# Persistent Storage

Two techniques to store data persistently in a cross platform project are discussed below. There are other approaches such as File IO which would also work.

# Serialization

For simple data persistence we will be using Xamarin.Essentials which allows access to the SecureStorage mechanism for saving string key value pairs.

In order to use the library, you’ll need a using Xamarin.Essentials; line in the file that will access the storage.

Setup in Android should already be done, in MainActivity.cs there should already be a call to initialize Xamarin Essentials.

# Accessing SecureStorage

There are two relevant commands, one to write and one to read. Each of the calls is asynchronous, so if you need to do something immediately with the result you can use the await keyword.

string result = await SecureStorage.GetAsync("SomeKeyName");

await SecureStorage.SetAsync("SomeKeyName", "SomeValue");

If nothing is found by Get for the passed in key a result of null will be returned.

# SQLite Database Access

We will be using Object Relational Mapping (ORM) to map objects to database tables.

Database access on both Android and iOS is supported through a library called sqlite-pcl.

Some references: <https://developer.xamarin.com/guides/xamarin-forms/application-fundamentals/databases/>

Some examples for SQLite code for ORM: <https://developer.xamarin.com/guides/android/application_fundamentals/data/part_3_using_sqlite_orm/>

## Installing SQLite PCL

Go to Tools, Nuget Packet Manager -> Package Manager Console. At the PM> Prompt, Type:

Install-Package sqlite-net-pcl -Version 1.4.118

Right click the solution, Manage Nuget Packages for Solution, click the installed tab, choose the sqlite-net-pcl package and check both other projects and choose install.

Both projects need a FileHelper.cs file which will allow a platform specific approach to locating the actual database file. These two classes will implement an interface that we will define in our shared project called IFileHelper. This interface can go into the same file as the App class:

public interface IFileHelper

{

string GetLocalFilePath(string filename);

}

Add a FileHelper.cs to both Android and iOS.

Android:

using System.IO;

using Xamarin.Forms;

// modify for your namespace ///

[assembly: Dependency(typeof(XamBookDatabase.Droid.FileHelper))]

namespace XamBookDatabase.Droid

{

public class FileHelper : IFileHelper

{

public string GetLocalFilePath(string filename)

{

string path = System.Environment.GetFolderPath(System.Environment.SpecialFolder.Personal);

return Path.Combine(path, filename);

}

}

}

iOS:

using System.IO;

using Xamarin.Forms;

// modify for your namespace ///

[assembly: Dependency(typeof(XamBookDatabase.iOS.FileHelper))]

namespace XamBookDatabase.iOS

{

public class FileHelper : IFileHelper

{

public string GetLocalFilePath(string filename)

{

string docFolder = Environment.GetFolderPath(Environment.SpecialFolder.Personal);

string libFolder = Path.Combine(docFolder, "..", "Library", "Databases");

if (!Directory.Exists(libFolder))

{

Directory.CreateDirectory(libFolder);

}

return Path.Combine(libFolder, filename);

}

}

}

The two classes above will handle locating the files on each platform.

## Database Helper Class

See our project for an implementation. The basics are that this class should:

* Define an SQLiteConnection object with a path to the database
* Call the CreateTable method as needed to create tables (the method will do nothing if the table already exists)
* Provide methods for accessing the database, usually by calling either the Table<>() or Query<>() methods.
* Provide methods for inserting, updating, and deleting – usually using the built in Insert, Update, and Delete methods provided by the SQLite library.

## Defining ORM Classes

Defining classes is straight forward, add a using SQLite; statement, then ensure there are public properties and that the primary key is called ID with the proper annotations, for example:

[PrimaryKey, AutoIncrement]

public int ID { get; set; }

## Connecting the App Class to the Database

Create a couple of static properties that will store a reference to the database and initialize it:

static Our\_Helper\_Database database;

public static Our\_Helper\_Database Database

{

get

{

if (database == null)

{

database = new Our\_Helper\_Database(DependencyService

.Get<IFileHelper>().GetLocalFilePath("OurFile.db3"));

}

return database;

}

}

Then in your constructor you can call database = Database; which will trigger the get and have the database initialize. From there, the database reference can be used to call any of the defined methods in the helper class, such as queries or DDL commands like Insert.

Note that this property will be accessible from anywhere in the application as App.Database. We will use the database variable to do queries and so on. We’ll be calling the “Dependency Service” to find out which platform’s file location to use.