XAM160

SQLite and Mobile Data

Download class materials from
university.xamarin.com

Microsoft

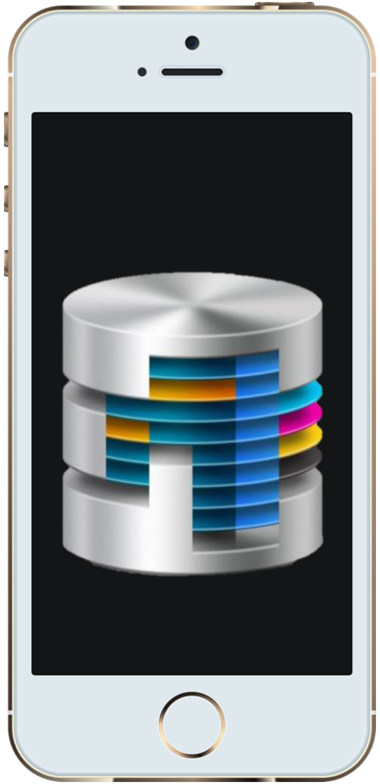
Xamarin University

Information in this document is subject to change without notice. The example companies, organizations, products, people, and events depicted herein are fictitious. No association with any real company, organization, product, person or event is intended or should be inferred. Complying with all applicable copyright laws is the responsibility of the user.

Microsoft or Xamarin may have patents, patent applications, trademarked, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any license agreement from Microsoft or Xamarin, the furnishing of this document does not give you any license to these patents, trademarks, or other intellectual property.

Objectives

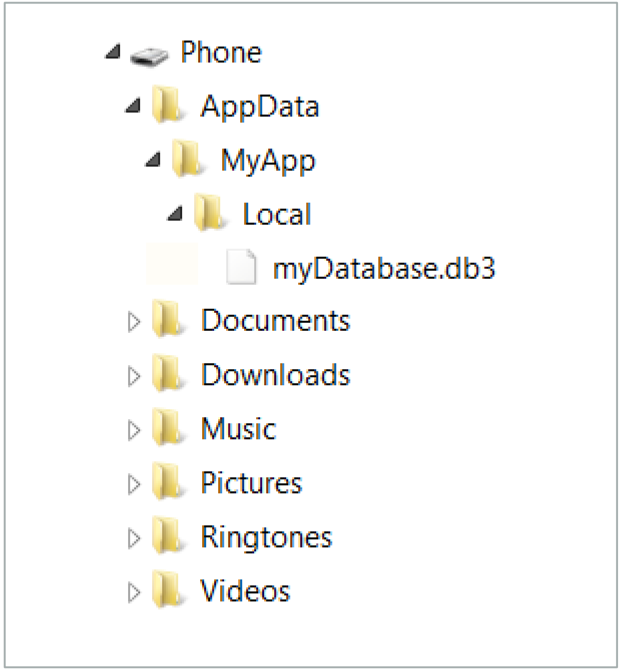
1. Choose a data storage strategy
2. Store data locally with SQLite
3. Use SQLite asynchronously



Choose a data storage strategy

Tasks

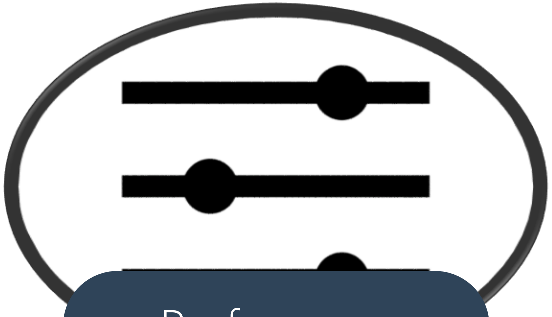
1. Understand the data storage options available to your app
2. Select a storage location
3. Use the correct storage path for each platform



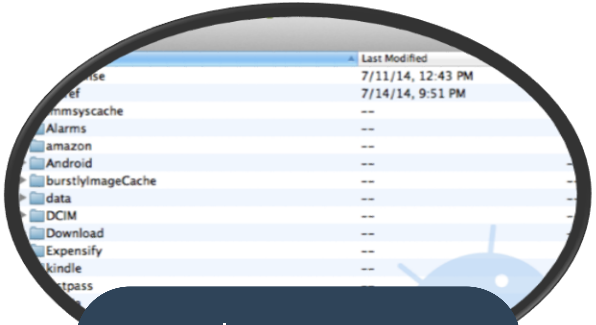
- Phone
  - AppData
    - MyApp
      - Local
        - myDatabase.db3
  - Documents
  - Downloads
  - Music
  - Pictures
  - Ringtones
  - Videos

Data Storage Options

- When storing local information, your app has several options to choose from



Preferences



	Last Modified
ise	7/11/14, 12:43 PM
ef	7/14/14, 9:51 PM
mmsyscache	--
Alarms	--
amazon	--
Android	--
burstlyImageCache	--
data	--
DCIM	--
Download	--
Expensify	--
kindle	--
tpass	--

File System

Database

Which approach makes the most sense for the type of data you are working with?

Preferences

- iOS, Android and UWP support storage of app-specific settings as simple key-value pairs
- Useful to store app configuration, user preferences, and other customization tweaks the user can control

Cross-platform settings support

- Nuget package **Xam.Plugins.Settings** enables platform-agnostic storage

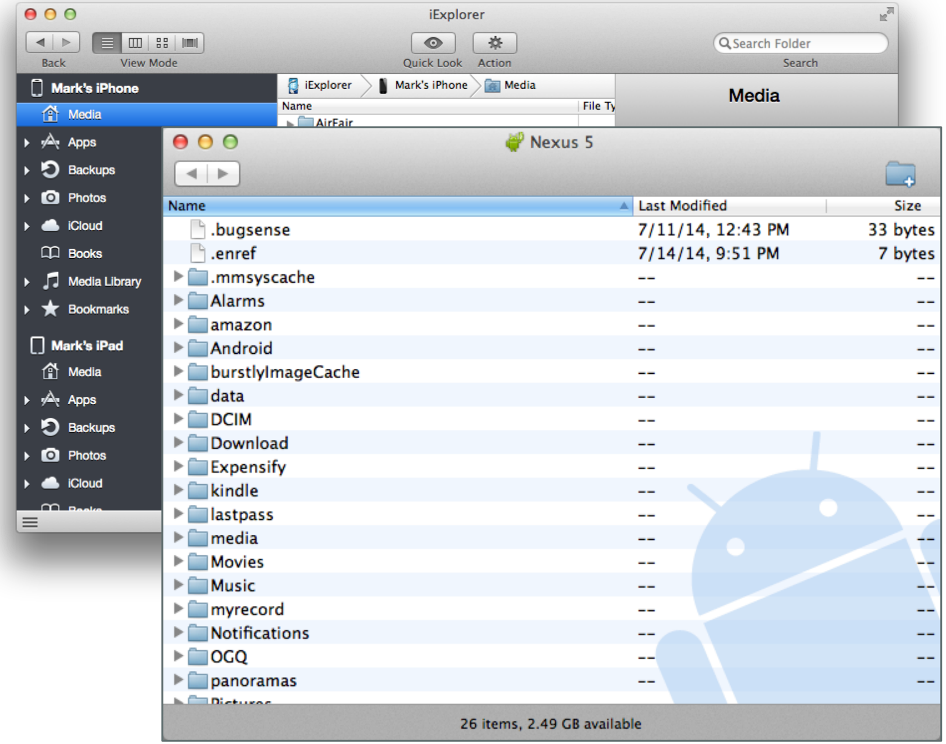
```
public static class Settings
{
   const string NameKey = "userName";

   public static string Name
   {
      get { return CrossSettings.Current
                .GetValueOrDefault<string>(NameKey, ""); }
      set { CrossSettings.Current
                .AddOrUpdateValue<string>(NameKey, value); }
    }
}
```

Plugin provides simple get/set API that does the persistent storage for you

The file system

- Devices have persistent file systems to store settings, applications, data, etc.

- File system structure and content vary based on the operating system



Name	Last Modified	Size
.bugsense	7/11/14, 12:43 PM	33 bytes
.enref	7/14/14, 9:51 PM	7 bytes
.mmsyscache	--	--
Alarms	--	--
amazon	--	--
Android	--	--
burstlyImageCache	--	--
data	--	--
DCIM	--	--
Download	--	--
Expensify	--	--
kindle	--	--
lastpass	--	--
media	--	--
Movies	--	--
Music	--	--
myrecord	--	--
Notifications	--	--
OGQ	--	--
panoramas	--	--

26 items, 2.49 GB available

Working with Files and Folders

Can work directly with files and folders using classes in **System.IO** namespace

```
using System.IO;
...
public IEnumerable<Todo> LoadTodoTasks(string filename)
{
    StreamReader reader = File.OpenText(filename);
    ...
}
```

Familiar classes such as **File**, **Directory**, and **StreamReader** are available in your platform-specific projects

File Formats

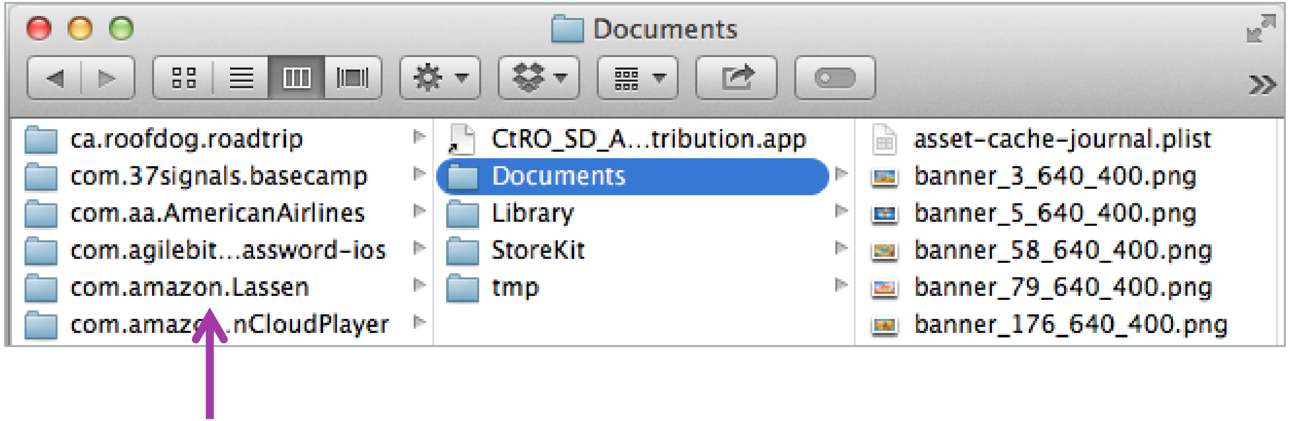
- Each platform support text, binary, XML and JSON formats – use the one that makes sense for your data style

```
using System.Xml.Linq;
...
public IEnumerable<string> LoadTasks(string filename)
{
    XDocument doc = XDocument.Load(filename);
    return (from item in doc.Root.Descendants("todo")
       select (string) item.Attribute("PartNumber"));
}
```

LINQ to XML makes working with XML easy… compared to native APIs

The app sandbox

- Your application is given a dedicated folder, called the app folder or sandbox, on the file system which contains app-specific content



Each iOS application has a folder, which contains sub-folders, which, in turn, contains your data and assets you create

File locations

- The recommended location for your data file differs across platforms

Android	`<AppHome>/files`
iOS	`<AppHome>/Library/[subdirectory]`
UWP	`<AppHome>\LocalState`

These locations are common, but other options are available (e.g. Android has a **database** folder)

Folder path [.NET]

- Can use .NET APIs to get the full path to the application folder

```
// <AppHome>/files for Android
string path = Environment.GetFolderPath(Environment.SpecialFolder.Personal);
```

```
// <AppHome>/Documents for iOS
string docFolder = Environment.GetFolderPath(Environment.SpecialFolder.Personal);
// to meet Apple’s iCloud terms, content that is not generated by the user
// should be placed in the /Library folder or a subdirectory inside it
string libFolder = System.IO.Path.Combine(docFolder,"..", "Library");
```

```
// <AppHome>\LocalState on Windows
string path = Windows.Storage.ApplicationData.Current.LocalFolder.Path;
```

Working with native APIs

- Can use platform-specific APIs to access unique features, for example to ensure internal files are not backed up to iCloud on iOS

```
void AddSkipBackupAttribute(string filename)
{
   if (File.Exists(filename)) {
     // Do not backup to iCloud
     NSFileManager.SetSkipBackupAttribute(filename, true);
   }
}
```

Can tell iOS to *not* backup a file in the documents folder to iCloud

Flash Quiz

Flash Quiz

① To retrieve the path to the root folder of the local data store within a UWP app, use:

a) `Environment.GetFolderPath(Environment.SpecialFolder.Personal);`

b) `Environment.GetFolderPath(ApplicationData.Current.LocalFolder);`

c) `Windows.Storage.ApplicationData.Current.LocalFolder.Path;`

Flash Quiz

① To retrieve the path to the root folder of the local data store within a UWP app, use:

a) `Environment.GetFolderPath(Environment.SpecialFolder.Personal);`

b) `Environment.GetFolderPath(ApplicationData.Current.LocalFolder);`

c) **`Windows.Storage.ApplicationData.Current.LocalFolder.Path;`**

Flash Quiz

② The recommended location for your data file is the same across platforms.

a) True

b) False

Flash Quiz

② The recommended location for your data file is the same across platforms.

a) True

b) False

Flash Quiz

③ In your Xamarin.iOS app, you should always place content generated by the app within the **Documents** folder.

a) True

b) False

Flash Quiz

③ In your Xamarin.iOS app, you should always place content generated by the app within the **Documents** folder.

a) True

b) [✓ False]

Individual Exercise

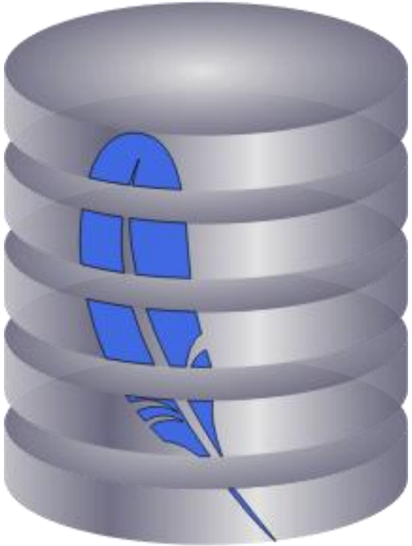
Select location to store local data

Store data locally with SQLite

Xamarin
University

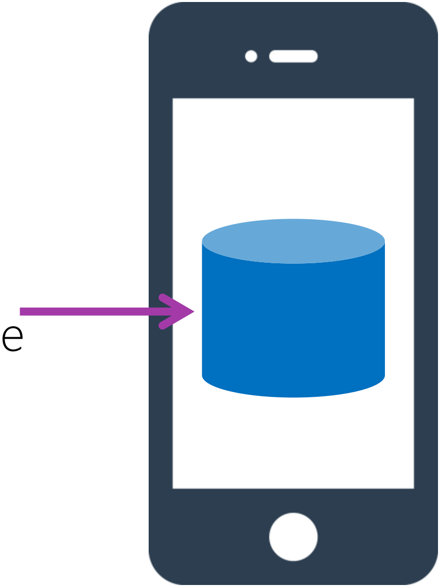
Tasks

1. Add a SQLite.Net to your app
2. Define SQLite table schema using attributes
3. Create and connect to a SQLite database
4. Perform CRUD operations against a SQLite database using SQLite.Net



What is SQLite?

- SQLite is a lightweight local database that has become the de-facto industry standard for mobile apps



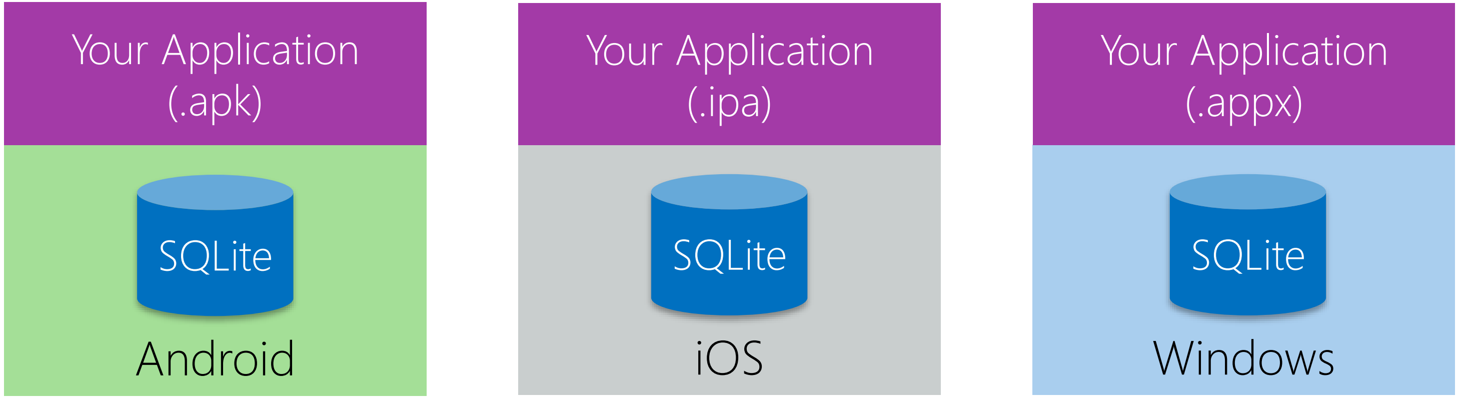
runs in-process (no server) and uses local file system for storage

SQLite

open-source and
maintained by sqlite.org

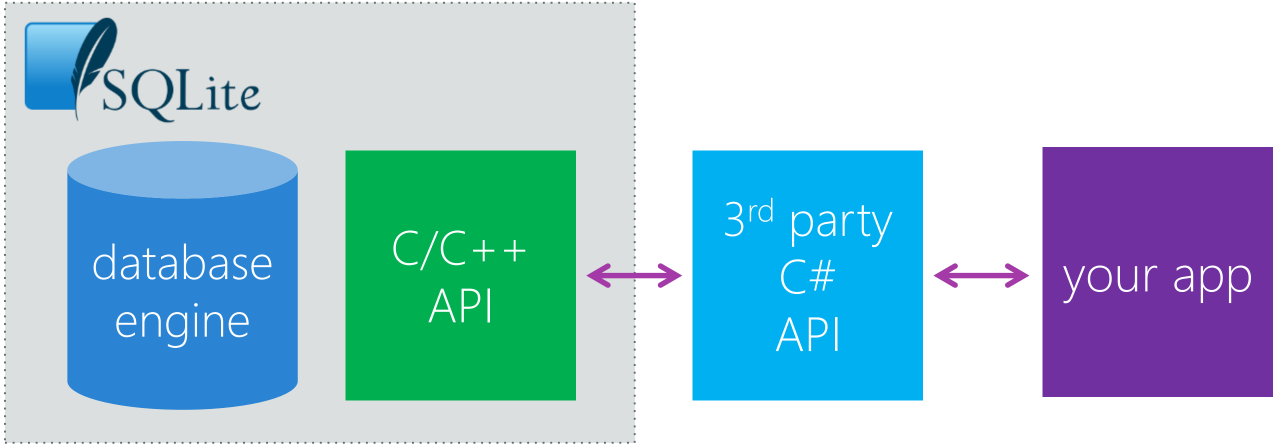
SQLite packaging

- SQLite engine is built-in to Android and iOS, and UWP



Accessing the SQLite API

- SQLite engine exposes C/C++ API which is then accessed by .NET through a C# wrapper



Available C# APIs

- There are multiple C# APIs available from different vendors, most work with Xamarin so you can choose based on features or coding style

SQLite mono	Microsoft Open Technologies Portable Class Library for SQLite https://sqlitepcl.codeplex.com/	Entity Framework Core (open source) GitHub SQLite.NET (open source)
ADO.NET (included with Xamarin)	Thin wrapper over C++ API	Object-Relational Mapper (ORM)

Choosing the right API

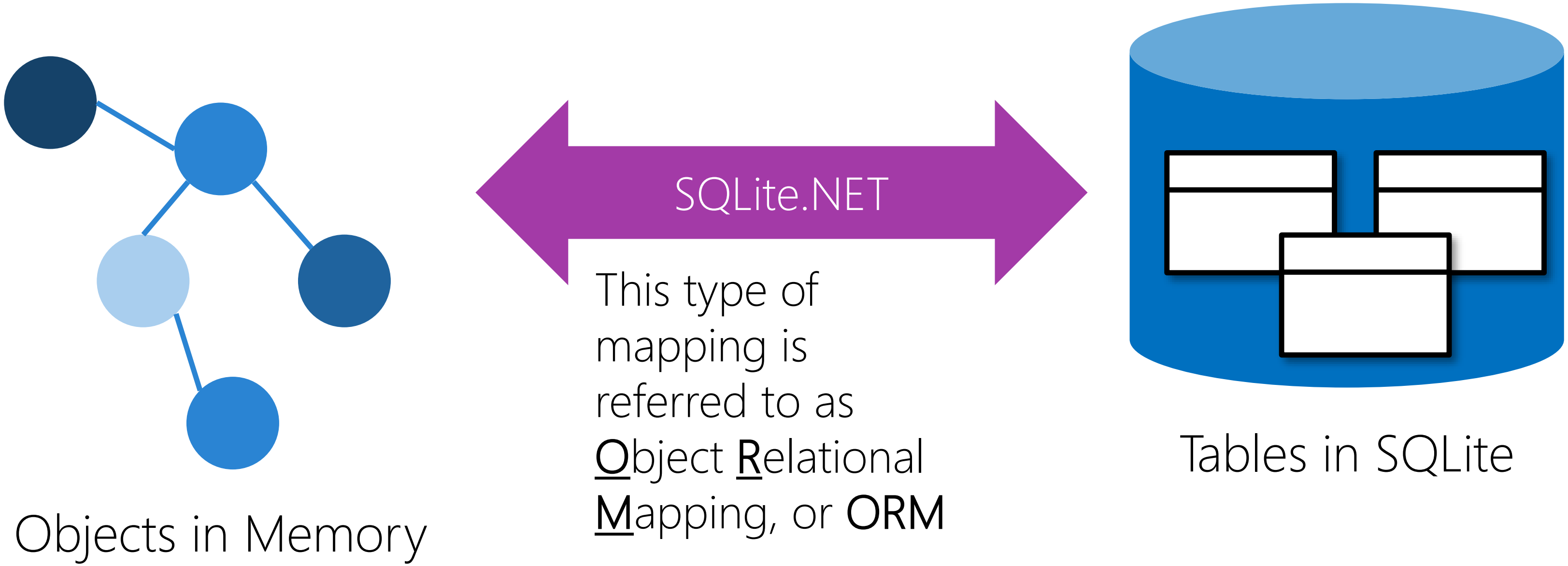
- Pros and Cons to each approach, pick the one best suited for your data and access needs

	ADO.NET	MS PCL for SQLite	SQLite.NET
Access Style	SQL + DataReader	SQL + rows / columns	LINQ + objects
Supported Platforms	iOS, Android, UWP	iOS, Android	iOS, Android, UWP
Maturity	Stable / Legacy	Stable	Stable

In this course, we'll be working with SQLite.NET

SQLite.NET

- SQLite.NET provides a mechanism to map classes to tables

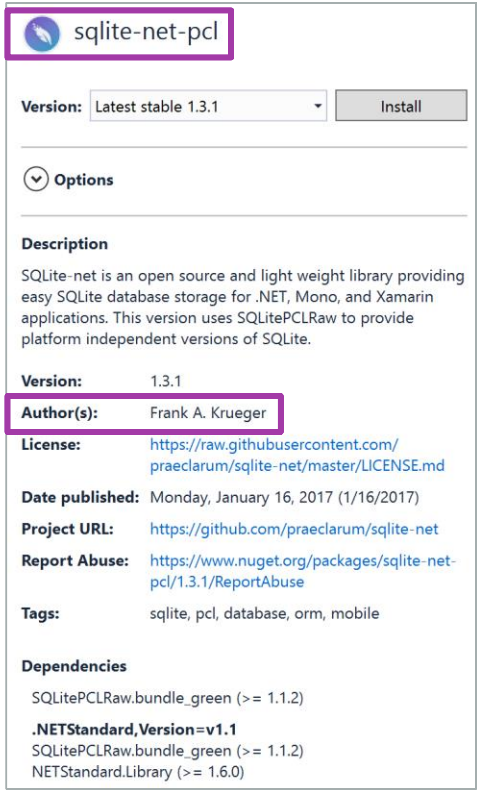


This type of mapping is referred to as Object Relational Mapping, or ORM

Adding support for SQLite.NET

- SQLite-Net is shipped as a Nuget component, adds different DLLs to the project based on the project type

- Several different implementations out there – make sure to use **SQLite-net PCL** by Frank Krueger



sqlite-net-pcl

Version: Latest stable 1.3.1

Description

SQLite-net is an open source and light weight library providing easy SQLite database storage for .NET, Mono, and Xamarin applications. This version uses SQLitePCLRaw to provide platform independent versions of SQLite.

Version:	1.3.1
Author(s):	Frank A. Krueger
License:	https://raw.githubusercontent.com/praeclarum/sqlite-net/master/LICENSE.md
Date published:	Monday, January 16, 2017 (1/16/2017)
Project URL:	https://github.com/praeclarum/sqlite-net
Report Abuse:	https://www.nuget.org/packages/sqlite-net-pcl/1.3.1/ReportAbuse
Tags:	sqlite, pcl, database, orm, mobile

Dependencies

SQLitePCLRaw.bundle_green (>= 1.1.2)

.NETStandard,Version=v1.1
SQLitePCLRaw.bundle_green (>= 1.1.2)
NETStandard.Library (>= 1.6.0)

Xamarin University

What is installed?

❖ Three components are added to the project with **SQLite-Net PCL**

Sqlite-net.pcl

- Contains support necessary to define data entities
- Contains sync and async APIs to interact with the SQLite engine

SQLitePCL. bundle_green

- Includes compiled copies of the native SQLite library for platforms that need it

SQLitePCL.raw

- Contains C# wrapper to access the SQLite engine API
- Used by the PCL Can use API directly for low-level operations

Group Exercise

Adding SQLite.NET to your projects

Connect to a SQLite database

- The first step required to access the database is to create a **`SQLiteConnection`** – this is the object that *talks* to the local database

```
using SQLite;
...

string filename = ...

SQLiteConnection conn = new SQLiteConnection(filename);
```

Must pass in the filename representing the database

Database storage file

- SQLite stores the database in a local file which must be placed in a writable folder path that is platform-specific

Common folder used for database files	Location
`Path.Combine(personalFolder, "databases");`	`<AppHome>/databases`
`Path.Combine(personalFolder, "..","Library");`	`<AppHome>/Library`
`ApplicationData.Current.LocalFolder.Path`	`<AppHome>\LocalState`

SQLiteConnection

- Connection has two constructors and a few optional parameters
  - prefer **true** for **storeDateTimeAsTicks**
  - use the **openFlags** to control the **Read|Write|Sharing** flags

```
public SQLiteConnection(
   string databasePath
   SQLiteOpenFlags openFlags,
   bool storeDateTimeAsTicks = true)
```

Connection Management

- Caching connections is a balance between memory and performance
- Better to use same connection for a set of operations vs. opening new one each time
- Call **`Dispose`** or **`Close`** when finished with it to cleanup

```
public static class MyConnectionFactory
{
  static SQLiteConnection connection;

  public static SQLiteConnection Instance
  {
    get { return connection ??
          (connection = CreateConnection());
    }
  }
  ...
}
```

common to hold shared connection in a shared singleton and create as necessary

Mapping classes to tables

- Database schema is defined through attributes applied to the class and public properties

```
[Table("people")]
public class Person
{
   // PrimaryKey is typically numeric
   [PrimaryKey, AutoIncrement, Column("_id")]
   public int Id { get; set; }

   [MaxLength(250), Unique]
   public string Name { get; set; }
   ...
}
```

Identifies which table this class is mapped to

Identify the primary key and column name

specify column metadata necessary to map property to column

Very common to add your *own logic* into these classes to supplement the data

Common attributes

- SQLite.NET includes several attributes to fully define the mapping between an object and the relational table holding the data

- No attribute support for foreign keys in the library, however you can manage the relationships in code

```
[Table(name)]
[Column(name)]
[PrimaryKey]
[AutoIncrement]
[Indexed]
[MaxLength(value)]
[Unique]
[NotNull]
[Ignore]
[Collation]
```

What if I want real FK relationships?

- SQLite.NET Extensions project adds attributes and extension methods for foreign key relationships and cascade operations

```
public class Students_Classes
{
  [ForeignKey(typeof(Students))]
  public int StudentFId { get; set; }

  [ForeignKey(typeof(Classes))]
  public int ClassFId { get; set; }
```

```
public class Students
{
  [PrimaryKey, AutoIncrement]
  public int Id { get; set; }
  public string Name { get; set; }

  [ManyToMany(typeof(Students_Classes))]
  public List<Classes> Classes { get; set; }
}

public class Classes
{
  [PrimaryKey, AutoIncrement]
  public int Id { get; set; }
  public string Title { get; set; }

  [ManyToMany(typeof(Students_Classes))]
  public List<Students> Students { get; set; }
```

This is currently based on an *older version* of SQLite-Net, might need to be changed and recompiled to be used with the latest version.

Supported data types

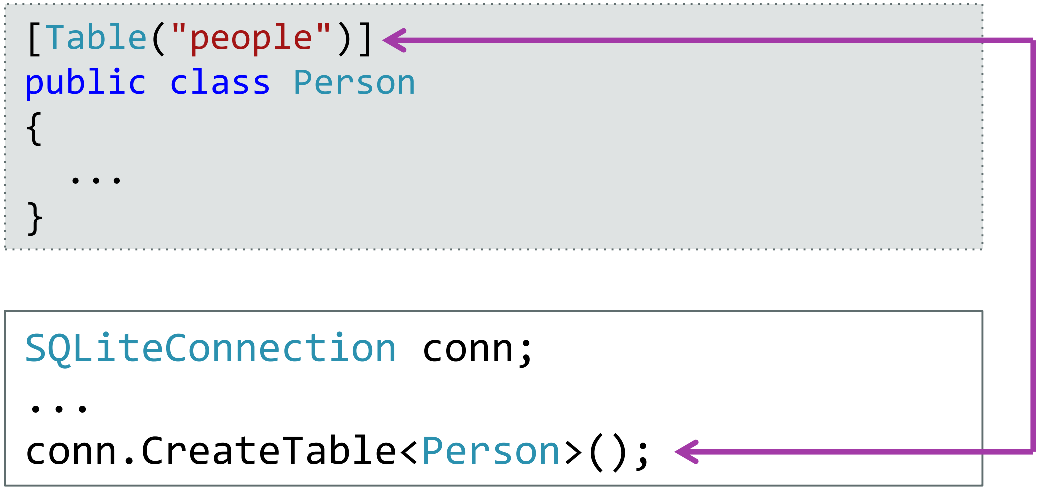
C# type	SQLite type
`int,long`	integer, bigint
`bool`	integer (1 = true)
`enum`	integer
`float`	real
`double`	real
`decimal`	real
`string, GUID`	varchar
`DateTime`	numeric or text
`byte[]`	blob

- SQLite.NET maps intrinsic .NET types to appropriate SQLite data types

- Mismatches result in a runtime exception if the table already exists

Creating a Table

- SQLite.NET supports either **creating** a new table, or **updating** the schema for an existing table using your defined class mappings



```
[Table("people")]
public class Person
{
   ...
}
```

```
SQLiteConnection conn;
...
conn.CreateTable<Person>();
```

Pass the annotated entity class to SQLite.NET and it creates/updates the table based on the attributes applied to the class and it's properties

Performing operations

❖ Once the table has been created, you can perform CRUD operations on it in a strongly-typed fashion using your entity classes

```
SQLiteConnection conn;
...
public int AddNewPerson(Person person)
{
    int result = conn.Insert(person);
    return result;
}
```

Insert, Update and Delete operations all require a primary key be defined

Returns the number of rows that were affected by the operation, in this case, inserted

Retrieving records

- SQLite.NET makes it easy to retrieve all the records from the table through the **`Table<T>()`** method, this returns a **`TableQuery<T>`**

```
SQLiteConnection conn;
...
public List<Person> GetAllPeople()
{
    List<Person> people = conn.Table<Person>().ToList();
    return people;
}
```

Add **`.ToList()`** from **`System.Linq`** to execute the query and return all the rows from the **people** table

Warning: this is a dangerous query if you are not certain how many people are in the db!

Individual Exercise

Access a SQLite database with the SQLite.NET

Xamarin University

What is LINQ?

- Language-INtegrated Query is a built-in feature of C# and VB.NET that allows for standardized data queries across different sources

LINQ to SQL

<xml/>

LINQ to XML

LINQ to Amazon

SQLite.NET query capabilities

- `TableQuery<T>` exposes common LINQ (Language-Integrated Query) methods which can be used to query the data

```
Where
Take
Skip
OrderBy
OrderByDescending
ThenBy
ElementAt
First
FirstOrDefault
ThenByDescending
Count
```

These methods enable the extension method syntax as well as the LINQ C# syntax!

Example: Filter results with LINQ

- LINQ statements translate the C# expression to a SQL query

```
SQLiteConnection conn;
...
public Person GetByName(string name)
{
   var person = from p in conn.Table<Person>()
                where p.Name == name
                select p;
   return person.SingleOrDefault();
}
```

```
SELECT Id, name
FROM people
WHERE name = 'Joe Smith'
```

Filter is applied directly to the SQL query issued to the DB

Note: SQLite.NET actually generates *parameterized queries* for better security

Where clause

- SQLite.NET supports converting common string and **`List<T>`** methods into proper SQL syntax for more efficient queries

`Contains`	Looks for a specific piece of text in the column
`StartsWith`	Column value must start with text
`EndsWith`	Column value must end with text
`Equals`	Direct comparison
`ToLower`	Lowercase the text
`ToUpper`	Uppercase the text

Example: Where clause

- Using **StartsWith** changes the resultant query to use **LIKE**

```
var records = from p in conn.Table<Person>()
              where p.Name.StartsWith("Joe")
              select p;
```

```
SELECT * FROM [people] WHERE ([Name] LIKE ('Joe' || '%'))
```

Flash Quiz

Flash Quiz

① Using SQLite.NET, which of the following creates a connection to a SQLite database:

a) `new SQLConnection(dbPath);`

b) `new SQLiteConnection(dbPath);`

c) `new SQLiteConnection(targetPlatform, dbPath);`

d) `new SQLConnection(targetPlatform, dbPath);`

Flash Quiz

① Using SQLite.NET, which of the following creates a connection to a SQLite database:

a) `new SQLConnection(dbPath);`

b) `new SQLiteConnection(dbPath);`

c) `new SQLiteConnection(targetPlatform, dbPath);`

d) `new SQLConnection(targetPlatform, dbPath);`

Flash Quiz

② Mapping a table to a class is done using the _______ attribute.

a) `[Table]`

b) `[Entity]`

c) `[TableEntity]`

d) None of the above

Flash Quiz

② Mapping a table to a class is done using the _______ attribute.

a) **[Table]**

b) [Entity]

c) [TableEntity]

d) None of the above

Flash Quiz

③ LINQ to SQLite.NET:

a) Allows you to perform queries against a SQLite database

b) Translates language integrated queries to SQL queries behind the scenes

c) Returns a filtered result set back to the application

d) All of the above

Flash Quiz

③ LINQ to SQLite.NET:

a) Allows you to perform queries against a SQLite database

b) Translates language integrated queries to SQL queries behind the scenes

c) Returns a filtered result set back to the application

d) All of the above

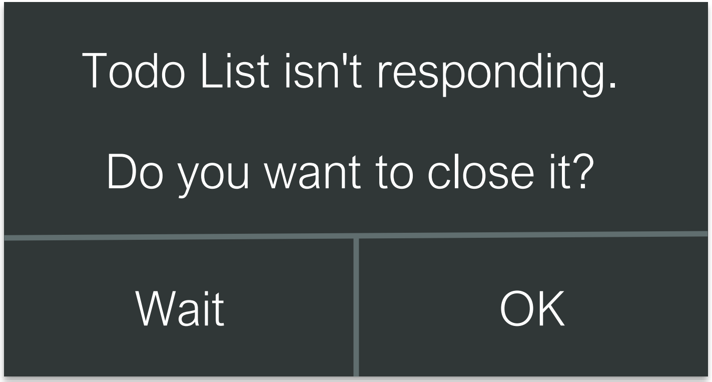
Use SQLite asynchronously

Tasks

1. Create an async capable database connection
2. Perform CRUD operations asynchronously

Performing async queries

- Reading and writing data into our database on the UI thread is a synchronous I/O operation which can block the UI thread



Todo List isn't responding.

Do you want to close it?

How do you think your users will answer this dialog?

What kind of rating will your app get in the App Store?

Instead, we want to perform our I/O *asynchronously* independent from the UI thread

Dealing with concurrency

- A SQLite connection can only have one outstanding operation at a time – if you want to use a connection with multiple threads, you must guard the access with a **`lock`**

This is common code to ensure the database does not get corrupted by a writer

```
SQLiteConnection dbConn;
object guard;
...
public int AddNewPerson(Person person)
{
    lock (guard)
    {
        return dbConn.Insert(person);
    }
}
```

Serialized thread access

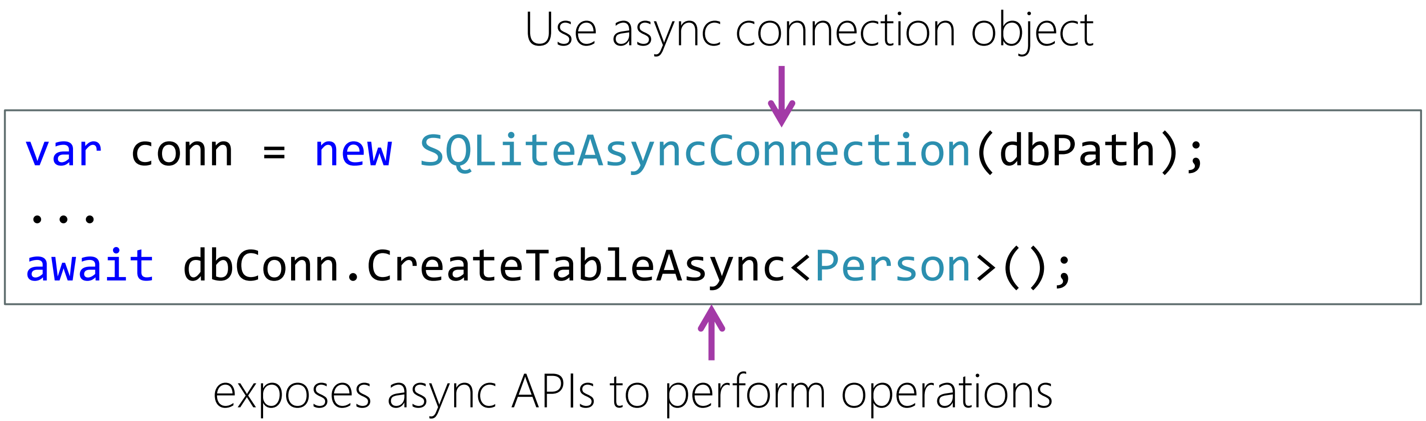
- Can configure SQLite to use a Serialized mode where it will serialize thread access on your behalf

```
SQLiteConnection.SetConfig(SQLiteConnection.Serialized);
```

Call the static **SetConfig** method to set the threading mode

Asynchronous execution

❖ SQLite.NET includes an asynchronous API through the `SQLiteAsyncConnection` class



Use async connection object

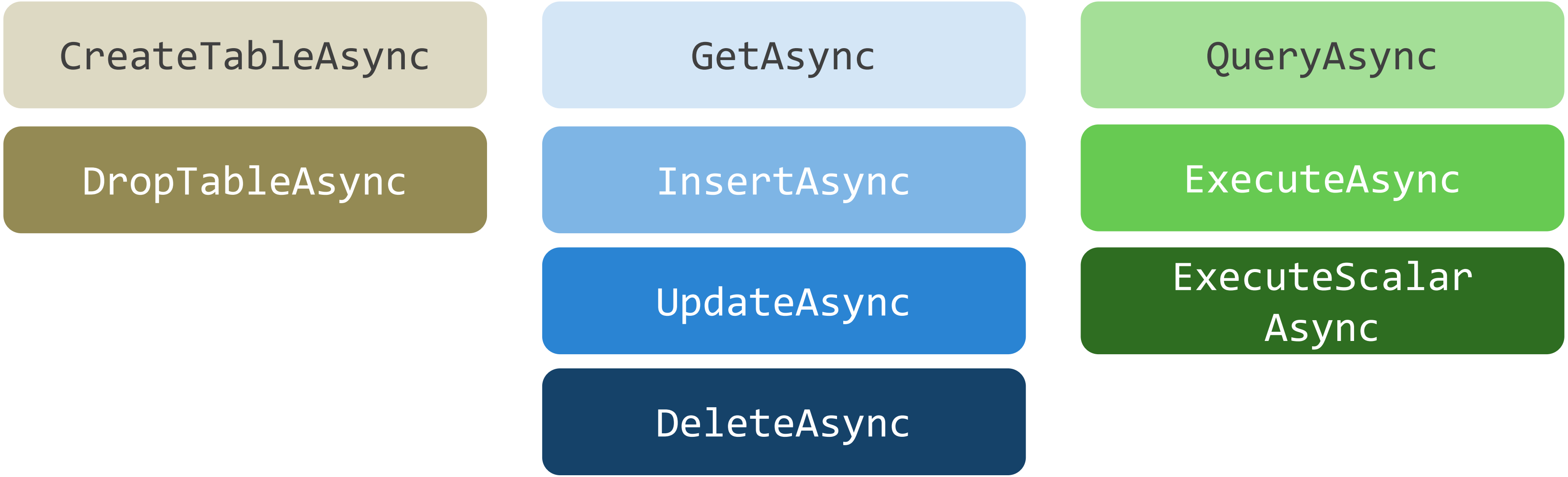
```
var conn = new SQLiteAsyncConnection(dbPath);
...
await dbConn.CreateTableAsync<Person>();
```

exposes async APIs to perform operations

This approach has no need of external locking – it's already provided in the library

Async operations

- **`SQLiteAsyncConnection`** exposes the same operations as the synchronous counterpart – but Task-based for background usage



CreateTableAsync

DropTableAsync

GetAsync

InsertAsync

UpdateAsync

DeleteAsync

QueryAsync

ExecuteAsync

ExecuteScalarAsync

Retrieving records asynchronously

- Use **ToListAsync** to turn **Table<T>** into an asynchronous call and use the **async** and **await** keywords in C# to marshal control back to the UI thread once the query has completed

```
SQLiteAsyncConnection dbConn;
ObservableCollection<Person> peopleList;  // Bound to UI
...
public async Task AddAllPeopleAsync()
{
   List<Person> people = await dbConn.Table<Person>().ToListAsync();
   // Must be on UI thread here!
   foreach (var p in people)
      peopleList.Add(p);
}
```

Query is executed on background thread and control *returns to UI thread* once query has completed

Working with Transactions

- Use **`RunInTransaction[Async]`** to execute a block of statements in a transaction

```
SQLiteAsyncConnection conn;
...
public async int UpdatePeople(Person newPerson, Person updatedPerson)
{
    int count = 0;
    await conn.RunInTransactionAsync(conn => {
        count += iconn.Insert(newPerson);
        count += iconn.Update(updatedPerson);
    });
    return count;
}
```

Must pass an **`Action`** that accepts a **`SQLiteAsyncConnection`** and do all your transactional work in the block

SQLite-net also exposes methods to create, commit and rollback transactions

Dropping down to SQLite

- SQLite.NET has several methods which you can use to execute direct SQL statements and queries with parameters

Method	Description
`ExecuteAsync`	Execute SQL statement, returns affected row count
`ExecuteScalarAsync`	Execute a statement which returns a scalar
`QueryAsync`	Issue an immediate SQL query, returns table mapping
`GetAsync`	Returns the first object that matches a predicate
`FindAsync`	Returns the object with the matching primary key

Example: retrieving a scalar

- Can use **ExecuteScalarAsync** to return a single value

```
SQLiteAsyncConnection conn;
...
int startingId = 10;

double count = await conn.ExecuteScalarAsync<double>(
      "SELECT MAX(age) FROM [people] WHERE Id > ?",
      startingId);
```

use '?' for placeholders in the queries, optional parameter list will then fill in each placeholder by position

Example: performing a SQL query

- Can use **`Query<T>`** to execute a raw SQL query and map it to a set of objects – this is most useful when pulling relationships

```
SQLiteAsyncConnection conn;
Class xam160 = ...
...
List<Student> students = await conn.QueryAsync<Student>(
   "SELECT * FROM [students] WHERE id in " +
   "(SELECT sid FROM students_classes WHERE cid=?)",
      xam160.Id);
```

Grab all the students enrolled in a specific class based on a table relationship

Processing in the background

❖ Often, the data you query can be retrieved and processed in the background; consider using `ConfigureAwait(false)` to stay on background thread after `await` finishes

```
SQLiteAsyncConnection dbConn;
...
public async Task TakeAttendanceAsync(Class class)
{
   var students = await dbConn.QueryAsync<Student>("...", class.Id)
                              .ConfigureAwait(false);
   // Background thread continues execution here .. NOT ON UI THREAD!
   foreach (var s in students) { ... }
}
```

Individual Exercise

Access SQLite database using asynchronous methods

Xamarin University

Thank You!

Please complete the class survey in your profile:
university.xamarin.com/profile

Microsoft