iOS Mobile Development

Today

- Demo Odds & Ends
  - Cleaning up unused image URLs.
  - A bit more Core Location error checking.
- Camera
  - Actually taking the photo.
  - Finish off Photomania Demo (also includes Action Sheet).
- Core Motion
  - Tracking the device's movement in space.
- Demo
  - Simple game based on Core Motion.
- Application Lifecycle
  - Application Delegate Methods and NSNotifications.

Demo

- Photomania Add Photo (continued)
  - Cleaning up unused image URLs.
  - A bit more Core Location error checking.

UIImagePickerController

- Modal view to get media from camera or photo library
  - Modal means you put it up with `presentViewController:animated:completion:`.
  - On iPad, you might also put it up in a `UIPopoverController`.
- Usage
  1. Create it with `alloc/init` and set `delegate`.
  2. Configure it (source, kind of media, user editability).
  3. Present it.
  4. Respond to delegate method when user is done picking the media.
- What the user can do depends on the platform
  - Some devices have cameras, some do not, some can record video, some can not.
  - Also, you can only offer camera OR photo library on iPad (not both together at the same time).
  - As with all device-dependent API, we want to start by check what's available.
  - `+ (BOOL)isSourceTypeAvailable:(UIImagePickerControllerSourceType)sourceType;`
  - Source type is `UIImagePickerControllerSourceTypePhotoLibrary/Camera/SavedPhotosAlbum`

UIImagePickerController

- But don't forget that not every source type can give video
  - So, you then want to check …
  - `+ (NSArray *)availableMediaTypesForSourceType:(UIImagePickerControllerSourceType)sourceType;`
  - Returns an array of strings you check against constants.
  - Check documentation for all possible, but there are two key ones …
  - `kUTTypeImage` // pretty much all sources provide this
  - `kUTTypeMovie` // audio and video together, only some sources provide this

UIImagePickerController

- But don't forget that not every source type can give video

  So, you then want to check …

  ```
  + (NSArray *)availableMediaTy
  ```

  Returns an array of strings yo

  Check documentation for [illegible] possible, but there are two key ones …

  ```
  kUTTypeImage  // pretty much all sources provide this
  kUTTypeMovie  // audio and video together, only some sources provide this
  ```

  These are declared in the MobileCoreServices framework.
  `#import <MobileCoreServices/MobileCoreServices.h>`
  and add MobileCoreServices to your list of linked frameworks.

- You can get even more specific about front/rear cameras

  (Though usually this is not necessary.)

  ```
  + (BOOL)isCameraDeviceAvailable:(UIImagePickerControllerCameraDevice)cameraDevice;
  ```

  Either `UIImagePickerControllerCameraDeviceFront` or `UIImagePickerControllerCameraDeviceRear`.

  Then check out more about each available camera:

  ```
  + (BOOL)isFlashAvailableForCameraDevice:(UIImagePickerControllerCameraDevice);
  + (NSArray *)availableCaptureModesForCameraDevice:(UIImagePickerControllerCameraDevice);
  ```

  This array contains NSNumber objects with constants `UIImagePic…lerCaptureModePhoto/Video`.

UIImagePickerController

- Set the source and media type you want in the picker

(From here out, UIImagePickerController will be abbreviated UIIPC for space reasons.)

```
UIIPC *picker = [[UIIPC alloc] init];
picker.delegate = self;  // self has to say it implements UINavigationControllerDelegate too
if ([UIIPC isSourceTypeAvailable:UIIPCSourceTypeCamera]) {
    picker.sourceType = UIIPCSourceTypeCamera;
} // else we'll take what we can get (photo library by default)
NSString *desired = (NSString *)kUTTypeMovie;  // e.g., could be kUTTypeImage
if ([[UIIPC availableMediaTypesForSourceType:picker.sourceType] containsObject:desired]) {
    picker.mediaTypes = @[desired];
    // proceed to put the picker up
} else {
    // fail, we can't get the type of media we want from the source we want
}
```

Notice the cast to NSString here.
kUTTypeMovie (and kUTTypeImage) are CFStrings (Core Foundation strings).
Unfortunately, the cast is required to avoid a warning here.

UIImagePickerController

- Editability

```
@property BOOL allowsEditing;
```

If YES, then the user will have opportunity to edit the image/video inside the picker.
When your delegate is notified that the user is done, you'll get both raw and edited versions.

- Limiting Video Capture

```
@property UIIPCQualityType videoQuality;
UIIPCQualityTypeMedium   // default
UIIPCQualityTypeHigh
UIIPCQualityType640x480
UIIPCQualityTypeLow
UIPCQualityTypeIFrame1280x720    // native on some devices
UIPCQualityTypeIFrame960x540     // native on some devices
@property NSTimeInterval videoMaximumDuration;
```

- Other

You can control which camera is used, how flash is used, etc., as well (or user can choose).

UIImagePickerController

- Present the picker

  Note that on iPad, if you are not offering Camera, you must present with popover.
  If you are offering the Camera on iPad, then full-screen is preferred.
  Remember: on iPad, it's Camera OR Photo Library (not both at the same time).

- Delegate will be notified when user is done

```
- (void)imagePickerController:(UIImagePickerController *)picker
didFinishPickingMediaWithInfo:(NSDictionary *)info
{
    // extract image/movie data/metadata here, more on the next slide
    [self dismissViewControllerAnimated:YES completion:...]; // or popover dismissal
}
```

- Also dismiss it when cancel happens

```
- (void)imagePickerControllerDidCancel:(UIImagePickerController *)picker
{
    [self dismissViewControllerAnimated:YES completion:…]; // or popover dismissal
}
```

  If on iPad, you'll want to implement popover's `didDismissPopover…` delegate method too.

UIImagePickerController

- What is in that info dictionary?

```
UIImagePickerControllerMediaType        // kUTTypeImage or kUTTypeMovie
UIImagePickerControllerOriginalImage    // UIImage
UIImagePickerControllerEditedImage      // UIImage
UIImagePickerControllerCropRect         // CGRect (in an NSValue)
UIImagePickerControllerMediaMetadata    // NSDictionary info about the image
UIImagePickerControllerMediaURL         // NSURL edited video
UIImagePickerControllerReferenceURL     // NSURL original (unedited) video
```

- Saving taken images or video into the device's photo library

  Check out `ALAssetsLibrary`.

UIImagePickerController

- Overlay View
  - `@property UIView *cameraOverlayView;`
  - Be sure to set this view's `frame` properly.
  - Camera is always full screen (on iPhone/iPod Touch anyway): `UIScreen`'s `bounds` property.
  - But if you use the built-in controls at the bottom, you might want your view to be smaller.
- Hiding the normal camera controls (at the bottom)
  - `@property BOOL showsCameraControls;`
  - Will leave a blank area at the bottom of the screen (camera's aspect 4:3, not same as screen's).
  - With no controls, you'll need an overlay view with a "take picture" (at least) button.
  - That button should send `- (void)takePicture` to the picker.
  - Don't forget to `dismissModalViewController:` when you are done taking pictures.
- You can zoom or translate the image while capturing
  - `@property CGAffineTransform cameraViewTransform;`
  - For example, you might want to scale the image up to full screen (some of it will get clipped).

Demo

- Photomania Add Photo (continued)
  - Photo taking.
  - Filtering via Action Sheet.

Core Motion

- API to access motion sensing hardware on your device
- Primary inputs: Accelerometer, Gyro, Magnetometer

  Not all devices have all inputs (e.g. only iPhone4-5 and 4th G iPod Touch and iPad 2 have a gyro).
- Class used to get this input is `CMMotionManager`

  Create with `alloc/init`, but use only one instance per application (else performance hit).

  It is a "global resource," so getting one via a class method somewhere is okay.
- Usage

  1. Check to see what hardware is available.
  2. Start the sampling going and poll the motion manager for the latest sample it has.

  ... or ...

  1. Check to see what hardware is available.
  2. Set the rate at which you want data to be reported from the hardware,
  3. Register a block (and a dispatch queue to run it on) each time a sample is taken.

Core Motion

- Checking availability of hardware sensors

  `@property (readonly) BOOL {accelerometer,gyro,magnetometer,deviceMotion}Available;`

  The "device motion" is a combination of all available (accelerometer, magnetometer, gyro).

  We'll talk more about that in a couple of slides.

- Starting the hardware sensors collecting data

  You only need to do this if you are going to poll for data.

  `- (void)start{Accelerometer,Gyro,Magnetometer,DeviceMotion}Updates;`

- Is the hardware currently collecting data?

  `@property (readonly) BOOL {accelerometer,gyro,magnetometer,deviceMotion}Active;`

- Stop the hardware collecting data

  It is a performance hit to be collecting data, so stop during times you don't need the data.

  `- (void)stop{Accelerometer,Gyro,Magnetometer,DeviceMotion}Updates;`

Core Motion

- Checking the data (polling not recommended, more later)

  `@property (readonly) CMAccelerometerData *accelerometerData;`

  CMAccelerometerData object provides `@property (readonly) CMAcceleration acceleration;`

  `typedef struct { double x; double y; double z; } CMAcceleration;` // x, y, z in "g"

  This raw data includes acceleration due to gravity.

  `@property (readonly) CMGyroData *gyroData;`

  CMGyroData object has one `@property (readonly) CMRotationRate rotationRate;`

  `typedef struct { double x; double y; double z; } CMRotationRate;` // x, y, z in rads/sec

  Sign of rotation rate follows right hand rule. This raw data will be biased.

  `@property (readonly) CMMagnetometerData *magnetometerData;`

  CMMagnetometerData object has one `@property (readonly) CMMagneticField magneticField;`

  `typedef struct { double x; double y; double z; } CMMagneticField;` // x, y, z in microteslas

  This raw data will be biased.

  `@property (readonly) CMDeviceMotion *deviceMotion;`

  CMDeviceMotion is an intelligent combination of gyro and acceleration.

  If you have multiple detection hardware, you can report better information about each.

CMDeviceMotion

- Acceleration Data in CMDeviceMotion

```
@property (readonly) CMAcceleration gravity;
@property (readonly) CMAcceleration userAcceleration;  // gravity factored out using gyro
typedef struct { double x; double y; double z; } CMAcceleration; // x, y, z in "g"
```

- Rotation Data in CMDeviceMotion

```
@property CMRotationRate rotationRate;  // bias removed from raw data using accelerometer
typedef struct { double x; double y; double z; } CMRotationRate; // x, y, z in rads/sec

@property CMAttitude *attitude;       // device's attitude (orientation) in 3D space

@interface CMAttitude : NSObject      // roll, pitch and yaw are in radians
@property (readonly) double roll;     // around longitudinal axis passing through top/bottom
@property (readonly) double pitch;    // around lateral axis passing through sides
@property (readonly) double yaw;      // around axis with origin at center of gravity and
                                      // perpendicular to screen directed down
   // other mathematical representations of the device's attitude also available
@end
```

CMDeviceMotion

- Magnetic Field Data in CMDeviceMotion

```
@property (readonly) CMCalibratedMagneticField magneticField;
struct {
    CMMagneticField field;
    CMMagneticFieldCalibrationAccuracy accuracy;
} CMCalibratedMagneticField;
enum {
    CMMagneticFieldCalibrationAccuracyUncalibrated,
                                       Low,
                                       Medium,
                                       High
} CMMagneticFieldCalibrationAccuracy;
```

Core Motion

- Registering a block to receive Accelerometer data

```
- (void)startAccelerometerUpdatesToQueue:(NSOperationQueue *)queue
                             withHandler:(CMAccelerometerHandler)handler;
typedef void (^CMAccelerationHandler)(CMAccelerometerData *data, NSError *error);
```

queue == [[NSOperationQueue alloc] init] or [NSOperation mainQueue (or currentQueue)].

- Registering a block to receive Gyro data

```
- (void)startGyroUpdatesToQueue:(NSOperationQueue *)queue
                    withHandler:(CMGyroHandler)handler;
typedef void (^CMGyroHandler)(CMGyroData *data, NSError *error)
```

- Registering a block to receive Magnetometer data

```
- (void)startMagnetometerUpdatesToQueue:(NSOperationQueue *)queue
                             withHandler:(CMMagnetometerHandler)handler;
typedef void (^CMMagnetometerHandler)(CMMagnetometerData *data, NSError *error)
```

Core Motion

- Registering a block to receive (intelligently) combined data

```
- (void)startDeviceMotionUpdatesToQueue:(NSOperationQueue *)queue
                            withHandler:(CMDeviceMotionHandler)handler;
typedef void (^CMDeviceMotionHandler)(CMDeviceMotion *motion, NSError *error);
Interesting NSError types: CMErrorDeviceRequiresMovement/CMErrorTrueNorthNotAvailable

- (void)startDeviceMotionUpdatesUsingReferenceFrame:(CMAttitudeReferenceFrame)frame
                                            toQueue:(NSOperationQueue *)queue
                                        withHandler:(CMDeviceMotionHandler)handler;
enum {
    CMAttitudeReferenceFrameXArbitraryZVertical,
                            XArbitraryCorrectedZVertical, // needs magnetometer; ++CPU
                            XMagneticZVertical,           // above + device movement
                            XTrueNorthZVertical           // requires GPS + magnetometer
}
@property (nonatomic) BOOL showsDeviceMovementDisplay; // whether to put up UI if required
```

Core Motion

- Setting the rate at which your block gets executed

```
@property NSTimeInterval accelerometerUpdateInterval;
@property NSTimeInterval gyroUpdateInterval;
@property NSTimeInterval magnetometerUpdateInterval;
@property NSTimeInterval deviceMotionUpdateInterval;
```

- It is okay to add multiple handler blocks

  Even though you are only allowed one `CMMotionManager`.
  However, each of the blocks will receive the data at the same rate (as set above).
  (Multiple objects are allowed to poll at the same time as well, of course.)

Demo

- Bouncer

  Using Accelerometer information to drive our user-interface.

Application State

- When your application's UI starts/stops receiving events …

  Your Application Delegate gets …

  ```
  - (void)applicationDidBecomeActive:(UIApplication *)sender;
  - (void)applicationWillResignActive:(UIApplication *)sender;
  ```

  Everyone gets these radio station broadcasts …

  `UIApplicationDidBecomeActiveNotification`

  `UIApplicationWillResignActiveNotification`

  These might happen because user switched to another app or maybe a phone call come in.

  Use these notifications to pause doing stuff in your UI and then restart it later.

Application State

- When you enter the background …
  - You only get a few seconds to respond to this.
  - `- (void)applicationDidEnterBackground:(UIApplication *)sender;`
  - and `UIApplicationDidEnterBackgroundNotification`
  - If you need more time, it is possible (see `beginBackgroundTaskWithExpirationHandler:`).
  - This is a notification for you to clean up any significant resource usage, etc.
- You find out when you get back to the foreground too …
  - Your Application Delegate gets …
  - `- (void)applicationWillEnterForeground:(UIApplication *)sender;`
  - and `UIApplicationWillEnterForegroundNotification`
  - Generally you undo whatever you did in `DidEnterBackground`.
  - You'll get `applicationDidBecomeActive:` soon after receiving the above.

Application State

- Other Application Delegate items of interest …

  Local Notifications (set timers to go off at certain times … will wake your application if needed).
  State Restoration (saving the state of your UI so that you can restore it even if you are killed).
  Data Protection (files can be set to be protected when a user's device's screen is locked).
  Open URL (in Xcode's Info tab of Project Settings, you can register for certain URLs).