iPhone Human Interface Guidelines

User Experience

2008-02-29

Apple Inc.
© 2008 Apple Inc.
All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, mechanical, electronic, photocopying, recording, or otherwise, without prior written permission of Apple Inc., with the following exceptions: Any person is hereby authorized to store documentation on a single computer for personal use only and to print copies of documentation for personal use provided that the documentation contains Apple's copyright notice.

The Apple logo is a trademark of Apple Inc.

Use of the "keyboard" Apple logo (Option-Shift-K) for commercial purposes without the prior written consent of Apple may constitute trademark infringement and unfair competition in violation of federal and state laws.

No licenses, express or implied, are granted with respect to any of the technology described in this document. Apple retains all intellectual property rights associated with the technology described in this document. This document is intended to assist application developers to develop applications only for Apple-labeled computers.

Every effort has been made to ensure that the information in this document is accurate. Apple is not responsible for typographical errors.

Apple Inc.
1 Infinite Loop
Cupertino, CA 95014
408-996-1010

Apple, the Apple logo, iPhoto, iPod, Mac, and Mac OS are trademarks of Apple Inc., registered in the United States and other countries.

iPhone, Multi-Touch, and Safari are trademarks of Apple Inc.

Java and all Java-based trademarks are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

Simultaneously published in the United States and Canada.

Even though Apple has reviewed this document, APPLE MAKES NO WARRANTY OR REPRESENTATION, EITHER EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT, ITS QUALITY, ACCURACY, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. AS A RESULT, THIS DOCUMENT IS PROVIDED "AS IS," AND YOU, THE READER, ARE ASSUMING THE ENTIRE RISK AS TO ITS QUALITY AND ACCURACY.

IN NO EVENT WILL APPLE BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY DEFECT OR INACCURACY IN THIS DOCUMENT, even if advised of the possibility of such damages.

THE WARRANTY AND REMEDIES SET FORTH ABOVE ARE EXCLUSIVE AND IN LIEU OF ALL OTHERS, ORAL OR WRITTEN, EXPRESS OR IMPLIED. No Apple dealer, agent, or employee is authorized to make any modification, extension, or addition to this warranty.

Some states do not allow the exclusion or limitation of implied warranties or liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you. This warranty gives you specific legal rights, and you may also have other rights which vary from state to state.

Contents

Figures and Tables

2008-02-29 | © 2008 Apple Inc. All Rights Reserved.

2008-02-29 | © 2008 Apple Inc. All Rights Reserved.

Introduction

Important: This is a preliminary document for an API or technology in development. Although this document has been reviewed for technical accuracy, it is not final. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent documentation seeds.

iPhone and iPod touch are sophisticated devices that combine the revolutionary Multi-Touch interface with powerful features, such as email and instant-messaging capability, a full-featured web browser, iPod, and, in iPhone, a mobile phone. iPhone OS is the system software that runs on iPhone and iPod touch. With the advent of the iPhone SDK, these powerful features are extended to include significant developer opportunities. In addition to creating web content for use on iPhone OS–based devices, developers can use the iPhone SDK to create native applications people can store and use on their devices.

Read this document to learn about the range of application types you can develop for iPhone OS and the human interface design principles that inform all great applications. In this document you also learn how to follow those principles as you design a superlative user interface and user experience for your application. Whether you're an experienced computer application developer, an experienced mobile-device application developer, or a newcomer to the field, the guidelines in this document will help you produce an iPhone application users want.

Note: This document briefly summarizes web-based development for iPhone OS–based devices. For more in-depth information specific to designing web content for these devices, see *iPhone Human Interface Guidelines for Web Applications*.

Organization of This Document

iPhone Human Interface Guidelines is divided into two parts, each of which contains several chapters:

- The first part, "Planning Your iPhone Software Product" (page 13) describes the iPhone OS environment and the types of software you can develop for it. It also covers the fundamental human interface design principles that inform the user interfaces of all great software and it describes how to apply these principles to the design of your iPhone application.
- The second part, "Working with iPhone OS User Interface Elements" (page 45), delves into the components you use to create the user interface of your iPhone application. It describes the various view and controls that are available to you and provides guidance on how to use them effectively.

See Also

To learn how to code your iPhone application, read:

- *iPhone OS Programming Guide*

Planning Your iPhone Software Product

This part of *iPhone Human Interface Guidelines* describes ways to think about designing and developing software for iPhone OS and presents the fundamental principles that underlie great software design. Read Part I to learn about the different types of software you can develop for iPhone OS and the design principles you can use to inform your work. You'll also learn how to apply those principles to specific aspects and tasks in your application, so you can create a superlative product that provides an intuitive and compelling user interface.

P A R T   I

Planning Your iPhone Software Product

The iPhone OS Platform: Rich with Possibilities

iPhone OS supports numerous types of software, ranging from webpages that users view in Safari on iPhone to iPhone applications that run natively on iPhone OS–based devices. This chapter outlines on the different types of software solutions you can create for iPhone OS–based devices.

If you're new to the platform, be sure to begin with the summary of differences between iPhone OS–based devices and computers given in "Overview of Platform Differences" (page 15). Although the information in that section is not comprehensive, it touches on the issues you need to be aware of as you design an iPhone application.

To help you plan an iPhone application, this chapter describes ways to think about different application styles and the characteristics that define them. This chapter also describes how some of the bundled Mac OS X applications were transformed into versions appropriate for iPhone OS. If you have an existing computer application you'd like to refashion for iPhone OS, understanding this process is key.

Overview of Platform Differences

An iPhone OS–based device is not a desktop or laptop computer, and an iPhone application is not the same as a desktop application. Although these seem merely common-sense statements, it is nonetheless paramount to keep them in mind as you embark on developing software for these devices.

Note: If you are designing web-only content for an iPhone OS–based device, you are probably already familiar with the platform differences that affect you. If not, you can learn more about them in *iPhone Human Interface Guidelines for Web Applications*. This section focuses on how platform differences affect the development of iPhone applications, which run natively on iPhone OS–based devices.

Designing software for iPhone OS–based devices requires a state of mind that may or may not be second nature to you. In particular, if the bulk of your experience lies in developing desktop applications, you should be aware of the significant differences between designing software for a mobile platform and for a computer.

This section summarizes the concrete differences that have the highest potential impact on your design decisions. For detailed information on how to handle these and other issues in your iPhone application development process, see *iPhone OS Programming Guide*.

2008-02-29 | © 2008 Apple Inc. All Rights Reserved.

Compact Screen Size

The small, high-resolution screens of iPhone OS–based devices make them powerful display devices that fit into users' pockets. But that very advantage to users may be challenging to you, the developer, because it means that you must design a user interface that may be very different from those you're accustomed to designing.

Keep in mind the screen size of 480 x 320 points and use that as a motivation to focus the user interface on the essentials. You don't have the room to include design elements that aren't absolutely necessary, and crowding user interface elements makes your application unattractive and difficult to use.

Memory is Not Unlimited

Memory is a critical resource in iPhone OS, so managing memory in your application is crucial. Because the iPhone OS virtual memory model does not include disk swap space, you must take care to avoid allocating more memory than is available on the device. When low-memory conditions occur, iPhone OS warns the running application and may terminate the application if the problem persists. Be sure your application is responsive to memory usage warnings and cleans up memory in a timely manner.

As you design your application, strive to reduce the application's memory footprint by, for example, eliminating memory leaks, making resource files as small as possible, and loading resources lazily. See *iPhone OS Programming Guide* for extensive information about how to design iPhone applications that handle memory appropriately.

One Window at a Time

One of the biggest differences between the iPhone OS environment and the computer environment is the window paradigm. With the exceptions of some modal windows, users see a single application window at a time on an iPhone OS–based device screen. iPhone applications can contain as many different views as necessary, but users access and see them sequentially, never simultaneously.

If the desktop version of your application requires users to see several windows simultaneously, you need to decide if there's a different way users can accomplish the same task in a single window. If not, you should focus your iPhone application on a single subtask of your computer application, instead of trying to replicate a wider feature set.

One Application at a Time

Only one iPhone application can run at a time, and third-party applications never run in the background. This means that when users switch to another application, answer the phone, or check their email, the application they were using quits. It's important to make sure that users do not experience any negative effects because of this reality. In other words, users should not feel that leaving your iPhone application and returning to it later is any more difficult than switching among applications on a computer.

The most effective thing you can do to ensure that users have a positive application-switching experience is to pare the launch time of your application to the minimum. See *iPhone OS Programming Guide* for guidance on how to make your application's launch time as short as possible.

In general, users quit your application by switching to another application or device service; they take no specific action to close your application. Therefore, do not expect users to select Quit from a menu or click a close button. This means that your application is likely to quit without much warning, so you should be prepared to save user changes as they are made, as quickly as possible. Doing so allows a fast, smooth transition between applications and ensures that your application can reflect the user's most recent changes the next time it starts.

Another important facet of the single application model is the way you handle application-specific preferences. On iPhone OS–based devices, users set preferences in the Settings application. Your iPhone application can supply settings that users can access, but this means that they must quit your application when they want to access them in Settings. If you follow the standard guidelines and offer settings that users need to set once, and then rarely, if ever, again, the user experience of your application should be smooth.

Minimal User Help

Mobile users don't have the time to read through a lot of help content before they can use your application. What's more, you don't want to give up valuable space to display or store it. A hallmark of the design of iPhone OS–based devices is ease of use, so it's crucial that you meet users' expectations and make the use of your application immediately obvious. There are a few things you can do to achieve this:

- Use standard controls correctly. Users are familiar with the standard controls they see in the built-in applications, so they already know how to use them in your application.
- Be sure the path through the information you present is logical and easy for users to predict. In addition, be sure to provide markers, such as back buttons, that users can use to find out where they are and how to retrace their steps.

What Are Your Options?

Before you decide how to present your product to iPhone OS users, you need to understand the range of options you have. Depending on the implementation details of your proposed product and its intended audience, some types of software may be better suited to your needs than others.

This section divides software for iPhone OS–based devices into three broad categories, primarily based on implementation method. Roughly speaking, you can create:

- An **iPhone application**, which is an application you develop using the iPhone SDK to run natively on iPhone OS–based devices.
- **Web-only content**, including web applications, which are websites that behave like built-in iPhone applications.
- A **hybrid application**, which is an iPhone application that provides access to web content primarily through a web-content viewing area, but includes some iPhone OS user interface elements.

iPhone Applications

iPhone applications resemble the built-in applications on iPhone OS–based devices in that they reside on the device itself and take advantage of features of the iPhone OS environment. Users install iPhone applications on their devices and use them just as they use built-in applications, such as Stocks, Maps, Calculator, and Mail.

An iPhone application is quick to launch and easy to use. Whether the application enables a task like sending email or provides entertainment to users, it is characterized by responsiveness, simplicity, and a beautiful, streamlined user interface.

Web-only Content

You have a few different options when it comes to providing **web-only content** to iPhone OS users:

- **Web applications**

  Webpages that provide a focused solution to a task and conform to certain display guidelines are known as web applications, because they behave similarly to the built-in iPhone OS applications. A web application, like all web-only content, runs in Safari on iPhone; users do not install it on their devices, instead they go to the web application's URL.

- **Optimized webpages**

  Webpages that are optimized for Safari on iPhone display and operate as designed (with the exception of any elements that rely on unsupported technologies, such as plug-ins, Flash, and Java). In addition, an optimized webpage correctly scales content for the device screen and is often designed to detect when it is being viewed on iPhone OS–based devices, so that it can adjust the content it provides accordingly.

- **Compatible webpages**

  Webpages that are compatible with Safari on iPhone display and operate as designed (with the exception of any elements that rely on unsupported technologies, such as plug-ins, Flash, and Java). A compatible webpage does not tend to take extra steps to optimize the viewing experience on iPhone OS–based devices, but the device usually displays the page successfully.

If you have an existing website or web application, first ensure that it works well on iPhone OS–based devices. Also, you should consider creating a custom icon users can put on their Home screens using the Web Clip feature. In effect, this allows users to keep on their Home Screens a bookmark to your website that looks like a native application icon. To learn more about creating a custom icon and how to make web content look great on iPhone OS–based devices, see *iPhone Human Interface Guidelines for Web Applications*.

Hybrid Applications

With iPhone OS, you can create an application that combines features of native applications and webpages. A **hybrid application** is a native iPhone application that provides most of its structure and functionality through a web viewing area, but also tends to contain standard iPhone OS user interface elements.

A hybrid application gives users access to web content with an element called a web view (described in "Web Views" (page 77)). Precisely how you use a web view in your application is up to you, but it's important to avoid giving users the impression that your application is merely a mini web browser. A hybrid application should behave and appear like a native iPhone application; it should not draw attention to the fact that it depends upon web sources.

Three General Application Styles

This document identifies three application styles, based on visual and behavioral characteristics, data model, and user experience. Before you read further, it's important to emphasize that these varieties are named and described to help you clarify some of your design decisions, not to imply that there is a rigid classification scheme that all iPhone software must follow. Instead, these styles are described to help you see how different approaches can be suitable for different types of information and functionality.

Note: Bear in mind that application style does not dictate a particular type of software. This document focuses on designing iPhone applications, but the application styles explored here can be implemented in any type of software.

As you read about these three application styles, think about how the characteristics of each might enhance your proposed feature set and the overall user experience you plan to deliver in your iPhone application. To help you discover the combination of characteristics that best suit your application, keep the following questions in mind as you learn about different design styles for iPhone applications:

- What do you expect to be the user's motivation for using the application?
- What do you intend to be the user's experience while using the application?
- What is the goal or focus of your application?
- How does your application organize and display the information people care about? Is there a natural organization associated with the main task of the application?

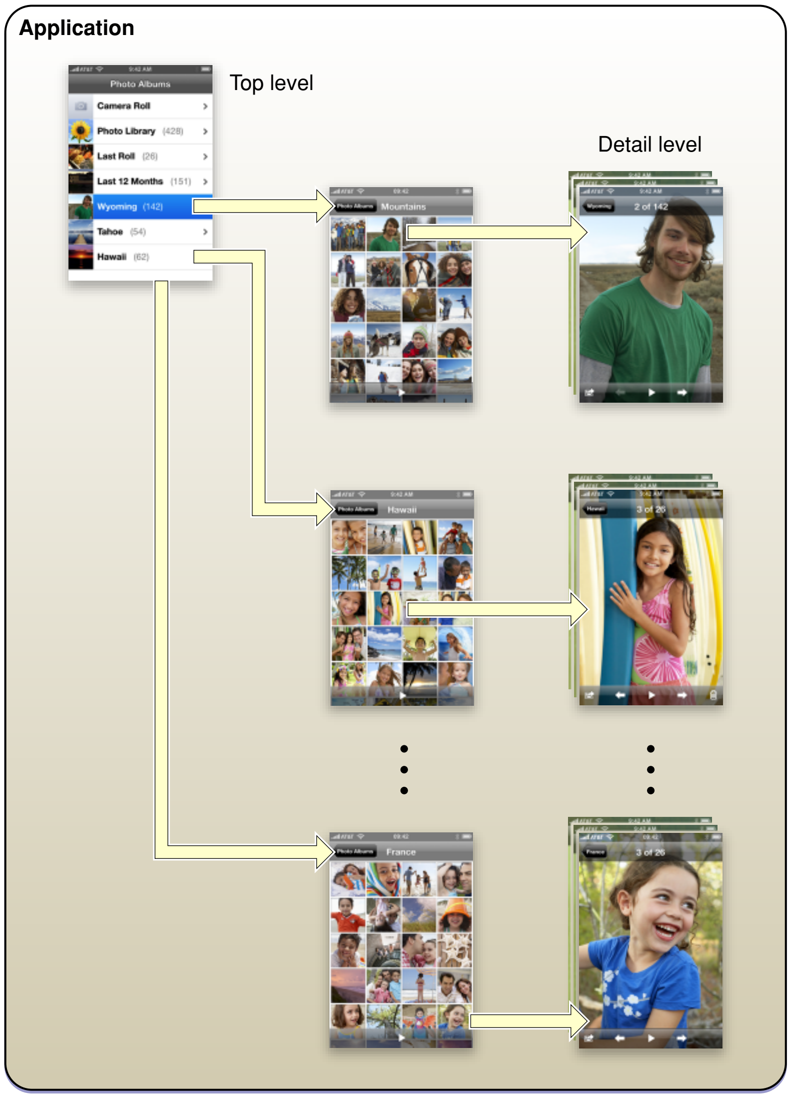
Productivity Applications

A **productivity application** enables tasks that are based on the organization and manipulation of detailed information. People use productivity applications to accomplish important tasks. Mail is a good example of an iPhone productivity application.

Seriousness of purpose does not mean that productivity applications should attempt to appear serious by providing a dry, uninspiring user experience, but it does mean that users appreciate a streamlined approach. To this end, successful productivity applications keep the user experience focused on the task, so people can quickly find what they need, easily perform the necessary actions, complete the task, and move on to something else.

Productivity applications often organize user data hierarchically. In this way, people can find information by making progressively more specific choices until they arrive at the desired level of detail. iPhone OS provides table elements that make this process extremely efficient on iPhone OS devices (see "Table Views" (page 67) for more information about these user interface elements). Figure 1-1 shows an example of this type of data organization.

Figure 1-1 Productivity applications tend to organize information hierarchically



Typically, the user interaction model in a productivity application consists of:

- Organizing the list
- Adding to and subtracting from the list
- Drilling down through successive levels of detail until the desired level is reached, then performing tasks with the information on that level

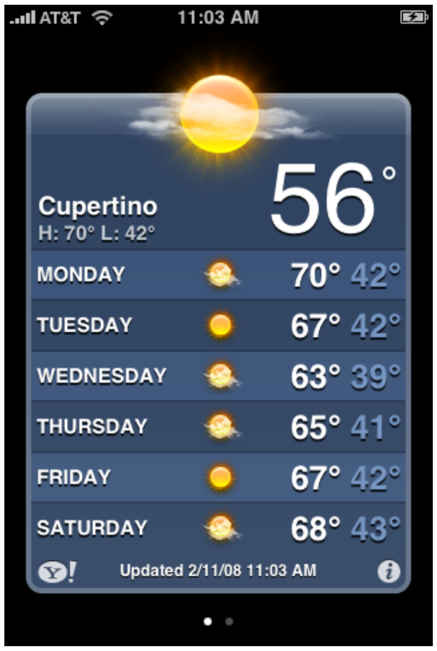
Productivity applications tend to use multiple views, usually displaying one level of the hierarchy per view. The user interface tends to be simple, uncluttered, and composed of standard views and controls. Productivity applications do not tend to customize the interface much, because the focus is on the information and the task, and not as much on the environment or the experience.

Among all types of iPhone applications, a productivity application is the most likely to supply preferences, or settings, the user can specify in the Settings application. This is because productivity applications work with lots of information and, potentially, many ways to access and manage it. It's important to emphasize, however, that the user should seldom need to change these settings, so the settings should not target simple configuration changes that could be handled in the main user interface.

Utility Applications

A **utility application** performs a simple task that requires a minimum of user input. People open a utility application to see a quick summary of information or to perform a simple task on a limited number of objects. The Weather application (shown in Figure 1-2) is a good example of a utility application because it displays a narrowly focused amount of information in an easy-to-scan summary.

Figure 1-2 Weather is an example of a utility application

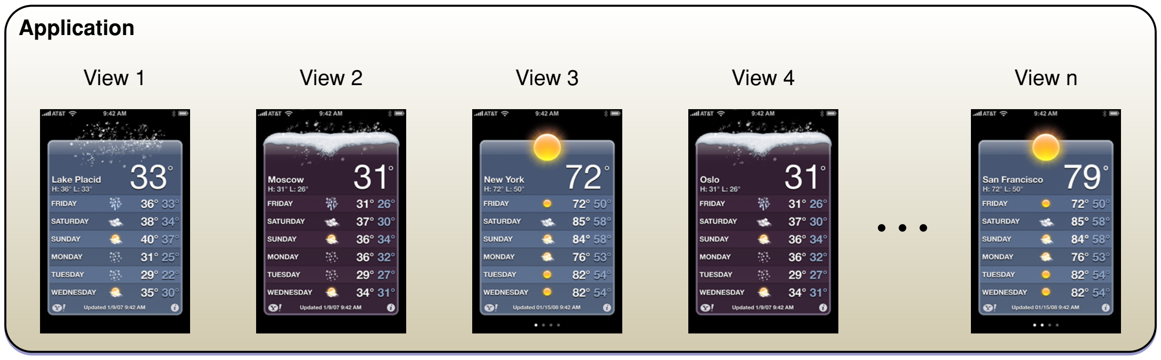


Utility applications are visually attractive, but in a way that enhances the information they display without overshadowing it. People use utility applications to check the status of something or to look something up, so they want to be able to spot the information they're interested in quickly and easily. To facilitate this, a utility application's user interface is uncluttered and provides simple, often standard, views and controls.

A utility application tends to organize information into a flattened list of items; users do not usually need to drill down through a hierarchy of information. Typically, each view in a utility application provides the same organization of data and depth of detail, but can be served by a different source.

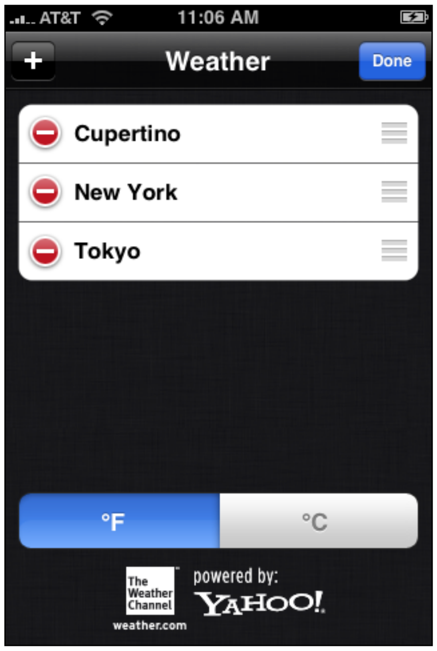
In this way, users can open one application to see similar treatments of multiple subjects. Some utility applications indicate the number of open views; users can navigate through them sequentially, selecting one view after another. Figure 1-3 shows an example of this type of data organization.

Figure 1-3 Utility applications tend to present data in a flattened list



The user interaction model for a utility application is very simple: Users open the application to scan a summary of information and, optionally, change the configuration or source of that information. Utility application may need to support frequent changes to configuration or information source, so they often provide a small set of such options on the back of the main view. Users tap the familiar Info button in the lower-right corner of the main view to see the back. After making adjustments, users tap the Done button to return to the front of the main view. In a utility application, the options on the back of the main view are part of the functioning of the application, not a group of preference-style settings users access once and then rarely, if ever, again. For this reason, utility applications should not supply application-specific settings in the Settings application. Figure 1-4 shows how the Weather application provides configuration options on the back of the main view.

Figure 1-4 Users can make adjustments on the back of Weather

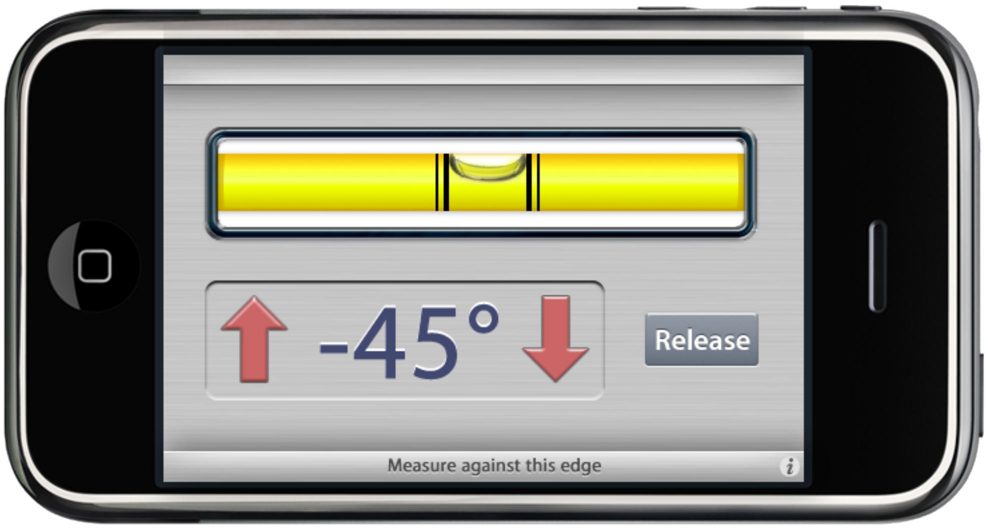


Immersive Applications

An **immersive application** offers a full-screen, visually rich environment that's focused on the content and the user's experience with that content. People use immersive applications to have fun, whether playing a game, viewing media-rich content, or performing a simple task.

It's easy to see how games fit this style of iPhone application, but you can also imagine how characteristics of immersive applications can enhance other types of tasks. Tasks that present a unique environment, don't display large amounts of text-based information, and reward users for their attention are good candidates for the immersive approach. For example, an application that replicates the experience of using a bubble level works well in a graphics-rich, full-screen environment, even though it doesn't fit the definition of a game. In such an application, as in a game, the user's focus is on the visual content and the experience, not on the data behind the experience. Figure 1-5 shows an example of an immersive application that replicates a simple task.

Figure 1-5 An immersive application doesn't have to be a game



An immersive application tends to hide much of the device's user interface, replacing it with a custom user interface that strengthens the user's sense of entering the world of the application. Users expect seeking and discovery to be part of the experience of an immersive application, so the use of nonstandard controls is often appropriate.

Immersive applications may work with large amounts of data, but they do not usually organize and expose it so that users can view it sequentially or drill down through it. Instead, immersive applications present information in the context of the game-play, story, or experience. Also for this reason, immersive applications often present custom navigational methods that complement the environment, rather than the standard, data-driven methods used in utility or productivity applications.

The user interaction model for an immersive application is determined by the experience the application provides. Although it's not likely that a game would need to offer application-specific settings in Settings, other types of immersive applications might. Immersive applications might also furnish configuration options on the back of the main view.

Choosing an Application Style

After reading about productivity, utility, and immersive application styles, think about the type of information your application displays and the task it enables. In theory, the type of application you should create is obvious to you and you're ready to get started; in practice, it's not always that simple. Here is a hypothetical scenario to consider as you make your decision.

If you have a subject you'd like to explore, think about the objects and tasks related to it. Imagine the different perceptions people have of that subject. For example, consider the subject of baseball. Baseball brings to mind, among other things, teams, games, statistics, history, and players. Baseball is probably too extensive a subject for a single application, so consider just the players. Now imagine how you might create an application that relates to players—for example, using their likenesses on baseball cards.

You could develop a productivity application that helps serious collectors manage their baseball card collections. Using list-based formats, you could display cards in a hierarchy of teams, then players, then seasons. In the most detailed view, you could give users the ability to note where they acquired the card, how much they paid for it, its current market value, and how many copies they have. Because the focus of this application is on the data that defines the collection, the user interface streamlines the tasks of seeking and adding information.

You could also develop a utility application that displays the current market value of particular baseball cards. Each view could look like a baseball card with its current value added to the picture, and the back of the view could allow users to select specific cards to track and display. The focus of this application is on individual cards, so the user interface emphasizes the look of the cards and provides a simple control or two that allows users to look for new cards.

Or, of course, you could develop a game. Perhaps the game would focus on the user's memory of certain statistics on individual baseball cards or ability to recognize famous cards. Or perhaps it would simply use baseball cards as icons in another type of game, such as a sliding puzzle. In all of these cases, the focus of the application is on the images on the baseball cards and the game play. The user interface complements this by displaying a few baseball-themed controls and hiding the iPhone OS user interface.

It's important to reiterate that you're not restricted to a single application style. You may find that your application idea is best served by a combination of characteristics from different application styles.

When in doubt, make it simple. Pare the feature list to the minimum and create an application that does one simple thing (see "Create a Product Definition Statement" (page 35) for advice on how to focus your application). When you see how people use and respond to the application, you might choose to create another version of the application with a slightly shifted focus or altered presentation. Or, you might discover a need for a more (or less) detail-oriented version of the same concept.

When You Have an Existing Computer Application

If you have an existing computer application, don't just port it to iPhone OS. People use iPhone OS–based devices very differently than they use desktop and laptop computers, and they have different expectations for the user experience.

Remember that people use iPhone OS–based devices while on the go, and often in environments filled with distractions. This generally means that they want to open your application, use it briefly, and move on to something else. If your application relies on the user's undivided attention for long stretches of time, you need to rethink its structure and goals if you want to bring it to iPhone OS.

If your desktop application enables a complex task or set of tasks, examine how people use it in order to find a couple of subtasks they might appreciate being able to accomplish while they're mobile. For example, a business-oriented application that supports project scheduling, billing, and expense reporting could spawn an iPhone utility application that shows progress summaries for a project, or an iPhone productivity application that allows mobile users to keep track of their business-related expenses.

As you think about how to bring ideas from your desktop application to an iPhone application, apply the 80-20 rule to the design of your application. Estimate that the largest percentage of users (at least 80 percent) will use a very limited number of features in an application, while only a small percentage (no more than 20 percent) will use all the features. Then, consider carefully whether you want to load your iPhone application with the power features that only a small percentage of users want. Be aware that a desktop computer application might be the better environment in which to offer those features, and that it's usually a good idea to focus your iPhone application on the features that meet the needs of the greatest number of people.

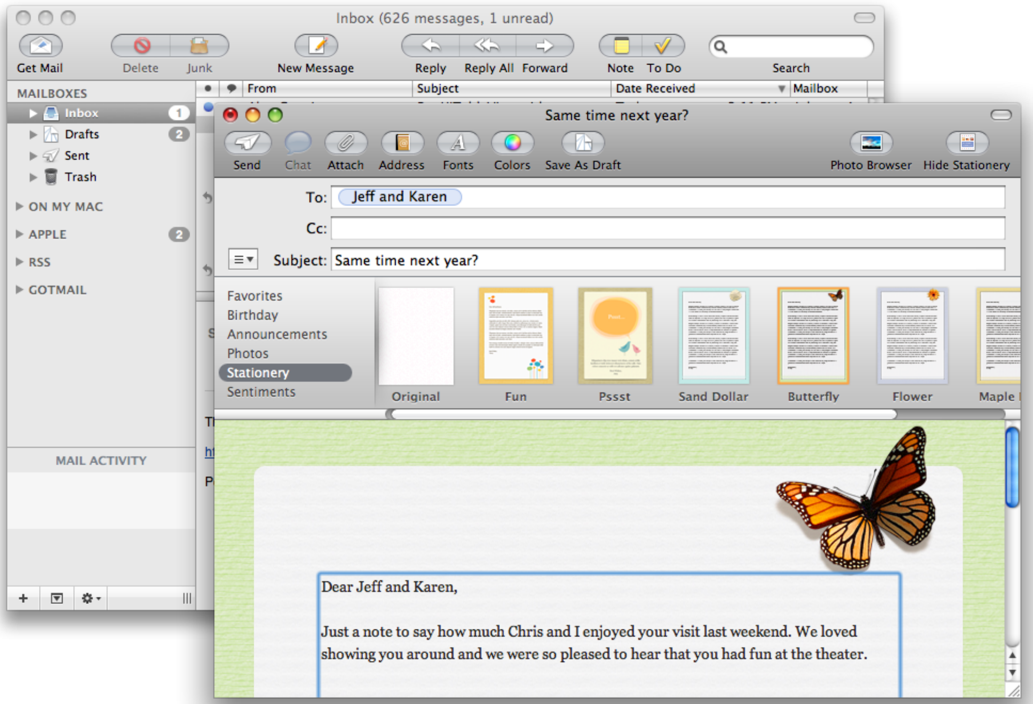
Case Studies: Bringing a Desktop Application to iPhone OS

To help you visualize ways you can create an iPhone OS version of a desktop computer application, this section describes some of the design differences between familiar Mac OS X applications and their iPhone OS counterparts. As you learn about which features and functions in each application were adapted for its iPhone OS version, you will gain insight into the types of design decisions you need to make for your own iPhone application.

Mail

Mail is one of the most highly visible, well-used, and appreciated applications in Mac OS X. It is also a very powerful program, one that allows users to create, receive, prioritize, and store email, track action items and events, and create notes and invitations. Mail provides most of this functionality in a single multipane window. This is convenient for people using a desktop computer, because they can leave a Mail window on the display screen (or minimized to the Dock) all the time and switch to it whenever they choose. Figure 1-6 illustrates many of the features available in the Mail message-viewing and compose windows on the desktop.

Figure 1-6 Mail on the desktop offers a wide range of powerful features in a couple of windows



To: Jeff and Karen

Cc:

Subject: Same time next year?

Dear Jeff and Karen,

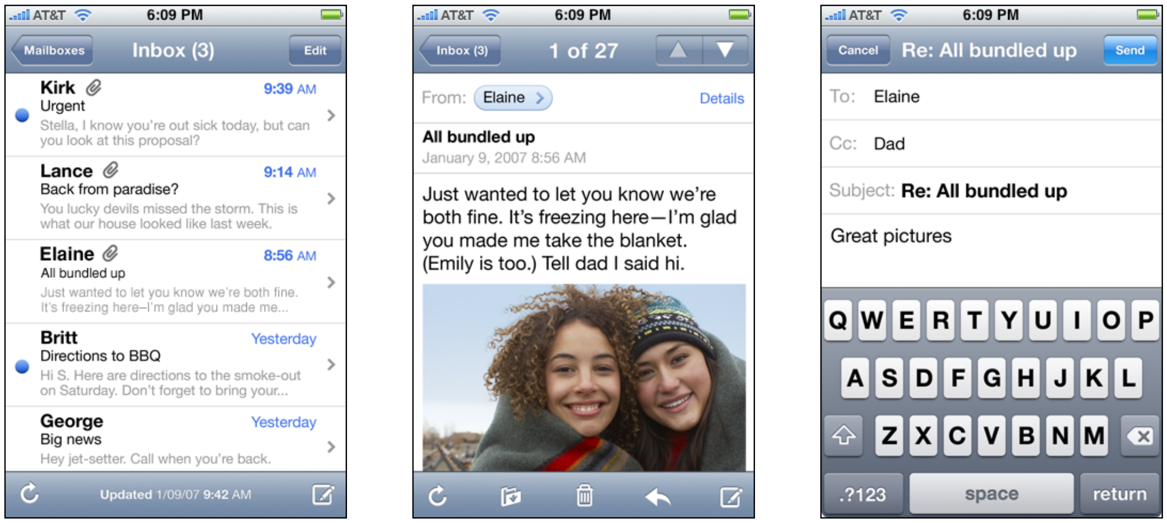
Just a note to say how much Chris and I enjoyed your visit last weekend. We loved showing you around and we were so pleased to hear that you had fun at the theater.

But when people are mobile, their needs for an email application are simpler, and they want access to core functionality quickly. For this reason, Mail on iPhone OS–based devices focuses on the most important things people do with their email: receive, create, send, and organize messages. To do this, it displays a pared-down user interface that makes the organization of the user's accounts and mailboxes clear and centers the user's attention on the messages.

Mail in iPhone OS is a perfect example of a productivity style application: To ease navigation through the content, Mail in iPhone OS takes advantage of the naturally hierarchical organization of people's email and displays on successive pages accounts, mailboxes, message lists, and individual messages. Users drill down from the general (the list of accounts) to the specific (a message) by selecting an item in a list and viewing the things associated with that item. To learn more about the productivity style of iPhone applications, see "Productivity Applications" (page 19).

In addition, Mail in iPhone OS enables actions, such as create and send, by displaying a handful of familiar controls that are easy to tap. Figure 1-7 shows how Mail makes it simple to view and send email in iPhone OS. Note how elements at the top of each screen make it easy for users to know both their current and previous location in the application.

Figure 1-7 Mail in iPhone OS makes it easy to view and send email



Kirk 9:39 AM
Urgent
Stella, I know you're out sick today, but can you look at this proposal?

Lance 9:14 AM
Back from paradise?
You lucky devils missed the storm. This is what our house looked like last week.

Elaine 8:56 AM
All bundled up
Just wanted to let you know we're both fine. It's freezing here–I'm glad you made me...

Britt Yesterday
Directions to BBQ
Hi S. Here are directions to the smoke-out on Saturday. Don't forget to bring your...

George Yesterday
Big news
Hey jet-setter. Call when you're back.

Updated 1/09/07 9:42 AM

From: Elaine

All bundled up
January 9, 2007 8:56 AM

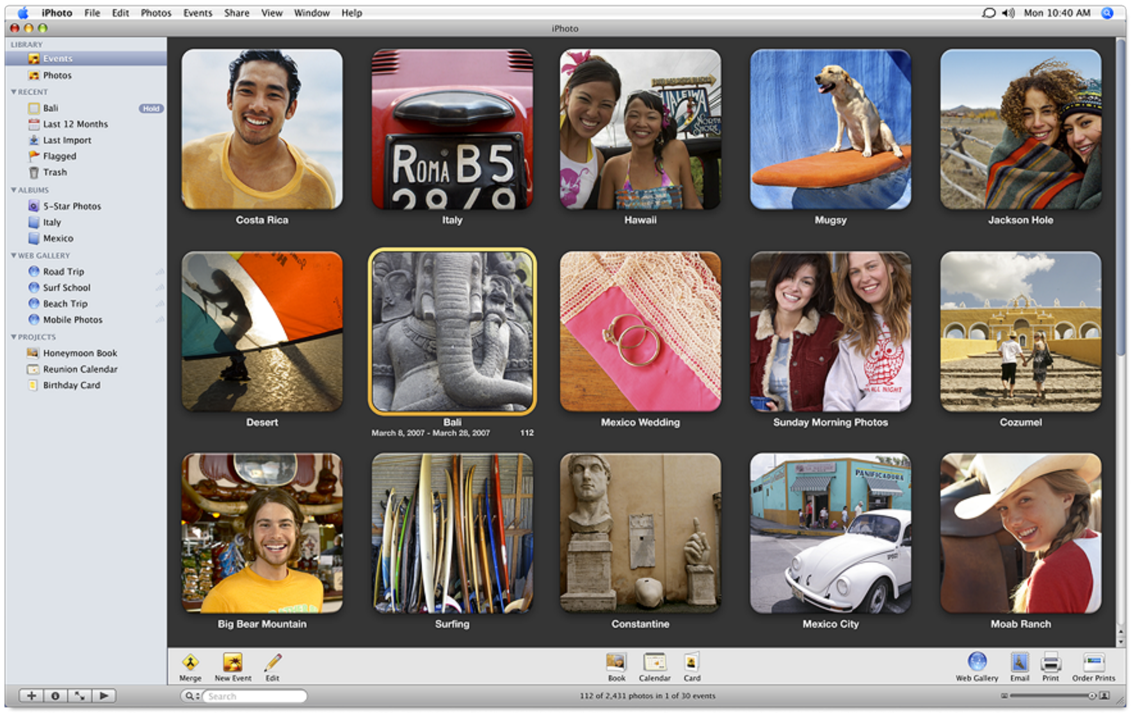
Just wanted to let you know we're both fine. It's freezing here—I'm glad you made me take the blanket. (Emily is too.) Tell dad I said hi.

To: Elaine
Cc: Dad
Subject: **Re: All bundled up**
Great pictures

iPhoto

Another instructive example of a Mac OS X application that was reimagined for iPhone OS is iPhoto. On the desktop, iPhoto supports comprehensive searching and organization, powerful editing capabilities, and creative printing options. When people use iPhoto on their desktop or laptop computers, they appreciate being able to see and organize their entire collection, make adjustments to photos, and manipulate them in various ways. Although the main focus of iPhoto is on the user's content, the application also offers extensive functionality in its window. Figure 1-8 shows the iPhoto user interface on the desktop.

Figure 1-8 The iPhoto user interface

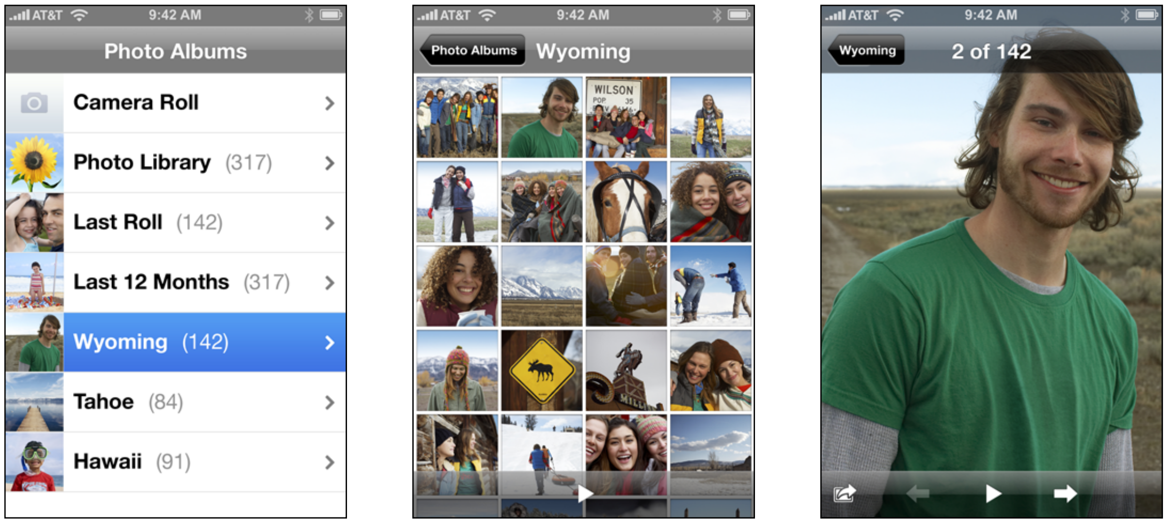


But when they're mobile, people don't have time to edit their photos (and they don't expect to print them); instead, they want to be able to quickly see and share their photos.

To meet this need on iPhone OS–based devices, Apple has provided the Photos application, which focuses on viewing photos and sharing them with others. The Photos user interface revolves around photos; so much so, in fact, that even parts of the device user interface can be hidden. When users choose to view a slideshow of their photos, the Photos application hides the navigation bar, toolbar, and even status bar, and displays translucent versions of these elements when users need to see them.

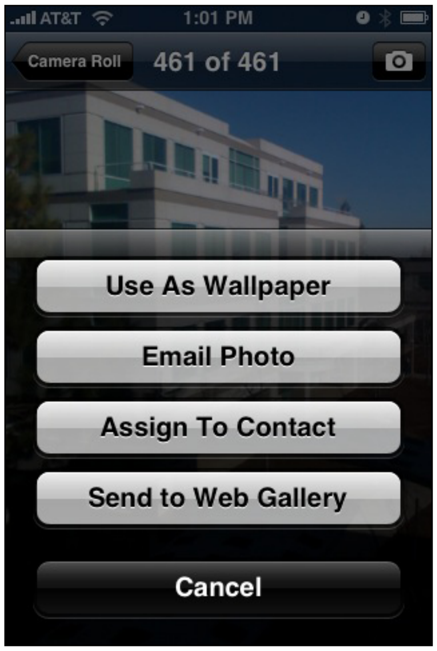
Photos makes it easy for users to organize and find their photos by using a hierarchical arrangement: Users select an album, which contains a collection of photos, and then they select a single photo from the collection. In this way, Photos is an example of an application that combines features of the productivity style and the immersive style (to learn more about these styles, see "Three General Application Styles" (page 19)). Figure 1-9 shows how users can view photos in the Photos application.

Figure 1-9 Three screens in the Photos application



In addition, Photos uses a transient view, called an action sheet (described in "Alerts, Action Sheets, and Modal Views" (page 57)), to give users additional functionality without taking them out of the photo-viewing experience. Figure 1-10 shows how Photos provides options for using an individual photo.

Figure 1-10 Photos gives users options in an action sheet



Human Interface Principles: Creating a Great User Interface

Great applications follow certain fundamental human interface design principles, regardless of the hardware they run on. That's because these principles are based on the way people—users—think and work, not on the capabilities of the device.

Don't underestimate the powerful effect the user interface has on people. A user interface that is unattractive, convoluted, or illogical can make even a great application seem like a chore to use. But a great user interface enhances the application and inspires a positive emotional attachment in users, earning their loyalty.

This chapter describes the human interface principles that underlie every great user interface. You should read this chapter even if you are already familiar with these principles, because it focuses on how to apply them to iPhone applications.

Metaphors

When possible, model your application's objects and actions on objects and actions in the real world. This technique especially helps novice users quickly grasp how your application works. Folders are a classic software metaphor. People file things in folders in the real world, so they immediately understand the idea of putting data into folders on a computer.

Metaphors in iPhone OS include iPod playback controls, tapping controls to make things happen, sliding on-off switches, and flicking through the data shown on picker wheels.

Although metaphors suggest a use for objects and actions in the iPhone OS interface, that use does not limit the software implementation of the metaphor.

As you design your application, be aware of the metaphors that exist in iPhone OS and don't redefine them. At the same time, examine the task your application performs to see if there are natural metaphors you can use. Bear in mind, though, that it's better to use standard controls and actions than to stretch a real-world object or action just to fit your application's user interface. Unless the metaphors you choose are likely to be recognized by most of your users, including them will increase confusion instead of decrease it.

Direct Manipulation

Direct manipulation means that people feel they are controlling something tangible, not abstract. The benefit of following the principle of direct manipulation is that users more readily understand the results of their actions when they can directly manipulate the objects involved.

iPhone OS users enjoy a heightened sense of direct manipulation because of the Multi-Touch interface. Using gestures, people feel a greater affinity for, and sense of control over, the objects they see on screen, because they do not use any intermediate device (such as a mouse) to manipulate them.

To enhance the sense of direct manipulation in your iPhone application, make sure that:

- Objects on the screen remain visible while the user performs actions on them
- The result of the user's action is immediately apparent

See and Point

An iPhone application is better than a person at remembering lists of options, commands, data, and so on. Take advantage of this by presenting choices or options in list form, so users can easily scan them and make a choice. Keeping text input to a minimum frees users from having to spend a lot of time typing and frees your application from having to perform a lot of error checking.

Presenting choices to the user, instead of asking for more open-ended input, also allows them to concentrate on accomplishing tasks with your application, instead of remembering how to operate it.

Feedback

In addition to seeing the results of their actions, users need immediate feedback when they operate controls and ongoing status reports during lengthy operations. Your application should respond to every user action with some visible change. For example, make sure list items highlight briefly when users tap them. Audible feedback also helps, but it can't be the primary or sole feedback mechanism because people may use iPhone OS–based devices in places where they can't hear or where they must turn off the sound. In addition, you don't want to compete with the iPhone OS system sounds users already associate with system alerts.

iPhone OS automatically provides feedback when it's temporarily busy by displaying the activity indicator. During operations that last more than a few seconds, your application should show elapsing progress and, if appropriate, display an explanatory message.

User Control

Allow users, not your application, to initiate and control actions. Keep actions simple and straightforward so users can easily understand and remember them. Whenever possible, use standard controls and behaviors that users are already familiar with.

Provide ample opportunity to cancel operations before they begin, and be sure to get confirmation when the user initiates a potentially destructive action. Whenever possible, allow users to gracefully stop an operation that's underway.

Aesthetic Integrity

Although the ultimate purpose of an application is to enable a task, even if that task is playing a game, the importance of an application's appearance cannot be underestimated. This is because appearance has a strong impact on functionality: An application that appears cluttered or illogical is harder to understand and use.

Aesthetic integrity is not a measure of how beautifully your application is decorated. It's a measure of how well the appearance of your application integrates with its function. For example, a productivity application should keep decorative elements subtle and in the background, while giving prominence to the task by providing standard controls and behaviors.

An immersive application is at the other end of the spectrum, and users expect a beautiful appearance that promises fun and discovery. Although an immersive application tends to be focused on providing diversion, however, its appearance still needs to integrate with the task. Be sure you design the user interface elements of such an application carefully, so that they provide an internally consistent experience.

Designing an iPhone Application: From Product Definition to Branding

As you develop an iPhone application you need to learn how iPhone OS and various aspects of the mobile environment impact your design decisions. This chapter covers a range of guidelines for application design issues, from product definition to branding, and describes how to address them in an iPhone application.

Create a Product Definition Statement

Before you begin designing your application, it's essential to define precisely what your application does. A good way to do this is to craft a product definition statement—a concise declaration of your application's main purpose and its intended audience. Creating a product definition statement isn't merely an exercise. On the contrary, it's one of the best ways to turn a list of features into a coherent product.

To begin with, spend some time defining your user audience: Are they experienced or novice, serious or casual, looking for help with a specific task or looking for entertainment? Knowing these things about your users helps you customize the user experience and user interface to their particular needs and wants.

Because you're designing an iPhone application, you already know a lot about your users. For example:

- They're mobile.
- They want to be able to open your application quickly and see useful content immediately.
- They need to be able to accomplish things in your application with just a few taps.

Now ask yourself what traits might set your users apart from all other iPhone OS users. Are they business people, teenagers, or retirees? Will they use your application at the end of every day, every time they check their email, or whenever they have a few extra moments? The more accurately you define your audience, the more accurate are your decisions about the look, feel, and functionality of your user interface.

For example, if your application helps business people keep track of their expenses, your user interface should focus on providing the right categories and making it easy to enter costs, without asking for a lot of details that aren't central to the task. In addition, you might choose a subtle color palette that appears professional and is pleasant to look at several times a day.

2008-02-29 | © 2008 Apple Inc. All Rights Reserved.

Or, if your application is a game for a target audience of teenagers, you might instead want a user interface that is exciting, language that imparts a feeling of exclusivity, and a color palette that evokes current fashions.

Finally, examine the set of features you intend to deliver. With the image of your user audience in mind, try to distill the list of features into a single statement, a product definition statement, that describes the solution your product offers and who your users are. For example, the desktop iPhoto application allows users to, among other things, organize, edit, share, print, and view photos. But a good product definition statement doesn't focus on features; therefore a sound product definition statement for iPhoto could be "An easy-to-use photo management application for amateur photographers." Notice how important it is to include a definition of your user audience in the product definition statement: Imagine how different an application iPhoto would be if it was designed to be "an easy-to-use photo management application for professional photographers."

A good product definition statement is a tool you can use throughout the development process to determine the suitability of features, tools, and terminology. It's especially important to eliminate those elements that don't support the product definition statement, because iPhone applications have no room to spare for functionality that isn't focused on the main task.

Imagine, for example, that you're thinking of developing an iPhone application people can use when they shop for groceries. In the planning stage, you might consider including a wide range of activities users might like to perform, such as:

- Getting nutritional information about specific foods
- Finding coupons and special offers
- Creating and using shopping lists
- Locating stores
- Looking up recipes
- Comparing prices
- Keeping a running total of prices

However, you believe that your users are most concerned with remembering everything they need to buy, that they would like to save money if possible, and that they're probably in a hurry to get home with their purchases. Using this audience definition, you craft a product definition statement for your application, such as "A shopping list creation and coupon-finding tool for people in a hurry." Filtering your list of potential features through this product definition statement, you decide to focus primarily on making shopping lists easy to create, store, and use. You also offer users the ability to find coupons for the items on their list. Even though the other features are useful (and might become primary features of other applications), they don't fit the product definition statement for this application.

When you've settled on a solid product definition statement and you've started to use it as a filter for your proposed features, you might also want to use it to make sure your initial decision on application type is still the right one. If you began you development process with a specific application type in mind, you might find that the process of defining a product definition statement has changed the landscape.

Incorporate Characteristics of Great iPhone Applications

Great iPhone applications do precisely what users need while providing the experience users want. To help you achieve this balance in your application, this section examines some of the characteristics of great iPhone applications and provides advice on how to build them into your product.

Build in Simplicity and Ease of Use

Simplicity and ease of use are fundamental principles for all types of software, but in iPhone applications they are critical. iPhone OS users are probably doing other things while they simultaneously use your application. If users can't quickly figure out how to use your application, they're likely to move on to a competitor's application and not come back.

As you design the flow of your application and its user interface, follow these guidelines to build in simplicity and ease of use:

- Make it obvious how to use your application.
- Concentrate frequently used, high-level information near the top of the screen.
- Minimize text input.
- Express essential information succinctly.
- Provide a fingertip-size target area for all tappable elements.

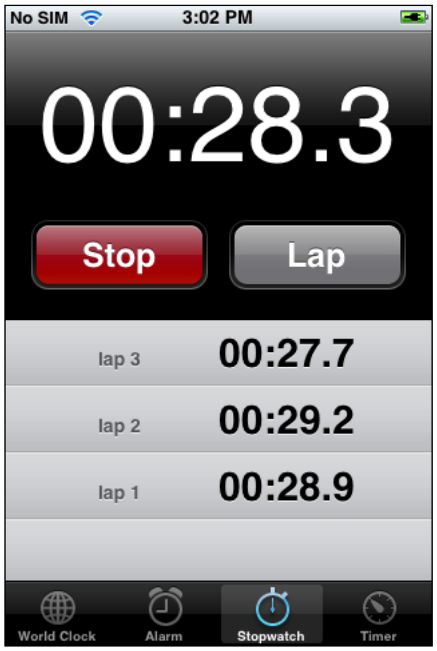
The following sections explain each guideline for simplicity and ease of use in more detail.

Make It Obvious

You can't assume that users have the time (or can spare the attention) to figure out how your application works. Therefore, you should strive to make your application instantly understandable to users.

The main function of your application should be immediately apparent. You can make it so by minimizing the number of controls from which users have to choose and labeling them clearly so users understand exactly what they do. For example, in the built-in Stopwatch function (part of the Clock application), shown in Figure 3-1, users can see at a glance which button stops and starts the stopwatch and which button records lap times.

Figure 3-1 The built-in Stopwatch function makes its usage obvious



Think Top Down

People can tap the screen of an iPhone OS–based device with their fingers or their thumbs. When they use a finger, people tend to hold the device in their nondominant hand (or lay it on a surface) and tap with a finger of the dominant hand. When they use thumbs, people either hold the device in one hand and tap with that thumb, or hold the device between their hands and tap with both thumbs. Whichever method people use, the top of the screen is most visible to them.

Because of these usage patterns, you should design your application's user interface so that the most frequently used (usually higher level) information is near the top, where it is most visible and accessible. As the user scans the screen from top to bottom, the information displayed should progress from general to specific and from high level to low level.

Minimize Required Input

Inputting information takes users' time and attention, whether they tap controls or use the keyboard. If your application requires a lot of user input before anything useful happens, it slows users down and can discourage them from persevering with it.

Of course, you often need some information from users, but you should balance this with what you offer them in return. In other words, strive to provide as much information or functionality as possible for each piece of information users provide. That way, users feel they are making progress and are not being delayed as they move through your application.

When you ask for input from users, consider using a table view (or a picker) instead of text fields. It's usually easier for users to select an item from a list than to type words.

Express Information Succinctly

When your user interface text is short and direct, users can absorb it quickly and easily. Therefore, identify the most important information, express it concisely, and display it prominently so users don't have to read too many words to find what they're looking for or to figure out what to do next.

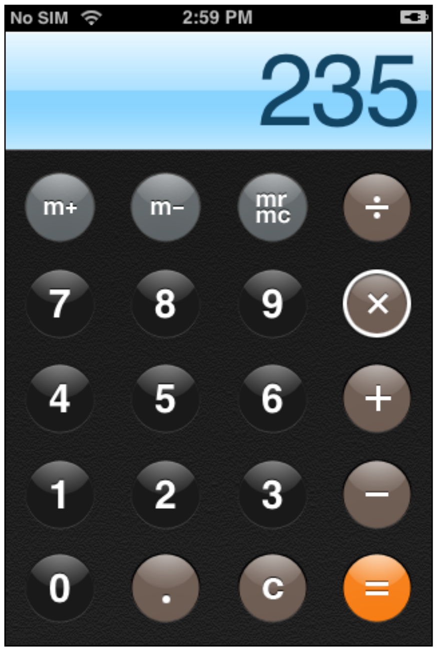
To help you do this, think like a newspaper editor and strive to convey information in a condensed, headline style. Give controls short labels (or use well-understood symbols) so that users understand how to use them at a glance.

Provide Fingertip-Size Targets

If your layout places controls too close together, users must spend extra time and attention being careful where they tap. Even so, they are more likely to tap the wrong element. A simple, easy-to-use user interface spaces controls and other user-interaction elements so that users can tap accurately with a minimum of effort.

For example, the built-in Calculator application displays large, easy-to-tap controls that each have a target area of about 44 x 44 points. Figure 3-2 shows the Calculator application.

Figure 3-2 The built-in Calculator application displays fingertip-size controls



Focus on the Primary Task

An iPhone application that establishes and maintains focus on its primary functionality is satisfying and enjoyable to use. As you design your application, therefore, stay focused on your product definition statement and make sure every feature and user interface element supports it. See "Create a Product Definition Statement" (page 35) for some advice on how to create a product definition statement.

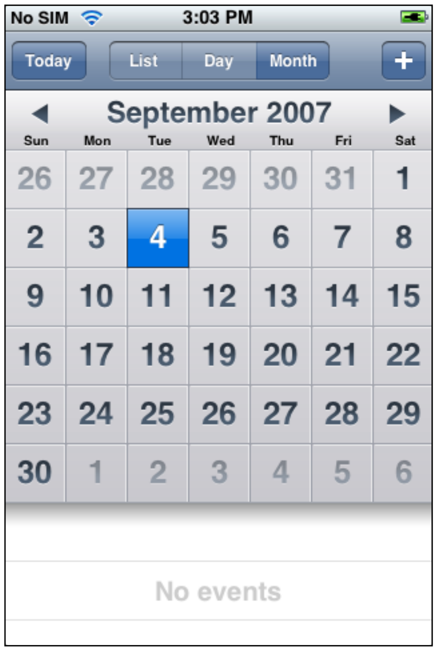
2008-02-29 | © 2008 Apple Inc. All Rights Reserved.

A good way to achieve focus is to determine what's most important in each context. As you decide what to display in each screen always ask yourself, Is this critical information or functionality users need right now? In other words, Is this information or functionality the user needs while shopping in a store or while walking between meetings? If not, decide if the information or functionality is critical in a different context or if it's not that important after all. For example, an application that helps users keep track of car mileage loses focus on this functionality if it also keeps track of car dealer locations.

When you follow the guidelines for making your application simple and easy to use, you help make your solution focused. In particular, you want to make the use of your application obvious and minimize user input. This makes it easier for users to arrive quickly at the most important parts of your application, which tightens the focus on your solution (for specifics on these guidelines, see "Build in Simplicity and Ease of Use" (page 37)).

For example, the built-in Calendar application (shown in Figure 3-3) is focused on days and the events that occur on them. Users can use the clearly labeled buttons to highlight the current day, select a viewing option, and add events. The most important information, that is, the days and the events associated with them, is the most prominent. User input is simplified by allowing users to choose from lists of event times, repetition intervals, and alert options, instead of requiring keyboard entry for all input.

Figure 3-3 The built-in Calendar application is focused on days and events

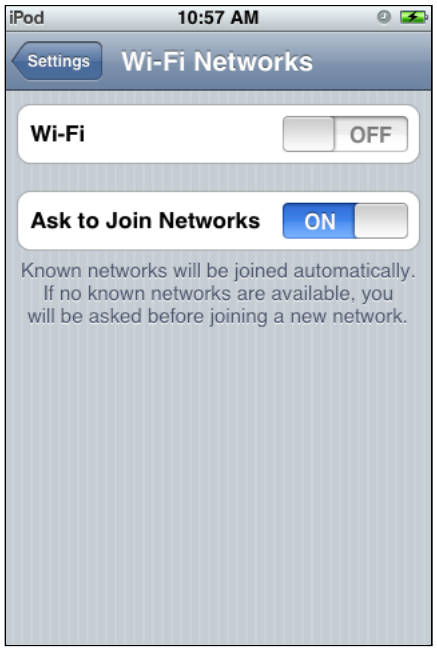


Communicate Effectively

Communication and feedback are as important in iPhone applications are they are in desktop computer applications. Users need to know whether their requests are being processed and when their actions might result in data loss or other problems. That said, it's also important to avoid overdoing it by, for example, alerting the user to conditions that aren't really serious or asking for confirmation too often.

In all your text-based communication with users, be sure to use user-centric terminology; specifically, avoid technical jargon in the user interface. Use what you know about your users to determine whether the words and phrases you plan to use are appropriate. For example, the Wi-Fi Networks preferences screen uses clear, nontechnical language to describe how the device connects to networks, as shown in Figure 3-4.

Figure 3-4 Use user-centric terminology in your application's user interface



Wi-Fi: OFF

Ask to Join Networks: ON

Known networks will be joined automatically. If no known networks are available, you will be asked before joining a new network.

Handle Common Tasks

iPhone applications handle many common tasks in ways that may seem different to you, if your experience is with desktop or laptop computer applications. This section describes these tasks from the human interface perspective; for the technical details you need to implement these guidelines in code, see *iPhone OS Programming Guide*.

Starting. iPhone applications should start instantly so users can begin using them without delay. When starting, iPhone applications should:

- Specify the appropriate status bar style (see "The Status Bar" (page 51) for information about the available styles).
- Display a launch image that closely resembles the first screen of the application. This decreases the perceived launch time of your application. For more information, see "Launch Images" (page 94).
- Avoid displaying an About window, a splash screen, or providing any other type of startup experience that prevents people from using your application immediately.
- By default, launch in portrait orientation. If you intend your application to be used only in landscape orientation, launch in landscape and allow users to rotate the device accordingly.

A landscape-only application should support both landscape orientations—that is, with the Home button on the right or on the left. By default, a landscape-only application should launch in the orientation with the Home button on the right.

- Restore state from the last time your application ran.

Stopping. iPhone applications stop when users open a different application or use a device feature, such as the phone. In particular, note that users don't tap a close button or select Quit from a menu. iPhone applications should:

- Be prepared to receive an exit or terminate notification at any time. Therefore, save user data as soon as possible and as often as reasonable.
- Save the current state when stopping, at the finest level of detail possible. For example, if your application displays scrolling data, save the current scroll position.

Handling settings. iPhone applications can offer settings that define preferred application behaviors or configuration options users can set to change some functionality of the application. Settings that define preferred application behaviors, similar to preferences in desktop computer applications, are accessible in the built-in Settings application. Configuration options should be available within the application context; often they are on the back of the main view (for an example of this, see "Utility Applications" (page 21))

As you design your iPhone application, you need to decide which method makes sense. Be aware that users must quit your application to adjust settings in the Settings application, so you should not provide settings that users need to set more than once. If you can simplify your application to make settings unnecessary, you should do so. See *iPhone OS Programming Guide* for more information about providing settings in your iPhone application.

Configuration options should be offered within your application; typically, they are displayed on the back of a window. Unlike settings, configuration options are likely to be changed frequently as users choose to see information from new sources or in different arrangements. You can react dynamically to changes users make to these options, because users do not leave your application to access them.

Handling orientation changes. Users can rotate iPhone OS–based devices at any time, and they expect the content they're viewing to respond appropriately. In your iPhone application, be sure to:

- Be aware of accelerometer values (for more information on the accelerometer and references to accelerometer programming interfaces, see *iPhone OS Programming Guide*). If appropriate, your application should respond to all changes in device orientation.
- If there's a part of your application's user interface that displays in one orientation only, it's appropriate for that area to appear in that orientation and not respond to changes in device orientation. For example, when a user selects an iPod video to view, the video displays in landscape orientation, regardless of the current device orientation. This signals the user to physically rotate the device to view the video. The important point about this example is that iPod does not provide a "rotate now" button; instead, the user knows to rotate the device because the video appears in landscape orientation.

  Allow users to physically rotate the device to correctly view the parts of your application's user interface that require a specific orientation. Avoid creating a control or defining a gesture that tells users to rotate the device.

Using sound. iPhone OS–based devices use a wide range of sounds to communicate with users, such as alert sounds, ringtones, and keyboard sounds. For the most part, users can customize the sounds used for these functions, or turn them off altogether.

Because users tend to associate particular sounds with specific events or situations, it's important to avoid redefining these sounds in your iPhone application. If your application uses sound to communicate with users in some way, be sure you don't confuse users by using established sounds in a conflicting way.

Providing selections. iPhone OS includes a few elements that support selection tasks. You should use these selection controls because users are already familiar with their behavior. In general, you should not try to replicate the appearance and behavior of selection controls you might see in a desktop computer application, such as an application menu. iPhone OS provides the following elements you can use to offer choices to users:

- Lists (that is, table views). Users tap a row in a list to select an item. Lists are suitable for displaying almost any number of choices. For in-depth information on the ways you can use table views in your application, see "Table Views" (page 67).
- Pickers, including date and time pickers. Users spin the wheels in a picker until each wheel displays the desired part of a multipart value, such as a calendar date that comprises year, month, and day. For more information about using pickers in your iPhone application, see "Date and Time Pickers" (page 80) and "Pickers" (page 85).
- Switch controls. Users slide a switch control from one side to the other, revealing one of two values. A switch control is intended to offer a simple choice within a list. For more information about switch controls, see "Switch Controls" (page 75).

Understand the User Interaction Model

People use their fingers to operate the unique Multi-Touch interface of iPhone OS–based devices, tapping, flicking, and pinching to select, navigate, and read web content and use applications. There are real advantages to using fingers to operate a device: They are always available, they are capable of many different movements, and they give users a sense of immediacy and connection to the device that's impossible to achieve with an external input device, such as a mouse.

However, fingers have one major disadvantage: They are much bigger than a mouse pointer, regardless of their size, their shape, or the dexterity of their owner. In the context of a display screen, fingers can never be as precise as a mouse pointer. Additionally, there are some actions users can take with the combination of a mouse and keyboard that are difficult to replicate using fingers alone. These actions include text selection, drag-and-drop operations, and cut, copy, and paste actions.

Fortunately, you can meet the challenges of a finger-based input system by having a good user interface design. For the most part, this means making sure your layout accommodates the average size of a fingertip and by finding alternatives to drag-and-drop and cut, copy, and paste.

iPhone users perform specific movements, called **gestures**, to get particular results. Table 3-1 lists the gestures users can perform. Your application can receive notifications of gesture events and respond accordingly. For more information on how to handle events created by gestures, see *iPhone OS Programming Guide*.

Table 3-1 Gestures users make to interact with iPhone

Gesture	Action
Tap	To press or select a control or item (analogous to a single mouse click).
Double tap	To zoom in and center a block of content or an image. To zoom out (if already zoomed in).
Flick	To scroll or pan quickly.
Drag	To scroll or pan.
Swipe	In a table-view row, to reveal the Delete button.
Pinch open	To zoom in.
Pinch close	To zoom out.
Touch and hold	In editable text, to display a magnified view for cursor positioning.
Two-finger scroll	To scroll up or down within a content area.

Incorporate Branding Elements Cautiously

Branding is most effective when it is subtle and understated. People use your iPhone application to get things done or to be entertained; they don't want to feel as if they're being forced to watch an advertisement. Therefore, you should strive to incorporate your brand's colors or images in a refined, unobtrusive way. For example, you might use a custom color scheme in views and controls.

The exception to this is the Home screen icon, which should be focused on your brand. (The Home screen icon is the icon users can put on their Home screens after they install your application.) Because users see your Home screen icon frequently, it's important to spend some time balancing eye-appeal with brand recognition. For some guidelines on creating a Home screen icon, see "Application Icons" (page 93).

Working with iPhone OS User Interface Elements

User interface elements in iPhone OS include views and controls. **Views** provide content regions with well-defined sets of functionality. **Controls** are graphic objects that cause instant actions or visible results. Users make gestures in the Multi-Touch interface to manipulate these onscreen user interface elements to drive the application and communicate with the device (see "Understand the User Interaction Model" (page 43) for more information on gestures).

iPhone OS defines the standard appearance of these user interface elements, and delivers consistent behaviors that users expect. Read Part II to learn about the types of user interface elements available and how to use them in your application.

P A R T   I I

Working with iPhone OS User Interface Elements

A Tour of the Application Window

To help you understand how iPhone OS user interface elements fit together, this chapter introduces the application window and its contents. To learn more about the appearance, behavior, and usage guidelines of individual user interface elements, be sure to read the chapters following this one. Understanding how each user interface element is designed to be used helps you use it correctly in your application and, if appropriate, customize it to meet your needs.

The Application Window and its Contents

Every application, regardless of type, has an application window. You can think of the application window as representing the device screen, because while your application is running, its window fills the available screen area. To users, there is little, if any, difference between the concepts of application window and device screen.

Inside the application window are various combinations of views and controls. Some views include specific controls that do not belong anywhere else, and some controls can be used in a variety of views.

Alerts, action sheets, and modal views are distinct types of views that do not exist in an application window like most other views; instead, they float above an application window and its views. See "Alerts, Action Sheets, and Modal Views" (page 57) for more information about these views.

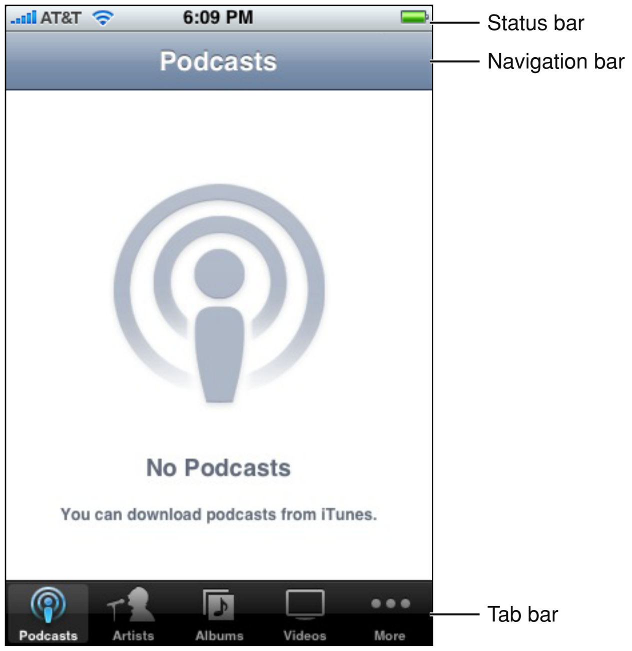
Four types of views have special status in the user interface of an application, although they do not need to be included or always visible in every application. These are:

- The status bar. This is a unique view that isn't technically part of the application window, although an application can customize the appearance of the status bar to some extent. See "The Status Bar" (page 51) for more information.
- The navigation bar. This optional view appears just beneath the status bar and includes text and segmented controls. See "Navigation Bars" (page 52) for more information.
- The tab bar. This optional view appears at the lower edge of the window and contains segments that activate different modes in the application. See "Tab Bars" (page 55) for more information.
- The toolbar. This optional view appear at the lower edge of the window and includes controls that perform specific actions in the application. See "Toolbars" (page 54) for more information.

Figure 4-1 shows three of these views in a window.

2008-02-29 | © 2008 Apple Inc. All Rights Reserved.

Figure 4-1 A window that contains a status bar, a navigation bar, and a tab bar



In addition to some combination of status bar, navigation bar, tab bar, and toolbar, an application window can contain arbitrary views to display content, such as table views, web views, and image views. Figure 4-2 shows examples of two of the content views available in iPhone OS: a style of table view and image views. To learn more about the behavior and appearance of some of these views, in addition to controls associated with them, see "Content Views" (page 67).

Figure 4-2 Two types of content views

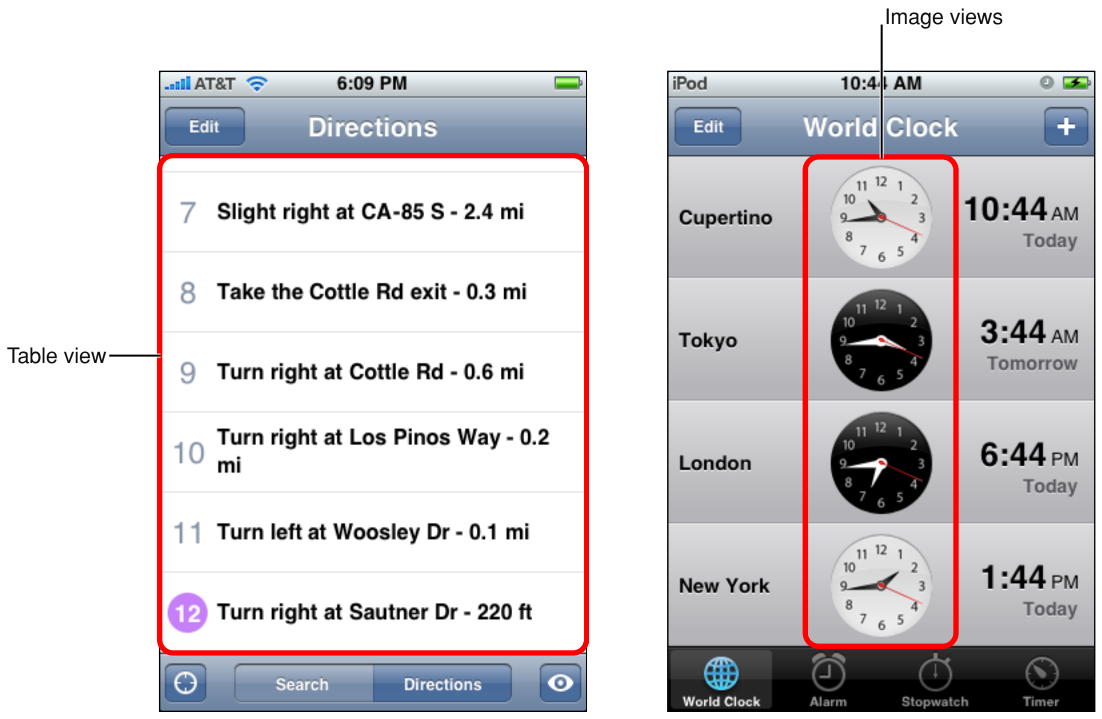


Image views
Table view
AT&T
6:09 PM
Edit
Directions
7 Slight right at CA-85 S - 2.4 mi
8 Take the Cottle Rd exit - 0.3 mi
9 Turn right at Cottle Rd - 0.6 mi
10 Turn right at Los Pinos Way - 0.2 mi
11 Turn left at Woosley Dr - 0.1 mi
12 Turn right at Sautner Dr - 220 ft
Search
Directions
iPod
10:44 AM
Edit
World Clock
+
Cupertino
10:44 AM Today
Tokyo
3:44 AM Tomorrow
London
6:44 PM Today
New York
1:44 PM Today
World Clock
Alarm
Stopwatch
Timer

As mentioned above, there are some controls that are available only in specific views. An example of such a control is the disclosure indicator, which has a specific use in a table view. You can see an example of the disclosure indicator (it looks like >) in the left-hand list in Figure 7-1 (page 68). These controls are described in the sections that cover their associated views. In addition to these, however, there are a handful of controls, such as the progress bar and the slider, that you can use in arbitrary views. These controls are described in "Application Controls" (page 79).

Using Views and Controls in an Application Window

In iPhone OS, UIKit determines the behavior and default appearance of views and controls. As much as possible, you should use the standard user interface elements UIKit provides and follow their recommended usages. Doing this helps you in two important ways:

- Users are accustomed to the look and behavior of standard views and controls. When you use familiar user interface elements, users can depend on their prior experience to help them as they learn to use your application.
- If iPhone OS changes the look or behavior of standard views or controls, you benefit from receiving the updated appearance.

Many controls support some kind of customization, usually in color and content (such as the addition of a text label or an image). If you're developing an immersive application, it's reasonable to create controls that are completely different from the default controls. This is because you're creating a unique environment, and discovering how to control that environment is an experience users expect in such applications.

In general, though, you should avoid radically changing the appearance of a control that performs a standard action. If you use unfamiliar controls to perform standard actions, users will have to spend time discovering how to use them and will wonder what, if anything, your controls do that the standard ones do not.

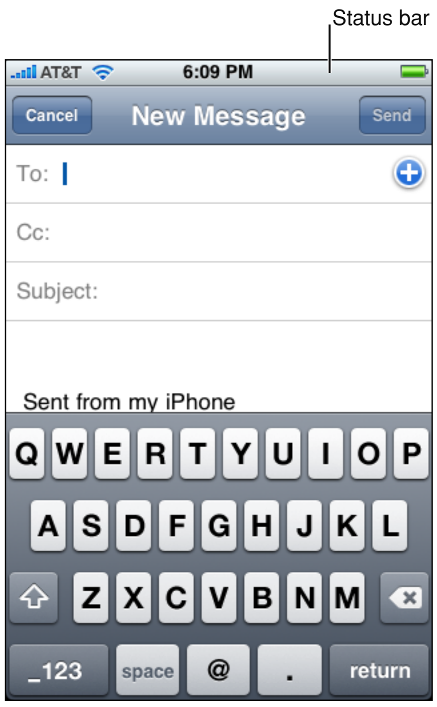
Views Having Special Status in an Application

The status bar, navigation bar, tab bar, and toolbar are views that have specifically defined appearances and behaviors in an iPhone application window. These views are not required to be present in every application window (immersive applications often don't display any of them), but if they are present, it's important to use them correctly. This is because they provide familiar anchors to users of iPhone OS–based devices, who are accustomed to the information they display and the functions they perform.

The Status Bar

The status bar shows users important information about their device, including cell signal strength, the current network connection, and battery charge. Figure 5-1 shows an example of a status bar.

Figure 5-1 A status bar contains important information for users



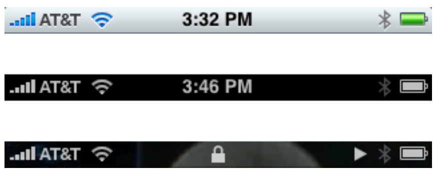
Although a full-screen, immersive application can hide the status bar, you should carefully consider the ramifications of this design decision. People expect to be able to see the current battery charge of their devices; hiding this information, and requiring them to quit your application to get it, is not an ideal user experience.

If you do hide the status bar, you should consider making it accessible with a simple gesture. For example, Photos displays individual photos from a camera roll in a full-screen view that fades out the status bar, navigation bar, and toolbar after a few seconds. This is appropriate because in Photos, users focus on viewing the content, not interacting with it. However, the user can bring back the status bar, navigation bar, and toolbar with a single tap on the screen. If you do something like this, be sure the gesture you define to bring back these views is an obvious one, such as the tap, and not one that users expect to use to perform some other action in your application. To return to the Photos example, it makes sense to use the single tap to reveal the status bar, navigation bar, and toolbar, because users expect to use the flick gesture to see more photos.

Although you have little control over the contents of the status bar, you can customize its appearance and, to some extent, its behavior. Specifically, you can:

- Indicate whether the network activity indicator should be visible. You should display the network activity indicator if your application is performing a network operation that will take more than a couple of seconds. If the operation will finish sooner than that, you don't have to show the network activity indicator, because it would be likely to disappear before users notice its presence.
- Specify the color of the status bar. You can choose gray (the default color), opaque black, or translucent black (that is, black with an alpha value of 0.5). Figure 5-2 shows these styles. (Note that you set a value in your `Info.plist` file to specify the status bar style; see *iPhone OS Programming Guide* for more information.)
- Set whether the change from the current status bar color to the new color should be animated. (Note that the animation causes the old status bar to slide up until it disappears off the screen, while the new status bar slides into place.)

Figure 5-2 Three styles of status bars



Navigation Bars

A navigation bar appears at the upper edge of an application window, just below the status bar. In a navigation bar, users expect to see the title of the current view and controls that affect its contents, in addition to navigational controls when appropriate. Navigation bars are especially useful in productivity applications (described in "Productivity Applications" (page 19)), because these applications typically arrange information in a hierarchy.

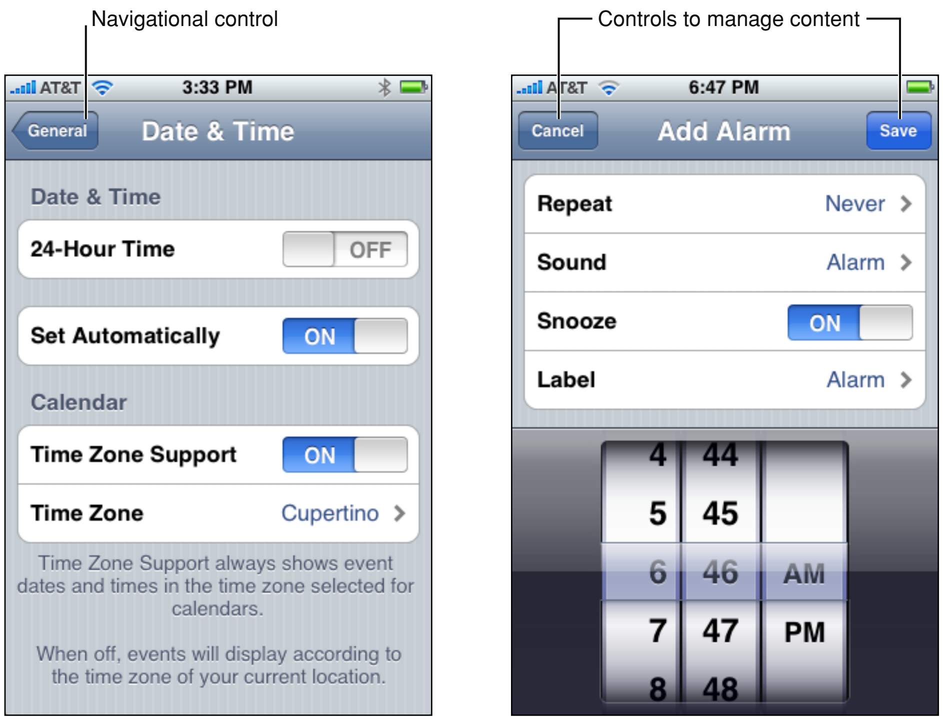
Navigation bars have two purposes:

- To enable navigation among different views in an application

- To provide controls that manage the items in a view

Figure 5-3 shows examples of both these uses.

Figure 5-3 Navigation bars can contain navigational controls and controls to manage content



Date & Time
24-Hour Time OFF
Set Automatically ON
Calendar
Time Zone Support ON
Time Zone Cupertino

Time Zone Support always shows event dates and times in the time zone selected for calendars.

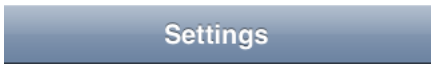
When off, events will display according to the time zone of your current location.

Repeat Never
Sound Alarm
Snooze ON
Label Alarm

4 44
5 45
6 46 AM
7 47 PM
8 48

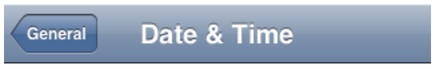
At a minimum, a navigation bar displays the title of the current view, centered along its width, as shown in Figure 5-4. The initial view in a productivity application should include a navigation bar that displays only the title of the first view because the user hasn't yet navigated to another location.

Figure 5-4 A navigation bar displays the title of the current view



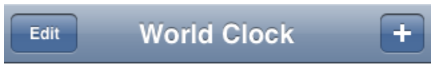
As soon as the user navigates to another view, the navigation bar should change the title to be the title of the new location, and should provide a back button that contains the title of the previous location. Note that you need to support this behavior in your code. For example, Figure 5-5 shows the navigation bar in Date & Time settings, which is one of the settings categories in General settings.

Figure 5-5 A navigation bar can contain a navigational control



Optionally, you can choose to display a second button to the right of the title. Also, if you do not need to display a back button (because your application does not support hierarchical navigation), you can opt instead to display a button that affects the contents of the view, such as an edit button, to the left of the title. Figure 5-6 shows an example of this.

Figure 5-6 A navigation bar can contain controls that manage the content in the view



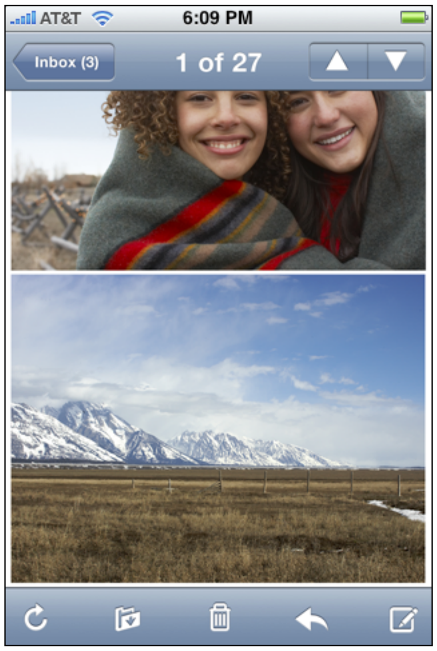
Although you can specify a font for all text displayed in a navigation bar, it's recommended that you use the system font for maximum readability. When you use the appropriate UIKit programming interfaces to create your navigation bar, the system font is used automatically to display the title.

Toolbars

If your application provides a number of actions users can take in the current context, it might be appropriate to provide a toolbar. A toolbar appears at the bottom edge of the window.

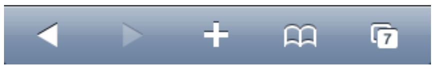
For example, when users view a message in Mail, the application provides a toolbar that contains items for deleting, replying to, and moving the message, in addition to checking for new mail and composing a new message. In this way, users can stay within the message-viewing context and still have access to the commands they need to manage their email. Figure 5-7 shows what this looks like.

Figure 5-7 A toolbar provides functionality within the context of a task



The toolbar displays toolbar items equally spaced across the width of the toolbar. It's a good idea to constrain the number of items you display in a toolbar, so users can easily tap the one they want. Remember that the hit-region of a user interface element is recommended to be 44 x 44 points, so providing no more than five toolbar items is reasonable.Figure 5-8 shows an example of appropriate spacing of toolbar items in a toolbar.

Figure 5-8 Appropriately spaced toolbar items

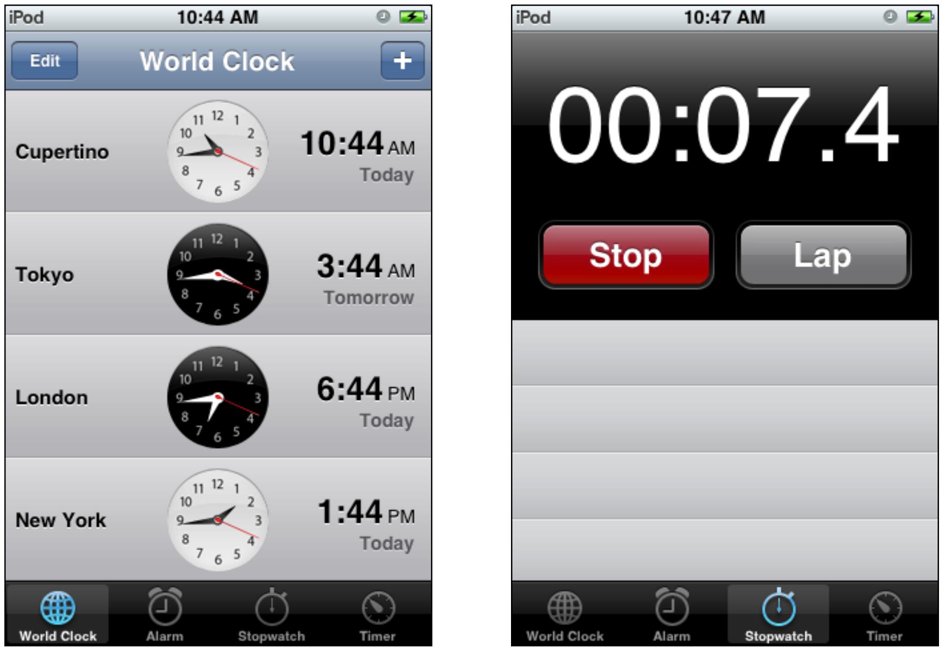


Tab Bars

If your application provides different perspectives on the same set of data, or different subtasks related to the overall function of the application, you might want to use a tab bar. A tab bar appears at the bottom edge of the window.

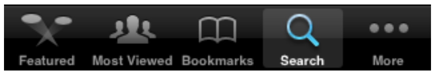
For example, on iPhone, iPod uses a tab bar to allow users to choose which part of their media collection to focus on, such as Podcasts, artists, videos, or playlists. The Clock application, on the other hand, uses a tab bar to give users access to the four functions of the application, namely, World Clock, Alarm, Stopwatch, and Timer. Figure 5-9 shows how selecting a tab in a tab bar changes the view in Clock.

Figure 5-9 A tab bar switches views in an application



The tab bar displays icons and text in tabs, each of which are equal in width and display a black background. Because it's recommended that the hit-region of a user interface element be 44 x 44 points, it's a good idea to display no more than five tabs in a tab bar; otherwise, it can be difficult for users to tap a specific one. When a tab is selected, its background lightens and the image in the tab is highlighted. Figure 5-10 shows how this looks.

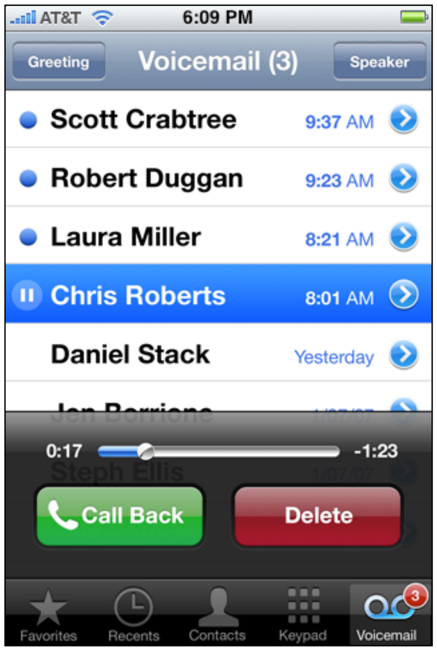
Figure 5-10 A selected tab in a tab bar



You can display a badge on a tab to communicate with users in a nonintrusive, understated way. This type of feedback is suitable for communicating information that isn't critical to the user's task or context, but that is useful to know. The badge looks similar to the one Phone displays on the Voicemail tab to indicate the number of unheard messages: it is a red oval with white text inside that's near the upper right corner of the tab. Associating a badge with a specific tab allows you to connect the information in the badge with a particular mode in your application. (Note that your application cannot display a badge on your application icon in the Home screen, because third-party iPhone applications do not run in the background.)

Figure 5-11 shows an example of a badge on a tab.

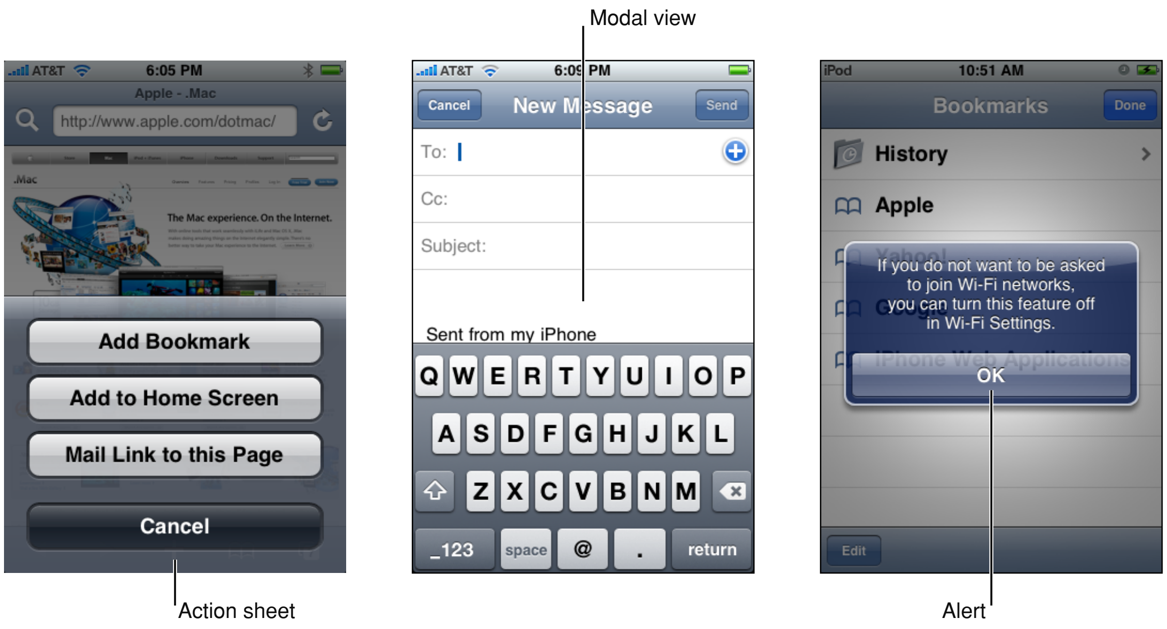
Figure 5-11 A badge conveys information in a tab bar



Alerts, Action Sheets, and Modal Views

Alerts, action sheets, and modal views are types of views that appear when something requires the user's attention or when additional choices or functionality need to be offered. Figure 6-1 shows examples of these types of views.

Figure 6-1 An action sheet, a modal view, and an alert



Usage and Behavior

Alerts, action sheets, and modal views are all modal, which means that users must explicitly dismiss them, by tapping a button, before they can continue to use the application. Although there are times when you need to warn users of potentially dangerous actions or provide extra choices, it's important to avoid overusing these views. This is because:

- All types of modal views interrupt the user's workflow.
- The too-frequent appearance of a view requesting confirmation of the user's actions is likely to be more annoying than helpful.

Alerts, in particular, should be used only rarely. When too many alerts appear, users are likely to dismiss them without reading them, just to get them out of the way.

Alerts, action sheets, and modal views are designed to communicate different things:

- Alerts give users information that affects their use of the application (or the device). Alerts are usually unexpected, because they generally tell users about a problem or a change in the current situation.
- Action sheets give users additional choices related to the action they are taking. Users learn to expect the appearance of an action sheet when they begin a potentially destructive action (such as deleting all recent calls) or when the action they initiated can be completed in different ways (such as a send action for which users can specify one of several destinations).
- Modal views provide more extensive functionality in the context of the current task or provide a way to perform a subtask directly related to the user's workflow.

These types of views also differ in appearance and behavior, which underscores the difference in the messages they send. Because users are accustomed to the appearance and behavior of these views, it's important to use them consistently and correctly in your application. Read the following sections to learn more about using alerts, action sheets, and modal views.

Using Alerts

An alert pops up in the middle of the screen and floats above the application window and its views (as shown in the far right of Figure 6-1 (page 57)). The unattached appearance of an alert emphasizes the fact that its arrival is due to some change in the application or the device, not necessarily as the result of the user's most recent action. An alert should display text that explains the situation and a button the user taps to dismiss the view.

Users are accustomed to seeing alerts from the device or from built-in applications that run in the background, such as Text, but you should seldom need to use them in your application. For example, you might use an alert to tell users that, because their network connection has changed, the current task will take longer than it would otherwise. It makes sense to display an alert with this message, because it's important to tell the user why your application is behaving differently than expected and because you're not asking for further input from them. However, it would not be appropriate to use an alert to update users on tasks that are progressing normally or to ask for confirmation of a potentially dangerous action. To provide feedback on the processing of a task, you should use a progress bar or activity indicator (these controls are described in "Progress Bars" (page 86) and "Activity Indicators" (page 79)). To get confirmation for an action, you should use an action sheet.

Using Action Sheets

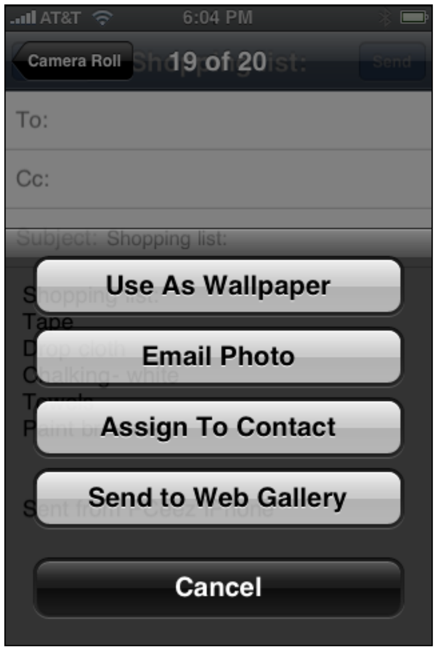
An action sheet usually emerges from the lower edge of the screen and hovers over the application window and its views (as shown in the far left of Figure 6-1 (page 57)). Unlike an alert, however, the side edges of an action sheet are anchored to the sides of the screen, reinforcing its connection to the application and the user's most recent action.

An action sheet does not need to display a message, but typically contains a few buttons that allow the user to choose how to complete their task. When the user taps the selected button, the action sheet disappears.

You might use an action sheet to, for example, warn users that their action will result in the deletion of the message they're viewing. By displaying an action sheet in such a situation, you ensure that the user understands the dangerous effects of the step they're about to take and you can provide some alternatives. This type of communication is particularly important on iPhone OS–based devices because sometimes users tap controls without meaning to. However, imagine that you display a list of messages in your application and that you provide a way to edit the contents of the entire list. In this case, it would not be appropriate to warn the user every time they choose to delete a message, because the destruction of the message is the desired behavior, not a side effect of different action.

It's also appropriate to display an action sheet that contains additional choices in the current context. In Photos, for example, the user can tap the send button when viewing an individual photo. An action sheet appears, giving the user a choice of four destinations for the photo (in addition to the standard Cancel button). This is a good use of an action sheet, because all four choices make sense as potential destinations for a photo, and there is not an appropriate place in the Photos user interface to display all four as separate items. Figure 6-2 shows this action sheet in Photos.

Figure 6-2 An action sheet that displays several choices



Using Modal Views

A modal view slides up from the lower edge of the window and usually covers the entire application window (as shown in the middle of Figure 6-1 (page 57)). Because a modal view hides the current application window, it strengthens the user's perception of entering a different, transient mode in which they can accomplish something.

A modal view can display text if appropriate, and contains the controls necessary to perform the task. In addition, a modal view generally displays a button that completes the task and dismisses the view, and a Cancel button users can tap to abandon the task.

A modal view supports more extensive user interaction than an action sheet. Unlike an action sheet, which accepts a single choice, a modal view supports multistep user interaction, such as the selection of more than one option and the inputting of information.

Use a modal view when you need to offer the ability to accomplish a self-contained task related to your application's primary function. A modal view is especially appropriate for a multistep subtask that requires user interface elements that don't belong in the main application user interface all the time. A good example of a modal view is the compose view in Mail. When users tap the compose button, a modal view appears that contains text areas for the addresses and message, a keyboard for input, a Cancel, and a Send button.

Configuring an Alert

You can't customize the background appearance of an alert, but you can specify the text, the number of buttons, and the button contents.

In most cases, you should include a single button in an alert: an OK button users tap to acknowledge that they read the alert message and to dismiss the alert. You should avoid putting more than two buttons in an alert because an alert displays buttons in a single row, resizing them as necessary. If there are three or more buttons in a single row across the limited width of an alert, they become difficult for users to tap accurately.

The main function of an alert is to tell users about something that happened, perhaps unexpectedly. To do this, you need to display enough text to clearly explain the situation and what, if anything, the user can do about it. Often, a title that succinctly describes the situation or event is sufficient for an alert.

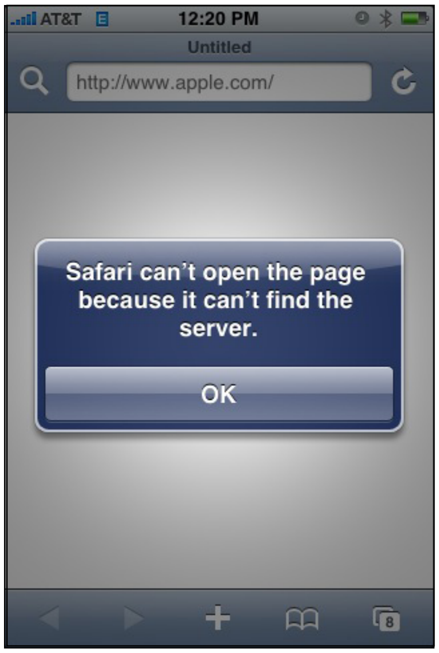
If you display an alert title, you should use title-style capitalization and bold-style font, and you should not put a period after the title. Title-style capitalization is the capitalization of every word except:

- Articles (*a*, *an*, *the*)
- Coordinating conjunctions (*and*, *or*)
- Prepositions of four or fewer letters, except when the preposition is part of a verb phrase, as in "Set Up the Network"

Sometimes, it's a good idea to add to the alert a sentence that provides context or additional information. For example, if an event or situation can be caused by different things, you might choose to provide an alert that always uses the same title to describe the event, but displays different explanatory sentences, depending on the cause. At other times, a title might not add any useful information. When this is the case, you can dispense with the title and instead display a brief sentence that describes the situation. When you display an explanatory sentence, the sentence text should be in regular font and you should use sentence-style capitalization (only capitalize the first word in the sentence).

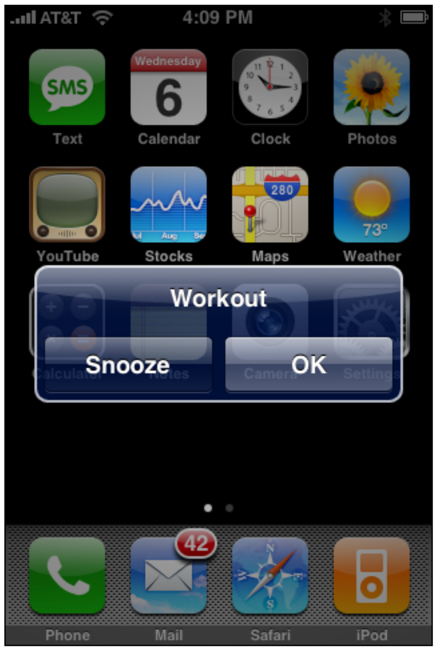
Figure 6-3 shows an alert with one button, no title, and a good explanation of the cause of the condition.

Figure 6-3 An alert should explain the situation and include a button used to dismiss the alert


AT&T
12:20 PM
Untitled
http://www.apple.com/
Safari can't open the page because it can't find the server.
OK

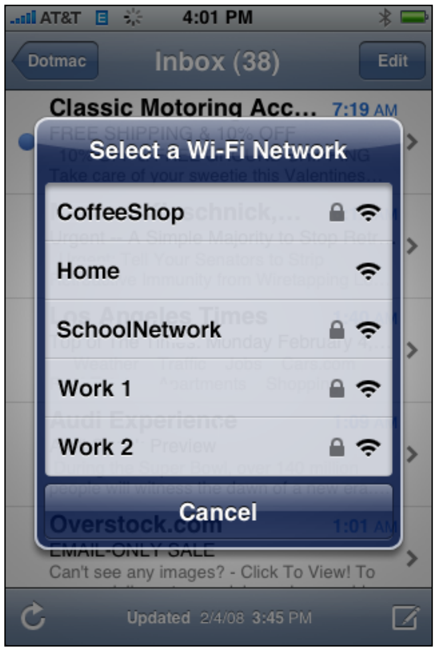
As mentioned above, an alert can display two buttons, although this is unusual; Figure 6-4 shows an alert with two buttons. Because the alert shown in Figure 6-4 comes from the Alarm function of Clock, and is accompanied by the user-designated sound, there's no need for further explanation in its text. Note, however, that your iPhone application cannot display an alert that appears on the Home screen, like the one shown in Figure 6-4, because third-party iPhone applications never run in the background.

Figure 6-4 A two-button alert is unusual



In the rare cases where you need to offer the user a number of choices related to a condition or event, you can use a table view in an alert. In this case, the alert disappears when the user either selects a list item or taps Cancel. In alerts like these, you should display a title, but no explanatory text. This is because the title, which describes the reason for the choices, combined with the list itself provides sufficient context for the user to understand why the alert appeared and what to do about it. Figure 6-5 shows an alert with a table view.

Figure 6-5 In rare cases, an alert can display multiple choices in a table view



Note that alert buttons should contain text, not images. Because images can be misinterpreted, it's safer to use text so you can be sure that users understand what happens when they tap a button. The text you supply for alert buttons should be in bold-style font and use title-style capitalization.

Configuring an Action Sheet

You can specify the background of an action sheet to coordinate with the look of your application, and you can specify the number of buttons and their contents.

Unlike an alert, an action sheet should not need to display a textual message. This is because an action sheet appears as the result of a user action, such as tapping a Delete or Send button, so there should be no need to explain its arrival.

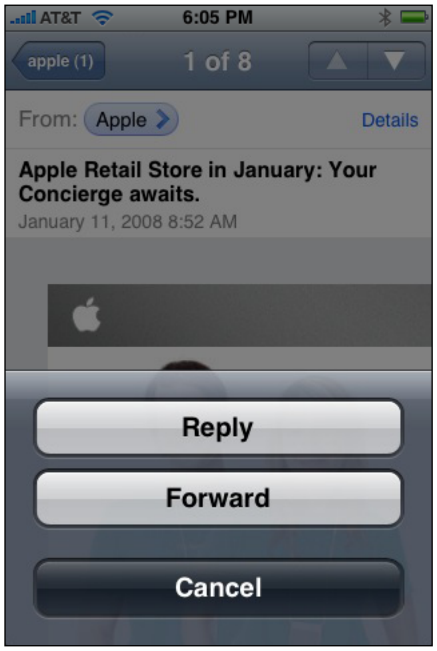
The contents of an action sheet's buttons should be text, not images. This is because it's very important that users understand precisely what their options are, and images can be misinterpreted. Note that the text in action sheet buttons should use bold-style font and title-style capitalization (see "Configuring an Alert" (page 60) for an explanation of this capitalization style).

Action sheets can have a few different background appearances. By default, iPhone OS displays action sheets with a gray background. Typically, you should ensure that the background of an action sheet matches the appearance of your application's toolbar or tab bar. Alternatively, the action sheet background can also be translucent black.

By default, iPhone OS displays the Cancel button at the bottom of an action sheet. This encourages the user to read through all the alternatives before reaching the Cancel option.

Figure 6-6 shows an action sheet with the default background appearance and a Cancel button in the default location.

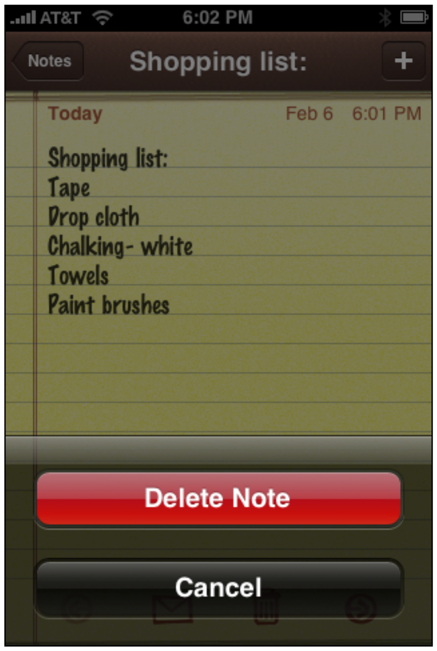
Figure 6-6 A typical action sheet



If you need to provide a button that performs a potentially destructive action, such as deleting all the items in a user's shopping list, you should use the red button color. By default, such destructive buttons are displayed at the top of the action sheet where they are most visible.

Figure 6-7 shows an action sheet with the translucent black background appearance and both a Cancel and a destructive button in their default positions.

Figure 6-7 By default, an action sheet displays at the top a button that performs a destructive action



You can display several buttons in an action sheet, as long as you make sure each button is easily distinguished from the others. Figure 6-8 shows an action sheet with a background that matches the blue toolbar and that provides three alternatives in addition to Cancel.

Figure 6-8 An action sheet with four buttons

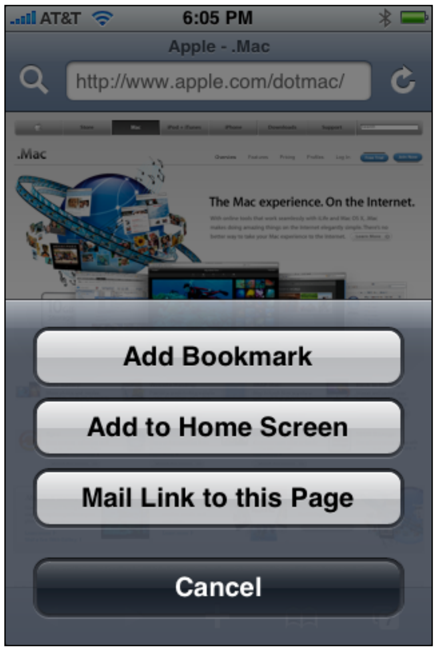
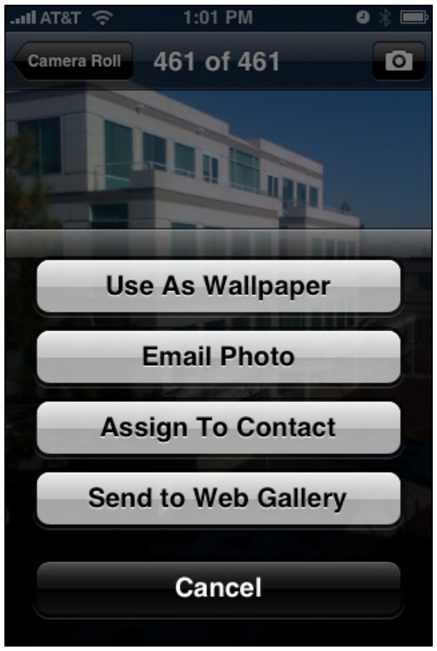


Figure 6-9 shows an action sheet with four buttons, in addition to Cancel. The background appearance of this action sheet matches the translucent gray toolbar.

Figure 6-9 An action sheet with five buttons

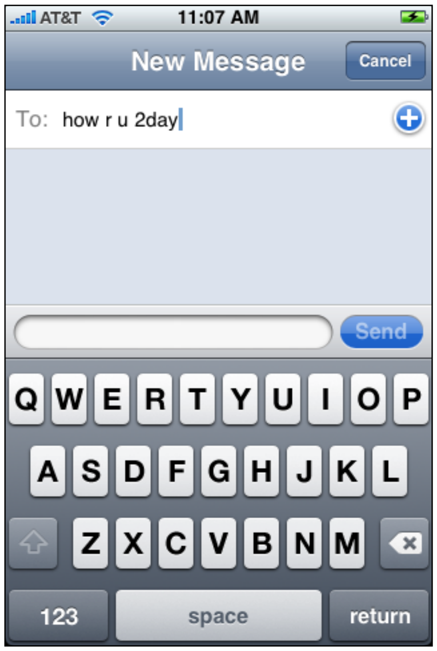


Configuring a Modal View

The overall look of a modal view should coordinate with the application that displays it. For example, a modal view often includes a navigation bar that contains a title and buttons that cancel or complete the modal view's task. The navigation bar should have the same background appearance as the navigation bar in the application.

A modal view should usually display a title that identifies the task in some way. If appropriate, you can also display text in other areas of the view that more fully describes the task or provides some guidance. For example, the Text application provides a modal view when users want to compose a text message. This modal view, shown in Figure 6-10, displays a navigation bar with the same background as the application navigation bar and with the title New Message.

Figure 6-10 A modal view should coordinate with the application window



In a modal view, you can use whichever controls are required to accomplish the task. For example, you can include text fields, buttons, and table views.

Content Views

UIKit defines a few views that display application-specific content within an application window. Three of these—table, text, and web views—inherit from the scroll view, which tracks scrolling gestures, handles zooming and panning, and displays scrolling indicators. This means that in addition to their other features, table, text, and web views can display content that exceeds the size of the application window.

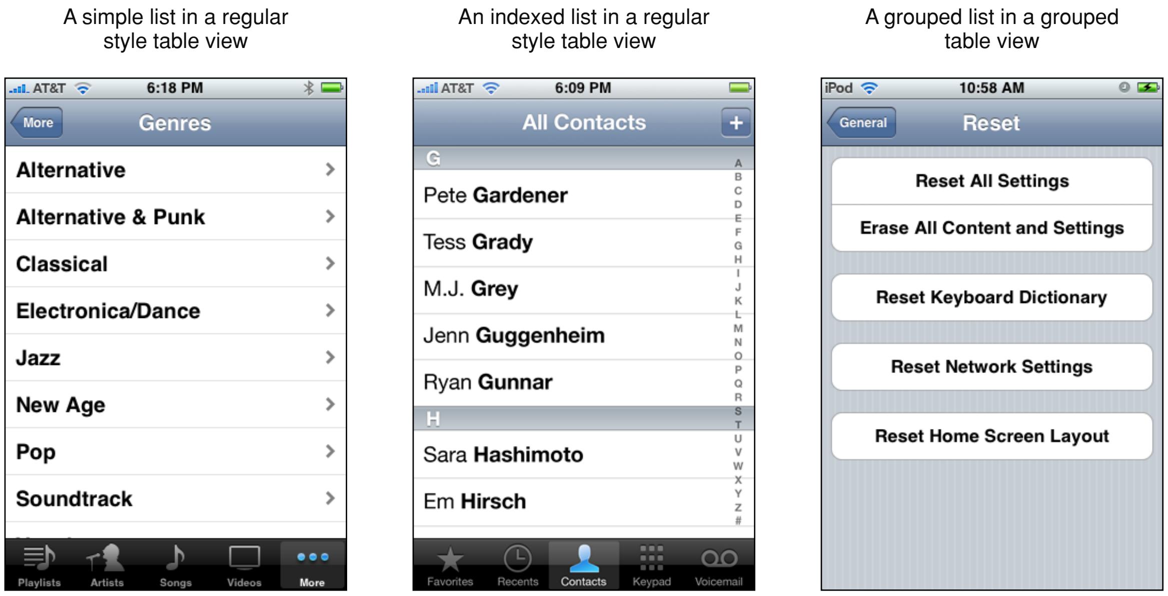
All three types of views are versatile elements that lend themselves to different uses in your iPhone application. For example, table views can be configured to display short lists of choices, indexed lists of items, and grouped lists of detailed information. Text views and web views are relatively unconstrained containers you can use to accept and display content. This chapter provides guidance for using these elements in your iPhone application.

Table Views

A **table view** presents data in a single-column list of multiple rows. Each row can contain some combination of text, images, and controls, and rows can be divided into sections or groups. Users flick or drag to scroll through large numbers of rows or groups of rows. Figure 7-1 shows how different styles of table views can display lists in different ways.

2008-02-29 | © 2008 Apple Inc. All Rights Reserved.

Figure 7-1 Three ways to display lists



This section covers what table views support and how to use them in your application. Then, it describes two styles of table view: the regular style (two variations of which are shown in on the left and in the middle in Figure 7-1) and the grouped style (shown on the right in Figure 7-1).

Usage and Behavior

Table views are extremely useful in iPhone applications because they provide attractive ways to organize both large and small amounts of information. Table views are most useful in productivity applications that tend to handle lots of user items, although utility applications can make use of smaller-scale table views, as well. An immersive application would probably not use a table view to display information, but it might use one to display a short list of options.

Table views provide built-in elements that allow users to navigate and manipulate information. Table views also provide feedback when users select list items: When an item can be selected, the row highlights briefly to show users that their action has been received. In addition, table views support:

- The display of header and footer information. You can display descriptive text above or below each section or group in a list, and above or below the list as a whole.
- List editing. You can allow users to add, remove, and reorder list items in a consistent way. Table views also support the selection and manipulation of multiple list items, which you might use to give users a convenient way to delete more than one list item at a time.

Note that you can choose to animate the changes users make to list items. Doing so is a good way to provide feedback and strengthen the user's sense of direct manipulation. In Settings, for example, when you turn off the automatic date and time setting (be selecting Off in Date & Time > Set Automatically), the list group expands smoothly to display two new items, Time Zone and Set Date & Time.

The two styles of table views are distinguished mainly by appearance:

- **Regular**. This style of table view displays rows that stretch from side edge to side edge of the screen. The background of the rows is white. The rows can be separated into labeled sections and the table view can display an optional index view that appears vertically along the right edge of the view.
- **Grouped**. This style of table view displays groups of rows that are inset from the side edges of the screen. The groups are displayed on a distinctive blue-striped background, while inside the groups the background is white. A grouped table view can contain an arbitrary number of groups, and each group can contain various numbers of rows. Each group can be preceded by header text and followed by footer text. This table style does not provide an index view.

Table views are particularly versatile user interface elements, because they can be configured in different ways to support different user actions. For example, you can use table views to support the following common actions:

- Selecting options

  iPhone OS does not include multi-item selection controls analogous to menus or pop-up menus, but a table view works well to display a list of options from which the user can choose. This is because table views display items in a simple, uncluttered way. In addition, the table view provides a checkmark image that shows users the currently selected option in a list.

  If you need to display choices related to an item in a list, you can use either style of table view. But if you need to display a list of choices users see when they tap a button or other user interface element, use the regular style.

- Navigating hierarchical information

  A table view works well to display a hierarchy of information in which each node (that is, list item) can contain its own subset of information, because each subset can be displayed in a separate list. This makes it easy for users to follow a path through the hierarchy by selecting one item in each successive list. The table view provides the disclosure indicator element that reveals the subset of information belonging to an item, and the detail disclosure button element that displays detailed information about an item.

  Although you can configure either style of table view for this usage, it’s best to use the regular style if there are three or more levels in the information hierarchy.

- Looking up indexed information

  You can use a table view to display information ordered according to a scheme, for example, alphabetization. You can provide header (and, optionally, footer) information to label separate sections, such as the “A” section and the “B” section. You can also choose to display the index view, which shows the entire range of list content. Users can scroll through the rows of the sectioned list or pinpoint a location in the index view.

  Only the regular table view provides the index view, but both styles support categorization of list items.

- Viewing conceptually grouped information.

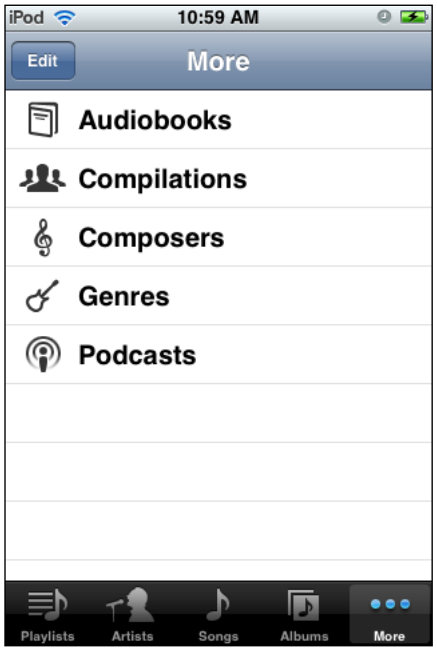
  You can use a table view to cluster information into logical groups, such as work, home, or school.

  Although you can define logical sections in a regular table view, it’s usually better to use a grouped table view because it provides a better visual indication of grouping and more space for contextual information in headers and footers.

Configuring a Regular Table View

A regular table view, in its simplest configuration, displays only the list items, with no section divisions and no index view. The list items can include text, images, and table-view elements, such as the disclosure indicator (for more information about the elements a table view provides, see "Table-View Elements" (page 74)). For example, Figure 7-2 shows an example of a simple list in a regular table view.

Figure 7-2 A simple list in a regular style table view



The disclosure indicator element (shown in the left-hand list in Figure 7-1 (page 68)) is necessary if you're using the table to present hierarchical information. This is because users know that this element means "show the next page of information." It should appear to the right of every list item that contains a sublist of items.

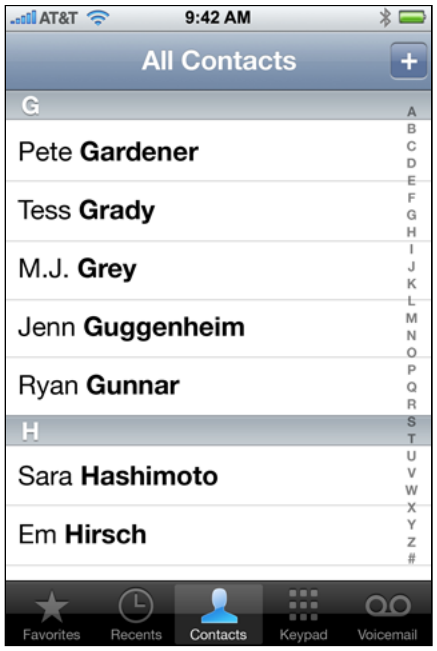
If you're configuring a regular table view as a selection mechanism, however, you should not need to display the disclosure indicator element, because a selection list should seldom be hierarchical. However, you should display the checkmark element next to the option the user selects (for more information about this element, see "Table-View Elements" (page 74)). This way, you inform the user which selection is currently in effect.

As mentioned in "Usage and Behavior" (page 68), a table view automatically highlights a row in response to the user's tap, when the row item supports selection. This brief highlight is a feedback mechanism that allows users to see that they've tapped the row they intended and lets them know their selection is being processed. You should never make this highlight permanent in an effort to indicate the current selection in a list; instead, use the checkmark.

If you want to display a list of items divided into sections, you can configure a regular table view to display section headers, with each header describing a section and providing visual distinction from the other sections. You typically want to do this when the number of items in the list is long, and when the items lend themselves to a familiar categorization scheme. For example, the contact list in

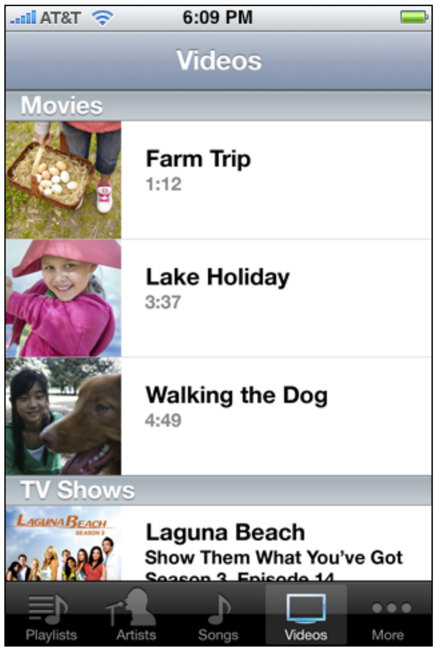
Phone can be very long and people are accustomed to organizing such information alphabetically. Figure 7-3 shows an alphabetical list of names in a contacts list that includes section headers and an index view along the right edge.

Figure 7-3 An indexed list in a regular style table view



In general, you should consider displaying the index view if the items in the list fill more than about two screenfuls. If the list content is not that extensive, you can configure a regular table to display information with section headers, but without the index view, as shown in Figure 7-4. It's also possible to specify footers for each section, but this is not shown in Figure 7-4.

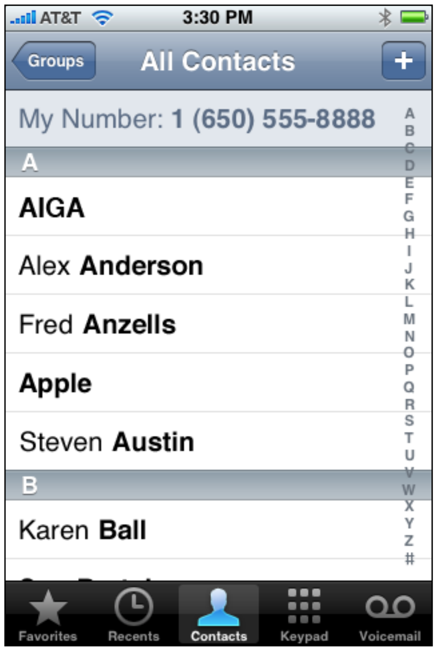
Figure 7-4 A sectioned list without an index view in a regular style table view



Avoid using table elements that display on the right edge of a table (such as the disclosure indicator) in a regular table with an index view, because these elements interfere with the index view.

If it makes sense in your application, you can display a header at the beginning of the table view, before the first item. Figure 7-5 shows an example of this. You can also display a footer after the last item, but this is not shown in Figure 7-5.

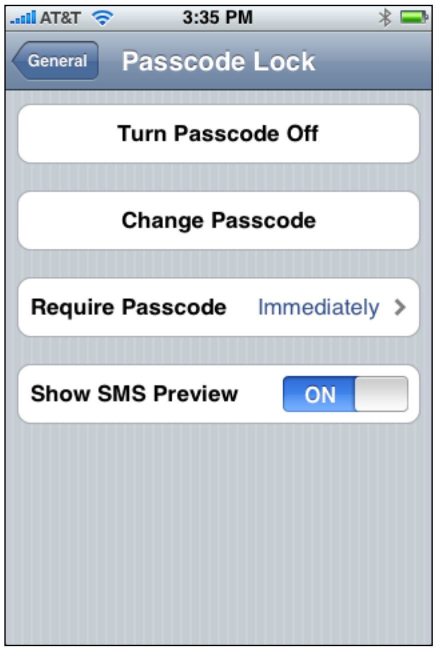
Figure 7-5 A regular style table view can display a table header before the first list item



Configuring a Grouped Table View

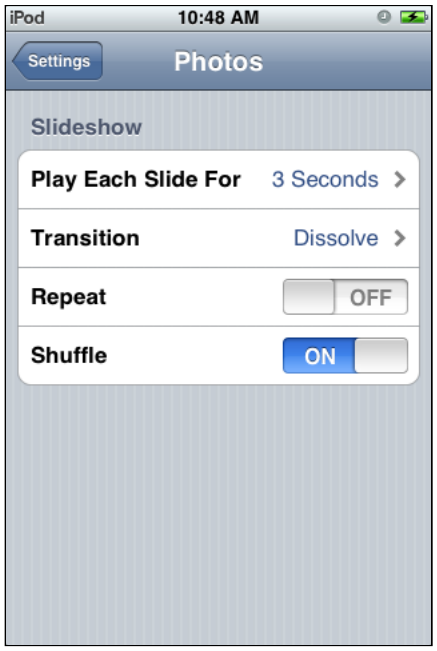
A **grouped table view** always contains at least one group of list items (one per row), and each group always contains at least one item. A list item can include text, images, and table-view elements (described in "Table-View Elements" (page 74)). For example, Figure 7-6 shows a grouped table view with four groups, each containing one list item.

Figure 7-6 A grouped table view with four groups



In contrast, Figure 7-7 shows a grouped table view with a single group that contains four list items.

Figure 7-7 A grouped table with a single group



Slideshow

Play Each Slide For	3 Seconds
Transition	Dissolve
Repeat	OFF
Shuffle	ON

Notice the different types of elements in the list items shown in Figure 7-6 (page 73) and in Figure 7-7. Both lists include the disclosure indicator element, which users tap to see the next list related to an item, and both lists include a switch control, which is a table-view control users slide to make an on/off choice.

Table-View Elements

In general, users can tap anywhere on a list item to select it. However, table views also provide a few elements that are designed to display more information about list items, or to delete them. These **table-view elements** are used consistently in iPhone applications, and users are accustomed to their appearance and meanings. Specifically, a table view provides the following elements:

- **Disclosure indicator**. Users tap this element to see the next level in a hierarchy, or the choices associated with the list item. (See Figure 7-7 (page 74) for an example of this element.)

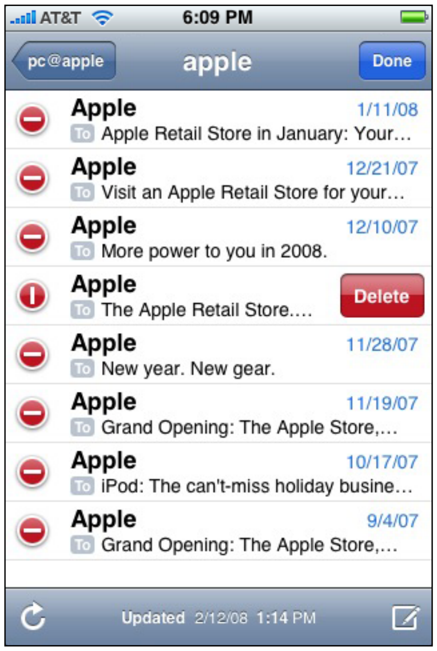
  Use a disclosure indicator in a row when selecting the row results in the display of another list. In other words, the presence of a disclosure indicator tells users that tapping the row reveals another list; it does not offer functionality that is separate from the selection of the row.

- **Detail disclosure button**. Users tap this element to see detailed information about the list item. (Note that you can use this element outside of a table view, to reveal additional details about something; see "Detail Disclosure Buttons" (page 82) for more information.)

  In a table view, use a detail disclosure button in a row to display details about the list item. Note that the detail disclosure button, unlike the disclosure indicator, can perform an action that is separate from the selection of the row. For example, in Phone Favorites, selecting the row initiates a call to the contact; tapping the detail disclosure button in the row reveals more information about the contact.

- **Delete button**. Users tap this element to delete the list item. This element appears to the right of a list item when users swipe in the row or when they tap the red minus button while in an editing context. (See Figure 7-8 for an example of this element.)
- **Red minus button**. Users tap this element to reveal and hide the Delete button for each list item. To give additional feedback to users, the horizontal minus symbol inside this button becomes vertical when users tap it to reveal the Delete button. This element appears to the left of a list item. (See Figure 7-8 for an example of this element.)
- **Green plus button**. Users tap this element to add something to the list.
- **Checkmark**. This element appears to the right of a list item to show that it is currently selected.

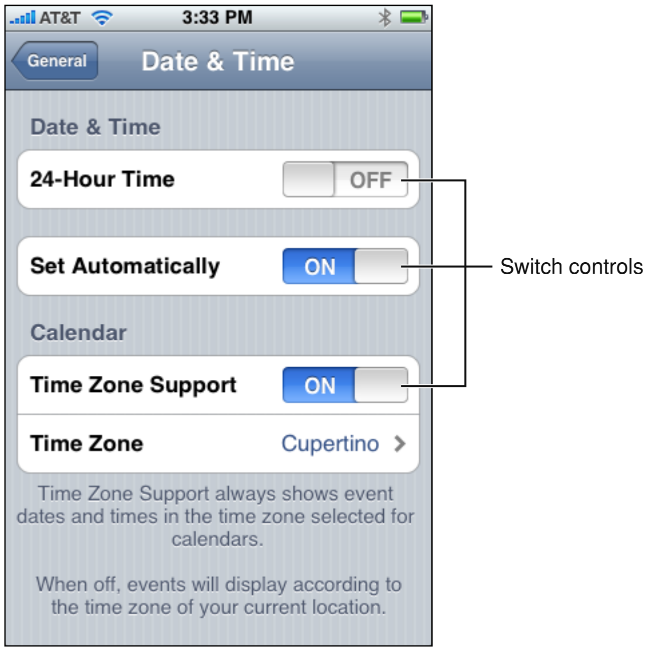
Figure 7-8 A table view can display the Delete button and the red minus button


AT&T
6:09 PM
pc@apple
apple
Done
Apple
1/11/08
To Apple Retail Store in January: Your…
Apple
12/21/07
To Visit an Apple Retail Store for your…
Apple
12/10/07
To More power to you in 2008.
Apple
Delete
To The Apple Retail Store.…
Apple
11/28/07
To New year. New gear.
Apple
11/19/07
To Grand Opening: The Apple Store,…
Apple
10/17/07
To iPod: The can't-miss holiday busine…
Apple
9/4/07
To Grand Opening: The Apple Store,…
Updated 2/12/08 1:14 PM

Switch Controls

A **switch control** presents to the user two mutually exclusive choices or states, such as yes/no or on/off. A switch control shows only one of the two possible choices at a time; users slide the control to reveal the hidden choice or state. Figure 7-9 shows examples of switch controls.

Figure 7-9 Switch controls in a table view


AT&T
3:33 PM
General
Date & Time
Date & Time
24-Hour Time
OFF
Set Automatically
ON
Calendar
Time Zone Support
ON
Time Zone
Cupertino
Time Zone Support always shows event dates and times in the time zone selected for calendars.
When off, events will display according to the time zone of your current location.
Switch controls

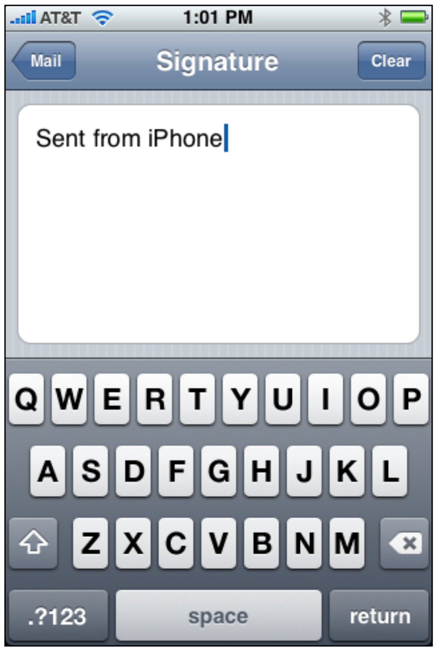
Use a switch control in a grouped table view when you need to offer the user two simple, diametrically opposed choices. Because one choice is always hidden, it's best to use a switch control when the user already knows what both values are. In other words, don't make the user slide the switch control just to find out what the other option is.

You can use a switch control to change the state of other user interface elements in the view. Depending on the choice users make, new list items might appear or disappear, or list items might become active or inactive.

Text Views

A **text view** is a region that displays multiple lines of text. Recall that a text view inherits from a scroll view, which means that the text view supports scrolling when the content is too large to fit inside its bounds. Mail uses a text view to allow users to create a signature that appears at the end of each email message they compose, as shown in Figure 7-10.

Figure 7-10 A text view displays multiple lines of text



Although you might use a text view to display many lines of text, such as the content of a large text document, you can also use a text view to support user editing. If you make a text view editable, a keyboard appears when the user taps inside the text view.

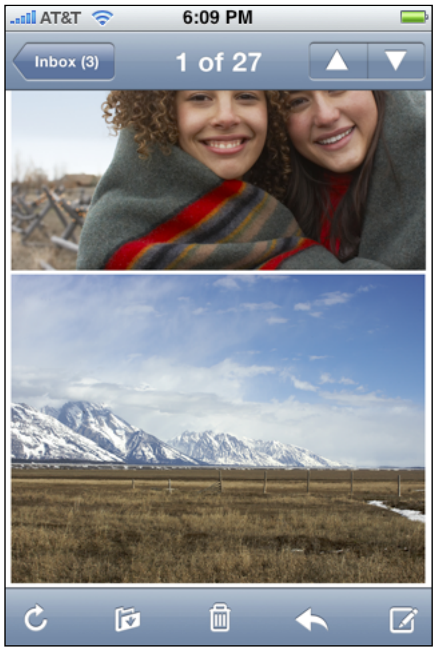
You have control over the font, color, and alignment of the text in a text view, but only as they apply to the entirety of the text. In other words, you can't change any of these properties for only part of the text. The defaults for the font and color properties are, as you would expect, the system font and black, because they tend to be the most readable. The default for the alignment property is left (you can change this to center or right).

If you must enable variable fonts, colors, or alignments within a view that displays text, you can use a web view instead of a text view, and style the text using HTML.

Web Views

A **web view** is a region that can display rich, HTML content in your application window. For example, Mail uses a web view to display message content, because it can contain elements in addition to plain text (Figure 7-11 shows an example of this).

Figure 7-11 A web view can display web-based content



In addition to displaying web content, a web view provides elements that support navigation through open webpages. Although you can choose to provide webpage navigation functionality, it's best to avoid creating an application that looks and behaves like a mini web browser.

If you have a webpage or web application, you might choose to use a web view to implement a simple iPhone application that provides a wrapper for it. If you plan to access web content that you control, be sure to read *Safari Web Content Guide for iPhone* to learn how to create web content that is compatible with and optimized for display on iPhone OS–based devices.

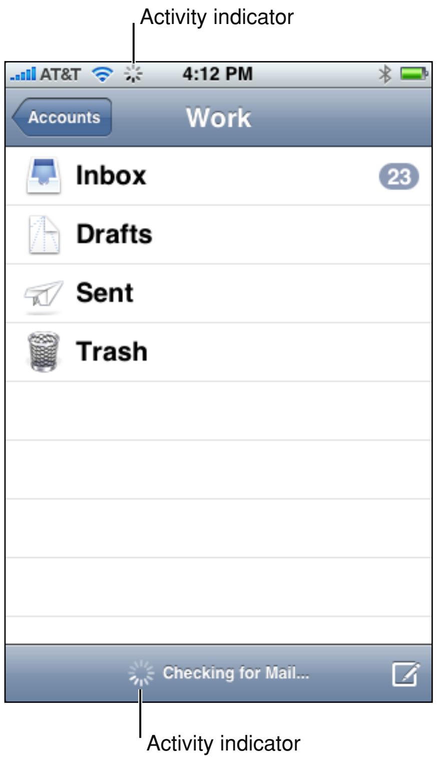
Application Controls

iPhone OS provides several controls you can use in your application, most of which are already familiar to the users of iPhone OS–based devices. For the most part, these controls can be used in any content view or in the body of the window. As long as you don't repurpose them to mean something entirely different, users will understand how to use them in your application.

Activity Indicators

An **activity indicator** shows the progress of a task or process that is of unknown duration. If you need to display progress for a task of known duration, use a progress bar instead (see "Progress Bars" (page 86) for more information about this control). The "spinning gear" appearance of the activity indicator shows users that processing is occurring, but does not suggest when it will finish. Figure 8-1 shows two uses of an activity indicator.

Figure 8-1 Two activity indicators



An activity indicator is a good feedback mechanism to use when it's more important to reassure users that their task or process has not stalled than it is to suggest when processing will finish.

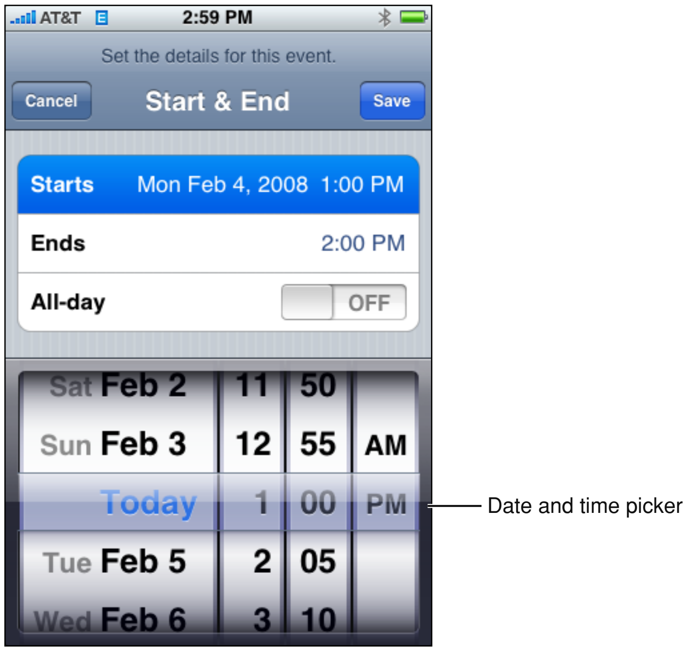
You can choose the size and color of an activity indicator to coordinate with the background of the view in which it appears. By default, an activity indicator is white.

An activity indicator disappears when the task or process has completed. This default behavior is recommended, because users expect to see an activity indicator when something is happening and they associate a stationary activity indicator with a stalled process.

Date and Time Pickers

A **date and time picker** gives users an easy way to select a specific date or time. A date and time picker can have up to four independent spinning wheels, each of which displays values in a single category, such as month or hour. Users flick or drag to spin each wheel until it displays the desired value beneath the clear selection bar that stretches across the middle of the picker. The final value comprises the values displayed in each wheel. Figure 8-2 shows an example of a date and time picker.

Figure 8-2 A date and time picker



Use a date and time picker to allow users to avoid typing values that consist of multiple parts, such as the day, month, and year of a date. A date and time picker works well because the values in each part have a relatively small range and users already know what the values are.

Depending on the mode you specify, a date and time picker displays a different number of wheels, each with a set of different values. The date and time picker defines the following modes:

- **Time**. The time mode displays wheels for the hour and minute values, with the optional addition of a wheel for the AM/PM designation.
- **Date**. The date mode displays wheels for the month, day, and year values.
- **Date and time**. The date and time mode displays wheels for the calendar date, hour, and minute values, with the optional addition of a wheel for the AM/PM designation. This is the default mode.
- **Countdown timer**. The countdown timer mode displays wheels for the hour and minute. You can specify the total duration of a countdown, up to a maximum of 23 hours and 59 minutes.

By default, a minutes wheel displays 60 values (0 to 59). However, if you need to display a coarser granularity of choices, you can set a minutes wheel to display intervals of minutes, as long as the interval divides evenly into 60. For example, you might want to display the quarter-hour intervals 0, 15, 30, and 45.

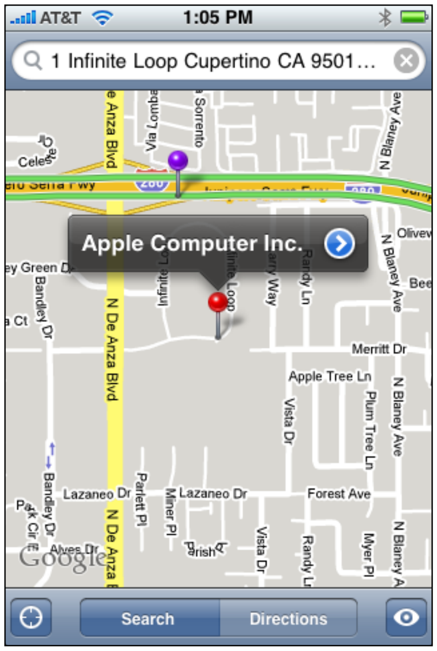
Regardless of its configuration, the overall size of a date and time picker is fixed, and is the same size as the keyboard. You might choose to make a date and time picker a focal element in your view, or cause it to appear only when needed. For example, the timer mode of the built-in Clock application displays an always-visible date and time picker because the selection of a time is central to the function of the Timer. On the other hand, the Set Date & Time preference (available in Settings > General > Date & Time, when you turn off Set Automatically) displays transient date and time pickers, depending on whether users want to set the date or the time.

Detail Disclosure Buttons

A **detail disclosure button** reveals additional or more detailed information about something. Usually, you use detail disclosure buttons in table views, where they give users a way to see detailed information about a list item (for more information about this usage, see "Table-View Elements" (page 74)). However, you can use this element in other types of views to provide a way to reveal more information or functionality.

For example, the Maps application displays a detail disclosure button users can tap to access more functionality related to the dropped pin. Figure 8-3 shows an example of a detail disclosure button.

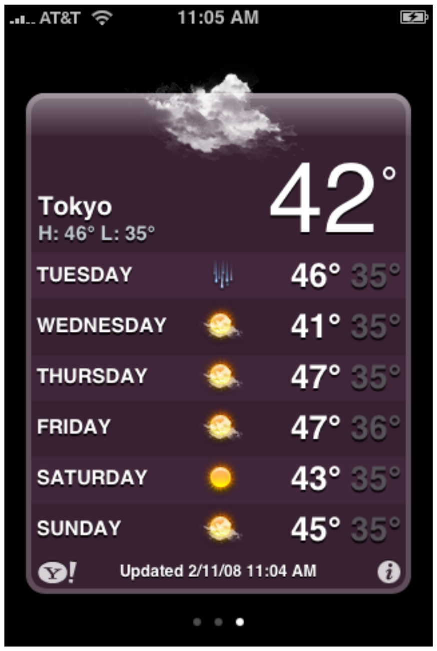
Figure 8-3 A detail disclosure button reveals additional details or functionality



Info Buttons

An **Info button** provides a way to reveal configuration details about an application, often on the back side of the window. For this reason, Info buttons are especially well suited to utility applications. You can see an example of an Info button in the lower-right corner of the Weather application (shown in Figure 8-4).

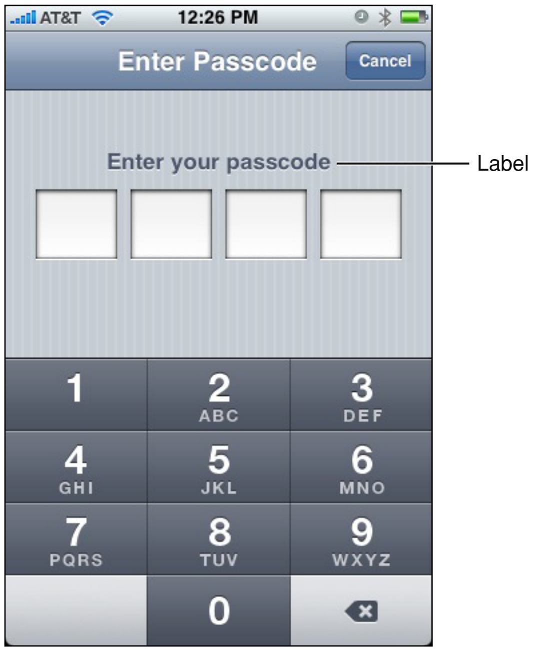
Figure 8-4 An Info button reveals information, often configuration details



Labels

A **label** is a variably sized amount of static text. Figure 8-5 shows an example of a label.

Figure 8-5 A label gives users information



You can use a label to name parts of your user interface or to provide limited help to the user. A label is best suited to display a relatively small amount of text.

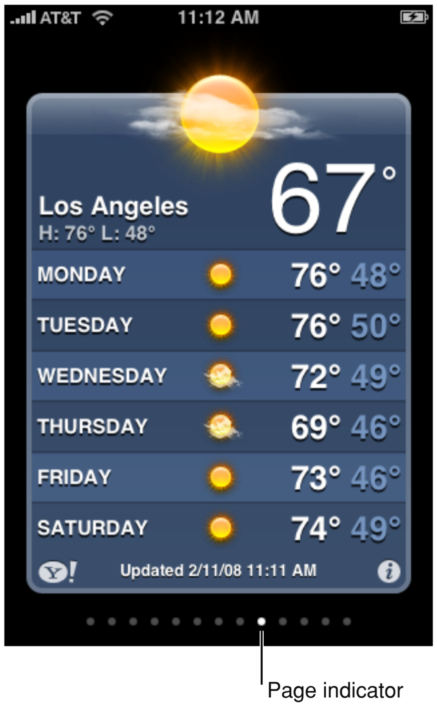
You can determine various properties of the label's text, such as font, text color, and alignment, but above all, you should take care to make your labels legible. Don't sacrifice clarity for fancy fonts or showy colors.

As you compose the text of your labels, be sure to use the user's vocabulary. Examine the text in your application for developer-centric terms and replace them with user-centric terms.

Page Indicators

A **page indicator** displays a dot for each currently open view in an application. From left to right, the dots represent the order in which the views were opened (the leftmost dot represents the first view). The currently visible view is indicated by a glow on the dot that represents it. Users tap to the left or the right of the glowing dot to view the previous or next open view. Figure 8-6 shows an example of a page indicator.

Figure 8-6 A page indicator



A page indicator gives users a quick way to see how many views are open and an indication of the order in which they were opened; it does not help users keep track of the steps they took through a hierarchy of views. Because the views in a utility application tend to be peers of each other, a page indicator is sufficient to help users navigate through them. A productivity application that displays hierarchical information, on the other hand, should offer navigation through the elements in the navigation bar (for more on this, see "Navigation Bars" (page 52)).

Typically, page indicators work well near the bottom edge of the application window, below the views they represent. This leaves the more important information (the view itself) in the upper part of the window where users can see it easily. Be sure to vertically center a page indicator between the lower edge of the view and the lower edge of the window.

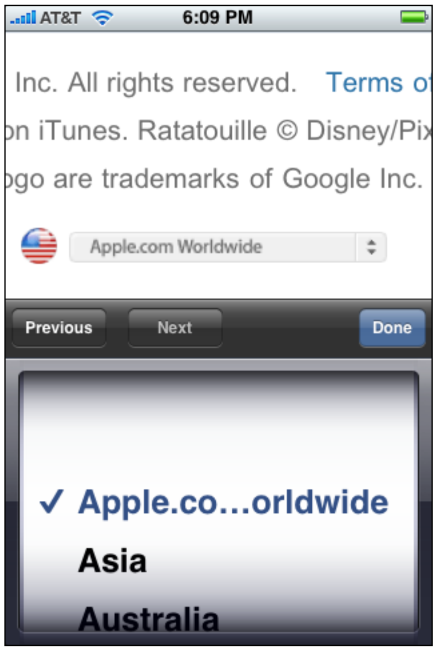
Although there is no programmatic limit to the number of dots you can display in a page indicator, be aware that the dots do not shrink or squeeze together as more appear. For example, in portrait orientation, you can display at most 20 dots in a page indicator before clipping occurs. Therefore, you should provide logic in your application to avoid this situation.

Although you can hide a page indicator when there is only one open view, the default behavior is to display it.

Pickers

A **picker** is a generic version of the date and time picker (see "Date and Time Pickers" (page 80) for more information about this control). You can use a picker to display any set of values. As with a date and time picker, users spin the wheel (or wheels) of a picker until the desired value appears. Figure 8-7 shows how this might look.

Figure 8-7 A picker as displayed in Safari on iPhone


AT&T
6:09 PM
Inc. All rights reserved. Terms o
n iTunes. Ratatouille © Disney/Pi
go are trademarks of Google Inc.
Apple.com Worldwide
Previous
Next
Done
✓ Apple.co...orldwide
Asia
Australia

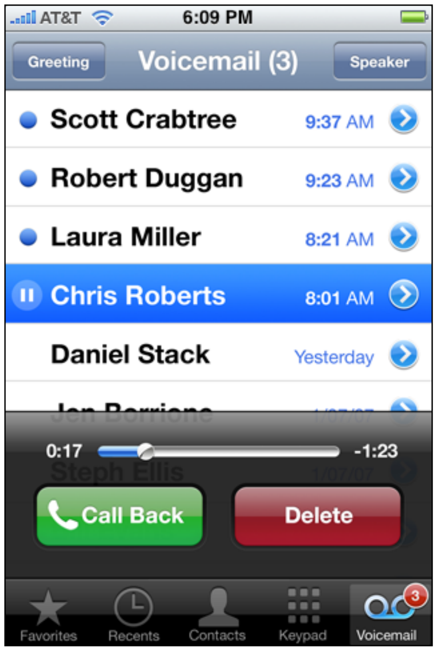
As you decide whether to use a picker in your application, consider that many, if not most, of the values in a wheel are hidden from the user when the wheel is stationary. This is not necessarily a problem, especially if users already know what those values are. For example, in a date and time picker, users understand that the hidden values in the month wheel can only be numbers between 1 and 12. If you need to provide choices that aren't members of such a well-known set, however, a picker might not be the appropriate control.

As with a date and time picker, a generic picker can be visible all the time (as a focal point of your user interface) or it can appear only when needed. The overall size of a picker is fixed, and is the same size as a keyboard.

Progress Bars

A **progress bar** shows the progress of a task or process that has a known duration. If you need to display progress for a task of unknown duration, use an activity indicator instead (see "Activity Indicators" (page 79) for more information about this control). As the task or process proceeds, the progress bar is filled from left to right; at any given time, the proportion of filled to unfilled area in the bar gives the user an indication of how soon the task or process will finish. Figure 8-8 shows an example of a progress bar.

Figure 8-8 A progress bar



A progress bar is a good way to provide feedback to users on tasks that have a well-defined duration, especially when it's important to show users approximately how long the task will take. When you display a progress bar, you tell the user that their task is being performed and you give them enough information to decide if they want to wait until the task is complete or cancel it.

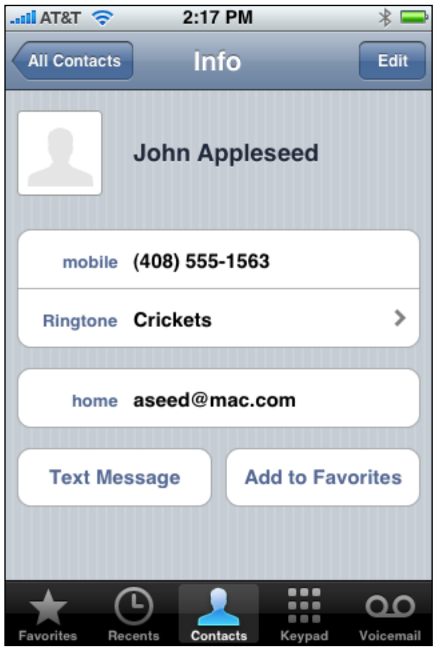
The height of a progress bar is fixed, but you can adjust the width to fit your layout. You can also choose the color of a progress bar to coordinate with the background of the view in which it appears. Specifically, a progress bar can be:

- **White**. This style looks good on dark backgrounds. (White is the default color of a progress bar.)
- **Gray**. This style looks good on light backgrounds.

Rounded Rectangle Buttons

A **rounded rectangle button** is a versatile button you can use in a view to perform an action. You can see examples of this type of button at the bottom of an individual's Contacts view: The Text Message and Add to Favorites buttons are rounded rectangle buttons, as shown in Figure 8-9.

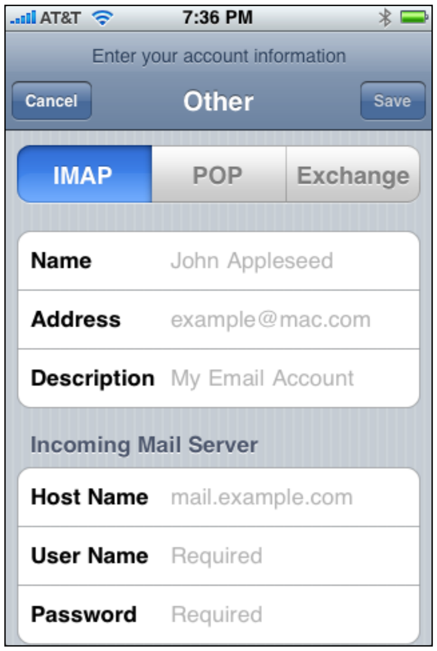
Figure 8-9 Rounded rectangle buttons perform application-specific actions



Segmented Controls

A **segmented control** is a linear set of segments, each of which functions as a button. Collectively, the segments in a segmented control offer closely related, but mutually exclusive choices. When users tap a segment in a segmented control, an instantaneous action or visible result should occur. For example, Settings displays different information when users use the segmented control to select an email protocol, as shown in Figure 8-10.

Figure 8-10 A segmented control



The length of a segmented control is determined by the number of segments you display and by the size of the largest segment. The height of a segmented control is fixed. Although you can display an arbitrary number of segments, be aware that users must be able to comfortably tap a segment without worrying about tapping a neighboring segment. Because hit regions should be 44 x 44 points, it's recommended that a segmented control have no more than five segments.

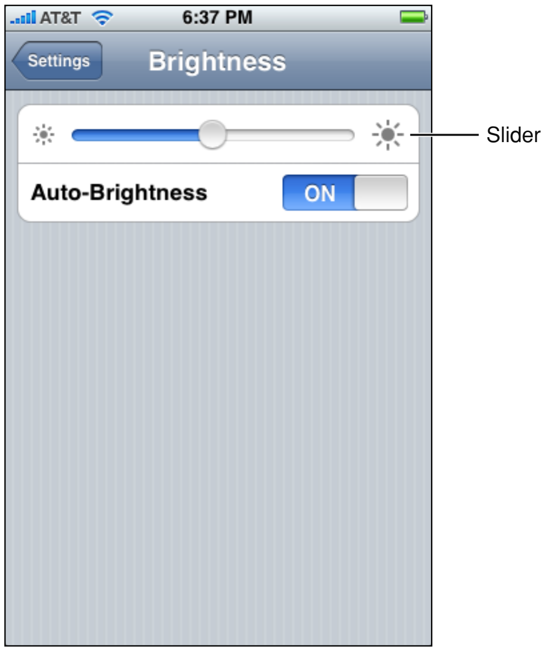
A segmented control can contain text or images; an individual segment can contain either text or an image, but not both. In general, it's usually best to avoid mixing text and images in a single segmented control.

A segmented control ensures that the width of each segment is proportional, based on the total number of segments. This means that you need to ensure that the content you design for each segment is roughly equal in size.

Sliders

A **slider** allows users to make adjustments to a value or process throughout a range of allowed values. When users drag a slider, the value or process is updated continuously. Figure 8-11 shows an example of a slider with minimum and maximum images.

Figure 8-11 A slider

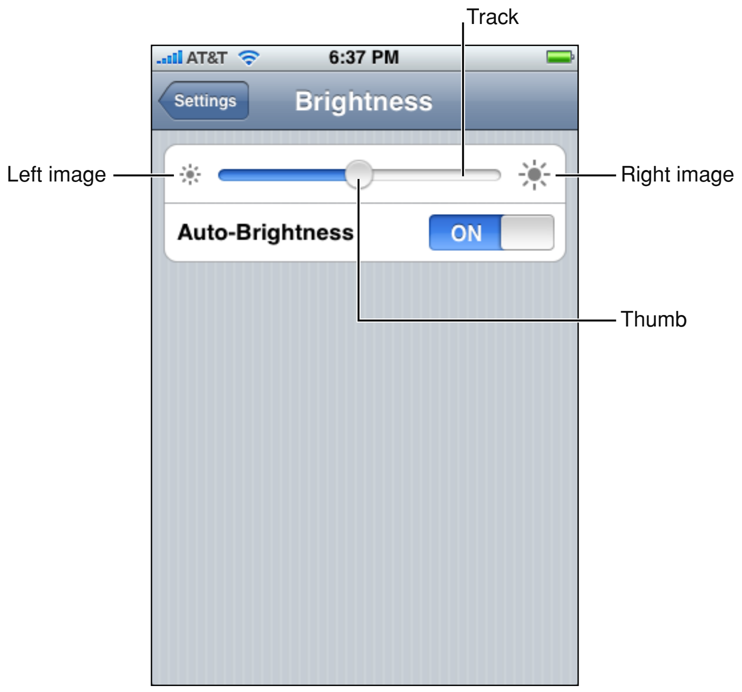


Sliders are useful in two main situations:

- When you want to allow users to have fine-grained control over the values they choose
- When you want to allow users to have fine-grained control over the current process

A slider consists of a track, a thumb, and optional right and left value images. Figure 8-12 shows these parts of a slider.

Figure 8-12 Four parts of a slider



You can set the width of a slider to fit in with the user interface of your application. In addition, you can display a slider either horizontally or vertically.

There are several ways to customize a slider:

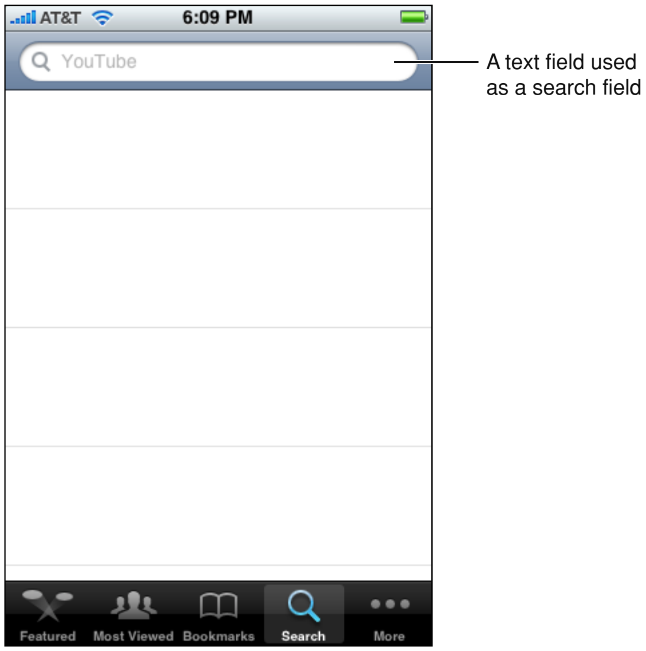
- You can define the appearance of the thumb, so users can see at a glance whether the slider is active.
- You can supply images to appear at either end of the slider (typically, these correspond to minimum and maximum values), to help users understand what the slider does.

  A slider that controls font size, for example, could display a very small character at the minimum end and a very large character at the maximum end.
- You can define a different appearance for the track, depending on which side of the thumb it is on and which state the control is in.

Text Fields

A **text field** is a rounded rectangular field that accepts user input. When the user taps a text field a keyboard appears; when the user taps Return in the keyboard, the text field handles the input in an application-specific way. A text field can contain a single line of input. Figure 8-13 shows an example of a text field used to implement a search field.

Figure 8-13 A text field can be used to implement a search field



As you can see in Figure 8-13, a text field supports the display of images that allow you to use it as a search field. For example, you might display a magnifying glass image in the left end of the text field to indicate that users can type in search terms. In general, you should use the left end of a text field to indicate its purpose and the right end to indicate the presence of additional features, such as bookmarks.

You can also cause the clear button to appear in the right end of a text field. When this element is present, tapping it clears the contents of the text field, regardless of any other image you might display over it.

Icons and Images

The user interfaces of iPhone applications are characterized by beautiful images and lush color. As an application designer, you want to fit into this environment by providing an aesthetically pleasing user interface. Although iPhone OS provides a wide range of elegant and attractive user interface elements, there are two custom elements every application needs: an application icon and a launch image. This chapter describes how to design these elements.

Application Icons

An **application icon** is an icon users put on their Home screens and tap to start an application. This is a place where branding and strong visual design should come together into an compact, instantly recognizable, attractive package.

Users can display as many application icons on their Home screens as they want, so you should design an icon that is:

- Attractive, so users want to keep it on their Home screens
- Distinctive, so users can easily find it among all other icons

When a user decides to display your icon on the Home screen, iPhone OS automatically adds some visual effects so that it coordinates with the built-in icons. Specifically, iPhone OS adds:

- Rounded corners
- Drop shadow
- Reflective shine

For example, Figure 9-1 shows a simple icon as it might be provided by an application.

Figure 9-1 A simple application icon before it is displayed on a Home screen

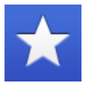
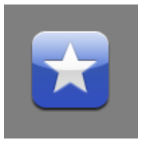


Figure 9-2 shows the same icon as it is displayed on a Home screen by iPhone OS.

Figure 9-2 A simple application icon displayed on a Home screen



To ensure that your icon can take advantage of these visual enhancements, provide an image in PNG format that:

- Measures 57 x 57 points, with 90 degree corners (if the image measures other than this size, iPhone OS scales it)
- Does not have any shine or gloss

Note: Although the easiest way to provide an application icon is to follow the guidelines above, you can also also supply an application icon that already includes the rounded-corner, shadow, and reflection effects. If you choose to do this, you need to add the `UIPrerenderedIcon` key to your application's `Info.plist` file.

As with other user interface elements in iPhone OS, icons that use bold shapes and lines and pleasing color combinations work best. It's advisable to spend some time simplifying your icon design so it clearly conveys the essence of your web content. Also, it's a good idea to investigate how your choice of image and color might be interpreted by people from different cultures.

Launch Images

To enhance the user's experience at application launch, you should provide a launch image. A **launch image** looks very similar to the first window your application displays. iPhone OS displays this image instantly when the user taps your application icon on the Home screen. As soon as it's ready for use, your application displays its first screen, replacing the launch placeholder image.

It's important to emphasize that the reason to supply a launch image is to improve user experience; it is *not* an opportunity to provide:

- An "application entry experience," such as a splash screen
- An About window
- Branding elements, unless they are a static part of your application's first screen

Because users are likely to switch among applications frequently and quickly, you should make every effort to cut launch time to a minimum, and you should design a launch image that downplays the experience rather than drawing attention to it.

To do this, you should design an image in PNG format that:

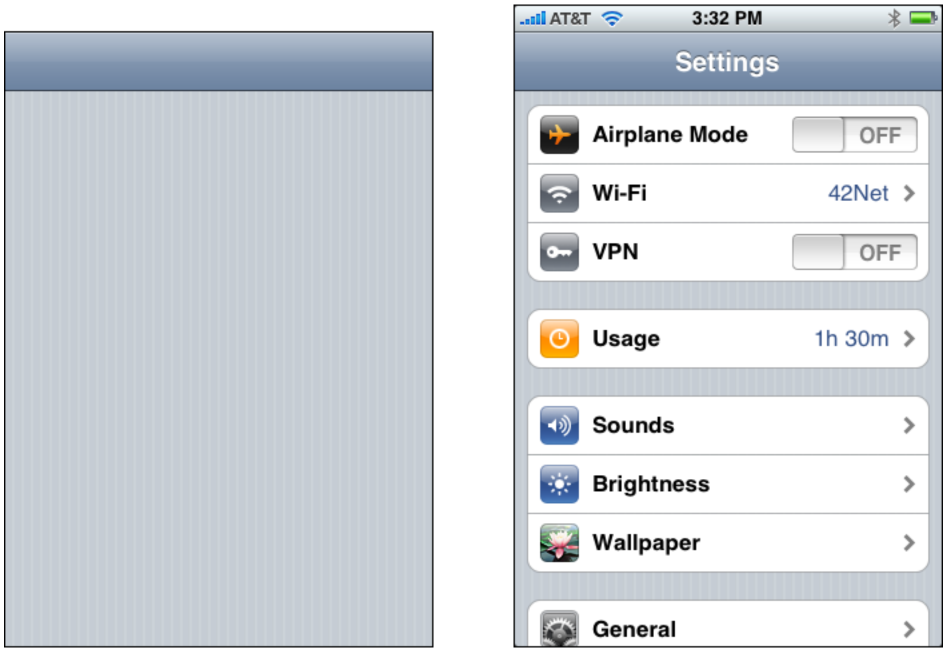
- Measures 320 x 480 points. Including the status bar area allows you to display the status bar color you've chosen immediately, instead of displaying it after your application has finished starting.
- Is identical to the first screen of the application, except for:

- Text. The launch image is static, so any text you display in it will not be localized.
- User interface elements that might change. Avoid including elements that might look different when the application finishes launching, so that users don't experience a flash between the launch image and the first application screen.

If you think that following these guidelines will result in a very plain, boring launch image, you're right. Remember, the launch image is not meant to provide an opportunity for artistic expression; it is solely intended to enhance the user's perception of your application as quick to launch and immediately ready for use. The following examples show you how plain a launch image can be.

The first example is the launch image for the built-in Settings application, shown in Figure 9-3. Note that the Settings launch image on the left does not show the status bar area, but your launch image should, because it allows you to display the status bar color you've chosen.

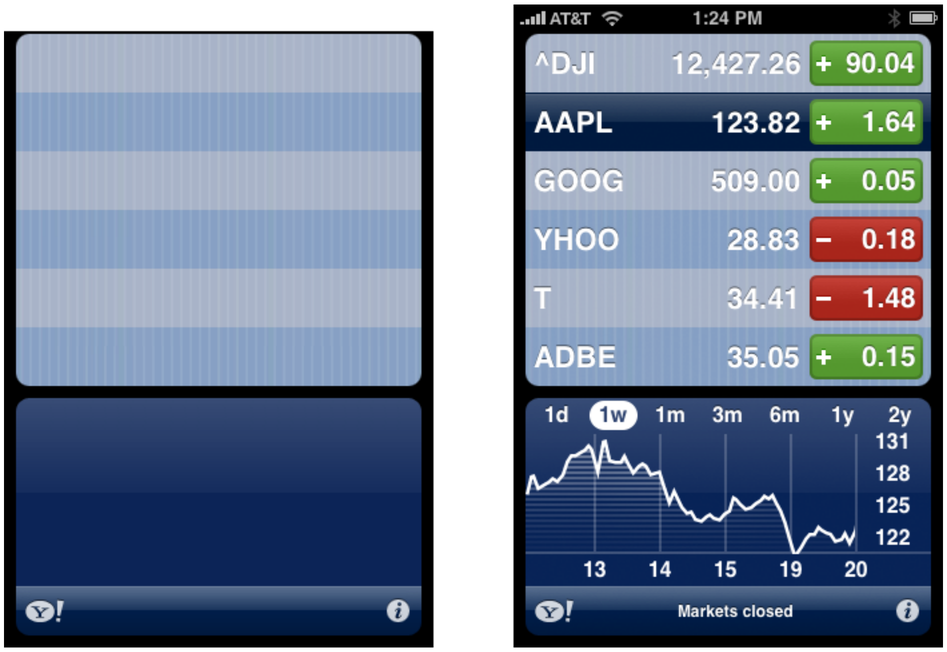
Figure 9-3 The launch image for the Settings application



Airplane Mode OFF
Wi-Fi 42Net
VPN OFF
Usage 1h 30m
Sounds
Brightness
Wallpaper
General

Another launch image example comes from the built-in Stocks application, shown in Figure 9-4. Again, be aware that your launch image should include the status bar area, even though the Stocks launch image shown in Figure 9-4 does not.

Figure 9-4 The launch image for the Stocks application



^DJI	12,427.26	+ 90.04
AAPL	123.82	+ 1.64
GOOG	509.00	+ 0.05
YHOO	28.83	– 0.18
T	34.41	– 1.48
ADBE	35.05	+ 0.15

Settings Images

A **settings image** is an icon you provide that identifies your application's available settings in the built-in Settings application. If your iPhone application must offer settings users can adjust, you provide a settings bundle in your application bundle that lists them (see *iPhone OS Programming Guide* for more information on these bundles). In addition, you provide an icon that appears in the Settings application to mark where your settings are located.

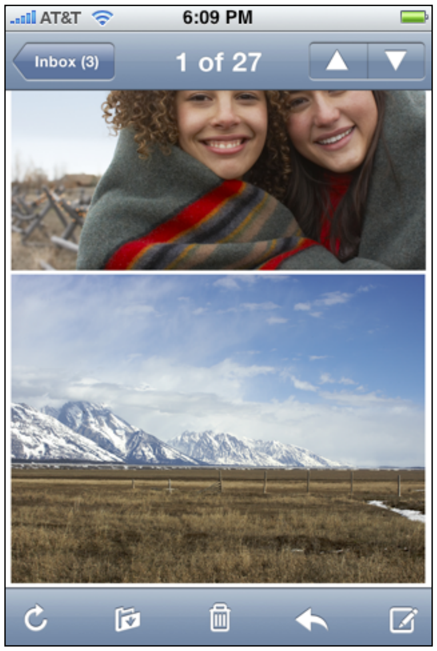
The icon you provide should clearly identify your application so users can easily find your settings among the settings of all the other applications they have installed. Therefore, you should create a streamlined, attractive icon that measures 29 x 29 points.

View and Control Images

iPhone OS provides the compose image for use in user interface elements, including toolbar and navigation bar items. You can see an example of this image in the right end of the toolbar in Figure 9-5.

2008-02-29 | © 2008 Apple Inc. All Rights Reserved.

Figure 9-5 The compose image in the Mail toolbar



Document Revision History

This table describes the changes to *iPhone Human Interface Guidelines*.

Date	Notes
2008-02-29	New document that describes how to design the user interface of an iPhone application.