iPhone Programming

CMSC 498i – Spring 2010

Miscellaneous

Lecture #22 – Chuck Pisula

Performance

Tips & Tricks

Performance Tips – Drawing

- Drawing is slow
- Drawing is slow
- Drawing is slow

Performance Tips – Drawing

- Avoid drawing
  - Use pre-rendered images as much as possible
  - Use UIImageView
- Should only need to draw a view once unless the bounds change
  - Use appropriate **contentMode** instead of redrawing when possible
- Avoid layout – delay all your setFrame: calls to -layoutSubviews
- If you are **repeatedly redrawing** something, draw into an image with `UIGraphicsBeginImageContext` and stamp that over and over

Performance Tips – Drawing

- Less overlapping views is better
- Mark all you views as **opaque** as much as possible
- In general, less views is better
  - Especially for scrolling
  - Use separate view for parts needing frequent redraws
- In layout/draw methods, avoid anything but layout/drawing
  - Especially for scrolling content
  - Keep other calculations to a minimum

Performance Tips

- Don’t be a memory hog
  - Allocate lazily
  - Keep memory use low, use inner **pools** if necessary
  - Respond to memory notifications, and override points like `–viewDidUnload`
- Cache wisely (reuse)
  - Candidates for caching – Slow to create, small in size
- `UITableView` Tips
  - `UITableView` cell reuse APIs
  - Avoid `-reloadData`, and instead use `-setNeedsDisplay` for updates
- Don’t poll – use notifications, target / action, delegation, etc…

Performance Tips

- Never block the main thread
- Apps must launch fast, or face the consequences
- Use asynchronous API whenever possible

Performance Tips

- In **hot spots**, every message send can matter
  - -autorelease cost more than a simple -release
  - Use NSFastEnumeration
  - Avoid multiple message sends when one will do
  - ObjC 2.0 dot notation is still a message send…

```
- (void)doSomething
{
   for (i=0; i<BIG_NUMBER; i++) {
      Person* person = [bigArray objectAtIndex:i];
      NSString *name = [person name];
      if ([name length]>1 && [name length]<10) {
         [anotherView setTitle: name];
      }
   }
}
```

Performance Tips – Threading

- Hold locks as short a possible
- Release lock before calling external functions
  - If you must, use a recursive lock
- Consider breaking up work into smaller, main thread chunks
- Too many threads can slow you down
  - Context switch, contention, etc…
- NSOperation is teh sweet…

Performance Tips

- Only ask for what you need
  - Memory
  - Core Location accuracy
  - Accelerometer resolution
  - Use orientation changes instead of accelerometer
- Sample and observe performance, don’t guess or pre-optimize
- Use the tools
  - Integrate static analysis into your builds
  - Always profile performance on the device

Drawing Performance

- Avoid transparency when possible
  - Opaque views are much faster to draw than transparent views
  - Especially important when scrolling
- Don’t call -drawRect: yourself
- Use -setNeedsDisplayInRect: instead of -setNeedsDisplay
- Use CoreAnimation Instrument

Get Notified

- Don’t continuously poll!
  - Unless you must, which is rare
- Hurts both responsiveness and battery life
- Look in the documentation for a notification, delegate callback or other asynchronous API

Take Samples

- CPU sampler / usage Instrument is your friend
- Backtrace taken every fraction of a second
- Higher # samples = better candidates for optimization

Miscellaneous

Some High Level Overviews

Address Book

- Two frameworks
- `AddressBook.framework` **(AB)** – “database” APIs
  - Programmatic access to user’s address book
  - Access or modify contacts, groups and properties
  - C-based API with CoreFoundation types and semantics
- `AddressBookUI.framework` **(ABUI)** – user interface APIs
  - UIViewController based interfaces
  - Standard UI for display, editing, creation and contact picking

AB Framework

- Defines types used by both AB, and ABUI
- Database style API for working with a users address book
  - **Address Book** – A CFTypeRef representing a db connection

```
typedef CFTypeRef ABAddressBookRef;
```

  - **Records** – A CFTypeRef for representing people, and groups

```
typedef CFTypeRef ABRecordRef;
```

  - **Properties** – anything in Contacts app, identified by `ABPropertyID`
    - **Single Value** – a single value, e.g. name
    - **Multi-Value** – multiple values with labels, e.g. telephone numbers

AB's CFTypes

- Like any other CFType – CFRetain, CFRelease
- Toll Free Bridging

```
// ABRecord.h
CFTypeRef ABRecordCopyValue (ABRecordRef record, ABPropertyID property);

// Your Code
- (NSString *)firstName {
   CFStringRef fname = ABRecordCopyValue(record, kABPersonFirstNameProperty)
   return [(NSString *)fname autorelease];
}
```

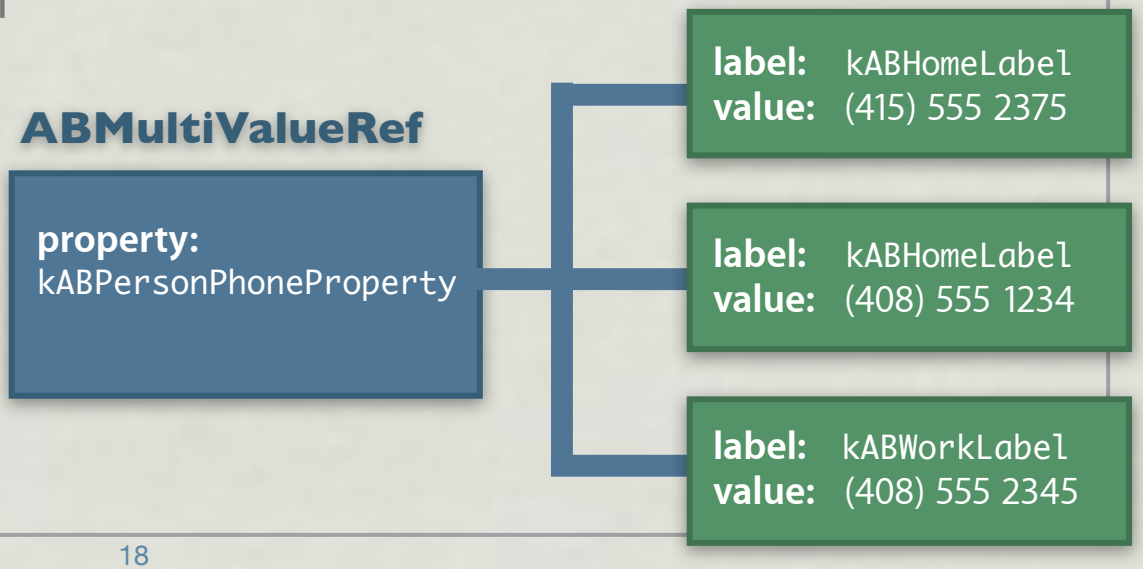
- One CFType for both person and group

```
ABRecordType recordType = ABRecordGetRecordType(record);

if (recordType ==kABPersonType) {
   ...
} else if (recordType == kABGroupType) {
   ...
}
```

AB Values

- Single Values
  - First Name, last name, birthday, etc…
  - Types: CFStringRef, CFDateRef, CFNumberRef, CFDictionaryRef
- Multi–Value
  - Phone numbers, URLs, Emails, etc…
  - Type: ABMultiValueRef



AddressBook UI

- Prompt user to choose a record
  - `ABPeoplePickerNavigationController`
- Display or edit a record
  - `ABPersonViewController`
- Prompt to create a new records
  - `ABNewPersonViewController, or ABUnknownPersonViewController`

UTI

- **Uniform Type Identifier** – a unique string that identifies a class of entity (file format, bundle type, etc…)
  - e.g – ‘public.jpeg’ (or `kUTTypeJPEG`)
- Meant to be a solution to proliferation of “types”
  - MIME type – ‘image/jpeg’
  - File extension based type – e.g. ‘.jpg’, or ‘.jpeg’
  - OS Types (pasteboard, etc…) – e.g. ‘JPEG’
- UTIs declared in a conformance hierarchy
  - e.g. public.html conforms to public.text

UTI

- Benefits
  - Convert to UTI, then just deal with one value!
  - No more guessing if you should use .jpg, or .jpeg...
  - No need to look up what a MIME type is anymore
  - Use conformance to know if your app supports a type, even if you don't know the type
  - iPhone copy / paste uses UTIs to identify content
- Use `MobileCoreServices.framework`
  - `UTCoreTypes.h` declares standard UTIs
  - `UTType.h` declares helper functions

Hardware APIs

- Camera
- Microphone
- Accelerometer
- Magnetometer (Compass)
- GPS (Location)
- External Accessories
- **Device Info**
- **Proximity Sensor**

Determining Availability

- Use “capability” checking APIs

Feature	Option	Sim
Camera	+[UIImagePickerController isSourceTypeAvailable:]	
Microphone	-[AVAudioSession inputIsAvailable]	
External Accessory	-[EAAccessoryManager connectedAccessories]	
Compass	-[CLLocationManager headingAvailable]	
Accelerometer	– Every device has one!	
GPS	– No API exposing specific HW – Specify resolution and check in delegate callbacks	

- Declare requirements – Info.plist `UIRequiredDeviceCapabilities`
  - AppStore can enforce compatibility...

Declaring Requirements

- Add the `UIRequiredDeviceCapabilities` key to Info.plist
- App Store and runtime can enforce requirements
  - Camera – still camera, auto focus capability, video camera
  - Microphone
  - Telephony – phone, sms
  - WiFi
  - Location Services – general location access, GPS
  - Accelerometer, magnetometer
  - OpenGL – ES1, ES2

UIDevice – Device Info

- Identification
  - unique identifier
  - system name, version
  - model (e.g. @”iPhone”, @”iPod Touch”)
- Orientation
  - Portrait, Landscape Left / Right, ...
  - Update notifications
- Battery state and notifications
- Proximity state and notifications

Integration

Working with other applications

Launching Other Apps

- UIApplication provides a simple way to launch another application
- Example of apps you can launch
  - Safari, Maps, Mail
  - Phone, SMS
  - iTunesStore, AppStore
- Great way to quickly integrate with the system
  - “In App” versions (mail, maps) may offer a better user experience

Launching Other Apps

- URL schemes mapped to application handlers
  - App to open based on the URL scheme
  - Pass data to the app with parameters in the URL
  - Web browser links and application code
- Examples
  - HTML Link

```
<a href=”mailto:fred@csmc498i.umd.edu”>Mail Fred</a>
```

  - Application Code

```
UIApplication *app = [UIApplication sharedApplication];
[app openURL:@”mailto:fred@csmc498i.umd.edu”];
```

System URL Schemes

- Mail – **mailto scheme**
  - Can provide subject, message, recipient list, etc…
  - Example

```
mailto:feedback@example.com?subject=Feature%20Request&body
=Please%20Consider%20Adding...
```

- Phone - **tel scheme**

```
tel:1-301-555-5555
```

- SMS - **sms scheme**

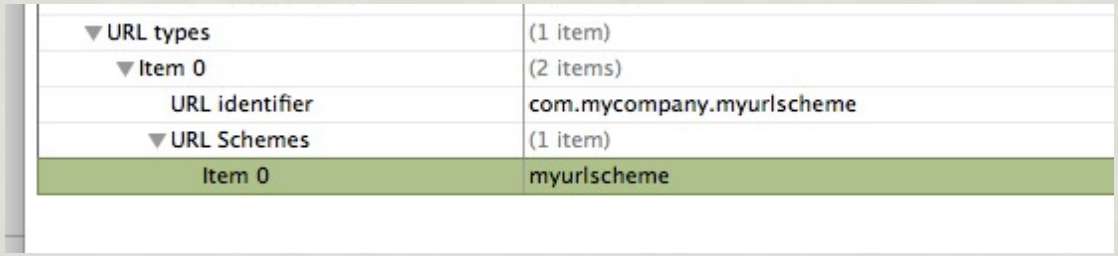
```
sms:1-301-555-5555
```

System URL Schemes

- Some apps opened using http instead of custom scheme
  - Maps
    - Links starting with – http://maps.google.com/maps sent to Maps App
    - List of supported parameters – “Apple URL Scheme Reference.pdf”
  - YouTube
    - Links starting with – http://www.youtube.com sent to YouTube App
    - List of supported parameters – “Apple URL Scheme Reference.pdf”
  - iTunes / App Store
    - Links starting with – http://phobos.apple.com sent to iTunes / App Store
    - Details – http://www.apple.com/itunes/linkmaker/faq

Custom URL Schemes

- You can make your apps launch-able by others!
  - Transfer data from a Lite to a paid version
  - Allow others, even web pages, to open your app and sent it data
- Define a URL identifier, and scheme in your Info.plist



- Implement the URL handler in your app delegate

```
- (BOOL)application:(UIApplication *)application handleOpenURL:(NSURL *)url
{
    // handler code here
}
```

“In App” Purchases

- Communicate with App Store through StoreKit
  - Get Product List
  - Request Payment Collection

“In App” Purchases

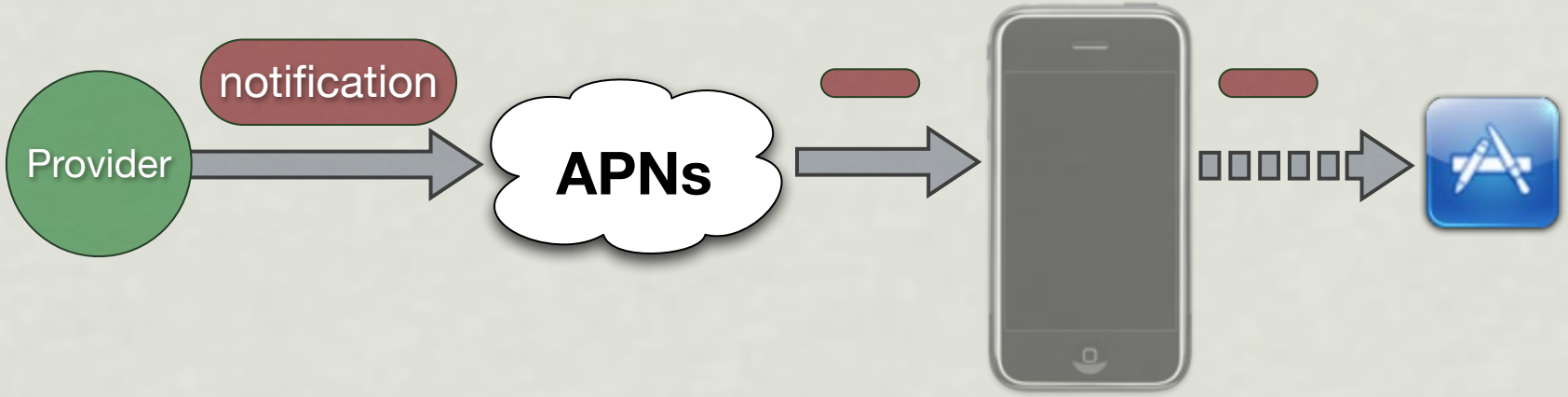
- Use `StoreKit.framework`
- Allows products from within an application
  - Content – e.g. additional game levels
  - Features – e.g. unlock premium features within a basic app
  - Subscriptions or One–time Services
- StoreKit handles financial aspects
  - Secure payments through users iTunes account
  - Provides application information about completed transactions
- Apple provides a “sandbox” for testing!

“In App” Purchases

- Store Kit API is only a small part of the process
- App responsibilities – Additional work required by developer
  - Decide how to track products you want to deliver
  - Provide “store front” UI to the user
  - Provide the product delivery mechanism (download, unlock, …)
  - Register each product with App Store
    - Product ID, Name, description, pricing, and other metadata

Remote Notifications

- Apple Push Notification Service (APNs)



Remote Notifications

- Uses

User Alerts

Badging

Alert Sounds

Push Notifications

- Performance
  - iPhone OS passively listens on your behalf
  - Instead of polling… better for battery life!
- Display alerts even when your app is not running
- iPhone OS handles details for you
  - Receives the notification “payload” JSON
  - Displays alert, plays sounds, badges icon for you
- Obtain a certificates from iPhone Dev Program Portal
- Apple provides a “sandbox” for testing!

Are You Connected?

- A question of network reachability…
- Use functions in `SystemConfiguration.framework`
  - Found in `SCNetworkReachability.h`
  - On Mac OS X, this framework provides more features…
- Status of system's current networking configuration
- Reachability of a target host
  - **"reachable"** – when a data packet, can leave the local host
  - Reachability cannot tell if you can connect to a particular host, only that an interface is available that might allow a connection

Network Reachability Flags

- APIs report information for each target address you check
- Is a connection required?
  - Will network traffic cause automatic connection?
  - Will user intervention be required? – e.g. a password
- Is the target reachable?
  - Is the target actually the local host?
  - Will traffic need to go through a router or not?
  - Is it using a cellular interface? – e.g. GPRS, EDGE, 3G
- Sample code – “Reachability” on Apple’s iPhone dev site

Security

- More than just built-in user security features…
- iPhone OS provides `Security.framework` applications can use to ensure security of their own data
- Features
  - Management of certificates, public/private keys, …
  - Generation of cryptographic secure pseudo random numbers
  - Certificate and key storage in a “key chain”
  - CommonCrypto interface for encryption (HMAC, SHA-1, ...)
- **Keychain** – secure repository for sensitive user data, that can be unlocked with a single password