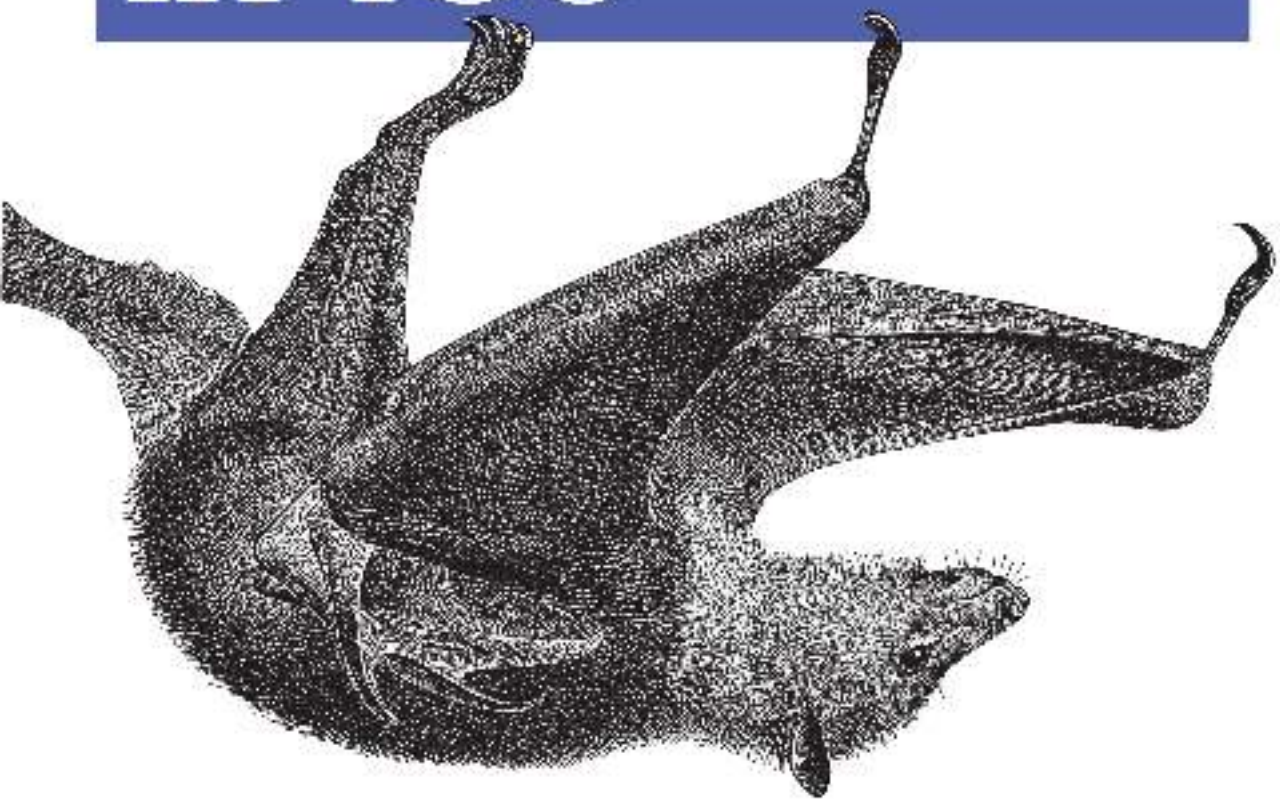
Programming the Accelerometer, Gyroscope, Camera, and Magnetometer

Basic Sensors in iOS



O'REILLY®

Alasdair Allan

Basic Sensors in iOS

What really sets the iPhone apart from laptops and PCs is its use of onboard sensors, including those that are location enabled. This concise book takes experienced iPhone and Mac developers on a detailed tour of iPhone and iPad hardware by explaining how these sensors work and what they're capable of doing.

With this book, you'll build sample applications for each sensor, and learn hands-on how to take advantage of the data each sensor produces. You'll gain valuable experience that you can immediately put to work inside your own iOS applications for the iPhone, iPod touch, and iPad.

Topics Include:

- **Camera:** learn how to take pictures and video, create video thumbnails, customize video, and save media to the photo album
- **Audio:** use the media picker controller and access the iPod music library in your own application, and enable your app to record and play sampled audio
- **Accelerometer:** write an application that uses this sensor to determine device orientation
- **Magnetometer:** learn how this sensor verifies compass headings
- **Core Motion:** use this framework to receive motion data from both the accelerometer and the vibrational gyroscope

US $19.99 CAN $22.99
ISBN: 978-1-449-30846-9

Twitter: @oreillymedia
facebook.com/oreilly

O'REILLY®
oreilly.com

Basic Sensors in iOS

Basic Sensors in iOS

Alasdair Allan

O'REILLY®
Beijing • Cambridge • Farnham • Köln • Sebastopol • Tokyo

Basic Sensors in iOS
by Alasdair Allan

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*http://my.safaribooksonline.com*). For more information, contact our corporate/institutional sales department: (800) 998-9938 or *corporate@oreilly.com*.

Editor: Brian Jepson
Proofreader: O'Reilly Production Services
Cover Designer: Karen Montgomery
Interior Designer: David Futato
Illustrator: Robert Romano

Printing History:
July 2011: First Edition.

ISBN: 978-1-449-30846-9

[LSI]

1311179730

Table of Contents

Preface

Over the last few years the new generation of smart phones, such as Apple's iPhone, has finally started to live up to their name and have become the primary interface device for geographically tagged data. However not only do these devices know where they are, they can tell you how they're being held, they are sufficiently powerful to overlay data layers on the camera view, and record and interpret audio data, and they can do all this in real time. These are not just smart phones, these are computers that just happen to be able to make phone calls.

This book should provide a solid introduction to using the hardware features in the iPhone, iPod touch, and iPad.

Who Should Read This Book?

This book provides an introduction to the hot topic of location-enabled sensors on the iPhone. If you are a programmer who has had some experience with the iPhone before, this book will help you push your knowledge further. If you are an experienced Mac programmer, already familiar with Objective-C as a language, this book will give you an introduction to the hardware specific parts of iPhone programming.

What You Should Already Know?

The book assumes some previous experience with the Objective-C language. Additionally some familiarity with the iPhone platform would be helpful. If you're new to the iPhone platform you may be interested in *Learning iPhone Programming*, also by Alasdair Allan (O'Reilly).

What Will You Learn?

This book will guide you through guide you through developing applications for the iPhone platform that make use of the onboard sensors: the three-axis accelerometer,

the magnetometer (digital compass), the gyroscope, the camera and the global positioning system

What's In This Book?

Chapter 1, *The Hardware*
: This chapter summarizes the available sensors on the iPhone and iPad platforms and how they have, or could be, used in applications. It talks about the differences between the hardware platforms.

Chapter 2, *Using the Camera*
: Walkthrough of how to use the iPhone's camera for still and video. How to create video thumbnails and customise video.

Chapter 3, *Using Audio*
: Walkthrough of how to playback iPod media, as well as how to play and record audio on your device.

Chapter 4, *Using the Accelerometer*
: Walkthrough of how to use the accelerometer, discussion of what is implied with respect to the orientation of the device by the raw readings.

Chapter 5, *Using the Magnetometer*
: Walkthrough of how to use the magnetometer, discussion of combining the magnetometer and accelerometer to get the yaw, pitch and roll of the device.

Chapter 6, *Using Core Motion*
: This paragraph discusses the new Core Motion framework; this new framework allows your application to receive motion data from both the accelerometer and (on the latest generation of devices) the gyroscope.

Chapter 7, *Going Further*
: Provides a collection of pointers to more advanced material on the topics we covered in the book, and material covering some of those topics that we didn't manage to talk about in this book.

Conventions Used in This Book

The following typographical conventions are used in this book:

Italic
: Indicates new terms, URLs, email addresses, filenames, and file extensions.

`Constant width`
: Used for program listings, as well as within paragraphs to refer to program elements such as variable or function names, databases, data types, environment variables, statements, and keywords.

`Constant width bold`
: Shows commands or other text that should be typed literally by the user.

`Constant width italic`
: Shows text that should be replaced with user-supplied values or by values determined by context.

This icon signifies a tip, suggestion, or general note.

This icon signifies a warning or caution.

Using Code Examples

This book is here to help you get your job done. In general, you may use the code in this book in your programs and documentation. You do not need to contact us for permission unless you're reproducing a significant portion of the code. For example, writing a program that uses several chunks of code from this book does not require permission. Selling or distributing a CD-ROM of examples from O'Reilly books does require permission. Answering a question by citing this book and quoting example code does not require permission. Incorporating a significant amount of example code from this book into your product's documentation does require permission.

We appreciate, but do not require, attribution. An attribution usually includes the title, author, publisher, and ISBN. For example: "*Basic Sensors in iOS*, by Alasdair Allan. Copyright 2011 O'Reilly Media, Inc., 978-1-4493-0846-9."

If you feel your use of code examples falls outside fair use or the permission given here, feel free to contact us at *permissions@oreilly.com*.

A lot of the examples won't work completely in the simulator, so you'll need to deploy them to your device to test the code.

Safari® Books Online

Safari Books Online is an on-demand digital library that lets you easily search over 7,500 technology and creative reference books and videos to find the answers you need quickly.

With a subscription, you can read any page and watch any video from our library online. Read books on your cell phone and mobile devices. Access new titles before they are

available for print, and get exclusive access to manuscripts in development and post feedback for the authors. Copy and paste code samples, organize your favorites, download chapters, bookmark key sections, create notes, print out pages, and benefit from tons of other time-saving features.

O'Reilly Media has uploaded this book to the Safari Books Online service. To have full digital access to this book and others on similar topics from O'Reilly and other publishers, sign up for free at *http://my.safaribooksonline.com*.

How to Contact Us

Please address comments and questions concerning this book to the publisher:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
800-998-9938 (in the United States or Canada)
707-829-0515 (international or local)
707-829-0104 (fax)

We have a web page for this book, where we list errata, examples, and any additional information. You can access this page at:

http://www.oreilly.com/catalog/9781449308469

Supplementary materials are also available at:

http://www.programmingiphonesensors.com

To comment or ask technical questions about this book, send email to:

bookquestions@oreilly.com

For more information about our books, courses, conferences, and news, see our website at *http://www.oreilly.com*.

Find us on Facebook: *http://facebook.com/oreilly*

Follow us on Twitter: *http://twitter.com/oreillymedia*

Watch us on YouTube: *http://www.youtube.com/oreillymedia*

Acknowledgments

Everyone has one book in them. This is my second, or depending how you want to look at it, my Platform 9¾, since this book, along with the other three forthcoming short books on iOS and sensor technology, will form the bulk of *Programming iOS4 Sensors*, which would probably be classed by most as my second real book for O'Reilly.

At which point, I'd like to thank my editor at O'Reilly, Brian Jepson, for holding my hand just one more time. As ever his hard work made my hard work much better than it otherwise would have been. I also very much want to thank my wife Gemma Hobson for her continued support and encouragement. Those small, and sometimes larger, sacrifices an author's spouse routinely has to make don't get any less inconvenient the second time around. I'm not sure why she let me write another, perhaps because I claimed to enjoy writing the first one so much. Thank you Gemma. Finally to my son Alex, still too young to read what his daddy has written, hopefully this volume will keep you in books to chew on just a little longer.

CHAPTER 1

The Hardware

The arrival of the iPhone changed the whole direction of software development for mobile platforms, and has had a profound impact on the hardware design of the smart phones that have followed it. The arrival of the iPad has turned what was a single class of device into a platform.

Available Sensor Hardware

While the iPhone is almost unique amongst mobile platforms in guaranteeing that your application will run on all of the current devices (see Figure 1-1), however there is an increasing amount of variation in available hardware between the various models, as shown in Table 1-1.

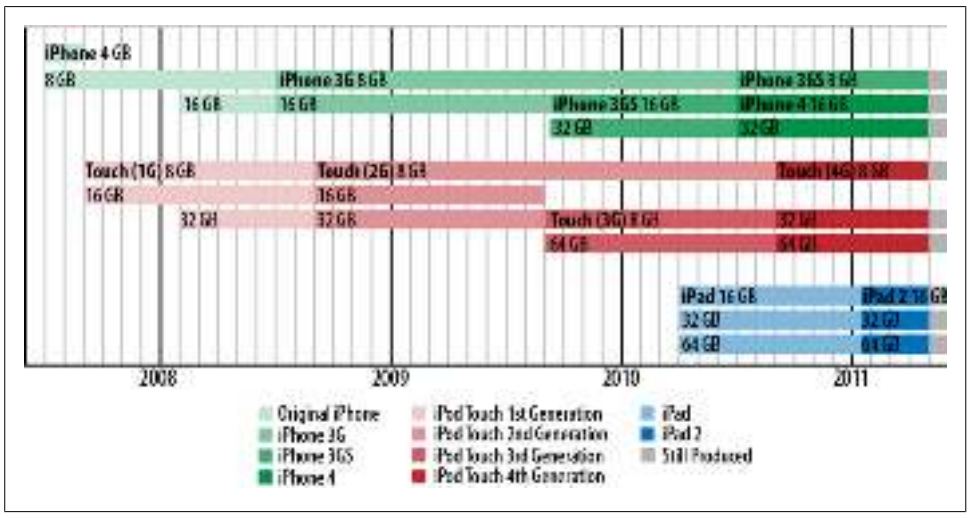


Figure 1-1. Timeline showing the availability of iPhone, iPod Touch, iPad models

Table 1-1. Hardware support in various iPhone, iPod touch, and iPad

	iPhone				iPod touch				iPad		iPad 2	
Hardware Feature	**Original**	**3G**	**3GS**	**4**	**1st Gen**	**2nd Gen**	**3rd Gen**	**4th Gen**	**WiFi**	**3G**	**WiFi**	**3G**
Cellular	☑	☑	☑	☑	☐	☐	☐	☐	☐	☑	☐	☑
WiFi	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑
Bluetooth	☑	☑	☑	☑	☐	☑	☑	☑	☑	☑	☑	☑
Speaker	☑	☑	☑	☑	☐	☑	☑	☑	☑	☑	☑	☑
Audio In	☑	☑	☑	☑	☐	☑	☑	☑	☑	☑	☑	☑
Accelerometer	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑
Magnetometer	☐	☐	☑	☑	☐	☐	☐	☐	☑	☑	☑	☑
Gyroscope	☐	☐	☐	☑	☐	☐	☐	☑	☐	☐	☑	☑
GPS	☐	☑	☑	☑	☐	☐	☐	☐	☐	☑	☐	☑
Proximity Sensor	☑	☑	☑	☑	☐	☐	☐	☐	☐	☐	☐	☐
Camera	☑	☑	☑	☑	☐	☐	☐	☑	☐	☐	☑	☑
Video	☐	☐	☑	☑	☐	☐	☐	☑	☐	☐	☑	☑
Vibration	☑	☑	☑	☑	☐	☐	☐	☐	☐	☐	☐	☐

Most of the examples in this book will be built as iPhone however depending on the availability of hardware the examples will run equally well on the iPod touch and iPad; the underlying code is equally applicable as we're dealing for the most part directly with that hardware.

Differences Between iPhone and iPad

The most striking, and obvious, difference between the iPhone and the iPad is screen size. The original iPhone screen has 480×320 pixel resolution at 163 pixels per inch. The iPhone 4 and 4th generation iPod touch Retina Displays have a resolution of 960×640 pixel at 326 pixels per inch. Meanwhile both generations of the iPad screen have 1024×768 pixel resolution at 132 pixels per inch. This difference will be the single most fundamental thing to affect the way you design your user interface on the two platforms. Attempting to treat the iPad as simply a rather oversized iPod touch or iPhone will lead to badly designed applications. The metaphors you use on the two different platforms

The increased screen size of the device means that you can develop desktop-sized applications, not just phone-sized applications, for the iPad platform. Although in doing so, a rethink of the user interface to adapt to multi-touch is needed. What works for the iPhone or the desktop, won't automatically work on an iPad. For example, Apple

totally redesigned the user interface of the iWork suite when they moved it to the iPad. If you're intending to port a Mac OS X desktop application to the iPad you should do something similar.

Interestingly there is now an option for iOS developers to port their iPhone and iPad projects directly to Mac OS X. The Chameleon Project *http://chameleonproject.org* is a drop in replacement for `UIKit` that runs on Mac OS X, allowing iOS applications to be run on the desktop with little modification, in some cases none.

Due to its size and function the iPad is immediately associated in our minds with other more familiar objects like a legal pad or a book. Holding the device triggers powerful associations with these items, and we're mentally willing to accept the iPad has a successor to these objects. This is simply not true for the iPhone; the device is physically too small.

The Human Interface Guidelines

Apple has become almost infamous for strict adherence to its Human Interface Guidelines. Designed to present users with "a consistent visual and behavioral experience across applications and the operating system" the interface guidelines mean that (most) applications running on the Mac OS X desktop have a consistent look and feel. With the arrival of the iPhone and the iPad, Apple had to draw up new sets of guidelines for their mobile platforms, radically different from the traditional desktop.

Even for developers who are skeptical about whether they really needed to strictly adhere to the guidelines (especially when Apple periodically steps outside them) the Human Interface Guidelines have remained a benchmark against which the user experience can be measured.

Copies of the Human Interface Guidelines for both the iPhone and the iPad are available for download from the App Store Resource Center.

I would recommend that you read the mobile Human Interface Guidelines carefully, if only because violating them could lead to your application being rejected by the review team during the App Store approval process.

However this book is not about how to design your user interface or manage your user experience. For the most part the examples I present in this book are simple view-based applications that could be equally written for the iPhone and iPod touch or the iPad. The user interface is only there to illustrate how to use the underlying hardware. This book is about how to use the collection of sensors in these mobile devices.

Device Orientation and the iPad

The slider button on the side of the iPad can, optionally, be used to lock the device's orientation. This means that if you want the screen to stay in portrait mode, it won't move when you turn it sideways if locked. However despite the presence of the rotation lock (and unlike the iPhone where many applications only supported Portrait mode) an iPad application is expected to support all orientations equally.

Apple has this to say about iPad applications: "An application's interface should support all landscape and portrait orientations. This behavior differs slightly from the iPhone, where running in both portrait and landscape modes is not required."

To implement basic support for all interface orientations, you should implement the `shouldAutorotateToInterfaceOrientation:` method in all of your application's view controllers, returning `YES` for all orientations. Additionally, you should configure the auto-resizing mark property of your views inside Interface Builder so that they correctly respond to layout changes (i.e. rotation of the device).

Going beyond basic support

If you want to go beyond basic support for alternative orientations there is more work involved. Firstly for custom views, where the placement of subviews is critical to the UI and need to be precisely located, you should override the `layoutSubviews` method to add your custom layout code. However, you should override this method only if the autoresizing behaviors of the subviews are not what you desire.

When an orientation event occurs, the `UIWindow` class will work with the front-most `UIViewController` to adjust the current view. Therefore if you need to perform tasks before, during, or after completing device rotation you should use the relevant rotation `UIViewController` notification methods. Specifically the view controller's `willRotateToInterfaceOrientation:duration:`, `willAnimateRotationToInterfaceOrientation:duration:`, and `didRotateFromInterfaceOrientation:` methods are called at relevant points during rotation allowing you to perform tasks relevant to the orientation change in progress. For instance you might make use of these callbacks to allow you to add or remove specific views and reload your data in those views.

Detecting Hardware Differences

Because your application will likely support multiple devices, you'll need to write code to check which features are supported and adjust your application's behavior as appropriate.

Camera Availability

We cover the camera in detail in Chapter 2, however it is simple matter to determine whether a camera is present in the device:

```
BOOL available = [UIImagePickerController
  isSourceTypeAvailable:UIImagePickerControllerSourceTypeCamera];
```

Once you have determined that a camera is present you can enquire whether it supports video by making a call to determine the available media types the camera supports:

```
NSArray *media = [UIImagePickerController availableMediaTypesForSourceType:
                  UIImagePickerControllerSourceTypeCamera];
```

If the `kUTTypeMovie` media type is returned as part of the array, then the camera will support video recording:

```
if ( [media containsObject:(NSString *)kUTTypeMovie ] ){
    NSLog(@"Camera supports movie capture.");
}
```

Audio Input Availability

An initial poll of whether audio input is available can be done using the `AVAudioSession` class by checking the `inputIsAvailable` class property:

```
AVAudioSession *audioSession = [AVAudioSession sharedInstance];
BOOL audioAvailable = audioSession.inputIsAvailable;
```

You will need to add the *AVFoundation.Framework* (right-click/Control-click on the Frameworks folder in Xcode, then choose Add→Existing Frameworks). You'll also need to import the header (put this in your declaration if you plan to implement the `AVAudioSessionDelegate` protocol discussed later):

```
#import <AVFoundation/AVFoundation.h>
```

You can also be notified of any changes in the availability of audio input, e.g., if a second generation iPod touch user has plugged in headphones with microphone capabilities. First, nominate your class as a delegate:

```
audioSession.delegate = self;
```

And then declare it as implementing the `AVAudioSessionDelegate` protocol in the declaration:

```
@interface YourAppDelegate : NSObject <UIApplicationDelegate,
  AVAudioSessionDelegate >
```

Then implement the `inputIsAvailableChanged:` in the implementation:

```
- (void)inputIsAvailableChanged:(BOOL)audioAvailable {
      NSLog(@"Audio availability has changed");
}
```

GPS Availability

The short answer to a commonly asked question is that the Core Location framework does not provide any way to get direct information about the availability of specific hardware such as the GPS at application run time, although you can check whether location services are enabled:

```
BOOL locationAvailable = [CLLocationManager locationServicesEnabled];
```

However, you can require the presence of GPS hardware for your application to load (see "Setting Required Hardware Capabilities").

Magnetometer Availability

Fortunately Core Location does allow you to check for the presence of the magnetometer (digital compass) fairly simply:

```
BOOL magnetometerAvailable = [[CLLocationManager headingAvailable];
```

Setting Required Hardware Capabilities

If your application requires specific hardware features in order to run you can add a list of required capabilities to your application's *Info.plist* file. Your application will not start unless those capabilities are present on the device.

To do this, open the project and click on the application's *Info.plist* file to open it in the Xcode editor. Click on the bottommost entry in the list. A plus button will appear to the right-hand side of the key-value pair table.

Click on this button to add a new row to the table, and scroll down the list of possible options and select "Required device capabilities" (the `UIRequiredDeviceCapabilities` key). This will add an (empty) array to the *plist* file.

The allowed values for the keys are:

- `telephony`
- `wifi`
- `sms`
- `still-camera`
- `auto-focus-camera`
- `front-facing-camera`
- `camera-flash`

- `video-camera`
- `accelerometer`
- `gyroscope`
- `location-services`
- `gps`
- `magnetometer`
- `gamekit`
- `microphone`
- `opengles-1`
- `opengles-2`
- `armv6`
- `armv7`
- `peer-peer`

A full description of the possible keys is given in the Device Support section of the iPhone Application Programming Guide available from the iPhone Development Center.

Persistent WiFi

If your application requires a persistent WiFi connection you can set the Boolean `UIRequiresPersistentWiFi` key in the Application's *Info.plist* file to ensure that WiFi is available. If set to `YES` the operating system will open a WiFi connection when your application is launched and keep it open while the application is running. If this key is not present, or is set to `NO`, the Operating System will close the active WiFi connection after 30 minutes.

Background Modes

Setting the `UIBackgroundModes` key in the Application's *Info.plist* file notifies the operating systems that the application should continue to run in the background, after the user closes it, since it provides specific background services.

Apple has this to say about background modes, "These keys should be used sparingly and only by applications providing the indicated services. Where alternatives for running in the background exist, those alternatives should be used instead. For example, applications can use the significant location change interface to receive location events instead of registering as a background location application."

There are three possible key values: `audio`, `location`, and `voip`. The `audio` key indicates that after closing the application will continue to play audible content. The `location` key indicates that the application provides location-based information for the user using the standard Core Location services, rather than the newer significant location change service. Finally, the `voip` key indicates that the application provides Voice-over-IP services. Applications marked with this key are automatically launched after system boot so that the application can attempt to re-establish VoIP services.

CHAPTER 2

Using the Camera

Phones with cameras only started appearing on the market in late 2001; now they're everywhere. By the end of 2003 more camera phones were sold worldwide than stand-alone digital cameras, and by 2006 half of the world's mobile phones had a built-in camera.

The social impact of this phenomenon should not be underestimated; the ubiquity of these devices has had a profound affect on society and on the way that news and information propagate. Mobile phones are constantly carried, which means their camera is always available. This constant availability has led to some innovative third party applications, especially with the new generation of smart phones. The iPhone has been designed with always-on connectivity in mind.

The Hardware

Until recently, only the iPhone has featured a camera in all of the available models. However the latest generation of both the iPod touch and iPad now also have cameras.

The original iPhone and iPhone 3G feature a fixed-focus 2.0-megapixel camera, while the iPhone 3GS features a 3.2-megapixel camera with auto-focus, auto-white balance and auto-macro focus (up to 10cm). The iPhone 3GS camera is also able of capturing 640×480 pixel video at 30 frames per second. Although the earlier models are physically capable of capturing video, they are limited in software and this feature is not available at the user level. The latest iPhone 4 features a 5-megapixel camera with better low-light sensitivity and backside illuminated sensor. The camera has an LED flash and is capable of capturing 720p HD video at 30 frames per second. The iPhone 4 also has a lower-resolution front-facing camera, which is capable of capturing 360p HD video at 30 frames per second.

The iPhone 3GS and iPhone 4 cameras are known to suffer from *rolling shutter effect* when used to take video. This effect is a form of aliasing that may result in distortion of fast moving objects, or image effects due to lighting levels that change as a frame is captured. At the time of writing it's not clear whether the 4th generation iPod touch and iPad 2 cameras suffer the same problem.

The latest generation of iPod touch and iPad also have both rear- and front-facing cameras, both of which are far lower resolution than the camera fitted to the iPhone 4, see Table 2-1 for details. You'll notice the difference in sizes between still and video images on the iPod touch and the iPad 2. It's unclear whether Apple is using a 1280×720 sensor and cropping off the left and right sides of the video image for still images, or whether it is using a 960×720 sensor and up-scaling it on the sides for video. The later would be an unusual approach for Apple, but is not inconceivable.

Table 2-1. Camera hardware support in various iPhone models

Model	Focus	Flash	Megapixels	Size	Video
Original iPhone	Fixed	No	2.0	1600×1200	No
iPhone 3G	Fixed	No	2.0	1600×1200	No
iPhone 3GS	Autofocus	No	3.2	2048×1536	VGA at 30fps
iPhone 4	Autofocus	LED flash	5.0 for still	2592×1944	720p at 30fps
			1.4 for video	1280×1024	
	Fixed	No	1.4	1280×1024	360p at 30fps
iPod touch (4th Gen)	Fixed	No	0.69 for still	960×720	720p at 30fps
			0.92 for video	1280×720	
	Fixed	No	1.4	1280×1024	VGA at 30fps
iPad 2	Fixed	No	0.69 for still	960×720	720p at 30fps
			0.92 for video	1280×720	
	Fixed	No	1.4	1280×1024	VGA at 30fps

All models produce geocoded images by default.

Capturing Stills and Video

The `UIImagePickerViewController` is an Apple-supplied interface for choosing images and movies, and taking new images or movies (on supported devices). This class handles all of the required interaction with the user and is very simple to use. All you need to do is tell it to start, then dismiss it after the user selects an image or movie.