Data Toolkit Quick Tutorial

1. A brief introduction

The data toolkit focuses on solving the problems of data form and accessibility for data-driven develop. It includes the following functions to provide a simple one-stop service for game development:

- Database CRUD operation utils based on SQLite.
- Common local file I/O interface in a cross-platform way.
- Analysis and generator of standard CSV format.
- Provide Unity JSON serialization extension for dictionary.

The data toolkit will focus on engine changes and provide long-term support to ensure the project stability of users. And more functions will be also updated for free in the subsequent versions.

If you want to keep up with updates, please follow our GitHub official website. You can also find the latest version of this document on our online documents page.

2. Prerequisite of the Data Toolkit package

By downloading and then importing the Data Toolkit package from the unity package manager, you have completed its installation just like importing other packages.

This package is applicable to all versions after Unity 2020, and is also applicable to Android/iOS/Windows and Mac OS platforms.

Please note that if you want to use the **SQLite Toolkit** in the package, please check the following one additional settings to ensure that it behaves correctly at runtime:

That setting item is located in Project Settings - Player - Other Settings - Scripting - Backend, Choose **Mono** for SQLite Toolkit.

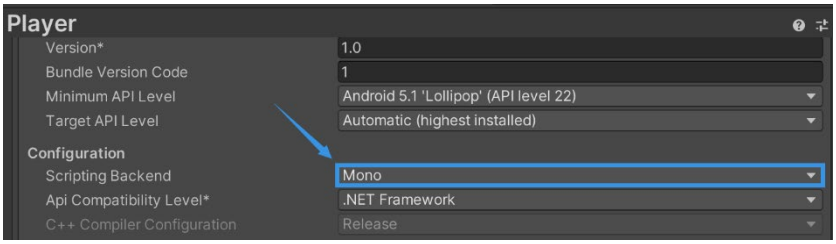

Figure 1. Additional Setting

If you don't need to use the functionality in the SQLite toolkit, you don't need to change any options, and you don't have to worry about throwing exceptions or crash -- it's safe all the time.

3. SQLite Toolkit tutorial

SQLite Toolkit provides a type for abstraction of SQL data connection: SQLiteDatabase. To write your own SQL logic code, just use the following two namespaces:

```
using Arcspark.Core;
using Mono.Data.Sqlite;
```

3.1 Create SQLDatabase

Create an SQLDatabase object using SQL connection parameters to start operating the corresponding SQLite database.

A connection string is used to specify how to connect to the database.

On Windows, iOS and Mac OS platforms, you can access database files In StreamingAssets path and Persistent data path. But on Android platform, Persistent data path is the only data source for SQLiteDatabase.

For a simple example, if you want to connect a database file, and its path is Assets/StreamingAssets/test.db, you should use the following code to build a correct connection:

```
string connectionStr = Application.streamingAssetsPath+ "/test.db";
SQLiteDatabase db = new SQLiteDatabase(connectionStr);
```

Similarly, if you want to access a database file “test.db” that located under persistent data path (in Windows systems, it is usually located under “C:\Users\user name\AppData\LocalLow\your company name\your game name” folder), you should use the following code to build a correct connection:

```
string connectionStr = Application.persistentDataPath + "/test.db";
SQLiteDatabase db = new SQLiteDatabase(connectionStr);
```

Don't forget to use CloseConnection function to close the database link when completing the transaction operation on the database:

```
db.CloseConnection();
```

3.1 CRUD

Use the InsertValues method to complete the field insertion of a table:

```
db.InsertValues("TableName",
    new string[] { "10000", "'String Data'", "'More String Data'" });
```

Note that the number of value lists should be the same as the number of columns in the data table(the excess part will be ignored).

The syntax for values here always needs to comply with the SQL syntax specification. For example, when we want to insert a string, we need to wrap it in a pair of single quotes.

Use the DeleteValues method to complete the field insertion of a table:

```
db.DeleteValues("TableName", "Amount < 0");
```

A conditional SQL clause as the second parameter to indicate what data should be deleted.