User Defaults

User Defaults

- User defaults are a way to store simple key/value pairs
  - Similar to a dictionary, but with persistence
  - Useful for small amounts of data (<100 KB)
- Very easy mechanism, but limited capabilities
- Typically used for application configuration data
- Data types you can store in User Defaults:
  - NSData
  - NSString; String
  - NSNumber; UInt/Int/Float/Double/Bool
  - NSDate
  - NSArray; Array
  - NSDictionary; Dictionary

User Defaults

You use the singleton object `UserDefaults.standard` to access the user defaults store.

1. Associate the singleton object with a variable:

```
let defaults = UserDefaults.standard
```

2. Call a method to save the data you want to store, providing key and value

```
defaults.set(<value>, forKey:<keyName>)
```

`<value>` is the data you want to store (most any type)
`<keyName>` is a String that you want to use to identify the data

Writing to User Defaults

```
// Define keys for the values to store
    let kUserIdKey = "userId”
    let kTotalKey = "total”
    let kNameKey = "name”

// Define the values to store
    let userId = 900
    let total = 1275.55
    let name = "University of Texas”

// Get a reference to the global user defaults object
    let defaults = UserDefaults.standard

// Store various values
    defaults.set(userId, forKey: kUserIdKey)
    defaults.set(total, forKey: kTotalKey)
    defaults.set(name, forKey: kNameKey)
```

Reading From User Defaults

There are several convenience methods for retrieving values from User Defaults:

```
func defaults.integer(forKey: <keyName>)

func defaults.double(forKey: <keyName>)

func defaults.string(forKey: <keyName>)
```

etc.

Reading From User Defaults

```
// Retrieve the previously stored values
    let retrievedUserId =
         defaults.integer(forKey: kUserIdKey)

    let retrievedTotal =
         defaults.double(forKey: kTotalKey)

    let retrievedName =
         defaults.string(forKey: kNameKey)
```

Removing From User Defaults

```
// Remove the objects from User Defaults
    defaults.removeObject(forKey: kUserIdKey)

    defaults.removeObject(forKey: kTotalKey)

    defaults.removeObject(forKey: kNameKey)
```

User Defaults

Some caveats:

- The methods that return Bool, Int, Float, and Double do not return Optionals. In those cases, they return sentinel values appropriate to their types which are false (or NO), 0, 0.0, and 0.0 respectively.
- Methods that return AnyObject indicates you'll need to cast to the correct type before using

Core Data

Core Data

- One of the ways to persist data in an iOS app
- Some consider it the best for non-trivial storage
- Makes it simple and fast to work with large amounts of data (> 100KB)
- It's technically not a true “database”, but you can do database-like stuff
- Multiple choices for the “backing store”:
  - Sqlite (default)
  - XML
  - Binary

Core Data

Core Data is an iOS framework that provides powerful functionality to query and persist non-trivial data.

- It lets developers store (or retrieve) data in a database in an object-oriented way.
- With Core Data, you can easily map the objects in your apps to the table records in the database without even knowing any SQL.
- That said, it is much more than a storage mechanism:
  - Manages data object life cycles
  - Tracks changes to the data
  - Effortless undo support
  - Saves data to disk

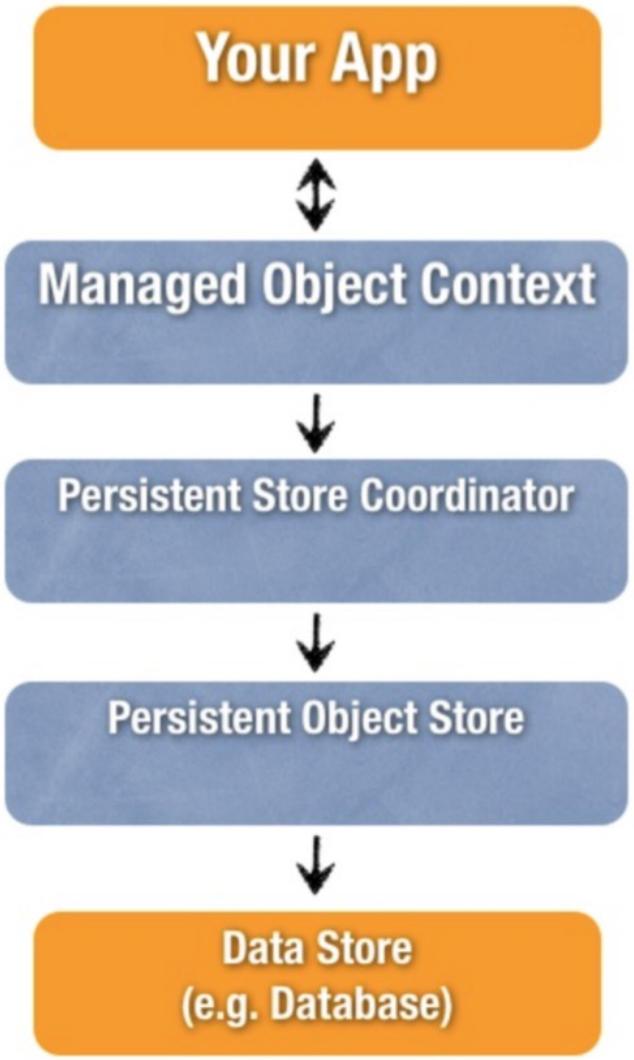
Core Data Architectural Overview

Managed Object Model

- Defines the structure of the data – the data schema (data type, relationships)
- Use the Data Model Design Tool in Xcode to define the models: it's a way of creating an object graph backed by a database.

Managed Object Context

- Acts as a temporary scratch space
- Objects fetched from the persistent store are stored in the context, where we can manipulate them
- Changes are monitored here



Core Data Architectural Overview: Entities

Entities are instances of the data models.

- Example: an Employee or a Company.
- In a relational database, an entity corresponds to a *table*. (Think of a table like an Excel spreadsheet, with columns and rows.)

In Core Data, an entity is a *class definition*.

Core Data Architectural Overview: Attributes

Attributes are values stored in the Entities.

- Example: an Employee entity could have attributes for the employee's name, position and salary.
- In a relational database, an attribute corresponds to a particular field in a table.

In Core Data, an attribute is a *property* representing a piece of information attached to a particular entity.

Core Data Architectural Overview: Relationships

Relationships are connections between the Entities.

- Relationships between two entities are called *to-one* relationships.
- Relationships between one and many entities are called *to-many* relationships.
- Example: A Manager can have a to-many relationship with a set of employees, whereas an individual Employee will have a to-one relationship with his manager.

In Core Data, a relationship is a link between multiple entities. It's a *property* that points to other entities in the database.