MultiDocumentPlusUUID is a working example of an iOS 7 application with cloud-syncing UIManagedDocuments. This project is open-sourced and released under the MIT License by Don Briggs at DonBriggs/MultiDocumentPlusUUID · GitHub

_____

Purpose
MultiDocumentPlusUUID provides a limited working example of cloud-synced UIManagedDocuments. It derives from Rich Warren's MultiDocument example: Freelance Mad Science Labs - Blog - Syncing multiple Core Data documents using iCloud

_____

Credits
WWDC 2013 Video - Session 207 [What's New in Core Data and iCloud](https://developer.apple.com/wwdc/videos/index.php?id=207)

See Drew McCormack's note:
http://mentalfaculty.tumblr.com/post/25241910449/under-the-sheets-with-icloud-and-core-data
Under the Sheets with iCloud and Core Data: Troubleshooting
> *"Unfortunately, the most apt conclusion is probably that iCloud syncing of Core Data is not really ready for prime time, at least not for any app with a complex data model. If you have a simple model, and patience, it is doable, even if very few have achieved a shipping app at this point."*
Drew's observation was certainly true at the time—June 16, 2012. This MultiDocumentPlusUUID effort has had a rather long and tragic history; it seems to break spectacularly with major releases of iOS. Many Bothans died... With iOS 7, we're in somewhat better shape. Perhaps we've achieved a plateau of stability in the Core Data Cloud-sync API.

See Erica Sudun's iOS 5 Developer's Cookbook:
http://books.google.com/books?id=YeHQzA6UrcEC&pg=PT1173&lpg=PT1173&dq=yorn+BOOL+ios&source=bl&ots=vynUYXoVpH&sig=PCjk79J9uGt0EHHNgNF28JpQ-HA&hl=en&sa=X&ei=euGFT6K2BY-s8ATBgvyXCA&ved=0CB4Q6AEwAA#v=onepage&q=yorn%20BOOL%20ios&f=false

See: David Trotz
http://stackoverflow.com/questions/18971389/proper-use-of-icloud-fallback-stores
https://github.com/dtrotzjr/APManagedDocument

This sample code includes Stav Ashuri's UIBAlertView, Copyright (c) 2013. Thanks!
See: stavash/UIBAlertView · GitHub

_____

A Broad Overview
Primarily, MultiDocumentPlusUUID uses UIManagedDocument directly. Optionally, it uses a light-weight subclass (RobustDocument) to demonstrate an approach to handling errors in Core Data.

The root view shows all the application's documents in a table view: those created locally and those discovered through NSMetadataQuery.

As with Warren's original, each document associates one-to-one with a dictionary called a record. The record acts as a helper class for the document, and avoids (postpones?) subclassing UIManagedDocument. Each cell in the root view's table view associates one-to-one with a record, and displays the status of its record's document.

The data model is simple; each document has a singleton TextEntry. Its fields include a text string and a modification date. Neglect the rest.
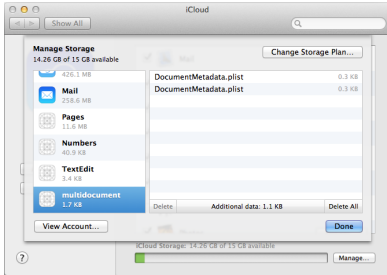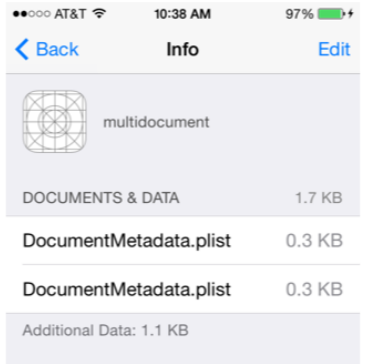
The application uses UUIDs to distinguish its documents. Each document has its own UUID to distinguish it from any documents with the same file name. Different devices might create documents with the same name near the same time—"Untitled" for example—and we must be able to distinguish them. In MultiDocumentPlusUUID, a document's UUID precedes its file name in both its sandbox and cloud URLs. The UUID also provides the value of `NSPersistentStoreUbiquitousContentNameKey` in the document's persistent store options.

MultiDocumentPlusUUID can provide update latencies of cloud-syncing between two devices viewing the same document.
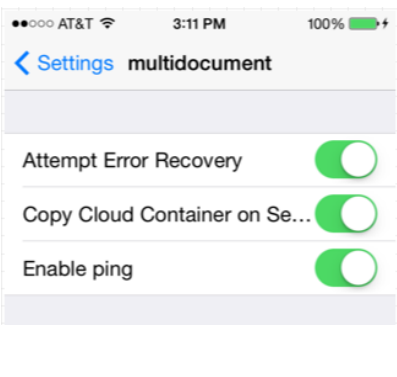
MultiDocumentPlusUUID focusses on creating and discovering documents. Generally, documents should persist between launches in the sandbox, but this limited example neglects all that. Nor is MultiDocumentPlusUUID agile over iCloud accounts. Stick to one iCloud account at a time across all devices in your tests.
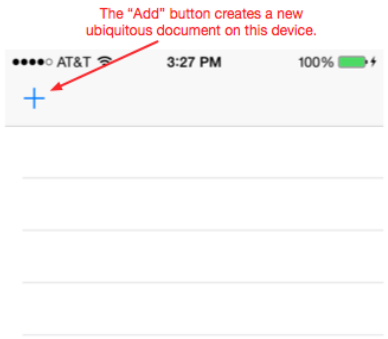
How to "Make it Go"

Add your own entitlements plist to enable Cloud persistence. Compile the application and run it on two devices under Xcode's debugger. Make sure on both devices to log into the same iCloud account. Typically, it seems best to clear the application's cloud "Storage" (as in "Storage and Backup") after each session. It's handy to do this on the desktop, but sometimes it's necessary to visit the Settings of each device, too. Don't be discouraged if you find it necessary to do so.

	On OS X, use Preferences->iCloud
	On iOS, use Settings->iCloud->Storage and Backup->ManageStorage->multidocument

Use Settings.app to set a few properties on both devices. In the storyboard below, the first device is an iPhone named "A", and the second device is an iPad named "B".
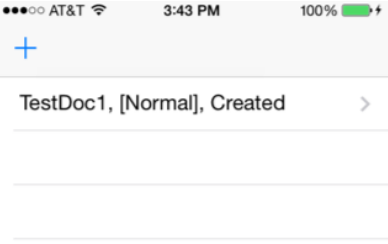
	"Attempt Error Recovery" uses RobustDocument instead of UIManagedDocument. It's a merely a starting stub for managing Core Data errors.
	"Copy Cloud Container on Segue" enables a handy tool: copy the cloud container to the sandbox. Apple suggests including such a copy in your bug reports and Technical Incidents.
	"Enable Ping" enables automatic pinging: when the detail view of a device detects a change in the document's object model, it automatically adds a short ping message to TextEntry.text and saves the document. Just two devices can participate safely.

The "Add" button creates a new ubiquitous document on this device.

●●●●○ AT&T 🛜   3:27 PM   100% 🔋⚡

＋
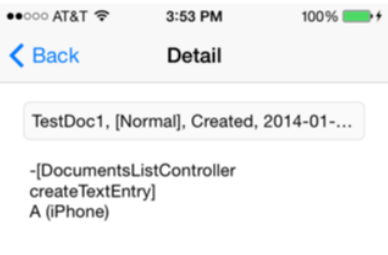
Initially, the root table views on both devices show no documents —if you've successfully cleared any zombie documents (below). Touch the "Add" button in the upper left to create a new ubiquitous document on the first device.

(Suddenly with OS X 10.9.2 Build 13C39, it seems much more likely that this initial screen will display zombie documents— documents previously deleted from the application's cloud "Data and Storage." A workaround is simply to quit and restart MultiDocumentPlusUUID from Xcode; the zombies vanish. Your mileage may vary...}

●●●○○ AT&T 🛜   3:43 PM   100% 🔋⚡

＋

TestDoc1, [Normal], Created   〉

Very quickly, the device's created document appears in its root table view. The row shows the document's status. The status includes the document's file name, document state, and whether it was created on this device or discovered. Touch the row to segue to the detail view, and inspect the object graph.

(In a "real" application, the user should be unaware which device created a document; this example application is for developers.)

●●●○○ AT&T 🛜   3:53 PM   100% 🔋⚡

❮ Back   **Detail**

TestDoc1, [Normal], Created, 2014-01-...

-[DocumentsListController createTextEntry]
A (iPhone)

The newly created document's object graph is quickly available for inspection in the detail view. The TextEntry's text property shows the name and model of the device that created it.

---

iPad 🛜   3:43 PM   100% 🔋⚡

＋

TestDoc1, [Normal], Discovered, Current, 100% Dn   〉

Meanwhile, the second device discovers the existence of newly created document within a few seconds. The document's status includes its download status and its "percent downloaded." (A document is discovered through an instance of NSMetadataItem, which has a value for NSMetadataUbiquitousItemPercentDownloadedKey.) But, the newly discovered document's object graph is not immediately available for inspection. Its cell is gray and disabled.

iPad 📶   3:45 PM   100% 🔋⚡

+

TestDoc1, [Normal], Discovered, Current, 100% Dn   ›
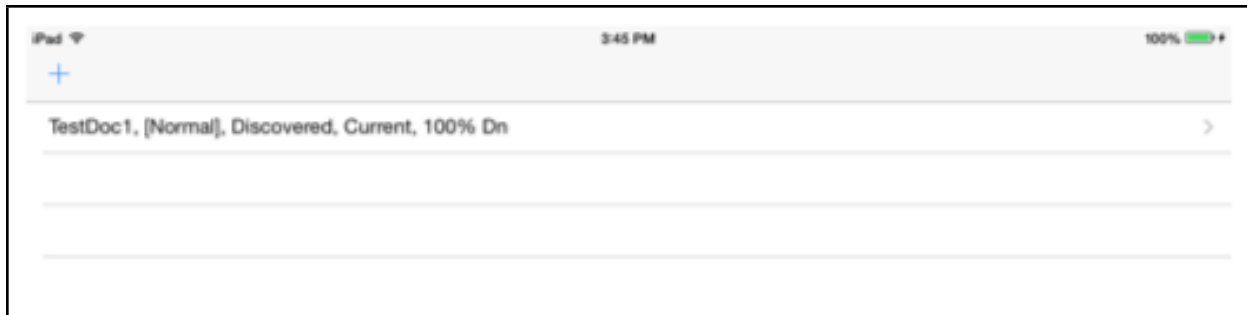
Perhaps 90 seconds later, the discovered document's object graph becomes available for inspection. Its cell becomes white and enabled: touch the row to segue to the detail view. The root view controller observes all its documents' NSPersistentStoreDidImportUbiquitousContentChangesNotification. Its first occurrence signals the availability of a document's object graph. (If the delay for that signal is much longer than 90 seconds, expect a Core Data error.)

---

iPad 📶   4:26 PM   100% 🔋⚡

‹ Back   Detail

TestDoc1, [Normal], Discovered, Current, 100% Dn, 2014-01-17 21:43:11 +0000

-[DocumentsListController createTextEntry]
A (iPhone)
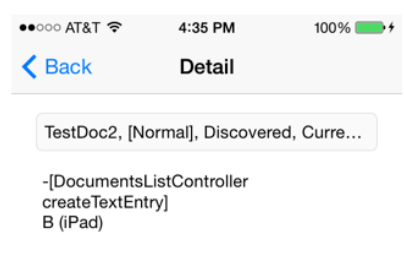
The detail view on the second device shows the same object graph as on the first device, where the document was created.

---

iPad 📶   4:29 PM   100% 🔋⚡

+

TestDoc1, [Normal], Discovered, Current, 100% Dn   ›

TestDoc2, [Normal], Created   ›

Similarly, we can create a document on the second device (TestDoc2). (A document's file name depends on its table view row in the creating device.)

●●○○○ AT&T 📶   4:32 PM   100% 🔋⚡

+

TestDoc1, [Normal], Created   ›

TestDoc2, [Normal], Discovered, Current, 100% Dn   ›
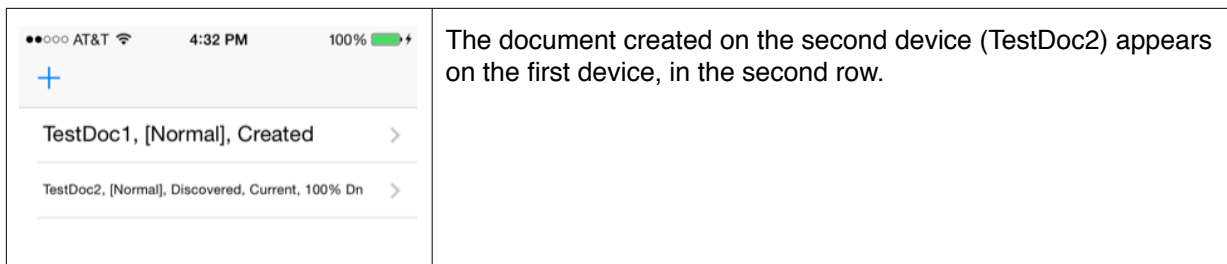
The document created on the second device (TestDoc2) appears on the first device, in the second row.

	In due time, TestDoc2's object graph becomes available on the first device.
••○○○ AT&T 🛜 4:35 PM 100% 🔋⚡ **‹ Back** **Detail** TestDoc2, [Normal], Discovered, Curre… -[DocumentsListController createTextEntry] B (iPad)	

Pinging

MultiDocumentPlusUUID includes a stunt to demonstrate sustained cloud syncing and to record the latencies of updates between two devices. With "Enable Ping" enabled on both devices, and both devices' detail views showing the same document, when either detail view detects a change to its document's object graph (NSPersistentStoreDidImportUbiquitousContentChangesNotification), it appends a short ping message programmatically.

iPad 🛜   4:26 PM   100% 🔋⚡ **‹ Back**   **Detail** TestDoc1, [Normal], Discovered, Current, 100% Dn, 2014-01-17 21:43:11 +0000 -[DocumentsListController createTextEntry] A (iPhone)
••○○○ AT&T 🛜 4:52 PM 100% 🔋⚡ **‹ Back**   **Detail** TestDoc1, [Normal], Created, 2014-01-… -[DocumentsListController createTextEntry] A (iPhone)
On both devices, select the same document—"Created" on one; "Discovered" on the other—and segue to both detail views.

‹ Back   **Detail**

TestDoc1, [Normal], Discovered, Current, 100% Dn, 2014-01-17 21:43:11 +0000

-[DocumentsListController createTextEntry]
A (iPhone) Hello, document. Having a wonderful time, wish you were ubiquitous.

1	2	3	4	5	6	7	8	9	0	⌫
-	/	:	;	(	)	$	&	@		Done
#+=	undo		.	,	?	!	'	"		#+=
ABC							ABC			⌨

Modify the TextEntry text on one device. (Here, we choose the iPad.)

‹ Back   **Detail**

TestDoc1, [Normal], Created, 2014-01-...

-[DocumentsListController
createTextEntry]
A (iPhone) Hello, document. Having a
wonderful time, wish you were
ubiquitous. A pings

Within a few seconds, the modified text appears on the iPhone. Because "Enable Ping" is set to YES in the iPhone's Settings,app, the receiving device quickly appends a short ping message: "A pings".

‹ Back   **Detail**

TestDoc1, [Normal], Discovered, Current, 100% Dn, 2014-01-17 22:59:50 +0000

-[DocumentsListController createTextEntry]
A (iPhone) Hello, document. Having a wonderful time, wish you were ubiquitous. A pings B pings

Within a few seconds, the iPad detects the iPhone's modification and appends its own ping message, "B pings", because its Settings.app also has "Enable Ping" set to YES.

‹ Back   **Detail**
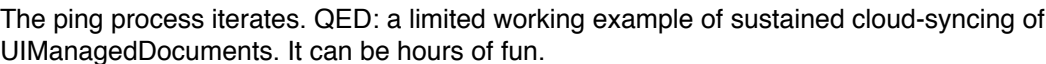
TestDoc1, [Normal], Discovered, Current, 100% Dn, 2014-01-17 23:16:00 +0000

-[DocumentsListController createTextEntry]
A (iPhone) Hello, document. Having a wonderful time, wish you were ubiquitous. A pings B pings

‹ Back   **Detail**

TestDoc1, [Normal], Created, 2014-01-...

-[DocumentsListController
createTextEntry]
A (iPhone) Hello, document. Having a
wonderful time, wish you were
ubiquitous. A pings B pings A pings B
pings A pings B pings A pings B pings A
pings B pings A pings B pings A pings B
pings A pings B pings A pings B pings A
pings B pings A pings B pings A pings B
pings A pings B pings A pings B pings A
pings B pings A pings B pings A pings B
pings A pings B pings A pings B pings A
pings B pings A pings B pings A pings B
pings A pings B pings A pings B pings A
pings B pings A pings B pings A pings B
pings A pings B pings A pings B pings A
pings B pings A pings B pings A pings B
pings A pings B pings A pings B pings A
pings B pings A pings B pings A pings B
pings A pings B pings A pings B pings A
pings B pings A pings B pings A pings B
pings A pings B pings A pings B pings A
pings B pings A pings B pings A pings B

The ping process iterates. QED: a limited working example of sustained cloud-syncing of UIManagedDocuments. It can be hours of fun.

Latencies:

After many ping iterations, set a breakpoint in `-ping:` (`DetailViewController +Pinging`). In the debugger console, print the latencies:

```
(lldb) po [self reportLatencies]
2014-01-17 14:17:18.623 multidocument[221:60b] 22.40558397769928
5.156464993953705
4.969072997570038
7.16798198223114
. . .
5.988353967666626
4.766466975212097
6.195563971996307
(lldb)
```

Here's a graph of typical latencies:

Typical Cloud-Sync Latencies [seconds], iOS 7.1 (11D5127c)

Untitled 1
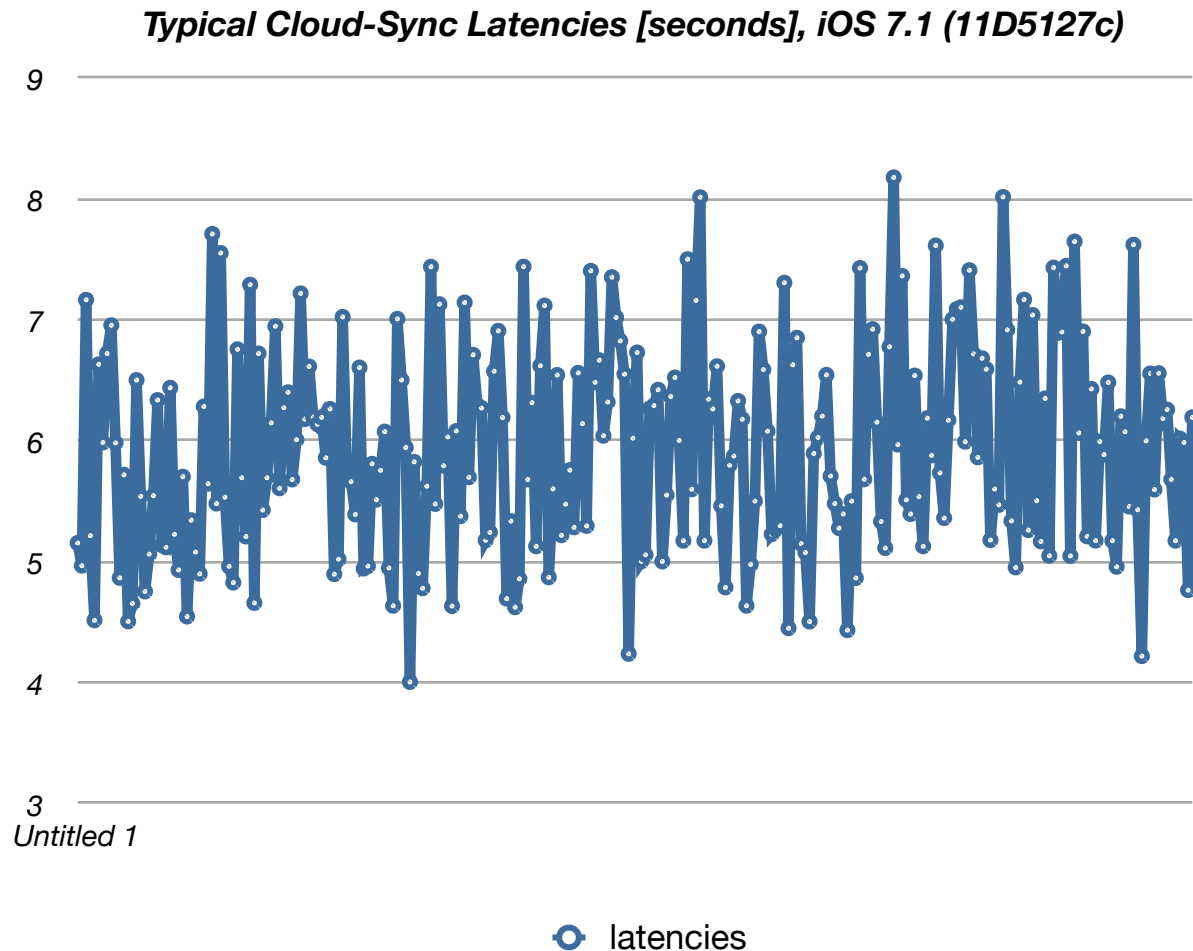
⊙ latencies

Note that the latencies above came from just two devices interoperating in a very simple environment, syncing one very simple document. Your mileage may vary...