Stanford CS193p

Developing Applications for iOS
Fall 2011

Today

- Persistence

  How to make things stick around between launchings of your app (besides NSUserDefaults)

Persistence

- Property Lists

  Use `writeToURL:atomically:` and `initWithContentsOfURL:` in `NSArray` or `NSDictionary`

  Or `NSUserDefaults` if appropriate.

  Also `NSPropertyListSerialization` converts Property Lists to/from NSData.
- Archiving Objects
- Storing things in the Filesystem
- SQLite
- Core Data

  Next week

Archiving

- There is a mechanism for making ANY object graph persistent
  Not just graphs with NSArray, NSDictionary, etc. in them.
- For example, the view hierarchies you build in Interface Builder
  Those are obviously graphs of very complicated objects.
- Requires all objects in the graph to implement NSCoding protocol
  ```
  - (void)encodeWithCoder:(NSCoder *)coder;
  - initWithCoder:(NSCoder *)coder;
  ```
- It is extremely unlikely you will use this in this course
  Certainly not in the homework assignments.
  But almost certainly not in your Final Project either.
  There are other, simpler, (or more appropriate), persistence mechanisms (more on this later).

Archiving

- Object graph is saved by sending all objects `encodeWithCoder:`

```
- (void)encodeWithCoder:(NSCoder *)coder {
    [super encodeWithCoder:coder];
    [coder encodeFloat:scale forKey:@“scale”];
    [coder encodeCGPoint:origin forKey:@“origin”];
    [coder encodeObject:expression forKey:@“expression”];
}
```

Absolutely, positively must call `super`’s version or your superclass’s data won’t get written out!

- Object graph is read back in with `alloc`/`initWithCoder:`

```
- initWithCoder:(NSCoder *)coder {
    self = [super initWithCoder:coder];
    scale = [coder decodeFloatForKey:@“scale”];
    expression = [coder decodeObjectForKey:@“expression”];
    origin = [coder decodeCGPointForKey:@“origin”];  // note that order does not matter
}
```

Archiving

- NSKeyed[Un]Archiver classes used to store/retrieve graph

  Storage and retrieval is done to NSData objects.

  NSKeyedArchiver stores an object graph to an NSData …

```
+ (NSData *)archivedDataWithRootObject:(id <NSCoder>)rootObject;
```

  NSKeyedUnarchiver retrieves an object graph from an NSData …

```
+ (id <NSCoder>)unarchiveObjectWithData:(NSData *)data;
```

- What do you think this code does?

```
id <NSCoder> object = …;
NSData *data = [NSKeyedArchiver archivedDataWithRootObject:object];
id <NSCoder> dup = [NSKeyedArchiver unarchiveObjectWithData:data];
```

  It makes a "deep copy" of object.

  But beware, you may get more (or less) than you bargained for.

  Object graphs like "view hierarchies" can be very complicated.

  For example, does a view's superview get archived?

File System

- Your application sees iOS file system like a normal Unix filesystem
  It starts at /.
  There are file protections, of course, like normal Unix, so you can't see everything.
- You can only WRITE inside your "sandbox"
- Why?
  Security (so no one else can damage your application)
  Privacy (so no other applications can view your application's data)
  Cleanup (when you delete an application, everything its ever written goes with it)
- So what's in this "sandbox"
  Application bundle directory (binary, `.storyboards`, `.jpgs`, etc.). This directory is NOT writeable.
  Documents directory. This is where you store permanent data created by the user.
  Caches directory. Store temporary files here (this is not backed up by iTunes).
  Other directories (check out `NSSearchPathDirectory` in the documentation).

File System

- What if you want to write to a file you ship with your app?
  Copy it out of your application bundle into the documents (or other) directory so it's writeable.
- How do you get the paths to these special sandbox directories?
  Use this `NSFileManager` method …

```
- (NSArray *)URLsForDirectory:(NSSearchPathDirectory)directory      // see below
                    inDomains:(NSSearchPathDomainMask)domainMask;  // NSUserDomainMask
```

- Notice that it returns an `NSArray` of paths (not a single path)
  Since the file system is limited in scope, there is usually only one path in the array in iOS.
  No user home directory, no shared system directories (for the most part), etc.
  Thus you will almost always just use `lastObject` (for simplicity).
- Examples of `NSSearchPathDirectory` values
  `NSDocumentsDirectory`, `NSCachesDirectory`, `NSAutosavedInformationDirectory`, etc.

File System

- NSFileManager

  Provides utility operations (reading and writing is done via NSData, et. al.).

  Check to see if files exist; create and enumerate directories; move, copy, delete files; etc.

  Just alloc/init an instance and start performing operations.

  Thread safe (as long as a given instance is only ever used in one thread).

```
NSFileManager *manager = [[NSFileManager alloc] init];
- (BOOL)createDirectoryAtPath:(NSString *)path
  withIntermediateDirectories:(BOOL)createIntermediates
                   attributes:(NSDictionary *)attributes  // permissions, etc.
                        error:(NSError **)error;
- (BOOL)isReadableFileAtPath:(NSString *)path;
- (NSArray *)contentsOfDirectoryAtPath:(NSString *)path error:(NSError **)error;
```

  Has a delegate with lots of "should" methods (to do an operation or proceed after an error).

  And plenty more. Check out the documentation.

File System

- NSString

  Path construction methods and reading/writing strings to files.

```
- (NSString *)stringByAppendingPathComponent:(NSString *)component;
- (NSString *)stringByDeletingLastPathComponent;
- (BOOL)writeToFile:(NSString *)path
         atomically:(BOOL)flag
           encoding:(NSStringEncoding)encoding  // e.g. ASCII, ISOLatin1, etc.
              error:(NSError **)error;
- (NSString *)stringWithContentsOfFile:(NSString *)path
                          usedEncoding:(NSStringEncoding *)encoding
                                 error:(NSError **)error;
```

  And plenty more. Check out the documentation.

SQLite

- SQL in a single file

  Fast, low memory, reliable.

  Open Source, comes bundled in iOS.

  Not good for everything (e.g. not video or even serious sounds/images).

  Not a server-based technology (not great at concurrency, but usually not a big deal on a phone).

- API

```
int sqlite3_open(const char *filename, sqlite3 **db);   // get a database into db
int sqlite3_exec(sqlite3 *db,                           // execute SQL statements
                 const char *sql,
                 int (*callback)(void *, int, char **, char **),
                 void *context,
                 char **error);
int mycallback(void *context, int count, char **values, char **cols);  // data returned
int sqlite3_close(sqlite3 *db);                         // close the database
```

Coming Up

- Next Week

  Core Data (object-oriented database)

- Tomorrow's Section

  Time Profiler

  How to measure the performance of your application