LEARNING Core Data FOR iOS

A Hands-On Guide to Building Core Data Applications

TIM ROADLEY

FREE SAMPLE CHAPTER

SHARE WITH OTHERS

Learning Core Data for iOS

Addison-Wesley Learning Series

LEARNING JavaScript
A Hands-On Guide to the Fundamentals of Modern JavaScript
TIM WRIGHT
LEARNING OBJECTIVE-C 2.0
A Hands-on Guide to Objective-C for Mac and iOS Developers
SECOND EDITION
ROBERT CLAIR
LEARNING Cocos2D
Hands-On Guide to Building iOS Games with Cocos2D, Box2D, and Chipmunk
ROD STROUGO
RAY WENDERLICH
LEARNING ANDROID GAME PROGRAMMING
A Hands-On Guide to Building Your First Android Game
RICK ROGERS
Addison-Wesley

Visit informit.com/learningseries for a complete list of available publications.

The Addison-Wesley Learning Series is a collection of hands-on programming guides that help you quickly learn a new technology or language so you can apply what you've learned right away.

Each title comes with sample code for the application or applications built in the text. This code is fully annotated and can be reused in your own projects with no strings attached. Many chapters end with a series of exercises to encourage you to reexamine what you have just learned, and to tweak or adjust the code as a way of learning.

Titles in this series take a simple approach: they get you going right away and leave you with the ability to walk off and build your own application and apply the language or technology to whatever you are working on.

Addison-Wesley informIT the trusted technology learning source | Safari Books Online

ALWAYS LEARNING PEARSON

Learning Core Data for iOS

Tim Roadley

Addison-Wesley

Upper Saddle River, NJ • Boston • Indianapolis • San Francisco
New York • Toronto • Montreal • London • Munich • Paris • Madrid
Cape Town • Sydney • Tokyo • Singapore • Mexico City

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and the publisher was aware of a trademark claim, the designations have been printed with initial capital letters or in all capitals.

The author and publisher have taken care in the preparation of this book, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

The publisher offers excellent discounts on this book when ordered in quantity for bulk purchases or special sales, which may include electronic versions and/or custom covers and content particular to your business, training goals, marketing focus, and branding interests. For more information, please contact:

U.S. Corporate and Government Sales
(800) 382-3419
corpsales@pearsontechgroup.com

For sales outside the United States, please contact:

International Sales
international@pearsoned.com

AirPlay, AirPort, AirPrint, AirTunes, App Store, Apple, the Apple logo, Apple TV, Aqua, Bonjour, the Bonjour logo, Cocoa, Cocoa Touch, Cover Flow, Dashcode, Finder, FireWire, iMac, Instruments, Interface Builder, iOS, iPad, iPhone, iPod, iPod touch, iTunes, the iTunes logo, Leopard, Mac, Mac logo, Macintosh, Multi-Touch, Objective-C, Quartz, QuickTime, QuickTime logo, Safari, Snow Leopard, Spotlight, and Xcode are trademarks of Apple, Inc., registered in the United States and other countries. OpenGL and the logo are registered trademarks of Silicon Graphics, Inc. The YouTube logo is a trademark of Google, Inc. Intel, Intel Core, and Xeon are trademarks of Intel Corp. in the United States and other countries.

Library of Congress Control Number: 2013946325

Visit us on the Web: informit.com/aw

Copyright © 2014 Pearson Education, Inc.

All rights reserved. Printed in the United States of America. This publication is protected by copyright, and permission must be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise. To obtain permission to use material from this work, please submit a written request to Pearson Education, Inc., Permissions Department, One Lake Street, Upper Saddle River, New Jersey 07458, or you may fax your request to (201) 236-3290.

ISBN-13: 978-0-321-90576-5
ISBN-10: 0-321-90576-8

Text printed in the United States on recycled paper at RR Donnelley in Crawfordsville, Indiana.

First printing: November 2013

Editor-in-Chief
Mark Taub

Senior Acquisitions Editor
Trina MacDonald

Senior Development Editor
Chris Zahn

Development Editor
Sheri Cain

Managing Editor
Kristy Hart

Senior Project Editor
Betsy Gratner

Copy Editor
Bart Reed

Indexer
Brad Herriman

Proofreader
Paula Lowell

Technical Reviewers
Carl Brown
Mark H. Granoff
Ricky O’Sullivan
Rich Warren

Publishing Coordinator
Olivia Basegio

Cover Designer
Chuti Prasertsith

Compositor
Nonie Ratcliff

❖

The more I learn, the more I learn I need to learn more.

I dedicate this book to my wonderful wife, Tracey, who has given up many nights and weekends with me to help make this book a reality. Finally, we can sit back and relax together again! I'm sure Tyler & Taliah will let us....

<3

❖

Contents at a Glance

Preface xv

1 Your First Core Data Application 1

2 Managed Object Model Basics 19

3 Managed Object Model Migration 45

4 Managed Object Model Expansion 71

5 Table Views 91

6 Views 125

7 Picker Views 159

8 Preloading Data 189

9 Deep Copy 215

10 Performance 239

11 Background Processing 263

12 Search 281

13 Back Up and Restore with Dropbox 297

14 iCloud 331

15 Taming iCloud 351

16 Web Service Integration 379

A Preparing Grocery Dude for Chapter 1 411

B Preparing Grocery Cloud for Chapter 16 417

Index 425

Table of Contents

Preface xv

1 Your First Core Data Application 1

What Is Core Data? 1

Persistent Store Coordinator 2

Managed Object Model 4

Managed Object Context 4

When to Use Core Data 5

Introducing Grocery Dude 5

Adding Core Data to an Existing Application 6

Introducing Core Data Helper 7

Core Data Helper Implementation 9

Files 9

Paths 10

Setup 11

Saving 14

Summary 16

Exercises 16

2 Managed Object Model Basics 19

What Is a Managed Object Model? 19

Adding a Managed Object Model 20

Entities 20

Attributes 22

Integer 16/32/64 23

Float and Double 24

Decimal 25

String 25

Boolean 25

Date 25

Binary Data 26

Transformable 26

Attribute Settings 28

Subclassing `NSManagedObject` 29

Scalar Properties for Primitive Data Types 31
Snippet Demo Method 31
Creating a Managed Object 32
Backend SQL Visibility 33
Fetching Managed Objects 36
    Fetch Request Sorting 38
    Fetch Request Filtering 39
    Fetch Request Templates 40
Deleting Managed Objects 42
Summary 43
Exercises 43

3 Managed Object Model Migration 45
Changing a Managed Object Model 45
Adding a Model Version 47
Lightweight Migration 48
Default Migration 52
Migration Manager 57
Summary 69
Exercises 69

4 Managed Object Model Expansion 71
Relationships 71
Delete Rules 77
Validation Errors 81
Entity Inheritance 86
Summary 90
Exercises 90

5 Table Views 91
Table Views 101 91
Core Data Table Views 93
Introducing `CoreDataTVC` 93
    Fetching 95
    `DATASOURCE: UITableView` 96
`DELEGATE: NSFetchedResultsController` 98
AppDelegate's `CoreDataHelper` Instance 104

Introducing `PrepareTVC` 105
Data 106
View 108
Interaction 114
Introducing `ShopTVC` 117
Data 117
View 118
Interaction 121
Summary 122
Exercises 122

6 Views 125
Overview 125
The Target View Hierarchy 126
Introducing `ItemVC` 127
Keeping Reference to a Selected Item 128
Passing a Selected Item to `ItemVC` 128
Configuring the Scroll View and Text Fields 129
`ItemVC` Implementation 134
Interaction 134
`DELEGATE: UITextField` 136
View 137
Data 139
Units, Home Locations, and Shop Locations 143
Adding and Editing Units 145
Implementing `UnitsTVC` 147
Implementing `UnitVC` 149
Segue from `UnitsTVC` to `UnitVC` 153
Adding and Editing Home or Shop Locations 154
Configuring the Home Location Views 154
Configuring the Shop Location Views 156
Summary 158
Exercises 158

7 Picker Views 159
Overview 159
Introducing `CoreDataPickerTF` 160

DELEGATE+DATASOURCE: UIPickerView 162
Interaction 164
Data 165
View 165
Introducing UnitPickerTF 168
Creating the Unit Picker 170
Connecting the Unit Picker 171
Configuring ItemVC for the Unit Picker 171
Introducing LocationAtHomePickerTF 176
Introducing LocationAtShopPickerTF 178
Creating the Location Pickers 179
Connecting the Location Pickers 180
Configuring ItemVC for the Location Pickers 181
Picker-Avoiding Text Field 186
Summary 188
Exercises 188

8 Preloading Data 189
Default Data 189
Is an Import Required? 190
Importing from XML 193
Creating an Import Context 196
Preventing Duplicate Default Data 197
Triggering a Default Data Import 198
Introducing CoreDataImporter 200
Selecting Unique Attributes 205
Mapping XML Data to Entity Attributes 206
Importing from a Persistent Store 209
Using the Default Data Store as the Initial Store 210
Summary 213
Exercises 214

9 Deep Copy 215
Overview 215
Configuring a Source Stack 218
Configuring the Source Store 219

Enhancing `CoreDataImporter` 221
Object Info 221
Array For Entity 222
Copy Unique Object 223
Establish To-One Relationship 224
Establish To-Many Relationship 226
Establish Ordered To-Many Relationship 227
Copy Relationships 228
Deep Copy Entities 230
Triggering a Deep Copy 232
Summary 236
Exercises 236

10 Performance 239
Identifying Performance Issues 239
Implementing the Camera 240
Generating Test Data 245
Measuring Performance with SQLDebug 250
Measuring Performance with Instruments 251
Improving Performance 254
Clean Up 261
Summary 262
Exercises 262

11 Background Processing 263
Background Save 263
Background Processing 268
Introducing `Faulter` 269
Introducing `Thumbnailer` 273
Summary 279
Exercises 279

12 Search 281
Updating `CoreDataTVC` 282
Updating `PrepareTVC` 289
Summary 294
Exercises 294

13 Back Up and Restore with Dropbox 297
Dropbox Integration 298
Supporting Frameworks 299
Linking to Dropbox 300
Introducing `DropboxHelper` 302
Introducing `DropboxTVC` 304
Preparing `CoreDataHelper` 307
Building `DropboxHelper` 308
Local File Management 308
Dropbox File Management 310
Backup & Restore 312
Building `DropboxTVC` 316
Creating Backups 319
Displaying Backups 322
Restore 324
Summary 328
Exercises 328

14 iCloud 331
Overview 331
Enabling iCloud 333
Updating `CoreDataHelper` for iCloud 334
The iCloud Store 335
iCloud Notifications 337
The Debug Navigator 341
Disabling iCloud 342
Summary 348
Exercises 348

15 Taming iCloud 351
De-Duplication 351
Seeding 360
Developing with a Clean Slate 367
Configurations 369
Finishing Touches 370
Summary 371
Exercises 372

16 Web Service Integration 379
Introducing StackMob 379
The StackMob SDK 381
Creating a StackMob Application 382
Managed Object Model Preparation 383
Configuring a StackMob Client 385
Saving 387
Underlying Changes 389
Automatic Schema Generation 390
Schema Permissions 393
Authentication 394
  Securing the User Class 396
  Introducing `LoginVC` 397
Maintaining Responsiveness 406
Summary 408
Exercises 409

A Preparing Grocery Dude for Chapter 1 411
New Xcode Project 411
Storyboard Design 412
App Icons and Launch Images 415

B Preparing Grocery Cloud for Chapter 16 417
Renaming Grocery Dude 417
Repointing File Paths 418
Renaming Groups and Tests 419
Renaming the Scheme 420
Updating the Artwork 421
Disabling Camera and Image Support 421
Workaround: Section Name Key Path Issue 422
Summary 423

Index 425

Acknowledgments

A resounding thank-you first goes out to Trina MacDonald for giving me the opportunity to write this book. Her guidance throughout the whole process has been invaluable, as has the assistance of the fantastic technical reviewers Rich Warren, Carl Brown, Mark Granoff, and Ricky O'Sullivan. You guys saved this book from a few bugs that crept through on those late nights and also provided some great insight and coding technique suggestions. Special thanks also go to Betsy Gratner, Olivia Basegio, Bart Reed, Sheri Cain, Chris Zahn, and Matt Vaznaian for your assistance in making this book a reality.

About the Author

Tim Roadley is a senior analytics software consultant at Emite Pty Ltd. He is primarily focused on delivering business intelligence dashboards, currently for one of Australia's major banks. Prior to Emite, Tim was Infrastructure Manager at Cuscal Pty Ltd, where he was heavily involved in designing and implementing a payments switch that drives 1,300+ ATMs throughout Australia. By night he is an avid iOS developer and tutorial writer, with several apps on the App Store, including Teamwork, iSoccer, and now Grocery Dude and Grocery Cloud. In his downtime, he enjoys spending time with his wonderful wife, Tracey, and two lovely children, Tyler and Taliah.

Preface

Every day, millions of Apple devices run applications, or apps, which rely on Core Data. This has led to a mature, stable, and incredibly fast platform for apps to access their data. Core Data itself is not a database. In fact, Core Data is a framework that, among other things, automates how you interact with a database. Instead of writing SQL code, you use Objective-C objects. All the associated SQL you would otherwise have to write yourself is generated automatically. This leaves you with all the benefits of a relational database without the headache of writing, testing, and optimizing SQL queries within your Objective-C code. The SQL code generated automatically "under the hood" is the product of years of refinement and optimization by Apple's masterful engineers. Using Core Data will not only speed up your own application development time, it will also significantly reduce the amount of code you have to write.

Here are some notable features of Core Data:

- Change management (undo and redo)
- Relationships
- Data model versioning and migration
- Efficient fetching (through batching and faulting)
- Efficient filtering (through predicates)
- Data consistency and validation

With this book, you'll be introduced to Core Data features and best practices. As you progress through the chapters, you'll also build a fully functional Core Data iPhone app from scratch. Each key piece of information will be explained in succinct detail so you can apply what you've learned straight away. The sample application built throughout this book has been especially designed to demonstrate as many aspects of Core Data as possible. At the same time it is a completely real-world application available on the App Store today. This should make it easier to absorb concepts as you relate them to real-life scenarios.

The arrival of iOS 7 has seen major improvements in the speed, reliability, and simplicity of Core Data integration with iCloud. I encourage anyone who has previously given up on this technology to give it another go, because you will be pleasantly surprised.

If you have feedback, bug fixes, corrections, or anything else you would like to contribute to a future edition, please contact me at timroadley@icloud.com. Finally, thank you for taking an interest in this book. I have put a lot of effort into meticulously crafting it, so I truly hope it helps you on your way to mastering this brilliant technology.

—Tim Roadley (*@TimRoadley*), September 2013

Who Is This Book For?

This book is aimed at Objective-C programmers who wish to learn how to efficiently manage data in their iOS apps. Prior experience with databases may help you pick up some topics faster, yet is not essential knowledge. As old habits die hard, some SQL programmers may find it more difficult to wrap their heads around some topics. Whatever your scenario, don't worry. Every step of the way will be explained and demonstrated clearly.

What You'll Need

As an Objective-C programmer, it is expected that you already have a reasonably modern Mac running Xcode 5 or above. You should also be quite familiar with Xcode and have an iOS device to test with. This is particularly true once you reach Chapter 10, "Performance," which is all about device performance.

You should already know what the Objective-C terms *property*, *method*, *delegate*, *class*, and *class instance* mean. If you're now uncertain that this book is for you, I suggest a detour via the following resources:

- *iOS Programming: The Big Nerd Ranch Guide* (search amazon.com)
- The iOS Newbie Tutorial Series (search timroadley.com)
- Learning Objective-C: A Primer (search apple.com)

How This Book Is Organized

This book takes you through the entire process of building the **Grocery Dude** and **Grocery Cloud** apps, which are available from the App Store today. Grocery Dude demonstrates Core Data integration with iCloud. Grocery Cloud demonstrates Core Data integration with StackMob. Each chapter in this book builds on the last, so you're introduced to topics in the order you need to implement them. Along the way you'll build helper classes that simplify redeployment of what you've learned into your own applications. In fact, the exercises at the end of Chapter 15, "Taming iCloud," guide you through a redeployment of these helper classes into an existing non–Core Data app. In next to no time, you'll have a fully functional Core Data app that is reliably integrated with iCloud.

Here's a brief summary of what you'll find in each chapter:

- **Chapter 1, "Your First Core Data Application"**—The groundwork is laid as the fundamental concepts of Core Data are introduced. You'll be shown what Core Data is, and just as importantly what it isn't. In addition, Core Data integration with an existing application is demonstrated as the `CoreDataHelper` class is implemented.

- **Chapter 2, "Managed Object Model Basics"**—Data models are introduced as parallels are drawn between traditional database schema design and Core Data. You'll be shown how to configure a basic managed object model as entities and attributes are discussed, along with accompanying advice on choosing the right data types. Inserting, fetching, filtering, sorting, and deleting managed objects is also covered and followed up with an introduction to fetch request templates.
- **Chapter 3, "Managed Object Model Migration"**—Experience lightweight migration, default migration, and using a migration manager to display migration progress. Learn how to make an informed decision when deciding between migration options for your own applications and become comfortable with the model-versioning capabilities of Core Data.
- **Chapter 4, "Managed Object Model Expansion"**—The true power of a relational data model is unlocked as different types of relationships are explained and added to Grocery Dude. Other model features such as abstract and parent entities are also covered, along with techniques for dealing with data validation errors.
- **Chapter 5, "Table Views"**—The application really comes to life as Core Data is used to drive memory-efficient and highly performing table views with a fetched results controller. Of course, most of the generic legwork is put into a reusable table view controller subclass called `CoreDataTVC`. By dropping this class into your own applications, you can easily deploy Core Data–driven table views yourself.
- **Chapter 6, "Views"**—Working with managed objects takes a front seat as you're shown how to pass them around the application. Objects selected on a table view are passed to a second view, ready for editing. The editing interface is added to Grocery Dude, demonstrating how to work with objects and then save them back to the persistent store.
- **Chapter 7, "Picker Views"**—As a nice touch, Core Data–driven picker views are added to the editing views. Picker views allow the user to quickly assign existing items to a unit of measurement, home location, or shop location. A special reusable text field subclass called `CoreDataPickerTF` is introduced, which replaces the keyboard with a Core Data picker view whenever an associated text field is tapped.
- **Chapter 8, "Preloading Data"**—Techniques for generating a persistent store full of default data from XML are explained and demonstrated in this chapter as the generic `CoreDataImporter` helper class is introduced. Once you have a persistent store to include with a shipping application, you'll then be shown how to determine whether a default data import is required or even desired by the user.
- **Chapter 9, "Deep Copy"**—A highly flexible and fine-grained alternative to `migratePersistentStore`, deep copy enables you to copy objects and relationships from selected entities between persistent stores. In this chapter, the `CoreDataImporter` helper class is enhanced with the deep copy capability.
- **Chapter 10, "Performance"**—Gain experience with Instruments as you identify and eliminate performance issues caused by the common pitfalls of a Core Data application. The camera functionality is introduced to highlight these issues and demonstrates just how important good model design is to a well-performing application.

- **Chapter 11, "Background Processing"**—Top-notch performance requires intensive tasks be offloaded to a background thread. Learn just how easy it is to run processes in the background as the example of photo thumbnail generation is added with a generic helper class called `Thumbnailer`. Also learn how to keep memory usage low with another helper class, called `Faulter`.
- **Chapter 12, "Search"**—Learn how to handle twin fetched results controllers in the one table view as you implement efficient search in `CoreDataTVC`.
- **Chapter 13, "Back Up and Restore with Dropbox"**—Create backups and synchronize them to Dropbox using their Sync API. Restore data to any iOS device using the same Dropbox account at the touch of a button.
- **Chapter 14, "iCloud"**—Enjoy the easiest, most reliable Core Data integration with iCloud yet. Handle multiple accounts and varying preferences on using iCloud without missing a beat.
- **Chapter 15, "Taming iCloud"**—Take iCloud integration to the next level with entity-level seeding and unique object de-duplication. Accurately emulate first-time iCloud use by resetting ubiquitous content globally, the right way.
- **Chapter 16, "Web Service Integration"**—Enable collaboration as cross-platform data sharing between multiple users is introduced with StackMob. StackMob has one of the best free Backend-as-a-Service (BaaS) offerings available, and its iOS API is native to Core Data. Thanks to StackMob for generously allowing its art assets to be used in this book and for its assistance with Chapter 16.
- **Appendix A, "Preparing Grocery Dude for Chapter 1"**—Every (non–Core Data) step involved in preparing the starting-point application for Chapter 1 is documented here for completeness.
- **Appendix B, "Preparing Grocery Cloud for Chapter 16"**—Every (non–Core Data) step involved in preparing the starting-point application for Chapter 16 is documented here for completeness.

Getting the Sample Code

The sample code built throughout this book is available for download from timroadley.com. Links are given in each chapter, or you can use Table P.1 as a reference, which is arranged in the order of implementation.

Table P.1 **Grocery Dude Code**

Final Code	Link
Appendix A	http://timroadley.com/LearningCoreData/GroceryDude-AfterAppendixA.zip
Chapter 1	http://timroadley.com/LearningCoreData/GroceryDude-AfterChapter01.zip
Chapter 2	http://timroadley.com/LearningCoreData/GroceryDude-AfterChapter02.zip

Final Code	Link
Chapter 3	http://timroadley.com/LearningCoreData/GroceryDude-AfterChapter03.zip
Chapter 4	http://timroadley.com/LearningCoreData/GroceryDude-AfterChapter04.zip
Chapter 5	http://timroadley.com/LearningCoreData/GroceryDude-AfterChapter05.zip
Chapter 6	http://timroadley.com/LearningCoreData/GroceryDude-AfterChapter06.zip
Chapter 7	http://timroadley.com/LearningCoreData/GroceryDude-AfterChapter07.zip
Chapter 8	http://timroadley.com/LearningCoreData/GroceryDude-AfterChapter08.zip
Chapter 9	http://timroadley.com/LearningCoreData/GroceryDude-AfterChapter09.zip
Chapter 10	http://timroadley.com/LearningCoreData/GroceryDude-AfterChapter10.zip
Chapter 11	http://timroadley.com/LearningCoreData/GroceryDude-AfterChapter11.zip
Chapter 12	http://timroadley.com/LearningCoreData/GroceryDude-AfterChapter12.zip
Chapter 13	http://timroadley.com/LearningCoreData/GroceryDude-AfterChapter13.zip
Chapter 14	http://timroadley.com/LearningCoreData/GroceryDude-AfterChapter14.zip
Chapter 15	http://timroadley.com/LearningCoreData/GroceryDude-AfterChapter15.zip
Chapter 15 "Mini-project"	http://timroadley.com/LearningCoreData/EasyiCloud.zip
Helper classes, for your own projects	http://timroadley.com/LearningCoreData/Generic%20Core%20Data%20Classes.zip
Appendix B	http://timroadley.com/LearningCoreData/GroceryCloud-AfterAppendixB.zip
Chapter 16	http://timroadley.com/LearningCoreData/GroceryCloud-AfterChapter16.zip

Note that occasionally lines of code in the chapters are too long to fit on the printed page. Where that occurs, a code-continuation arrow (➥) has been used to mark the continuation. For example:

```
[[NSURL fileURLWithPath:[self applicationDocumentsDirectory]]
➥URLByAppendingPathComponent:@"Stores"];
```

Editor’s Note: We Want to Hear from You!

As the reader of this book, you are our most important critic and commentator. We value your opinion and want to know what we’re doing right, what we could do better, what areas you’d like to see us publish in, and any other words of wisdom you’re willing to pass our way.

You can email or write me directly to let me know what you did or didn’t like about this book—as well as what we can do to make our books stronger.

Please note that I cannot help you with technical problems related to the topic of this book, and that due to the high volume of mail I receive, I might not be able to reply to every message.

When you write, please be sure to include this book’s title and author as well as your name and phone number or email address. I will carefully review your comments and share them with the author and editors who worked on the book.

Email: trina.macdonald@pearson.com

Mail: Trina MacDonald
Senior Acquisitions Editor
Addison-Wesley/Pearson Education, Inc.
75 Arlington St., Ste. 300
Boston, MA 02116

1

Your First Core Data Application

If you can't explain it simply, you don't understand it well enough.

Albert Einstein

Kinesthetic learning, or learning by doing, is one of the best ways to absorb and retain information. The topic of Core Data has been a great hurdle for many seasoned programmers, so it's about time a book with a hands-on approach to Core Data was written. In order to avoid side tracking into deep topics too early, this chapter has many pointers to later chapters. First things first: It will give you a Core Data essentials primer, then dive right in and show how to add Core Data to the sample application. The sample application will be expanded over the course of this book as increasingly advanced topics are introduced.

What Is Core Data?

Core Data is a framework that enables you to work with your data as objects, regardless of how they're persisted to disk. This is useful to you as an Objective-C programmer, because you should be comfortable using objects in code already. To provide data objects, known as *managed objects*, Core Data sits between your application and a **persistent store**, which is the generic term given to a data file such as an SQLite database, XML file (which can't be used as a persistent store on iOS), or Binary (atomic) store. These files are called "persistent" because they can survive the underlying hardware being reset. Another (oddly named) persistent store option is the In-Memory store. Although it isn't really "persistent," an In-Memory store allows you to leverage all the functional benefits of Core Data to manage your data, such as change management and validation, not to mention performance.

To map data to a persistent store from managed objects, Core Data uses a **managed object model**, where you configure your application's data structure using an **object graph**. You can think of an object graph as a collection of cookie cutters used to make managed objects from. The "object" in object graph refers to something called an **entity**, which is used as a cookie cutter to make a customized managed object. Once you have managed objects, you're then free to manipulate them natively in Objective-C, without having to write any SQL code (assuming you're using SQLite as the persistent store, which is the most common scenario). Core Data will transparently map those objects back to a persistent store when you save to disk.

A managed object holds a copy of data from a persistent store. If you use a database as a persistent store, then a managed object might represent data from a table row in that database. If you use an XML file as a persistent store (Mac only), then a managed object would represent data found within certain data elements. A managed object can be an instance of `NSManagedObject`; however, it's usually an instance of a *subclass* of `NSManagedObject`. This is discussed in detail in Chapter 2, "Managed Object Model Basics."

All managed objects exist in a **managed object context**. A managed object context exists in high-speed volatile memory, also known as RAM. One reason a managed object context is required is the overhead involved with transferring data to and from disk. Disk is much slower than RAM, so you don't want to use it more than necessary. Having a managed object context allows access to data that has been previously retrieved from disk to be very fast. The downside, however, is that you need to call `save:` on the managed object context periodically to write changes back to disk. The managed object context exists also to track changes to its objects in order to provide full undo and redo support.

> **Note**
>
> "If you can't explain it simply, you don't understand it well enough" is a famous quote from the late great Albert Einstein. Each chapter of this book is headed by a famous Albert Einstein quote. Core Data can be a difficult topic to learn; however, that doesn't mean it cannot be broken down and explained in understandable chunks. Whenever I write technical tutorials or documentation, I remember this quote and strive for easy-to-read, highly informative material.

To help visualize how the main pieces of Core Data fit together, examine Figure 1.1.

Persistent Store Coordinator

On the left of Figure 1.1, a **persistent store coordinator** is shown containing a persistent store with table rows. When you set up a persistent store coordinator, you'll commonly choose an SQLite database as the persistent store. Other options for the persistent store are Binary, XML, and In-Memory stores. The thing to note about Binary and XML stores is that they are atomic. This means that even if you only want to change a small amount of data, you still have to write out the whole file to disk when you save. Of course, the same issue applies when reading an atomic store into memory in the first place. This can become problematic if you have a lot of data because it consumes valuable memory.

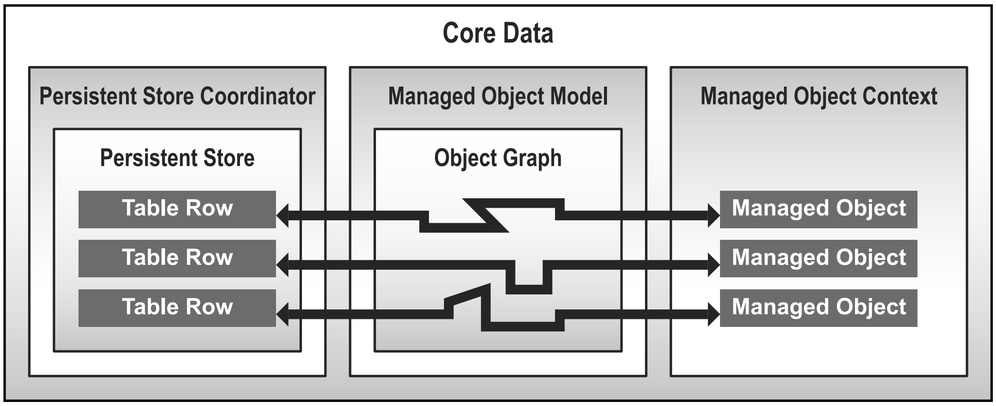


Figure 1.1 Core Data overview

An SQLite database, on the other hand, is updated incrementally as change logs, also known as *transaction logs,* are committed. As a result, the SQLite database memory footprint is comparably very small. For these reasons, you'll typically choose an SQLite database, especially when integrating Core Data with iCloud.

Note

Persistent stores should only ever be created by Core Data. You should not configure Core Data to use a database it did not originally create. If you need to use existing data, you should import it. This topic is covered in Chapter 8, "Preloading Data."

A persistent store coordinator can have multiple persistent stores. One situation where this may be appropriate is when Core Data is integrated with iCloud. By putting data that doesn't belong in iCloud into one store, and data that does in another, you will save network bandwidth and iCloud storage space. Even though you would then have two persistent stores, it does not mean that you need two separate object graphs. Using Core Data model configurations allows you to use separate stores, yet still have the one object graph. When you set up a Core Data model configuration, you can select what parts of the object graph belong in what persistent store. If you do use separate persistent stores, you'll need to ensure there's no requirement for a relationship between data in each store. Core Data configurations are discussed in Chapter 15, "Taming iCloud."

A persistent store is created from an instance of `NSPersistentStore` and a persistent store coordinator is created from an instance of `NSPersistentStoreCoordinator`.

Managed Object Model

In the middle of Figure 1.1, a **managed object model** is shown sitting between a persistent store coordinator and a managed object context. As its name suggests, a managed object model is the model or graphical representation of a data structure. It forms the basis on which managed objects are produced. This is similar to a database schema and is also referred to as an *object graph*. To create one, you'll use Xcode to configure entities and the relationships between them. An entity is similar to a table schema in a database. Entities don't contain data; they only dictate the properties that managed objects that are based on them will have. They're cookie cutters! Just as a database table has fields, similarly an entity has attributes. An attribute can have one of several data types, such as integer, string, or date. Chapter 2 and Chapter 4, "Managed Object Model Expansion," cover these topics in more detail.

A managed object model is created from an instance of `NSManagedObjectModel`.

Managed Object Context

On the right of Figure 1.1, a **managed object context** is shown with managed objects inside. A managed object context manages the lifecycle of objects within and provides powerful features such as faulting, change tracking, and validation. Faulting simply means that when you fetch data from a persistent store, only the parts you need are retrieved. Faulting is covered further in Chapter 10, "Performance." Change tracking is used for undo and redo support. Validation is the enforcement of rules set in the managed object model. For example, a minimum or maximum value rule can be enforced at an attribute level on an entity. Validation is discussed in Chapter 2.

Much like you can have multiple persistent stores, you may also have more than one managed object context. Typically you would use multiple contexts for background processing, such as saving to disk or importing data. When you call `save:` on a foreground context, you may notice user interface lag, especially when there are a lot of changes. An easy way to get around this issue is to simply call `save:` only when the home button is pressed and the application enters the background. Another more complicated yet flexible way is to use two managed object contexts. Remember that a managed object context is an area in high-speed memory. Well, you can actually configure a managed object context to save to *another* managed object context. Once you save a foreground context to a background context, you may then save the background context to disk asynchronously. This staged approach ensures the writes to disk never interfere with the user interface responsiveness.

The ability to configure a parent and child context hierarchy has been available since iOS 5. A child context treats its parent as a persistent store, when really the parent is another context that exists to process heavy workloads, such as saving in the background. This is discussed in further detail in Chapter 11, "Background Processing."

A managed object context is created from an instance of `NSManagedObjectContext`.

When to Use Core Data

Once your application outgrows trivial “settings” storage, such as `NSUserDefaults` and property lists, you’re going to run into memory usage issues. The solution is to use a database either directly or instead indirectly with Core Data. If you choose Core Data, you’ll save time otherwise spent coding a database interface. You’ll also enjoy big performance gains, as well as some functional benefits such as undo and validation. The time you would have spent developing, testing, and generally speaking “reinventing the wheel,” you’ll free up to focus on more important areas of your application.

Now you might be thinking, “I just want to save lots of stuff to disk, so why does it have to be so complicated?” Well, it’s not that difficult once a few key points are understood. Sure, you could write your own database interfaces and they would probably work great for a while. What happens, though, when your requirements change or you want to add, say, data synchronization between devices? How are your skills at building multithreaded data-import routines that don’t impact the user interface? Would your code also support undo and validation yet still be fast and memory efficient on an old iPhone?

The good news for you is that all the hard work has already been done and is wrapped up in the tried and tested Core Data Framework. Even if your application’s data requirements are minimal, it’s still worth using Core Data to ensure your application is as scalable as possible without compromising performance.

Once you start using Core Data, you’ll appreciate how robust and optimized it really is. The millions of people worldwide using Core Data applications every day has led to a mature feature set with performance to match. In short, you’ll save more time learning Core Data than throwing it in the too-hard basket and writing your own database interfaces. You’ll also benefit from loads of additional functionality for free.

> **Note**
>
> Before you continue, you should have at least Xcode 5 installed on your Mac. The code used in this book is targeted at iOS 7, so it won’t work in lower versions of Xcode. It is also recommended that you become a member of the iOS Developer Program, so you can run the sample application on your device as required. Go to http://developer.apple.com for further information on becoming a member.

Introducing Grocery Dude

Grocery Dude is the sample iPhone application you’ll create over the course of this book. As the features and best practices of Core Data are introduced, you can apply what you’ve learned to Grocery Dude. By the end of the book, you’ll have created a fast and fully functional Core Data application that integrates seamlessly with iCloud. If you would like to see the end result upfront, head over to the App Store and download Grocery Dude now. Note that Grocery Dude is written only for iPhone. This is because Core Data doesn't care what size screen you display data on; the concepts are the same. Without further ado, it's time to begin!

Have you ever stood in front of the fridge, pantry, cupboard, or some other location at home wondering what you're forgetting to put on your shopping list? Then, when you get to the store, you can't find something because you have no idea what aisle it's in? To top it off, after zigzagging all the way from aisle 8 (and finally finding what you're looking for in aisle 2), you discover the next item you need is back in aisle 8!

Here's what Grocery Dude will do for you:

- Remind you what you *might* need by sorting potential items by their storage location in your house.
- Help you locate items at the grocery store by showing what aisle they're in.
- Group your list by aisle so you only need to visit each aisle once, and in order.
- Sync between your devices with iCloud.
- Help you learn Core Data!

Note

Appendix A, "Preparing Grocery Dude for Chapter 1," shows the steps required to create the master project "Grocery Dude" from scratch. You may run through those steps manually or alternatively download the starting point project from http://www.timroadley.com/LearningCoreData/GroceryDude-AfterAppendixA.zip. Once you have downloaded the project, you should open it in Xcode 5 or above.

Adding Core Data to an Existing Application

When you create an iOS Application project in Xcode, you can choose from various starting-point templates. Using Core Data in your project is as easy as ticking the **Use Core Data** check box during creation of a Master-Detail, Utility Application, or Empty Application template-based project. Adding Core Data manually is more educational, so the "Grocery Dude" project is created based on the **Single View Application** template, which doesn't include Core Data. To use the Core Data Framework, you'll need to link it to the project.

Update Grocery Dude as follows to link to the Core Data Framework:

1. Select the **Grocery Dude Target**, as shown in Figure 1.2.
2. Click the + found in the **Linked Frameworks and Libraries** section of the **General** tab and then link to the **CoreData.framework**, as shown in Figure 1.2.

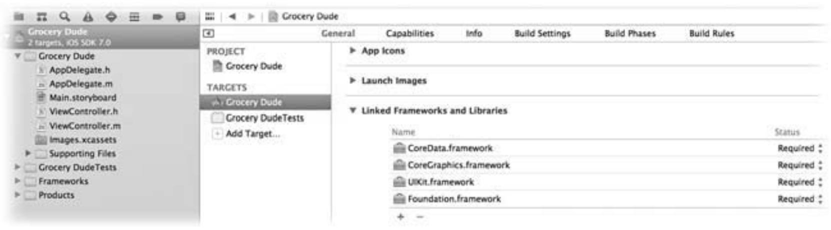


Figure 1.2 Linking the Core Data Framework

Introducing Core Data Helper

If you've ever examined the built-in Core Data–enabled templates, you may have noticed a lot of Core Data setup is done in the application delegate. So that you may apply the approach used in this book to your own projects, Core Data will be set up using a helper class. This keeps the Core Data components modular and portable. The application delegate will be used to lazily create an instance of the `CoreDataHelper` class. An instance of this class will be used to do the following:

- Initialize a managed object model
- Initialize a persistent store coordinator with a persistent store based on the managed object model
- Initialize a managed object context based on the persistent store coordinator

Update Grocery Dude as follows to create the `CoreDataHelper` class in a new Xcode group:

1. Right-click the **Grocery Dude** group in Xcode and then create a new group called **Generic Core Data Classes**, as shown in Figure 1.3.

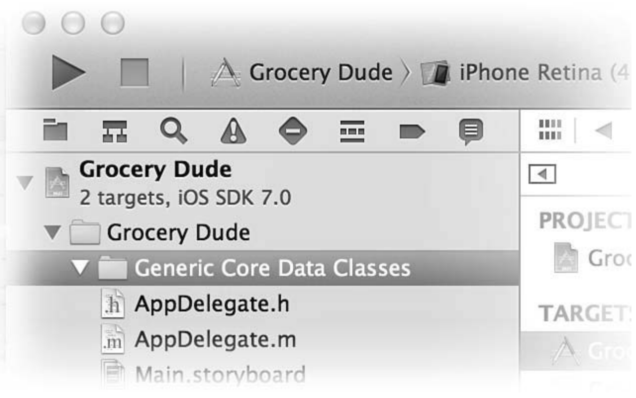


Figure 1.3 Xcode group for generic Core Data classes

2. Select the **Generic Core Data Classes** group.
3. Click **File** > **New** > **File...**.
4. Create a new **iOS** > **Cocoa Touch** > **Objective-C class** and then click **Next**.
5. Set **Subclass of** to `NSObject` and **Class** name to `CoreDataHelper` and then click **Next**.
6. Ensure the Grocery Dude target is ticked and then create the class in the Grocery Dude project directory.

Listing 1.1 shows new code intended for the `CoreDataHelper` header file.

Listing 1.1 **`CoreDataHelper.h`**

```
#import <Foundation/Foundation.h>
#import <CoreData/CoreData.h>

@interface CoreDataHelper :NSObject

@property (nonatomic, readonly) NSManagedObjectContext       *context;
@property (nonatomic, readonly) NSManagedObjectModel         *model;
@property (nonatomic, readonly) NSPersistentStoreCoordinator *coordinator;
@property (nonatomic, readonly) NSPersistentStore            *store;

- (void)setupCoreData;
- (void)saveContext;
@end
```

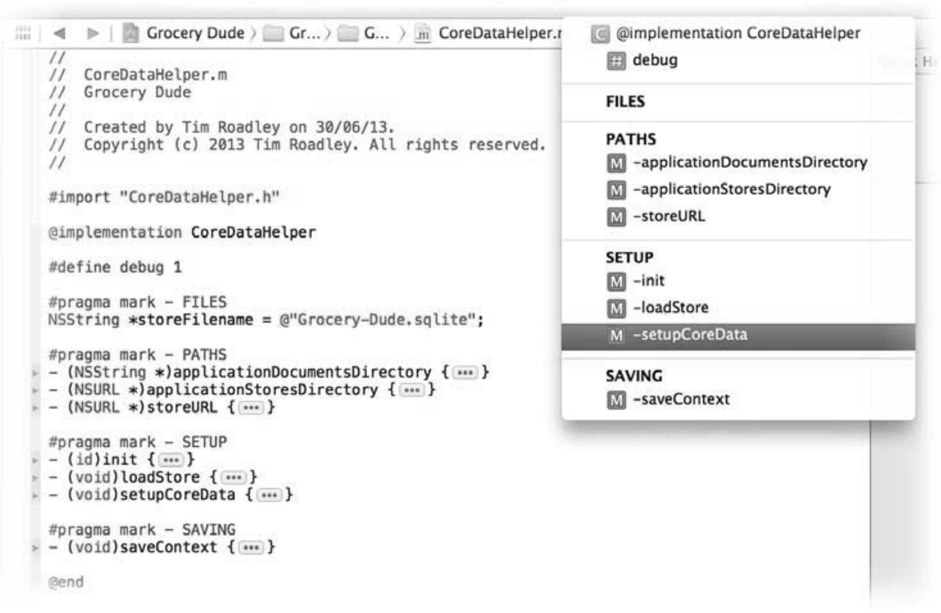
As an Objective-C programmer, you should be familiar with the purpose of header (`.h`) files. `CoreDataHelper.h` is used to declare properties for the context, model, coordinator and the store within it. The `setupCoreData` method will be called once an instance of `CoreDataHelper` has been created in the application delegate. The `saveContext` method may be called whenever you would like to save changes from the managed object context to the persistent store. This method can cause interface lag if there are a lot of changes to be written to disk. It is recommended that it only be called from the `applicationDidEnterBackground` and `applicationWillTerminate` methods of `AppDelegate.m`—at least until background save is added in Chapter 11.

Update Grocery Dude as follows to configure the `CoreDataHelper` header:

1. Replace all code in `CoreDataHelper.h` with the code from Listing 1.1. If you select `CoreDataHelper.m`, Xcode will warn that you haven't implemented the `setupCoreData` and `saveContext` methods, which is okay for now.

Core Data Helper Implementation

The helper class will start out with four main sections. These sections are `FILES`, `PATHS`, `SETUP`, and `SAVING`. For easy navigation and readability, these areas are separated by pragma marks. As shown in Figure 1.4, the pragma mark feature of Xcode allows you to logically organize your code and automatically provides a nice menu for you to navigate with.



```
//
//  CoreDataHelper.m
//  Grocery Dude
//
//  Created by Tim Roadley on 30/06/13.
//  Copyright (c) 2013 Tim Roadley. All rights reserved.
//

#import "CoreDataHelper.h"

@implementation CoreDataHelper

#define debug 1

#pragma mark - FILES
NSString *storeFilename = @"Grocery-Dude.sqlite";

#pragma mark - PATHS
- (NSString *)applicationDocumentsDirectory {…}
- (NSURL *)applicationStoresDirectory {…}
- (NSURL *)storeURL {…}

#pragma mark - SETUP
- (id)init {…}
- (void)loadStore {…}
- (void)setupCoreData {…}

#pragma mark - SAVING
- (void)saveContext {…}

@end
```

Figure 1.4 Pragma mark generated menu

Files

The `FILES` section of `CoreDataHelper.m` starts out with a persistent store filename stored in an `NSString`. When additional persistent stores are added later, this is where you'll set their filenames. Listing 1.2 shows the code involved along with a new `#define` statement, which will be used in most of the classes in Grocery Dude to assist with debugging. When `debug` is set to `1`, debug logging will be enabled for that class. Most `NSLog` commands will be wrapped in an `if (debug == 1)` statement, which will only work when debugging is enabled.

Listing 1.2 **CoreDataHelper.m: FILES**

```
#define debug 1

#pragma mark - FILES
NSString *storeFilename = @"Grocery-Dude.sqlite";
```

Update Grocery Dude as follows to add the `FILES` section:

1. Add the code from Listing 1.2 to the bottom of `CoreDataHelper.m` before `@end`.

Paths

To persist anything to disk, Core Data needs to know where in the file system persistent store files should be located. Three separate methods help provide this information. Listing 1.3 shows the first method, which is called `applicationDocumentsDirectory` and returns an `NSString` representing the path to the application's documents directory. You'll also notice the first use of an `if (debug==1)` statement wrapping a line of code that shows what method is running. This `NSLog` statement is useful for seeing the order of execution of methods in the application, which is great for debugging.

Listing 1.3 **CoreDataHelper.m: PATHS**

```
#pragma mark - PATHS
- (NSString *)applicationDocumentsDirectory {
if (debug==1) {
    NSLog(@"Running %@ '%@'", self.class,NSStringFromSelector(_cmd));
}
return [NSSearchPathForDirectoriesInDomains(NSDocumentDirectory, NSUserDomainMask,YES)
➥lastObject];
}
```

Update Grocery Dude as follows to add the `PATHS` section:

1. Add the code from Listing 1.3 to the bottom of `CoreDataHelper.m` before `@end`.

The next method, `applicationStoresDirectory`, appends a directory called Stores to the application's documents directory and then returns it in an `NSURL`. If the Stores directory doesn't exist, it is created as shown in Listing 1.4.

Listing 1.4 **CoreDataHelper.m: applicationStoresDirectory**

```
- (NSURL *)applicationStoresDirectory {
if (debug==1) {
    NSLog(@"Running %@ '%@'", self.class, NSStringFromSelector(_cmd));
}
NSURL *storesDirectory =
[[NSURL fileURLWithPath:[self applicationDocumentsDirectory]]
                                    URLByAppendingPathComponent:@"Stores"];

NSFileManager *fileManager = [NSFileManager defaultManager];
if (![fileManager fileExistsAtPath:[storesDirectory path]]) {
    NSError *error = nil;
    if ([fileManager createDirectoryAtURL:storesDirectory
              withIntermediateDirectories:YES
                               attributes:nil
                                    error:&error]) {
        if (debug==1) {
            NSLog(@"Successfully created Stores directory");}
        }
        else {NSLog(@"FAILED to create Stores directory: %@", error);}
    }
    return storesDirectory;
}
```

Update Grocery Dude as follows to add to the `PATHS` section:

1. Add the code from Listing 1.4 to the bottom of `CoreDataHelper.m` before `@end`.

The last method, which is shown in Listing 1.5, simply appends the persistent store filename to the store's directory path. The end result is a full path to the persistent store file.

Listing 1.5 **`CoreDataHelper.m`: `storeURL`**

```
- (NSURL *)storeURL {
if (debug==1) {
    NSLog(@"Running %@ '%@'", self.class, NSStringFromSelector(_cmd));
}
return [[self applicationStoresDirectory]
              URLByAppendingPathComponent:storeFilename];
}
```

Update Grocery Dude as follows to add to the `PATHS` section:

1. Add the code from Listing 1.5 to the bottom of `CoreDataHelper.m` before `@end`.

Setup

With the files and paths ready to go, it's time to implement the three methods responsible for the initial setup of Core Data. Listing 1.6 shows the first method, called `init`, which runs automatically when an instance of `CoreDataHelper` is created.

Listing 1.6 **CoreDataHelper.m: SETUP**

```
#pragma mark - SETUP
- (id)init {
if (debug==1) {
    NSLog(@"Running %@ '%@'", self.class, NSStringFromSelector(_cmd));
}
    self = [super init];
    if (!self) {return nil;}

    _model = [NSManagedObjectModel mergedModelFromBundles:nil];
    _coordinator = [[NSPersistentStoreCoordinator alloc]
                         initWithManagedObjectModel:_model];
    _context = [[NSManagedObjectContext alloc]
                         initWithConcurrencyType:NSMainQueueConcurrencyType];
    [_context setPersistentStoreCoordinator:_coordinator];
    return self;
}
```

The `_model` instance variable points to a managed object model. The managed object model is initiated from all available data model files (object graphs) found in the main bundle by calling `mergedModelFromBundles` and passing `nil`. At the moment, there are no model files in the project; however, one will be added in Chapter 2. It is possible to pass an `NSArray` of `NSBundles` here in case you wanted to merge multiple models. Usually you won't need to worry about this.

Note

Another way to initialize a managed object model is to specify the exact model file to use. This takes over twice the amount of code, as opposed to just merging bundles. Here's how you would manually specify the model to use: `_model = [[NSManagedObjectModel alloc] initWithContentsOfURL: [[NSBundle mainBundle] URLForResource:@"Model" withExtension:@"momd"]];`.

The `_coordinator` instance variable points to a persistent store coordinator. It is initialized based on the `_model` pointer to the managed object model that has just been created. So far, the persistent store coordinator has no persistent store files because they will be added later by the `setupCoreData` method.

The `_context` instance variable points to a managed object context. It is initialized with a concurrency type that tells it to run on a "main thread" queue. You'll need a context on the main thread whenever you have a data-driven user interface. Once the context has been initialized, it is configured to use the existing `_coordinator` pointer to the persistent store coordinator. Chapter 8 will demonstrate how to use multiple managed object contexts, including a background (private queue) concurrency type. For now, the main thread context will do.

Update Grocery Dude as follows to add the SETUP section:

1. Add the code from Listing 1.6 to the bottom of CoreDataHelper.m before @end.

The next method required in the SETUP section is loadStore and is shown in Listing 1.7.

Listing 1.7 **CoreDataHelper.m: loadStore**

```
- (void)loadStore {
if (debug==1) {
NSLog(@"Running %@ '%@'", self.class, NSStringFromSelector(_cmd));
}
    if (_store) {return;} // Don't load store if it's already loaded
    NSError *error = nil;
    _store = [_coordinator addPersistentStoreWithType:NSSQLiteStoreType
                                        configuration:nil
                                                  URL:[self storeURL]
                                              options:nil error:&error];
    if (!_store) {NSLog(@"Failed to add store. Error: %@", error);abort();}
    else         {if (debug==1) {NSLog(@"Successfully added store: %@", _store);}}
}
```

The loadStore method is straightforward. Once a check for an existing _store has been performed, a pointer to a nil NSError instance is created as error. This is then used when setting the _store instance variable to capture any errors that occur during setup. If _store is nil after an attempt to set it up fails, an error is logged to the console along with the content of the error.

When the SQLite persistent store is added via addPersistentStoreWithType, a pointer to the persistent store is held in _store. The storeURL of the persistent store is the one returned by the methods created previously.

Update Grocery Dude as follows to add to the SETUP section:

1. Add the code from Listing 1.7 to the bottom of CoreDataHelper.m before @end.

Finally, it's time to create the setupCoreData method. With the other supporting methods in place, this is a simple task. Listing 1.8 shows the contents of this new method, which at this stage only calls loadStore. This method will be expanded later in the book as more functionality is added.

Listing 1.8 **CoreDataHelper.m: setupCoreData**

```
- (void)setupCoreData {
if (debug==1) {
    NSLog(@"Running %@ '%@'", self.class, NSStringFromSelector(_cmd));
}
    [self loadStore];
}
```

Update Grocery Dude as follows to add to the SETUP section:

1. Add the code from Listing 1.8 to the bottom of CoreDataHelper.m before @end.

Saving

The next puzzle piece is a method called whenever you would like to save changes from the _context to the _store. This is as easy as sending the context a save: message, as shown in Listing 1.9. This method will be placed in a new SAVING section.

Listing 1.9 **CoreDataHelper.m: SAVING**

```
#pragma mark - SAVING
- (void)saveContext {
if (debug==1) {
    NSLog(@"Running %@ '%@'", self.class, NSStringFromSelector(_cmd));
}
    if ([_context hasChanges]) {
        NSError *error = nil;
        if ([_context save:&error]) {
            NSLog(@"_context SAVED changes to persistent store");
        } else {
            NSLog(@"Failed to save _context: %@", error);
        }
    } else {
        NSLog(@"SKIPPED _context save, there are no changes!");
    }
}
```

Update Grocery Dude as follows to add the SAVING section:

1. Add the code from Listing 1.9 to the bottom of CoreDataHelper.m before @end.

The Core Data Helper is now ready to go! To use it, a new property is needed in the application delegate header. The CoreDataHelper class also needs to be imported into the application delegate header, so it knows about this new class. The bold code shown in Listing 1.10 highlights the changes required to the application delegate header.

Listing 1.10 **AppDelegate.h**

```
#import <UIKit/UIKit.h>
#import "CoreDataHelper.h"
@interface AppDelegate : UIResponder <UIApplicationDelegate>
@property (strong, nonatomic) UIWindow *window;
@property (nonatomic, strong, readonly) CoreDataHelper *coreDataHelper;
@end
```

Update Grocery Dude as follows to add CoreDataHelper to the application delegate:

1. Replace all code in AppDelegate.h with the code from Listing 1.10.

The next step is to update the application delegate implementation with a small method called cdh, which returns a non-nil CoreDataHelper instance. In addition, a #define debug 1 statement needs to be added for debug purposes, as shown in Listing 1.11.

Listing 1.11 **AppDelegate.m: cdh**

```
#define debug 1

- (CoreDataHelper*)cdh {
if (debug==1) {
    NSLog(@"Running %@ '%@'", self.class, NSStringFromSelector(_cmd));
}
    if (!_coreDataHelper) {
        _coreDataHelper = [CoreDataHelper new];
        [_coreDataHelper setupCoreData];
    }
    return _coreDataHelper;
}
```

Update Grocery Dude as follows to add the cdh method to the application delegate:

1. Add the code from Listing 1.11 to AppDelegate.m on the line after @implementation AppDelegate.

The final step required is to ensure the context is saved each time the application enters the background or is terminated. This is an ideal time to save changes to disk because the user interface won't lag during save as it is hidden. Listing 1.12 shows the code involved in saving the context.

Listing 1.12 **AppDelegate.m: applicationDidEnterBackground**

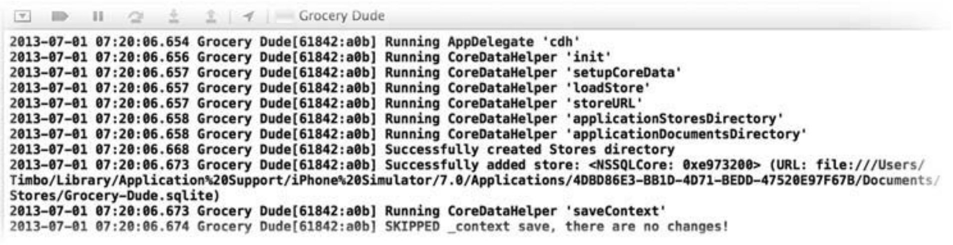
```
- (void)applicationDidEnterBackground:(UIApplication *)application {
    [[self cdh] saveContext];
}
- (void)applicationWillTerminate:(UIApplication *)application {
    [[self cdh] saveContext];
}
```

Update Grocery Dude as follows to ensure the context is saved when the application enters the background or is terminated:

1. Add [[self cdh] saveContext]; to the bottom of the applicationDidEnterBackground method in AppDelegate.m.

2. Add `[[self cdh] saveContext];` to the bottom of the `applicationWillTerminate` method in `AppDelegate.m`.

Run Grocery Dude on the iOS Simulator and examine the debug log window as you press the home button (**Shift+⌘+H** or **Hardware** > **Home**). The log is initially blank because Core Data is set up on demand using the `cdh` method of the application delegate. The first time Core Data is used is during the `save:` that's triggered when the application enters the background. As the application grows, the `cdh` method will be used earlier. Figure 1.5 shows the order of method execution once you press the home button.



```
2013-07-01 07:20:06.654 Grocery Dude[61842:a0b] Running AppDelegate 'cdh'
2013-07-01 07:20:06.656 Grocery Dude[61842:a0b] Running CoreDataHelper 'init'
2013-07-01 07:20:06.657 Grocery Dude[61842:a0b] Running CoreDataHelper 'setupCoreData'
2013-07-01 07:20:06.657 Grocery Dude[61842:a0b] Running CoreDataHelper 'loadStore'
2013-07-01 07:20:06.657 Grocery Dude[61842:a0b] Running CoreDataHelper 'storeURL'
2013-07-01 07:20:06.658 Grocery Dude[61842:a0b] Running CoreDataHelper 'applicationStoresDirectory'
2013-07-01 07:20:06.658 Grocery Dude[61842:a0b] Running CoreDataHelper 'applicationDocumentsDirectory'
2013-07-01 07:20:06.668 Grocery Dude[61842:a0b] Successfully created Stores directory
2013-07-01 07:20:06.673 Grocery Dude[61842:a0b] Successfully added store: <NSSQLCore: 0xe973200> (URL: file:///Users/
Timbo/Library/Application%20Support/iPhone%20Simulator/7.0/Applications/4DBD86E3-BB1D-4D71-BEDD-47520E97F67B/Documents/
Stores/Grocery-Dude.sqlite)
2013-07-01 07:20:06.673 Grocery Dude[61842:a0b] Running CoreDataHelper 'saveContext'
2013-07-01 07:20:06.674 Grocery Dude[61842:a0b] SKIPPED _context save, there are no changes!
```

Figure 1.5 The debug log window showing order of execution

Summary

You've now been introduced to the key components of Core Data. The sample application Grocery Dude has been updated to include an SQLite persistent store, persistent store coordinator, managed object model, and managed object context. The data model has not been configured, so the application isn't very interesting yet. Chapter 2 is where the real fun begins with the introduction of data models. If you're still unclear on the role some parts of Core Data play, don't worry too much at this stage. As you come to use each component more, it should become easier to understand how they fit together.

Exercises

Why not build on what you've learned by experimenting?

1. Add the following code to the top of each method in the application delegate to assist with debugging:

```
if (debug==1) {
    NSLog(@"Running %@ '%@'", self.class, NSStringFromSelector(_cmd));
}
```

2. Examine the console log to compare the different locations persistent store files are saved to when running the application on a device versus running on the iOS Simulator. This is useful information when it comes time to open the persistent store for troubleshooting.
3. Change the persistent store type in the `loadStore` method of `CoreDataHelper.m` from `NSSQLStoreType` to `NSXMLStoreType` and try running the application. You won't be able to run the application because this store type is not available on iOS.

This page intentionally left blank

Index

A

app icons, Grocery Dude sample app, 415

AppDelegate: demo (Deleting) listing (2.9), 43

AppDelegate: demo (Fetch Request Template) listing (2.8), 41

AppDelegate: demo (Filtering) listing (2.7), 40

AppDelegate.h: cdh listing (5.6), 104

AppDelegate.m: applicationDidEnterBackground listing (1.12), 15

AppDelegate.m: cdh listing (1.11), 15

AppDelegate.m: cdh listing (5.7), 105

AppDelegate.m: demo (Fetch Request) listing (2.4), 37

AppDelegate.m: demo (Fetching) listing (2.5), 37

AppDelegate.m: demo (Fetching Test Amount Data) listing (3.4), 55

AppDelegate.m: demo (Fetching Test Measurement Data) listing (3.3), 51

AppDelegate.m: demo (Fetching Test Unit Data) listing (3.10), 67

AppDelegate.m: demo (Inserting) listing (2.3), 32

AppDelegate.m: demo (Inserting Test Measurement Data) listing (3.2), 50

AppDelegate.m: demo listing (2.2), 31-32

AppDelegate.m: demo listing (5.11), 110-111

AppDelegate.m: demo (Relationship Creation) listing (4.1), 75-76

AppDelegate.m: demo (Sorting) listing (2.6), 38

AppDelegate.m: demo (Unit Deletion) listing (4.3), 79

AppDelegate.m: demo (Validate for Delete) listing (4.4), 80-81

AppDelegate.m: generateStackMobSchema listing (16.6), 390-392

AppDelegate.m listing (13.1), 300-301

AppDelegate.m: showUnitAndItemCount listing (4.2), 78

Application Stores Directory, 332

applicationDidBecomeActive method, 31

applications

- Core Data, adding to existing applications, 6-16
- Grocery Cloud sample app, 417, 423
  - disabling camera and image support, 421
  - renaming groups and tests, 419
  - renaming scheme, 420
  - repointing file paths, 418
  - updating artwork, 421
  - working around section name key path issue, 422-423
- Grocery Dude sample app, 5-6, 16, 411
  - app icons, 415
  - creating new Xcode project, 411
  - launch images, 416
  - renaming to Grocery Cloud, 417-418
  - storyboard design, 412-414
- profiling, 251
- StackMob, creating, 382

arrays, entities, 222

artwork, Grocery Cloud sample app, updating, 421

attribute data types

- Binary Data, 26
- Boolean, 25
- Date, 25
- Decimal, 25
- doubles, 24-27
- floats, 24-27
- integers, 23-24
- String, 25
- Transformable, 26-27

attributes, managed object model, 22-23

- settings, 28-29

authentication, StackMob, 394-406

- LoginVC class, 397-406
- user class, 396-397

automatic schema generation, StackMob, 390-392

B

backend SQL visibility, 33-36

Backend-as-a-Service (BaaS), StackMob, 379

background processing, 263, 268-269, 279

- background save, 263-268
- Faulter class, 269-273
- Thumbnailer class, 273-279

BACKUP & RESTORE section, DropboxHelper, 312-316

backups
displaying, 322-323
Dropbox, 297-298, 328
building DropboxHelper, 308-316
building DropboxTVC, 316-327
CoreDataHelper, 307-308
Integration, 298-307
performing, 319-321
Binary Data attribute data type, 26
Boolean attribute data type, 25
building, DropboxHelper, 308-316

C

camera
disabling, 421
implementing, 240-245
Cascade delete rule, 77
classes
CoreDataHelper, 7-8
FILES section, 9-10
Implementation, 9
PATHS section, 10-11
SAVING section, 14-16
SETUP section, 11-14
updating for iCloud, 334-341
Deduplicator, 351-360
DropboxHelper, 302-304
Faulter, 269-273
LoginVC, 397-406
Thumbnailer, 273-279
user, StackMob, 396-397
clean slate, reveting iCloud to, 367-369
clients, StackMob, configuring, 385-387
code listings
AppDelegate: demo (Deleting) (2.9), 43
AppDelegate: demo (Fetch Request Template) (2.8), 42
AppDelegate: demo (Filtering) (2.7), 40
AppDelegate.h (1.10), 14
AppDelegate.h: cdh (5.6), 104
AppDelegate.m (13.1), 300-301
AppDelegate.m: applicationDidEnterBackground (1.12), 15
AppDelegate.m: cdh (1.11), 15
AppDelegate.m: cdh (5.7), 105
AppDelegate.m: demo (2.2), 32
AppDelegate.m: demo (5.11), 110-111
AppDelegate.m: demo (Fetch Request) (2.4), 37
AppDelegate.m: demo (Fetching) (2.5), 37
AppDelegate.m: demo (Fetching Test Amount Data) (3.4), 55
AppDelegate.m: demo (Fetching Test Measurement Data) (3.3), 51
AppDelegate.m: demo (Fetching Test Unit Data) (3.10), 67
AppDelegate.m: demo (Inserting) (2.3), 32
AppDelegate.m: demo (Inserting Test Measurement Data) (3.2), 50
AppDelegate.m: demo (Relationship Creation) (4.1), 76
AppDelegate.m: demo (Sorting) (2.6), 38
AppDelegate.m: demo (Unit Deletion) (4.3), 79
AppDelegate.m: demo (Validate for Delete) (4.4), 81

AppDelegate.m: generateStackMobSchema (16.6), 390-392

AppDelegate.m: showUnitAndItemCount (4.2), 78

CoreDataHelper.h (1.1), 8

CoreDataHelper.h (9.1), 218

CoreDataHelper.h (14.2), 335

CoreDataHelper.h (15.8), 361

CoreDataHelper.h (16.1), 385

CoreDataHelper.h: deepCopyEntities (9.14), 231

CoreDataHelper.m: alertView:clickedButtonAtIndex (8.7), 198-199

CoreDataHelper.m: alertView:clickedButtonAtIndex (9.16), 233

CoreDataHelper.m: applicationStoresDirectory (1.4), 10-11

CoreDataHelper.m: backgroundSaveContext (11.2), 266

CoreDataHelper.m: checkIfDefaultDataNeedsImporting (8.2), 192

CoreDataHelper.m: confirmMergeWithiCloud (15.13), 365-366

CoreDataHelper.m: CORE DATA RESET (13.7), 307-308

CoreDataHelper.m: CORE DATA RESET (14.9), 343-344

CoreDataHelper.m: CORE DATA RESET (15.10), 362

CoreDataHelper.m: deepCopyFromPersistentStore (9.15), 232-233

CoreDataHelper.m: deepCopyFrom PersistentStore (9.18), 235

CoreDataHelper.m: DELEGATE: NSXMLParser (8.14), 206-208

CoreDataHelper.m: demo (Unit Deletion) (4.5), 81

CoreDataHelper.m: ensureAppropriateStoreIsLoaded (14.10), 344-345

CoreDataHelper.m: FILES (1.2), 10

CoreDataHelper.m: FILES (9.3), 219

CoreDataHelper.m: ICLOUD (14.6), 338-339

CoreDataHelper.m: ICLOUD RESET (15.14), 367-368

CoreDataHelper.m: iCloudAccountIsSignedIn (14.1), 334

CoreDataHelper.m: iCloudEnabledByUser (14.8), 342-343

CoreDataHelper.m: iCloudStoreURL (14.3), 336

CoreDataHelper.m: importFromXML (8.3), 194

CoreDataHelper.m: importGroceryDudeTestData (10.5), 260-261

CoreDataHelper.m: importGroceryDudeTestData (11.6), 272-273

CoreDataHelper.m: init (8.5), 197

CoreDataHelper.m: init (9.2), 219

CoreDataHelper.m: init (10.4), 248-249

CoreDataHelper.m: init (11.1), 264-265

CoreDataHelper.m: init (15.9), 361

CoreDataHelper.m: init (16.2), 385-386

CoreDataHelper.m: isDefaultDataAlreadyImportedFor-StoreWithURL (8.1), 191

CoreDataHelper.m: isMigration-NecessaryForStore (3.5), 60

CoreDataHelper.m: listenForStore-Changes (14.5), 338

CoreDataHelper.m: loadiCloudStore (14.4), 336-337

CoreDataHelper.m: loadiCloudStore (14.7), 340

CoreDataHelper.m: loadNoniCloud-StoreAsSeedStore (15.11), 363

CoreDataHelper.m: loadSourceStore (9.5), 220-221

CoreDataHelper.m: loadStore (1.7), 13

CoreDataHelper.m: loadStore (3.1), 49

CoreDataHelper.m: loadStore (3.9), 65

CoreDataHelper.m: mergeNoniCloudDataWithiCloud (15.12), 364-365

CoreDataHelper.m: migrateStore (3.6), 60-62

CoreDataHelper.m: observeValueForKeyPath and replaceStore (3.7), 62-63

CoreDataHelper.m: PATHS (1.3), 10

CoreDataHelper.m: PATHS (9.4), 220

CoreDataHelper.m: performBackgroundManaged-MigrationForStore (3.8), 64

CoreDataHelper.m: SAVING (1.9), 14

CoreDataHelper.m: SAVING (16.3), 387-388

CoreDataHelper.m: selectedUniqueAttributes (8.13), 205-206

CoreDataHelper.m: setDefaultDataAsImportedForStore (8.6), 197-198

CoreDataHelper.m: setDefaultDataStoreAsInitialStore (8.15), 212

CoreDataHelper.m: SETUP (1.6), 12

CoreDataHelper.m: setupCoreData (1.8), 13

CoreDataHelper.m: setupCoreData (10.3), 247

CoreDataHelper.m: setupCoreData (14.11), 346

CoreDataHelper.m: showValidationError (4.6), 83-84

CoreDataHelper.m: showValidationError (Additional Cases) (4.7), 85-86

CoreDataHelper.m: somethingChanged (9.17), 234

CoreDataHelper.m: storeURL (1.5), 11

CoreDataHelper.m: TEST DATA IMPORT (10.2), 246-247

CoreDataImporter.h (8.8), 200-201

CoreDataImporter.m: arrayForEntity (9.7), 222

CoreDataImporter.m: copyRelationshipsFromObject (9.12), 229-230

CoreDataImporter.m: copyUniqueObject (9.8), 223-224

CoreDataImporter.m: copyUniqueObject:toContext (15.7), 360

CoreDataImporter.m: deepCopyEntities (9.13), 231

CoreDataImporter.m: establishOrderedToManyRelationship (9.11), 227-228

CoreDataImporter.m: establishToManyRelationship (9.10), 226-227

CoreDataImporter.m: establishToOneRelationship (9.9), 224-225

CoreDataImporter.m: existingObjectInContext (8.10), 202-203

CoreDataImporter.m: insertBasicObjectInTargetEntity (8.12), 204-205

CoreDataImporter.m: insertUniqueObjectInTargetEntity (8.11), 203-204

CoreDataImporter.m: objectInfo (9.6), 221-222

CoreDataImporter.m: saveContext, initWithUniqueAttributes, uniqueAttributeForEntity (8.9), 201-202

CoreDataPickerTF.h (7.1), 160-161

CoreDataPickerTF.m: DATA (7.4), 165

CoreDataPickerTF.m: DELEGATE+DATASOURCE: UIPickerView (7.2), 163-164

CoreDataPickerTF.m: INTERACTION (7.3), 164

CoreDataPickerTF.m: VIEW (7.5), 166-167

CoreDataTVC.h (5.1), 95

CoreDataTVC.h (12.1), 282

CoreDataTVC.h: reloadSearchFRCForPredicate (12.7), 287

CoreDataTVC.h: searchFRC and searchDC (12.2), 282

CoreDataTVC.m: configureSearch (12.9), 288

CoreDataTVC.m Content Changes (5.4), 98

CoreDataTVC.m: DATASOURCE: UITableView (5.3), 97-98

CoreDataTVC.m: DATASOURCE: UITableView (12.4), 283-284

CoreDataTVC.m: DELEGATE: NSFetchedResultsController (5.5), 99-100

CoreDataTVC.m: DELEGATE: NSFetchedResultsController (12.5), 284-286

CoreDataTVC.m: DELEGATE: UISearchDisplayController (12.6), 286

CoreDataTVC.m: FETCHING (5.2), 95-96

CoreDataTVC.m: GENERAL (12.3), 283

CoreDataTVC.m: performFetch (16.13), 407

CoreDataTVC.m: reloadSearchFRCForPredicate (12.8), 287-288

CoreDataTVC.m: viewDidLoad (16.14), 408

Deduplicator.h: deDuplicateEntityWithName (15.3), 355

Deduplicator.m: deDuplicateEntityWithName (15.2), 354-355

Deduplicator.m: deDuplicateEntityWithName (15.5), 358-359

Deduplicator.m: duplicatesForEntityWithName (15.1), 353

Deduplicator.m: saveContextHierarchy (15.4), 358

Default Data XML Example Format (8.4), 195

DropboxHelper.h (13.11), 316

DropboxHelper.h (13.3), 302

DropboxHelper.m (13.4), 303

DropboxHelper.m: BACKUP / RESTORE (13.10), 312-315

DropboxHelper.m: DROPBOX FILE MANAGEMENT (13.9), 310-312
DropboxHelper.m: LOCAL FILE MANAGEMENT (13.8), 309
DropboxTVC.h (13.12), 316
DropboxTVC.h (13.5), 304
DropboxTVC.m (13.6), 305
DropboxTVC.m: BACKUP (13.15), 319-320
DropboxTVC.m: backup / restore (15.15), 371
DropboxTVC.m: DATA (13.13), 317-318
DropboxTVC.m: DATASOURCE: UITableView (13.16), 322-323
DropboxTVC.m: RESTORE (13.17), 325-327
DropboxTVC.m: VIEW (13.14), 318-319
Faulter.h (11.4), 270
Faulter.m (11.5), 270-271
Grocery Cloud Object Insertions (16.4), 388-389
Grocery Dude-Info.plist (13.2), 301-302
Item.h (2.1), 30
ItemVC.h (6.1), 128
ItemVC.h (7.7), 172
ItemVC.m: CAMERA (10.1), 243-244
ItemVC.m: DATA (6.6), 140-141
ItemVC.m: DELEGATE: UITextField (6.4), 136-137
ItemVC.m: INTERACTION (6.3), 135
ItemVC.m: INTERACTION (7.19), 186-187
ItemVC.m: PICKERS (7.9), 173-174
ItemVC.m: refreshInterface (7.11), 174-175
ItemVC.m: refreshInterface (7.16), 183
ItemVC.m: refreshInterface (11.10), 278
ItemVC.m: selectedObjectClearedFor-PickerTF (7.18), 185
ItemVC.m: selectedObjectID:changed-ForPickerTF (7.17), 184
ItemVC.m: textFieldDidBeginEditing (7.10), 174
ItemVC.m: textFieldDidBeginEditing (7.14), 182
ItemVC.m: VIEW (6.5), 138
ItemVC.m: viewDidDisappear (7.21), 188
ItemVC.m: viewDidDisappear (10.6), 262
ItemVC.m: viewDidLoad (7.15), 182
ItemVC.m: viewDidLoad (7.8), 172
ItemVC.m: viewWillAppear (7.20), 187
LocationAtHomePickerTF.m (7.12), 176-177
LocationAtHomeVC.m, LocationAtShopVC.m, and UnitVC.m: viewDidDisappear (11.3), 267
LocationAtShopPickerTF.m (7.13), 178-179
LoginVC.h (16.9), 397
LoginVC.m (16.10), 399-401
LoginVC.m: authenticate (16.12), 404-405
LoginVC.m: create (16.11), 402-404
PrepareTVC.h (5.8), 106
PrepareTVC.m, ShopTVC.m, UnitsTVC.m, LocationsAtHomeTVC.m, and LocationsAtShopTVC.m: viewWillAppear (16.5), 390
PrepareTVC.m: accessoryButtonTappedForRowWith-IndexPath (12.14), 293-294

PrepareTVC.m and ShopTVC.m: prepareForSegue and accessoryButtonTappedForRow WithIndexPath (6.2), 128-129

PrepareTVC.m and ShopTVC.m: viewDidAppear (11.9), 276-277

PrepareTVC.m: cellForRowAtIndexPath (12.11), 290-291

PrepareTVC.m: cellForRowAtIndexPath (B.1), 422-423

PrepareTVC.m: commitEditingStyle (12.12), 292

PrepareTVC.m: DATA (5.9), 106-107

PrepareTVC.m: didSelectRowAtIndexPath (12.13), 293

PrepareTVC.m: INTERACTION (5.13), 115-116

PrepareTVC.m: SEARCH (12.10), 289-290

PrepareTVC.m: VIEW (5.10), 108-109

PrepareTVC.m: VIEW (Selection and Deletion) (5.12), 112-113

PrepareTVC.m: viewDidAppear (15.6), 359

ShopTVC.m: DATA (5.14), 118

ShopTVC.m: INTERACTION (5.16), 121

ShopTVC.m: VIEW (5.15), 119-120

Thumbnailer.h (11.7), 274

Thumbnailer.m (11.8), 275-276

UnitPickerTF.m (7.6), 168-169

UnitsTVC.m (6.7), 147-149

UnitsTVC.m: prepareForSegue (6.10), 153-154

UnitVC.h (6.8), 150

UnitVC.m (6.9), 150-152

User.h (16.7), 396

User.m (16.8), 396

configuration

Core Data, 369-370

home location views, 154-156

shop location views, 156-157

source stack, 218-221

StackMob clients, 385-387

connections

LocationAtShopPickerTF, 180-181

UnitPickerTF, 171

context, import, creating, 196-197

copy relationships, 228-230

copyRelationshipsFromObject method, 228-230

copyUniqueObject method, 223-224

Core Data, 1-4, 16

adding to existing applications, 6-16

configurations, 369-370

entity mapping, 21

table views, 93

CoreDataTVC, 93-103

when to use, 5

CoreDataHelper class, 7-8, 307-308

FILES section, 9-10

implementation, 9

PATHS section, 10-11

SAVING section, 14-16

SETUP section, 11-14

updating for iCloud, 334-341

CoreDataHelper instance, 104-105

CoreDataHelper.h: deepCopyEntities listing (9.14), 231

CoreDataHelper.h listing (1.1), 8

CoreDataHelper.h listing (9.1), 218

CoreDataHelper.h listing (14.2), 335

CoreDataHelper.h listing (15.8), 361

CoreDataHelper.h listing (16.1), 385

CoreDataHelper.m: alertView:clickedButtonAtIndex listing (8.7), 198-199

CoreDataHelper.m: alertView:clickedButtonAtIndex listing (9.16), 233

CoreDataHelper.m: applicationStores-Directory listing (1.4), 10-11

CoreDataHelper.m: backgroundSave-Context listing (11.2), 266

CoreDataHelper.m: checkIfDefaultDataNeedsImporting listing (8.2), 192

CoreDataHelper.m: confirmMergeWithiCloud listing (15.13), 365-366

CoreDataHelper.m: CORE DATA RESET listing (13.7), 307-308

CoreDataHelper.m: CORE DATA RESET listing (14.9), 343-344

CoreDataHelper.m: CORE DATA RESET listing (15.10), 362

CoreDataHelper.m: deepCopyFromPersistentStore listing (9.15), 232-233

CoreDataHelper.m: deepCopyFromPersistentStore listing (9.18), 235

CoreDataHelper.m: DELEGATE: NSXMLParser listing (8.14), 206-208

CoreDataHelper.m: demo (Unit Deletion) listing (4.5), 81

CoreDataHelper.m: ensureAppropriateStoreIsLoaded listing (14.10), 344-345

CoreDataHelper.m: FILES listing (1.2), 8

CoreDataHelper.m: FILES listing (9.3), 219

CoreDataHelper.m: ICLOUD listing (14.6), 338-339

CoreDataHelper.m: ICLOUD RESET listing (15.14), 367-368

CoreDataHelper.m: iCloudAccountIsSignedIn listing (14.1), 334

CoreDataHelper.m: iCloudEnabledByUser listing (14.8), 342-343

CoreDataHelper.m: iCloudStoreURL listing (14.3), 336

CoreDataHelper.m: importFromXML listing (8.3), 194

CoreDataHelper.m: importGroceryDudeTestData listing (10.5), 260-261

CoreDataHelper.m: importGroceryDudeTestData listing (11.6), 272-273

CoreDataHelper.m: init listing (8.5), 197

CoreDataHelper.m: init listing (9.2), 219

CoreDataHelper.m: init listing (10.4), 248-249

CoreDataHelper.m: init listing (11.1), 264-265

CoreDataHelper.m: init listing (15.9), 361

CoreDataHelper.m: init listing (16.2), 385-386

CoreDataHelper.m: isDefaultDataAlreadyImportedForStore-WithURL listing (8.1), 191

CoreDataHelper.m: isMigrationNecessary-ForStore listing (3.5), 60

CoreDataHelper.m: listenForStoreChanges listing (14.5), 338

CoreDataHelper.m: loadiCloudStore listing (14.4), 336-337

CoreDataHelper.m: loadiCloudStore listing (14.7), 340

CoreDataHelper.m: loadNoniCloudStoreAsSeedStore listing (15.11), 363

CoreDataHelper.m: loadSourceStore listing (9.5), 220-221

CoreDataHelper.m: loadStore listing (1.7), 13

CoreDataHelper.m: loadStore listing (3.1), 49

CoreDataHelper.m: loadStore listing (3.9), 65

CoreDataHelper.m: mergeNoniCloudDataWithiCloud listing (15.12), 364-365

CoreDataHelper.m: migrateStore listing (3.6), 60-62

CoreDataHelper.m: observeValueForKeyPath and replaceStore listing (3.7), 62-63

CoreDataHelper.m: PATHS listing (1.3), 10

CoreDataHelper.m: PATHS listing (9.4), 220

CoreDataHelper.m: performBackgroundManagedMigrationForStore listing (3.8), 64

CoreDataHelper.m: SAVING listing (1.9), 14

CoreDataHelper.m: SAVING listing (16.3), 387-388

CoreDataHelper.m: selectedUniqueAttributes listing (8.13), 205-206

CoreDataHelper.m: setDefaultDataAsImportedForStore listing (8.6), 197-198

CoreDataHelper.m: setDefaultDataStoreAsInitialStore listing (8.15), 212

CoreDataHelper.m: SETUP listing (1.6), 11

CoreDataHelper.m: setupCoreData listing (1.8), 13

CoreDataHelper.m: setupCoreData listing (10.3), 247

CoreDataHelper.m: setupCoreData listing (14.11), 346

CoreDataHelper.m: showValidationError (Additional Cases) listing (4.7), 83-86

CoreDataHelper.m: somethingChanged listing (9.17), 234

CoreDataHelper.m: storeURL listing (1.5), 11

CoreDataHelper.m: TEST DATA IMPORT listing (10.2), 246-247

CoreDataImporter, 200-205
- enhancing, 221-232
- entity attributes, mapping XML data to, 206-208
- selecting unique attributes, 205-206

CoreDataImporter.h listing (8.8), 200-201

CoreDataImporter.m: arrayForEntity listing (9.7), 222

CoreDataImporter.m: copyRelationshipsFromObject listing (9.12), 229-230

CoreDataImporter.m: copyUniqueObject listing (9.8), 223-224

CoreDataImporter.m: copyUniqueObject:toContext listing (15.7), 360

CoreDataImporter.m: deepCopyEntities listing (9.13), 231

CoreDataImporter.m: establishOrderedToManyRelationship listing (9.11), 227-228

CoreDataImporter.m: establishToManyRelationship listing (9.10), 226-227

CoreDataImporter.m: establishToOneRelationship listing (9.9), 224-225

CoreDataImporter.m: existingObjectInContext listing (8.10), 202-203

CoreDataImporter.m: insertBasicObjectInTargetEntity listing (8.12), 204-205

CoreDataImporter.m: insertUniqueObjectInTargetEntity listing (8.11), 203-204

CoreDataImporter.m: objectInfo listing (9.6), 221-222

CoreDataImporter.m: saveContext, initWithUniqueAttributes, uniqueAttributeForEntity listing (8.9), 201-202

CoreDataPickerTF, 160-168

DATA section, 165

INTERACTION section, 164-165

VIEW section, 165-168

CoreDataPickerTF.h listing (7.1), 160-161

CoreDataPickerTF.m: DATA listing (7.4), 165

CoreDataPickerTF.m: DELEGATE+DATASOURCE: UIPickerView listing (7.2), 163-164

CoreDataPickerTF.m: INTERACTION listing (7.3), 164

CoreDataPickerTF.m: VIEW listing (7.5), 166-167

CoreDataTVC table view, 93-103

DATASOURCE: UITableView, 96-98

FETCHING section, 95-96

NSFetchedResultsControllerDelegate protocol, 96-98

updating to support search, 282-289

CoreDataTVC.h listing (5.1), 95

CoreDataTVC.h listing (12.1), 282

CoreDataTVC.h: reloadSearchFRCForPredicate listing (12.7), 287

CoreDataTVC.h: searchFRC and searchDC listing (12.2), 282

CoreDataTVC.m: configureSearch listing (12.9), 288

CoreDataTVC.m: DATASOURCE: UITableView listing (5.3), 97-98

CoreDataTVC.m: DATASOURCE: UITableView listing (12.4), 283-284

CoreDataTVC.m: DELEGATE: NSFetchedResultsController listing (5.5), 99-100

CoreDataTVC.m: DELEGATE: NSFetchedResultsController listing (12.5), 284-286

CoreDataTVC.m: DELEGATE: UISearchDisplayController listing (12.6), 286

CoreDataTVC.m: FETCHING listing (5.2), 95-96

CoreDataTVC.m: GENERAL listing (12.3), 283

CoreDataTVC.m: performFetch listing (16.13), 407

CoreDataTVC.m: reloadSearchFRCForPredicate listing (12.8), 287-288

CoreDataTVC.m: viewDidLoad listing (16.14), 408

D

data

backups

displaying, 322-323

Dropbox, 297-328

performing, 319-321

default, 189-190

importing, 190-192

preventing duplicate, 197-198

importing, 213
CoreDataImporter, 200-208, 221-232
creating import context, 196-197
from persistent stores, 209-213
importing from XML, 193-195
restoring, 324-327
test, generating, 245-249
DATA section
CoreDataPickerTF, 165
ItemVC table view, 139-142
PrepareTVC table view, 106-107
ShopTVC table view, 117-118
Date attribute data type, 25
Debug Navigator, iCloud, 341-342
Decimal attribute data type, 25
Deduplicator class, 351-360
Deduplicator.h: deDuplicateEntityWithName listing (15.3), 355
Deduplicator.m: deDuplicateEntityWithName listing (15.2), 354-355
Deduplicator.m: deDuplicateEntityWithName listing (15.5), 358-359
Deduplicator.m: duplicatesForEntityWithName listing (15.1), 353
Deduplicator.m: saveContextHierarchy listing (15.4), 358
deep copy, 215-218, 236
configuring source stack, 218-221
CoreDataImporter, enhancing, 221-232
entities, 230-232
triggering, 232-236
default data, 189-190
importing, 190-192
triggering, 198-200
preventing duplicate, 197-198
using default data store as initial store, 210-213
Default Data XML Example Format listing, 195
default migration, managed object model, 52-56
delete rules, objects, 77-81
deleting managed objects, 42-43
demo method, 32
Deny delete rule, 77, 79-80
disabling iCloud, 342-348
displaying data backups, 322-323
doubles, 24-27
Dropbox, 297-298, 328
DropboxHelper class, 302-304
building, 308-316
DropboxTVC, 304-307
building, 316-327
iCloud, 370-371
integration, 298-307
linking to, 300-302
preparing CoreDataHelper, 307-308
supporting frameworks, 299-300
DROPBOX FILE MANAGEMENT section, DropboxHelper, 310-312
DropboxHelper class, 302-304
BACKUP & RESTORE section, 312-316
building, 308-316
DROPBOX FILE MANAGEMENT section, 310-312
LOCAL FILE MANAGEMENT section, 308-309

DropboxHelper.h listing (13.11), 316
DropboxHelper.h listing (13.3), 302
DropboxHelper.m: BACKUP / RESTORE listing (13.10), 312-315
DropboxHelper.m: DROPBOX FILE MANAGEMENT listing (13.9), 310-312
DropboxHelper.m listing (13.4), 303
DropboxHelper.m: LOCAL FILE MANAGEMENT listing (13.8), 309
DropboxTVC table view, 304-307
  building, 316-327
DropboxTVC.h listing (13.12), 316
DropboxTVC.h listing (13.5), 304
DropboxTVC.m: BACKUP listing (13.15), 319-320
DropboxTVC.m: backup / restore listing (15.15), 371
DropboxTVC.m: DATA listing (13.13), 317-318
DropboxTVC.m: DATASOURCE: UITableView listing (13.16), 322-323
DropboxTVC.m listing (13.6), 305
DropboxTVC.m: RESTORE listing (13.17), 325-327
DropboxTVC.m: VIEW listing (13.14), 318-319
duplicate default data, preventing, 197-198

E

editing units, ItemVC, 145-147
entities
  arrays, 222
  deep copy, 230-232
  inheritance, 86-90
  managed object model, 20-22
  relationships, 71-76
    creating, 75-76
entity attributes, CoreDataImporter, mapping XML data to, 206-208
establishOrderedToManyRelationship method, 227-228
establishToManyRelationship method, 226-227
establishToOneRelationship method, 224-225

F

Faulter class, 269-273
Faulter.h listing (11.4), 270
Faulter.m listing (11.5), 270-271
fetch request filtering, 39-40
fetch request sorting, 38-39
fetch request templates, 40-42
fetching
  managed objects, 36-42
    fetch request sorting, 38-39
FETCHING section, CoreDataTVC, 95-96
file paths, renaming, 418
FILES section, CoreDataHelper class, 9-10
floats, 24-27

G

Grocery Cloud Object Insertions listing (16.4), 388-389
Grocery Cloud sample app, 417, 423
  disabling camera and image support, 421
  Grocery Dude sample app, renaming, 417-418
  renaming groups and tests, 419
  renaming scheme, 420
  repointing file paths, 418
  updating artwork, 421
  working around section name key path issue, 422-423

Grocery Dude sample app, 5-6, 16, 411
app icons, 415
creating new Xcode project, 411
launch images, 416
renaming to Grocery Cloud, 417-418
storyboard design, 412-414
updating, 7-8
Grocery Dude-Info.plist listing (13.2), 301-302
groups, renaming, 419

H-K

hierarchy, table views, 126-127
home location views
adding and editing, 153-154
configuring, 154-156
iCloud, 331-333, 346, 351, 371
Debug Navigator, 341-342
de-duplication, 351-360
disabling, 342-348
Dropbox backup, 370-371
enabling, 333
iCloud Store, 335-337
notifications, 337-341
reverting to clean slate, 367-369
seeding, 360-366
updating CoreDataHelper for, 334-341
identifying performance issues, 239-240
importing data, 213
CoreDataImporter, 200-208, 221-232
creating import context, 196-197
default, 190-192, 198-200
from persistent stores, 209-213
from XML, 193-195
inheritance, entities, 86-90
Instruments, measuring performance, 251-252
integers, managed object model, ranges, 23-24
INTERACTION section
CoreDataPickerTF, 164-165
ItemVC table view, 134-137
PrepareTVC table view, 114-116
ShopTVC table view, 121-122
iOS SDK, StackMob, 381-382
Item.h listing (2.1), 30
ItemVC, 127-144
adding and editing units, 145-147
DATA section, 139-142
implementation, 134
INTERACTION section, 134-137
LocationAtShopPickerTF, configuring for, 181-185
passing selected items to, 128-129
UnitPickerTF, configuring for, 171-176
VIEW section, 137-139
ItemVC.h (7.7) listing, 172
ItemVC.h listing (6.1), 128
ItemVC.m: CAMERA listing (10.1), 243-244
ItemVC.m: DATA listing (6.6), 140-141
ItemVC.m: DELEGATE: UITextField listing (6.4), 136-137
ItemVC.m: INTERACTION listing (6.3), 135
ItemVC.m: INTERACTION listing (7.19), 186-187
ItemVC.m: PICKERS listing (7.9), 173-174
ItemVC.m: refreshInterface listing (7.11), 174-175
ItemVC.m: refreshInterface listing (7.16), 183
ItemVC.m: refreshInterface listing (11.10), 278

ItemVC.m: selectedObjectClearedForPickerTF listing (7.18), 185

ItemVC.m: selectedObjectID:changedFor-PickerTF listing (7.17), 184

ItemVC.m: textFieldDidBeginEditing listing (7.10), 174

ItemVC.m: textFieldDidBeginEditing listing (7.14), 182

ItemVC.m: VIEW listing (6.5), 138

ItemVC.m: viewDidDisappear listing (7.21), 188

ItemVC.m: viewDidDisappear listing (10.6), 262

ItemVC.m: viewDidLoad listing (7.15), 182

ItemVC.m: viewDidLoad listing (7.8), 172

ItemVC.m: viewWillAppear listing (7.20), 187

L

launch images, Grocery Dude sample app, 416

lightweight migration, managed object model, 48-51

linking to Dropbox, 300-302

listings

AppDelegate: demo (Deleting) (2.9), 43

AppDelegate: demo (Fetch Request Template) (2.8), 42

AppDelegate: demo (Filtering) (2.7), 40

AppDelegate.h (1.10), 14

AppDelegate.h: cdh (5.6), 104

AppDelegate.m (13.1), 300-301

AppDelegate.m: applicationDidEnterBackground (1.12), 15

AppDelegate.m: cdh (1.11), 15

AppDelegate.m: cdh (5.7), 105

AppDelegate.m: demo (2.2), 32

AppDelegate.m: demo (5.11), 110-111

AppDelegate.m: demo (Fetch Request) (2.4), 37

AppDelegate.m: demo (Fetching) (2.5), 37

AppDelegate.m: demo (Fetching Test Amount Data) (3.4), 55

AppDelegate.m: demo (Fetching Test Measurement Data) (3.3), 51

AppDelegate.m: demo (Fetching Test Unit Data) (3.10), 67

AppDelegate.m: demo (Inserting) (2.3), 32

AppDelegate.m: demo (Inserting Test Measurement Data) (3.2), 50

AppDelegate.m: demo (Relationship Creation) (4.1), 76

AppDelegate.m: demo (Sorting) (2.6), 38

AppDelegate.m: demo (Unit Deletion) (4.3), 79

AppDelegate.m: demo (Validate for Delete) (4.4), 81

AppDelegate.m: generateStackMobSchema (16.6), 390-392

AppDelegate.m: showUnitAndItemCount (4.2), 78

CoreDataHelper.h (1.1), 8

CoreDataHelper.h (9.1), 218

CoreDataHelper.h (14.2), 335

CoreDataHelper.h (15.8), 361

CoreDataHelper.h (16.1), 385

CoreDataHelper.h: deepCopyEntities (9.14), 231

CoreDataHelper.m: alertView:clickedButtonAtIndex (8.7), 198-199

CoreDataHelper.m: alertView:clickedButtonAtIndex (9.16), 233

CoreDataHelper.m: applicationStoresDirectory (1.4), 10-11

CoreDataHelper.m: backgroundSaveContext (11.2), 266

CoreDataHelper.m: checkIfDefaultDataNeedsImporting (8.2), 192

CoreDataHelper.m: confirmMergeWithiCloud (15.13), 365-366

CoreDataHelper.m: CORE DATA RESET (13.7), 307-308

CoreDataHelper.m: CORE DATA RESET (14.9), 343-344

CoreDataHelper.m: CORE DATA RESET (15.10), 362

CoreDataHelper.m: deepCopyFromPersistentStore (9.15), 232-233

CoreDataHelper.m: deepCopyFromPersistentStore (9.18), 235

CoreDataHelper.m: DELEGATE: NSXMLParser (8.14), 206-208

CoreDataHelper.m: demo (Unit Deletion) (4.5), 81

CoreDataHelper.m: ensureAppropriateStoreIsLoaded (14.10), 344-345

CoreDataHelper.m: FILES (1.2), 10

CoreDataHelper.m: FILES (9.3), 219

CoreDataHelper.m: ICLOUD (14.6), 338-339

CoreDataHelper.m: ICLOUD RESET (15.14), 367-368

CoreDataHelper.m: iCloudAccountIsSignedIn (14.1), 334

CoreDataHelper.m: iCloudEnabledByUser (14.8), 342-343

CoreDataHelper.m: iCloudStoreURL (14.3), 336

CoreDataHelper.m: importFromXML (8.3), 194

CoreDataHelper.m: importGroceryDudeTestData (10.5), 260-261

CoreDataHelper.m: importGroceryDudeTestData (11.6), 272-273

CoreDataHelper.m: init (8.5), 197

CoreDataHelper.m: init (9.2), 219

CoreDataHelper.m: init (10.4), 248-249

CoreDataHelper.m: init (11.1), 264-265

CoreDataHelper.m: init (15.9), 361

CoreDataHelper.m: init (16.2), 385-386

CoreDataHelper.m: isDefaultDataAlreadyImportedFor-StoreWithURL (8.1), 191

CoreDataHelper.m: isMigrationNecessaryForStore (3.5), 60

CoreDataHelper.m: listenForStoreChanges (14.5), 338

CoreDataHelper.m: loadiCloudStore (14.4), 336-337

CoreDataHelper.m: loadiCloudStore (14.7), 340

CoreDataHelper.m: loadNoniCloudStoreAsSeedStore (15.11), 363

CoreDataHelper.m: loadSourceStore (9.5), 220-221

CoreDataHelper.m: loadStore (1.7), 13

CoreDataHelper.m: loadStore (3.1), 49

CoreDataHelper.m: loadStore (3.9), 65

CoreDataHelper.m: mergeNoniCloudDataWithiCloud (15.12), 364-365

CoreDataHelper.m: migrateStore (3.6), 60-62

CoreDataHelper.m: observeValueForKeyPath and replaceStore (3.7), 62-63

CoreDataHelper.m: PATHS (1.3), 10

CoreDataHelper.m: PATHS (9.4), 220

CoreDataHelper.m: performBackgroundManaged-MigrationForStore (3.8), 64

CoreDataHelper.m: SAVING (1.9), 14

CoreDataHelper.m: SAVING (16.3), 387-388

CoreDataHelper.m: selectedUniqueAttributes (8.13), 205-206

CoreDataHelper.m: setDefaultDataAsImportedForStore (8.6), 197-198

CoreDataHelper.m: setDefaultDataStoreAsInitialStore (8.15), 212

CoreDataHelper.m: SETUP (1.6), 12

CoreDataHelper.m: setupCoreData (1.8), 13

CoreDataHelper.m: setupCoreData (10.3), 247

CoreDataHelper.m: setupCoreData (14.11), 346

CoreDataHelper.m: showValidationError (4.6), 83-84

CoreDataHelper.m: showValidationError (Additional Cases) (4.7), 85-86

CoreDataHelper.m: somethingChanged (9.17), 234

CoreDataHelper.m: storeURL (1.5), 11

CoreDataHelper.m: TEST DATA IMPORT (10.2), 246-247

CoreDataImporter.h (8.8), 200-201

CoreDataImporter.m: arrayForEntity (9.7), 222

CoreDataImporter.m: copyRelationshipsFromObject (9.12), 229-230

CoreDataImporter.m: copyUniqueObject (9.8), 223-224

CoreDataImporter.m: copyUniqueObject:toContext (15.7), 360

CoreDataImporter.m: deepCopyEntities (9.13), 231

CoreDataImporter.m: establishOrderedToManyRelationship (9.11), 227-228

CoreDataImporter.m: establishToManyRelationship (9.10), 226-227

CoreDataImporter.m: establishToOneRelationship (9.9), 224-225

CoreDataImporter.m: existingObjectInContext (8.10), 202-203

CoreDataImporter.m: insertBasicObjectInTargetEntity (8.12), 204-205

CoreDataImporter.m: insertUniqueObjectInTargetEntity (8.11), 203-204

CoreDataImporter.m: objectInfo (9.6), 221-222

CoreDataImporter.m: saveContext, initWithUniqueAttributes, uniqueAttributeForEntity (8.9), 201-202

CoreDataPickerTF.h (7.1), 160-161

CoreDataPickerTF.m: DATA (7.4), 165
CoreDataPickerTF.m: DELEGATE+DATASOURCE: UIPickerView (7.2), 163-164
CoreDataPickerTF.m: INTERACTION (7.3), 164
CoreDataPickerTF.m: VIEW (7.5), 166-167
CoreDataTVC.h (5.1), 95
CoreDataTVC.h (12.1), 282
CoreDataTVC.h: reloadSearchFRCForPredicate (12.7), 287
CoreDataTVC.h: searchFRC and searchDC (12.2), 282
CoreDataTVC.m: configureSearch (12.9), 288
CoreDataTVC.m Content Changes (5.4), 98
CoreDataTVC.m: DATASOURCE: UITableView (5.3), 97-98
CoreDataTVC.m: DATASOURCE: UITableView (12.4), 283-284
CoreDataTVC.m: DELEGATE: NSFetchedResultsController (5.5), 99-100
CoreDataTVC.m: DELEGATE: NSFetchedResultsController (12.5), 284-286
CoreDataTVC.m: DELEGATE: UISearchDisplayController (12.6), 286
CoreDataTVC.m: FETCHING (5.2), 95-96
CoreDataTVC.m: GENERAL (12.3), 283
CoreDataTVC.m: performFetch (16.13), 407
CoreDataTVC.m: reloadSearchFRCForPredicate (12.8), 287-288
CoreDataTVC.m: viewDidLoad (16.14), 408
Deduplicator.h: deDuplicateEntityWithName (15.3), 355
Deduplicator.m: deDuplicateEntityWithName (15.2), 354-355
Deduplicator.m: deDuplicateEntityWithName (15.5), 358-359
Deduplicator.m: duplicatesForEntityWithName (15.1), 353
Deduplicator.m: saveContextHierarchy (15.4), 358
Default Data XML Example Format (8.4), 195
DropboxHelper.h (13.11), 316
DropboxHelper.h (13.3), 302
DropboxHelper.m (13.4), 303
DropboxHelper.m: BACKUP / RESTORE (13.10), 312-315
DropboxHelper.m: DROPBOX FILE MANAGEMENT (13.9), 310-312
DropboxHelper.m: LOCAL FILE MANAGEMENT (13.8), 309
DropboxTVC.h (13.12), 316
DropboxTVC.h (13.5), 304
DropboxTVC.m (13.6), 305
DropboxTVC.m: BACKUP (13.15), 319-320
DropboxTVC.m: backup / restore (15.15), 371
DropboxTVC.m: DATA (13.13), 317-318
DropboxTVC.m: DATASOURCE: UITableView (13.16), 322-323

DropboxTVC.m: RESTORE (13.17), 325-327
DropboxTVC.m: VIEW (13.14), 318-319
Faulter.h (11.4), 270
Faulter.m (11.5), 270-271
Grocery Cloud Object Insertions (16.4), 388-389
Grocery Dude-Info.plist (13.2), 301-302
Item.h (2.1), 30
ItemVC.h (6.1), 128
ItemVC.h (7.7), 172
ItemVC.m: CAMERA (10.1), 243-244
ItemVC.m: DATA (6.6), 140-141
ItemVC.m: DELEGATE: UITextField (6.4), 136-137
ItemVC.m: INTERACTION (6.3), 135
ItemVC.m: INTERACTION (7.19), 186-187
ItemVC.m: PICKERS (7.9), 173-174
ItemVC.m: refreshInterface (7.11), 174-175
ItemVC.m: refreshInterface (7.16), 183
ItemVC.m: refreshInterface (11.10), 278
ItemVC.m: selectedObjectClearedForPickerTF (7.18), 185
ItemVC.m: selectedObjectID:changed-ForPickerTF (7.17), 184
ItemVC.m: textFieldDidBeginEditing (7.10), 174
ItemVC.m: textFieldDidBeginEditing (7.14), 182
ItemVC.m: VIEW (6.5), 138
ItemVC.m: viewDidDisappear (7.21), 188
ItemVC.m: viewDidDisappear (10.6), 262
ItemVC.m: viewDidLoad (7.15), 182
ItemVC.m: viewDidLoad (7.8), 172
ItemVC.m: viewWillAppear (7.20), 187
LocationAtHomePickerTF.m (7.12), 176-177
LocationAtHomeVC.m, LocationAtShopVC.m, and UnitVC.m: viewDidDisappear (11.3), 267
LocationAtShopPickerTF.m (7.13), 178-179
LoginVC.h (16.9), 397
LoginVC.m (16.10), 399-401
LoginVC.m: authenticate (16.12), 404-405
LoginVC.m: create (16.11), 402-404
PrepareTVC.h (5.8), 106
PrepareTVC.m, ShopTVC.m, UnitsTVC.m, LocationsAtHomeTVC.m, and LocationsAtShopTVC.m: viewWillAppear (16.5), 390
PrepareTVC.m: accessoryButtonTappedForRow-WithIndexPath (12.14), 293-294
PrepareTVC.m and ShopTVC.m: prepareForSegue and accessory-ButtonTappedForRowWithIndexPath (6.2), 128-129
PrepareTVC.m and ShopTVC.m: viewDidAppear (11.9), 276-277
PrepareTVC.m: cellForRowAtIndexPath (12.11), 290-291
PrepareTVC.m: cellForRowAtIndexPath (B.1), 422-423
PrepareTVC.m: commitEditingStyle (12.12), 292
PrepareTVC.m: DATA (5.9), 106-107
PrepareTVC.m: didSelectRowAtIndexPath (12.13), 293

PrepareTVC.m: INTERACTION (5.13), 115-116

PrepareTVC.m: SEARCH (12.10), 289-290

PrepareTVC.m: VIEW (5.10), 108-109

PrepareTVC.m: VIEW (Selection and Deletion) (5.12), 112-113

PrepareTVC.m: viewDidAppear (15.6), 359

ShopTVC.m: DATA (5.14), 118

ShopTVC.m: INTERACTION (5.16), 121

ShopTVC.m: VIEW (5.15), 119-120

Thumbnailer.h (11.7), 274

Thumbnailer.m (11.8), 275-276

UnitPickerTF.m (7.6), 168-169

UnitsTVC.m (6.7), 147-149

UnitsTVC.m: prepareForSegue (6.10), 153-154

UnitVC.h (6.8), 150

UnitVC.m (6.9), 150-152

User.h (16.7), 396

User.m (16.8), 396

LOCAL FILE MANAGEMENT section, DropboxHelper, 308-309

LocationAtHomePickerTF, 176-177

LocationAtHomePickerTF.m listing (7.12), 176-177

LocationAtHomeVC.m, LocationAtShopVC.m, and UnitVC.m: viewDidDisappear listing (11.3), 267

LocationAtShopPickerTF, 178-185

configuring ItemVC for, 181-185

connecting to, 180-181

creating, 179-180

LocationAtShopPickerTF.m (7.13), 178-179

LoginVC class, 397-406

LoginVC.h listing (16.9), 397

LoginVC.m: authenticate listing (16.12), 404-405

LoginVC.m: create listing (16.11), 402-404

LoginVC.m listing (16.10), 399-401

M

managed object context, 4

managed object model, 2, 4, 19, 43, 71

adding, 20, 47-48

attribute settings, 28-29

attributes, 22-23

backend SQL visibility, 33-36

changing, 45-46

creating managed objects, 32-33

doubles, 24-27

entities, 20-22

entity inheritance, 86-90

fetching managed objects, 36-42

floats, 24-27

integer ranges, 23-24

migration, 45, 69

default, 52-56

lightweight, 48-51

migration manager, 57-68

NSManagedObject subclass, 29-31

primitive data types, scalar properties, 31

relationships, 71-76

creating, 75-76

delete rules, 77-81

StackMob, preparing for, 383-384

validation errors, 81-86

managed objects
cleaning up, 261-262
deep copy, 215-218
configuring source stack, 218-219
deleting, 42-43
fetch request sorting, 38-39
mapping XML data to entity attributes, 206-208
measuring performance
with Instruments, 251-252
with SQLDebug, 250-251
methods
applicationDidBecomeActive, 31
copyRelationshipsFromObject, 228-230
copyUniqueObject, 223-224
deepCopyEntities, 230-232
demo, 32
establishOrderedToManyRelationship, 227-228
establishToManyRelationship, 226-227
establishToOneRelationship, 224-225
objectInfo, 221-222
showValidationError, 84
viewDidDisappear, 262
migration, managed object model, 45, 69
default, 52-56
lightweight, 48-51
migration manager, 57-68
View Controller, 57-59

N-O

No Action delete rule, 77
notifications, iCloud, 337-341
Nullify delete rule, 77
objectInfo method, 221-222
objects
managed
creating, 32-33
deep copy, 215-218
deleting, 42-43
fetching, 36-42
managed object model, 19, 43, 71
adding, 20, 47-48
attributes, 22-23
backend SQL visibility, 33-36
changing, 45-46
doubles, 24-27
entities, 20-22
entity inheritance, 86-90
floats, 24-27
integer ranges, 23-24
migration, 45, 48-56, 69
NSManagedObject subclass, 29-31
relationships, 71-76
copy, 228-230
creating, 75-76
delete rules, 77-81
to-many, 226-227
to-one, 224-225
ordered to-many, 227-228
validation errors, 81-86
ordered to-many relationships, establishing, 227-228

P-Q

PATHS section, CoreDataHelper class, 10-11
performance issues, 239, 262
generating test data, 245-249
identifying, 239-240

improving performance, 254-261
with Instruments, 251-252
with SQLDebug, 250-251
persistent store coordinator, 2-3
persistent stores, 3
importing data from, 209-213
picker views, 159, 188
CoreDataPickerTF, 160-168
DATA section, 165
INTERACTION section, 164-165
VIEW section, 165-168
LocationAtHomePickerTF, 176-177
LocationAtShopPickerTF, 178-185
configuring ItemVC for, 181-185
connecting to, 180-181
creating, 179-180
picker-avoiding text field, 186-188
UnitPickerTF, 168-176
configuring ItemVC for, 171-176
connecting to, 171
creating, 170
preloading data, 189
default data, 189-190
PrepareTVC table view, 105-116
DATA section, 106-107
INTERACTION section, 114-116
updating to support search, 289-294
PrepareTVC.h listing (5.8), 106
PrepareTVC.m, ShopTVC.m, UnitsTVC.m, LocationsAtHomeTVC.m, and LocationsAtShopTVC.m: viewWillAppear listing (16.5), 390
PrepareTVC.m: accessoryButtonTappedForRowWith-IndexPath listing (12.14), 293-294
PrepareTVC.m and ShopTVC.m: prepareForSegue and accessoryButtonTappedForRow-WithIndexPath listing (6.2), 128-129
PrepareTVC.m and ShopTVC.m: viewDidAppear listing (11.9), 276-277
PrepareTVC.m: cellForRowAtIndexPath listing (12.11), 290-291
PrepareTVC.m: cellForRowAtIndexPath listing (B.1), 422-423
PrepareTVC.m: commitEditingStyle listing (12.12), 292
PrepareTVC.m: DATA listing (5.9), 106-107
PrepareTVC.m: didSelectRowAtIndexPath listing (12.13), 293
PrepareTVC.m: INTERACTION listing (5.13), 115-116
PrepareTVC.m: SEARCH listing (12.10), 289-290
PrepareTVC.m: VIEW (5.10) listing, 108-109
PrepareTVC.m: VIEW (Selection and Deletion) listing (5.12), 112-113
PrepareTVC.m: viewDidAppear listing (15.6), 359
preventing duplicate default data, 197-198
processing
background, 263, 268-269, 279
save, 263-268
Thumbnailer class, 273-279
profiling applications, 251

R

relationships
copy, 228-230
delete rules, 77-81
entities, 71-76
creating, 75-76

to-many, establishing, 226-227
to-one, establishing, 224-225
ordered to-many, establishing, 227-228
REST (Representational State Transfer), 380
restoring data, Dropbox, 324-327

S

SAVING section, CoreDataHelper class, 14-16
schema generation, StackMob, 390-392
schema permissions, StackMob, 393
Scroll View, configuring, 129-134
SDK, StackMob, 381-382
Search, 281, 294
CoreDataTVC, updating to support, 282-289
PrepareTVC, updating to support, 289-294
seeding, iCloud, 360-366
selected items, keeping reference to, 128
SETUP section, CoreDataHelper class, 11-14
shop location views
adding and editing, 153-154
configuring, 156-157
ShopTVC table view, 117-122
DATA section, 117-118
INTERACTION section, 121-122
VIEW section, 118-121
ShopTVC.m: DATA listing (5.14), 118
ShopTVC.m: INTERACTION listing (5.16), 121
ShopTVC.m: VIEW listing (5.15), 119-120
showValidationError method, 84
SQLDebug, measuring performance, 250-251
StackMob, 379-381, 408
applications, creating, 382
authentication, 394-406
LoginVC class, 397-406
user class, 396-397
automatic schema generation, 390-392
clients, configuring, 385-387
iOS SDK, 381-382
maintaining responsiveness, 406-408
managed object model, preparing, 383-384
saving, 387-389
schema permissions, 393
underlying changes, 389-390
storyboard design, Grocery Dude sample app, 412-414
String attribute data type, 25

T

table views, 91, 93
CoreDataHelper instance, 104-105
CoreDataTVC, 93-103
DATASOURCE: UITableView, 96-98
FETCHING section, 95-96
NSFetchedResultsController-Delegate protocol, 96-98
updating to support search, 282-289
DropboxTVC, 304-307
building, 316-327
fundamental components, 91
hierarchy, 126-127
home location views
adding and editing, 153-154
configuring, 154-156

- PrepareTVC, 105-116
  - DATA section, 106-107
  - INTERACTION section, 114-116
  - updating to support search, 289-294
- shop location views
  - adding and editing, 153-154
  - configuring, 156-157
- ShopTVC, 117-122
  - DATA section, 117-118
  - INTERACTION section, 121-122
  - VIEW section, 118-121

test data, generating, 245-249

tests, renaming, 419

text fields, configuring, 129-134

Thumbnailer class, 273-279

Thumbnailer.h listing (11.7), 274

Thumbnailer.m listing (11.8), 275-276

to-many relationships, establishing, 226-227

to-one relationships, establishing, 224-225

Transformable attribute data type, 26-27

triggering
- deep copy, 232-236
- default data import, 198-200

U

Ubiquity Container, 332

UITableViewController subclass, 92

UnitPickerTF, 168-176
- connecting to, 171
- creating, 170
- ItemVC, configuring for, 171-176

UnitPickerTF.m listing (7.6), 168-169

units, ItemVC, adding and editing, 145-147

UnitsTVC
- Implementing, 147-149
- seguing from, 153-154

UnitsTVC.m listing (6.7), 147-149

UnitsTVC.m: prepareForSegue listing (6.10), 153-154

UnitVC
- implementing, 149-152
- seguing from, 153-154

UnitVC.h listing (6.8), 150

UnitVC.m listing (6.9), 150-152

updating
- CoreDataTVC, search support, 282-289
- Grocery Dude, 7-8
- PrepareTVC, search support, 289-294

user class, StackMob, 396-397

User.h listing (16.7), 396

User.m listing (16.8), 396

V

validation errors, objects, 81-86

VIEW section
- CoreDataPickerTF, 165-168
- ItemVC, 137-139
- PrepareTVC table view, 108-113
- ShopTVC table view, 118-121

viewDidDisappear method, 262

views, 125-126, 158
- ItemVC, 127-144
  - adding and editing units, 145-147
  - passing selected items to, 128-129
  - VIEW section, 137-139

- picker, 159, 188
  - CoreDataPickerTF, 160-168
  - LocationAtHomePickerTF, 176-177
  - LocationAtShopPickerTF, 178-185
  - picker-avoiding text field, 186-188
  - UnitPickerTF, 168-176
- Scroll View, configuring, 129-134
- table views, 91-93
  - adding and editing home and shop location, 153-154
  - CoreDataTVC, 93-103
  - hierarchy, 126-127
  - PrepareTVC, 105-116
  - ShopTVC, 117-121
- UnitsTVC
  - implementing, 147-149
  - seguing from, 153-154
- UnitVC
  - implementing, 149-152
  - seguing from, 153-154

W

web services, StackMob, 379-381, 408
- authentication, 394-406
- automatic schema generation, 390-392
- configuring clients, 385-387
- creating applications, 382
- iOS SDK, 381-382
- maintaining responsiveness, 406-408
- managed object model preparation, 383-384
- saving, 387-389
- schema permissions, 393
- underlying changes, 389-390

X-Z

Xcode, Grocery Dude sample app, creating new project, 411

Xcode 5, 5

XML, importing data from, 193-195