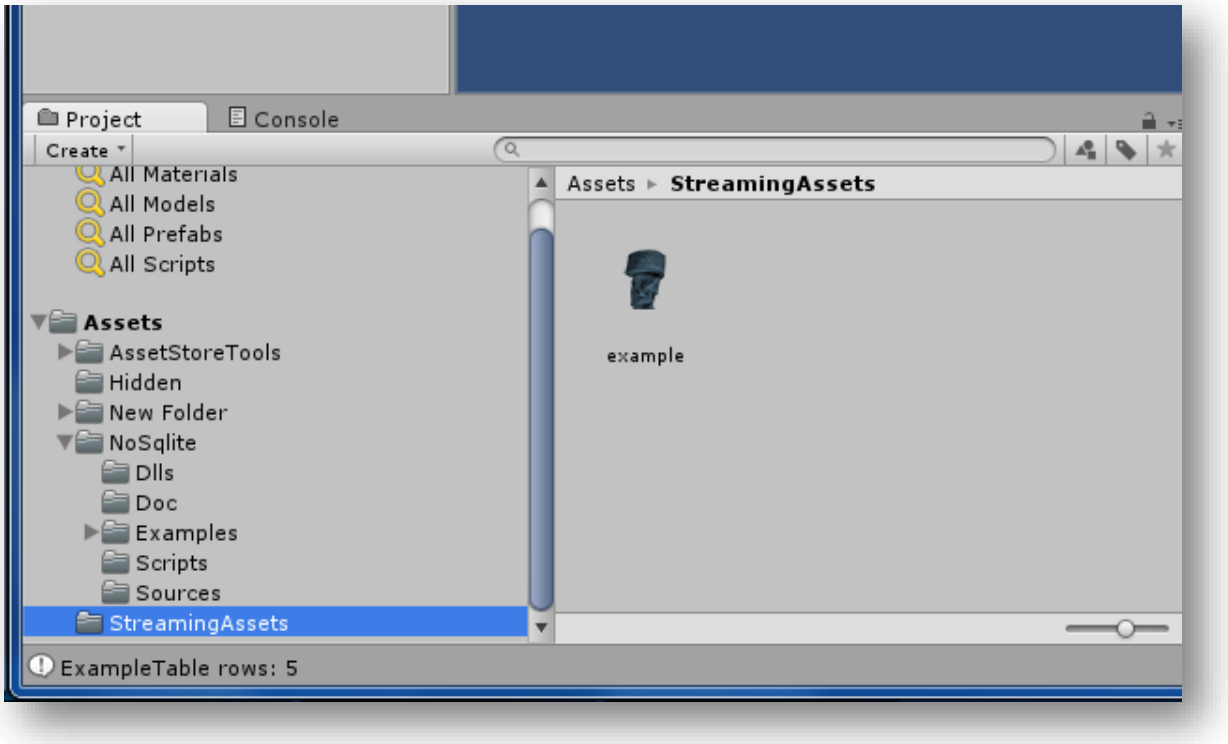
- **PK**: Primary Key attribute.
- **SqlName**: Allows you to customize the name of the column or table in the database. For example, your entity can be called UserEntity, but you can specify that the actual table's name is "UserTable".
- **NotNull**: Specifies that the property or field cannot be null.
- **Unique**: Defines that the property or field is unique.
- **Ignore**: Excludes a specific field or property from being mapped.

Getting started

1. Create your SQLite database (Optional)

If you prefer to create the database manually, you'll first need to create an SQLite 3 database using an editor. There are many free SQLite editors available. My favorite is "SQLite Studio", which you can download here: http://sqlitestudio.pl/.

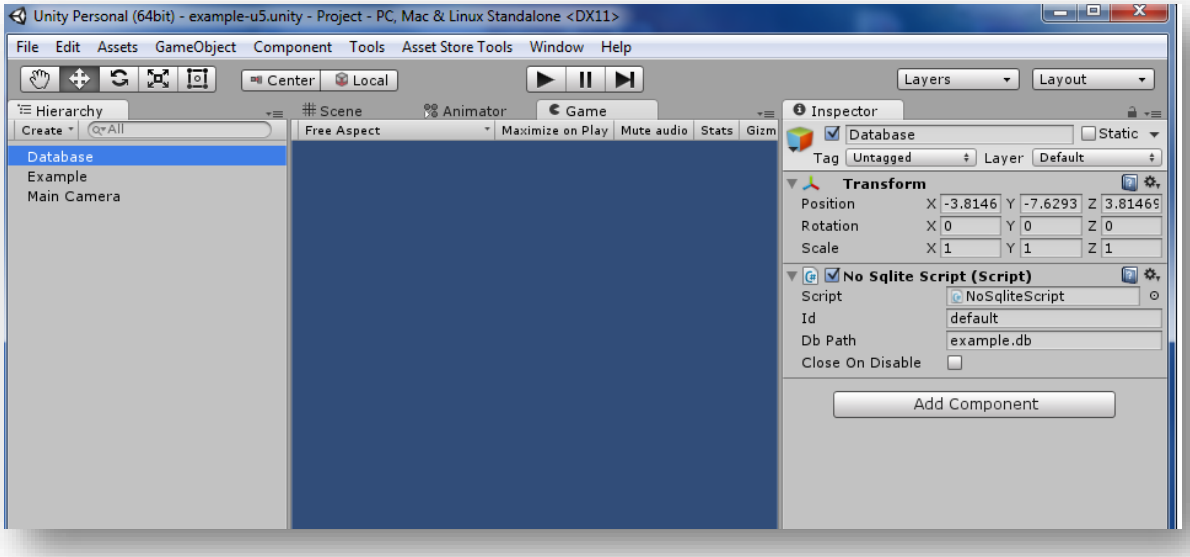
Once the database is created, copy it to the /Assets/StreamingAssets folder (create the directory if it doesn't already exist).



This step is optional because if the table doesn't exist when you store an object, it will be created automatically.

2. Add the Script "NoSqliteScript" to a GameObject

Create a GameObject for the database and then drop the "NoSqliteScript" inside.



The parameter ID is optional. You can use a different Id if you want to manage multiple databases. In other cases you should use default.

3. Create the Entities for the Tables.

Explained in “Mapping” section.

4. Initialization.

If you create your database programmatically, this step is important. You should add in a Start or Awake method of a Unity Script the async calling to the creation of your tables.

```
using System.Collections;
using NoSqlite.Orm;
using UnityEngine;

namespace NoSqlite.Example
{
    public class ExampleStartup : MonoBehaviour
    {
        private void Awake()
        {
            StartCoroutine(CreateTables());
        }

        IEnumerator CreateTables()
        {
            yield return NoSqliteScript.GetInstance().WaitForLoad();

            var db = NoSqliteScript.Get();
            db.CreateTable<ExampleUserTable>(false);
        }
    }
}
```

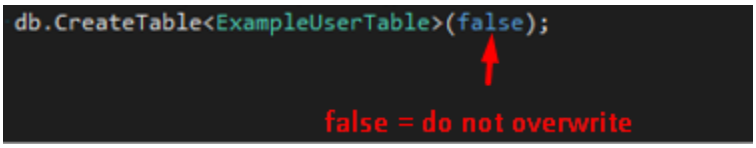
To do this, you should create a method that waits for the async loading of the database.

```
IEnumerator CreateTables()
{
    yield return NoSqliteScript.GetInstance().WaitForLoad();
```

Then you can manipulate all entities of your database. Then next step is get the database handler:

```
var db = NoSqliteScript.Get()
```

And finally, you can call the method CreateTable to create your tables in the database. The Boolean argument that you can see in the example is used to specify if the table must be overwritten. If you use false, you are telling that the table should be generated only if it doesn't exist.



Querying - DML

Get a list of entities

To do this, you just call the List method.

```
var db = NoSqliteScript.Get();
var list = db.List<UserEntity>();
```

If you want to filter the query, you can use the argument “Where”.

```
var list = db.List<UserEntity>(“status=’activated’”);
```

Get a single entity by Id

You should use Select method. It will return null if the entity doesn’t exists, or the instance if it exists.

```
var entity = db.Select<UserEntity>(new UserEntity() { id = 145 });
```

Insert an entity

To insert an entity, you should use the Insert method. If the entity has an autoincremental column, you can get the generated value using the result of the method.

```
var unsavedEntity = new UserEntity();
unsavedEntity.name = “Dexter Morgan”;

var newEntity = db.Insert(unsavedEntity);
Debug.Log(newEntity.id);
```

Update an entity

To update an entity, you should use the Update method. If you want, you can specify the affected columns to optimize the query.

```
var entity = db.Select<UserEntity>(new UserEntity() { id = 1 });
entity.age += 1;

db.Update(entity);
// Or better:
// db.Update(entity, "age");
```

Insert or update an entity

If you want to insert or update an entity, you should use the Save method.

Delete an entity

To update an entity, you should use the Delete method. You can use DeleteMany method to delete multiple entities.

Querying - Misc functions

Count rows

Counts the rows of a table.

```
var number = db.Count<UserEntity>();
```

You can optionally specify a filter.

```
var numberOfYounger = db.Count<UserEntity>("age < 18");
```

Existence of a row

Checks if exists a row.

```
if (db.Exists( new UserEntity() { Id = 100 } )) {
      Debug.Log("The user exists!");
}
```

Querying - DDL

Create a table

To create a table, you should use the CreateTable method. It accepts a Boolean value as parameter, that allows you to re-generate the table (true) or skip the creation if it already exists (false).

```
db.CreateTable<UserEntity>(false);
```

Existence of a table

To check if a table exists, you should use the ExistsTable method.

```
if (db.ExistsTable<UserEntity>()) {
    Debug.Log("The table exists!");
}
```

Drop a table

To drop a table, you should use the DropTable method.

```
db.DropTable<UserEntity>();
```

Querying - Custom

Execute a query without a result

To execute a query without expecting a result, you should use the method ExecuteNoQuery.

```
db.ExecuteNoQuery(“”);
string viewSelect
    = "SELECT * FROM Users u JOIN Messages m ON(m.IdUser = u.Id)";
db.ExecuteNoQuery("CREATE VIEW IF NOT EXISTS JoinView AS " + viewSelect);
```

Execute multiple queries without a result

To execute a query without expecting a result, you should use the method ExecuteNoQueries.

Execute a query getting the first value of the first row

To get the first value of the first row of a query result, you should use the method ExecuteQueryScalar.

Execute a query getting multiple rows

To get all rows from a custom query, you should use the method ExecuteQuery.

```
var result = db.ExecuteQuery("SELECT * FROM Users " +
    "u JOIN UserHasFriends f ON (f.Id = u.Id)").ToList();
```

API

You can find the full API here: Updated API docs website.

Support

Need a feature or support? We're here to help!

For real-time support, join our Discord: https://discord.com/invite/dqGdSpVcAc

Forja Games website: https://forjagames.com

Developed by Forja Games