What's New in iOS

Developer

Contents

Figures and Listings

Introduction

Important: This is a preliminary document for an API or technology in development. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein for use on Apple-branded products. 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 betas of the API or technology.

This document describes developer-level features that were introduced in different versions of iOS. For each shipping release, this document provides links to "delta" reference documents, which list the new and changed programming interfaces that were introduced in that release.

Organization of This Document

This document includes the following articles:

- iOS 9.0 (page 10) describes the new and updated features in iOS 9.
- iOS 8.4 (page 30) describes the new and updated features in iOS 8.4.
- iOS 8.3 (page 31) describes the new and updated features in iOS 8.3.
- iOS 8.2 (page 32) describes the new and updated features in iOS 8.2.
- iOS 8.1 (page 34) describes the new and updated features in iOS 8.1.
- iOS 8.0 (page 35) describes the new and updated features in iOS 8.
- iOS 7.1 (page 53) describes the new and updated features in iOS 7.1.
- iOS 7.0 (page 55) describes the new and updated features in iOS 7.
- iOS 6.1 (page 72) describes the new and updated features in iOS 6.1.
- iOS 6.0 (page 73) describes the new and updated features in iOS 6.
- iOS 5.1 (page 85) describes the new and updated features in iOS 5.1.
- iOS 5.0 (page 86) describes the new and updated features in iOS 5.

iOS 9.0

This article summarizes the key developer-related features introduced in iOS 9, which runs on currently shipping iOS devices. The article also lists the documents that describe new features in more detail.

For late-breaking news and information about known issues, see *iOS 9 Release Notes* . For the complete list of new APIs added in iOS 9, see *iOS 9.0 API Diffs* . For more information on new devices, see *iOS Device Compatibility Reference* .

Multitasking Enhancements for iPad

iOS 9 enhances the user's multitasking experience on iPad with Slide Over, Split View, and Picture in Picture. The Slide Over feature lets users pick a secondary app and quickly interact with it. The Split View feature gives users the ability to use two apps side by side on iPad Air 2. The Picture in Picture feature (also known as PiP) lets users watch video in a window that floats above other onscreen apps.

Users decide when they want to view two apps on the screen at the same time; you have no control over when this happens. Even though Split View and Slide Over are user-controlled, there are a few things you need to do to ensure that users have a great multitasking experience.

- It's crucial that your app use system resources efficiently so that it can run well when it shares the system with another running app. Under memory pressure, the system preemptively quits the app that's consuming the most memory.
- If you haven't already, be sure to adopt size classes so that your app looks good when the user decides to view it in a portion of the device screen.

To learn more about preparing your app to respond well when users use Split View and Slide Over, see *Adopting Multitasking Enhancements on iPad* .

As with Split View and Slide Over, users control whether they want to use PiP to view a video on top of another running app. If video playback is not your app's primary functionality, there's nothing you need to do to support the PiP experience.

To participate when users choose Picture in Picture, use AVKit or AV Foundation APIs. The video playback classes defined in the Media Player framework are deprecated in iOS 9 and do not support PiP. To learn how to prepare your video playback app for PiP, see Picture in Picture Quick Start.

Search

App Search in iOS 9 gives users great new ways to access information inside of your app, even when it isn't installed. When you adopt iOS 9 Search, users can access activities and content deep within your app through Handoff, Siri Reminders, and Search results.

In addition to improving the user experience, adopting Search helps you increase the usage of your app and improve its discoverability by displaying your content when users search across the system and on the web. For an example of how this works, imagine that your app helps users handle minor medical conditions, such as a sunburn or a sprained ankle. When you adopt iOS 9 Search, users searching their devices for "sprained ankle" can get results for your app even when they don't have your app installed. When users tap on a result for your app, they get the opportunity to download your app. Similarly, users can get results for your app and related web content when they search for "sprained ankle" in Safari. Tapping on a result in Safari takes users to your website, where they can download your app from your App Banner.

Adopting iOS 9 Search is easy: You don't need any prior experience with implementing search, and most developers find that it takes only a few hours to make their content universally searchable. iOS 9 introduces the following APIs you can use to adopt Search:

- The `NSUserActivity` class includes new methods and properties that help you index activities and app states to make them available in search results. Just about every app can take advantage of the `NSUserActivity` APIs to make useful content available to users. For more details, see Use NSUserActivity APIs to Make App Activities Searchable (page 12).
- The new Core Spotlight framework (`CoreSpotlight.framework`) provides APIs that help you index the content in your app and enable deep links to that content. Core Spotlight is designed for apps that handle persistent user data, such as documents, photos, and other types of content created by or on behalf of users. To learn more about this framework, see Use Core Spotlight APIs to Make App Content Searchable (page 13).
- Adding the appropriate web markup to your website makes your related web content searchable and helps you enrich the user's search results. Adding a Smart App Banner gives users an easy way to link directly to your app. For more information about ways to make your web content searchable, see Use Web Markup to Make Web Content Searchable (page 18). To learn how to use a Smart App Banner, see Promoting Apps with Smart App Banners.

The three Search-related APIs are designed to work in concert. If you mirror app content on the web, you can adopt all three; apps that don't have related content on the web can adopt `NSUserActivity` and the Core Spotlight APIs.

Adopting these APIs appropriately can improve the relevancy and ranking of the results related to your app. To give users the best search experience, the system measures how often users interact with app content and with Search and Safari results. iOS computes relevancy and ranking using information such as the frequency with which users interact with app activities, the time that passes between tapping a result and opening a deep link within your app, and (when appropriate) the reputation of your website.

Important: Be sure to avoid over-indexing your app content or adding unrelated keywords and attributes in an attempt to improve the ranking of your results. Because iOS measures the level of user engagement with search results, items that users don't find useful are quickly identified and can eventually stop showing up in results.

There are two main things you can do to ensure that users have a great experience when they search for content related to your app. First, make sure that your results are rich, descriptive, and useful. When users view compelling information that's directly related to their query, they're much more likely to engage with your app. Second, when users tap a result, take them directly to the appropriate area in your app. As much as possible, avoid presenting intervening screens or experiences that delay users from reaching the content they're interested in.

Use NSUserActivity APIs to Make App Activities and States Searchable

Currently, you use the `NSUserActivity` API to support Handoff (to learn more about enabling Handoff in your app, see *Handoff Programming Guide*). In iOS 9, `NSUserActivity` adds API that lets you designate specific activities or app states as searchable. When a searchable activity or state appears in Search or Safari results, users can tap the result to return to the relevant area in your app.

`NSUserActivity` also introduces properties you can use to provide indexable metadata about an activity or state, which helps you provide rich information in search results. For example, you can specify a title, description, and thumbnail image for a result.

To make an app activity or state searchable, create an `NSUserActivity` object to represent it. Use `NSUserActivity` properties to fully describe the activity or state and to make it eligible for search. Listing 1 shows how to set up an activity.

Listing 1 Creating a new activity

```
NSUserActivity *userActivity = [[NSUserActivity alloc]
  initWithActivityType:@“com.mycompany.activity-type”];
// Set properties that describe the activity and that can be used in search.
userActivity.title = @"...";
userActivity.keywords = [NSSet setWithArray:@[...]];
```

```
// Set values needed to restore state
userActivity.userInfo = @{ … };
// Enable the activity to participate in search results.
[userActivity.eligibleForSearch = YES];
```

When a user performs the activity or enters the app state associated with the `NSUserActivity` object you created, your app calls `[userActivity becomeCurrent]` to mark the activity as current. A current activity that's eligible for search is automatically added to the universal index (that is, `CSSearchableIndex`).

When a user taps on a result that's associated with an activity or state, your app uses `NSUserActivity` APIs to continue the activity and restore the user's position. Listing 2 shows how to continue an activity.

Listing 2 Continuing an activity

```
- (BOOL)application:(UIApplication *)application
  continueUserActivity:(NSUserActivity *)userActivity
  restorationHandler: {
    NSString *activityType = userActivity.activityType;
    if ([activityType isEqual:@“com.mycompany.activity-type”]) {
        // Handle restoration for values provided in userInfo
        return YES;
    }
    return NO;
}
```

In some cases, it's appropriate to make activities available to all users. An activity that's marked public can be shown to other users and can help encourage them to use your app. By default, activities are private. You should designate an activity as public only when the activity might be useful for other users. In general, user-created content is never useful for public results. To mark an activity as public, set the `eligibleForPublicIndexing` property to `YES`.

Use Core Spotlight APIs to Make App Content Searchable

Core Spotlight provides a database-like API that lets you add, retrieve, update, and delete items that represent searchable app content. When you use Core Spotlight to index items, you make it easy for users to search their own content.

To make content searchable, first create an attribute set that contains properties that specify the metadata you want to display about an item in a search result. The attributes you choose depend on your domain. You can use the attributes that Core Spotlight provides in categories defined on `CSSearchableItemAttributeSet`, or you can define your own. If you want to define a custom attribute, be as specific as possible in your definition and use the `contentTypeTree` property (defined in `CSSearchableItemAttributeSet_General.h`) so that your custom attribute can inherit from a known type.

Listing 3 shows how to create a `CSSearchableItemAttributeSet` object and set its properties.

Listing 3 Creating an attribute set for a searchable item

```
// Create an attribute set for an item that represents an image.
CSSearchableItemAttributeSet* attributeSet = [[CSSearchableItemAttributeSet alloc]
 initWithItemContentType:(NSString *)kUTTypeImage];
// Set properties that describe attributes of the item such as title, description,
 and image.
attributeSet.title = @"…";
attributeSet.contentDescription = @"…";
```

Next, create a `CSSearchableItem` object to represent the item and add it to the index. Listing 4 shows how to create a `CSSearchableItem` object and index it.

Listing 4 Creating a searchable item

```
// Create a searchable item, specifying its ID, associated domain, and the attribute
 set you created earlier.
CSSearchableItem *item;
item = [[CSSearchableItem alloc] initWithUniqueIdentifier:@"123456"
domainIdentifier:@"domain1.subdomainA" attributeSet:attributeSet];
// Index the item.
[[CSSearchableIndex defaultSearchableIndex] indexSearchableItems:@[item]
completionHandler: ^(NSError * __nullable error) {
NSLog(@"Search item indexed");
}];
```

When users tap a search result for an item that you added to the index, your app should open and restore the context associated with that item. To accomplish this, your app delegate implements `application:continueUserActivity:restorationHandler:`, checking the type of the incoming activity to see if the app is opening because the user tapped an indexed item in a search result. Listing 5 shows a skeletal implementation of `application:continueUserActivity:restorationHandler:`.

Listing 5 Implementing `continueUserActivity`… in the app delegate

```
- (BOOL)application:(UIApplication *)application continueUserActivity:(NSUserActivity
 *)userActivity restorationHandler:(void(^)(NSArray
*restorableObjects))restorationHandler {


    if ([[userActivity activityType] isEqualToString:CSSearchableItemActionType])
 {
       // This activity represents an item indexed using Core Spotlight, so restore
 the context related to the unique identifier.
    // The unique identifier of the Core Spotlight item is set in the activity's
userInfo for the key CSSearchableItemActivityIdentifier.
        NSString *uniqueIdentifier = [activity.userInfo
objectForKey:CSSearchableItemActivityIdentifier];
    }
}
```

It's recommended that you adopt both `NSUserActivity` and Core Spotlight functionality if your app lets users create or store content. Adopting both APIs lets you create relationships between activities and items that represent the same thing, which improves the user's experience. Listing 6 shows how to use a unique ID to relate a user activity and an item.

Listing 6 Relating a user activity and a searchable item

```
// Create an attribute set that specifies a related unique ID for a Core Spotlight
 item.
CSSearchableItemAttributeSet *attributes = [[CSSearchableItemAttributeSet alloc]
initWithItemContentType:@"public.image"];
attributes.relatedUniqueIdentifier = coreSpotlightUniqueIdentifier;


// Use the attribute set to create an NSUserActivity that's related to a Core
Spotlight item.
NSUserActivity *userActivity = [[NSUserActivity alloc]
  initWithActivityType:@“com.mycompany.viewing-message”];
```

```
userActivity.contentAttributeSet = attributes;
```

Core Spotlight provides several APIs you can use to work with the index and keep it up to date. Be sure to set the `indexDelegate` property in your app and implement the required `CSSearchableIndexDelegate` methods. For example, Listing 7 shows how to batch updates to the index.

Listing 7 Batching updates to the index

```
CSSearchableIndex *index = [CSSearchableIndex new];
[index beginIndexBatch];
[index indexSearchableitems:items completionHandler:nil];
[index deleteSearchableItemsWithIdentifiers:identifiers completionHandler:nil];
[index endIndexBatchWithClientState:clientState completionHandler:^(NSError *error)
 {
    // Handle errors.
}];
```

If a crash occurs while you're batching index updates, use the value of `clientState` to help you recover, as shown in Listing 8.

Listing 8 Using `clientState` to help you handle a crash while batching

```
CSSearchableIndex *index = [CSSearchableIndex new];
[index fetchLastClientStateWithCompletionHandler:^(NSData *clientState, NSError
*error) {

    if(error == nil) {
        NSArray *items = // Fetch a batch of items for the specified client state.

         [index beginIndexBatch];
         [index indexSearchableitems:items completionHandler:nil];
        [index endIndexBatchWithClientState:clientState completionHandler:^(NSError
 *error) {
            // Handle errors.
         }];
    }

}];
```

Core Spotlight defines a few ways you can remove data from the index. For example, Listing 9 shows different ways to specify items to delete from the index.

Listing 9 Removing items from the index

```
// Remove an item specified by identifier.
[[CSSearchableIndex defaultSearchableIndex]
deleteSearchableItemsWithIdentifiers:@[@“123456”] completionHandler:^(NSError
*)error) {
    // Handle errors.
})];
CodeLine
// Delete items in the specified domain and all subdomains.
[[CSSearchableIndex defaultSearchableIndex]
deleteSearchableItemsWithDomainIdentifiers:@[@“domain1”] completionHandler:^(NSError
 *)error) {
    // Handle errors.
})];


// Delete items in the specified subdomain.
[[CSSearchableIndex defaultSearchableIndex]
deleteSearchableItemsWithDomainIdentifiers:@[@“domain1.subdomainA”]
completionHandler:^(NSError *)error) {
    // Handle errors.
})];


// Delete all items.
[[CSSearchableIndex defaultSearchableIndex]
deleteAllSearchableItemsWithCompletionHandler:^(NSError *)error) {
    // Handle errors.
})];
```

In general, you should update the index while your app is running, but you can also create an index-maintenance app extension that lets the system communicate with your app while it’s not running and give you the opportunity to update the index or verify the validity of an item. Listing 10 shows one way to write this type of app extension.

Listing 10 Creating an index-maintenance app extension

```
@interface MyIndexRequestExtensionHandler : CSIndexExtensionRequestHandler


@end
```

```
@implementation MyIndexRequestExtensionHandler


- (void)searchableIndex:(CSSearchableIndex *)searchableIndex
reindexAllSearchableItemsWithAcknowledgementHandler:(void
(^)(void))acknowledgementHandler {
    // Use index API to re-index all your data.
    // Call the acknowledgement handler when this is done.
}


- (void)searchableIndex:(CSSearchableIndex *)searchableIndex
reindexSearchableItemsWithIdentifiers:(NSArray *)identifiers
acknowledgementHandler:(void (^)(void))acknowledgementHandler {
    // Use index API to re-index data with the specified identifiers.
    // Call the acknowledgement handler when this is done.
}


@end
```

Use Web Markup to Make App Content Searchable

If you mirror your app content on a website (or your app gets all its content from a website), you can use web markup to give users access to your app content in Search results. Because Apple indexes web content and makes it available in Search and Safari, it's crucial that you add markup to help Apple discover and index your content and display rich results.

Adopting Smart App Banners is the best way to help users of your website discover your app. Including an app-argument in your Smart App Banner markup allows Apple to index your content.

As an alternative to using Smart App Banners to describe deep links on your website, you can use one of the open standards Apple supports.

Use standards-based markup for structured data (such as that defined at Schema.org) to annotate your web content so that users can see rich search results. For example, a recipe website might use the markup shown in Listing 11 to provide richer information about a recipe.

Listing 11 Using markup to provide more information

```
<div itemscope itemtype="http://schema.org/Recipe">
```

```
  <span itemprop="name">Apple Pie</span>
  <span itemprop="description">This yummy recipe uses healthy Gala apples!</span>
  Prep Time: <meta itemprop="prepTime" content="PT30M”>30 minutes
  Cook Time: <meta itemprop="cookTime" content="PT2H”>2 hours
  Yields: <span itemprop="recipeYield”>8 servings</span>
  Ingredients:
  - <span itemprop="recipeIngredient”>6-8 Gala apples, diced</span>
  - <span itemprop="recipeIngredient”>1/4 cup of flour egg</span>
  - <span itemprop="recipeIngredient”>1 pie dough</span>
</div>
```

In addition to using structured data markup from Schema.org, you can provide Open Graph markup for specifying an image, or preferred title and description to accompany the result.

Use Universal Links to Enable Your App to Handle Links to Your Website

In iOS 9, your app can register to open web links (using https or http) directly, bypassing Safari. This connection between your app and website helps Apple surface your app content in search results.

Support for universal links is built on the same mechanism that powers Handoff between a web browser and a native app, and shared web credentials (for more information about these technologies, see Web Browser–to–Native App Handoff and *Shared Web Credentials Reference*). A trust relationship between the app and the website is established by adding a `com.apple.developer.associated-domains` entitlement to your app and an `apple-app-site-association` file to your website.

In your `com.apple.developer.associated-domains` entitlement, include a list of all the domains your app wants to handle as universal links. For example:

```
applinks:developer.apple.com
```

In your `apple-app-site-association` file, you can specify which paths from your website should be handled as universal links. For example:

```
{
    "applinks": {
        "apps": [],
        "details": {
            "9JA89QQLNQ.com.apple.wwdc": {
                "paths": [
                    "/wwdc/news/",
                    "/videos/wwdc/2015/*"
                ]
            }
        }
    }
}
```

After you place the signed JSON file on your website and your app adds the entitlement, when a user taps on a link to your website, your app opens to handle the link. To receive the link and handle it in within your app, you need to adopt the `UIApplicationDelegate` methods for Handoff (specifically `application:continueUserActivity:restorationHandler:`).

Adopting universal links in your app is strongly recommended over the use of custom URL schemes. Benefits of adopting universal links include:

- Security—universal links can't be claimed by another app developer, unlike custom URL schemes.
- Fallback to Safari for users who don't have your app installed.
- A single link works for all users of your content, regardless of whether they view it in your app or on your website.

Gaming

iOS 9 includes several technology improvements that make it easier than ever to implement your game's graphics and audio features. Take advantage of high-level frameworks for ease-of-development, or use new low-level enhancements to harness the power of the GPU.

GameplayKit

The GameplayKit framework (`GameplayKit.framework`) provides foundational technologies for building games. Use GameplayKit to develop gameplay mechanics, and combine it with any high-level graphics engine—such as SceneKit or SpriteKit—to build a complete game. This framework provides building blocks for creating games with modular architecture, including:

- Randomization tools for adding unpredictability to gameplay without compromising debugging
- Entity-component architecture to design gameplay code for better reusability

- State machines for untangling complex procedural code in gameplay systems

GameplayKit also includes standard implementations of common gameplay algorithms, so you can spend less time reading white papers and more time working on the mechanics that make your game unique. Several of the standard algorithm implementations in GameplayKit are listed below.

- A minmax artificial intelligence for adversarial turn-based games.
- An agent simulation that lets you describe movement behaviors in terms of high-level goals to be automatically pursued.
- Rule systems for building data-driven game logic, fuzzy reasoning, and emergent behavior.

To learn more about GameplayKit, see *GameplayKit Programming Guide* and *GameplayKit Framework Reference* . To see GameplayKit in action, download the sample code projects *FourInARow: Using the GameplayKit Minmax Strategist for Opponent AI* , *AgentsCatalog: Using the Agents System in GameplayKit* , and *DemoBots: Building a Cross Platform Game with SpriteKit and GameplayKit* .

Model I/O

The Model I/O framework (`ModelIO.framework`) provides a system-level understanding of 3D model assets and related resources. You can use this framework for several types of tasks, such as:

- Importing mesh data, material descriptions, lighting and camera settings, and other scene information from file formats used by popular authoring software and game engines
- Processing or generating such data—for example, to bake lighting information into a mesh, or create procedural sky textures
- Together with MetalKit, GLKit, or SceneKit APIs, efficiently loading asset data into GPU buffers for rendering
- Exporting processed or generated asset data to any of several file formats

To learn more about Model I/O, see *Model I/O Framework Reference* .

MetalKit

The MetalKit framework (`MetalKit.framework`) provides a set of utility functions and classes that reduce the effort required to create a Metal app. MetalKit provides development support for three key areas:

- Texture loading helps your app easily and asynchronously load textures from a variety of sources. Common file formats such as PNG and JPEG are supported, as well as texture-specific formats such as KTX and PVR.
- Model handling provides Metal-specific functionality that makes it easy to interface with Model I/O assets. Use these highly-optimized functions and objects to transfer data efficiently between Model I/O meshes and Metal buffers.

- View management provides a standard implementation of a Metal view that drastically reduces the amount of code needed to create a graphics-rendering app.

To learn more about MetalKit APIs, see *MetalKit Framework Reference* . For more information about Metal in general, see *Metal Programming Guide* , *Metal Framework Reference* , and *Metal Shading Language Guide* .

Metal Performance Shaders

The Metal Performance Shaders framework (`MetalPerformanceShaders.framework`) provides highly-optimized compute and graphics shaders that are designed to integrate easily and efficiently into your Metal app. These data-parallel shaders are specially tuned to take advantage of the unique hardware characteristics of each Metal-supported iOS GPU.

Use the Metal Performance Shader classes to achieve optimal performance for all supported hardware, without having to target or update your shader code to specific iOS GPU families. `MetalPerformanceShader` objects fit seamlessly into your Metal apps and can be used with Metal resource objects such as buffers and textures.

Common shaders provided by the Metal Performance Shader framework include:

- Gaussian blur—provided by the `MPSImageGaussianBlur` class.
- Image histogram—provided by the `MPSImageHistogram` class.
- Sobel edge detection—provided by the `MPSImageSobel` class.

New Features in Metal

The Metal framework (`Metal.framework`) adds new features to make your graphics-rendering apps look even better and be more performant. These features include:

- Improvements to the Metal Shading Language and Metal Standard Library
- Compute shaders can now write to a wider range of pixel formats
- The addition of private and depth stencil textures to align with OS X
- The addition of depth clamping and separate front and back stencil reference values for improved shadow quality

New Features in SceneKit

The SceneKit framework (`SceneKit.framework`) includes new features in iOS 9, including:

- Metal rendering support. See the `SCNView` and `SCNSceneRenderer` classes to enable high-performance Metal rendering on supported devices.

- A new Scene Editor in Xcode. Build games and interactive 3D apps in less time and with less code by designing scenes in Xcode (for a related sample code project, download *Fox: Building a SceneKit Game with the Xcode Scene Editor*).
- Positional audio. See the `SCNAudioPlayer` and `SCNNode` classes to add spatial audio effects that automatically track the listener's position in a scene.

For details on these and many other new features, see *SceneKit Framework Reference*.

New Features in SpriteKit

The SpriteKit framework (`SpriteKit.framework`) includes new features in iOS 9, such as:

- Metal rendering support. On devices that support Metal, metal rendering is automatically used, even in cases where you are using custom OpenGL ES shaders.
- An improved Scene Editor and a new Action Editor in Xcode. Build games and interactive 2D apps in less time and with less code by designing scenes in Xcode (for a related sample project, download *DemoBots: Building a Cross Platform Game with SpriteKit and GameplayKit*.)
- Camera nodes (that is, `SKCameraNode` objects) make it even easier to create scrolling games. Simply drop a camera node into your scene and set the scene's camera property.
- Positional audio. To learn how to add spatial audio effects that automatically track the listener's position in a scene, see *SKAudioNode Class Reference*.

For details on these and many other new features, see *SpriteKit Framework Reference*.

App Thinning

App thinning helps you develop apps for diverse platforms and deliver an optimized installation automatically. App thinning includes the following elements:

- Slicing. Artwork incorporated into the Asset Catalog and tagged for a platform allows the App Store to deliver only what is needed for installation.
- On-Demand Resources. Host additional content for your app in the iTunes App Store repository, allowing it to fetch resources as needed using asynchronous download and installation. To learn more about this technology, see *On-Demand Resources Guide*.
- Bitcode. Archive your app for submission to the App Store in an intermediate representation, which is compiled into 64- or 32-bit executables for the target devices when delivered.

To learn more about app thinning, see App Thinning (iOS, watchOS).

Support for Right-to-Left Languages

iOS 9 brings comprehensive support for right-to-left languages, which makes it easier for you to provide a flipped user interface. For example:

- Standard UIKit controls automatically flip in a right-to-left context.
- `UIView` defines semantic content attributes that let you specify how particular views should appear in a right-to-left context.
- `UIImage` adds the `imageFlippedForRightToLeftLayoutDirection` method, which makes it easy to flip an image programmatically when appropriate.

To learn more about providing a flipped user interface, see Supporting Right-to-Left Languages.

App Transport Security

App Transport Security (ATS) enforces best practices in the secure connections between an app and its back end. ATS prevents accidental disclosure, provides secure default behavior, and is easy to adopt; it is also on by default in iOS 9 and OS X v10.11. You should adopt ATS as soon as possible, regardless of whether you're creating a new app or updating an existing one.

If you're developing a new app, you should use HTTPS exclusively. If you have an existing app, you should use HTTPS as much as you can right now, and create a plan for migrating the rest of your app as soon as possible. In addition, your communication through higher-level APIs needs to be encrypted using TLS version 1.2 with forward secrecy. If you try to make a connection that doesn't follow this requirement, an error is thrown. If your app needs to make a request to an insecure domain, you have to specify this domain in your app's `Info.plist` file.

Extension Points

iOS 9 introduces several new extension points (an extension point defines usage policies and provides APIs to use when you create an app extension for that area). Specifically:

- Network extension points:
  - Use the Packet Tunnel Provider extension point to implement the client side of a custom VPN tunneling protocol.
  - Use the App Proxy Provider extension point to implement the client side of a custom transparent network proxy protocol.

    - Use the Filter Data Provider and the Filter Control Provider extension points to implement dynamic, on-device network content filtering.

  Each of the network extension points requires special permission from Apple.

- Safari extension points:
    - Use the Shared Links extension point to enable users to see your content in Safari's Shared Links.
    - Use the Content Blocking extension point to give Safari a block list describing the content that you want to block while your users are browsing the web.
- Spotlight extension points:
    - Use the app indexing extension point to index data in your app.
    - Use the Index Maintenance extension point to support the reindexing of app data without launching the app.
- The Audio Unit extension point allows your app to provide musical instruments, audio effects, sound generators, and more for use within apps like GarageBand, Logic, and other Audio Unit host apps. The extension point also brings a full audio plug-in model to iOS and lets you sell Audio Units on the App Store.

To learn more about creating app extensions in general, see *App Extension Programming Guide* .

Contacts and Contacts UI

iOS 9 introduces the Contacts and Contacts UI frameworks (`Contacts.framework` and `ContactsUI.framework`), which provide modern object-oriented replacements for the Address Book and Address Book UI frameworks. To learn more, see *Contacts Framework Reference* and *ContactsUI Framework Reference* .

Watch Connectivity

The Watch Connectivity framework (`WatchConnectivity.framework`) provides two-way communication between an iPhone and a paired Apple Watch. Use this framework to coordinate activities between your iOS app and your corresponding Watch app. The framework supports immediate messaging between the apps when they are both running, and background messaging in other cases. To learn more, see *Watch Connectivity Framework Reference* .

Keychain

The keychain provides more item protection options and a new type of encryption keys owned by the secure enclave. Specifically:

- New constraints for access control lists that allow creating constraints with Touch ID only or passcode only.
- A new Touch ID constraint that invalidates keychain items when a fingerprint is added or removed.
- Support for app-provided entropy for keychain item encryption using the Application Password option of the access control list.
- Support for an authentication context that lets you invoke the authentication separately from `SecItem` calls.
- Support for keys generated and used inside the secure enclave using the `kSecAttrTokenIDSecureEnclave` attribute. Note that access to these keys can be controlled by all constraints supported by access control lists.

Swift Enhancements

To learn about what's new in Swift, see Swift Language.

Additional Framework Changes

In addition to the major changes described above, iOS 9 includes many other improvements.

AV Foundation Framework

The AV Foundation framework (`AVFoundation.framework`) adds new `AVSpeechSynthesisVoice` API that lets you specify a voice by identifier, instead of by language. You can also use the `name` and `quality` properties to get information about a voice.

AVKit Framework

The AVKit framework (`AVKit.framework`) includes the `AVPictureInPictureController` and `AVPlayerViewController` classes, which help you participate in Picture in Picture. For more information about Picture in Picture, see Multitasking Enhancements for iPad (page 10).

CloudKit Framework

If you have a CloudKit app, you can use CloudKit web services or CloudKit JS, a JavaScript library, to provide a web interface for users to access the same data as your app. You must have the schema for your databases already created to use a web interface to fetch, create, update, and delete records, zones, and subscriptions. For more information, see *CloudKit JS Reference*, *CloudKit Web Services Reference*, and *CloudKit Catalog: An Introduction to CloudKit (Cocoa and JavaScript)*.

Foundation Framework

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

- APIs for on-demand loading of `NSBundle` resources.
- Strings file support for context-dependent variable width strings.
- `NSProcessInfo` APIs for power and thermal management.

HealthKit Framework

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

- New support for tracking areas such as reproductive health and UV exposure. To learn about the new constants that describe characteristics, quantities, and other items, see *HealthKit Constants Reference*.
- New support for bulk-deleting entries and tracking deleted entries. For more information, see `HKDeletedObject`, `HKAnchoredObjectQuery`, and the `deleteObjects:withCompletion:` and `deleteObjectsOfType:predicate:withCompletion:` methods in *HKHealthStore Class Reference*.

Local Authentication Framework

The Local Authentication framework (`LocalAuthentication.framework`) includes the following enhancements:

- The ability to get a representation of the current set of enrolled fingers so that apps can change behavior when a finger is enrolled or removed.
- Support for canceling a user prompt from code.
- Support for evaluating keychain access control lists and the use of an authentication context in keychain calls.
- Support for reusable Touch ID matches. A match from the previous phone unlock can be used by `evaluateAccessControl:` and `evaluatePolicy:localizedReason:reply:`.

MapKit Framework

The MapKit framework (`MapKit.framework`) includes several additions that help you provide a richer user experience. Specifically:

- MapKit supports querying transit ETAs and launching Maps into transit directions.
- Map views support a 3D flyover mode.
- Annotations can be fully customized.
- Search results for MapKit and `CLGeocoder` can provide a time zone for the result.

PassKit Framework

The PassKit framework (`PassKit.framework`) includes several additions that support enhancements in Apple Pay. Specifically:

- In iOS 9, Apple Pay supports Discover cards and store debit and credit cards. For more information, see "Payment Networks" in *PKPaymentRequest Class Reference*.
- Card issuers and payment networks can add cards to Apple Pay directly in their apps. For more information, see *PKAddPaymentPassViewController Class Reference*.

Safari Services Framework

The Safari Services framework (`SafariServices.framework`) includes the following enhancement.

`SFSafariViewController` can be used to display web content within your app. It shares cookies and other website data with Safari, and has many of Safari's great features, such as Safari AutoFill and Safari Reader. Unlike Safari itself, the `SFSafariViewController` UI is tailored for displaying a single page, featuring a Done button that takes users back to where they were in your app.

If your app displays web content, but does not customize that content, consider replacing your `WKWebView` or `UIWebView`-based browsers with `SFSafariViewController`.

UIKit Framework

The UIKit framework (`UIKit.framework`) includes many enhancements, such as:

- The `UIStackView` class, which helps you manage a set of subviews as a stack that can be arranged vertically or horizontally.
- New layout anchors in `UIView` (such as `leadingAnchor` and `widthAnchor`), `NSLayoutAnchor`, and `NSLayoutDimension`, all of which help make layout easy.

- New layout guides that help you adopt readable content margins and define where within a view the content should draw. For more information, see `UILayoutGuide`.
- A new `UIApplicationDelegate` method you can use to open a document (and modify it) in place, instead of working with a copy of the document. To support the open-in-place functionality, an app also adds to its `Info.plist` file the `LSSupportsOpeningDocumentsInPlace` key with a value of `YES`.
- The `UITextInputAssistantItem` class, which helps you lay out bar button groups in the shortcuts bar.
- Enhancements to touch events, such as the ability to get access to intermediate touches that may have occurred since the last refresh of the display and touch prediction.
- Enhancements to UIKit Dynamics, such as support for nonrectangular collision bounds, the new `UIFieldBehavior` class, which supports various field types in addition to being customizable, and additional attachment types in `UIAttachmentBehavior`.
- The new `behavior` property in `UIUserNotificationAction`, which lets you support text input from users in notifications.
- The new `NSDataAsset` class, which makes it easy to fetch content tailored to the memory and graphics capabilities of your device.
- All standard UIKit controls flip appropriately to support right-to-left languages. In addition, navigation, gestures, collection views, and table cell layouts also flip appropriately.

Deprecated APIs

The following APIs are deprecated:

- The Address Book and Address Book UI frameworks. Use the Contacts and Contacts UI frameworks instead.

For a complete list of specific API deprecations, see *iOS 9.0 API Diffs* .

iOS 8.4

iOS 8.4 includes a completely redesigned music app as well as bug fixes and performance improvements. This release has no API changes from iOS 8.3 and targets the same iOS devices as the previous release.

For late-breaking news and information about known issues, see *iOS 8.4 Release Notes* .

iOS 8.3

This article summarizes the key developer-related features introduced in iOS 8.3. 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 8.3 Release Notes* . For the complete list of new APIs added in iOS 8.3, see *iOS 8.3 API Diffs* .

Apple Pay

You can use a new class, `PKPaymentButton`, to create buttons that initiate Apple Pay purchases. These buttons are appropriately styled with the Apple Pay logo.

Apple Pay now supports different shipping types, such as “shipping,” “delivery,” “pickup from store,” and “pickup from customer.” You can also now request the user’s name without requesting their address.

For more information on Apple Pay, see *Apple Pay Programming Guide* ; to learn more about `PKPaymentButton`, see *PassKit Framework Reference* .

Metal

Apps that use Metal now have additional ways to manipulate shader buffers and introspect vertex attribute types:

- `MTLComputeCommandEncoder` and `MTLRenderCommandEncoder` objects include new methods for copying data directly to a shader’s buffer table and for relocating the initial offset for using data in an already bound buffer. Use these options to attach small, one-time-use data buffers or reconfigure attached buffers without needing to create or re-create a `MTLBuffer` object.
- `MTLVertexAttribute` objects now include an `attributeType` property for introspecting the types of attributes declared in Metal shader source code.

For more information on Metal, see *Metal Programming Guide* , *Metal Framework Reference* , and *Metal Shading Language Guide* .

iOS 8.2

This article summarizes the key developer-related features introduced in iOS 8.2. 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 8.2 Release Notes* . For the complete list of new APIs added in iOS 8.2, see *iOS 8.2 API Diffs* .

Apple Watch

iOS 8.2 lets you give Apple Watch users access to data and functionality related to your iOS app. The primary interface you can create is a WatchKit app, which gives users quick, focused ways to access their content without opening your iOS app on their iPhone.

You can also enhance your WatchKit app by providing two optional Apple Watch interfaces that give users timely, high-value information:

- A Glance provides a screenful of meaningful information related to a WatchKit app. As its name implies, a Glance displays information that users can absorb instantly, without interaction; in fact, tapping a Glance on Apple Watch launches your WatchKit app.
- A custom notification interface displays information from the local or remote notifications that get delivered to your iOS app. And if your iOS app registers for interactive notifications, you can create a custom actionable notification interface on Apple Watch that lets users take action from their wrists.

Xcode 6.2 with iOS 8.2 SDK Beta provides the templates and other tools you need to design, develop, and debug all three types of Watch interfaces. To learn how to extend your iOS app for Apple Watch, including how to set up your Xcode project, read *Apple Watch Programming Guide* . You can also explore the sample projects *Lister (for watchOS, iOS, and OS X)* and *WatchKit Catalog: Using WatchKit Interface Elements* .

As you might imagine, designing a WatchKit app is very different from designing an iOS app. For some guidance on creating an experience users will love, see *Apple Watch Human Interface Guidelines* . For additional design resources, such as templates and videos, visit https://developer.apple.com/watchkit/.

WatchKit Framework

WatchKit (`WatchKit.framework`) is a new framework for developing interfaces for Apple Watch. To help you create, configure, and manipulate a WatchKit app, Glance, and custom notification interface, the WatchKit framework provides:

- Interface controllers that you use to implement hierarchical or page-based navigation
- Interface objects—such as labels, images, buttons, and tables—that let you display your content and enable interaction

You can learn more about the WatchKit objects you use to create these interfaces in Interface Navigation and Interface Objects.

iOS 8.1

This article summarizes the key developer-related features introduced in iOS 8.1. 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 8.1.1 Release Notes* . For the complete list of new APIs added in iOS 8.1, see *iOS 8.1 API Diffs* .

Apple Pay

The PassKit framework adds support for payment passes and payment requests. Payment requests let users securely provide you with their payment and contact information to pay for physical goods and services. Payment passes are added to Passbook by payment networks, such as credit card issuers and banks, and let the user select their account on that payment network to fund payments.

For more information, see *PassKit Framework Reference* .

iOS 8.0

This article summarizes the key developer-related features introduced in iOS 8, which runs on currently shipping iOS devices. The article also lists the documents that describe new features in more detail.

For late-breaking news and information about known issues, see *iOS 8 Release Notes* . For the complete list of new APIs added in iOS 8, see *iOS 8.0 API Diffs* . For more information on new devices, see *iOS Device Compatibility Reference* .

App Extensions

iOS 8 lets you extend select areas of the system by supplying an *app extension*, which is code that enables custom functionality within the context of a user task. For example, you might supply an app extension that helps users post content to your social sharing website. After users install and enable this extension, they can choose it when they tap the Share button in their current app. Your custom sharing extension provides the code that accepts, validates, and posts the user's content. The system lists the extension in the sharing menu and instantiates it when the user chooses it.

In Xcode, you create an app extension by adding a preconfigured app extension target to an app. After a user installs an app that contains an extension, user enables the extension in the Settings app. When the user is running other apps, the system makes the enabled extension available in the appropriate system UI, such as the Share menu.

iOS supports app extensions for the following areas, which are known as extension points:

- *Share*. Share content with social websites or other entities.
- *Action*. Perform a simple task with the selected content.
- *Today*. Provide a quick update or enable a brief task in the Today view of Notification Center.
- *Photo editing*. Perform edits to a photo or video within the Photos app.
- *Storage provider*. Provide a document storage location that can be accessed by other apps. Apps that use a document picker view controller can open files managed by the Storage Provider or move files into the Storage Provider.
- *Custom keyboard*. Provide a custom keyboard that the user can choose in place of the system keyboard for all apps on the device.

Each extension point defines appropriate APIs for its purposes. When you use an app extension template to begin development, you get a default target that contains method stubs and property list settings defined by the extension point you chose.

For more information on creating extensions, see *App Extension Programming Guide* .

Touch ID Authentication

Your app can now use Touch ID to authenticate the user. Some apps may need to secure access to all of their content, while others might need to secure certain pieces of information or options. In either case, you can require the user to authenticate before proceeding.

- Your app can use Touch ID to unlock individual keychain items. For a working sample, see *KeychainTouchID: Using Touch ID with Keychain and LocalAuthentication* .
- Use the Local Authentication Framework (`LocalAuthentication.framework`) to display an alert to the user with an application-specified reason for why the user is authenticating. When your app gets a reply, it can react based on whether the user was able to successfully authenticate. For more information, see Local Authentication Framework Reference.

Photos

Take better photos in your app, provide new editing capabilities to the Photos app, and create new, more efficient workflows that access the user's photo and video assets.

Photos Framework

The Photos framework (`Photos.framework`) provides new APIs for working with photo and video assets, including iCloud Photos assets, that are managed by the Photos app. This framework is a more capable alternative to the Assets Library framework. Key features include a thread-safe architecture for fetching and caching thumbnails and full-sized assets, requesting changes to assets, observing changes made by other apps, and resumable editing of asset content.

For more information, see *Photos Framework Reference* .

Use the related Photos UI framework (`PhotosUI.framework`) to create app extensions for editing image and video assets in the Photos app. For more information, see *App Extension Programming Guide* .

Manual Camera Controls

The AV Foundation framework (`AVFoundation.framework`) makes it easier than ever to take great photos. Your app can take direct control over the camera focus, white balance, and exposure settings. In addition, your app can use bracketed exposure captures to automatically capture images with different exposure settings.

For more information see *AV Foundation Framework Reference* .

Improved Camera Functionality

Use the following APIs to discover and enable new camera features found on the iPhone 6 and iPhone 6 Plus:

- A new property (`videoHDRSupported`) can determine whether a capture device supports high dynamic range streaming.
- A new video stabilization mode (`AVCaptureVideoStabilizationModeCinematic`) provides more cinematic results in the captured video.
- A new property (`highResolutionStillImageOutputEnabled`) can be used to set an `AVCaptureStillImageOutput` object to capture still pictures at a higher resolution.
- A new property (`autoFocusSystem`) can be used to determine how the camera performs auto focusing.

Games

Technology improvements in iOS 8 make it easier than ever to implement your game's graphics and audio features. Take advantage of high-level frameworks for ease-of-development, or use new low-level enhancements to harness the power of the GPU.

Metal

Metal provides extremely low-overhead access to the A7 and A8 GPUs, enabling extremely high performance for your sophisticated graphics rendering and computational tasks. Metal eliminates many performance bottlenecks—such as costly state validation—that are found in traditional graphics APIs. Metal is explicitly designed to move all expensive state translation and compilation operations out of the critical path of your most performance sensitive rendering code. Metal provides precompiled shaders, state objects, and explicit command scheduling to ensure your application achieves the highest possible performance and efficiency for your GPU graphics and compute tasks. This design philosophy extends to the tools used to build your app. When your app is built, Xcode compiles Metal shaders in the project into a default library, eliminating most of the runtime cost of preparing those shaders.

Graphics, compute, and blit commands are designed to be used together seamlessly and efficiently. Metal is specifically designed to exploit modern architectural considerations, such as multiprocessing and shared memory, to make it easy to parallelize the creation of GPU commands.

With Metal, you have a streamlined API, a unified graphics and compute shading language, and Xcode-based tools, so you don't need to learn multiple frameworks, languages, and tools to take full advantage of the GPU in your game or app.

For more information on using Metal, see *Metal Programming Guide* , *Metal Framework Reference* , and *Metal Shading Language Guide* .

SceneKit

SceneKit is an Objective-C framework for building simple games and rich app user interfaces with 3D graphics, combining a high-performance rendering engine with a high-level, descriptive API. SceneKit has been available since OS X v10.8 and is now available in iOS for the first time. Lower-level APIs (such as OpenGL ES) require you to implement the rendering algorithms that display a scene in precise detail. By contrast, SceneKit lets you describe your scene in terms of its content—geometry, materials, lights, and cameras—then animate it by describing changes to those objects.

SceneKit's 3D physics engine enlivens your app or game by simulating gravity, forces, rigid body collisions, and joints. Add high-level behaviors that make it easy to use wheeled vehicles such as cars in a scene, and add physics fields that apply radial gravity, electromagnetism, or turbulence to objects within an area of effect.

Use OpenGL ES to render additional content into a scene, or provide GLSL shaders that replace or augment SceneKit's rendering. You can also add shader-based postprocessing techniques to SceneKit's rendering, such as color grading or screen space ambient occlusion.

For more information, see *SceneKit Framework Reference* .

SpriteKit

The SpriteKit framework (`SpriteKit.framework`) adds new features to make it easier to support advanced game effects. These features include support for custom OpenGL ES shaders and lighting, integration with SceneKit, and advanced new physics effects and animations. For example, you can create physics fields to simulate gravity, drag, and electromagnetic forces using the `SKFieldNode` class. You can easily create physics bodies with per-pixel collision masks. And it is easier than ever to pin a physics body to its parent, even if its parent does not have a physics body of its own. These new physics features make complex simulations much easier to implement.

Use constraints to modify the effects of physics and animations on the content of your scene—for example, you can make one node always point toward another node regardless of where the two nodes move.

Xcode 6 also incorporates new shader and scene editors that save you time as you create your game. Create a scene's contents, specifying which nodes appear in the scene and characteristics of those nodes, including physics effects. The scene is then serialized to a file that your game can easily load.

For information about the classes of this framework, see *SpriteKit Framework Reference* and *SpriteKit Programming Guide*.

AV Audio Engine

AVFoundation framework (`AVFoundation.framework`) adds support for a broad cross-section of audio functionality at a higher level of abstraction than Core Audio. These new audio capabilities are available on both OS X and iOS and include automatic access to audio input and output hardware, audio recording and playback, and audio file parsing and conversion. You also gain access to audio units for generating special effects and filters, pitch and playback speed management, stereo and 3D audio environments, and MIDI instruments.

For more information, see *AV Foundation Framework Reference*.

OpenGL ES

iOS 8 adds the following new extensions to OpenGL ES.

- The APPLE_clip_distance extension adds support for hardware clip planes to OpenGL ES 2.0 and 3.0.
- The APPLE_texture_packed_float adds two new floating-point texture formats, `R11F_G11F_B10F` and `RGB9_E5`.
- The APPLE_color_buffer_packed_float extension builds on `APPLE_texture_packed_float` so that the new texture formats can be used by a framebuffer object. This means that an app can render into a framebuffer that uses one of these formats.

HealthKit Framework

HealthKit (`HealthKit.framework`) is a new framework for managing a user's health-related information. With the proliferation of apps and devices for tracking health and fitness information, it's difficult for users to get a clear picture of how they are doing. HealthKit makes it easy for apps to share health-related information, whether that information comes from devices connected to an iOS device or is entered manually by the user. The user's health information is stored in a centralized and secure location. The user can then see all of that data displayed in the Health app.

When your app implements support for HealthKit, it gets access to health-related information for the user and can provide information about the user, without needing to implement support for specific fitness-tracking devices. The user decides which data should be shared with your app. Once data is shared with your app, your app can register to be notified when that data changes; you have fine-grained control over when your app is notified. For example, request that your app be notified whenever users takes their blood pressure, or be notified only when a measurement shows that the user's blood pressure is too high.

For more information, see *HealthKit Framework Reference* .

HomeKit Framework

HomeKit (`HomeKit.framework`) is a new framework for communicating with and controlling connected devices in a user's home. New devices for the home are offering more connectivity and a better user experience. HomeKit provides a standardized way to communicate with those devices.

Your app can use HomeKit to communicate with devices that users have in their homes. Using your app, users can discover devices in their home and configure them. They can also create actions to control those devices. The user can group actions together and trigger them using Siri. Once a configuration is created, users can invite other people to share access to it. For example, a user might temporarily offer access to a house guest.

Use the HomeKit Accessory Simulator to test the communication of your HomeKit app with a device.

For more information, see *HomeKit Framework Reference* .

iCloud

iCloud includes some changes that affect the behavior of existing apps and that affect users of those apps.

Document-Related Data Migration

The iCloud infrastructure is more robust and reliable when documents and data are transferred between user devices and the server. When a user installs iOS 8 and logs into the device with an iCloud account, the iCloud server performs a one-time migration of the documents and data in that user's account. This migration involves copying the documents and data to a new version of the app's container directory. The new container is accessible only to devices running iOS 8 or OS X v10.10. Devices running older operating systems can continue to access to the original container, but changes made in that container do not appear in the new container and changes made in the new container do not appear in the original container.

CloudKit

CloudKit (`CloudKit.framework`) is a conduit for moving data between your app and iCloud. Unlike other iCloud technologies where data transfers happen transparently, CloudKit gives you control over when transfers occur. You can use CloudKit to manage all types of data.

Apps that use CloudKit directly can store data in a repository that is shared by all users. This public repository is tied to the app itself and is available even on devices without a registered iCloud account. As the app developer, you can manage the data in this container directly and see any changes made by users through the CloudKit dashboard.

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

Document Picker

The document picker view controller (`UIDocumentPickerViewController`) grants users access to files outside your application's sandbox. It is a simple mechanism for sharing documents between apps. It also enables more complex workflows, because users can edit a single document with multiple apps.

The document picker lets you access files from a number of document providers. For example, the iCloud document provider grants access to documents stored inside another app's iCloud container. Third-party developers can provide additional document providers by using the Storage Provider extension.

For more information, see the *Document Picker Programming Guide* .

Handoff

Handoff is a feature in OS X and iOS that extends the user experience of continuity across devices. Handoff enables users to begin an activity on one device, then switch to another device and resume the same activity on the other device. For example, a user who is browsing a long article in Safari moves to an iOS device that's signed in to the same Apple ID, and the same webpage automatically opens in Safari on iOS, with the same scroll position as on the original device. Handoff makes this experience as seamless as possible.

To participate in Handoff, an app adopts a small API in Foundation. Each ongoing activity in an app is represented by a user activity object that contains the data needed to resume an activity on another device. When the user chooses to resume that activity, the object is sent to the resuming device. Each user activity object has a delegate object that is invoked to refresh the activity state at opportune times, such as just before the user activity object is sent between devices.

If continuing an activity requires more data than is easily transferred by the user activity object, the resuming app has the option to open a stream to the originating app. Document-based apps automatically support activity continuation for users working with iCloud-based documents.

For more information, see *Handoff Programming Guide* .

Supporting New Screen Sizes and Scales

Apps linked against iOS 8 and later should be prepared to support the larger screen size of iPhone 6 and iPhone 6 Plus. On the iPhone 6 Plus, apps should also be prepared to support a new screen scale. In particular, apps that support OpenGL ES and Metal can also choose to size their rendering `CAEAGLLayer` or `CAMetalLayer` to get the best possible performance on the iPhone 6 Plus.

To let the system know that your app supports the iPhone 6 screen sizes, include a storyboard launch screen file in your app's bundle. At runtime, the system looks for a storyboard launch screen file. If such an file is present, the system assumes that your app supports the iPhone 6 and 6 Plus explicitly and runs it in fullscreen mode. If such an image is not present, the system reports a smaller screen size (either 320 by 480 points or 320 by 568 points) so that your app's screen-based calculations continue to be correct. The contents are then scaled to fit the larger screen.

For more information about specifying the launch images for your app, see Adding App Icons and a Launch Screen File.

iOS 8 adds new features that make dealing with screen size and orientation much more versatile. It is easier than ever to create a single interface for your app that works well on both iPad and iPhone, adjusting to orientation changes and different screen sizes as needed. Using size classes, you can retrieve general information about the size of a device in its current orientation. You can use this information to make initial assumptions about which content should be displayed and how those interface elements are related to each other. Then, use Auto Layout to resize and reposition these elements to fit the actual size of the area provided. Xcode 6 uses size classes and autolayout to create storyboards that adapt automatically to size class changes and different screen sizes.

Traits Describe the Size Class and Scale of an Interface

Size classes are traits assigned to a user interface element, such as a screen or a view. There are two types of size classes in iOS 8: regular and compact. A regular size class denotes either a large amount of screen space, such as on an iPad, or a commonly adopted paradigm that provides the illusion of a large amount of screen space, such as scrolling on an iPhone. Every device is defined by a size class, both vertically and horizontally.

Figure 1 and Figure 2 show the native size classes for the iPad. With the amount of screen space available, the iPad has a regular size class in the vertical and horizontal directions in both portrait and landscape orientations.

Figure 1 iPad size classes in portrait

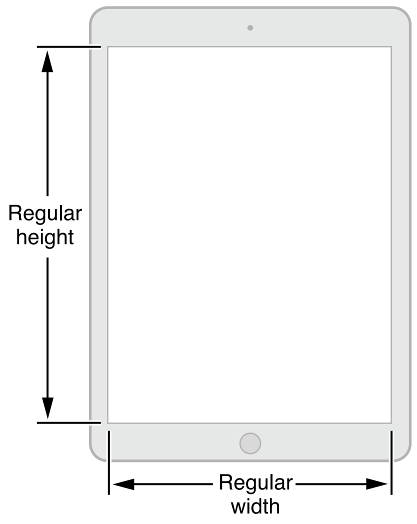
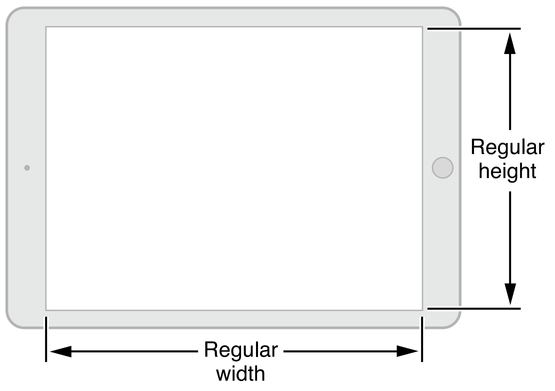


Figure 2 iPad size classes in landscape



The size classes for iPhones differ based on the kind of device and its orientation. In portrait, the screen has a compact size class horizontally and a regular size class vertically. This corresponds to the common usage paradigm of scrolling vertically for more information. When iPhones are in landscape, their size classes vary.

Most iPhones have a compact size class both horizontally and vertically, as shown in Figure 3 and Figure 4. The iPhone 6 Plus has a screen large enough to support regular width in landscape mode, as shown in Figure 5 (page 44).

Figure 3 iPhone size classes in portrait

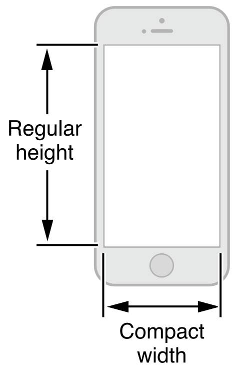


Figure 4 iPhone size classes in landscape

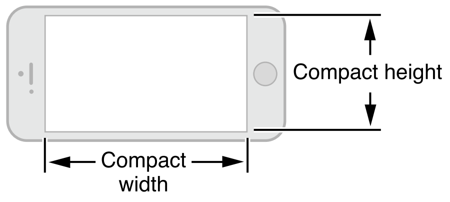
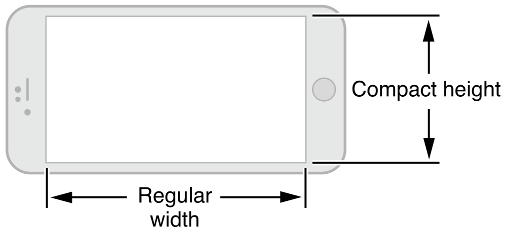


Figure 5 iPhone 6 Plus classes in landscape



You can change the size classes associated with a view. This flexibility is especially useful when a smaller view is contained within a larger view. Use the default size classes to arrange the user interface of the larger view and arrange information in the subview based on whatever size classes you feel is most appropriate to that subview.

To support size classes, the following classes are new or modified:

- The `UITraitCollection` class is used to describe a collection of traits assigned to an object. Traits specify the size class, display scale, and idiom for a particular object. Classes that support the `UITraitEnvironment` protocol (such as `UIScreen`, `UIViewController` and `UIView`) own a trait collection. You can retrieve an object's trait collection and perform actions when those traits change.
- The `UIImageAsset` class is used to group like images together based on their traits. Combine similar images with slightly different traits into a single asset and then automatically retrieve the correct image for a particular trait collection from the image asset. The `UIImage` class has been modified to work with image assets.
- Classes that support the `UIAppearance` protocol can customize an object's appearance based on its trait collection.
- The `UIViewController` class adds the ability to retrieve the trait collection for a child view. You can also lay out the view by changing the size class change through the `viewWillTransitionToSize:withTransitionCoordinator:` method.

Xcode 6 supports unified storyboards. A storyboard can add or remove views and layout constraints based on the size class that the view controller is displayed in. Rather than maintaining two separate (but similar) storyboards, you can make a single storyboard for multiple size classes. First, design your storyboard with a common interface and then customize it for different size classes, adapting the interface to the strengths of each form factor. Use Xcode 6 to test your app in a variety of size classes and screen sizes to make sure that your interface adapts to the new sizes properly.

Supporting New Screen Scales

The iPhone 6 Plus uses a new Retina HD display with a very high DPI screen. To support this resolution, iPhone 6 Plus creates a `UIScreen` object with a screen size of 414 x 736 points and a screen scale of `3.0` (1242 x 2208 pixels). After the contents of the screen are rendered, UIKit samples this content down to fit the actual screen dimensions of 1080 x 1920. To support this rendering behavior, include new artwork designed for the new `3x` screen scale. In Xcode 6, asset catalogs can include images at `1x`, `2x`, and `3x` sizes; simply add the new image assets and iOS will choose the correct assets when running on an iPhone 6 Plus. The image loading behavior in iOS also recognizes an `@3x` suffix.

In a graphics app that uses Metal or OpenGL ES, content can be easily rendered at the precise dimensions of the display without requiring an additional sampling stage. This is critical in high-performance 3D apps that perform many calculations for each rendered pixel. Instead, create buffers to render into that are the exact resolution of the display.

A `UIScreen` object provides a new property (`nativeScale`) that provides the native screen scale factor for the screen. When the `nativeScale` property has the same value as the screen's`scale` property, then the rendered pixel dimensions are the same as the screen's native pixel dimensions. When the two values differ, then you can expect the contents to be sampled before they are displayed.

If you are writing an OpenGL ES app, a `GLKView` object automatically creates its renderbuffer objects based on the view's size and the value of its `contentScaleFactor` property. After the view has been added to a window, set the view's `contentScaleFactor` to the value stored in the screen's `nativeScale` property, as shown in Listing 1.

Listing 1 Supporting native scale in a GLKView object

```
- (void) didMoveToWindow
{
    self.contentScaleFactor = self.window.screen.nativeScale;
}
```

In a Metal app, your own view class should have code similar to the code found in Listing 1 (page 46). In addition, whenever your view's size changes, and prior to asking the Metal layer for a new drawable, calculate and set the metal layer's `drawableSize` property as shown in Listing 2 (page 46). (An OpenGL ES app that is creating its own renderbuffers would use a similar calculation.)

Listing 2 Adjusting the size of a Metal layer to match the native screen scale

```
CGSize drawableSize = self.bounds.size;
drawableSize.width  *= self.contentScaleFactor;
drawableSize.height *= self.contentScaleFactor;
metalLayer.drawableSize = drawableSize;
```

See *MetalBasic3D* for a working example. The Xcode templates for OpenGL ES and Metal also demonstrate these same techniques.

Additional Framework Changes

In addition to the major changes described above, iOS 8 includes other improvements.

API Modernization

Many frameworks on iOS have adopted small interface changes that take advantage of modern Objective-C syntax:

- Getter and setter methods are replaced by properties in most classes. Code using the existing getter and setter methods continues to work with this change.

- Initialization methods are updated to have a return value of `instancetype` instead of `id`.
- Designated initializers are declared as such where appropriate.

In most cases, these changes do not require any additional work in your own app. However, you may also want to implement these changes in your own Objective-C code. In particular, you may want to modernize your Objective-C code for the best experience when interoperating with Swift code.

For more information, see *Adopting Modern Objective-C* .

AV Foundation Framework

The AV Foundation framework (`AVFoundation.framework`) enables you to capture metadata over time while shooting video. Arbitrary types of metadata can be embedded with a video recording at various points in time. For example, you might record the current physical location in a video created by a moving camera device.

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

AVKit Framework

The AVKit framework (`AVKit.framework`) previously introduced on OS X is available on iOS. Use it instead of Media Player framework when you need to display a video.

CoreAudioKit Framework

The new CoreAudioKit framework simplifies the effort required to create user interfaces for apps that take advantage of inter-app audio.

Core Image Framework

The Core Image framework (`CoreImage.framework`) has the following changes:

- You can create custom image kernels in iOS.
- Core image detectors can detect rectangles and QR codes in an image.

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

Core Location Framework

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

- You can determine which floor the device is on, if the device is in a multistory building.

- The visit service provides an alternative to the significant location change service for apps that need location information about interesting places visited by the user.

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

Core Motion Framework

Core Motion provides two new classes (`CMAltimeter` and `CMAltitudeData`) which allow you to access the barometer on the iPhone 6 and iPhone 6 Plus. On these two devices, you can also use a `CMMotionActivity` object to determine whether the user is on a bicycle.

Foundation Framework

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

- The `NSFileVersion` class provides access to past versions of iCloud documents. These versions are stored in iCloud, but can be downloaded on request.
- The `NSURL` class supports storing document thumbnails as metadata.
- The `NSMetadataQuery` class can search for external iCloud documents that your app has opened.

Game Controller Framework

The Game Controller framework (`GameController.framework`) has the following changes:

- If the controller is attached to a device, you can now receive device motion data directly from the Game Controller framework.
- If you are working with button inputs and do not care about pressure sensitivity, a new handler can call your game only when the button’s pressed state changes.

GameKit Framework

The GameKit framework (`GameKit.framework`) has the following changes:

- Features that were added in iOS 7 are available on OS X 10.10, making it easier to use these features in a cross-platform game.
- The new `GKSavedGame` class and new methods in `GKLocalPlayer` make it easy to save and restore a user’s progress. The data is stored on iCloud; GameKit does the necessary work to synchronize the files between the device and iCloud.

- Methods and properties that use player identifier strings are now deprecated. Instead, use `GKPlayer` objects to identify players. Replacement properties and methods have been added that take `GKPlayer` objects.

iAd Framework

The iAd framework (`iAd.framework`) adds the following new features:

- If you are using AV Kit to play a video, you can play preroll advertisements before the video is played.
- You can look up more information about the the effectiveness of advertisements for your app.

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

Media Player Framework

Two Media Player framework (`MediaPlayer.framework`) classes are extended with new metadata information.

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

Network Extension Framework

Use the new Network Extension framework (`NetworkExtension.framework`) to configure and control virtual private networks (VPN).

SpriteKit Framework Changes

The SpriteKit framework (`SpriteKit.framework`) adds many new features:

- An `SKShapeNode` object can specify textures to be used when the shape is either stroked or filled.
- The `SKSpriteNode`, `SKShapeNode`, `SKEmitterNode`, and `SKEffectNode` classes include support for custom rendering. Use the `SKShader` and `SKUniform` classes to compile an OpenGL ES 2.0-based fragment shader and provide input data to the shader.
- `SKSpriteNode` objects can provide lighting information so that SpriteKit automatically generates lighting effects and shadows. Add `SKLightNode` objects to the scene to specify the lights, and then customize the properties on these lights and any sprites to determine how the scene is lit.
- The `SKFieldNode` class provides physics special effects you can apply to a scene. For example, create magnetic fields, add drag effects, or even generate randomized motion. All effects are constrained to a specific region of the scene, and you can carefully tune both the effect’s strength and how quickly the effect is weakened by distance. Field nodes make it easy to drop in an effect without having to search the entire list of physics bodies and apply forces to them.

- A new `SK3DNode` class is used to integrate a SceneKit scene into your game as a sprite. Each time that SpriteKit renders your scene, it renders the 3D scene node first to generate a texture, then uses that texture to render a sprite in SpriteKit. Creating 3D sprites can help you avoid creating dozens of frames of animation to produce an effect.
- New actions have been added, including support for inverse kinematic animations.
- A new system of constraints has been added to scene processing. SpriteKit applies constraints after physics is simulated, and you use them to specify a set of rules for how a node is positioned and oriented. For example, you can use a constraint to specify that a particular node in the scene always points at another node in the scene. Constraints make it easier to implement rendering rules in your game without having to tweak the scene manually in your event loop.
- A scene can implement all of the run-loop stages in a delegate instead of subclassing the `SKScene` class. Using a delegate often means that you can avoid needing to subclass the `SKScene` class.
- The `SKView` class provides more debugging information. You can also provide more performance hints to the renderer.
- You can create normal map textures for use in lighting and physics calculations (or inside your own custom shaders). Use the new `SKMutableTexture` class when you need to create textures whose contents are dynamically updated.
- You can generate texture atlases dynamically at runtime from a collection of textures.

Xcode 6 also incorporates many new SpriteKit editors. Create or edit the contents of scenes directly, specifying the nodes that appear in the scene as well as their physics bodies and other characteristics. The scene is serialized to a file and can be loaded directly by your game. The editors save you time because often you don't need to implement your own custom editors to create your game's assets.

For information about the classes of this framework, see *SpriteKit Framework Reference* and *SpriteKit Programming Guide*.

UIKit Framework

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

- Apps that use local or push notifications must explicitly register the types of alerts that they display to users by using a `UIUserNotificationSettings` object. This registration process is separate from the process for registering remote notifications, and users must grant permission to deliver notifications through the requested options.
- Local and push notifications can include custom actions as part of an alert. Custom actions appear as buttons in the alert. When tapped, your app is notified and asked to perform the corresponding action. Local notifications can also be triggered by interactions with Core Location regions.

- Collection views support dynamically changing the size of cells. Typically, you use this support to accommodate changes to the preferred text size, but you can adapt it for other scenarios too. Collection views also support more options for invalidating different portions of the layout and thereby improving performance.
- The `UISearchController` class replaces the `UISearchDisplayController` class for managing the display of search-related interfaces.
- The `UIViewController` class adopts traits and the new sizing techniques for adjusting the view controller's content, as described in Unified Storyboards for Universal Apps (page 42).
- The `UISplitViewController` class is now supported on iPhone as well as iPad. The class adjusts its presented interface to adapt to the available space. The class also changes the way it shows and hides the primary view controller, giving you more control over how to display the split view interface.
- The `UINavigationController` class has new options for changing the size of the navigation bar or hiding it altogether.
- The new `UIVisualEffect` class enables you to integrate custom blur effects into your view hierarchies.
- The new `UIPresentationController` class lets you separate the content of your view controllers from the chrome used to display them.
- The new `UIPopoverPresentationController` class handles the presentation of content in a popover. The existing `UIPopoverController` class uses the popover presentation controller to show popovers on the screen.
- The new `UIAlertController` class replaces the `UIActionSheet` and `UIAlertView` classes as the preferred way to display alerts in your app.
- The new `UIPrinterPickerController` class offers a view controller-based way to display a list of printers and to select one to use during printing. Printers are represented by instances of the new `UIPrinter` class.
- You can take the user directly to your app-related settings in the Settings app. Pass the `UIApplicationOpenSettingsURLStringUIApplicationOpenSettingsURLString` constant to the `openURL:` method of the `UIApplication` class.

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

Video Toolbox Framework

The Video Toolbox framework (`VideoToolbox.framework`) includes direct access to hardware video encoding and decoding.

Webkit Framework

A number of new classes have been added to make it easier to create high-performance native apps that utilize web content. See *WebKit Framework Reference* .

Deprecated APIs

The following APIs are deprecated:

- The `UIApplication` methods and properties for registering notifications. Use the new API instead.
- The `UIViewController` methods and properties for interface orientation. Traits and size classes replace them, as described in Unified Storyboards for Universal Apps (page 42). There are other smaller changes to UIKit API to support size classes; often older interfaces that used specific device idioms have been replaced.
- The `UISearchDisplayController` class. This class is replaced by the `UISearchController` class.
- Methods and properties in GameKit that use player identifier strings.

For a complete list of specific API deprecations, see *iOS 8.0 API Diffs* .

iOS 7.1

This article summarizes the key developer-related features introduced in iOS 7.1. 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 7.1 Release Notes* . For the complete list of new APIs added in iOS 7.1, see *iOS 7.1 API Diffs* .

Support for External Media Players

Apps can now receive and respond to events sent by an external media player. This process lets users interact solely with the external media player, removing the need to interact directly with an iOS device.

When an app starts, it loads a data source, either from the device or from a server, that contains the available media items and provides this information to the media player. The media player reads and displays this information to the user.

The media player interacts with the app by sending events that the app has registered for. The app responds to an event and changes its behavior based on the event received.

To support this behavior, the Media Player framework has several new classes, including the following:

- The `MPPlayableContentManager` class controls the interactions between the app and the external media player. A data source is provided to the content manager through the `MPPlayableContentDataSource` protocol.
- The `MPContentItem` class contains the metadata for a particular media item. This metadata is used to display the media item information outside the app. An `MPContentItem` object can represent any type of media item—for example, a song, movie, radio station, or podcast episode.
- The `MPRemoteCommand` class adds target-action pairs for event handling. There are several subclasses for handling specific events.
- The `MPRemoteCommandEvent` class provides information that is requested by the media player. There are several subclasses for providing information on specific events.

For information about all of the external media player APIs, see *Media Player Framework Reference* .

OpenGL ES

The OpenGL ES framework includes support for automatic multithreading of some OpenGL ES work. After creating a `EAGLContext` object, set its `multiThreaded` property to `YES`. OpenGL ES then creates a worker thread and attempts to unload some of the command processing onto the worker thread. As with implementing concurrency manually, you need to consider a concurrency strategy that works best for your app. Always test your app under a variety of different conditions and use the design that provides the best performance.

iOS 7.0

iOS 7 is a major update with compelling features for developers to incorporate into their apps. The user interface has been completely redesigned. In addition, iOS 7 introduces a new animation system for creating 2D and 2.5D games. Multitasking enhancements, peer-to-peer connectivity, and many other important features make iOS 7 the most significant release since the first iPhone SDK.

This article summarizes the key developer-related features introduced in iOS 7. This version of the operating system runs on current iOS 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 7.0 Release Notes* . For the complete list of new APIs added in iOS 7, see *iOS 7.0 API Diffs* .

User Interface Changes

iOS 7 includes many new features intended to help you create great user interfaces.

UI Redesign

The iOS 7 user interface has been completely redesigned. Throughout the system, a sharpened focus on functionality and on the user's content informs every aspect of design. Translucency, refined visual touches, and fluid, realistic motion impart clarity, depth, and vitality to the user experience. Whether you are creating a new app or updating an existing one, keep these qualities in mind as you work on the design.

Apps compiled against the iOS 7 SDK automatically receive the new appearance for any standard system views when the app is run on iOS 7. If you use Auto Layout to set the size and position of your views, those views are repositioned as needed. But there may still be additional work to do to make sure your interface has the appearance you want. Similarly, if you customize your app's views, you may need to make changes to support the new appearance fully.

For guidance on how to design apps that take full advantage of the new look in iOS 7, see iOS 7 Design Resources.

Dynamic Behaviors for Views

Apps can now specify dynamic behaviors for `UIView` objects and for other objects that conform to the `UIDynamicItem` protocol. (Objects that conform to this protocol are called *dynamic items*.) Dynamic behaviors offer a way to improve the user experience of your app by incorporating real-world behavior and characteristics, such as gravity, into your app's animations. UIKit supports the following types of dynamic behaviors:

- A `UIAttachmentBehavior` object specifies a connection between two dynamic items or between an item and a point. When one item (or point) moves, the attached item also moves. The connection is not completely static, though. An attachment behavior has damping and oscillation properties that determine how the behavior changes over time.
- A `UICollisionBehavior` object lets dynamic items participate in collisions with each other and with the behavior's specified boundaries. The behavior also lets those items respond appropriately to collisions.
- A `UIGravityBehavior` object specifies a gravity vector for its dynamic items. Dynamic items accelerate in the vector's direction until they collide with other appropriately configured items or with a boundary.
- A `UIPushBehavior` object specifies a continuous or instantaneous force vector for its dynamic items.
- A `UISnapBehavior` object specifies a snap point for a dynamic item. The item snaps to the point with a configured effect. For example, it can snap to the point as if it were attached to a spring.

Dynamic behaviors become active when you add them to an animator object, which is an instance of the `UIDynamicAnimator` class. The animator provides the context in which dynamic behaviors execute. A given dynamic item can have multiple behaviors, but all of those behaviors must be animated by the same animator object.

For information about the behaviors you can apply, see *UIKit Framework Reference* .

Text Kit

Text Kit is a full-featured set of UIKit classes for managing text and fine typography. Text Kit can lay out styled text into paragraphs, columns, and pages; it easily flows text around arbitrary regions such as graphics; and it manages multiple fonts. Text Kit is integrated with all UIKit text-based controls to enable apps to create, edit, display, and store text more easily—and with less code than was previously possible in iOS.

Text Kit comprises new classes and extensions to existing classes, including the following:

- The `NSAttributedString` class has been extended to support new attributes.
- The `NSLayoutManager` class generates glyphs and lays out text.
- The `NSTextContainer` class defines a region where text is laid out.
- The `NSTextStorage` class defines the fundamental interface for managing text-based content.

For more information about Text Kit, see *Text Programming Guide for iOS* .

64-Bit Support

Apps can now be compiled for the 64-bit runtime. All system libraries and frameworks are 64-bit ready, meaning that they can be used in both 32-bit and 64-bit apps. When compiled for the 64-bit runtime, apps may run faster because of the availability of extra processor resources in 64-bit mode.

iOS uses the same LP64 model that is used by OS X and other 64-bit UNIX systems, which means fewer problems when porting code. For information about the iOS 64-bit runtime and how to write 64-bit apps, see *64-Bit Transition Guide for Cocoa Touch* .

Multitasking Enhancements

iOS 7 supports two new background execution modes for apps:

- Apps that regularly update their content by contacting a server can register with the system and be launched periodically to retrieve that content in the background. To register, include the `UIBackgroundModes` key with the `fetch` value in your app's `Info.plist` file. Then, when your app is launched, call the `setMinimumBackgroundFetchInterval:` method to determine how often it receives update messages. Finally, you must also implement the `application:performFetchWithCompletionHandler:` method in your app delegate.
- Apps that use push notifications to notify the user that new content is available can fetch the content in the background. To support this mode, include the `UIBackgroundModes` key with the `remote-notification` value in your app's `Info.plist` file. You must also implement the `application:didReceiveRemoteNotification:fetchCompletionHandler:` method in your app delegate.

Apps supporting either the `fetch` or `remote-notification` background modes may be launched or moved from the suspended to background state at appropriate times. In the case of the `fetch` background mode, the system uses available information to determine the best time to launch or wake apps. For example, it does so when networking conditions are good or when the device is already awake. You can also send silent push notifications—that is, notifications that do not display alerts or otherwise disturb the user.

For small content updates, use the `NSURLRequest` class. To upload or download larger pieces of content in the background, use the new `NSURLSession` class. This class improves on the existing `NSURLConnection` class by providing a simple, task-based interface for initiating and processing `NSURLRequest` objects. A single `NSURLSession` object can initiate multiple download and upload tasks, and use its delegate to handle any authentication requests coming from the server.

For more information about the new background modes, see App States and Multitasking in *App Programming Guide for iOS* .

Games

iOS 7 includes enhanced support for games.

Sprite Kit Framework

The Sprite Kit framework (`SpriteKit.framework`) provides a hardware-accelerated animation system optimized for creating 2D and 2.5D games. Sprite Kit provides the infrastructure that most games need, including a graphics rendering and animation system, sound playback support, and a physics simulation engine. Using Sprite Kit frees you from creating these things yourself, and it lets you focus on the design of your content and the high-level interactions for that content.

Content in a Sprite Kit app is organized into scenes. A scene can include textured objects, video, path-based shapes, Core Image filters, and other special effects. Sprite Kit takes those objects and determines the most efficient way to render them onscreen. When it is time to animate the content in your scenes, you can use Sprite Kit to specify explicit actions you want performed, or you can use the physics simulation engine to define physical behaviors (such as gravity, attraction, or repulsion) for your objects.

In addition to the Sprite Kit framework, there are Xcode tools for creating particle emitter effects and texture atlases. You can use the Xcode tools to manage app assets and update Sprite Kit scenes quickly.

For more information about how to use Sprite Kit, see *SpriteKit Programming Guide* . To see an example of how to use Sprite Kit to build a working app, see *code:Explained Adventure* .

Game Controller Framework

The Game Controller framework (`GameController.framework`) lets you discover and configure Made-for-iPhone/iPod/iPad (MFi) game controller hardware in your app. Game controllers can be devices connected physically to an iOS device or connected wirelessly over Bluetooth. The Game Controller framework notifies your app when controllers become available and lets you specify which controller inputs are relevant to your app.

For more information about supporting game controllers, see *Game Controller Programming Guide* .

Game Center Improvements

Game Center includes the following improvements:

- Turn-based matches now support a new feature known as *exchanges*. Exchanges let players initiate actions with other players, even when it is not their turn. You can use this feature to implement simultaneous turns, player chats, and trading between players.
- The limit on per-app leaderboards has been raised from 25 to 100. You can also organize your leaderboards using a `GKLeaderboardSet` object, which increases the limit to 500.
- You can add conditions to challenges that define when the challenge has been met. For example, a challenge to beat a time in a driving game might stipulate that other players must use the same vehicle.
- The framework has improved its authentication support and added other features to prevent cheating.

For more information about how to use the new Game Center features, see *Game Center Programming Guide* . For information about the classes of the Game Kit framework, see *GameKit Framework Reference* .

Maps

The Map Kit framework (`MapKit.framework`) includes numerous improvements and features for apps that use map-based information. Apps that use maps to display location-based information can now take full advantage of the 3D map support found in the Maps app, including controlling the viewing perspective programmatically. Map Kit also enhances maps in your app in the following ways:

- Overlays can be placed at different levels in the map content so that they appear above or below other relevant data.
- You can apply an `MKMapCamera` object to a map to add position, tilt, and heading information to its appearance. The information you specify using the camera object imparts a 3D perspective on the map.
- The `MKDirections` class lets you ask for direction-related route information from Apple. You can use that route information to create overlays for display on your own maps.
- The `MKGeodesicPolyline` class lets you create a line-based overlay that follows the curvature of the earth.
- Apps can use the `MKMapSnapshotter` class to capture map-based images.
- The visual representation of overlays is now based on the `MKOverlayRenderer` class, which replaces overlay views and offers a simpler rendering approach.
- Apps can now supplement or replace a map's existing tiles using the `MKTileOverlay` and `MKTileOverlayRenderer` classes.

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

AirDrop

AirDrop lets users share photos, documents, URLs, and other kinds of data with nearby devices. AirDrop support is now built in to the existing `UIActivityViewController` class. This class displays different options for sharing the content that you specify. If you are not yet using this class, you should consider adding it to your interface.

To receive files sent via AirDrop, do the following:

- In Xcode, declare support for the document types your app supports. (Xcode adds the appropriate keys to your app's `Info.plist` file.) The system uses this information to determine whether your app can open a given file.
- Implement the `application:openURL:sourceApplication:annotation:` method in your app delegate. (The system calls this method when a new file is received.)

Files sent to your app are placed in the `Documents/Inbox` directory of your app's home directory. If you plan to modify the file, you must move it out of this directory before doing so. (The system allows your app to read and delete files in this directory only.) Files stored in this directory are encrypted using data protection, so you must be prepared for the file to be inaccessible if the device is currently locked.

For more information about using an activity view controller to share data, see *UIActivityViewController Class Reference* .

Inter-App Audio

The Audio Unit framework (`AudioUnit.framework`) adds support for Inter-App Audio, which enables the ability to send MIDI commands and stream audio between apps on the same device. For example, you might use this feature to record music from an app acting as an instrument or use it to send audio to another app for processing. To vend your app's audio data, publish a I/O audio unit (`AURemoteIO`) that is visible to other processes. To use audio features from another app, use the audio component discovery interfaces in iOS 7.

For information about the new interfaces, see the framework header files. For general information about the interfaces of this framework, see *Audio Unit Framework Reference* .

Peer-to-Peer Connectivity

The Multipeer Connectivity framework (`MultipeerConnectivity.framework`) supports the discovery of nearby devices and the direct communication with those devices without requiring Internet connectivity. This framework makes it possible to create multipeer sessions easily and to support reliable in-order data transmission and real-time data transmission. With this framework, your app can communicate with nearby devices and seamlessly exchange data.

The framework provides programmatic and UI-based options for discovering and managing network services. Apps can integrate the `MCBrowserViewController` class into their user interface to display a list of peer devices for the user to choose from. Alternatively, you can use the `MCNearbyServiceBrowser` class to look for and manage peer devices programmatically.

For more information about the interfaces of this framework, see *Multipeer Connectivity Framework Reference* .

New Frameworks

iOS 7 includes the following new frameworks:

- The Game Controller framework (`GameController.framework`) provides an interface for communicating with game-related hardware; see “`Game Controller Framework`” (page 58).
- The Sprite Kit framework (`SpriteKit.framework`) provides support for sprite-based animations and graphics rendering; see Sprite Kit Framework (page 58).
- The Multipeer Connectivity framework (`MultipeerConnectivity.framework`) provides peer-to-peer networking for apps; see Peer-to-Peer Connectivity (page 61).
- The JavaScript Core framework (`JavaScriptCore.framework`) provides Objective-C wrapper classes for many standard JavaScript objects. Use this framework to evaluate JavaScript code and parse JSON data. For information about the classes of this framework, see the framework header files.
- The Media Accessibility framework (`MediaAccessibility.framework`) manages the presentation of closed-captioned content in your media files. This framework works in conjunction with new settings that let the user enable the display of closed captions.
- The Safari Services framework (`SafariServices.framework`) provides support for programmatically adding URLs to the user’s Safari reading list. For information about the class provided by this framework, see the framework header files.

Enhancements to Existing Frameworks

In addition to its new features, iOS 7 also includes significant enhancements, organized here by framework. For a complete list of new interfaces, see *iOS 7.0 API Diffs* .

UIKit Framework

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

- All UI elements have been updated to present the new look associated with iOS 7.
- UIKit Dynamics lets you mimic real-world effects such as gravity in your animations; see Dynamic Behaviors for Views (page 56).
- Text Kit provides sophisticated text editing and display capabilities; see Text Kit (page 56).
- The `UIView` class defines the following additions:

  The `tintColor` property applies a tint color to both the view and its subviews. For information on how to apply tint colors, see *iOS 7 UI Transition Guide* .

  You can create keyframe-based animations using views. You can also make changes to your views and specifically prevent any animations from being performed.
- The `UIViewController` class defines the following additions:

  View controller transitions can be customized, driven interactively, or replaced altogether with ones you designate.

  View controllers can now specify their preferred status bar style and visibility. The system uses the provided information to manage the status bar style as new view controllers appear. You can also control how this behavior is applied using the `UIViewControllerBasedStatusBarAppearance` key in your app's `Info.plist` file.
- The `UIMotionEffect` class defines the basic behavior for motion effects, which are objects that define how a view responds to device-based motion.
- Collection views add support for intermediate layout transitions and invalidation contexts (invalidation contexts help you improve the performance of your custom layout code). You can also apply UIKit Dynamics to collection view layout attributes to animate the items in the collection.
- The `imageNamed:` method of `UIImage` supports retrieving images stored in asset catalogs, which are a way to manage and optimize assets that have multiple sizes and resolutions. You create asset catalogs in Xcode.
- There are methods on `UIView` and `UIScreen` for creating a snapshot of their contents. Generating snapshots using these new interfaces is significantly faster than rendering the view or screen contents yourself.

- Gesture recognizers can specify dependencies dynamically to ensure that one gesture recognizer fails before another is considered.
- The `UIKeyCommand` class wraps keyboard events received from an external hardware keyboard. These events are delivered to the app's responder chain for processing.
- A `UIFontDescriptor` object describes a font using a dictionary of attributes. Use font descriptors to interoperate with other platforms.
- The `UIFont` and `UIFontDescriptor` classes support dynamic text sizing, which improves legibility for text in apps. With this feature, the user controls the desired font size that all apps in the system should use.
- The `UIActivity` class now supports new activity types, including activities for sending items via AirDrop, adding items to a Safari reading list, and posting content to Flickr, Tencent Weibo, and Vimeo.
- The `UIApplicationDelegate` protocol adds methods for handling background fetch behaviors.
- The `UIScreenEdgePanGestureRecognizer` class is a new gesture recognizer that tracks pan gestures that originate near an edge of the screen.
- UIKit adds support for running in a guided-access mode, which allows an app to lock itself to prevent modification by the user. This mode is intended for institutions such as schools, where users bring their own devices but need to run apps provided by the institution.
- State restoration now allows the saving and restoration of any object. Objects adopting the `UIStateRestoring` protocol can write out state information when the app moves to the background and have that state restored during subsequent launches.
- Table views now support estimating the height of rows and other elements, which improves scrolling performance.
- You can now more easily configure a `UISearchDisplayController` object to work with a `UINavigationBar` object.

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

Store Kit Framework

The Store Kit framework (`StoreKit.framework`) has migrated to a new receipt system that developers can use to verify in-app purchases on the device itself. You can also use it to verify the app purchase receipt on the server.

For more information about how to use this new receipt system, see *Receipt Validation Programming Guide* .

Pass Kit Framework

The Pass Kit framework (`PassKit.framework`) includes new APIs for adding multiple passes in a single operation.

These new features were added to the pass file format:

- New keys specify the expiration date for a pass.
- You can specify that a pass is relevant only when it is in the vicinity of specific iBeacons.
- New attributes control how a pass is displayed. You can group passes together, display links with custom text on the back of a pass, and control how time values are displayed on the pass.
- You can now associate extra data with a pass. This data is available to your app but is not displayed to the user.
- You can designate which data detectors to apply to the fields of your passes.

For information about how to use Pass Kit in your app, see *Passbook Programming Guide*. For information about the pass file format, see *Passbook Package Format Reference*.

OpenGL ES

iOS 7 adds support for OpenGL ES 3.0 and adds new features to OpenGL ES 2.0.

- OpenGL ES 3.0 includes as core functionality the features of many extensions supported in OpenGL ES 2.0 on iOS. But OpenGL ES 3.0 also adds new features to the OpenGL ES shading language and new core functionality that has never been available on mobile processors before, including multiple render targets and transform feedback. You can use OpenGL ES 3 to more easily implement advanced rendering techniques, such as deferred rendering.

  To create an OpenGL ES 3 context on devices that support it, pass the `kEAGLRenderingAPIOpenGLES3` constant to the `initWithAPI:` method.
- OpenGL ES 2 adds the following new extensions:
  - The EXT_sRGB extension adds support for sRGB framebuffer operations.
  - The GL_EXT_pvrtc_sRGB extension adds support for sRGB texture data compressed in the PVRTC texture compression format. (This extension is also supported in OpenGL ES 3.0).
  - The GL_EXT_draw_instanced and GL_EXT_instanced_arrays extensions can improve rendering performance when your app draws multiple instances of the same object. You use a single call to draw instances of the same object. You add variation to each instance by specifying how fast each vertex attribute advances or by referencing an ID for each instance in your shader.

- Textures can be accessed in vertex shaders in both OpenGL ES 2.0 and 3.0. Query the value of the `MAX_VERTEX_TEXTURE_IMAGE_UNITS` attribute to determine the exact number of textures you can access. In earlier versions of iOS, this attribute always had a value of `0`.

For more information, see *OpenGL ES Programming Guide for iOS* and *iOS Device Compatibility Reference* .

Message UI Framework

In the Message UI framework, the `MFMessageComposeViewController` class adds support for attaching files to messages.

For information about the new interfaces, see the framework header files. For information about the classes of this framework, see *Message UI Framework Reference* .

Media Player Framework

In the Media Player framework, the `MPVolumeView` class provides support for determining whether wireless routes such as AirPlay and Bluetooth are available for the user to select. You can also determine whether one of these wireless routes is currently active. For information about the new interfaces, see the framework header files.

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

Map Kit Framework

The Map Kit framework (`MapKit.framework`) includes changes that are described in Maps (page 59).

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

Image I/O Framework

The Image I/O framework (`ImageIO.framework`) now has interfaces for getting and setting image metadata.

For information about the new interfaces, see the framework header files. For information about the classes of this framework, see *Image I/O Reference Collection* .

iAd Framework

The iAd framework (`iAd.framework`) includes two extensions to other frameworks that make it easier to incorporate ads into your app's content:

- The framework introduces new methods on the `MPMoviePlayerController` class that let you run ads before a movie.

- The framework extends the `UIViewController` class to make it easier to create ad-supported content. You can now configure your view controllers to display ads before displaying the actual content they manage.

For information about the new interfaces, see the framework header files. For information about the classes of this framework, see *Ad Support Framework Reference*.

Game Kit Framework

The Game Kit framework (`GameKit.framework`) includes numerous changes, which are described in Game Center Improvements (page 58).

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

Foundation Framework

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

- The `NSData` class adds support for Base64 encoding.
- The `NSURLSession` class is a new class for managing the acquisition of network-based resources. (You can use it to download content even when your app is suspended or not running.) This class serves as a replacement for the `NSURLConnection` class and its delegate; it also replaces the `NSURLDownload` class and its delegate.
- The `NSURLComponents` class is a new class for parsing the components of a URL. This class supports the URI standard (rfc3986/STD66) for parsing URLs.
- The `NSNetService` and `NSNetServiceBrowser` classes support peer-to-peer discovery over Bluetooth and Wi-Fi.
- The `NSURLCredential` and `NSURLCredentialStorage` classes let you create credentials with a synchronizable policy, and they provide the option of removing credentials with a synchronizable policy from iCloud.
- The `NSURLCache`, `NSURLCredentialStorage`, and `NSHTTPCookieStorage` classes now support the asynchronous processing of storage requests.
- The `NSCalendar` class supports new calendar types.
- The `NSProgress` class provides a general-purpose way to monitor the progress of an operation and report that progress to other parts of your app that want to use it.

For information about the new interfaces, see the framework header files and Foundation release notes. For general information about the classes of this framework, see *Foundation Framework Reference*.

Core Telephony Framework

The Core Telephony framework (`CoreTelephony.framework`) lets you get information about the type of radio technology in use by the device. Apps developed in conjunction with a carrier can also authenticate against a particular subscriber for that carrier.

For information about the new interfaces, see the framework header files. For general information about the classes of the Core Telephony framework, see *Core Telephony Framework Reference* .

Core Motion Framework

The Core Motion framework (`CoreMotion.framework`) adds support for step counting and motion tracking. With step counting, the framework detects movements that correspond to user motion and uses that information to report the number of steps to your app. Because the system detects the motion, it can continue to gather step data even when your app is not running. Alongside this feature, the framework can also distinguish different types of motion, including different motions reflective of travel by walking, by running, or by automobile. Navigation apps might use that data to change the type of directions they give to users.

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

Core Location Framework

The Core Location framework (`CoreLocation.framework`) supports region monitoring and ranging using Bluetooth devices. Region monitoring lets you determine whether the iOS device enters a specific area, and ranging lets you determine the relative range of nearby Bluetooth devices. For example, an art museum might use region monitoring to determine whether a person is inside a particular gallery, and then place iBeacons near each painting. When the person is standing by a painting, the app would display information about it.

The framework also supports deferring the delivery of location updates until a specific time has elapsed or the user has moved a minimum distance.

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

Core Foundation Framework

The Core Foundation framework (`CoreFoundation.framework`) now lets you schedule stream objects on dispatch queues.

For information about the new interfaces, see the framework header files. For general information about the interfaces of this framework, see *Core Foundation Framework Reference* .

Core Bluetooth Framework

The Core Bluetooth framework (`CoreBluetooth.framework`) includes the following enhancements:

- The framework supports saving state information for central and peripheral objects and restoring that state at app launch time. You can use this feature to support long-term actions involving Bluetooth devices.
- The central and peripheral classes now use an `NSUUID` object to store unique identifiers.
- You can now retrieve peripheral objects from a central manager synchronously.

For information about the classes of this framework, see *Core Bluetooth Framework Reference* and *Core Bluetooth Programming Guide* .

AV Foundation Framework

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

- The `AVAudioSession` class supports the following new behaviors:
  - Selecting the preferred audio input, including audio from built-in microphones
  - Multichannel input and output
- The `AVVideoCompositing` protocol and related classes let you support custom video compositors.
- The `AVSpeechSynthesizer` class and related classes provide speech synthesis capabilities.
- The capture classes add support and interfaces for the following features:
  - Discovery of a camera's supported formats and frame rates
  - High fps recording
  - Still image stabilization
  - Video zoom (true and digital) in recordings and video preview, including custom ramping
  - Real-time discovery of machine-readable metadata (barcodes)
  - Autofocus range restriction
  - Smooth autofocus for capture
  - Sharing your app's audio session during capture
  - Access to the clocks used during capture
  - Access to capture device authorization status (user must now grant access to the microphone and camera)
  - Recommended settings for use with data outputs and asset writer

- There are new metadata key spaces for supported ISO formats such as MPEG-4 and 3GPP, and improved support for filtering metadata items when copying those items from source assets to output files using the `AVAssetExportSession` class.
- The `AVAssetWriter` class provides assistance in formulating output settings, and there are new level constants for H.264 encoding.
- The `AVPlayerLayer` class adds the `videoRect` property, which you can use to get the size and position of the video image.
- The `AVPlayerItem` class supports the following changes:
    - Asset properties can be loaded automatically when `AVPlayerItem` objects are prepared for playback.
    - When you link your app against iOS 7 SDK, the behavior when getting the values of player item properties—such as the `duration`, `tracks`, or `presentationSize` properties—is different from the behaviors in previous versions of iOS. The properties of this class now return a default value and no longer block your app if the `AVPlayerItem` object is not yet ready to play. As soon as the player item's status changes to `AVPlayerItemStatusReadyToPlay`, the getters reflect the actual values of the underlying media resource. If you use key-value observing to monitor changes to the properties, your observers are notified as soon as changes are available.
- The `AVPlayerItemLegibleOutput` class can process timed text from media files.
- The `AVAssetResourceLoaderDelegate` protocol now supports loading of arbitrary ranges of bytes from a media resource.

For information about the new interfaces, see the framework header files. For general information about the classes of this framework, see *AV Foundation Framework Reference* .

Accelerate Framework

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

- Improved support for manipulating Core Graphics data types
- Support for working with grayscale images of 1, 2, or 4 bits per pixel
- New routines for converting images between different formats and transforming image contents
- Support for biquad (IIR) operations

For information about the new interfaces, see the framework header files. For general information about the functions and types of this framework, see *Accelerate Framework Reference* .

Objective-C

The Objective-C programming language has been enhanced to support modules, which yield faster builds and shorter project indexing times. Module support is enabled in all new projects created using Xcode 5. If you have existing projects, you must enable this support explicitly by modifying your project's Enable Modules setting.

Deprecated APIs

From time to time, Apple adds deprecation macros to APIs to indicate that those APIs should no longer be used in active development. When a deprecation occurs, it is not an immediate end-of-life to the specified API. Instead, it is the beginning of a grace period for transitioning off that API and onto newer and more modern replacements. Deprecated APIs typically remain present and usable in the system for a reasonable amount of time past the release in which they were deprecated. However, active development on them ceases and the APIs receive only minor changes—to accommodate security patches or to fix other critical bugs. Deprecated APIs may be removed entirely from a future version of the operating system.

As a developer, it is important that you avoid using deprecated APIs in your code as soon as possible. At a minimum, new code you write should never use deprecated APIs. And if you have existing code that uses deprecated APIs, update that code as soon as possible. Fortunately, the compiler generates warnings whenever it spots the use of a deprecated API in your code, and you can use those warnings to track down and remove all references to those APIs.

This release includes deprecations in the following technology areas:

- The Map Kit framework includes deprecations for the `MKOverlayView` class and its various subclasses. The existing overlay views have been replaced with an updated set of overlay renderer objects that descend from the `MKOverlayRenderer` class. For more information about the classes of this framework, see *MapKit Framework Reference* .
- The Audio Session API in the Audio Toolbox framework is deprecated. Apps should use the `AVAudioSession` class in the AV Foundation framework instead.
- The `CLRegion` class in the Core Location framework is replaced by the `CLCircularRegion` class. The `CLRegion` class continues to exist as an abstract base class that supports both geographic and beacon regions.
- The `UUID` property of the `CBCentral` class is deprecated. To specify the unique ID of your central objects, use the `identifier` property instead.
- The Game Kit framework contains assorted deprecations intended to clean up the existing API and provide better support for new features.

- The UIKit framework contains the following deprecations:
    - The `wantsFullScreenLayout` property of `UIViewController` is deprecated. In iOS 7 and later, view controllers always support full screen layout.
    - `UIColor` objects that provided background textures for earlier versions of iOS are gone.
    - Many drawing additions to the `NSString` class are deprecated in favor of newer variants.
- The `gethostuuid` function in the `libsyscall` library is deprecated.
- In iOS 7 and later, if you ask for the MAC address of an iOS device, the system returns the value `02:00:00:00:00:00`. If you need to identify the device, use the `identifierForVendor` property of `UIDevice` instead. (Apps that need an identifier for their own advertising purposes should consider using the `advertisingIdentifier` property of `ASIdentifierManager` instead.)

For a complete list of specific API deprecations, see *iOS 7.0 API Diffs* .

iOS 6.1

This article summarizes the key developer-related features introduced in iOS 6.1. 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 6.1 Release Notes* . For the complete list of new APIs added in iOS 6.1, see *iOS 6.1 API Diffs* .

Map Kit Searches

The Map Kit framework now lets you programmatically search for map-based addresses and points of interest. The `MKLocalSearch` class initiates a search for map-based content using a natural language string. In other words, you can enter place name information or portions of an address and have Map Kit return search results that match the information you provide. For example, searching for the string "coffee" would return the location of local coffee bars along with information about each one.

For more information about the classes of the Map Kit framework, including the new search-related classes, see *MapKit Framework Reference* .

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 and Maps 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 *Passbook 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 that performs all the authentication steps. Once set, Game Kit automatically authenticates the local player when necessary. However, Game Kit does not display the authentication interface directly. Instead, it lets your handler present the provided authentication view controller, giving you more control over exactly when the authentication user interface appears.
- 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 Center 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 *EventKit 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 *StoreKit 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 using collection views to implement your user interface, see *Collection View Programming Guide for iOS* .

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 *App Programming Guide for iOS* .

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 *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 77).
- Views now support constraint-based layout; see Auto Layout (page 77).
- The `UICollectionView` class (and its supporting classes and protocols) support the presentation of ordered collections of items; see Collection Views (page 76).
- `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 74).
- The `UIDevice` class adds a vendor-specific identifier.
- 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_EXT_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_EXT_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 75).
- 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* .

Ad Support Framework

The Ad Support framework (`AdSupport.framework`) is a new framework that provides access to an identifier that apps can use for advertising purposes. This framework also provides a flag that indicates whether the user has opted out of ad tracking. Apps are required to read and honor the opt-out flag before trying to access the advertising identifier.

For more information about this framework, see *Ad Support 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.

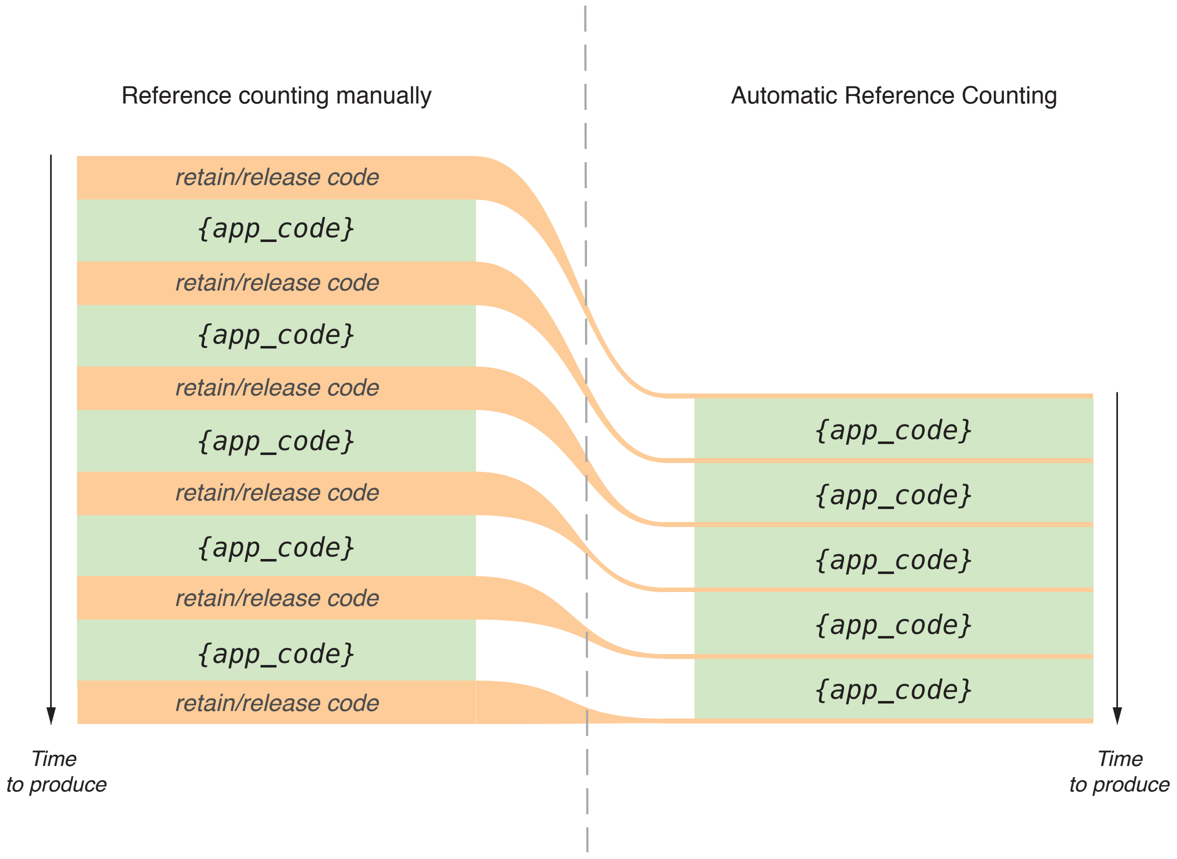
The placement of files in your app's home directory determines what gets backed up and what does not. Anything that would be backed up to a user's computer is also backed up wirelessly to iCloud. Thus, everything in the `Documents` directory and most (but not all) of your app's `Library` directory. To 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 *App Programming Guide for iOS* .

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 C 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 90).
3. Update your view controllers to handle storyboard transitions; see Preparing to Transition to a New View Controller (page 91).
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 91).

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 it is not possible to set up the segue entirely in Interface Builder—for example, when doing so involves 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 *NewsstandKit Framework Reference*. For information about how to use push notifications to notify your apps, see *Local and Remote Notification Programming Guide*. For more information about setting up Newsstand subscriptions, see *In-App Purchase 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 94).

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 86).

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 *App Programming Guide for iOS* .

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 *App Programming Guide for iOS* . 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 88).)
- 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 89).)
- 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 JavaScript Reference for iOS*.

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 97).
- The UIKit framework provides support for loading and using storyboards; see Storyboards (page 89).
- Bars and bar button items can now be tinted and customized for your app; see Custom Appearance for UIKit Controls (page 96).
- 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 *MapKit 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 105).

iAd

The iAd framework (`iAd.framework`) provides new callback methods for developers who use multiple ad 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 about the classes of the Game Kit framework, see *GameKit 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 *EventKit Framework Reference* . For information about the classes of the Event Kit UI framework, see *EventKit 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 96).
- 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.

Document Revision History

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

Date	Notes
2015-06-25	Updated for iOS 9.
2015-06-26	Updated for iOS 8.4.
2015-04-08	Added information about new features in iOS 8.3.
2014-11-18	Added information about new features in iOS 8.2.
2014-10-20	Updated for iOS 8.1.
2014-09-17	Updated for final release of iOS 8.
2014-06-02	Updated to describe new features in iOS 8.
2014-03-10	Added information about new features in iOS 7.1.
2013-09-18	Added information about new features in iOS 7.
2013-01-28	Added new features introduced in iOS 6.1.
2012-09-25	The links to the OpenGL ES extensions added in iOS 6 now point to the final versions posted on the Khronos website.
2012-09-19	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.

Date	Notes
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.