Extracted from:

iCloud for Developers

Automatically Sync Your iOS Data, Everywhere, All the Time

This PDF file contains pages extracted from *iCloud for Developers*, published by the Pragmatic Bookshelf. For more information or to purchase a paperback or PDF copy, please visit http://www.pragprog.com.

Note: This extract contains some colored text (particularly in code listing). This is available only in online versions of the books. The printed versions are black and white. Pagination might vary between the online and printed versions; the content is otherwise identical.

Copyright © 2013 The Pragmatic Programmers, LLC.

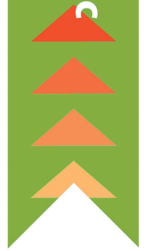
All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form, or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior consent of the publisher.

The Pragmatic Bookshelf

Dallas, Texas • Raleigh, North Carolina

The Pragmatic Programmers



iCloud for Developers

Automatically Sync Your iOS Data, Everywhere, All the Time

Cesare Rocchi

edited by John Osborn

iCloud for Developers

Automatically Sync Your iOS Data, Everywhere, All the Time

Cesare Rocchi

The Pragmatic Bookshelf

Dallas, Texas • Raleigh, North Carolina

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and The Pragmatic Programmers, LLC was aware of a trademark claim, the designations have been printed in initial capital letters or in all capitals. The Pragmatic Starter Kit, The Pragmatic Programmer, Pragmatic Programming, Pragmatic Bookshelf, PragProg and the linking *g* device are trademarks of The Pragmatic Programmers, LLC.

Every precaution was taken in the preparation of this book. However, the publisher assumes no responsibility for errors or omissions, or for damages that may result from the use of information (including program listings) contained herein.

Our Pragmatic courses, workshops, and other products can help you and your team create better software and have more fun. For more information, as well as the latest Pragmatic titles, please visit us at *http://pragprog.com*.

The team that produced this book includes:

John Osborn (editor)
Kim Wimpsett (copyeditor)
David J Kelly (typesetter)
Janet Furlow (producer)
Juliet Benda (rights)
Ellie Callahan (support)

Copyright © 2013 The Pragmatic Programmers, LLC.
All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form, or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior consent of the publisher.

Printed in the United States of America.
ISBN-13: 978-1-937785-60-4
Encoded using the finest acid-free high-entropy binary digits.
Book version: P1.0—July 2013

You have created an attractive application that friends and families can use to jointly manage their shopping lists, to-do lists, and other types of lists. Your customers say they like it, but it lacks two features they'd like to see: backup and synchronization.

If you haven't yet received such feedback, chances are you will. Apple users expect more from their applications these days because Calendar, Contacts, and many other Apple applications that ship with the latest iPhones, iPads, and Macs can both store their data in the cloud and sync it across multiple devices.

Take Calendar, Apple's appointment application, for example. If you own two or more Apple devices—an iPhone or iPad or two—and they all run iOS 6 or greater, open Calendar on one of them and enter a new appointment for today. Now, switch to another device, open Calendar, and go to your entries for today. Provided that you have activated iCloud on both devices, you'll find the very same appointment on the second device that you just entered on the first.

Here's what happened. When you entered it, your iPhone or iPad pushed the appointment to servers operated by Apple. The Calendar application on the second device was listening for changes to the calendar, found yours, and updated itself. You'll have the same experience whenever you enter a new contact, save a photo, buy music, and more.[1]

Naturally, you'd like to provide features like this to your own users. Fortunately, Apple has not kept iCloud to itself but opened it to app developers like you and me. Now when you write an iOS application, you'll be able to use the information in this book to add iCloud support that works on all of Apple's devices running iOS 5 or greater.[2] Users who install the application on each of their iOS devices will be able to store their data and keep it in sync. And we are talking about any kind of data: properties, configurations, documents, binary files, and even information in a relational database.

In this chapter, you will get acquainted with iCloud, learn how it works, and go over the steps to take to prepare an app to use the service. You'll learn about the following:

---

1. For an overview of how Apple uses iCloud in its own applications, see http://www.apple.com/icloud/features/. The example of the Calendar app is used just to show a familiar scenario where data synchronization happens. Although I am not sure, it is likely the Calendar app in iOS and Mac OS is not using iCloud API to synchronize.
2. While it's possible to add iCloud support to versions of your application that run on a Mac, we will not cover that topic in this book.

• Click HERE to purchase this book now. discuss

- What iCloud provides and how you can take advantage of it
- How iCloud handles data and synchronizes updates
- How to prepare an iOS project for iCloud

We will also introduce and start work on Grocery, the application that we're going to build together in this book to flex and show off iCloud's features (Section 1.3, *Introducing the Grocery Application*, on page 5).

By the end of this chapter you will have a clearer idea of the scenarios iCloud supports and the steps needed to start building an iCloud-enabled application. Let's begin by describing what iCloud does and how it works.

1.1 What Is iCloud?

iCloud is a cloud-based tool that can store data for an application at a central server and synchronize updates served up by the iPhones, iPads, or Macs that use it. For both developers and users, iCloud solves two problems: backup and data synchronization.

For backup, data for an application need only be made to adhere to certain formats and specifications and stored in one or more special folders that iCloud provides. For data synchronization, the application has to listen for iCloud notifications indicating changes have occurred and then, when one is received, resolve any conflicts that exist and update the local data store. You will learn to handle both cases in this book, but in order to understand how iCloud handles its data, you first need to understand how it works under the hood.

1.2 What's Behind iCloud

From an application's perspective, iCloud consists of one or more "special folders" whose contents iCloud synchronizes with files stored at a central location. This special folder is called a *ubiquity container*. An application can have one or more ubiquity containers, each of which is assigned its own unique *container ID* when you enable an application to use the service. As a user adds or modifies application data, iCloud pushes the changes to a central server, which in turn pushes them to other devices that have signed up to share it. An application doesn't need to query iCloud for updates to its ubiquity containers but instead simply queues itself as an observer. When notified of new content, the application takes steps to integrate it into its local data stores.

To make this mechanism perform efficiently, the contents of files in a container are broken into *chunks*. Whenever you change a file in a ubiquity container,

the synchronization mechanism pushes the bits that have changed, not the entire file. The same thing happens when an application is notified of changes made on other devices: the application running on your device receives only the bits that have changed and integrates them into the files in its ubiquity container.

The synchronization of data across devices is managed by a background process on each device known as the *daemon*. The daemon is not under the control of the developer, who is responsible for managing the main thread of a program. The daemon is an independent process, whose job is to detect changes to a resource (for example, a document or database) and send these changes to a central iCloud server. The daemon acts as a sort of middle man to the file system on a device. This is summarized in Figure 1, *Architecture of iCloud*, on page 4, which diagrams the flow of data between an application, its containers, and iCloud.

It will be up to you to write the code that opens and closes a file used by an iCloud-enabled application. Those operations will in turn trigger the read or write procedure that is managed by the daemon. Although this might seem inflexible, such an architecture relieves you of having to manage concurrency. Without the daemon, you would need to implement thread-safe procedures to read, write, and push changes to the cloud, not to mention managing file updates. iCloud takes care of these tasks as well as two others: bandwidth management and conflict resolution.

To optimize its consumption of bandwidth, especially on mobile devices that are battery powered most of the time, iCloud makes use of *metadata*. When a change occurs on a device, the first thing pushed to iCloud is metadata that describes it. This information includes, for example, the size of the file and the date and time it was modified. Metadata is also sent to iCloud when you work with media such as pictures, videos, or audio recordings. As soon as a save operation completes on such a resource file, a 1KB element pops up on the cloud to serve as a placeholder while the actual file is uploaded.

iCloud also breaks down files into *chunks* to simplify their push to the cloud when they are updated. Only the modified chunks are sent to iCloud, which saves bandwidth and also makes it easier to resolve conflicts. To detect conflicts between updates, only the modified chunks of a file need to be compared. Changes that don't conflict are merged with the existing iCloud file, while those that do will trigger notifications so the developer can implement policies to resolve them, which could include asking the user to pick the "right" version.

• Click HERE to purchase this book now. discuss

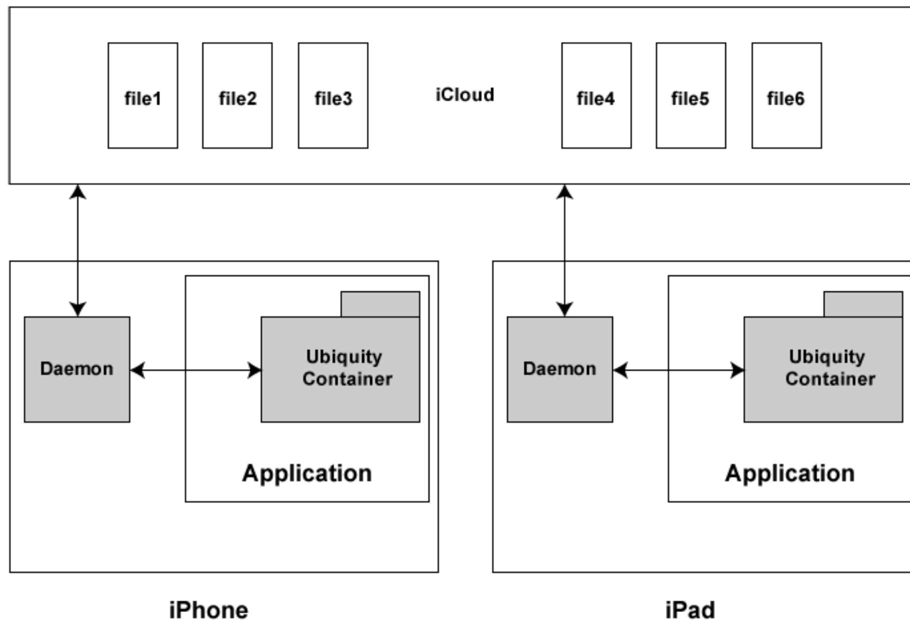


Figure 1—Architecture of iCloud. Each device has a daemon in charge of pushing and receiving changes to and from iCloud. Each application has one or more ubiquity containers.

Changes to the contents of an iCloud-enabled application file are pushed to iCloud as quickly as possible. Such a policy keeps the data on the server fresh. But the way iOS pulls changes from iCloud depends on the characteristics of the host device, such as the quality of the connection (3G, LTE, Wi-Fi) and the status of its battery. In general, changes are pulled when they are "appropriate" and won't degrade performance. On devices, such as phones with limited battery life, iOS pulls changes only when it needs to, such as when you open or close a document. The use of metadata, however, guarantees that the devices that share the application are synchronized and that an iPad or iPhone are "on the same page," even if one of them has yet to integrate the most recent changes made to an application file.

To sum up, when you create or change an application file on a device, its metadata (name, creation date, and so on) is pushed immediately to iCloud. When you run the application on another mobile device, that device will be "aware" that new content is available, but the changes will be replicated there only when

- you open the file or

- the daemon decides that downloading the file will not impact the performance of the OS.

Although it's important to be aware of such policies, you will not need to write "special" code to address them, since the daemon does all the work. If a file is unchanged (for example, it was created on the current device or it was pulled recently from iCloud), its contents will be displayed without delay when you open it. If changes have occurred, the daemon will start downloading the file and notify you when it's done. We will look more closely at this behavior as we develop our Grocery application.

1.3 Introducing the Grocery Application

To show what's possible with iCloud, we're going to build a real application that uses it. I'll name the app Grocery. Grocery will allow users to share a common grocery list between their devices. Each item in the list will have a name, will include an image (so we can show how to store binary files), and will be assigned to one or more categories (to show how we can work with relational data). When a user creates or modifies an item on one device, it will be replicated on any others that are connected to the same iCloud account.

The application will have two views, as shown in Figure 2, *Two views of the Grocery application*, on page 6. The first is a table view that displays the list of grocery items to be bought. This view also lets users add and delete items. The second view will appear whenever the user taps an item in the first view and will display some pertinent details about it, such as its name and an image of the item.

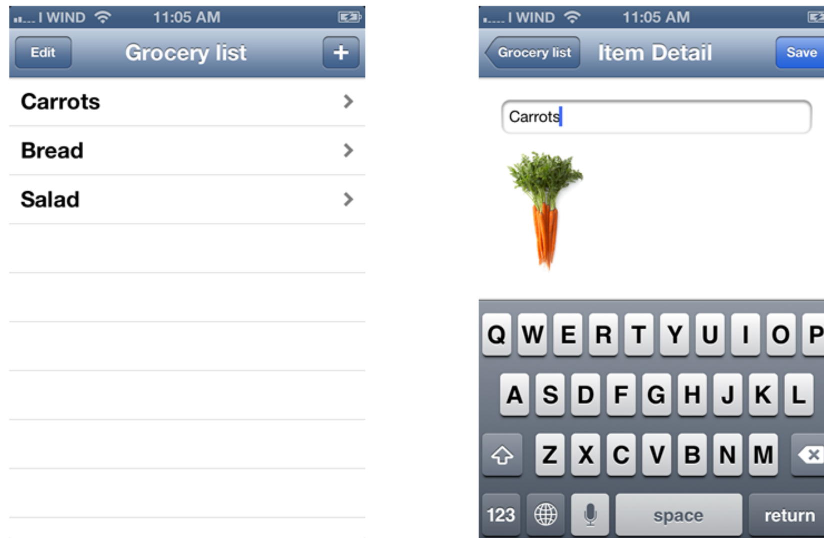


Figure 2—Two views of the Grocery application. The first view shows a list of items, and the second shows a detailed view of a single item.

As we move through the book, we're going to encounter slightly different versions of this application, but its core will remain the same: two views, one to display the list and one to show the details of each item.

While the Grocery application is a simple one, it's complex enough for us to learn some important iCloud skills, such as building a data model, reacting to update notifications, detecting and resolving conflicts, and working with relational data.

In the next section, we will focus on the very first steps you'll need to get started with iCloud.