Lesson 4: Handling Data

INTRODUCTION

There are several considerations that a developer should keep in mind for handling data. First, what types of data need to be stored? Second, determine where framework classes can be used versus classes with custom functionality. Thirdly, a developer has to look at how the data will be supplied to the user interface.

LESSON OBJECTIVES

By the end of this lesson, the student will be able to:

1. Define data persistence.
2. Identify and differentiate among the different methods discussed for handling data persistence on the iPhone.
3. Identify the subfolders in the iOS applications folder and purpose of each.
4. Explain the purpose of an application's sandbox and how this purpose impacts the application's design and functionality.
5. Differentiate between single-file persistence and multiple-file persistence.
6. Discuss two protocols used in archiving data persistence and identify which of these protocols is required and which is optional.
7. Discuss the advantages and disadvantages of using SQLlite3.
8. Discuss the advantages and disadvantages of using Core Data.

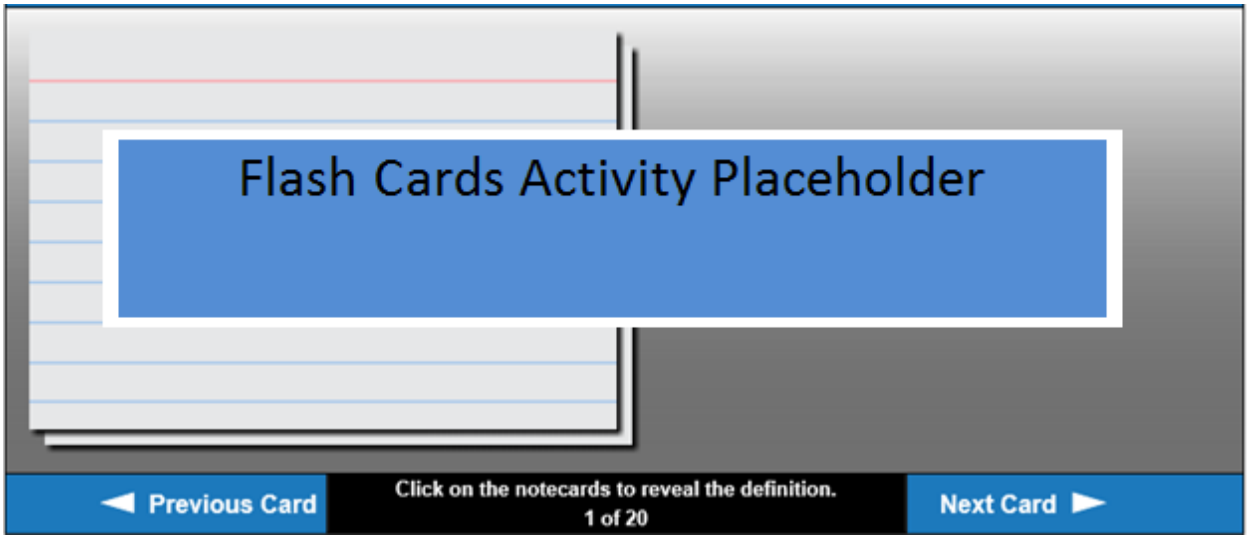
LEARNING SEQUENCE

Required Reading	Read the following: • Lesson 4: Handling Data
Resources	View the following: • *iOS App Dev 105: The Foundation Classes – 5. NSArray* (5:20) • *iOS App Dev 105: The Foundation Classes – 9. NSCoding* (10:44) • *Create a MySQL/SQLite Database on a MAC (to use in iPhone App)* (6:37) • *Xcode Database – How to create, read, update and delete (CRUD)* (9:35) • *Xcode 5 Tutorial: iOS 7 Core Data – Getting Started* (16:19)
Assignments	Complete the following: 1. Practice Example - Handling Data

	2. Quiz 4

KEY TERMS

As you read your lesson, pay close attention to the key terms and phrases listed throughout the lesson. These terms and concepts are important to your understanding of the information provided in the lesson.



INSTRUCTION

The iOS App Folders

The following folders are part of the iOS app:

- **AppName.app** contains the app and its related resource (icons, graphics, and property files). .app is an extension that turns a regular directory into an **application bundle.**
- **Documents** stores user app data files. This is data that the app generates.
- **Library** stores files that are not user data files. This directory should not be used for user data files.
- **tmp** stores temporary files that do not need to be kept between launches of the app. The app should remove files from the directory when they are no longer needed. This folder is not backed up by iTunes, but it is also not deleted unless it is explicitly deleted with code.

An iOS app may create additional directories in Documents, Library, or tmp.

The Sandbox

Each app installs in its own **sandbox** directory. A sandbox limits an app's access to files, preferences, and network resources, for example. The sandbox directory acts as the home for the app and its data. Therefore, an app cannot access files in directories outside of its home directory. The path to the Documents directory must be found in order to save data for the app.

Data Persistence

Data persistence is the ability to save data and have it available at a later date. There are four common methods used for data persistence:

1. **Property lists**; also known as **plists**, this is an XML format introduced by Apple for iOS. A plist is a nested list of key-value pairs containing common data types such as strings, numbers, arrays, or dictionaries
2. **Object archiving** takes data created by a user in an app to store for later retrieval
3. **SQLite** uses sqlite queries to store and manage data within an app
4. **Core Data** provides a visual interface to build the data model. Core data maintains the consistency of relationships among objects in the app.

Strategies for Saving Files

There are two strategies for saving files.

- **Single-file Persistence**: there is only one file, so the entire contents are rewritten whenever a save is done. The problem here is that all of the app data must be loaded into memory and rewritten even if there is a small change. This could pose a problem if the developer is trying to manage more than a couple of megabytes of data.
- **Multiple-file Persistence**: uses different files to store data. Only the data that is needed is loaded. This makes it easier to flush memory if the low memory warning is received. The biggest problem with this strategy is that it is more complicated to handle, since it requires more coding.

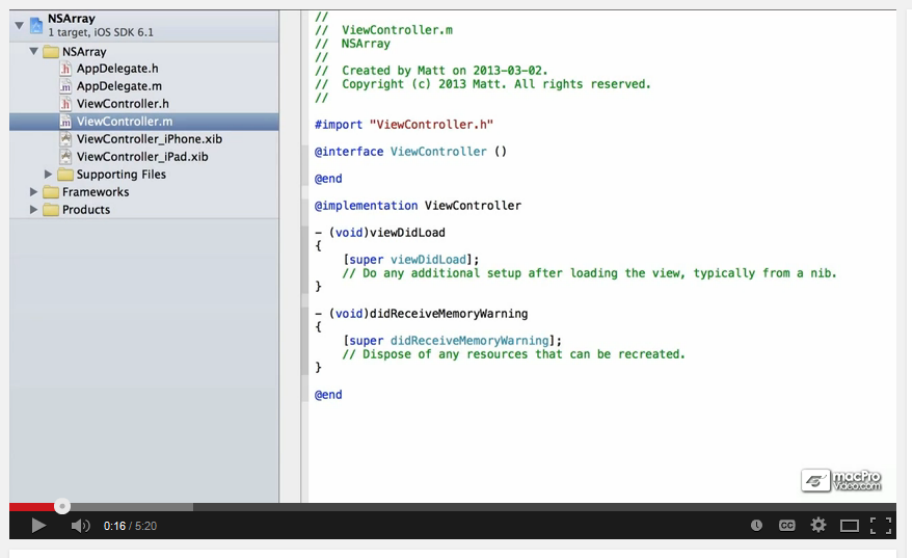
Property Lists

One of the simplest ways to store data is to create a property list. This requires a serialized object, an object that has been converted into a stream of bytes. Only classes that are collection classes can be stored in a property list in Objective-C and include the following:

- NSArray
- NSMutableArray
- NSDictionary
- NSMutableDictionary
- NSData
- NSMutableData
- NSString
- NSMutableString
- NSNumber
- NSDate

The benefits of property lists are that they are easy to build and edit and they are useful with a small quantity of data. The downside is that property lists do not work with custom objects and it is not easy to do calculations.

Watch the video, *iOS App Dev 105: The Foundation Classes – 5. NSArray* (5:20). This video uses the collection class, NSArray to create an array with objects.

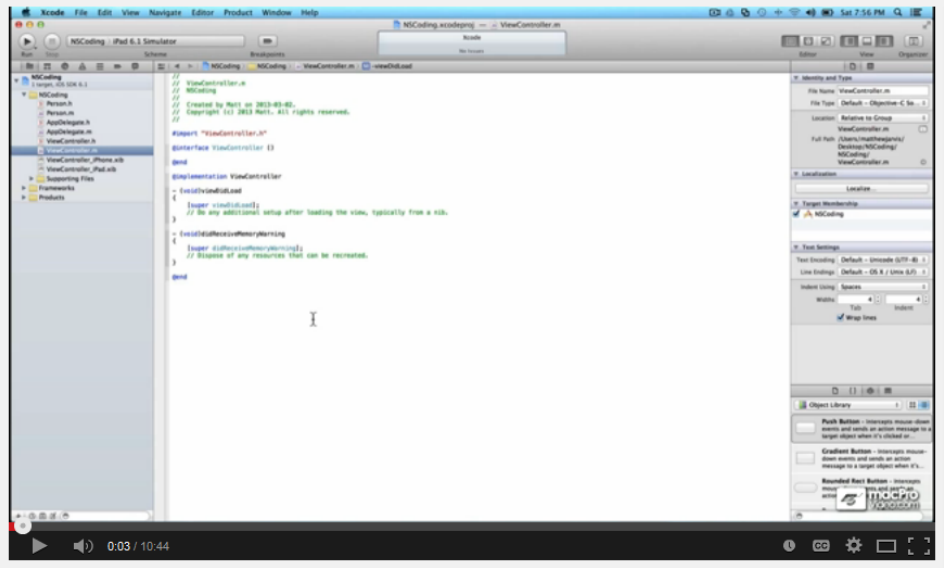


Object Archiving

A custom class can be created. In this case, object archiving would be used. Classes must be saved and conform to the **NSCoding** and **NSCopying** protocol. The NSCoding protocol encodes the object into an archive and creates a new object when decoding the archive. NSCopying protocol creates a new instance of the same class with all of the same properties and values, i.e., a duplicate.

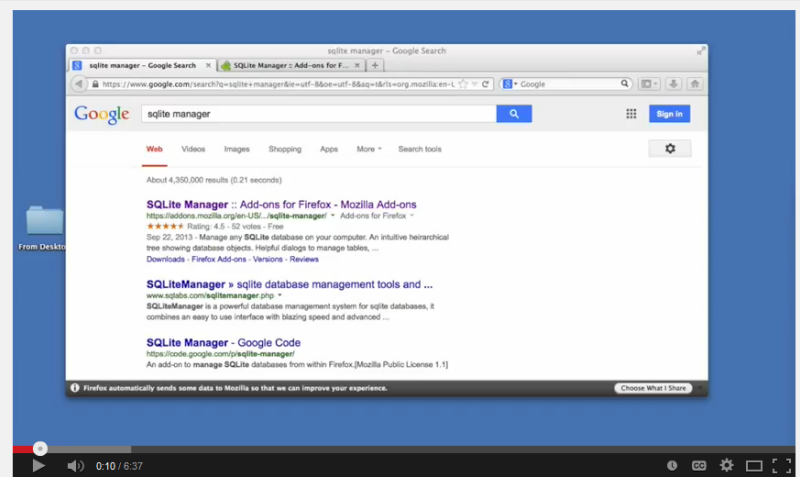
Object Archiving sets an array of objects that can be archived by archiving (saving as a backup) the array itself. In a property list, the developer would have to loop through all of the data and handle each piece of data. Object Archiving takes all of the data that has been entered by the user, stores it in an array object and then archives it to a binary property file. When the application is reloaded, the data is un-archived. The array object is recreated and the restored data is extracted from the array object and presented to the user.

Watch the video, *iOS App Dev 105: The Foundation Classes – 9. NSCoding* (10:44). This video implements the NSCoding protocol to declare the methods that a class must implement so that instances of that class can be encoded and decoded. This capability provides the basis for archiving for custom objects created using classes.



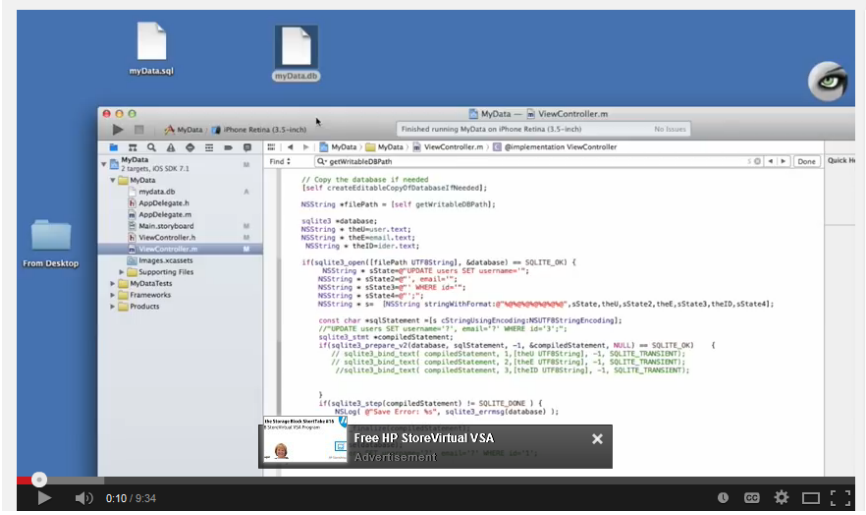
SQLite

SQLite is an iPhone embedded SQL database that uses the Structured Query Language. When dealing with SQLite, the developer must open or create the database, find or add a table, add data, and close the database. View the video, Create a MySQL/SQLite Database on a MAC (to use in iPhone App) (6:37), which is part one of two parts. The first part shows the viewer how to create an SQLite database using SQLite Manager, a Firefox add-on.



Data can be inserted into a database by using **bind variables** which is a placeholder in an SQL query. The value is entered at runtime. A link to the database must also be set up when a new project is started.

SQLite can handle large amounts of data and can deal with calculations and queries, but it is complicated and one has to deal with SQL statements. Watch part two, *Xcode Database – How to create, read, update and delete (CRUD)* (9:35), to add the database created in part one to an Xcode project which will add, update view and delete database information.



Core Data

Core Data was added is iOS3. It is the best solution for handling complex data and calculations, especially for those that are unfamiliar with SQL. It provides a visual interface to build the data model. Core Data still uses SQLite, the Core Data interface and model editor. A data model editor is a program that graphically creates a data model. There are also Objective-C wrappers available that are pre-built so not much coding is required.

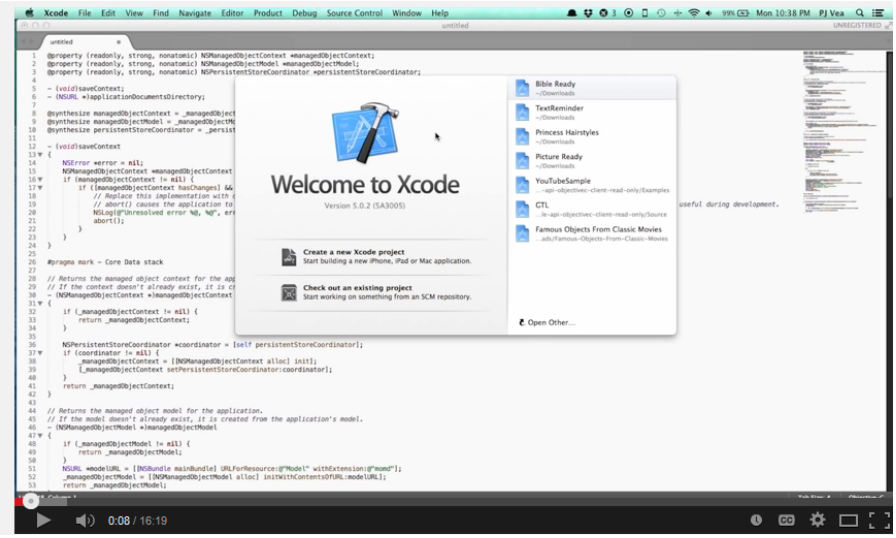
Core Data uses database terminology. Instead of referring to classes, the Core Data term for a description of an object is **entities,** something about data can be stored. Entities have three types of properties:

- **Attributes** – a variable instance
- **Relationships** – how entities relate to each other (one-to-many or many-to-one)
- **Fetched Properties** – similar to a relationship, but only the "related" entity will load if it is actually used.

Key-value coding sets properties and/or retrieves values. Each value has a unique key.

A **persistent store** is the Core Data stored in SQLite. Core Data classes do all of the work so there is no SQL coding required. A developer does not usually work with the persistent store directly. Rather, the **managed object** context keeps track of what has changed since the last save. A managed object is an instance which is created at runtime. Managed Object Context (or **context**) manages access to persistent store and keeps track of what has changed since the last save. A **fetch request** is used to retrieve a managed object from persistent store. With a fetch request, data is being pulled from where it is actually stored.

Watch the video, *Xcode 5 Tutorial: iOS 7 Core Data – Getting Started* (16:19) to become familiar with the basics of Core Data.



SUMMARY

This lesson introduced handling data in an iOS app. Storing data, determining where framework classes can be used versus classes with custom functions, and determining how the data will be supplied to the user interface were discussed.

ASSIGNMENTS

1. Practice Example - Handling Data
2. Quiz 4