

Data Storage

Mobile Application Development in iOS

School of EECS

Washington State University

Instructor: Larry Holder

Outline

- Already seen
 - UserDefaults
 - iCloud
- File I/O (review)
- Database support: Core Data, SQLite
- Data backup to iCloud
- Data protection

File I/O

- Class objects to be written must inherit from `NSObject` and `NSCoding`
- Class must implement
 - `required init(coder aDecoder: NSCoder)`
 - `func encode(with aCoder: NSCoder)`

File I/O

```
class MarioCharacter: NSObject, NSCoder {
    var name: String
    var health: Int

    init (_ name: String, _ health: Int) {
        self.name = name
        self.health = health
    }

    required init(coder aDecoder: NSCoder) {
        name = aDecoder.decodeObject(forKey: "name") as! String
        health = aDecoder.decodeInteger(forKey: "health")
    }

    func encode(with aCoder: NSCoder) {
        aCoder.encode(name, forKey: "name")
        aCoder.encode(health, forKey: "health")
    }
}
```

File I/O

- Get document directory
- Create URL to file
- Use `NSKeyedArchiver` to write
- Use `NSKeyedUnarchiver` to read

File I/O

```
let marioCharactersFile = "MarioCharactersFile"

func readFromFile () { // call from initial view controller's viewDidLoad
    let fileDir = FileManager.default.urls(for: .documentDirectory,
                                           in: .userDomainMask).first!
    let fileURL = fileDir.appendingPathComponent(marioCharactersFile)
    if FileManager.default.fileExists(atPath: fileURL.path) {
        marioCharacters = NSKeyedUnarchiver.unarchiveObject(withFile: fileURL.path)
        as! [MarioCharacter]
    }
}

func writeToFile () { // call whenever marioCharacters array changed
    let fileDir = FileManager.default.urls(for: .documentDirectory,
                                           in: .userDomainMask).first!
    let fileURL = fileDir.appendingPathComponent(marioCharactersFile)
    NSKeyedArchiver.archiveRootObject(marioCharacters, toFile: fileURL.path)
}
```

Database Support

- Core Data
 - iOS specific
 - Object store
- SQLite
 - Cross-platform (already available in iOS)
 - Table store
- Realm
 - Cross-platform
 - Object store

Core Data

- Check “Use Core Data”
for New Project
- Includes empty data
model
- Includes boilerplate
code to create database

Choose options for your new project:

Product Name: CoreDataDemo1

Team: Washington State University (Office of Gr... ▾)

Organization Name: Washington State University

Organization Identifier: edu.wsu

Bundle Identifier: edu.wsu.CoreDataDemo1

Language: Swift ▾

Devices: Universal ▾

☒ Use Core Data

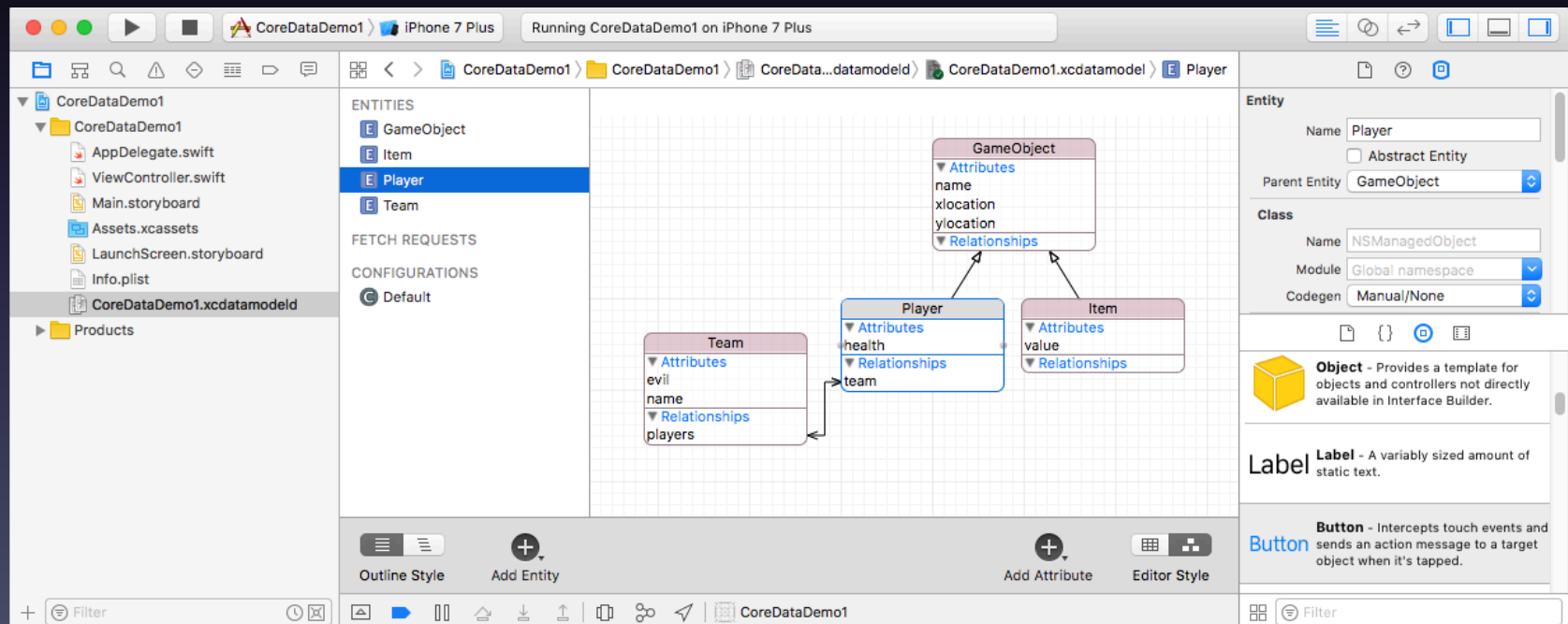
☐ Include Unit Tests

☐ Include UI Tests

Cancel Previous Next

Core Data: Create Managed Object Model (Schema)

- Schema consists of entities, their attributes, and relationships



Core Data Stack

- Handles interactions with data store
 - Persistent container (`NSPersistentContainer`)
 - Obtained from `UIApplication.shared.delegate`
 - Managed object context (`NSManagedObjectContext`)
 - Obtained from `NSPersistentContainer.viewContext`

Core Data: Insert

- Methods
 - `NSEntityDescription.insertNewObject(forEntityName: String, into: NSManagedObjectContext) -> NSManagedObject`
 - `NSManagedObject.setValue(value: Any?, forKey: String)`
 - `NSManagedObjectContext.save()`

Core Data: Insert

```
import CoreData

class ViewController: UIViewController {

    var managedObjectContext: NSManagedObjectContext!
    var appDelegate: AppDelegate!

    override func viewDidLoad() {
        super.viewDidLoad()
        self.appDelegate = UIApplication.shared.delegate as! AppDelegate
        self.managedObjectContext = appDelegate.persistentContainer.viewContext
    }

    func addPlayer() {
        let player = NSEntityDescription.insertNewObject(forEntityName:
            "Player", into: self.managedObjectContext)
        player.setValue("Mario", forKey: "name")
        player.setValue(100, forKey: "health")
        self.appDelegate.saveContext() // In AppDelegate.swift
    }
}
```

Core Data: Fetch

- Methods
 - `NSFetchRequest<NSManagedObject>(entityName: String) -> NSFetchRequest<NSManagedObject>`
 - `NSFetchRequest<NSManagedObject>.predicate = NSPredicate(format: String, args...)`
 - `NSManagedObjectContext.fetch(request: NSFetchRequest<NSManagedObject>) throws`

Core Data: Fetch

```
func getPlayers() {
    let fetchRequest = NSFetchRequest<NSManagedObject>(entityName: "Player")
    var players: [NSManagedObject]!
    do {
        players = try self.managedObjectContext.fetch(fetchRequest)
    } catch {
        print("getPlayers error: \(error)")
    }
    print("Found \(players.count) players")
    for player in players {
        let name = player.value(forKey: "name") as! String
        let health = player.value(forKey: "health") as! Int
        print("    Found player \(name) with health \(health)")
    }
}
```

Core Data Operations

- Methods
 - `NSManagedObjectContext.delete(object: NSManagedObject)`
 - `NSManagedObjectContext.save()`

Core Data: Delete

```
func removePlayers() {
    let playerName = "Mario"
    let fetchRequest = NSFetchRequest<NSManagedObject>(entityName: "Player")
    fetchRequest.predicate = NSPredicate(format: "name == %@", playerName)
    var players: [NSManagedObject]!
    do {
        players = try self.managedObjectContext.fetch(fetchRequest)
    } catch {
        print("removePlayers error: \(error)")
    }
    for player in players {
        self.managedObjectContext.delete(player)
    }
    self.appDelegate.saveContext() // In AppDelegate.swift
}
```


SQLite

- C API to SQL database
- Swift wrappers provided by SQLite.swift
 - <https://github.com/stephencelis/SQLite.swift>
 - Install via CocoaPods

SQLite.swift: Create Database

- `Connection(pathToDB: String)` throws -> `Connection`
- `Table(name: String)`
- `Expression<Type>(name: String)`
- `Connection.run(Table.create() { t in t.column(Expression<Type>) })` throws

SQLite.swift: Create Database

```
import SQLite
class ViewController: UIViewController {
    var db: Connection!
    var playersTable: Table!
    var nameExp: Expression<String>!
    var healthExp: Expression<Int>!

    func createDatabase() {
        // Connect to writable database in app's Documents directory.
        let path = NSSearchPathForDirectoriesInDomains(.documentDirectory,
                                                       .userDomainMask, true).first!
        db = try? Connection("\(path)/db.sqlite3")
        // Create Players table
        playersTable = Table("Players")
        nameExp = Expression<String>("name")
        healthExp = Expression<Int>("health")
        do {
            try db?.run(playersTable.create(ifNotExists: true) { t in
                t.column(nameExp)
                t.column(healthExp)})
        } catch {
            print("error creating table")
        }
    }
}
```

SQLite.swift: Insert

- `Connection.run(Table.insert(Expression<Type>
<- Any?, ...) throws`

SQLite.swift: Insert

```
func addPlayers() {  
    do {  
        try db.run(playersTable.insert(nameExp <- "Mario", healthExp <- 100))  
        try db.run(playersTable.insert(nameExp <- "Yoshi", healthExp <- 200))  
    } catch {  
        print("insert error")  
    }  
}
```

SQLite.swift: Fetch

- `Connection.scalar(Table.count) -> Int`
- `Connection.prepare(Table)` throws

SQLite.swift: Fetch

```
func getPlayers() {  
    if let count = try? db.scalar(playersTable.count) {  
        print("Found \(count) players")  
        if let players = try? db.prepare(playersTable) {  
            for player in players {  
                print(" Found player \(player[nameExp]) with health \(player[healthExp])")  
            }  
        }  
    }  
}
```

SQLite.swift: Delete

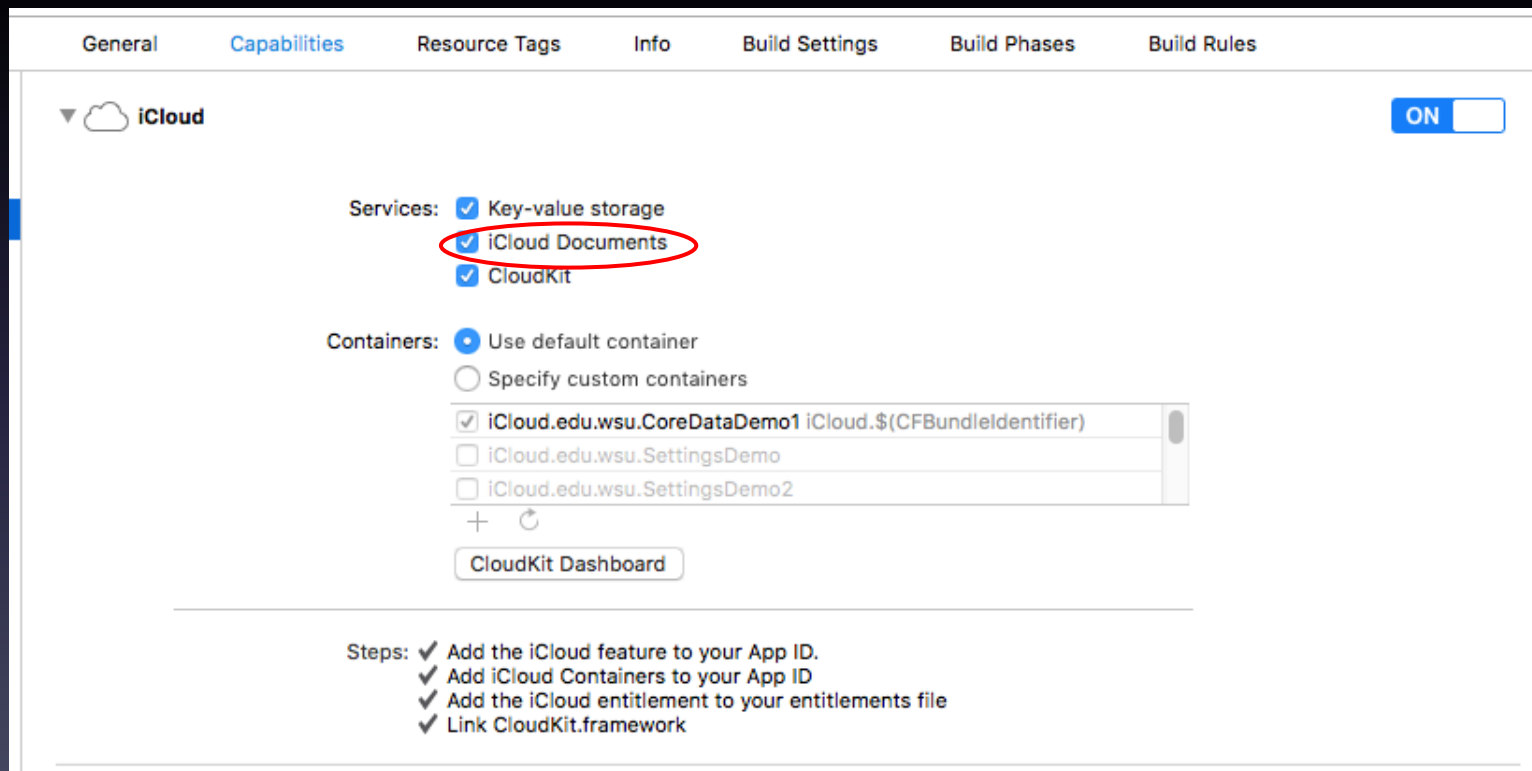
- `Table.Filter(Expression<Type> == Any?)`
- `Connection.run(Filter.delete())` throws

SQLite.swift: Delete

```
func removePlayers() {  
    let marioFilter = playersTable.filter(nameExp == "Mario")  
    do {  
        try db.run(marioFilter.delete())  
    } catch {  
        print("delete error")  
    }  
}
```

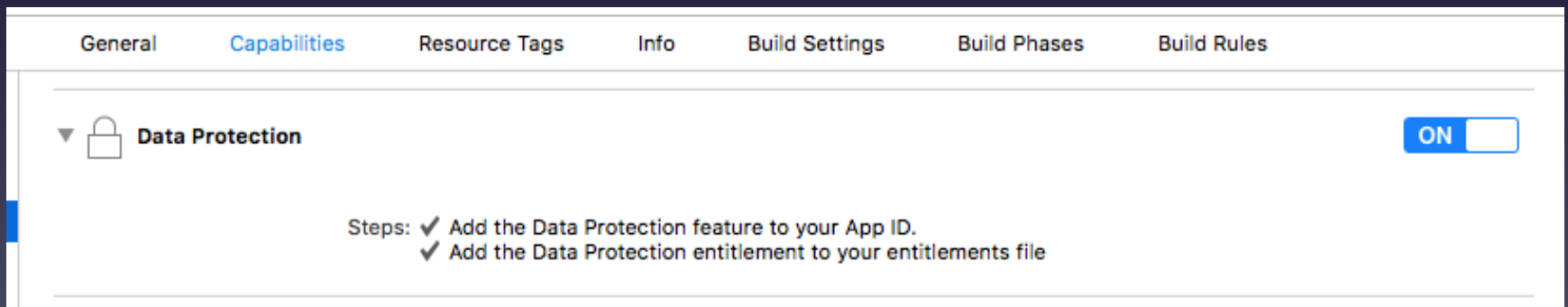
Data Backup to iCloud

- Available with iCloud capability



Data Protection

- Files encrypted and inaccessible with device locked
- Can selectively encrypt files



Resources

- Core Data
 - developer.apple.com/library/content/documentation/Cocoa/Conceptual/CoreData/
 - www.raywenderlich.com/145809/getting-started-core-data-tutorial
- SQLite
 - www.raywenderlich.com/123579/sqlite-tutorial-swift
 - github.com/stephencelis/SQLite.swift
- iCloud
 - developer.apple.com/icloud/
- Data Protection
 - developer.apple.com/library/content/documentation/IDEs/Conceptual/AppDistributionGuide/AddingCapabilities/AddingCapabilities.html