Stanford CS193p

Developing Applications for iOS
Fall 2011

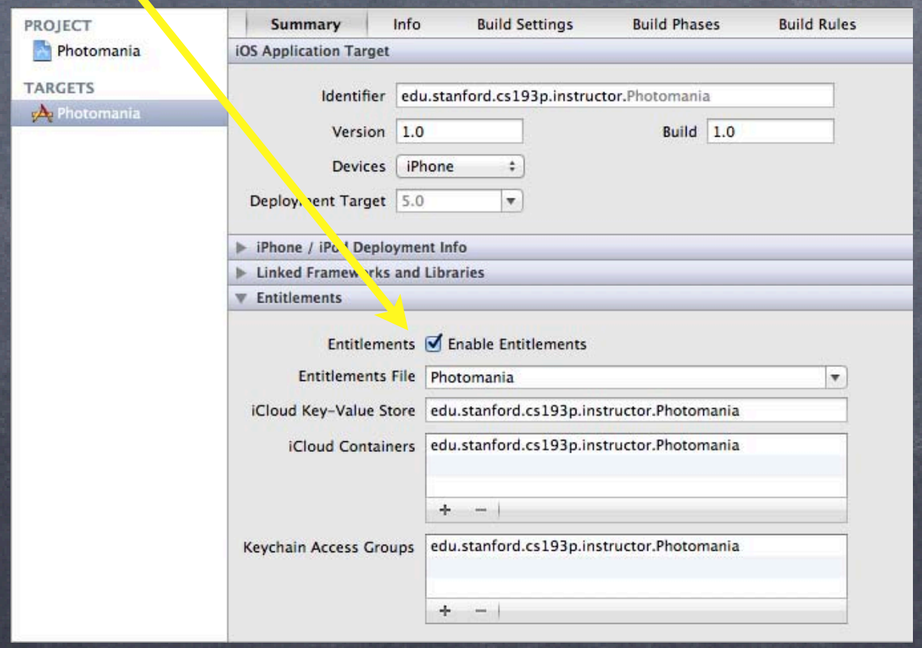
Today

- iCloud

  Sharing documents among a user's devices

iCloud

- Fundamentally: nothing more than a URL of a shared directory
  - However, since it is over the network, there are lots of things to consider.
  - Most importantly, latency and the ramifications of shared access.
- Entitlements
  - In order for your application to access the cloud, you must set up its entitlements.
  - Luckily this is a single button click in Xcode:



iCloud

- The URL

  Assuming you just have one container in your Entitlements …

  `[[NSFileManager defaultManager] URLForUbiquityContainerIdentifier:nil];`

  Usually you put your documents in the `@"Documents"` directory (special) inside this URL.

- `NSFileManager`

  You can use it, but you don't want to use it in the main thread (it's happening over the network).

  Don't use it to enumerate the contents of directories (need an always-updating query for that).

  And you also will want to "coordinate" any file operations with other devices.

  Coordination happens among `NSFilePresenters` using the `NSFileCoordinator` class.

- `NSFilePresenter`

  This class is an abstraction for something that "presents" the data of a file/directory to the user.

  Your presenter would get involved whenever a coordinated change to the file/directory occurs.

  Not going to cover `NSFilePresenter`'s API in this class.

  Instead, we'll talk about `UIManagedDocument` (which is an `NSFilePresenter`).

iCloud

- **UIDocument**

  An NSFilePresenter.

  Abstraction for a document.

  Automatically coordinates changes to the document.

  Primary methods you'd override would be …

  ```
  - (id)contentsForType:(NSString *)type error:(NSError **)error;
  - (BOOL)loadFromContents:(id)contents ofType:(NSString *)type error:(NSError **)error;
  ```

  If your data is stored in a Core Data database, though, you'd use UIManagedDocument.

- **UIManagedDocument**

  Fully iCloud capable.

  You need only sign up for the appropriate notifications to keep your UI in sync.

  Put your UIManagedDocuments into the Documents directory in iCloud.

  ```
  iCloudDocumentsURL = [iCloudURL URLByAppendingPathComponent:@"Documents"];
  ```

iCloud

- Only UIManagedDocument changes should be synced to iCloud
  Uploading only a log of changes is a lot better performing than uploading the whole database!
  To get this, you must set your document's persistentStoreOptions dictionary to include these …
  NSPersistentStoreUbiquitousContentNameKey (the "name" of the document)
  NSPersistentStoreUbiquitousContentURLKey (where all change logs are stored in iCloud)
  Change logs directory (ContentURLKey) should not be stored in the Documents directory in iCloud!
  ContentURLKey is something like [iCloudURL URLByAppendingPathComponent:@"CoreData"]
  All of your documents can share the same ContentURLKey.
- Document Metadata
  Strictly speaking, you should get the ContentNameKey from the document's metadata.
  You can read it from DocumentMetadata.plist inside the document's file wrapper.
  For example, if "docURL" were the URL that pointed to the UIManagedDocument in the cloud,
  then create a dictionary from the following URL (coordinated!) to get the values for the keys …

```
NSURL *metadataURL = [docURL URLByAppendingPathComponent:@"DocumentMetadata.plist"];
```

  Then set the keys in persistentStoreOptions, and then call openWithCompletionHandler:.

iCloud

- Enumerating what's in the Cloud

  Create a query, start it, then watch for NSNotifications that the results have changed.

- Creating a query

```
NSMetadataQuery *query = [[NSMetadataQuery alloc] init];
query.searchScopes = [NSArray arrayWithObjects:scope1, scope2, nil];
```

  NSMetadataQueryUbiquitousDocumentsScope is all files in the Documents directory.

  NSMetadataQueryUbiquitousDataScope is all files NOT in the Documents directory.

```
query.predicate =
    [NSPredicate predicateWithFormat:@"%K like '*'", NSMetadataItemFSNameKey]; // all
```

- Starting/stopping the query

  [query startQuery] or [query stopQuery]

- Enabling/disabling NSNotifications

  [query enableUpdates] or [query disableUpdates]

  Probably a good idea to do this in viewWillAppear:/viewWillDisappear:

iCloud

- Signing up to receive the query NSNotifications

```
NSNotificationCenter *center = [NSNotificationCenter defaultCenter];
[center addObserver:self
           selector:@selector(processQueryResults:)
               name:NSMetadataQueryDidFinishGatheringNotification // first results
             object:query];
[center addObserver:self
           selector:@selector(processQueryResults:)
               name:NSMetadataQueryDidUpdateNotification // subsequent updates
             object:query];
```

- Don't forget to remove yourself as an observer

```
- (void)dealloc {
    [[NSNotificationCenter defaultCenter] removeObserver:self];
}
```

iCloud

- Processing NSMetadataQuery NSNotifications

```
- (void)processQueryResults:(NSNotification *)notification
    [query disableUpdates];
    int resultCount = [query resultCount];
    for (int i = 0; i < resultCount; i++) {
        NSMetadataItem *item = [query resultAtIndex:i];
        NSURL *url = [item valueForAttribute:NSMetadataItemURLKey];
        // do something with the urls here
        // but remember that these are URLs of the files inside the file wrapper!
    }
    [query enableUpdates];
}
```

iCloud

- Coordinating changes

  Should be done whenever you access an iCloud URL using NSFileManager.

  Do this outside the main thread!

- NSFileCoordinator

```
NSFileCoordinator *coordinator = [[NSFileCoordinator alloc] initWithFilePresenter:nil];
NSError *coordinationError;
[coordinator coordinateReadingItemAtURL:(NSURL *)url
                                options:(NSFileCoordinatorReadingOptions)options
                                  error:(NSError **)error
                             byAccessor:^(NSURL *urlToUse) {
    // do your NSFileManager stuff here using urlToUse
}];
if (coordinationError) { } // handle error
```

  Similar methods for coordinating writing, or reading and writing together, or multiple URLs.

iCloud

- ## Document State

  It is more important to pay attention to `documentState` in an iCloud environment.
  There are two states that occur more often now: `EditingDisabled` and `InConflict`.
  It is also important to watch for `SavingError` state (and perhaps retry your saves).

- ## Conflict

  What if a device detached from the network changed a document that another device changed?
  And then the detached device reattached and tried to apply the change and it conflicted?
  You must manage this conflict by looking for the `InConflict` document state.
  When it happens, you must decide which version of the document to use and/or merge changes.
  Probably want to set your `NSManagedObjectContext`'s `mergePolicy` to other than default (`Error`).
  Then update the old, conflicting `NSFileVersions` to no longer conflict (and remove those versions).
  See the demo for a simple example of how to just take the most recent version.

- ## Editing Disabled

  Don't let the user modify a document that is in the `EditingDisabled` state.

iCloud

- Moving a file to or from iCloud

  It would be nice to allow your users to choose whether a document is shared via iCloud or not.
  You can switch a document from being shared or not at any time that you are connect to iCloud.
  Always do this outside the main thread.

- API

  Example of exporting to iCloud (you can also go the opposite direction) …

```
dispatch_async(dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0), ^{
    NSURL *localURL = …; // url of document to transfer to iCloud
    NSString *name = [localURL lastPathComponent];
    NSURL *iCloudURL = [iCloudDocumentsURL URLByAppendingPathComponent:name];
    NSError *error = nil;
    [[[NSFileManager alloc] init] setUbiquitous:YES
                                      itemAtURL:localURL
                                 destinationURL:iCloudURL
                                          error:&error]; // move the document
});
```

iCloud

- Sharing a file via iCloud

  iCloud is mostly for a single user to share stuff between his or her own devices.
  However, it is also possible to temporarily export a URL to share a file between users.
  Only files (not directories) maybe exported in this way.
  The URL points to a copy of the file (so it is a read-only sharing mechanism).

- API

```
NSURL *urlToShare = …;
NSDate *date;
NSError *error;
NSURL *sharedURL =
    [[NSFileManager defaultManager] URLForPublishingUbiquitousItemAtURL:urlToShare
                                                         expirationDate:&date
                                                                  error:&error];
```

iCloud

- NSUbiquitousKeyValueStore

  Like NSUserDefaults, but iCloud-style.

  Very limited in size (64kb) (overall and per value).

  Use `[NSUbiquitousKeyValueStore defaultStore]` to get the shared instance.

- Set and get just like NSUserDefaults

```
NSUbiquitousKeyValueStore *store = [NSUbiquitousKeyValueStore defaultStore];
[store setObject:@“MyValue” forKey:@“MyKey”];
[store synchronize]; // not instantaneous! this is networking!
```

  Synchronize synchronizes both ways, so a good idea to call it in your app’s launch somewhere.

iCloud

- Since store can change at any time, need NSNotification

```
NSNotificationCenter *center = [NSNotificationCenter defaultCenter];
[center addObserver:self
           selector:@selector(ubiquitousKeyValueStoreChanged:)
               name:NSUbiquitousKeyValueStoreDidChangeExternallyNotification
             object:[NSUbiquitousKeyValueStore defaultStore]];

- (void)ubiquitousKeyValueStoreChanged:(NSNotification *)notification
{
    notification.userInfo contains …
    NSUbiquitousKeyValueStoreChangeReasonKey (Server, InitialSync, QuotaViolation)
    NSUbiquitousKeyValueStoreChangedKeysKey (NSArray of NSStrings of keys that changed)
}
```

Does NOT get called if YOU change a key, only if the cloud does!

Demo

- iCloud (two-day demo)
  - Finding out what documents are stored in the cloud
  - Storing/Retrieving Core Data documents (`UIManagedDocuments`)
  - Deleting documents from the cloud (`NSFileCoordinator`)
  - Noticing changes to documents in real time
  - Handling file version conflicts by watching for `documentState` changes
  - Cloud `NSUserDefaults` equivalent
- Along the way …
  - Getting a "random queue" to do work on (rather than creating a named one)
  - Generic iCloud-document-handling view controller
  - `NSNotifications`
  - Flexible table view `prepareForSegue:sender:` implementation
  - Migrating (automatically) your Core Data schema to a new version
  - Using `AskerViewController` again
  - Editing table views (by allowing rows to be deleted)

Coming Up

- Thursday
  More demo!
- Friday Section
  OpenGL