Part 2 - Configuration

In order to use SQLite in your Xamarin iOS or Android application you will need to determine the correct file location for your database file.

Database File Path

Regardless of which data access method you use, you must create a database file before data can be stored with SQLite. Depending on what platform you are targeting the file location will be different. For iOS and Android you can use Environment class to construct a valid path, as shown in the following code snippet:

```
string dbPath = Path.Combine ( Environment.GetFolderPath (Environment.SpecialFolder.Personal), "database.db3"); // dbPath contains a valid file path for the database file to be stored
```

There are other things to take into consideration when deciding where to store the database file. On iOS you may want the database to backed-up automatically (or not); and on Android you can choose whether to use internal or external storage. These factors may require slightly different code on each platform, but the above example will work on both iOS and Android and is used for the samples in this document.

If you wish to use a different location on each platform you can use a compiler directive as shown to generate a different path for each platform:

```
var sqliteFilename = "MyDatabase.db3"; #if __ANDROID__ // Just use whatever directory SpecialFolder.Personal returns string libraryPath = Environment.GetFolderPath(Environment.SpecialFolder.Personal); ; #else // we need to put in /Library/ on iOS5.1 to meet Apple's iCloud terms // (they don't want non-user-generated data in Documents) string documentsPath = Environment.GetFolderPath (Environment.SpecialFolder.Personal); // Documents folder string libraryPath = Path.Combine (documentsPath, "..", "Library"); // Library folder instead #endif var path = Path.Combine (libraryPath, sqliteFilename);
```

Refer to the Working with the File System article for more information on what file locations to use on iOS. For hints on using the file system in Android, check out the Browse Files recipe. See the Building Cross Platform Applications document for more information on using compiler directives to write code specific to each platform.

Threading

You should not use the same SQLite database connection across multiple threads. Be careful to open, use and then close any connections you create on the same thread.

To ensure that your code is not attempting to access the SQLite database from multiple threads at the same time, manually take a lock whenever you are going to access the database, like this:

```
object locker = new object(); // class level private field // rest of class code lock (locker){ // Do your query or insert here }
```

All database access (reads, writes, updates, etc) should be wrapped with the same lock. Care must be taken to avoid a deadlock situation by ensuring that the work inside the lock clause is kept simple and does not call out to other methods that may also take a lock!