Learning iCloud
Data Management
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.
## Contents at a Glance

<table>
<thead>
<tr>
<th>Section</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preface</td>
<td>xvii</td>
</tr>
<tr>
<td>Acknowledgments</td>
<td>xxiii</td>
</tr>
<tr>
<td>About the Author</td>
<td>xxv</td>
</tr>
<tr>
<td>Introduction</td>
<td>1</td>
</tr>
<tr>
<td><strong>I Introducing iCloud</strong></td>
<td>3</td>
</tr>
<tr>
<td>1 Exploring iCloud and Its User Experience</td>
<td>5</td>
</tr>
<tr>
<td>2 Setting Up iCloud for Development</td>
<td>17</td>
</tr>
<tr>
<td><strong>II Using the APIs</strong></td>
<td>33</td>
</tr>
<tr>
<td>3 Introducing the APIs and the First Apps</td>
<td>35</td>
</tr>
<tr>
<td>4 Working with the AddressBook API for Contacts</td>
<td>57</td>
</tr>
<tr>
<td>5 Managing Calendars and Reminders with the Event Kit API</td>
<td>69</td>
</tr>
<tr>
<td>6 Protecting the Privacy of User Data</td>
<td>87</td>
</tr>
<tr>
<td><strong>III Using the Technologies</strong></td>
<td>95</td>
</tr>
<tr>
<td>7 Introducing Blocks, Threads, and Notifications</td>
<td>97</td>
</tr>
<tr>
<td>8 Using Key-Value Coding (KVC)</td>
<td>105</td>
</tr>
<tr>
<td>9 Using Preferences, Settings, and Keychains with iCloud</td>
<td>121</td>
</tr>
<tr>
<td>10 Managing Persistent Storage with Core Data</td>
<td>133</td>
</tr>
<tr>
<td>11 Using Xcode Workspaces for Shared Development</td>
<td>157</td>
</tr>
<tr>
<td>12 Adding Data to Apps with Bundles and Resources</td>
<td>169</td>
</tr>
</tbody>
</table>
Contents at a Glance

IV Using iCloud Documents and Data  185
13 Adding the iCloud Infrastructure  187
14 Working with File Wrappers in iCloud  231
15 Working with iOS Documents  273
16 Working with OS X Documents  317
17 Working with Core Data and iCloud  339
18 Completing the Round Trip  349

Index  379
Contents

Preface xvii
Acknowledgments xxiii
About the Author xxv

Introduction 1

1 Introducing iCloud 3

1 Exploring iCloud and Its User Experience 5
Looking at Cloud Computing 5
Understanding the iCloud Paradigm 7
Organizing Files by App 8
Managing Documents with iCloud, Time Machine, and Auto Save 12
Syncing Data Across Devices 13
Making the Round Trip 14
Chapter Summary 14
Exercises 15

2 Setting Up iCloud for Development 17
Managing App Security on iOS and OS X 18
Identifying Yourself and Your App on developer.apple.com 18
Identifying Your User and Your Ubiquity Container at Runtime 22
Looking Inside the iCloud Basics 23
Apple ID 24
Bundle Identifier 26
Entitlements and Capabilities 28
Ubiquity Container 30
Using iCloud in Your App 30
Chapter Summary 31
Exercises 32
## Contents

### II Using the APIs  
33  
3 **Introducing the APIs and the First Apps** 35  
   Getting Started as an Apple Developer  35  
   Looking at the APIs  37  
   Introducing the Built-In Data Apps  38  
      Keeping Up with Apple  38  
      App Overview  40  
      Creating Separate Xcode Projects for iOS and OS X  41  
   Wiring Up the Interfaces  50  
      Wiring Up the iOS Interface  51  
      Wiring Up the OS X Interfaces  54  
   Chapter Summary  55  
   Exercises  55  
4 **Working with the AddressBook API for Contacts** 57  
   Considering the AddressBook API on iOS and OS X  57  
   Sending Mail from the iOS App  58  
      Making Sure You Can Send Mail  59  
      Sending the Message  60  
      Checking That Mail Is Configured and the Internet Is Available  63  
   Sending Mail from the OS X App  65  
   Using Property Lists for Storing and Syncing  65  
   Chapter Summary  66  
   Exercises  67  
5 **Managing Calendars and Reminders with the Event Kit API**  69  
   Exploring the Event Class Hierarchy  70  
   Setting OS X Permissions  71  
   Working with the Calendar Database  72  
      Allocating and Getting Access to the Event Store  72  
      Creating a New Event or Reminder  75  
      Searching for an Event or Reminder  76  
      Setting or Modifying Properties  77  
      Committing Changes  79
Adding a Reminder to the App on iOS 80
Adding an Event to the App on OS X 83
Chapter Summary 85
Exercises 85

6 Protecting the Privacy of User Data 87
The Need for Privacy 87
Looking at Apple’s Rules and Guidelines 88
Best Practices in App Privacy 88
  Know What Should Be Private 88
  Use Good Programming Style to Enforce Privacy 89
  Be Careful When Debugging 89
  Ask Permission and Explain What You’ll Do with the Data 90
  Do Not Require Personal Data to Unlock Your App 91
  Add Extra Measures to Protect Minors 91
  Provide Privacy for Support Materials 91
  Consider User Issues 92
Chapter Summary 93
Exercises 93

III Using the Technologies 95

7 Introducing Blocks, Threads, and Notifications 97
Catching Up with Blocks and Threads 98
  Queues and Threads 98
  Blocks 99
Getting Up to Speed with Notifications 100
  Notification Properties 101
  Registering for Notifications 101
  Posting Notifications 102
Receiving Notification of iCloud Availability Changes 102
Introducing the Second Project 103
Getting Ready to Move On 103
Chapter Summary 104
Exercises 104
Contents

Including a Core Data Store  183
Chapter Summary  184
Exercises  184

IV Using iCloud Documents and Data  185
13 Adding the iCloud Infrastructure  187
Exploring the Workspace for the App  188
  Exploring iOS and OS X Document Architecture Differences  190
  Dealing with UI Differences  191
  Designing the Shared App Folder Structure  191
  Checking Out the End Result  192
  Scoping the Project  194
Debugging iCloud Apps with developer.icloud.com  195
Building the App  199
Creating the Shared Folder  201
  Constants.h  201
  Constants.m  201
  SharediCloudController.h  202
  SharediCloudController.m  204
Creating the App’s Classes  215
  AppDelegate  215
  MasterViewController  217
  DetailViewController  224
  ReportDocument  227
Storyboards  230
Chapter Summary  230
Exercises  230

14 Working with File Wrappers in iCloud  231
Exploring Files, File Wrappers, and Documents  231
  Looking at Files  232
  Exploring File Wrappers  232
  Exploring Documents  233
How Users Manage iCloud Files  233
Contents

How the Developer Sees the Round Trip 362
   Using developer.icloud.com 362
   Using Xcode 364
Configuring the Shared Ubiquity Container 366
Using a Shared iCloud Controller 368
   Making the App Delegate Link to the Controller 369
   Declaring the Shared iCloud Controller 369
   Implementing the Shared iCloud Controller 370
Moving Documents to iCloud 376
Moving Documents from iCloud to Local Storage 377
Chapter Summary 378
Exercises 378

Index 379
Preface

When Apple announces new products or new versions of its operating systems, there is usually a big press event, and frequently there are lines of people waiting at Apple stores. There’s generally a pattern to these announcements. In the case of the operating systems, the major announcements are made at the Apple Worldwide Developers Conference in June. In some years, developer previews of one or both operating systems are made available earlier in the spring. Over the course of the summer, developer releases are made available. Rumors of the availability of the new iPhone begin circulating, and, sometime in the fall, Apple sends invitations to a media event to be held in a week. At that event, a new version of iOS is shown to the public along with a new iPhone. The public release of iOS comes a week later, followed by the availability of the new iPhone. Later (often the following month) the process is repeated for the iPad, Macs, and OS X.

This has been the schedule over the past few years, but there is no guarantee it will be repeated. What is important to note is that there are specific dates for the announcement and release of the products and operating systems. iCloud is a very different matter. Over a number of years, Apple has built a significant hardware and telecommunications support structure to power iCloud and its other network operations. As is the case with many such infrastructures, the details of it are kept confidential. We know the location of some of Apple’s data centers because they often require building permits and other public documents and permissions, but they are usually kept out of the public view. There has been no ribbon cutting or turning of a key to launch iCloud—it has been a years-long process (and it will continue for years to come).

In addition to the hardware infrastructure, iCloud has a software component. However, that, too, has been a years-long development process. As you will see in this book, parts of iCloud are implemented in the user interface of the operating systems, and other parts of it are implemented with relatively small changes to existing frameworks and APIs. For developers as well as consumers, public announcements about iCloud have been part of the announcements of new versions of the operating systems as well as of hardware.

In short, iCloud is not a product: it’s a pervasive technology and a companywide strategy for Apple. Unlike Apple’s hardware and software products, iCloud has no part number and no version. It is part of products across the company.
For that reason, it is not easy to write about iCloud or to learn to develop for it. This book was first envisioned in early 2012, but as it took shape, it became clear that some of the most powerful pieces of iCloud were not yet in place. Rather than rushing out a partial book and relying on the possibility of a revised edition sometime in the future, Trina MacDonald and Addison-Wesley agreed to push back the publication date so as to include the information from WWDC in June 2013, and I’m very grateful to them for doing that.

As you will see, the book culminates in what I call the iCloud Round Trip. In the final chapter, you’ll see how to build an iOS app and an OS X app that let you share data via iCloud on both OS X and iOS. Having the tools to be able to implement the Round Trip seems to me to be a good time to publish the book. That’s as close to a product launch event as you can get in the world of iCloud.

Who Should Read This Book

This book is written for developers who want to explore iCloud. Because iCloud is implemented in so many areas of the operating systems, you need a bit of familiarity with many parts of Cocoa and Cocoa Touch. As the book presents iCloud, an attempt has been made to at least summarize the various components that it touches. This means that the discussion of a topic such as notifications is at a fairly high level: some people will think “everyone knows that” and other people may think that more details are needed.

The attempt has been to provide a medium road for both experts and novices in the various Cocoa technologies that interact with iCloud. Apple’s documentation on developer.apple.com provides the primary resource for more details if you feel you need them. If you hit an area where you feel that you already know the topic, feel free to skip to the details of iCloud. Even among engineers at Apple, there are many areas of Cocoa that they know inside out (and may have written) and other areas with which they’re not familiar.

In terms of skills and knowledge, you should have a basic knowledge of Cocoa and/or Cocoa Touch as well as of Xcode. Objective-C is a must for understanding the code. The author’s Sams Teach Yourself Objective-C in 24 Hours provides an introduction to that topic.

In addition, you should have experience in using iCloud. It is always amazing how many people attempt to develop for a technology that they have not used. There’s nothing like hands-on user experience.

Downloading the Example Files

The example files for each chapter that has them can be downloaded from the author’s site at http://northcountryconsulting.com and from http://informit.com/title/9780321889119. In addition to the examples, you will find any updates and
corrections on both sites. Some of the downloadable examples contain additional code, such as an iPad interface in addition to the iPhone interface for Chapter 14, “Working with File Wrappers in iCloud.”

The files are arranged by chapter, and they represent the code as of the end of the chapter. Thus, in the cases where one chapter builds on the previous chapter’s code, download the previous chapter and work through it to add the new chapter’s code.

iCloud requires code signing, so you’ll see in this book how to set up your projects to accomplish that. Note that the code in this book and in the downloadable code contains code signing that will not work on your computer. You must use your own developer credentials. Rather than leaving the code signing information blank, I have used my own credentials (the password is not provided, and even the developer account name has been changed). This means that the code will not run unless you customize it for your own developer account. This is deliberate and necessary.

The code has been written against Xcode 5.0 and OS X Mavericks (10.9).

**How This Book Is Organized**

There are four parts to this book.

**Part I: Introducing iCloud**

The first part provides perspectives on iCloud from the user’s point of view and from that of the developer.

- Chapter 1, “Exploring iCloud and Its User Experience”: As iCloud has evolved, it has been incorporated into apps such as the iWork suite. You’ll see the user interface aspects of iCloud for apps and the operating systems.
- Chapter 2, “Setting Up iCloud for Development”: This chapter provides an overview of the API structure of iCloud. It’s a roadmap to the rest of the book.

**Part II: Using the APIs**

This part explores how you use iCloud data that the user enters and maintains. For many users, iCloud plays some role with the storage of their music and with the synchronization of their calendars and contacts. There are APIs that allow developers to tap into this synchronized user data, and they are described in this part of the book. This use of iCloud can reap big payoffs for the developer: the engineers at Apple and the users have done all the work—all you have to do is empower the users to employ their own data in new and imaginative ways.

- Chapter 3, “Introducing the APIs and the First Apps”: The simplest part of iCloud consists of the APIs that manage user data. This chapter provides the roadmap to this part of the book.
Chapter 4, “Working with the AddressBook API for Contacts”: The AddressBook API lets developers access and update address book data. This chapter shows you the basics of doing so.

Chapter 5, “Managing Calendars and Reminders with the Event Kit API”: You’ll see how to leverage calendars and reminders in this chapter.

Chapter 6, “Protecting the Privacy of User Data”: iCloud brings up many privacy issues that you need to address in your apps. This is user data, and you have to play by the rules described in this chapter.

Part III: Using the Technologies

Various data management technologies and design patterns are integrated with iCloud. Using these technologies can mean that your apps can take the most advantage of iCloud synchronization. These technologies are integrated with iCloud, but they existed long before iCloud came to be. It’s the integration that’s new.

Chapter 7, “Introducing Blocks, Threads, and Notifications”: This chapter provides a roadmap to the technologies in the context of iCloud. Even if you know the technologies, it’s important to review them in the iCloud world.

Chapter 8, “Using Key-Value Coding (KVC)”: Key-value coding has been used in Cocoa for years. It’s a very efficient way of storing relatively small amounts of data. And it works very easily for you and your users with iCloud.

Chapter 9, “Using Preferences, Settings, and Keychains with iCloud”: Preferences (OS X) and Settings (iOS) are a special case of key-value coding. This chapter shows how you can add them to your apps so that they apply to all of a user’s devices. You’ll also see how to exclude certain preferences and settings from iCloud if they don’t make sense for a specific device.

Chapter 10, “Managing Persistent Storage with Core Data”: Core Data is the major data persistence tool in Cocoa and Cocoa Touch. This chapter provides a high-level overview. It is followed on by Chapter 17, “Working with Core Data and iCloud.”

Chapter 11, “Using Xcode Project Workspaces for Shared Development”: Introduced in Xcode 4, Xcode workspaces make it easy to set up multiple targets within a project and to share certain files among the targets. For example, this will enable you to share a Core Data data model (schema) and its specific managed object classes with an OS X/iOS Round Trip.

Chapter 12, “Adding Data to Apps with Bundles and Resources”: This is one of the most general ways of managing data in apps. It doesn’t use iCloud directly, but it may be an appropriate addition to an iCloud app to complement iCloud-synchronized data.
Part IV: Using iCloud Documents and Data

The final part of the book brings together the APIs and technologies in documents and file wrappers. You’ll see how to implement them on OS X as well as on iOS. In addition, you’ll see how to complete a Round Trip as the documents synchronize across iOS and OS X.

- Chapter 13, “Adding the iCloud Infrastructure”: This chapter shows you the basic infrastructure to use with iCloud—the code to establish contact with iCloud, manage changes in iCloud availability, and make iCloud account changes. Note that this is code that will need to be implemented in any of the following chapters. In order to focus on the specific issues of the following chapters in this part of the book, it is not repeated in them.

- Chapter 14, “Working with File Wrappers in iCloud”: File wrappers implement a structure akin to packages in the finder: a collection of files that appear to be a single file to the user. They are a very efficient structure to take advantage of iCloud synchronization.

- Chapter 15, “Working with iOS Documents”: This chapter provides the iOS document model based on UIDocument. You’ll see how to monitor changes in your iCloud documents in real time.

- Chapter 16, “Working with OS X Documents”: On OS X, Cocoa takes care of the changes in iCloud documents for you, so you have less work to do than in Chapter 15. However, there is still work to be done, and this chapter shows you how to use NSDocument to accomplish what is necessary.

- Chapter 17, “Working with Core Data and iCloud”: This chapter provides you with the code you’ll need to manage Core Data-based apps with iCloud. It builds on Chapter 10.

- Chapter 18, “Completing the Round Trip”: Finally, you’ll see how to put together a Round Trip. Remember to add the code from Chapter 13 to both of your targets (OS X and iOS).
This page intentionally left blank
Acknowledgments

As always, Carole Jelen at Waterside Productions provided help and guidance in bringing this book to fruition. At Addison-Wesley, Trina MacDonald helped move this book along from idea to publication. Michael Thurston provided excellent editorial advice. The production manager, Julie Nahil, kept things moving along in the very complicated process of creating a technical book. Anna Popick, the freelance project manager, and Carol Lallier, freelance copy editor, contributed mightily to the book’s development. The elegant cover design is by Chuti Prasertsith.

Notwithstanding the help of these and many other people, any errors are the author’s.
This page intentionally left blank
About the Author

**Jesse Feiler** is a developer and author. He has been an Apple developer since before it became fashionable. His books include *Sams Teach Yourself Core Data for Mac and iOS in 24 Hours* (Sams Publishing, 2011), *Sams Teach Yourself Objective-C in 24 Hours* (Sams Publishing, 2012), *FileMaker 12 in Depth* (Que Publishing, 2012), and *iWork for Dummies* (Wiley, 2012).

Jesse has written about Objective-C and the Apple frameworks beginning with *Rhapsody Developer’s Guide* (Academic Press, 1997) and *Mac OS X Developer’s Guide* (Morgan Kaufmann, 2001). His books on Apple technologies such as Cyberdog, OpenDoc, ODF, Bento (in both incarnations), and Apple Guide occupy a special place on the shelf of developer books.

He is the author of Minutes Machine, the meeting management app for iPad, as well as the Saranac River Trail app for iPhone and iPad. They are available on the App Store; more details are available at champlainarts.com.

A native of Washington, DC, Jesse has lived in New York City and currently lives in Plattsburgh, New York, where he serves on the board of the Plattsburgh Public Library and as chair of the Saranac River Trail Advisory Committee.

He can be reached at http://northcountryconsulting.com.
This page intentionally left blank
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 us 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 we cannot help you with technical problems related to the topic of this book and that due to the high volume of mail we receive, we 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 or email address.

Email: trina.macdonald@pearson.com
Mail: Reader Feedback
Addison-Wesley Learning Series
800 East 96th Street
Indianapolis, IN 46240 USA

Reader Services
Visit our website and register this book at informit.com/register for convenient access to any updates, downloads, or errata that might be available for this book.
This page intentionally left blank
Cocoa and Cocoa Touch consist of frameworks that contain classes as well as protocols, defined constants, and some other supporting items including dynamic sharable libraries. The most basic frameworks are Foundation, UIKit (Cocoa Touch—iOS) and AppKit (Cocoa—OS X). More specialized frameworks, such as the Core Audio Kit Framework, are used as needed by developers.

iCloud is different. Don’t search for an iCloud framework: there is none. Don’t even search for an iCloud API. There are a couple iCloud-specific methods, but they are few and far between. In fact, they’re very far between in the sense that they are scattered across various classes and frameworks. `URLForPublishingUbiquitousItemAtURL:expirationDate:error:` is part of the `NSFileManager` class (there are seven iCloud-related methods among the 52 methods in this class), while `NSPersistentStoreDidImportUbiquitousContentChangesNotification` is part of the `NSPersistentStoreCoordinator` class (it is one of two iCloud-related notifications in this class).

The implementation of iCloud in this way means that existing apps that don’t use iCloud aren’t affected. In addition, because iCloud spans multiple devices as well as both operating systems (OS X and iOS), it is hard to imagine how it could have been implemented in a single framework or API.

Along with these few additions to the Cocoa and Cocoa Touch APIs, the implementation of iCloud relies on long-time best practices, which now have been converted to essential practices. Design patterns such as key-value coding that date back to the very early versions of NeXTSTEP have been used for a quarter of a century now, and they are used in new ways in iCloud, although in most cases you don’t have to do anything to take advantage of the iCloud functionality.

Core Data, which has long been the most powerful solution to managing an app’s persistent data, is deeply integrated with iCloud. However, that integration is largely (but not totally) done behind the scenes. If you don’t use iCloud, your existing Core Data code is just fine. Perhaps the most significant impact of iCloud on Core Data is that, in the past, there were two ways of creating a data store that could be distributed with an app. You could place a seed database in the app’s bundle, or you could add seed
data programmatically to an empty data store that you create the first time the app launches (or whenever the seed data needs to be recreated). Both techniques have been used for years. The biggest impact that iCloud has on Core Data is that with iCloud, the second technique needs to be used; the first one will not work properly. This is scarcely a major change.

Perhaps the most visible impact of iCloud on developers is the enhancement of entitlements that control what an app can do in its runtime environment. Entitlements implement the new sandboxing rules that come into play with shared documents on iCloud. Explicit entitlements and sandboxing define the functions and capabilities of the operating system that an app will use along with the specific parts of disk storage where the app can write data. They increase the stability and security of both operating systems. They are required on iOS and are optional on OS X. On both operating systems, they are more aggressively implemented. Furthermore, from a developer’s point of view, you’ll probably be happy to hear that the developer-facing interface for entitlements in Xcode 5 is now vastly changed and dramatically simplified. (Sandboxing is related to iCloud, but they are two separate functionalities.)

The implementation of iCloud has proceeded over several years; in mid-2012, the release of OS X Mountain Lion (10.8) and iOS 6 brought together some of the pieces that had been released over the previous year. In the fall of 2013, OS X Mavericks (10.9) and iOS 7 refined iCloud and expanded its behind-the-scenes tools for developers. If you have not used any of the iWork apps (Numbers, Pages, and Keynote), try one of them on multiple iCloud-enabled devices. They provide the best demonstration of iCloud from the user’s point of view.

Actually, that statement is wrong. They provide the best demonstration of iCloud from the user’s point of view—until you write your iCloud-enabled app.
This page intentionally left blank
In Chapter 1, “Exploring iCloud and Its User Interface,” you saw how iCloud looks to users. For many users, it’s just a logical way of working, and iCloud really isn’t an issue that they think about. For other users who have become used to managing their own files and folders on their various devices and desktops, there can be a significant effort at familiarization—an unlearning process. As technology advances, these unlearning events happen from time to time, but in the long run, the old way of doing things is forgotten. You may have had to configure a dial-up modem with bit rates and parity settings, for example. Now, even dial-up modems are automated and managed with handshakes to adjust their own settings. Dial-up modems are (fortunately) becoming artifacts of the past. And reports are surfacing of mystified children who don’t know what a computer mouse is in the world of touchscreens that they inhabit.

This chapter looks at iCloud from the developer’s perspective. As noted in the Introduction, iCloud isn’t a monolithic API or framework that you just plug into your code. It’s a collection of additions and modifications to many parts of Cocoa and Cocoa Touch. In this chapter, you’ll find a high-level view of those additions and modifications with particular emphasis on setting up iCloud in your app.

iCloud involves synchronizing data across a user’s devices, and as soon as you start thinking about sharing data among various devices, you have to consider the security issues involved. Fortunately, the engineers at Apple have done this: iCloud takes advantage of the security mechanisms that are built into the App Store and the Mac App Store. App security has not changed dramatically over time; however, configuring security has been difficult for many developers. In part, this is because it is a relatively complex process that, for most developers, is done relatively infrequently. That combination is a classic recipe for difficulty.

With the introduction of Xcode 5 in 2013, the implementation and setup of app security has changed insofar as the developer interface is involved (the underlying security mechanism is not changed). The changes make it easier for you to set up your app’s security, but for many developers, it is a new process. Once again, there is an
unlearning process involved when you use the new and simpler tools. Because iCloud requires security to be in place and because the way in which you implement it is changed, this chapter begins with an introduction to the new, improved, 2013 version of app security.

Managing App Security on iOS and OS X

The heart of the app security system is digitally signing your code with two digital signatures. Both of these signatures are generated by Apple, and each one references the other. (This is a common security mechanism that you can read about on Wikipedia in articles such as “Code Signing,” which explains the process.) These signatures will not match if either the one identifying Apple or the one identifying the developer has been altered; in addition, part of the digital signature contains a checksum mechanism that causes the security system to fail if the code has been altered since it was signed.

Part of the complexity arises from the use of these digital signatures. The security for apps is built on a combination of developer.apple.com tools, Xcode tools, and Keychain Access tools. It is important to note that Apple IDs are used to identify people, and there are two categories of people (thus Apple IDs) that come into play: During development, you as a developer have an Apple ID. At runtime, the user’s Apple ID comes into play with iCloud.

Identifying Yourself and Your App on developer.apple.com

Along with changes in Xcode 5, during 2013 developer.apple.com was revisited to consolidate the process of managing certificates, identifiers, devices, and provisioning profiles. These are the components of the security system for apps on iOS and OS X. Although the terminology hasn’t changed, the layout of developer.apple.com has changed. Furthermore, with Xcode 5, it is easier to manage these security features, but the process is slightly changed.

Here is an overview of the process. It is required for you to set up your app to use iCloud as well as to ultimately distribute it.

After you register on developer.apple.com, go to Certificates, Identifiers & Profiles (currently at the right of developer.apple.com). It handles security for both your iOS and OS X apps. However, you can now do this through Xcode 5 and later: it’s much easier there.

- You must identify yourself as a registered developer with a signing identity.
- You must identify your app with an App ID.
- You must identify the devices that you want to use for testing your app during development.
- You must create a provisioning profile that brings together your developer ID, your app ID, and the IDs for your test devices.
Managing Your Developer Signing Identity

A key part of the security mechanism is the certificate that you can download for each signing identity that you create. That is the link between developer.apple.com and Keychain Access on your Mac. The certificate is a portable and secure representation of your signing identity. In developer documentation, you may find the terms used interchangeably.

You can manage your signing identities in Xcode or on developer.apple.com. Xcode 5 introduced the ability to manage one or more Apple IDs for developer accounts, as you can see in Figure 2.1. A given Apple ID does not uniquely identify a specific developer account because a developer can be invited to join one or more development teams. You normally continue to use your developer Apple ID even though you may be working on several teams. Figure 2.1 shows the simplest scenario: a single developer Apple ID working on a single team. For iOS and Mac, one developer can have different roles.

![Figure 2.1 Manage your accounts on Xcode](image-url)
Perhaps the most important point to take away is that you must use your own Apple ID to avoid compromising the security system for apps. Use the tools described on developer.apple.com to manage development teams so that developers can be assigned to the appropriate team without destroying the security structure.

Select the team you want to work with, and click View Details (shown at the lower right of Figure 2.1) to see its signing identities and provisioning profiles, as you see in Figure 2.2.
Figure 2.3 shows the list of Mac certificates for a developer on developer.apple.com. Note that each one has a name that you provide, a type that you choose, and an expiration date that is set and enforced by Apple.

Managing Your App ID
Unlike your developer signing identity, which can be edited on developer.apple.com through Xcode accounts, your App ID must be managed for the most part on developer.apple.com. You give the app a name, which can be changed later on if you want. (This is not the name the user sees.) What is important to note is that when you register your App ID, you can enable services that you want to use, such as Game Center, In-App Purchase, Maps, Push Notifications, and most important for this book, iCloud.

Although you cannot create an App ID through Xcode, when you turn on a capability such as iCloud, Xcode offers to update your App ID to add the iCloud capability automatically. (You’ll see this demonstrated in Figure 2.9 later in this chapter.)

Managing Your Devices
You can register a number of devices that can be used for testing your apps. (As of this writing, the number is 100.) When you recruit people to test your app, ask them for the UDID (iOS) or UUID (OS X) of the device they want to test with. These people do not need to be registered developers, and sometimes it’s a good idea to recruit one
or two testers who are “real people” as opposed to developers. The rules for managing devices are detailed on developer.apple.com. There are limits to how many times you can change the list: this limit prevents you from allowing your app to be installed on a large number of devices without going through the App Store. (Ad hoc distribution is a specific option you may want to explore in this case.) You have one list of devices for your developer account. The provisioning profiles associate them with App IDs.

**Managing Provisioning Profiles**
Now that you have your App ID and a list of devices, you can create a provisioning profile to combine the two. As you see at the bottom of Figure 2.2, the provisioning profiles are listed by name and expiration date along with the various entitlements associated with them when you look at them in Xcode accounts. When you look at them in developer.apple.com, you’ll see that some are marked as being managed by Xcode. For the others, you can specify the devices and the services you want to enable on developer.apple.com.

Thus, at this point, you should have your developer signing identity; your app and its App ID; and your provisioning profile that brings together testing devices, your app ID, and the entitlements or services that it uses. You’re ready to start thinking about runtime.

**Identifying Your User and Your Ubiquity Container at Runtime**
As you saw in Chapter 1, iCloud helps users organize their data by app rather than by file and folder. Users can still work with documents, but those documents aren’t on a visible file system in most cases: they’re in iCloud. But where is iCloud?

As with all cloud computing, the cloud is an artifact of the Internet and large server farms. If you follow the trail of bits, you see that these server farms synchronize and store data so that it is accessible on an as-needed basis by users. The physical location of the data doesn’t matter, and in fact, the actual storage is so often duplicated across servers that there is frequently no single primary data store among the many stores that come into play.

A user’s data is available (subject to security constraints) whenever a user accesses the cloud with the appropriate account information and password. That’s not the model with iCloud. With iCloud, data is available with the presentation of two identifiers:

- **Apple ID**: This identifies the user.
- **A ubiquity container identifier**: A ubiquity container is the object that holds the iCloud data for the app. It typically is a bundle identifier such as com.champlainarts.colby. It is prefixed automatically by your developer Team ID.

When the user connects to iCloud (usually this happens through the iCloud System Preference panel whenever the user logs in), the Apple ID is made available to all apps that are entitled to use iCloud. The ubiquity container identifier is usually set in the
Capabilities tab of the target in Xcode. As you will see in Chapter 18, “Completing the Round Trip,” a shared ubiquity container may have an identifier that does not correspond to an app bundle identifier. In the case of the Round Trip, the two apps are com.champlainarts.ColbyOSX and com.champlainarts.ColbyiOS. They share a ubiquity container called com.champlainarts.Colby. It is the last component of the ubiquity container identifier that shows up in the System Preferences iCloud pane shown previously at the left in Figure 1.5.

With these two pieces of information, you can connect to the appropriate iCloud ubiquity container. That is your first task when your app starts to run.

Looking Inside the iCloud Basics

Bundle identifiers and Apple IDs have been around for a long time, but now they have key roles to play in iCloud. Both of them are needed to gain access to a section of iCloud. This is the implementation of the app-based file structure described in Chapter 1.

You might expect to find standard log-in methods in the iCloud API that enable your app to present an Apple ID and a bundle ID to iCloud in order to gain access to the data. That's not how it happens. Remember that there is no explicit iCloud API; beyond that, the notion of logging in to iCloud for an app isn’t what happens. (Users do log in to iCloud—often automatically with their settings in the iCloud pane of System Preferences.)

Your app interacts directly with a local copy of the iCloud data for the user and the app. This copy of the iCloud data for the user and app is stored locally in a ubiquity container. The ubiquity containers are stored on the local device, and their contents are synchronized by the local OS and its interaction with iCloud. Just as is the case with any other local data access, you can read and write as necessary, and you can expect (and even check on) the results of those read and write statements.

Because you are not reading and writing to the iCloud data directly in most cases, you can’t expect the changes that you have made to the local ubiquity data to be propagated to iCloud immediately. If you want to get into naming things, iCloud is an asynchronous and declarative implementation of cloud technology.

The key components of iCloud are

- Apple ID
- Bundle identifier
- Entitlements and capabilities
- Ubiquity container

The following sections cover the basics of what you need to know about them.
Declarative Programming

As is the case with more and more software today, iCloud relies heavily on declarative programming techniques. Declarative programming is distinguished from other styles that specify what happens and, frequently, in what order. (Common names for that style are imperative, functional, and procedural programming.) Declarative programming simply describes what should be done without specifying a control flow.

You find examples of declarative programming throughout OS X and iOS with more examples showing up with each new iteration of the Cocoa frameworks. Blocks, for example, allow you to specify code that is executed for each element of an enumerator or on completion of some task, but you do not hang around waiting for that trigger to occur. You define the block and then send it off (often as a parameter of a method), and it is executed at the appropriate time. You are out of the traffic-cop business, and, not coincidentally, multithreading at the system level is much easier for the OS to manage in your absence.

If you’re not familiar with declarative programming or are still not comfortable with it, explore the topic online (Wikipedia is a great place to start for this type of research).

Apple ID

We’re now looking at iCloud runtime behavior. The Apple ID discussed here is the user’s Apple ID.

An Apple ID uniquely identifies . . . something. It started in 2000 as an account name on Apple’s early Internet service, iTools, which provided free email accounts as <accountname>@mac.com. Over time, <accountname>@mac.com became <accountname>@me.com (MobileMe) and then <accountname>@icloud.com. With the advent of the iTunes store, customers used an Apple ID for their purchases. The email account name served as the first Apple IDs, but, particularly after Apple began charging for email accounts, Apple IDs no longer consisted of me.com or mac.com addresses. Every Apple ID did have to have an email address associated with it (for verification if for no other reason) and, for purchases in iTunes Store, a credit card number.

The idea that an Apple ID uniquely identifies an individual person has long gone away. An Apple ID has a name, a password, an email address, an optional rescue email address (in case the primary address is unreachable), and, if used for purchases, it may have a credit card associated with it. Apple suggests that people not share Apple IDs, but we know that sometimes a family or even a small business will share one.

Apple suggests that people may like to have one Apple ID to identify themselves to iTunes and another to identify themselves for other purposes such as iCloud, FaceTime, and the like. Developers often have one or more Apple IDs for their personal life and another for their developer account. iBook authors need their own Apple ID, so a developer who is also an iBook author needs two right there.
There is a unique identifier underneath all the attributes, so email address, name, password, and credit card can all be changed without creating a new Apple ID. Every iOS device requires that the user has an Apple ID in order to gain access to downloads of the operating system as well as any purchased apps or music.

On OS X, although the installation process encourages it, you do not need an Apple ID. If you want to use iCloud, you do need an Apple ID. Apple has recognized the proliferation of multi-Apple ID individuals in OS X Mountain Lion (10.8) and later versions of OS X. Figure 2.4 shows part of the Users & Groups pane in System Preferences.

Figure 2.4  You can have multiple Apple IDs on OS X.
Most of the time, people don’t pay attention to their Apple ID when they set up a device beyond checking that their email works (if it uses the Apple ID). However, for ongoing support of your iCloud app, remember (and let your tech support people remember) that the Apple ID is a critical part of iCloud access. If someone in an office uses an OS X account for business and another for personal matters, the iCloud documents created under those two OS X accounts may be using different Apple IDs. A Pages document under one account will not be shared with the other, although you can do so with sharing commands implemented in Pages and other apps.

The Apple ID that a user has used to sign into iCloud is available to the operating system at runtime, and that is how the Apple ID part of the iCloud authentication takes place: you don’t do anything.

**Bundle Identifier**

The bundle identifier is set in your app’s target settings in Xcode (in the General tab of the target). As you step through the process of creating a new project, you are asked for information, including the product name and the company identifier. You provide the product name, and the company identifier is editable (it actually is sticky—you start with the last company identifier you used).

---

**Note**

The management of bundle identifiers, product names, and targets, as described in this section, has been a matter of concern for a number of developers over the years. You can find many references on the Internet to what is going on. Unfortunately, some of them (particularly those from several years ago) are misleading. The information in this section is current as of Xcode 5, which is the version released with iOS 7 and OS X Mavericks (10.9).
The bundle identifier that Xcode starts with is the combination of the company identifier (which is usually your reverse domain name) and the product name, as in `com.yourcompany.yourproductname`.

If you look at the Info tab of your project, as shown in Figure 2.6, you’ll see that the bundle identifier is set to your product name. The product name is also used as the target name, so you begin with identical values for your target and the last component of your bundle identifier. You can change your target name in the left side of the project editor: just double-click and type in a new name. You’ll see that the last component of the bundle name also changes, because, as you see in Figure 2.6, it is picking up the product name.

![Figure 2.6 Editing the bundle identifier in Info](image)

However, you can edit the bundle identifier itself in the General tab. As initially set up, it is set to `com.yourcompany.$(PRODUCT_NAME:rfc1034identifier`. If you trace through the various settings, you’ll see that product name (in the Packaging section of Build Settings) is set to `$(TARGET_NAME)`. This means that if you change the target name, the product name will change, and because it’s used as the last component in the bundle identifier, that, too, will change. Anywhere along the line, you can double-click to edit the setting. If you change Product Name to be MyProject instead of `$(TARGET_NAME)`, you will change the product name, and indirectly, the last component of the bundle identifier. Generally, the best place to edit a bundle identifier is in the General tab of the project itself rather than in the Info tab. That is because the Info tab sets up the naming structure with placeholders such as `$(TARGET_NAME)` and the General tab lets you type in the actual name that you want to use, which overrides the placeholders.
Most of the time, the default settings are fine, and you don’t have to worry about them. However, they come into play with iCloud when you need a ubiquity container that is shared among several apps. (Perhaps most commonly, one is a Mac app and the other is an iOS app.)

**Entitlements and Capabilities**

Entitlements specify what your app can do. The Capabilities tab shown in Figure 2.7 lets you configure the capabilities and the related entitlements and other settings. As you can see, there’s a simple switch for each capability—iCloud, Game Center, Passbook, In-App Purchase, and more (still more are likely to come in the future).

*Note*

The Capabilities tab is new in Xcode 5. It replaces previous entitlements configurations that were different for iOS and OS X. The interface shown here is for iOS apps, but it is almost identical for OS X apps.

If a given capability is off, turning it on will also open the disclosure triangle to show you what additional steps you and/or Xcode must take, as you see in Figure 2.7.
When you turn a capability on, you’ll be asked to choose a development team to use in provisioning, as you see in Figure 2.8.

The steps that need to be taken, as shown in Figure 2.7, are checked off or, if a problem occurred, you are usually given an opportunity to have Xcode fix it, as you see in Figure 2.9.
Beginning in Xcode 5, this process replaces the manual configuration that you had to do in the past on developer.apple.com. You can still do that, and that is still the best place to actually see the details of your identities, provisioning profiles, and app IDs, but for many if not most of your transactions, Xcode will take care of those tasks. Also note that Xcode sets up the appropriate entries in your project’s plist.

As you can see in Figure 2.9, when you enable iCloud, you’ll be able to choose the entitlements file, but Xcode will begin by naming one for you. If you want to use a key-value store, you can enable it here: that is the topic of Chapter 8, “Using Key-Value Coding (KVC).” For documents (that is, data other than KVC data), you use a ubiquity container. You may have more than one, but the first one is always assumed to be the main one. If you are using KVC without documents, you don’t need a ubiquity container.

**Ubiquity Container**

As you can see in Figure 2.9, you can specify ubiquity containers for your app. The first one you create has a default name set by Xcode, and it has a special role to play. (You can change the default name if you want, and in some cases, you must, as you’ll see in the next paragraph.) The first ubiquity container is the primary ubiquity container. On OS X, its contents are displayed in the open and save dialogs available in NSDocument. (On iOS, you create your own interface to display documents in iCloud if you use them.)

The default name for the primary ubiquity container is the bundle identifier. In cases where you want to share a ubiquity container among several apps (such as an OS X version and an iOS version), change one of the ubiquity container names to the other one so it is shared. As you will see in Chapter 18, “Completing the Round Trip,” the shared ubiquity container may have any name you want. In Chapter 18, the two apps have bundle identifiers of com.champlainarts.ColbyiOS and com.champlainarts.ColbyOSX. The shared ubiquity container is com.champlainarts.colby.

**Using iCloud in Your App**

At this point, you’re ready to use iCloud in your app. You will see concrete examples of how to do so starting in Part III, “Using the Technologies.” There is one step that you can take now to confirm that your app has been properly set up and that the entitlements and provisioning are correct.

Create a new app or use an existing app that you want to enable for iCloud (starting from a new app is a simpler way in the long run until you’re more comfortable with iCloud). Set up the entitlements and provisioning as described in the previous sections. Add a single line of code to test if iCloud is available:

```objective-c
id currentiCloudToken = [[NSFileManager defaultManager] ubiquityIdentityToken];
```
On OS X, it should go in `applicationDidFinishLaunching:`, and on iOS, it should go in `application:didFinishLaunchingWithOptions:`. In both cases, it normally goes after your other initializations. (Note that this method was added in iOS 6 and OS X Mountain Lion (10.8). You can find older and more complex ways of performing this task on the web.)

The iCloud token that is returned is an opaque object identifying the iCloud account (that means that you can’t see inside it). There are two possibilities when you ask for the token:

- If it is `nil`, the user is not signed into an iCloud account.
- Although you can’t see the account details, you can check if a token is the same as another token using `isEqual:`. This lets you check to see if the user has changed iCloud accounts.

Note that if a user has been signed into an iCloud account and turns on Airplane mode or turns off networking on a Mac, the token is still returned. You can access the local copy of your ubiquity container’s data. When Airplane mode is turned on again, iCloud will take care of syncing the two stores and will let you know if there is anything for you to do. Because the operating systems manage these disruptions in connectivity, resist the temptation to store extra copies of data locally in the app’s sandbox.

Apple recommends as a best practice that you use either iCloud storage or sandbox storage. Mixing the two provides a suboptimal user experience. Along those lines, ask users if they want to use iCloud the first time they run your app. Unless they reinstall the app, don’t ask them again.

The iWork apps are a good example of how to manage documents in iCloud. Over the last few years, they have moved to an explicit Export command, which, among other things, can let you export the contents of an iCloud document to another format and to a non-iCloud location.

**Chapter Summary**

In this chapter, you’ve seen the basics of how iCloud works. iCloud for document data relies on a ubiquity container, which is identified by a user’s Apple ID and your app’s bundle ID and is enabled by entitlements. You can share ubiquity containers across several apps by using a single app’s bundle ID for all of them.

A high-level overview of the provisioning process has shown you where you enable iCloud for your app. Provisioning is done by registered developers on developer.apple.com and in Xcode accounts. Provisioning profiles as well as identity certificates are then downloaded. You install provisioning profiles in Xcode accounts, while certificates are installed automatically in Key Chain.
Exercises

1. If you have any doubts about the wisdom of Apple’s advice to either use iCloud for all storage or local (sandbox) storage for all storage, try to come up with a user interface of your own to manage them.

2. Set up entitlements for an iCloud-enabled app as described in this chapter. Start by following the steps exactly—either those in this chapter or those on developer.apple.com. Don’t take any shortcuts until your first provisioning profile is running properly. Then you can experiment.

3. TextEdit supports iCloud documents; it is installed as part of the OS X installation. Experiment with it and particularly note how iCloud has been integrated into the File Save dialog. You can access this dialog from your own code when you instantiate NSDocument objects.
Symbols
^ (caret) character, for block declarations, 99

A
ABPerson class, AddressBook API, 66
ABRecord class, AddressBook API, 66
Access, managing iCloud, 207–208
Accounts, managing changes of iCloud accounts, 344–345
Accounts tab, Xcode 5, 239–240
Add button, 219–220
AddDocument
   adding new document in Loon, 291–295
   making changes to storyboards, 215
   master view controller, 253
AddressBook API
   on iOS and OS X, 57–58
   review, 66–67
   sending Mail from iOS app, 58–65
   sending Mail from OS X app, 65
   using property lists for storing/syncing, 65–66
Ad hoc distribution, 22
Alerts
   checking that Mail is configured/available, 64–65
   code for Loon, 281–282, 284
Aliases, looking inside sandboxing containers, 179–180
APIs
AddressBook. See AddressBook API
   app overview, 40–41
   built-in data apps, overview, 38
   creating iOS Xcode project, 42–46
   creating OS X Xcode project, 47–50
   creating separate OS X/iOS Xcode projects, 41
Event Kit. See Event Kit API
   getting started as Apple Developer, 35–36
   keeping up with Apple, 38–40
   overview of, 35
   review, 55–56
   understanding, 37–38
   wiring up interfaces, 50–55
App delegate
   adding event to app on OS X, 83–84
   Calendar database, 72–74
   for Chazy project, 325–326
   creating iCloud apps, 215–217
   creating shared folders, 207
   linking to iCloud controller, 369
   for Loon project, 280–286
   monitoring store changes, 116–118
   for Placid project, 248–250
   preparing user interface in OS X, 113–114
   setting up store at runtime, 115–116
   storing data on iOS with, 119
   structuring app in Loon, 275
   Xcode file management, 159
App ID
   building iCloud app, 199–200
   identifying app with, 18
   managing, 21
   managing provisioning profiles, 22
   starting Placid project, 238
App Store
   never deleting App ID for app submitted to, 238
   Review Guidelines, 88–92
AppKit (Cocoa for OS X), 1
Apple Developer Forum, 44
Apple ID
   history of, 24
   iCloud runtime behavior and, 24–26
   identifying two categories of people, 18
   identifying user at runtime, 22–23
   managing developer signing identity, 19–21
   not requiring personal data to unlock app, 91
   setting up test devices to unlock app, 104
   understanding, 24–26
Apple’s Worldwide Developers Conference (WWDC)
keeping up with Apple changes, 40
WWDC 2012, 340
WWDC 2013, 340

Apps
APIs and first, See APIs
built-in data apps. See built-in data apps
managing, 18–23
managing documents with iCloud, Time Machine, and Auto Save, 12–13
organizing files by, 8–11
syncing data across devices, 13–14
using iCloud in, 30–31

Apps, document-based
accessors for WrappedDocument, 332–334
adding app delegates, 325–326
creating WrappedDocument, 323–324
defining WrappedDocument properties, 327–328
initialization and management code for WrappedDocument, 328–329
planning project app, 319–321
reading and writing code for WrappedDocument, 330–332
setting up in Xcode, 321–323
testing, 337
window management code for WrappedDocument, 329–330
WindowController subclass, 334–337
writing code for OS X app, 326–327

ARC (Automatic Reference Counting), 38

Architecture
of APIs relating to user data, 37
Calendar database, 72
iOS vs. OS X document, 191

Arrays
documents array. See documents array
retrieving events with, 77
Attributes. See also properties
adding to entities, 145–148
file wrapper file-system, 232
structuring data and, 135
Attributes inspector, 175
Auto Layout system, for Round Trip, 14
Auto Save, OS X, 13, 319

Automatic Reference Counting (ARC), 38

Automator app, 171
Availability, managing iCloud, 275

B
Backing variables
enforcing privacy, 89
keeping up with Apple, 38–39
Binary data, issues with iCloud, 146

Bindings, blocks containing bindings to variables, 99
Blocks
overview of, 97, 99–100
retrieving events with, 77
review, 104
types of queues in GCD, 99
using together with threads, 98

Blue boxes, 177
Breakpoints, in Loon code, 280, 304, 306
Build Phases
adding files to app's bundle, 173–175
adding files to project, 171
Build Settings tab, configuring Placid project, 246
Built-in data apps
app overview, 40–41
creating iOS Xcode project, 42–46
creating OS X Xcode project, 47–50
creating separate Xcode projects for iOS/OS X, 41
keeping up with Apple, 38–40
overview of, 38

Bundle identifier
as default name for primary ubiquity container, 30
naming, 276–277
sharing key–value store for Round Trip, 111
as storage area for files, 232
understanding, 26–28

Bundles
adding files to app's, 172–175
getting files out of, 175–176
overview of, 169–171

Buttons
adding event to app on OS X, 83–84
adding reminder to app on iOS, 80–83
adding to iOS Xcode project, 46
code for Loon, 286
wiring up iOS interface, 51–54
Calendar database
   accessing with Event Kit API, 69
   allocating/accessing event store, 72–74
   committing changes, 79–80
   creating new event/reminder, 75–76
   overview of, 72
   searching for event/reminder, 76–77
   setting/modifying properties, 77–79
   synchronization of, 69–70
Calendar management. See Event Kit API
CalendarItem Identifier property, 76
Capabilities tab
   accessing iCloud, 322–323
   building iCloud app, 200
   configuring entitlements/capabilities, 28–30
   configuring for Placid project, 242
   enabling iCloud for Loon project, 278–279
   extra feature of, 243
   keeping up with Apple, 39
   preparing project for testing, 108–109
   setting OS X permissions for calendar, 71
   setting up common ubiquity container for two projects, 367–368
   setting up sandboxing in, 178
Case sensitivity, project names, 276
Central library apps, iCloud working with, 139
Certificates
   configuring on developer.apple.com, 237–238
   configuring on Xcode 5, 239–240
   identifying user and app, 18–22
   managing in Xcode account preferences, 20
   managing on developer.apple.com, 21
$CFBundleIdentifier, sharing key-value store for Round Trip, 111
CFBundleTypeExtensions property, document types, 245
Chazy project
   accessor for WrappedDocument, 332–334
   adding app delegates, 325–326
   creating WrappedDocument, 323–324
   defining WrappedDocument properties, 327–328
   initialization and management code for WrappedDocument, 328–329
   planning, 319–321
   read/write code for WrappedDocument, 330–332
   setting up in Xcode, 321–323
   testing, 337
   window management code for WrappedDocument, 329–330
   WindowController subclass, 334–337
   writing code for, 326–327
CheckOnQuery, debugging method, 204
Classes. See also by individual class
   extensions, 207, 308
   methods, 111
Classes, creating iCloud apps
   AppDelegate, 215–217
   DetailViewController, 224–227
   MasterViewController, 217–224
Cloud computing. See also iCloud
   focus on app-centered content, 98
   understanding, 5–6
Cocoa
   Application template, 318, 350
   Cocoa Touch compared with, 191
   creating OS X Xcode project, 47–48
   keeping up with Apple changes, 38–40
   lazy loading in, 330–332
   MVC design pattern and, 318
   for OS X, 1
Cocoa Touch
   Cocoa compared with, 191
   for iOS, 1
   keeping up with Apple changes, 38–40
   lazy loading in, 330–332
   MVC design pattern and, 318
   UIDocument (iOS) and, 273
Codd, Edgar, 134
Code signing. See security
Colored card technique, controlling testing environment, 106
Columns, as fields, 136
Concurrent queues, 99
Configurations, data model, 144
ConfigureView, Loon project, 302–305
Connections inspector, wiring up iOS interface, 51–53
Constants, managing iCloud data, 209
Index

Constants.h class
  creating shared folders, 201
  managing iCloud data, 211

Constants.m class
  creating shared folders, 201–202
  managing iCloud data, 211

Contacts
  AddressBook vs., 57
  setting OS X permissions, 71

Container view controllers, 39–40

Containers, setting OS X permissions, 71

Containers folder, 178–180

Content view controllers, 39–40

Contents:forType: error;, WrappedDocument, 311

Contextual menu, opening package from, 170

Convert to Modern Objective-C Syntax, 39

Convert to Objective-C ARC, 39

Copy Bundle Resources, 174

Core Data
  adding files to app’s bundle, 172–175
  Core Data Project, 139–142
  goals, 134–135
  including store with app, 183–184
  managing persistent storage, 133–134
  managing versions, 144–145
  review, 155
  stack, 154–155
  structuring data, 135–139
  working with iCloud, 139

Core Data, integration with iCloud
  database migration, 345–347
  fallback stores for when iCloud is unavailable, 341–342
  managing account changes, 344–345
  managing persistent store changes, 343–344
  options dictionary, 340–341
  overview of, 339
  review, 348
  setting up persistent stores asynchronously, 342–343
  WWDC 2013 revisions, 340

Core Data, with Xcode data modeling tool
  converting entities to objects, 149–153
  managing data model, 144–145
  overview of, 142–144
  using objects, 154
  working with entities, 145–148

D

Data, adding to apps
  adding files to bundle, 172–175
  bundles, packages, and resources, 169–171
  getting files out of bundle, 175–176
  including Core Data store, 183–184
  including property lists, 181–183
  looking at sandboxed files, 176–181
  overview of, 168
  review, 184

Data, managing iCloud, 209–212

Data apps, built-in. See built-in data apps

Data model
  building persistent store from compiled, 183–184
  converting entities to objects, 149–153
  of Core Data stack, 154
  managing in Core Data, 144–145
  migrating to new version, 345–347
  using the object, 154
  working directly with in Core Data, 135
  working with entities, 149–153

Data Model editor, 142–144

Data model inspector, 147–148

Data store
  Calendar database, 72
  migration, 345–347

DBAs (database administrators), 345

DBMSs (database management systems), 345–347

DEBUG, enforcing privacy, 89

Debug navigator, Xcode, 364–366

Debugging
  checkOnQuery method for, 204
  enforcing privacy when, 89
  iCloud, 195–199, 364–366

Declarations
  block, 99
  WrappedDocument, 308

Declarative programming, iCloud reliance on, 24

Declared properties, keeping up with Apple, 38–39

#define, creating Shared folder, 201

Delegates. See app delegate

Deleting documents, 217, 223–224
DetailViewController
code for Loon, 301–306
creating iCloud apps, 224–227
structuring app in Loon, 275–276
using object in, 154
writing code for Placid project, 260–263
Developer Technical Support (DTS), 108
developer.apple.com
building iCloud app, 199–200
configuring certificates, identifiers, devices, profiles, 237–238
data management APIs, 37–38
documentation on, 340
getting started as Apple Developer. See APIs
identifying yourself and your apps, 18–19
keeping up with Apple changes, 39–40
managing your App ID, 21
rules for managing devices, 22
developer.icloud.com
accessing, 235
debugging iCloud apps, 195–199
viewing files and folders, 362–363
Development
getting started as Apple Developer, 35–36
Xcode workspaces for shared. See workspaces
Development, iCloud setup for Apple ID, 24–26
bundle identifier, 26–28
entitlements and capabilities, 28–30
managing app security on iOS and OS X, 18–23
overview of, 17–18
review, 31–32
ubiquity container, 30
understanding iCloud basics, 23–24
using iCloud in your app, 30–31
Development team, 109
Devices
configuring on developer.apple.com, 237–238
configuring on Xcode 5, 239–240
identifiers, 89
local and remote storage, 5–6
managing, 21–22
managing provisioning profiles, 22
syncing data across, 13–14
testing iCloud on iOS Simulator vs., 108
testing synchronization across iOS, 103–104
Digital Hub, 158–159
Digital signatures
identifying user/ubiquity container at runtime, 22–23
identifying yourself and your app, 18–22
managing App ID, 21
managing app security on iOS and OS X, 18
managing developer identity, 19–21
managing devices, 21–22
managing provisioning profiles, 22
setting up sandboxing on iOS, 177–178
Directories
folders as, 232
wrapped by file wrappers, 233
discoveredFiles array, iCloud query, 298, 300
displayComposerSheet, 63–65
DisplayDetailSegue, 215
Documentation, on developer.apple.com, 340
"Document-Based App Programming Guide for iOS," 274
documentDescription property,
WrappedDocument, 327–328
documentLocation property,
WrappedDocument, 327–328
Documents
code for Loon, 290–297
creating document types, 244–245
debugging iCloud apps, 195–199
definitions of, 273
exploring, 233
iCloud working with document-based apps, 139
iOS/OS X architecture differences, 191
managing with iCloud, Time Machine, and Auto Save, 12–13
managing with iWork apps, 31
managing with master view controller, 253–255
moving to iCloud, 376–377
opening file wrapper, 235–236
organizing files by app, 8–11
storing in iCloud, 105
Documents, NSDocument subclass
  accessors for WrappedDocument, 332–334
  adding app delegates, 325–326
  creating WrappedDocument, 323–324
  defining WrappedDocument properties, 327–328
  initialization and management code for
    WrappedDocument, 329–330
  NSDocument vs. UIDocument, 317–319
overview of, 317
  planning project app, 319–321
  reading and writing code for
    WrappedDocument, 330–332
review, 338
  setting up app in Xcode, 321–323
testing document app, 337
  window management code for
    WrappedDocument, 329–330
  WindowController subclass, 334–337
writing code for OS X app, 326–327

Documents, UIDocument subclass
  AppDelegate, 280–286
  DetailViewController, 301–306
  FileRepresentation, 314–315
  MasterViewController, 286–301
  planning app’s structure, 274–276
review, 315
  starting Loon project, 276–280
  UIDocument vs. NSDocument, 273–274
  WrappedDocument, 306–314
writing code, 280

Documents array
  creating new document, 217
  creating shared folders, 212–215
managing, 212–215
  managing iCloud data, 209–212

Documents folder
devices accessing iCloud, 197–198
  downloading files, 363
  documentsInICloud, 281–283

Entitlements
  building iCloud app, 200
  configuring capabilities and, 28–30, 243
  enabling iCloud for Loon project, 278–279
  sharing key-value store for Round Trip, 110–111

EOF (Enterprise Objects Framework), 134–135
Entitlements
  building iCloud app, 200
  configuring capabilities and, 28–30, 243
  enabling iCloud for Loon project, 278–279
  sharing key-value store for Round Trip, 110–111

Email. See AddressBook API

Enterprise Objects Framework (EOF), 134–135

Entities, data model
  converting to objects, 149–153
  defined, 144
  working with, 145–148

Event class hierarchy, 70–71

Event Kit API
  adding event to app on OS X, 83–84
  adding reminder to app on iOS, 80–83
  Event class hierarchy, 70–71
overview of, 69–70
review, 85
setting OS X permissions, 71
working with Calendar database, 72–80
Event store
allocating/accessing, 72–74
EKEventStore, Calendar database. See
EKEventStore, Calendar database
EventKitUI, 69
Extensions, file extensions, 232, 251

**F**

Fallback stores
for when iCloud is unavailable, 341–342
wiping persistent store and starting over, 346
Faulting, Core Data, 134–135
Fetch requests, data model, 144
File extensions
defined, 232
master view controller, 251
File inspector, exploring workspace, 189
File wrappers. See also WrappedDocument
adjusting general settings, 241
AppDelegate, 248–250
certificates, identifiers, devices, and profiles on developer.apple.com, 237–238
certificates, identifiers, devices, and profiles on Xcode 5, 238–240
checking build settings, 246
configuring capabilities, 242–243
Core Data store and, 339
DetailViewController, 260–263
exploring documents, 233
exploring files, 232
MasterViewController, 250–260
overview of, 231, 246–248
review, 270–271
setting document/universal type identifiers, 244–245
setting images, 242
starting Placid project, 236–237
users managing iCloud files, 233–236
working with storyboard, 270

FileMaker iOS, 135
FileRepresentation, 276, 314–315
Files
adding to app's bundle, 172–175
adding to project, 189–190
creating multiproject workspace, 163–167
downloading, 363
exploring, 232
getting out of bundle, 175–176
management of iCloud files, 233–236
managing with iCloud, Time Machine, and Auto Save, 12–13
organizing by app, 8–11
project navigator showing added project, 171
setting up multiproject workspace, 162–163
viewing with developer.icloud.com, 362–363
in workspace for iCloud apps, 188–190
Xcode in management of, 159–162
FileWrappers method, 233
Finder
creating multiproject workspace, 164–167
dragging files from iCloud, 320
managing files across various devices, 7
organizing files by app, 8–11
viewing workspace files in, 190
Xcode file management, 160–162
Fingerprint scan, Touch ID, 92
First normal form, 138
First Time setting, Loon project, 280
Folders
creating multiproject workspace, 162–167
as directories, 232
exploring workspace for iCloud apps, 188–190
viewing with developer.icloud.com, 362–363
Xcode file management, 159–162
Foreign keys, 137
Frameworks
Enterprise Objects Framework (EOF), 134–135
importing to ViewController.m, 60–61
Frameworks (continued)
MFMailComposeViewController framework. See MFMailComposeViewController framework
Functional programming, 24

G
GCD (Grand Central Dispatch). See Grand Central Dispatch (GCD)
General info settings, Loon project, 276–277
General tab
configuring Placid project, 241
designing folder structure for shared apps, 191–192
edit bundle identifier in, 27–28
examining files in System Preferences (OS X), 355
examining files with iOS Settings, 356–362
creating classes, 215–229
creating shared folders, 201–207
data management, 209–212
debugging, 195–199, 364–366
designing folder structure for shared apps, 191–192
Digital Hub as predecessor to, 159
document management with, 12–13
designing folder structure for shared apps, 191–192
exploring app workspace, 188–190
integration with Core Data. See Core Data, integration with iCloud
iCloud vs. OS X document architectures, 191
managing account changes, 344–345
managing availability of, 275
managing documents array, 212–215
moving documents to, 376–377
moving documents to local storage, 377–378
opening files, 320–321
overview of, 187–188
query, 298–300
review, 230
Round Trip. See Round Trip
scoping project, 194
storyboards, 230
synchronization, 346
tracking usage, 337
turning on, 108–110
types of storage in, 105
using user defaults with, 129–130
iCloud controller
declaring shared controller, 369–370
implementing shared controller, 370–375
making app delegate link to, 369
sharing, 368–369
iCloud gauge, Xcode, 364–366
iCloud Keychain, 92
iCloud Key-Value Store
methods, 111–112
sharing key-value store for Round Trip, 111
working with, 112
Icon view, OS X, 351

H
handleiCloudAvailabilityChange method, notifications, 102–103
hasChanges method, Event class, 70
Header file
AppDelegate, 215, 281
DetailViewController, 224, 261, 301–302
FileRepresentation, 314
MasterViewController, 217, 250–251, 286
ReportDocument, 227–228
WrappedDocument, 264, 307–308
High-speed Internet connections, cloud computing and, 6
Home button, 197–198

I
iCloud
access management, 207–208
accessing from Capabilities tab, 322–323
building apps, 199–200
checking out app built with, 192–194
creating classes, 215–229
creating shared folders, 201–207
data management, 209–212
debugging, 195–199, 364–366
designing folder structure for shared apps, 191–192
Digital Hub as predecessor to, 159
document management with, 12–13
designing folder structure for shared apps, 191–192
review, 230
Round Trip. See Round Trip
scoping project, 194
storyboards, 230
synchronization, 346
tracking usage, 337
turning on, 108–110
types of storage in, 105
using user defaults with, 129–130
iCloud controller
declaring shared controller, 369–370
implementing shared controller, 370–375
making app delegate link to, 369
sharing, 368–369
iCloud gauge, Xcode, 364–366
iCloud Keychain, 92
iCloud Key-Value Store
methods, 111–112
sharing key-value store for Round Trip, 111
working with, 112
Icon view, OS X, 351
Icons
  adding to document type, 245
  setting up UTIs, 245
Id field, 137
Identifiers
  configuring on developer.apple.com, 237–238
  configuring on Xcode 5, 239–240
device, 89
  setting document/universal type, 244–245
user and app, 18–22
Identity, in iCloud, 208
Images.xcassets file, 242
Imperative programming, 24
Info tab
  declaring document types for Placid project, 244–245
  setting documents for Loon project, 279
Information Property List dictionary, 123
Initialization
  creating shared folders, 204–207
  master view controller, 217
  WrappedDocument, 328–329
initializerCloudAccess, 204–206, 284
insertNewObject, 290
Instances, of properties, 136
Interface Builder
  wiring interfaces with, 50–51
  wiring up iOS interface, 51–54
Internet
  high-speed connections and cloud computing, 6
  sending Mail from iOS app, 59
iOS
  adding files to app’s bundle, 172–175
  adding reminders, 80–83
  AddressBook data, 58
  app security, 18–23
  Apple ID required for, 25
  Cocoa Touch for, 1
  Core Data project, 141–142
  document architecture, 191
documents. See documents, UIDocument subclass
event store declaration, 73
examining iCloud files with iOS Settings, 356–362
iCloud for, 2, 30–31
  monitoring interface changes, 118–119
  monitoring store changes, 116–118
  OS X compared with, 54, 317–318
  preparing project for testing, 108–110
  preparing user interface, 112–113
  requesting access to events/reminders, 74
Round Trip and, 14
  sending mail, 58–65
  setting up sandboxing, 177–178
  setting up store, 114–115
  shared ubiquity container for apps, 366–368
  sharing key-value store for Round Trip, 110–111
  split view controller in, 353–354
  testing iCloud on Simulator, 107–108
  user defaults, 121–122
  wiring up interfaces, 51–54
  Xcode data modeling tool and, 142–150
  Xcode project, 42–46
iOS Development certificate, 238
IP addresses, best practices in app privacy, 89
iPad
  app workspace, 193
  iPad vs. iPhone architecture, 274–275
  setting up with window property, 281
  working with iOS documents, 274–275, 353–354
iPhone
  iPad vs. iPhone architecture, 274–275
  navigation interface, 193–194, 302
  storyboards, 46, 56
  working with iOS documents, 354
isNew method, Event class, 70
iTunes, 104
iWork apps
  demonstrating iCloud to users, 2
  managing documents in iCloud, 31
  saving to iCloud, 198–199
J
Jobs, Steve, 177
Join tables, many-to-many relationships, 137
K
KeyForCurrentUbiquityToken, 208
Key-value coding (KVC)
  converting entities to objects, 149
  enabling key-value store, 30
  implementing, 106–107
  keeping track of defaults with ubiquity store, 129
  methods, 111–112
  monitoring interface changes, 118–119
  monitoring store changes, 116–118
  NSUbiquitousKeyValueStore, 111
  overview of, 105
  preparing project for testing, 108–110
  preparing user interface, 112–114
  review, 120
  setting up controlled testing environment, 106
  setting up store at runtime, 114–116
  sharing key-value store for Round Trip, 110–111
  testing iCloud on iOS simulator, 107–108
  user defaults settings using, 122
  working with store, 112

L
Latency, iCloud, 104
Lazy loading, in Cocoa and Cocoa Touch, 330–332
Library folder, 176, 178–180
List view, OS X, 351
Live queries, managing iCloud data, 209–211
Load section, Loon project, 288–289
Local storage
  devices and, 5–6
  moving documents from iCloud to, 377–378
Location data
  best practices in app privacy, 88–89
  setting OS X permissions, 71
Loon project
  adding settings, 280
  AppDelegate, 280–286
  DetailViewController, 301–306
  FileRepresentation, 314–315
  MasterViewController, 286–301
  overview of, 280
  Placid project vs., 273–274
  planning app’s structure, 274–276
  setting Capabilities, 278–279
  setting general info, 276–277
  setting up documents, 279
  WrappedDocument, 306–314
Loose coupling, notifications, 100
Lproj file, 160
LSHandlerRank property, document types, 245
LSTypeIsPackage property, document types, 245

M
Mac App Store, 88
Mail
  AddressBook API, 57–58
  checking that it is configured/available, 63–64
  property lists for storing/syncing, 65–66
  sending from iOS app, 58–65
  sending from OS X app, 65
Main queues, 99
MainViewController
  monitoring store changes, 116–118
  preparing user interface in iOS, 113
  setting up store/UI on iOS at runtime, 114–115
Managed objects
  of Core Data stack, 155
  managedObjectModel, 175–176
Many-to-many relationships, 137
Many-to-one relationships, 137
Master-Detail Application template
  adding files to app’s bundle, 172–175
  adding new document to, 219–220
  building iCloud app, 199–200
  checking out end result of new app, 192–193
  creating iOS Core Data project, 141–142
  getting files out of bundle, 175–176
  including Core Data store with app, 183–184
  iPad vs. iPhone architecture, 274–275
  Loon project, 286–301
  Placid project, 236–237
  Xcode data modeling tool, 142–150
  Xcode file management, 159–162
MasterViewController
- creating documents, 220–222
- deleting documents, 223–224
- handling segues, 223
- implementing, 218–219
- interacting with iCloud at document level, 301
- Loon project, 286–301
- managing list of documents, 217
- objects in, 154
- Placid project, 250–260
- responding to tap in table view, 222
- setting up header, 217
- structuring apps, 275–276
- viewing at iOS documents, 353–354
- wiring Add button, 219–220
- Memory section, Loon project, 288–289
- MessageComposer sample app, 58–65

Methods
- Calendar database, 72–74
- debugging, 204
- Event Kit class, 70–71
- file wrappers, 233
- iCloud access, 207–208
- key-value store, 111–113, 115–117, 119

MFMailComposeViewController framework
- adding reminders to app, 81
- AddressBook API, 57–58
- building into OS X code, 58
- checking that Mail is configured/available, 63–64
- sending mail, 58–59
- sending messages, 60–63
- Minors, protecting online privacy of, 91
- Mobile news, 98
- Model-view-controller (MVC) design pattern
  - implementing relational databases, 140
  - using with database apps, 318
- Monitoring
  - interface changes, 118–119
  - key-value store changes, 112, 116–118
- Multiproject workspace
  - creating, 163–167
  - setting up, 162–163
- Multitarget workspace, creating, 188–190
- Mutable documents array, 253
- MVC (model-view-controller) design pattern
  - implementing relational databases, 140
  - using with database apps, 318

N
- Name property, notifications, 101
- NameID field, 137
- Naming
  - blocks, 99
  - document types, 244–245
  - documents, 255, 291–295
  - projects, 276–277
  - text views, 51–52
  - UTIs, 245
- Navigation interface, iPhone
  - app workspace, 193–194
  - ViewWillAppear/ViewWillDisappear, 302
  - working with iOS documents, 274–275
- NeXT, 134
- NeXTSTEP, 139
- Nib files
  - creating Chazy project, 321, 323–324
  - creating OS X Xcode project, 47
  - First Responder commands in, 352
  - linking class to delegate or property, 327
  - overview of, 48
- Non-Retina versions of images, 242
- Normalizing data, 138–139
- Notifications
  - of iCloud availability changes, 102–103
  - Loon project, 280
  - managing iCloud data, 209–212
  - overview of, 97, 100
  - posting, 102
  - properties, 101
  - registering for, 101–102
  - for view appearance/disappearance, 226–227
- NSApplicationDelegate protocol, 325
- NSArray
  - implementing KVC, 107
  - property list class, 122
  - reading and writing property lists, 127
- NSData objects
  - property list class, 122
  - reading plist into, 182–183
  - using in property lists, 127
### NSDate
- creating new event, 76
- property list class, 122

### NSDictionary
- Core Data methods and, 340–341
- implementing KVC, 107
- property list class, 122
- reading and writing property lists, 127
- reading property list into, 182–183
- registering defaults, 130
- setting up/using key-value store, 112
- storing, 111

### NSDocument
See also documents,
- NSDocument subclass defined, 233
- NSPersistentDocument subclass, 339
- UIDocument compared with, 317–319

### NSHomeDirectory
180–181

### NSKeyValueProtocol
- implementing KVC, 106–107

### NSManagedObject
149–153, 155

### NSManagedObjectContext
155

### NSMetaDataQuery
- finding iCloud documents, 298–300
- managing iCloud data, 209–212
- master view controller and, 258–260

### NSNotificationCenter
- creating notifications, 101
- notifications as lightweight objects of, 100

### NSNotificationCenter
100

### NSNotificationCenter
100

### NSNumber
- property list class, 122
- using scalars in property lists, 127–128

### NSObject
106–107

### NSPersistentDocument subclass
339

### NSPersistentStore
- managing changes to, 343–344
- rebuilding local store, 346–347
- setting up persistent stores asynchronously, 342–343

### NSPersistentStoreCoordinator
155

### NSPropertyList
- methods, 111–112
- monitoring interface changes, 118–119
- monitoring store changes, 116–118
- overview of, 111
- preparing user interface, 112–114
- setting up store at runtime, 114–116
- working with store, 112

### NSUbiquitousKeyValueStore
- methods, 111–112
- monitoring interface changes, 118–119
- monitoring store changes, 116–118
- overview of, 111
- preparing user interface, 112–114
- setting up store at runtime, 114–116
- working with store, 112

### NSUserDefaults class method
- getting to app’s defaults with, 130
- managing iCloud access, 207–208
- monitoring interface changes, 119
- monitoring store changes, 117
- setting up store at runtime, 115–116

### NSWrapper
232–233. See also file wrappers

### Object property, notifications
101

### Objective-C
- enforcing privacy with, 89
- keeping up with Apple, 38–39

### Object-oriented programming
134

### Objects
- converting entities to, 149–153
- data objects. See NSManagedObject
- graphs, 134
- implementing KVC, 106–107
- inserting new, 290
- managing objects, 155, 175–176
- setting up/using key-value store, 112
- using, 154

### One-to-many relationships
137

### Open dialog, in OS X
350–353

### Operating systems
- Round Trip and, 14

### OS X
- adding events, 83–84
- app security, 18–23
- Apple ID for, 25
- Auto Save, 319
- calendar permissions, 71
- Cocoa for, 1
Core Data project, 140–142

document architecture, 191
document-based apps. See documents,
  NSDocument subclass
event store declaration, 73
iCloud for, 2, 30–31
icon and list views, 351
iOS compared with, 54, 317–318
looking inside sandboxing containers,
  178–180
monitoring interface changes,
  118–119
monitoring store changes, 116–118
multiple Apple IDs on, 25–26
Open dialog in, 350–353
preparing project for testing, 108–110
preparing user interface, 113–114
requesting access to events/reminders, 74
Round Trip requirements, 14
sending mail, 65
setting up store at runtime, 115–116
shared ubiquity container for iOS and OS
  X apps, 366–368
sharing key-value store for Round Trip,
  110–111
user defaults, 121–122
wiring up interfaces, 54–55
writing code for document-based app,
  326–327
Xcode project, 47–50

P

Packages
  overview of, 170
  WrappedDocument, 312–314
Pages documents, viewing Cloud-enabled apps,
  9–10
Parameters, registration for notifications, 102
Password managers, 92
Passwords
  storing, 107
  user issues, 92
Permissions
  asking user before accessing data, 79–80,
    90–91
  OS X, 71
  privacy rules and guidelines, 88
Persistent store coordinator (psc), 154–155,
  341, 343–344
Persistent stores. See also Core Data
  building from compiled data model,
    183–184
  of Core Data stack, 154
  managing changes to, 343–344
  rebuilding local store, 346–347
  setting up asynchronously, 342–343
Personal information, privacy of, 92
Pew Research Center’s Project for Excellence
in Journalism, 98
Phone phreaks, 177
Placid project
  adjusting general settings, 241
  certificates, identifiers, devices, and profiles,
    237–240
  checking build settings, 246
  configuring capabilities, 242–243
  Loon project vs., 273–274
  setting document/universal type identifiers,
    244–245
  setting images, 242
  starting, 236–237
Plists. See property lists
Posting notifications, 102
Predicates
  enumerating events with, 77
  searching for event/reminder, 77
Primary key, 137
Primary ubiquity container, 30
Privacy
  Apple rules and guidelines, 88
  asking permission and explaining use of
    data, 90–91
  best practices, 88–92
  debugging and, 89
  knowing what should be private, 88–89
  need for, 87
  not requiring personal data to unlock
    app, 91
  overview of, 87
  programming style enforcing, 89
  protecting minors, 91
  review, 93
  for support materials, 91–92
  user issues, 92
Index

Procedural programming, 24
Product name, editing bundle identifier, 27–28
Profiles
identifying user and app, 18–22
provisioning profiles, 237–240
Programming style, enforcing privacy with, 89
Project navigator
adding files to app's bundle, 173–175
adding property list data to app, 182
creating multiproject workspace, 166–167
exploring workspace for iCloud apps, 189–190
showing added project files, 171
Xcode file management, 160–162
Projects
Chazy project. See Chazy project
creating iOS Xcode, 42–46
creating multiproject workspace, 163–167
creating OS X Xcode, 47–50
Loon project. See Loon project
Placid project. See Placid project
preparing for testing, 108–110
setting up multiproject workspace, 162–163
working directly with iCloud. See key-value coding (KVC)
Xcode file management, 159–162
Properties. See also attributes
adding to document type, 245
calendar items, 77–79
Core Data, 135–136
declaring, 207
enforcing privacy with good programming style, 89
notification, 101
setting up UTIs, 245
WrappedDocument, 327–328
Property lists
adding to your app, 181–182
looking at, 122–125
looking inside, 125–126
NSData objects in, 127
overview of, 122
reading and writing, 127
scalars in, 127–128
storing/syncing in AddressBook, 65–66
user defaults settings, 122
Provisioning profiles
configuring on developer.apple.com, 237–238
configuring on Xcode 5, 239–240
creating and managing, 22
Psc (persistent store coordinator), 154–155, 341, 343–344
Q
Queries
managing iCloud data, 209–211
master view controller, 258–260
SQL queries, 136
working with iCloud, 298–300
Queues
defined, 98
enqueuing blocks in, 99–100
types in GCD, 98–99
R
Reachability sample code
adding reminder to app on iOS, 80–83
sending mail from OS X app, 59–60
Reading
property lists, 127, 182–183
to/from URL, 353
WrappedDocument, 330–332
Records, unique identifiers of table, 136
Redo, WrappedDocument, 309–311
Refactor commands, creating iOS Xcode project, 42–46
Refactor submenu, Edit menu, 39
References, notification, 100
RegisterDefaults, 130–131
Registration, for notifications
defined, 100
overview of, 101–102
receiving iCloud availability changes, 102–103
review; 104
Registration, of user defaults, 130–131
RegularFileContents method, file wrappers, 233
Relational databases
Core Data as merging of OOP and, 134
relationships in, 136–137
spreadsheet design of, 135
Index 393

Relationships
  in Core Data, 136–137
denormalizing data, 139
normalizing data, 138–139
Reminders, managing with Event Kit API. See
  Event Kit API
Remote storage devices, 6–7
ReportDocument, 227–229
Reset method, Event class, 71
Retina versions of images, 242
Review Guidelines, App Store/Mac App
  Store, 88
Rollback method, Event class, 71
Round’Tri
  Auto Layout system for, 14
declaring shared iCloud controller, 369–370
  from developer viewpoint, 362–363
  examining iCloud files in System
    Preferences (OS X), 355
  examining iCloud files with iOS Settings,
    356–362
  implementing shared iCloud controller,
    370–375
  linking app delegate to iCloud
    controller, 369
  making, 14
  moving documents from iCloud to local
    storage, 377–378
  moving documents to iCloud, 376–377
  overview of, 349
  review, 378
  shared iCloud controller, 368–369
  shared key-value store, 110–111
  shared ubiquity container, 23, 366–368
  from user viewpoint, 350
  working with Open dialog in OS X,
    350–353
  working with split view controller in iOS,
    353–354
  Xcode and, 364–366
Runtime
  getting files out of bundle for use at,
    175–176
  identifying user/ubiquity container at,
    22–23
  setting up store at, 112, 114–116
  understanding Apple ID at, 24–26

S
Sandboxing
  constraining access with, 87
  iCloud storage vs., 31
  looking inside sandboxing Containers on
    OS X, 178–180
  Loon project, 287–288, 300–301
  master view controller, 251
  overview of, 176–177
  setting up, 177–178
  writing to sandboxes, 180–181
Saving
  changes to reminders and events,
    79–80
  iOS Xcode project, 44
  Save button, 113, 119
  saveData method, 113, 119
  SBSendEmail sample app, 65
  Scalars, using in property lists, 127
  Scope, project, 194
  Scratchpad, Core Data stack, 155
Second normal form, 138
Security. See also privacy
devveloper.apple.com, 18
digital signatures and, 18
  identifying user/ubiquity container at
    runtime, 22–23
  managing App ID, 21
  managing developer identity, 19–21
  managing devices, 21–22
  managing provisioning profiles, 22
  sandboxing for. See sandboxing
Serial queues, 99
Setters, key-value store, 112
Settings
  examining iCloud files, 356–362
  iOS Simulator, 107–108
  legacy, 122
  locating iOS defaults, 129
  managing iOS defaults, 121
Shared development. See workspaces
Shared folders
  building iCloud app and, 200
  Constants.h, 201
  Constants.m, 201–202
  designing structure of, 191–192
Shared folders (continued)
exploring workspace for iCloud apps, 188–190
managing documents array, 212–215
managing iCloud access, 207–208
managing iCloud data, 209–212
multiproject workspace and, 163, 166
overview of, 201
SharedCloudController.h, 202–204
SharedCloudController.m, 204–207
SharedCloudController, 207
SharedCloudController class, 215–217
SharedCloudController.h class, 202–204
SharedCloudController.m class, 204–207
Shoebox apps
historical accuracy of, 140
iCloud working with, 139
managing iCloud availability, 275
Show Package Contents command, 170, 235
Simula 67 programming language, 134
Simulators
running Xcode project on, 44
sending Mail from iOS app, 59
testing iCloud on, 107–108
testing synchronization across iOS devices, 103–104
Single View Application template, 42–46
Span parameter, events, 79–80
Split view controller, iPAD
app workspace, 193
setting up with window property, 281
working with iOS documents, 274–275, 353–354
SQL queries, 136
SQLite database, 135, 139
Standards, calendar format, 69
Storage. See also Core Data
fallback stores for when iCloud is unavailable, 341–342
iCloud. See iCloud
key-value store. See key-value coding (KVC)
moving documents from iCloud to local storage, 377–378
persistent. See persistent stores
types of storage in iCloud, 105
using iCloud storage vs. sandboxing, 31

Storyboards
container views attached to, 40
creating iCloud apps, 230
creating iOS Xcode project, 44
customizing for Placid project, 270
iPhone, 46, 56
making Round Trip using, 14
master view controller, 251
preparing user interface in iOS, 113

Structure
app, 274–276
data, 97, 135–139
folder, 191–192
Support materials, providing privacy for, 91–92
Synchronization
AddressBook API, 65–66
doing data across devices, 13–14
file wrappers and, 233
iCloud Keychain and, 92
setting up controlled testing environment, 106
setting up key-value store, 112
setting up store at runtime, 114–116
structuring data for, 97
testing across iOS devices, 103–104
using iCloud with user defaults, 129–130

System Preferences
examining iCloud files in OS X, 355
legacy preferences, 122
locating OS X defaults, 129
managing OS X user defaults, 121
viewing iCloud documents, 11

T
Table style, Data Model editor, 143–144, 147
Table view
Loon project, 287, 295–297
master view controller updating, 256–257
Tables
managing storage with Core Data, 135–136
normalizing data, 138–139
relationships between, 136–137
Tap, responding in table view to, 222
tapButton
checking that Mail is configured/available, 63–65
sending message from iOS app, 60, 62–63
Index 395

Targets
  app bundles for, 170
  editing bundle identifier, 27–28

Templates
  Cocoa Application template, 318, 350
  creating Core Data project, 140–142
  creating iOS Xcode project, 42–46
  enabling Core Data in Xcode, 155
  Master-Detail Application template. See
    Master-Detail Application template
      using Cocoa application, 47–48

Testing
  debugging iCloud apps, 187–188
  iCloud on iOS simulator, 107–108
  Loon project, 280
  preparing project for, 108–110
  setting up controlled environment, 106

Text fields
  creating OS X Xcode project, 48–49
  preparing user interface in KVC, 112–113
  wiring up OS X interfaces, 54–55

Text view
  creating iOS Xcode project, 44–46
  making changes to storyboards, 215
  making Round Trip on iOS side using, 14
  sending message from iOS app, 60–63
  wiring up iOS interface, 51–54
  wiring up OS X interfaces, 54
  textFieldDidEndEditing, KVC, 112–113

Third normal form, 139

Threads
  defined, 98
  overview of, 97
  queues and, 98–99
  review, 104
  using blocks together with, 98

Time lag, in synchronization process, 106

Time Machine backup, 7, 12–13

Tokens
  managing iCloud access, 207–208
  ubiquity token, 281–286
  Touch ID, fingerprint scan, 92

Ubiquity container
  identifying at runtime, 22–23
  Loon project, 283–284
  navigating, 363
  sharing iOS and OS X apps, 366–368
  specifying for your app, 30
  Ubiquity token, 281–286
  UDID (iOS), 21–22
  UIAlertViewDelegate, 281
  UIDocument. See also documents,
    UIDocument subclass
      defined, 233
  NSDocument compared with, 273,
    317–319
  UIManagedDocument subclass, 339
  UIKit (Cocoa Touch for iOS), 1
  UILabel item, 215
  UIManagedDocument subclass, 339
  UITextField Delegate, 118
  UITextView, 54
  unconfigureView, 302–305
  Undo, WrappedDocument, 309–311
  Uniform Type Identifier (UTIs)
    defined, 232
    setting up, 245
  Unique identifiers
    EKEvent and EKReminder, 76
    as primary key, 137
    for records in tables, 136–137
    retrieving events with, 77
    setting up UTIs, 245
    updateCloudItems, 116–118

Updates, 106

URLForUbiquityContainerIdentifier
  creating new document, 220
  creating shared folders, 206–207
  working with iOS documents, 292
  writing code, 250, 253, 283

URLs
  including Core Data store with app, 185
  reading/writing, 353
  storing filenames, 314–315

Use iCloud setting, Loon project, 280

User data APIs, 38

User defaults
  legacy preferences/Settings, 122
  overview of, 121
  preparing user interface, 113
  property lists. See property lists
    registering, 130–131
User defaults (continued)
review, 131
setting up store at runtime, 114–116
working with, 128–130
User experience
cloud computing, 5–7
iCloud paradigm, 8–14
making Round Trip, 14
managing documents with iCloud, Time Machine and Auto Save, 12–13
organizing files by app, 8–11
review, 14–15
Round Trip from user viewpoint, 350
syncing data across devices, 13–14
User interface. See also user experience
advanced view techniques, 39–40
creating graphically, 47–48
detail view controller focus on, 301
iOS vs. OS X document architectures, 191
monitoring store changes, 118–119
overview, 50–51
preparing using KVC, 112–113
updating store from changes in, 112
wiring up OS X, 54–55
working with store, 112
writing up iOS, 51–54
userInfo property, notifications, 101
User-settable defaults, 128–129
Utility Application. See iOS
UTIs (Uniform Type Identifier)
defined, 232
setting up, 245
UTTTypeSpecification property, UTI, 245
UUID (OS X), 21–22
V
Values
code conflict resolution and, 105
data conflicts in synchronization, 106
setting up/using key-value store, 112
using iCloud with user defaults,
129–130
Variables
backing variables, 38–39, 89
blocks containing bindings to, 99
Versions
Auto Save document, 12–13
managing in Core Data, 144–145
View Controller Catalog for iOS, 39–40, 52–54
View controller, preparing user interface in iOS, 113
ViewDidLoad method, Calendar database, 72–74
ViewWillAppear, iPhone navigation interface, 302
ViewWillDisappear, iPhone navigation interface, 302
W
Warnings, Xcode, 38
WHERE clause, 77
WindowController subclass, 334–337
WindowDidLoad method, Calendar database, 72–74
Windows
managing for WrappedDocument, 329–330
setting up split view with, 281
Workspaces, 188–190
building on Digital Hub, 158–159
creating for iCloud app, 199–200
creating iOS Xcode project, 46
creating multiproject, 163–167
exploring for iCloud apps, 188–190
making Round Trip using multiproject, 14
overview of, 157–158
review, 167–168
setting up multiproject, 162–163
shared files in two places in, 192
Xcode file management, 159–162
Worldwide Developers Conference. See WWDC (Apple’s Worldwide Developers Conference)
Wozniak, Steve, 177
WrappedDocument
accessors, 332–334
Chazy project, 323–324
Loon project, 306–314
managing windows, 329–330
Placid project, 260–270
properties, 327–328
structuring apps, 276
WrappedDocumentDelegate, Loon project, 306–308
WrappedFile, 301
Writing, 330–332
  property lists, 127
to/from URL, 353
WrappedDocument, 330–332
to your sandbox, 180–181
WWDC (Apple’s Worldwide Developers Conference)
  keeping up with Apple changes, 40
WWDC 2012, 340
WWDC 2013, 340

X
xcassets file type, 242
Xcode
  adjusting general project settings, 241
  bundle identifier in, 26–28
  configuring entitlements/capabilities, 28–30
  creating workspace for two apps, 366–368
  Debug navigator, 364–366
  file management, 159–162
  IDE for building apps, 37
  inability to create App ID, 21
  keeping up with Apple, 38–39
  managing developer identity, 19–21
  planning document-based app project, 319–321
  viewing workspace files, 189
  with, 21
  workspaces. See workspaces
Xcode data modeling tool
  converting entities to objects, 149–153
  managing data models, 144–145
  review, 155
  using objects, 154
  working with Core Data, 142–144
  working with entities, 145–148
XML, inside property lists, 125–126

Z
zzID field, 137