What's New in iOS

Developer

Contents

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. This Apple confidential information is for use only by registered members of the applicable Apple Developer program. Apple is supplying this confidential 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.

This document provides information about features that were introduced in different versions of iOS and that affect the development of iOS apps. This document describes only features that affect the development of iOS apps. It does not include user-level features unless there is an aspect of the feature that affects app development. It also does not specifically address minor OS updates that include only bug fixes (and no new features). For additional information about a specific release, you should also see the corresponding release notes.

In addition to describing the new features for each iOS release, this document attempts to provide insight as to how and when you might use those features in your own software. Wherever possible, this document also provides links to other Apple conceptual and reference documentation for that feature. For a complete list of features you can use in your iOS apps, see *iOS Technology Overview* .

Organization of This Document

This document includes the following articles:

- "iOS 6.0" (page 8) describes the new and updated features in iOS 6.0.
- "iOS 5.1" (page 20) describes the new and updated features in iOS 5.1.
- "iOS 5.0" (page 21) describes the new and updated features in iOS 5.0.
- "iOS 4.3" (page 44) describes the new and updated features in iOS 4.3.
- "iOS 4.2" (page 47) describes the new and updated features in iOS 4.2.
- "iOS 4.1" (page 51) describes the new and updated features in iOS 4.1.
- "iOS 4.0" (page 53) describes the new and updated features in iOS 4.0.

Getting the iOS SDK

Development of iOS apps requires an Intel-based Macintosh computer and an up-to-date version of Mac OS X. You must also download and install the iOS SDK. For information about how to get the iOS SDK, go to the iOS Dev Center.

iOS 6.0

This article summarizes the key developer-related features introduced in iOS 6. This version of the operating system runs on current iOS-based devices. In addition to describing the key new features, this article lists the documents that describe those features in more detail.

For late-breaking news and information about known issues, see *iOS 6.0 Release Notes* . For the complete list of new APIs added in iOS 6, see *iOS 6.0 API Diffs* .

Maps

In addition to the new map tiles provided by Apple, the Maps app and MapKit framework now support additional interactions with other apps. Apps that do not incorporate their own map support now have an easier way to launch the Maps app and display points of interest or directions. Apps that offer routing information, such as turn-by-turn navigation services, can now register as a routing app and make those services available to the entire system.

Registering as a routing app gives you more opportunities to get your app in front of users. Routing apps are not limited to just driving or walking directions. Routing apps can also include apps that provide directions for the user's favorite bicycle or hiking trail, for air routes, and for subway or other public transportation lines. And your app does not even have to be installed on the user's device. Maps knows about routing apps in the App Store and can provide the user with the option to purchase those apps and use them for directions.

Apps that do not provide routing directions themselves can also take advantage of both Maps and routing apps. Apps can use new interfaces to ask the Maps app to display specific locations or to display routing directions between two locations.

For information about how to provide directions from your app, or how to use the Maps app to display directions, see *Location Awareness Programming Guide* .

Social Framework

The Social framework (`Social.framework`) provides a simple interface for accessing the user's social media accounts. This framework supplants the Twitter framework that was introduced in iOS 5 and adds support for other social accounts, including Facebook and Sina's Weibo service. Apps can use this framework to post status updates and images to a user's account. This framework works with the Accounts framework to provide a single sign-on model for the user and to ensure that access to the user's account is approved.

The UIKit framework also provides a new `UIActivityViewController` class for displaying actions that the user might perform on some selected content. One use of this class is to allow the user to post content to social accounts, such as Twitter or Facebook. You present this class modally on iPhone or using a popover controller on iPad. When the user taps one of the buttons, the view controller presents a new interface to perform the associated action.

For more information about the Social framework, see *Social Framework Reference* . For information about the `UIActivityViewController` class, see *UIKit Framework Reference* .

Pass Kit

Pass Kit is a new technology that uses web services, a new file format, and an Objective-C framework (`PassKit.framework`) to implement support for downloadable passes. Companies can create passes to represent items such as coupons, boarding passes, event tickets, and discount cards for businesses. Instead of carrying a physical representation of these items, users can now store them on their iOS device and use them the same way as before.

Passes are created by your company's web service and delivered to the user's device via email, Safari, or your custom app. The pass itself uses a special file format and is cryptographically signed before being delivered. The file format identifies relevant information about the service being offered so that the user knows what it is for. It might also contain a bar code or other information that you can then use to validate the card so that it can be redeemed or used.

For more information about Pass Kit and for information how to add support for it into your apps, see *Pass Kit Programming Guide* .

Game Center

Game Center has been updated and new features have been added to improve the game playing experience. The Game Kit framework (`GameKit.framework`) includes the following incremental and major changes:

- Challenges allow a player to challenge a friend to beat an achievement or score. Players can issue challenges from the Game Center app. You can also use the `GKChallenge` class to issue challenges from within your game.
- The `GKGameCenterViewController` class incorporates all of the previous capabilities of the leaderboard, achievement and friend request view controllers. You can use the previous view controllers too, but they now present the appropriate tab in the game center view controller.
- The process for authenticating a local player has changed. In the new process, you set the `authenticateHandler` property to a block and call the `authenticate` method. Game Kit no longer displays the authentication interface directly. Instead, your app is asked to present an appropriate authentication view controller, giving you more control over the authentication process.
- The turn-based match class now has support for player timeouts. When using a timeout, you specify a list of players in order. If a player times out, the next player in the list is asked to take a turn instead.
- The `GKMatchmaker` class has been updated to include better support for programmatic matchmaking. It is now easier to implement your own user interface for matchmaking on top of Game Center's built-in support.
- The `GKPlayer` class now supports display names for players.
- The `GKMatch` class provides a method to estimate which player has the best connection to Game Center. You can use this method when implementing your own client-server architecture on top of the `GKMatch` infrastructure.
- The `GKAchievement` class now allows multiple achievements to be submitted at the same time.

For more information about how to use Game Center in your app, see *Game Kit Programming Guide* .

Reminders

The Event Kit framework now includes interfaces for creating and accessing reminders on the user's device. The reminders you create show up in the Reminders app along with ones created by the user. Reminders can include proximity or time-based alarms.

For more information about the interfaces of the Event Kit framework, including the new interfaces for creating reminders, see *Event Kit Framework Reference* .

In-App Purchase

The Store Kit framework (`StoreKit.framework`) now supports the purchasing of iTunes content inside your app and provides support for having downloadable content hosted on Apple servers. With in-app content purchases, you present a view controller that lets users purchase apps, music, books, and other iTunes content directly from within your app. You identify the items you want to make available for purchase but the rest of the transaction is handled for you by Store Kit.

Prior to iOS 6, you were responsible for managing any downloadable content made available through in app purchase. Hosted downloads now simplify the work you need to do to make content available to users. The new `SKDownload` class represents a downloadable piece of content. In addition, the `SKPaymentTransaction` class has been modified to provide an array of download objects for any hosted content. To download a piece of content, you queue a download object on the payment queue. When the download completes, your payment queue observer is notified.

For more information about the classes of the Store Kit framework, see *Store Kit Framework Reference* .

Collection Views

The `UICollectionView` class offers a new way to present ordered data to users. With a collection view object, you are now able to define the presentation and arrangement of embedded views. The collection view class works closely with an accompanying layout object to define the placement of individual data items. UIKit provides a standard flow-based layout object that you can use to implement multi-column grids containing items of standard or varying sizes. And if you want to go beyond grid layouts, you can create your own custom layout objects to support any layout style you choose.

Collection views work with a dedicated group of classes to support the display of items. In addition to cells, collection views can have supplementary views and decoration views. The usage and placement of these views is controlled entirely by the layout object. For example, the flow-based layout object uses supplementary views to implement header and footer views for sections in the collection view.

Other noteworthy features of collection views include:

- A `UICollectionViewController` class for managing the presentation of collection views in your app
- Support for animating content within the collection view
- Batch support for inserting, moving, and deleting items
- A simplified model for creating and managing cells and other views

For more information about the new collection view classes, see *UIKit Framework Reference* .

UI State Preservation

State preservation makes it easier for apps to restore their user interface to the state it was in when the user last used it. Prior to iOS 6, apps were encouraged to write out information about their current interface state in the event that the app was terminated. Upon relaunch, the app could use this state to restore its interface and make it seem as if the app had never quit. State preservation simplifies this process by providing you with all of the core infrastructure for saving and restoring your app's interface.

Implementing state preservation still requires effort on your part to identify what parts of your interface to save. In addition, the preservation and restoration processes can be customized to accommodate unforeseen circumstances, such as missing content during a subsequent relaunch or changes to your app's UI.

For more information about how to add state preservation support to your app, see *iOS App Programming Guide* .

Auto Layout

Auto layout improves upon the "springs and struts" model previously used to lay out the elements of a user interface. With auto layout, you define rules for how to lay out the elements in your user interface. These rules express a larger class of relationships and are more intuitive to use than springs and struts.

The entities used in Auto Layout are Objective-C objects called constraints. This approach brings you a number of benefits:

- Localization through swapping of strings alone, instead of also revamping layouts.
- Mirroring of user-interface elements for right-to-left languages like Hebrew and Arabic.
- Better layering of responsibility between objects in the view and controller layers.

  A view object usually knows best about its standard size and its positioning within its superview and relative to its sibling views. A controller can override these values if something nonstandard is required.

For more information about using auto layout, see *Cocoa Auto Layout Guide* .

Data Privacy

In addition to location data, the system now asks the user's permission before allowing third-party apps to access certain user data, including:

- Contacts
- Calendars

- Reminders
- Photo Library

For contact, calendar, and reminder data, your app needs to be prepared to be denied access to these items and to adjust its behavior accordingly. If the user has not yet been prompted to allow access, the returned structure is valid but contains no records. If the user has denied access, the app receives a `NULL` value or no data. If the user grants permission to the app, the system subsequently notifies the app that it needs to reload or revert the data.

For the photo library, the existing interface supports the app being denied access.

Your app can provide a description for how it intends to use the data in its `Info.plist` file. That data is then displayed to the user when the system needs to prompt for access. For more information about the keys you need to add to your `Info.plist` file, see *Information Property List Key Reference* . For more information about accessing specific kinds of data, see the respective framework reference.

Additional Framework Enhancements

In addition to the items discussed in the preceding sections, the following frameworks have additional enhancements. For a complete list of new interfaces, see *iOS 6.0 API Diffs* .

UIKit Framework

The UIKit framework (`UIKit.framework`)includes the following enhancements:

- UIKit now supports state preservation for your app's user interface; see "UI State Preservation" (page 12).
- Views now support constraint-based layout; see "Auto Layout" (page 12).
- The `UICollectionView` class (and its supporting classes and protocols) support the presentation of ordered collections of items; see "Collection Views" (page 11).
- `UITextView` and `UITextField` now support styled text using attributed strings.
- UIKit now includes support for drawing `NSAttributedString` and `NSMutableAttributedString` objects. In addition, string-based views now support attributed strings for content.
- Accessibility support has been improved to include new VoiceOver enhancements:
    - VoiceOver can now use gestures to trigger specific actions.
    - Developers can order items together in the element list that VoiceOver creates to provide a more logical flow from element to element.
    - Scroll views can now provide a special status string during scrolling.

    - You can now post notifications that let the accessibility system know your app's layout or UI have changed.
    - New notifications provide information about the state of VoiceOver announcements.
- The `UIActivityViewController` class adds support for sharing content via services like email, Twitter, and Facebook; see also, "Social Framework" (page 9).
- The `UIDevice` class adds advertising-specific and vendor-specific identifiers.
- The `UIImage` class includes new initialization methods for specifying the scale factor of an image.
- Appearance customization support has been added to the `UIBarButtonItem`, `UIPageControl`, `UIPageViewController`, `UISwitch`, and `UIStepper` classes.
- The `UITableView` class includes the following changes:
    - Support for a new `UITableViewHeaderFooterView` class implements the table's headers and footers
    - A simplified model for creating and managing cells and other views.
- The `UITableViewController` class allows you to add a built-in refresh control (`UIRefreshControl`) to reload the table contents.
- Storyboards allow you to unwind segues and specify restoration identifiers.
- The `UIWebView` class provides a way to disable the incremental rendering of web content.
- The `UIViewController` class has new interfaces supporting the following behaviors:
    - New interfaces provide a clearer path for managing and tracking interface rotations.
    - You can prevent a segue from being triggered.
    - Support has been added for unwinding segues.
- You can now subclass `UINavigationBar` and incorporate your custom navigation bar into your app's navigation interfaces.

For information about the classes of the UIKit framework, see *UIKit Framework Reference* .

OpenGL ES

OpenGL ES includes the following new extensions:

- The GL_EXT_texture_storage extension allows your app to specify the entire structure of a texture in a single call, allowing your textures to be optimized further by OpenGL ES.
- The GL_APPLE_copy_texture_levels extension builds on top of the functionality of the `GL_EXT_texture_storage` extension and allows a set of texture mipmaps to be copied from one texture to another.

- The GL_APPLE_map_buffer_range extension improves performance when you only need to modify a subset of a buffer object.
- The GL_APPLE_sync extension provides fine-grain synchronization to your app. It allows you to choose a subset of submitted OpenGL ES commands and block until those commands complete.
- The GL_APPLE_shader_framebuffer_fetch extension is only available to OpenGL ES 2.0 applications and provides access to the framebuffer data as an input to your fragment shader.

These extensions are available on all devices capable of running iOS 6. As always, check for the existence of an extension before using it in your application.

Media Player Framework

The `MPVolumeView` class now provides interfaces for customizing the appearance of the volume view. You can use these interfaces to change the images associated with the volume slider and routing button.

For information about the classes of the Media Player framework, see *Media Player Framework Reference* .

Image IO Framework

The Image IO framework (`ImageIO.framework`) includes support for accessing EXIF and IPTC metadata properties for images. You can access this metadata using functions associated with the `CGImageSourceRef` and `CGImageDestinationRef` opaque types.

For information about the classes of the Image IO framework, see *Image I/O Reference Collection* .

iAd Framework

The iAd framework (`iAd.framework`) supports a new medium rectangle banner size for ads on iPad devices. For information about the classes of the iAd framework, see *iAd Framework Reference* .

Foundation Framework

The Foundation framework (`Foundation.framework`) includes the following enhancements:

- The `NSFileManager` class includes the `ubiquityIdentityToken` method for determining the availability of iCloud, and the `NSUbiquityIdentityDidChangeNotification` notification for responding to a user logging into or out of iCloud, or changing accounts.
- The `NSUUID` class helps you create objects that represent various types of UUIDs (Universally Unique Identifiers). For example, you can create an `NSUUID` object with RFC 4122 version 4 random bytes, or you can base it on an existing UUID string.

- The `NSURLRequest` class lets you specify whether a request should be allowed to happen over a cellular network.
- The `NSString` class has new methods for converting a string to uppercase, lowercase, or an initial capital case.

For information about the classes of the Foundation framework, see *Foundation Framework Reference* .

External Accessory Framework

The External Accessory framework (`ExternalAccessory.framework`) includes new interfaces for managing connections to Bluetooth devices. Apps can now post an alert panel that lists the Bluetooth devices available for pairing. Support is also provided for waking previously paired accessories that do not automatically connect.

For information about the classes of the External Accessory framework, see *External Accessory Framework Reference* .

Event Kit Framework

The Event Kit framework (`EventKit.framework`) includes the following enhancements:

- The framework supports the creation of reminders; see "Reminders" (page 10).
- Calendar and reminder events can now vend an external identifier that lets multiple devices refer to the same event on the server. The server provides the actual identifier string, which is accessed using the `calendarItemExternalIdentifier` property of `EKCalendarItem`.
- For apps linked against iOS 6 and later, the system asks for the user's permission before granting access to calendar and reminder data.
- Apps can now cancel editing programmatically from an `EKEventEditViewController` object.

For information about the classes of the Event Kit framework, see *Calendar and Reminders Programming Guide* .

Core Video Framework

The Core Video framework (`CoreVideo.framework`) adds support for two new pixel formats. These formats provide efficient storage for one-channel and two-channel images that interoperate with OpenGL ES. For information about the functions of the Core Video framework, see *Core Video Framework Reference* .

Core Media Framework

The Core Media framework (`CoreMedia.framework`) adds the `CMClockRef` and `CMTimebaseRef` types, which define fundamental media time service behaviors. For information about the functions of the Core Media framework, see *Core Media Framework Reference* .

Core Location Framework

The Core Location framework (`CoreLocation.framework`) includes the following changes.

- The `CLLocationManager` object now pauses the delivery of location events when the user has not moved for an extended period of time. This behavior saves power by allowing the framework to turn off the GPS and other hardware. It is enabled by default but may be disabled by changing the value in the `pausesLocationUpdatesAutomatically` property of the location manager object.
- You can refine location accuracy based on usage by assigning an appropriate value to the activityType property of `CLLocationManager`. This property currently lets you distinguish between driving usage and fitness usage.
- The framework supports improved location accuracy while offline and driving.

For information about the classes of the Core Location framework, see *Core Location Framework Reference* .

Core Bluetooth Framework

The Core Bluetooth framework (`CoreBluetooth.framework`) supports interacting with Bluetooth devices in peripheral mode. Previously, an iOS device could only interact in central mode but it can advertise itself in peripheral mode and communicate with other iOS devices.

For information about the classes of the Core Bluetooth framework, see *Core Bluetooth Framework Reference* .

Core Audio

Core Audio family of frameworks includes the following changes:

- There is a new AUDeferredRenderer audio unit that allows audio to be processed on a lower-priority thread using relatively larger time slices.
- The AudioQueueProcessingTap audio unit allows you to intercept the data in an audio queue and process it.

For information about the audio technologies available in iOS, see *Multimedia Programming Guide* . For information about the new audio units, see *Audio Unit Component Services Reference* .

AV Foundation Framework

The AV Foundation framework (`AVFoundation.framework`) includes the following enhancements:

- The `AVPlayer` class adds support for syncing playback to an external time source. A single player object can also play both HTTP Live Streams and file-based assets (including both local files and files that are progressively downloaded). And you can use multiple `AVPlayerLayer` objects to display the visual output of the same player object.
- The new `AVPlayerItemOutput` class works with an `AVPlayerItem` object to obtain decoded video frames during playback so that your app can process them.
- The `AVAssetResourceLoader` class allows you to insert a delegate object into the asset loading process and handle custom resource requests. This class works with the `AVURLAsset` class.
- The `AVAudioSession` class now exposes information about the current audio route in use.
- The `AVAudioMixInputParameters` class provides access to an `MTAudioProcessingTapRef` data type, which can be used to access audio data before it is played, read, or exported.
- The `AVAudioSession` class includes the following enhancements:
    - Use of an audio session delegate to detect interruptions and changes is now deprecated in favor of a set of new notifications.
    - Audio sessions support a new audio category, `AVAudioSessionCategoryMultiRoute`, that allows routing of audio to and from multiple audio hardware devices.
    - Audio sessions support a new mode, `AVAudioSessionModeMoviePlayback`, that engages appropriate output signal processing for movie playback scenarios.
- The `AVCaptureConnection` class allows you to enable or disable support for video stabilization on incoming video.
- The `AVCaptureMetadataOutput` class is a new class for intercepting metadata emitted by a capture connection object.
- The `AVMetadataFaceObject` is a new class that reports the face-detection data captured by an `AVCaptureMetadataOutput` object.
- The `AVTextStyleRule` class is a new class for specifying style options for subtitles, closed captions, and other text-related media assets.
- The `AVAudioPlayer` class can play audio items from the user's iPod library using URLs obtained from the `MPMediaLibrary` class. You can also set channel assignments using the `channelAssignments` property.

For information about the classes of the AV Foundation framework, see *AV Foundation Framework Reference*.

Accelerate Framework

The Accelerate framework (`Accelerate.framework`) includes new vector-scalar power functions, vDSP functions, discrete cosine transform functions, SSE-related vector functions, sine functions, and vImage functions.

For information about the functions of the Accelerate framework, see *Accelerate Framework Reference* .

iOS 5.1

This article summarizes the key developer-related features introduced in iOS 5.1. This version of the operating system runs on current iOS-based devices. In addition to describing the key new features, this article lists the documents that describe those features in more detail.

For late-breaking news and information about known issues, see *iOS 5.1 Release Notes* . For the complete list of new APIs added in iOS 5.1, see *iOS 5.1 API Diffs* .

Dictation Support in Text Input Views

On supported devices, iOS automatically inserts recognized phrases into the current text view when the user has chosen dictation input. The new `UIDictationPhrase` class (declared in `UITextInput.h`) provides you with a string representing a phrase that a user has dictated. In the case of ambiguous dictation results, the new class provides an array containing alternative strings. New methods in the `UITextInput` protocol allow your app to respond to the completion of dictation.

iOS 5.0

This article summarizes the key developer-related features introduced in iOS 5.0. This version of the operating system runs on current iOS-based devices. In addition to describing the key new features, this article lists the documents that describe those features in more detail.

For late-breaking news and information about known issues, see *iOS 5.0 Release Notes* . For the complete list of new APIs added in iOS 5.0, see *iOS 5.0 API Diffs* .

iCloud Storage APIs

The iCloud storage APIs let your app write user documents and data to a central location and access those items from all of a user's computers and iOS devices. Making a user's documents ubiquitous using iCloud means that a user can view or edit those documents from any device without having to sync or transfer files explicitly. Storing documents in a user's iCloud account also provides a layer of security for that user. Even if a user loses a device, the documents on that device are not lost if they are in iCloud storage.

There are two ways that apps can take advantage of iCloud storage, each of which has a different intended usage:

- **iCloud document storage**—Use this feature to store user documents and data in the user's iCloud account.
- **iCloud key-value data storage**—Use this feature to share small amounts of data among instances of your app.

Most apps will use iCloud document storage to share documents from a user's iCloud account. This is the feature that users think of when they think of iCloud storage. A user cares about whether documents are shared across devices and can see and manage those documents from a given device. In contrast, the iCloud key-value data store is not something a user would see. It is a way for your app to share small amounts of data (up to a per-app total of 1 MB and a maximum of 1,024 keys) with other instances of itself. Apps can use this feature to store important state information. A magazine app might save the issue and page that the user read last, while a stocks app might store the stock symbols the user is tracking.

For information on how to use iCloud document and key-value data storage, see *iCloud Design Guide* .

iCloud Backup

Users can now opt to have their apps and app data backed up directly to their iCloud account, making it easier to restore apps to their most recent state. Having data backed up in iCloud makes it easy for a user to reinstall that data to a new or existing iOS device. However, because the amount of space in a user's iCloud account is limited, apps must be even more selective about where they store files.

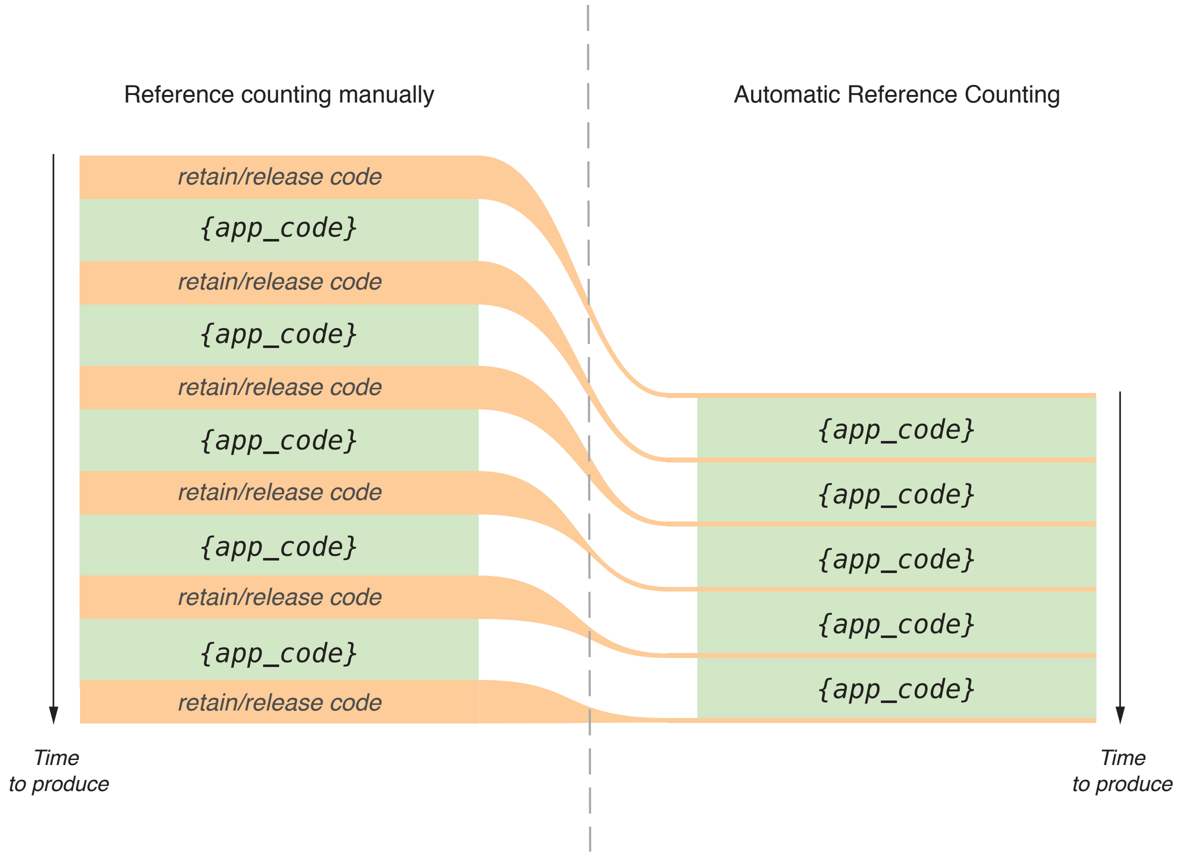
The placement of files in your app's home directory determines what gets backed up and what does not. Anything that would be backed up to a user's computer is also backed up wirelessly to iCloud. Thus, everything in the `Documents` directory and most (but not all) of your app's `Library` directory. To minimize the amount of data stored in the user's iCloud account, developers are encouraged to put more files in the `Library/Caches` directory, especially if those files can be easily re-created or obtained in another way.

Note Any documents that your app stores explicitly in iCloud (using the iCloud storage APIs) are not backed up with your app. (Those documents are already stored in the user's iCloud account and therefore do not need to be backed up separately.)

For information on which directories are backed up, and for information about iCloud document storage, see *iOS App Programming Guide* .

Automatic Reference Counting

Automatic Reference Counting (ARC) is a compiler-level feature that simplifies the process of managing the lifetimes of Objective-C objects. Instead of you having to remember when to retain or release an object, ARC evaluates the lifetime requirements of your objects and automatically inserts the appropriate method calls at compile time.



To be able to deliver these features, ARC imposes some restrictions—primarily enforcing some best practices and disallowing some other practices:

- Do not call the `retain`, `release`, `autorelease`, or `dealloc` methods in your code.

  In addition, you cannot implement custom `retain` or `release` methods.

  Because you do not call the `release` method, there is often no need to implement a custom `dealloc` method—the compiler synthesizes all that is required to relinquish ownership of instance variables. You can provide a custom implementation of `dealloc` if you need to manage other resources.

- Do not store object pointers in C structures.

  Store object pointers in other objects instead of in structures.

- Do not directly cast between object and nonobject types (for example, between `id` and `void*`).

You must use special functions or casts that tell the compiler about an object's lifetime. You use these to cast between Objective-C objects and Core Foundation objects.

- You cannot use `NSAutoreleasePool` objects.

  Instead, you must use a new `@autoreleasepool` keyword to mark the start of an autorelease block. The contents of the block are enclosed by curly braces, as shown in the following example:

```
@autoreleasepool
{
   // Your code here
}
```

ARC encourages you to think in terms of object graphs, and the relationships between objects, rather than in terms of retain and release. For this reason, ARC introduces new lifetime qualifiers for objects, including **zeroing weak references**. The value of a zeroing weak reference is automatically set to `nil` if the object to which it points is deallocated. There are qualifiers for variables, and new `weak` and `strong` declared property attributes, as illustrated in the following examples:

```
// The following declaration is a synonym for: @property(retain) MyClass *myObject;
@property(strong) MyClass *myObject;


// The following declaration is similar to "@property(assign) MyOtherClass
*delegate;"
// except that if the MyOtherClass instance is deallocated,
// the property value is set to nil instead of remaining as a dangling pointer
@property(weak) MyOtherClass *delegate;
```

Xcode provides migration tools to help convert existing projects to use ARC. For more information about how to perform this migration, see *What's New In Xcode*. For more information about ARC itself, see *Transitioning to ARC Release Notes*.

Storyboards

Storyboards are the new way to define your app's user interface. In the past, you used nib files to define your user interface one view controller at a time. A storyboard file captures your entire user interface in one place and lets you define both the individual view controllers and the transitions between those view controllers. As a result, storyboards capture the flow of your overall user interface in addition to the content you present.

If you are creating new apps, the Xcode templates come preconfigured to use storyboards. For other apps, the process for using storyboards is as follows:

1. Configure your app's `Info.plist` file to use storyboards:
   - Add the `UIMainStoryboardFile` key and set its value to the name of your storyboard file.
   - Remove the existing `NSMainNibFile` key. (Storyboards replace the main nib file.)
2. Create and configure the storyboard file in Xcode; see "Creating Storyboard Files" (page 25).
3. Update your view controllers to handle storyboard transitions; see "Preparing to Transition to a New View Controller" (page 26).
4. If you ever need to present a view controller manually (perhaps to support motion-related events), use the storyboard classes to retrieve and present the appropriate view controller; see "Presenting Storyboard View Controllers Programmatically" (page 26).

Apps can use a single storyboard file to store all of their view controllers and views. At build time, Interface Builder takes the contents of the storyboard file and divides it up into discrete pieces that can be loaded individually for better performance. Your app never needs to manipulate these pieces directly, though. All you must do is declare the main storyboard in your app's `Info.plist` file. UIKit handles the rest.

Creating Storyboard Files

You use Interface Builder to create storyboard files for your app. Most apps need only one storyboard file, but you can create multiple storyboard files if you want. Every storyboard file has a view controller known as the **initial view controller**. This view controller represents the entry point into the storyboard. For example, in your app's main storyboard file, the initial view controller would be the first view controller presented by your app.

Each view controller in a storyboard file manages a single scene. For iPhone apps, a **scene** manages one screen's worth of content, but for iPad apps the content from multiple scenes can be on screen simultaneously. To add new scenes to your storyboard file, drag a view controller from the library to the storyboard canvas. You can then add controls and views to the view controller's view just as you would for a nib file. And as before, you can configure outlets and actions between your view controller and its views.

When you want to transition from one view controller to another, Control-click a button, table view cell, or other trigger object in one view controller, and drag to the view controller for a different scene. Dragging between view controllers creates a **segue**, which appears in Interface Builder as a configurable object. Segues support all of the same types of transitions available in UIKit, such as modal transitions and navigation transitions. You can also define custom transitions and transitions that replace one view controller with another.

For more information about using Interface Builder to configure your storyboard files, see *What's New In Xcode* .

Preparing to Transition to a New View Controller

Whenever a user triggers a segue in the current scene, the storyboard runtime calls the `prepareForSegue:sender:` method of the current view controller. This method gives the current view controller an opportunity to pass any needed data to the view controller that is about to be displayed. When implementing your view controller classes, you should override this method and use it to handle these transitions.

For more information about implementing the methods of the `UIViewController` class, see *UIViewController Class Reference* .

Presenting Storyboard View Controllers Programmatically

Although the storyboard runtime usually handles transitions between view controllers, you can also trigger segues programmatically from your code. You might do so when setting up the segue in Interface Builder is not possible, such as when using accelerometer events to trigger the transition. There are several options for transitioning to a new view controller:

- If a storyboard file contains an existing segue between the current view controller and the destination view controller (perhaps triggered by some other control in the view controller), you can trigger that segue programmatically using the `performSegueWithIdentifier:sender:` method of `UIViewController`.
- If there is no segue between the view controllers but the destination view controller is defined in the storyboard file, first load the view controller programmatically using the `instantiateViewControllerWithIdentifier:` method of `UIStoryboard`. Then present the view controller using any of the existing programmatic means, such as by pushing it on a navigation stack.
- If the destination view controller is not in the storyboard file, create it programmatically and present it as described in *View Controller Programming Guide for iOS* .

Newsstand Support

Newsstand provides a central place for users to read magazines and newspapers. Publishers who want to deliver their magazine and newspaper content through Newsstand can create their own iOS apps using the Newsstand Kit framework (`NewsstandKit.framework`), although doing so is not required. A big advantage of the Newsstand Kit framework, however, is that you can use it to initiate background downloads of new magazine and newspaper issues. After you start a download, the system handles the download operation and notifies your app when the new content is available.

Unlike other iOS apps, Newsstand apps appears only in Newsstand itself, not in a user's Home screen. And instead of displaying an app icon, the app typically displays the cover of their most recent issue, with some additional adornments provided by Newsstand. When the user taps that cover art, your app launches normally to present the current issue or any back issues that were downloaded and are still available.

> **Note** Newsstand apps must include the `UINewsstandApp` key in their `Info.plist` file to indicate that they support Newsstand. For more information about this key, see *Information Property List Key Reference*.

Creating an app that uses Newsstand Kit requires some interplay between the actual app and the content servers that you manage. Your servers are responsible for notifying the app when new content is available, typically using a push notification. If your Newsstand app includes the `UIBackgroundModes` key with the `newsstand-content` value in its `Info.plist` file, your Newsstand app is launched in the background so that it can start downloading the latest issue. The download process itself is managed by the system, which notifies your app when the content is fully downloaded and available.

When your server is alerting your app of a new issue, that server should include the `content-available` property (with a value of `1`) in the JSON payload. This property tells the system that it should launch your app so that it can begin downloading the new issue. Apps are launched and alerted to new issues once in a 24-hour period at most, although if your app is running when the notification arrives, it can begin downloading the content immediately.

In addition to your server providing content for each new issue, it should also provide cover art to present in Newsstand when that issue is available. This cover art is displayed in place of the app's default icon, which is specified by the Newsstand-specific icons in the `CFBundleIcons` key of your app's `Info.plist` file. Cover art gives users a more visual cue that a new issue is available. Your app can also add a badge to new issues.

For information about the classes you use to manage Newsstand downloads, see *Newsstand Kit Framework Reference*. For information about how to use push notifications to notify your apps, see *Local and Push Notification Programming Guide*.

AirPlay Improvements

AirPlay lets users stream audio and video from an iOS-based device to AirPlay–enabled devices such as televisions and audio systems. In iOS 5, developers can now use AirPlay to present app content on a nearby Apple TV 2. Users can now mirror the content of an iPad 2 to an Apple TV using AirPlay for any app. And developers who want to display different content (instead of mirroring) can assign a new window object to any `UIScreen` objects connected to an iPad 2 via AirPlay.

In addition, you can now take advantage of AirPlay in the following ways:

- Apps that use AV Foundation can now use the `AVPlayer` class to stream audio and video content over AirPlay; see *AV Foundation Framework Reference*.
- The Media Player framework includes support for displaying "Now Playing" information in several locations, including as part of the content delivered over AirPlay; see *MPNowPlayingInfoCenter Class Reference*.
- The `UIWebView` class now supports the presentation of multimedia content over AirPlay. This support is enabled by default, but you can opt out of it if you want to.

Note In iOS 5, AirPlay is enabled by default for all objects that support it. If you do not want your app's content to be playable over AirPlay, you must opt out explicitly.

For more information about delivering content over AirPlay, and the support media formats, see *AirPlay Overview*.

New Frameworks

In iOS 5.0, there are several new frameworks you should investigate.

GLKit Framework

The GLKit framework (`GLKit.framework`) contains a set of Objective-C based utility classes that simplify the effort required to create an OpenGL ES 2.0 app. GLKit provides support for four key areas of app development:

- The `GLKView` and `GLKViewController` classes provide a standard implementation of an OpenGL ES–enabled view and associated rendering loop. The view manages the underlying framebuffer object on behalf of the app; your app just draws to it.

- The `GLKTextureLoader` class provides image conversion and loading routines to your app, allowing it to automatically load texture images into your context. It can load textures synchronously or asynchronously. When loading textures asynchronously, your app provides a completion handler block to be called when the texture is loaded into your context.
- The GLKit framework provides implementations of vector, matrix, and quaternions as well as a matrix stack operation to provide the same functionality found in OpenGL ES 1.1.
- The `GLKBaseEffect`, `GLKSkyboxEffect`, and `GLKReflectionMapEffect` classes provide configurable graphics shaders that implement commonly used graphics operations. In particular, the `GLKBaseEffect` class implements the lighting and material model found in the OpenGL ES 1.1 specification, simplifying the effort required to migrate an app from OpenGL ES 1.1 to OpenGL ES 2.0.

For information about the classes of the GLKit framework, see *GLKit Framework Reference* .

Core Image Framework

The Core Image framework (`CoreImage.framework`) provides a powerful set of built-in filters for manipulating video and still images. You can use the built-in filters for everything from simple operations (like touching up and correcting photos) to more advanced operations (like face and feature detection). The advantage of using these filters is that they operate in a nondestructive manner so that your original images are never changed directly. In addition, Core Image takes advantage of the available CPU and GPU processing power to ensure that operations are fast and efficient.

The `CIImage` class provides access to a standard set of filters that you can use to improve the quality of a photograph. To create other types of filters, you can create and configure a `CIFilter` object for the appropriate filter type.

For information about the classes and filters of the Core Image framework, see *Core Image Reference Collection* .

Twitter Framework

The Twitter framework (`Twitter.framework`) provides support for sending Twitter requests on behalf of the user and for composing and sending tweets. For requests, the framework handles the user authentication part of the request for you and provides a template for creating the HTTP portion of the request. (Refer to the Twitter API for populating the content of the request.) The composition of tweets is accomplished using the `TWTweetComposeViewController` class, which is a view controller that you post with your proposed tweet content. This class gives the user a chance to edit or modify the tweet before sending it.

Users control whether an app is allowed to communicate with Twitter on their behalf using Settings. The Twitter framework also works in conjunction with the Accounts framework (`Accounts.framework`) to access the user’s account.

For information about the classes of the Twitter framework, see *Twitter Framework Reference*. For information about the Accounts framework, see "Accounts Framework" (page 30).

Accounts Framework

The Accounts framework (`Accounts.framework`) provides a single sign-on model for certain user accounts. Single sign-on improves the user experience, because apps no longer need to prompt a user separately for login information related to an account. It also simplifies the development model for you by managing the account authorization process for your app. In iOS 5.0, apps can use this framework in conjunction with the Twitter framework to access a user's Twitter account.

For more information about the classes of the Accounts framework, see *Accounts Framework Reference*.

Generic Security Services Framework

The Generic Security Services framework (`GSS.framework`) provides a standard set of security-related services to iOS apps. The basic interfaces of this framework are specified in IETF RFC 2743 and RFC 4401. In addition to offering the standard interfaces, iOS includes some additions for managing credentials that are not specified by the standard but that are required by many apps.

For information about the interfaces of the GSS framework, see the header files.

Core Bluetooth

The Core Bluetooth framework (`CoreBluetooth.framework`) allows developers to interact specifically with Bluetooth Low-Energy ("LE") accessories. The Objective-C interfaces of this framework allow you to scan for LE accessories, connect and disconnect to ones you find, read and write attributes within a service, register for service and attribute change notifications, and much more.

For more information about the interfaces of the Core Bluetooth framework, see the header files.

App Design-Level Improvements

The following sections describe new capabilities that you can incorporate into the model, view, and controller layers of your app.

Document Support

Cocoa Touch now includes a `UIDocument` class for managing the data associated with user documents. If you are implementing a document-based app, this class reduces the amount of work you must do to manage your document data. In addition to providing a container for all of your document-related data, the `UIDocument` class provides built-in support for a number of features:

- Asynchronous reading and writing of data on a background queue, allowing your app to remain responsive to users while reading and writing operations occur.
- Support for coordinated reading and writing of documents, which is required for documents in iCloud storage.
- Safe saving of document data by writing data first to a temporary file and then replacing the current document file with it.
- Support for resolving conflicts between different versions of your document if a conflict occurs.
- Automatic saving of document data at opportune moments.
- Support for flat file and package file representations on disk.
- For apps that use Core Data to manage their content, there is also a `UIManagedDocument` subclass to manage interactions with documents in the database.

If you are implementing an app that supports iCloud storage, the use of document objects makes the job of storing files in iCloud much easier. Document objects are file presenters and handle many of iCloud-related notifications that you might otherwise have to handle yourself. For more information about supporting iCloud storage, see "iCloud Storage APIs" (page 21).

For information about the `UIDocument` class, see *UIDocument Class Reference*. For information about the `UIManagedDocument` class, see *UIManagedDocument Class Reference*.

Data Protection Improvements

Introduced in iOS 4.0, data protection lets you store app and user data files on disk in an encrypted format so that they can be accessed only when a user's device is unlocked. In iOS 5.0, you now have more flexibility regarding when your app can access protected files.

- Using the `NSFileProtectionCompleteUnlessOpen` option, your app can access a file while the device is unlocked and, if you keep the file open, continue to access that file after the user locks the device.
- Using the `NSFileProtectionCompleteUntilFirstUserAuthentication` option, your app cannot access the file while the device is booting or until the user unlocks the device. After the user unlocks the device for the first time, you can access the file even if the user subsequently locks the device again.

You should protect files as soon as possible after creating them. For information about how to protect files, see *iOS App Programming Guide* .

Custom Appearance for UIKit Controls

You can now customize the appearance of many UIKit views and controls to give your app a unique look and feel. For example, you might use these customizations to make the standard system controls match the branding for the rest of your app.

UIKit supports the following customizations:

- You can set the tint color, background image, and title position properties (among other) on a wide variety of objects, including toolbars, navigation bars, search bars, buttons, sliders, and some other controls.
- You can set attributes of some objects directly, or you can set the default attributes to use for a class using an appearance proxy.

  An appearance proxy is an object you use to modify the default appearance of visual objects such as views and bar items. Classes that adopt the `UIAppearance` protocol support the use of an appearance proxy. To modify the default appearance of such a class, retrieve its proxy object using the `appearance` class method and call the returned object's methods to set new default values. A proxy object implements those methods and properties from its proxied class that are tagged with the `UI_APPEARANCE_SELECTOR` macro. For example, you can use a proxy object to change the default tint color (through the `progressTintColor` or `trackTintColor` properties) of the `UIProgressView` class.

  If you want to set a different default appearance based on how a given object is used in your app, you can do so using the proxy object returned by the `appearanceWhenContainedIn:` method instead. For example, you use this proxy object to set specific default values for a button only when it is contained inside a navigation bar.

  Any changes you make with a proxy object are applied, at view layout time, to all instances of the class that exist or that are subsequently created. However, you can still override the proxy defaults later using the methods and properties of a given instance.

For information about the methods for customizing the appearance of a class, see the description of that class in *UIKit Framework Reference* .

Container View Controller Support

The `UIViewController` class now allows you to define your own custom container view controllers and present content in new and interesting ways. Examples of existing container view controllers include `UINavigationController`, `UITabBarController`, and `UISplitViewController`. These view controllers mix custom content with content provided by one or more separate view controller objects to create a unique presentation for app content. Container view controllers act as a parent for their contained view controllers, forwarding important messages about rotations and other relevant events to their children.

For more information about view controllers and the methods you need to implement for container view controllers, see *UIViewController Class Reference* .

Settings

Apps that deliver custom preferences can now use a new radio group element. This element is similar to the multivalue element for selecting one item from a list of choices. The difference is that the radio group element displays its choices inline with your preferences instead of on a separate page.

For more information about displaying app preferences using the Settings app, see "App-Related Resources" in *iOS App Programming Guide* . For information about the property-list keys you use to build your Settings bundle, see *Settings Application Schema Reference* .

Xcode Tools

The following sections describe the improvements to the Xcode tools and the support for developing iOS apps. For detailed information about the features available in Xcode 4.2, see *What's New In Xcode* .

Xcode Improvements

Xcode 4.2 adds support for many features that are available in iOS 5.0.

- The LLVM compiler supports Automatic Reference Counting (ARC), and Xcode includes a menu item to convert targets to use ARC. (For more information about ARC and about how to use it in your apps, see "Automatic Reference Counting" (page 23).)
- The Interface Builder user interface provides support for creating storyboard files for your iOS apps. (For more information about using storyboards in your iOS apps, see "Storyboards" (page 25).)
- In iOS Simulator, you can now simulate different locations for apps using the Core Location framework.
- You can download your app data from an iOS device and automatically restore that data when debugging or testing in iOS simulator or on a device.

OpenGL ES Debugging

The debugging experience in Xcode has been updated to include a new workflow for debugging OpenGL ES apps. You can now use Xcode to do the following for your OpenGL ES apps:

- Introspect OpenGL ES state information and objects such as view textures, shaders, and so on.
- Set breakpoints on OpenGL ES errors, set conditional OpenGL ES entry point breakpoints, break on frame boundaries, and so on.

UI Automation Enhancements

The Automation instrument now includes a script editor and the ability to capture (record) actions into your script as you perform them on a device. There are also enhancements to the objects that you use in the Automation instrument to automate UI testing:

- The `UIATarget` object can now simulate rotate gestures and location changes.
- The `UIAHost` object supports executing a task from the Instruments app itself.
- The `UIAElement` object can now simulate a rotate gesture centered on the element.
- Several functions that were previously available only in `UIAWindow` and `UIAPopover` were moved to `UIAElement` because they are common to all element objects.
- The `UIAKeyboard` object now supports performing a sequence of keyboard taps to simulate the typing of a string.

For information about the enhancements to the Automation instrument, see *Instruments New Features User Guide*. For information about the JavaScript objects and commands that you use in your scripts, see *UI Automation Reference Collection*.

Instruments

The Instruments app in Xcode 4.2 adds several new instruments for iOS developers:

- System Trace for iOS—Uses several instruments to profile aspects of Mac OS X or iOS that could be affecting app performance, such as system calls, thread scheduling, and VM operations.
- Network Connections instrument—Inspect how your app is using TCP/IP and UDP/IP connections. With this instrument it is possible to see how much data is flowing through each connection and for each app. You can also use it to display interesting statistics, such as round trip times and retransmission request information.
- Network Activity (located in Energy Diagnostics)—Helps bridge the gap between networking (cellular and WiFi) and energy usage. You use it to display device-wide data flow through each network interface alongside energy usage level data that is taken directly from the battery.

For information about using these new instruments, see *Instruments New Features User Guide*

Additional Framework Enhancements

In addition to the items discussed in the preceding sections, the following frameworks have additional enhancements. For a complete list of new interfaces, see *iOS 5.0 API Diffs* .

UIKit

The UIKit framework (`UIKit.framework`) includes the following enhancements:

- The `UIViewController` class can now be used to create custom container view controllers; see "Container View Controller Support" (page 32).
- The UIKit framework provides support for loading and using storyboards; see "Storyboards" (page 25).
- Bars and bar button items can now be tinted and customized for your app; see "Custom Appearance for UIKit Controls" (page 32).
- The `UIPageViewController` class is a new container view controller for creating page-turn transitions between view controllers.
- The `UIReferenceLibraryViewController` class adds support for presenting a custom dictionary service to the user.
- The `UIImagePickerController` class supports new options for specifying the quality of video recordings.
- The `UIStepper` class is a new control for incrementing a floating-point value value up or down within a configurable range.
- View-based animations now support cross-dissolve, flip-from-top, and flip-from-bottom animations; see *UIView Class Reference* .
- The `UIApplication` class now reports the language directionality of the running app.
- The `UITableView` class adds support for automatic row animations, moving rows and sections, multiselection, and copy and paste behaviors for cells.
- The `UIScreen` class lets you specify overscan compensation behaviors for video delivered over AirPlay or through an attached HDMI cable. You can also programmatically set a screen's brightness.
- The `UIScrollView` class now exposes its gesture recognizers so that you can configure them more precisely for your app.
- The `UISegmentedControl` class now supports proportional segment widths.
- The `UIAlertView` class now supports password-style text entry and special configurations for entering text securely.
- The `UIColor` class includes support for Core Image and new methods to retrieve individual color values.
- The `UIImage` class includes support for Core Image, support for stretching an image by tiling part of its content, and support for looping animations.

- The `UITextInputTraits` protocol adds support for a Twitter-specific keyboard and separate spell-checking behavior.
- The `UIAccessibility` protocol includes new interfaces that define the activation point within an element and indicate whether an element is modal or contains hidden elements. There are also new notifications that inform you of changes in system-provided accessibility features, such as zoom, audio status, and closed captioning.
- The `UIAccessibilityReadingContent` protocol allows you to provide a continuous, page-turning reading experience to VoiceOver users.
- The `UIAccessibilityIdentification` protocol allows you to uniquely identify elements in your app so that you can refer to them in automation scripts.
- The `UIWebView` class automatically supports the presentation of multimedia content over AirPlay. You can opt out of this behavior by changing the value in the `mediaPlaybackAllowsAirPlay` property of the class. This class also exposes a `scrollView` property so that you can access the scrolling properties of your web interfaces.

For information about the classes of the UIKit framework, see *UIKit Framework Reference* .

OpenGL ES

OpenGL ES (`OpenGLES.framework`) now includes the following new extensions:

- The EXT_debug_label and EXT_debug_marker extensions allow you to annotate your OpenGL ES drawing code with information specific to your app. The OpenGL ES Performance Detective, the OpenGL ES Debugger, and the OpenGL ES Analyzer tools provided by Xcode all take advantage of these annotations.
- The EXT_color_buffer_half_float extension allows 16-bit floating point formats to be specified for a frame buffer's color renderbuffer.
- The EXT_occlusion_query_boolean extension allows your app to determine whether any pixels would be drawn by a primitive or by a group of primitives.
- The EXT_separate_shader_objects extension allows your app to specify separate vertex and fragment shader programs.
- The EXT_shadow_samplers extension provides support for shadow maps.
- The EXT_texture_rg extension adds one-component and two-component texture formats suitable for use in programmable shaders.

As always, check for the existence of an extension before using it in your app.

OpenAL

The OpenAL framework (`OpenAL.framework`) has two significant extensions:

- You can get notifications about source state changes and changes regarding the number of audio buffers that have been processed.
- The Apple Spatial Audio extension in iOS 5.0 adds three audio effects that are especially useful for games: reverberation, obstruction effects, and occlusion effects.

For information about the OpenAL interfaces, see the header files.

Message UI

The Message UI framework (`MessageUI.framework`) adds a new notification for tracking changes to the device's ability to send text messages. For information about the interfaces of the Message UI framework, see *Message UI Framework Reference* .

Media Player

The Media Player framework (`MediaPlayer.framework`) includes the following enhancements:

- There is now support for displaying "Now Playing" information in the lock screen and multitasking controls. This information can also be displayed on an Apple TV and with content delivered via AirPlay.
- You can detect whether video is being streamed to an AirPlay device using the `airPlayVideoActive` property of the `MPMoviePlayerController` class.
- Apps can now use the framework to play content from iTunes University.

For information about the classes in the Media Player framework, see *Media Player Framework Reference* .

Map Kit

The Map Kit framework (`MapKit.framework`) supports the ability to use heading data to rotate a map based on the user's current orientation. As you can with the Maps app, you can configure your map view to scroll the map according to the user's current location. For example, a walking tour app might use this to show the user their current location on the tour.

For information on the interfaces you use to implement map scrolling and rotation, see *Map Kit Framework Reference* .

Note If you are currently using Map Kit for geocoding, you should switch to using the Core Location framework for that feature; see "Core Location" (page 40).

iAd

The iAd framework (`iAd.framework`) provides new callback methods for developers who use multiple add networks and want to be notified when a new ad is available. The `bannerViewWillLoadAd:` method (defined in the `ADBannerViewDelegate` protocol) is called when a banner has confirmed that an ad is available but before the ad is fully downloaded and ready to be presented. The `interstitialAdWillLoad:` method (defined in the `ADInterstitialAdDelegate` protocol) offers similar behavior for interstitial ads.

For information about the classes of the iAd framework, see *iAd Framework Reference* .

Game Kit

The Game Kit framework (`GameKit.framework`) and Game Center now have the following features:

- The `GKTurnBasedMatch` class provides support for turn-based gaming, which allows games to create persistent matches whose state is stored in iCloud. Your game manages the state information for the match and determines which player must act to advance the state of the match.
- Your game can now adjust the default leaderboard (implemented by the `GKLeaderboard` class) shown to each player. If your game does not change the default leaderboard for a player, that player sees the leaderboard configured for your app in iTunes Connect.
- The `GKNotificationBanner` class implements a customizable banner similar to the banner shown to players when they log in to Game Center. Your game may use this banner to display messages to the player.
- When your game reports an achievement, it can automatically display a banner to the player using the `GKAchievement` class.
- A `GKMatchmakerViewController` object can now add players to an existing match in addition to creating a new match.
- The `GKMatchDelegate` protocol now includes a method to reconnect devices when a two-player match is disconnected.

For information abou the classes of the Game Kit framework, see *Game Kit Framework Reference* .

Foundation

The Foundation framework (`Foundation.framework`) includes the following enhancements:

- The NSFileManager class includes new methods for moving a file to a user's iCloud storage.
- The new NSFileCoordinator class and NSFilePresenter protocol implement now locking support and notifications when manipulating documents in iCloud.
- The new NSFileVersion class reports and manages conflicts between different versions of a file in iCloud.
- The NSURL class includes new methods and constants to support syncing items to a user's iCloud storage.
- The new NSMetadataQuery class supports attributes for items synced to a user's iCloud storage. Several other metadata-related classes were also added, including NSMetadataItem, NSMetadataQueryResultGroup, and NSMetadataQueryAttributeValueTuple.
- The new NSJSONSerialization class is a new class that supports back-and-forth conversions between JSON data and Foundation types.
- The new NSLinguisticTagger class is a new class lets you break down a sentence into its grammatical components, allowing the determination of nouns, verbs, adverbs, and so on. This tagging works fully for English and the class also provides a method to find out what capabilities are available for other languages.
- This framework now includes the NSFileWrapper class for managing file packages—that is, files implemented as an opaque directory.
- The new NSOrderedSet collection class offers the semantics of sets, whereby each element occurs at most once in the collection, but where elements are in a specific order.
- Most delegate methods are now declared using formal protocols instead of as categories on NSObject.

For information about the classes of the Foundation framework, see *Foundation Framework Reference* .

External Accessory

Apps that use the External Accessory framework to communicate with external accessories can now ask to be woken up if the app is suspended when its accessory delivers new data. Including the UIBackgroundModes key with the external-accessory value in your app's Info.plist file keeps your accessory sessions open even when your app is suspended. (Prior to iOS 5, these sessions were closed automatically at suspend time.) When new data arrives for a given session, a suspended app is woken up and given time to process the new data. This type of behavior is designed for apps that work with heart-rate monitors and other types of accessories that need to deliver data at regular intervals.

For more information about the UIBackgroundModes key, see *Information Property List Key Reference* . For information about interacting with external accessories, see *External Accessory Programming Topics* .

Event Kit and Event Kit UI

The Event Kit framework (EventKit.framework) includes the following enhancements:

- The class hierarchy has been restructured. There is now a common base class called `EKObject` and portions of the `EKEvent` class have been moved into a new base class called `EKCalendarItem`.
- With the `EKEventStore` class, you can now create and delete calendars programmatically, fetch calendars based on their identifier, save and remove events in batches, and trigger a programmatic refresh of calendar data.
- The new `EKSource` class represents the source for creating new calendars and events.
- The `EKCalendar` class now provides access to a calendar's UUID, source, and other attributes.

The Event Kit UI framework (`EventKitUI.framework`) now includes the `EKCalendarChooser` class, which provides a standard way for selecting from the user's iCal calendars.

For information about the classes of the Event Kit framework, see *Event Kit Framework Reference*. For information about the classes of the Event Kit UI framework, see *Event Kit UI Framework Reference*.

Core Motion

The Core Motion framework (`CoreMotion.framework`) now supports reporting heading information and magnetometer data for devices that have the corresponding hardware.

For information about the classes of the Core Motion framework, see *Core Motion Framework Reference*.

Core Location

The Core Location framework (`CoreLocation.framework`) now includes support for both forward and reverse geocoding location data. This support allows you to convert back and forth between a set of map coordinates and information about the street, city, country (and so on) at that coordinate.

For information about the classes of the Core Location framework, see *Core Location Framework Reference*.

Core Graphics

The Core Graphics framework (`CoreGraphics.framework`) includes some new interfaces to support the creation of paths. Specifically, there are new interfaces for creating paths with an ellipse or rectangle and for adding arcs to existing paths.

For more information about the Core Graphics interfaces, see *Core Graphics Framework Reference*.

Core Data

The Core Data framework includes the following enhancements:

- Core Data provides integration with the iOS document architecture and iCloud storage. The `UIManagedDocument` class is a concrete subclass of `UIDocument` that uses a Core Data persistent store for document data storage.
- For apps built for iOS 5.0 or later, persistent stores now store data by default in an encrypted format on disk. The default protection level prevents access to the data until after the user unlocks the device for the first time. You can change the protection level by assigning a custom value to the `NSPersistentStoreFileProtectionKey` key when configuring your persistent stores. For additional information about the data protection that are new in iOS 5.0, see "Data Protection Improvements" (page 31).
- Core Data formalizes the concurrency model for the `NSManagedObjectContext` class with new options. When you create a context, you can specify the concurrency pattern to use with it: thread confinement, a private dispatch queue, or the main dispatch queue. The `NSConfinementConcurrencyType` option provides the same behavior that was present on versions of iOS prior to 5.0 and is the default. When sending messages to a context created with a queue association, you must use the `performBlock:` or `performBlockAndWait:` method if your code is not already executing on that queue (for the main queue type) or within the scope of a `performBlock...` invocation (for the private queue type). Within the blocks passed to those methods, you can use the methods of `NSManagedObjectContext` freely. The `performBlockAndWait:` method supports API reentrancy. The `performBlock:` method includes an autorelease pool and calls the `processPendingChanges` method upon completion.
- You can create nested managed object contexts, in which the parent object store of a context is another managed object context rather than the persistent store coordinator. This means that fetch and save operations are mediated by the parent context instead of by a coordinator. This pattern has a number of usage scenarios, including performing background operations on a second thread or queue and managing discardable edits, such as in an inspector window or view

  Nested contexts make it more important than ever that you adopt the "pass the baton" approach of accessing a context (by passing a context from one view controller to the next) rather than retrieving it directly from the app delegate.
- Managed objects support two significant new features: ordered relationships, and external storage for attribute values. If you specify that the value of a managed object attribute may be stored as an external record, Core Data heuristically decides on a per-value basis whether it should save the data directly in the database or store a URI to a separate file that it manages for you.
- There are two new classes, `NSIncrementalStore` and `NSIncrementalStoreNode`, that you can use to implement support for nonatomic persistent stores. The store does not have to be a relational database—for example, you could use a web service as the back end.

For more details about the new features in Core Data, see *Core Data Release Notes for OS X v10.7 and iOS 5.0* .

Core Audio

The Core Audio family of frameworks includes the following changes in iOS 5.0:

- Audio-session routing information is now specified using dictionary keys. There are also new modes for managing your app's audio behavior:
    - Voice chat mode optimizes the system for two-way voice conversation.
    - Video recording mode configures the device for video capture.
    - Measurement mode disables automatic compression and limiting for audio input.
    - Default mode provides iOS 4.3.3 behavior.
- Core Audio adds seven new audio units for handling advanced audio processing features in your app, such as reverb, adjustable equalization, and time compression and stretching. The new Sampler unit lets you create music instruments, for which you can provide your own sounds. The new AUFilePlayer unit lets you play sound files and feed them directly to other audio units.
- The 3D Mixer audio unit is enhanced in iOS 5.0 to provide reverb and other effects useful in game audio.
- You can automate audio unit parameters in an audio processing graph, which lets you build a music mixer that remembers fader positions and changes.
- You can now use the advanced features of Apple Core Audio Format files in iOS. For example, you might create new voices for the Sampler audio unit.
- There is now programmatic support for adjusting the audio input gain.
- Core Audio now supports 32-bit floating-point audio data for apps that need to provide high quality audio.

For information about the audio technologies available in iOS, see *Multimedia Programming Guide*. For information about the new audio units, see *Audio Unit Component Services Reference*.

AV Foundation

The AV Foundation framework includes the following enhancements:

- There is automatic support for playing audio and video content over AirPlay. Apps can opt out of transmitting video over AirPlay using the `allowsAirPlayVideo` property of the `AVPlayer` class.
- New properties on the `AVPlayerItem` class indicate whether the item supports fast-forwarding or rewinding of the content.

For information about the classes of the AV Foundation framework, see *AV Foundation Framework Reference*.

Assets Library

The Assets Library framework includes the following enhancements:

- Support for accessing photo streams
- Support for creating new albums in the user's photo library
- Support for adding assets to albums
- The ability to get an aspect ratio thumbnail for an asset
- The ability to modify saved assets

For information about the classes of the Assets Library framework, see *Assets Library Framework Reference* .

Address Book

The Address Book framework adds support for importing and exporting vCard data. It also adds new keys to associate social network affiliations with a user record.

For more information about the new features in the Address Book framework, see *Address Book Framework Reference for iOS* .

Security

The Security framework (`Security.framework`) now includes the Secure Transport interfaces, which are Apple's implementation of the SSL/TLS protocols. You can use these interfaces to configure and manage SSL sessions, manage ciphers, and manage certificates.

For information about the Secure Transport interfaces, see the `SecureTransport.h` header file of the Security framework.

iOS 4.3

This article summarizes the key developer-related features introduced in iOS 4.3. This version of the operating system runs on all iOS-based devices. In addition to describing the key new features, this article lists the documents that describe those features in more detail.

For the latest updates and information, you should also see *iOS 4.3 Release Notes* . For the complete list of new APIs added in iOS 4.3, see *iOS 4.3 API Diffs* .

AirPlay Video Support

Support for playing video using AirPlay is included in the `MPMoviePlayerController` class. This support allows you to play video-based content on AirPlay–enabled hardware such as Apple TV. When the `allowsAirPlay` property of an active `MPMoviePlayerController` object is set to `YES` and the device is in range of AirPlay–enabled hardware, the movie player presents the user with a control for sending the video to that hardware. (That property is set to `NO` by default.) Supported formats include:

- H.264 video with AAC audio
- HTTP streaming, both live and on demand
- progressive download content
- local content

For web-based content, you can enable AirPlay Video in the QuickTime Plug-in or HTML5 video element as follows:

- QTPlug-in:
  - `airplay="allow"`
  - `airplay="deny"` (Default)

  For example: `<embed src="movie.mov" width="320" height="240" airplay="allow">`
- HTML5 video element:
  - `x-webkit-airplay="allow"`
  - `x-webkit-airplay="deny"` (Default)

For example: `<video controls width="640" height="368" x-webkit-airplay="allow" src="content/side_with_the_seeds.mov"> </video>`

For more information about using the `MPMoviePlayerController` class to play video, see *MPMoviePlayerController Class Reference*.

Framework Enhancements

The following sections highlight the significant changes to frameworks and technologies in iOS 4.3. For a complete list of all new interfaces available in the system, see *iOS 4.3 API Diffs*.

AV Foundation

The AV Foundation framework includes the following enhancements:

- The AV Foundation framework added the `AVPlayerItem`, `AVPlayerItemAccessLogEvent`, and `AVPlayerItemErrorLogEvent` classes for tracking network playback statistics.
- The `AVMetadataItem` class added support for loading key data asynchronously.

For information about the classes of the UIKit framework, see *AV Foundation Framework Reference*.

Core Audio Frameworks

The Audio Unit and Audio Toolbox frameworks include the following enhancements:

- The `AudioUnitParameterHistoryInfo` struct (in the Audio Unit framework) along with supporting audio unit properties adds the ability to track and use parameter automation history.
- The `ExtendedAudioFormatInfo` struct (in the Audio Toolbox framework) lets you specify which codec to use when accessing the `kAudioFormatProperty_FormatList` property.
- The `kAFInfoDictionary_SourceBitDepth` dictionary key and the `kAudioFilePropertySourceBitDepth` property (in the Audio Toolbox framework) provide access to the bit depth of an audio stream.
- The `kAudioConverterErr_NoHardwarePermission` result code (in the Audio Toolbox framework) indicates that a request to create a new audio converter object cannot be satisfied because the application does not have permission to use the requested hardware codec.

For information about the functions and types of the Audio Unit framework, see *Audio Unit Framework Reference*. For information about the functions and types of the Audio Toolbox framework, see *Audio Toolbox Framework Reference*.

iAd

The `ADInterstitialAd` class is controller that you can use to present full-screen banners in your content. You present these banners in an existing view or as part of a transition from one page of content to another. For example, you might use this object to incorporate full-page ads into a page-based magazine layout.

For more information about the classes of the iAd framework, see *iAd Framework Reference* .

Media Player

The Media Player framework includes the following enhancements:

- The `MPMoviePlayerController` class supports playback of video content using AirPlay; see "AirPlay Video Support" (page 44).
- The `MPMovieAccessLog`, `MPMovieErrorLog`, `MPMovieAccessLogEvent`, and `MPMovieErrorLogEvent` classes allow you to track network playback statistics.
- The `MPMoviePlayerController` class now includes properties for accessing log information.

For more information about the classes of the Media Player framework, see *Media Player Framework Reference* .

UIKit

The `UIViewController` class added the `disablesAutomaticKeyboardDismissal` method, which you can use to override the default input view dismissal behavior.

For more information about the classes of the UIKit framework, see *UIKit Framework Reference* .

iOS 4.2

This article summarizes the key developer-related features introduced in iOS 4.2. This version of the operating system runs on all iOS-based devices. In addition to describing the key new features, this article lists the documents that describe those features in more detail.

For the latest updates and information, you should also see *iOS 4.2 Release Notes* . For the complete list of new APIs added in iOS 4.2, see *iOS 4.2 API Diffs* .

Printing

iOS now incorporates support for wireless printing from iPhone and iPad applications. For the most part, the objects provided by UIKit do all of the heavy lifting associated with printing. They manage the printing interfaces, work with your application to render the printable content, and handle the scheduling and execution of print jobs on the printer.

Print jobs submitted by your application are handed off to the printing system, which manages the actual printing process. Print jobs from all applications on a device are queued and printed on a first-come, first-served basis. Users can get the status of print jobs from the Print Center application and can even use that application to cancel print jobs. All other aspects of printing are handled for you automatically by the system.

Note Wireless printing is available only on devices that support multitasking. You can use the `UIPrintInteractionController` object to detect whether printing is available in your application.

To simplify the printing process, UIKit provides automatic support for some types of content to make printing easier. Specifically, applications provide printable content in one of three ways:

- An application can hand images or PDF content directly to UIKit and let it manage the printing of those items.
- An application can use print formatter objects to lay out specific types of content over multiple pages. UIKit provides print formatters for plain text, HTML text, and some system views (web views, text views, and map views).
- An application can choose to render the content itself using a print page renderer object. This option gives you the most control over your printed content but also lets you use print formatters to assist with the layout of parts of your content.

For information about how to incorporate the new printing support into your applications, see "Printing" in *Drawing and Printing Guide for iOS* .

AirPlay

AirPlay is a technology that lets your application stream audio to Apple TV and to third-party AirPlay speakers and receivers. AirPlay support is built in to the AV Foundation framework and the Core Audio family of frameworks. Any audio content you play using these frameworks is automatically made eligible for AirPlay distribution. Once the user chooses to play your audio using AirPlay, it is routed automatically by the system.

For information on how to play audio content, see *Multimedia Programming Guide* .

Core MIDI

The Core MIDI framework (`CoreMIDI.framework`) provides a standard way to communicate with MIDI devices, including hardware keyboards and synthesizers. You use this framework to send and receive MIDI messages and to interact with MIDI peripherals connected to an iOS-based device using the dock connector or network.

For more information about using this framework, see *Core MIDI Framework Reference* .

Weak Linking Support

Applications that link against iOS SDK 4.2 or later can now take advantage of new support for determining the availability of Objective-C classes. System frameworks now tag their classes with availability information that the compiler can use to link weakly to those classes as needed. This new tagging mechanism simplifies the code you need to use to do runtime checking for the availability of the class. Whereas previously, you needed to use the `NSClassFromString` function to determine whether a class was present, now you can simply check for the existence the class directly using code similar to the following:

```
if ([UIPrintInteractionController class])
{
   // Create an instance of the class and use it.
}
else
{
   // The print interaction controller is not available.
}
```

In order to use this feature, you must do the following:

- Build your project using LLVM and Clang. (The standalone GCC compiler does not currently support this feature.)
- Set your project's deployment target to iOS 3.1 or later.
- Change the linking option for each framework you want to link weakly. To do this, open an inspector for your application target and go to the inspector's General tab to view the list of frameworks and libraries linked against the target. For frameworks you want to link weakly, change the value in the Type column from Required to Weak.

  If the deployment target for your application is set to a version of iOS that does not contain a given framework (because it was introduced in a later version), set the link type of that framework to Weak. For most other frameworks, you should leave the link type set to Required.

For information about how to perform runtime availability checks for classes and methods, see *SDK Compatibility Guide* .

Framework Enhancements

The following sections highlight the significant changes to frameworks and technologies in iOS 4.2. For a complete list of all new interfaces available in the system, see *iOS 4.2 API Diffs* .

Core Location

The Core Location framework includes new interfaces for determining whether a device is authorized to use location services and for detecting changes in the current authorization status. For more information about how to use Core Location in your application, see *Location Awareness Programming Guide* .

Game Kit

The Game Kit framework includes a new `GKFriendRequestComposeViewController` class, which you can present from your application and use to invite friends to play a game. For more information about how to use Game Kit in your application, see *Game Kit Programming Guide* .

iAd

The iAd framework now supports banner ads sized appropriately for iPad devices. For more information about how to incorporate ads into your applications, see *iAd Programming Guide* .

Media Player

The Media Player framework includes the following enhancements:

- The interface presented by the `MPVolumeView` class now includes a control for routing audio content to AirPlay–enabled devices.
- The `MPMediaItem` and `MPMediaItemCollection` classes now descend from a common ancestor class: `MPMediaEntity`. This change in hierarchy allows you to create collections that include both items and other collections.
- Persistent IDs for more collection types are now available.
- Media queries now provide more information about how media items are grouped together.

For more information about the classes of the Media Player framework, see *Media Player Framework Reference* .

UIKit

The UIKit framework includes the following enhancements

- New accessibility constants allow "scroll by page" capabilities using VoiceOver.
- New classes to support printing; see "Printing" (page 47)
- The `UIDevice` class now provides the `playInputClick` method for playing the input-click sound directly.
- Application delegates can now use the `application:openURL:sourceApplication:annotation:` method to receive information about URLs that are opened while the application is running in the background.
- The `UITextInputMode` class now exposes the language in use for inputting text.
- The `UIDocumentInteractionController` class supports printing with the help of its delegate object.

For more information about the classes of the UIKit framework, see *UIKit Framework Reference* .

iOS 4.1

This article summarizes the developer-related features introduced in iOS 4.1. This version of the operating system runs on iPhone and iPod touch only and does not run on iPad. In addition to describing the new features, this article lists the documents that describe those features in more detail.

Note iOS 4.1 does not support iPad. It runs only on iPhone and iPod touch devices. You must use a different version of the iOS SDK to develop iPad applications.

For the latest updates and information, you should also see *iOS 4.1 Release Notes* . For the list of API differences between the iOS 4.1 and earlier versions of iOS, see *iOS 4.1 API Diffs* .

Game Center

Game Center is an extension to the Game Kit framework that provides support for the following features:

- Aliases allow users to create their own online persona. Users log in to Game Center and interact with other players anonymously through their alias. Players can set status messages as well as mark specific people as their friends.
- Leaderboards allow your application to post user scores to Game Center and retrieve them later. You might use this feature to show the best scores among all users of your application.
- Matchmaking allows you to create multiplayer games by connecting players who are logged into Game Center. Players do not have to be local to each other to join a multiplayer game.
- Achievements allows you to record the progress a player has made in your game.

The preceding features all contribute towards the experience users see in the Game Center application. For information about the classes of the Game Kit framework, see *Game Kit Framework Reference* .

Framework Enhancements

The following existing frameworks and technologies include additional incremental changes. For a complete list of new interfaces, see *iOS 4.1 API Diffs* .

AV Foundation

The AV Foundation framework includes the following enhancements:

- A new `AVQueuePlayer` class for playing a sequence of `AVPlayerItem` objects.
- A new `AVAssetReader` class for reading samples from media files.
- A new `AVAssetWriter` class for writing samples to a data file and optionally compressing them.

For more information about the classes of the AV Foundation framework, see *AV Foundation Framework Reference*.

Assets Library

The Assets Library framework includes new methods to save images to a user’s photo album. These new methods are declared in the `ALAssetsLibrary` class.

Core Text

The Core Text framework includes new functions for managing fonts. For more information about the functions available in the Core Text framework, see *Core Text Reference Collection*.

System Configuration

The System Configuration framework contains new interfaces for identifying captive networks. For more information about the functions available in the System Configuration framework, see *System Configuration Framework Reference*.

UIKit

The UIKit framework includes the following enhancements:

- Additional constants for the `UIImagePickerController` class that allow you to obtain additional information about an image.
- A decimal keyboard type that you can use for text entry. You can specify this keyboard type using the `UIKeyboardTypeDecimalPad` constant.

For more information about the classes of the UIKit framework, see *UIKit Framework Reference*.

iOS 4.0

This article summarizes the developer-related features introduced in iOS 4. This version of the operating system runs on iPhone and iPod touch only and does not run on iPad. In addition to describing the new features, this article lists the documents that describe those features in more detail.

> **Note** iOS 4 does not support iPad. It runs only on iPhone and iPod touch devices. You must use a different version of the iOS SDK to develop iPad applications.

For the latest updates and information, you should also see *iOS 4.0 Release Notes* . For the list of API differences between the iOS 4 and earlier versions of iOS, see *iOS 4.0 API Diffs* .

Multitasking

Applications built using iOS SDK 4 or later (and running in iOS 4 and later) are no longer terminated when the user presses the Home button; instead, they now shift to a background execution context. For many applications, this means that the application enters a suspended state of execution shortly after entering the background. Keeping the application in memory avoids the subsequent launch cycle and allows an application to simply reactivate itself, which improves the overall user experience. And suspending the application improves overall system performance by minimizing power usage and giving more execution time to the foreground application.

Although most applications are suspended shortly after moving to the background, applications that need to continue working in the background may do so using one of the following techniques:

- An application can request a finite amount of time to complete some important task.
- An application can declare itself as supporting specific services that require regular background execution time.
- An application can use local notifications to generate user alerts at designated times, whether or not the application is running.

Regardless of whether your application is suspended or continues running in the background, supporting multitasking does require some additional work on your part. Background applications can still be terminated under certain conditions (such as during low-memory conditions), and so applications must be ready to exit at any time. This means that many of the tasks you used to perform at quit time must now be performed when your application moves to the background. This requires implementing some new methods in your application delegate to respond to application state transitions.

For more information on how to handle the new background state transitions, and for information on how to continue running in the background, see *iOS App Programming Guide*.

Integration Technologies

The following sections describe the technologies you can use to enhance your application's user experience.

Local Notifications

Local notifications complement the existing push notifications by giving applications an avenue for generating the notifications locally instead of relying on an external server. Background applications can use local notifications as a way to get a user's attention when important events happen. For example, a navigation application running in the background can use local notifications to alert the user when it is time to make a turn. Applications can also schedule the delivery of local notifications for a future date and time and have those notifications delivered even if the application is not running.

The advantage of local notifications is that they are independent of your application. Once a notification is scheduled, the system manages the delivery of it. Your application does not even have to be running when the notification is delivered.

For more information about using local notifications, see *Local and Push Notification Programming Guide*.

Event Kit

The Event Kit framework (`EventKit.framework`) provides an interface for accessing calendar events on a user's device. You can use this framework to get existing events and add new events to the user's calendar. Calendar events can include alarms that you can configure with rules for when they should be delivered. In addition to using Event Kit for creating new events, you can use the view controllers of the Event Kit UI framework (`EventKitUI.framework`) to present standard system interfaces for viewing and editing events.

For more information about the classes and methods of these frameworks, see *Event Kit Framework Reference* and *Event Kit UI Framework Reference*.

Core Motion

The Core Motion framework (`CoreMotion.framework`) provides a single set of interfaces for accessing all motion-based data available on a device. The framework supports accessing both raw and processed accelerometer data using a new set of block-based interfaces. For devices with a built-in gyroscope, you can retrieve the raw gyro data as well as processed data reflecting the attitude and rotation rates of the device. You can use both the accelerometer and gyro-based data for games or other applications that use motion as input or as a way to enhance the overall user experience.

For more information about the classes and methods of this framework, see "Motion Events" in *Event Handling Guide for iOS* .

Data Protection

Applications that work with sensitive user data can now take advantage of the built-in encryption available on some devices to protect that data. When your application designates a particular file as protected, the system stores that file on-disk in an encrypted format. While the device is locked, the contents of the file are inaccessible to both your application and to any potential intruders. However, when the device is unlocked by the user, a decryption key is created to allow your application to access the file.

Implementing data protection requires you to be considerate in how you create and manage the data you want to protect. Applications must themselves be designed to secure the data at creation time and to be prepared for changes in access to that data when the user locks and unlocks the device.

For more information about how to add data protection to the files of your application, see "App States and Multitasking" in *iOS App Programming Guide* .

Core Telephony

The Core Telephony framework (`CoreTelephony.framework`) provides interfaces for interacting with phone-based information on devices that have a cellular radio. Applications can use this framework to get information about a user's cellular service provider. Applications interested in cellular call events can also be notified when those events occur.

For more information about using the classes and methods of this framework, see *Core Telephony Framework Reference* .

iAd

You can use iAd (`iAd.framework`) to deliver banner-based advertisements from your application. Advertisements are incorporated into standard views that you integrate into your user interface and present when you want. The views themselves work with Apple's ad service to automatically handle all the work associated with loading and presenting the ad content and responding to taps in those ads.

For more information about using iAd in your applications, see *iAd Framework Reference*.

Graphics and Multimedia

The following sections describe the new graphics and media-related technologies you can incorporate into your applications.

High-Resolution Screen Support

For the most part, applications running on devices with high-resolution screens should work with little or no modifications. The coordinate values you specify during drawing or when manipulating views are all mapped to a logical coordinate system, which is decoupled from the underlying screen resolution. Any content you draw is automatically scaled as needed to support high-resolution screens. For vector-based drawing code, the system frameworks automatically use any extra pixels to improve the crispness of your content. And if you use images in your application, UIKit provides support for loading high-resolution variants of your existing images automatically.

For detailed information about how to support high-resolution screens, see "Supporting High-Resolution Screens" in *Drawing and Printing Guide for iOS*.

Quick Look Framework

The Quick Look framework (`QuickLook.framework`) provides a direct interface for previewing the contents of files your application does not support directly. This framework is intended primarily for applications that download files from the network or that otherwise work with files from unknown sources. After obtaining the file, you use the view controller provided by this framework to display the contents of that file directly in your user interface.

For more information about the classes and methods of this framework, see *Quick Look Framework Reference for iOS*.

AV Foundation

The AV Foundation framework (`AVFoundation.framework`) is for applications that need to go beyond the music and movie playback features found in the Media Player framework. Originally introduced in iOS 3.0, this framework has been expanded in iOS 4 to include significant new capabilities, substantially broadening its usage beyond basic audio playback and recording capabilities. Specifically, this framework now includes support for the following features:

- Media asset management
- Media editing
- Movie capture
- Movie playback
- Track management
- Metadata management for media items
- Stereophonic panning
- Precise synchronization between sounds
- An Objective-C interface for determining details about sound files, such as the data format, sample rate, and number of channels

The AV Foundation framework is a single source for recording and playing back audio and video in iOS. This framework also provides much more sophisticated support for handling and managing media items.

For more information about the classes and methods of the AV Foundation framework, see *AV Foundation Framework Reference* .

Assets Library

The Assets Library framework (`AssetsLibrary.framework`) provides a query-based interface for retrieving a user's photos and videos. Using this framework, you can access the same assets that are nominally managed by the Photos application, including items in the user's saved photos album and any photos and videos that were imported onto the device. You can also save new photos and videos back to the user's saved photos album.

For more information about the classes and methods of this framework, see *Assets Library Framework Reference* .

Image I/O

The Image I/O framework (`ImageIO.framework`) provides interfaces for importing and exporting image data and image metadata. This framework is built on top of the Core Graphics data types and functions and supports all of the standard image types available in iOS.

For more information about the functions and data types of this framework, see *Image I/O Reference Collection* .

Core Media

The Core Media framework (`CoreMedia.framework`) provides the low-level media types used by AV Foundation. Most applications should never need to use this framework, but it is provided for those few developers who need more precise control over the creation and presentation of audio and video content.

For more information about the functions and data types of this framework, see *Core Media Framework Reference* .

Core Video

The Core Video framework (`CoreVideo.framework`) provides buffer and buffer pool support for Core Media. Most applications should never need to use this framework directly.

Core Services

The following sections describe the new lower-level technologies and features you can incorporate into your applications.

Block Objects

Block objects are a C-level language construct that you can incorporate into your C and Objective-C code. A block object is essentially an anonymous function and the data that goes with that function, something which in other languages is sometimes called a *closure* or *lambda* . Blocks are particularly useful as callbacks or in places where you need a way of easily combining both the code to be executed and the associated data.

In iOS, blocks are commonly used in the following scenarios:

- As a replacement for delegates and delegate methods
- As a replacement for callback functions
- To implement completion handlers for one-time operations
- To facilitate performing a task on all the items in a collection

- Together with dispatch queues, to perform asynchronous tasks

For an introduction to block objects and how you use them, see *A Short Practical Guide to Blocks* . For more information about blocks, see *Blocks Programming Topics* .

Grand Central Dispatch

Grand Central Dispatch (GCD) is a BSD-level technology that you use to manage the execution of tasks in your application. GCD combines an asynchronous programming model with a highly optimized core to provide a convenient (and more efficient) alternative to threading. GCD also provides convenient alternatives for many types of low-level tasks, such as reading and writing file descriptors, implementing timers, monitoring signals and process events, and more.

For more information about how to use GCD in your applications, see *Concurrency Programming Guide* . For information about specific GCD functions, see *Grand Central Dispatch (GCD) Reference* .

Accelerate Framework

The Accelerate framework (`Accelerate.framework`) contains interfaces for performing math, big-number, and DSP calculations, among others. The advantage of using this framework over writing your own versions of these libraries is that it is optimized for the different hardware configurations present in iOS–based devices. Therefore, you can write your code once and be assured that it runs efficiently on all devices.

For more information about the functions of the Accelerate framework, see *Accelerate Framework Reference* .

Xcode Tools

The following sections describe the improvements to the Xcode tools and the support for developing iOS applications.

Xcode Improvements

Xcode 3.2.3 introduces automatic device and provisioning-profile management in the Organizer window. With automatic device provisioning enabled, you can install applications on your device for debugging and testing without having to log in to your team portal to register the device and download a provisioning profile

> **Note** You still need to log in to your team portal to create provisioning profiles with specific application IDs for in-app purchase and push notifications. However, once created, those provisioning profiles will also be managed by Xcode if automatic device provisioning is enabled.

For more information about using Xcode, see *Tools Workflow Guide for iOS* .

UI Automation API

The Instruments application now provides support for automating the testing of your iOS applications. The built-in Automation instrument works from scripts (written in JavaScript) that you provide to drive the simulation of events in your application. These synthetic events are generated with the help of the accessibility interfaces built into iOS and integrated into all existing UIKit views. You can use this instrument to improve your testing process and deliver more robust applications.

For information about how to use the Automation instrument, see *Instruments User Guide* . For information about the JavaScript objects and commands you use in your scripts, see *UI Automation Reference Collection* .

Framework Enhancements

The following existing frameworks and technologies include additional incremental changes. For a complete list of new interfaces, see *iOS 4.0 API Diffs* .

UIKit Framework Enhancements

The UIKit framework includes the following enhancements:

- The `UIApplication` class and `UIApplicationDelegate` protocol include new methods for scheduling local notifications and for supporting multitasking.
- Drawing to a graphics context in UIKit is now thread-safe. Specifically:
    - The routines used to access and manipulate the graphics context can now correctly handle contexts residing on different threads.
    - String and image drawing is now thread-safe.
    - Using color and font objects in multiple threads is now safe to do.
- The `UIImagePickerController` class includes methods for programmatically starting and stopping video capture. It also includes options for selecting which camera you want to use on a device and for enabling a built-in flash.

- The `UILocalNotification` class supports the configuration of local notifications; see "Local Notifications" (page 54).
- The `UIView` class includes new block-based methods for implementing animations.
- The `UIWindow` class has a new `rootViewController` property that you can use to change the contents of the window.
- Media applications can now receive events related to the controls on an attached set of headphones. You can use these events to control the playback of media-related items.
- Several new accessibility interfaces help you make some UI elements more accessible and allow you to customize your application experience specifically for VoiceOver users:
  - The `UIAccessibilityAction` protocol makes it easy for VoiceOver users to adjust the value of UI elements, such as pickers and sliders.
  - `UIPickerViewAccessibilityDelegate` protocol enables access to the individual components of a picker.
  - `UIAccessibilityFocus` protocol allows you to find out when VoiceOver is focused on an element, so you can help users avoid making unnecessary taps.
  - The `UIAccessibilityTraitStartsMediaSession` trait allows you to prevent VoiceOver from speaking during a media session that should not be interrupted.
  - New interfaces in `UIAccessibility` protocol allow you to specify the language in which labels and hints are spoken, and provide announcements that describe events that don't update application UI in way that would be perceptible to VoiceOver users.
- The `UINib` class provides a way to instantiate multiple sets of objects efficiently from the same nib file.

For information about the classes of the UIKit framework, see *UIKit Framework Reference*.

Foundation Framework Enhancements

The Foundation framework includes the following enhancements:

- Most delegate methods are now declared in formal protocols instead of as categories on `NSObject`.
- Block-based variants are now available for many types of operations.
- There is new support for creating and formatting date information in `NSDate` and `NSDateFormatter`.
- The `NSDateComponents` class added support for specifying time zone and quarter information.
- There is support for regular-expression matching using the `NSRegularExpression`, `NSDataDetector`, and `NSTextCheckingResult` classes.
- The `NSBlockOperation` class allows you to add blocks to operation queues.
- You can use the `NSFileManager` class to mark files as protected; see "Data Protection" (page 55).

- The `NSFileWrapper` class allows you to work with package-based document types.
- The `NSOrthography` class describes the linguistic content of a piece of text.
- The `NSCache` class provides support for storing and managing temporary data.
- The URL-related classes have been updated so that you can now pipeline URL requests and set request priorities.

For information about the classes of the Foundation framework, see *Foundation Framework Reference* .

OpenGL ES Enhancements

The OpenGL ES framework includes the following enhancements:

- The `APPLE_framebuffer_multisample` extension enables full-scene anti-aliasing.
- The `EXT_framebuffer_discard` extension can be used to improve the performance of applications that use depth buffers or multisample framebuffers.
- The `APPLE_texture_max_level` and `EXT_shader_texture_lod` extensions provide more control over texture sampling.
- The `OES_vertex_array_object` (http://www.khronos.org/registry/gles/extensions/OES/OES_vertex_array_object.txt) API allows caching of vertex array state, to decrease driver overhead.
- The `OES_depth_texture` extension enables rendering real-time shadows using shadow maps.
- The `OES_texture_float` (http://www.khronos.org/registry/gles/extensions/OES/OES_texture_float.txt) and `OES_texture_half_float` (http://www.khronos.org/registry/gles/extensions/OES/OES_texture_float.txt) extensions adds texture formats with floating point components to enable High Dynamic Range rendering.
- The `APPLE_rgb_422` (http://www.opengl.org/registry/specs/APPLE/rgb_422.txt) extension enables texturing from some common video formats.
- Performance of texture creation and modification has been significantly improved.
- Driver performance has been generally improved.

Game Kit Enhancements

The Game Kit framework includes a beta implementation of a centralized service called Game Center. This service provides game developers with a standard way to implement the following features:

- Aliases allow users to create their own online persona. Users log in to Game Center and interact with other players anonymously through their alias. Players can set status messages as well as mark specific people as their friends.

- Leader boards allow your application to post scores to Game Center and retrieve them later.
- Matchmaking allows players to connect with other players with Game Center accounts.

Important GameCenter is available to developers only in iOS 4. It is introduced as a developer-only feature so that you can provide feedback as you implement and test Game Center features in your applications. However, Game Center is not a user feature in iOS 4 and you should not deploy applications that use it to the App Store.

For information about the classes of the Game Kit framework, see *Game Kit Framework Reference* .

Core Location Enhancements

The Core Location framework now supports the following features:

- A location monitoring service that tracks significant changes using only cellular information. This solution offers a lower-power alternative for determining the user's location.
- The ability to define arbitrary regions and detect boundary crossings into or out of those regions. This feature can be used for proximity detection regardless of whether the application is running.

For information about the classes of the Core Location framework, see *Core Location Framework Reference* .

Map Kit Enhancements

The Map Kit framework includes the following enhancements:

- Support for draggable map annotations
- Support for map overlays

Draggable map annotations make it much easier to reposition those annotations after they have been added to a map. The Map Kit framework handles most of the touch events associated with initiating, tracking, and ending a drag operation. However, the annotation view must work in conjunction with the map view delegate to ensure that dragging of the annotation view is supported.

Map overlays provide a way to create more complex types of annotations. Instead of being pinned to a single point, an overlay can represent a path or shape that spans a wider region. You can use overlays to layer information such as bus routes, election maps, park boundaries, and weather maps on top of the map.

For information about the functions and types of the Map Kit framework, see *Map Kit Framework Reference* .

Message UI Enhancements

The Message UI framework includes a new `MFMessageComposeViewController` class for composing SMS messages. This class manages a standard system interface for composing and sending SMS messages. In contrast with sending SMS messages using a specially formatted URL, this class allows you to create and send the message entirely from within your application.

For more information about the classes of the Message UI framework, see *Message UI Framework Reference* .

Core Graphics Enhancements

The Core Graphics framework includes the following enhancements:

- The ability to embed metadata into PDF files using the `CGPDFContextAddDocumentMetadata` function
- Support for creating color spaces using an ICC profile
- Graphics context support for font smoothing and fine-grained pixel manipulation

For information about the functions and types of the Core Graphics framework, see *Core Graphics Framework Reference* .

ICU Enhancements

The International Components for Unicode (ICU) libraries were updated to version 4.4. ICU is an open-source project for Unicode support and software internationalization. The installed version of ICU includes only a subset of the header files that are part of the broader ICU library. Specifically, iOS includes only the headers used to support regular expressions.

For more information about using the functions of the ICU 4.4 library, see the documentation at http://site.icu-project.org/.

Inherited Improvements

Although iOS 3.2 does not run on iPhone and iPod touch devices, many of the features introduced in that version of the operating system are also supported in iOS 4. Specifically, iOS 4 supports:

- Custom input views
- Connecting external displays
- File-sharing support
- Gesture recognizers

- Core Text for text layout and rendering
- Text input through integration with the keyboard
- Custom fonts
- ICU Regular Expressions
- Document types
- PDF generation
- Xcode Tools changes
- UIKit framework changes
- Media Player framework changes
- Core Animation changes
- Foundation framework changes

Document Revision History

This table describes the changes to *What's New in iOS* .

Date	Notes
2012-06-13	Updated to include features introduced in iOS 6. Removed older articles containing changes in iOS 3.x.
2012-03-07	Updated to include features introduced in iOS 5.1.
2011-10-12	Updated to include features introduced in iOS 5.0.
2011-02-28	Added features introduced in iOS 4.3.
2010-11-15	Added features introduced in iOS 4.2.
2010-08-17	Added features introduced in iOS 4.1.
2010-07-08	Changed the title from "What's New in iPhone OS."
2010-06-04	Added information about new features in iOS 4.0.
2010-03-24	Moved the iOS 3.1 information to its own article and added a new article covering features in iOS 3.2.
2009-08-27	Updated the iOS 3.0 article to reflect features introduced in all 3.x versions of iOS.
2009-06-16	Added new features related to the introduction of new iPhone hardware.