What's New in OS X

Developer

Contents

Figures and Tables

Introduction

This document describes developer-level features that were introduced in different versions of OS X.

This document is not intended as a complete list of features or changes for each new version of OS X. Instead, it focuses on those features or changes that affect the development of third-party software, providing overviews of each along with insight into how and when you might use them to create your own software. Wherever possible, this document also provides links to other Apple conceptual and reference documentation for that feature or change.

For each shipping release, this document also 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:

OS X Yosemite v10.10

This article summarizes the key technology changes and improvements in OS X v10.10. For a detailed list of API changes, see *OS X v10.10 API Diffs*.

Please file any bug reports about this release or this documentation at http://bugreport.apple.com/.

App Extensions

OS X lets you extend select areas of the system with an app extension, which is code that enables custom functionality within the context of a user task. For example, if you have an app for a great photo sharing service, you can provide an extension within your app so your service becomes available throughout OS X from the Share menu. In this example, your custom sharing extension provides the code that accepts, validates, and posts the user's photos. The system lists the extension in the sharing menu and instantiates it when the user chooses it.

You create an app extension by adding an app extension target to an app. After users install an app that contains extensions, they can enable the extensions in System Preferences. When users are running other apps, enabled extensions can be made available in the appropriate system UI, such as the sharing menu.

OS X 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 brief function in the Today view of Notification Center.
- **Finder Sync**. Enable services for custom file and sync providers.

Each extension point defines APIs that an app extension uses to communicate with the extension point. When you use an Xcode 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*.

Handoff

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 into 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 gets 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* .

Storyboards

Starting in Xcode 6, you can use storyboards to lay out all the different views you will show your users, arranging them much like the storyboard for a movie. Each view, also called a **scene**, is connected to other views by transition or containment relationships known as **segues**. You specify a trigger for a segue in terms of an action such as a button click or a menu item choice.

Mac storyboards work much like iOS storyboards do, but with a greater focus on containment of views rather than transitions between them. For example, on the Mac you express the containment between a tab view controller and its tab scenes with containment segues.

Mac storyboards depend on capabilities provided by the new view controller classes `NSSplitViewController`and `NSTabViewController`, and enhancements to the `NSViewController` and `NSWindowController` classes. For more information, refer to the overview sections in the corresponding reference documents.

iCloud

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

Document Data Migration

The iCloud infrastructure is more robust and reliable when transferring documents and data between user devices and the server. When a user installs OS X v10.10 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. This new container is accessible only to devices running OS X v10.10 or iOS 8. Devices running older operating systems will continue to have access to the original container, but changes made in that container will not appear in the new container and vice versa.

Cloud Kit

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

Apps that use Cloud Kit directly can store data in a repository that is shared by all users. This public repository is tied to the app itself, so it's 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 Cloud Kit dashboard.

For more information, see *CloudKit Framework Reference*.

Frameworks and Framework Technologies

The following sections highlight changes to frameworks and framework technologies in OS X v10.10.

API Modernization

Many frameworks on OS X 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 should continue 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*.

New Frameworks

The following frameworks are new in OS X v10.10:

- **Crypto Token Kit** (`CryptoTokenKit.framework`). The Crypto Token Kit framework provides native support for smart cards, including:
  - Enumerating connected smart card readers and monitoring them for card insertion and removal
  - Transmitting commands and responses to and from smart cards in the reader
  - Supporting new smart card reader hardware
- **FinderSync** (`FinderSync.framework`). The FinderSync framework enables Finder extensions to:
  - Express interest in specific folder hierarchies
  - Provide "badges" to indicate the status of items inside those hierarchies
  - Provide dynamic menu items in Finder contextual menus, when the selected items (or the window target) are in those hierarchies
  - Provide a Toolbar Item that displays a menu with dynamic items (even if the selection is unrelated)
- **Hypervisor** (`Hypervisor.framework`). The Hypervisor framework allows virtualization vendors to build virtualization solutions on top of OS X without needing to deploy third-party kernel extensions (KEXTs). Included is a lightweight hypervisor that enables virtualization of the host CPUs.
- **Multipeer Connectivity** (`MultipeerConnectivity.framework`). The Multipeer Connectivity framework brings OS X to parity with iOS and allows cross-platform connections between devices running iOS 7, iOS 8, and OS v10.10.
- **Notification Center** (`NotificationCenter.framework`). The Notification Center framework helps you create and manage extensions—typically called widgets—in the Today view. The framework provides an API you can use to specify whether a widget has content to display and to customize aspects of its appearance and behavior on both platforms. In OS X, the Notification Center framework also enables you to customize the editing and searching experience in a widget.

  For more information, see *Notification Center Framework Reference*.

AppKit Framework Changes

The following sections highlight changes and enhancements in the AppKit framework.

NSWindowController

`NSWindowController` has a new `contentViewController` property that mirrors the `contentViewController` property of the associated window. An `NSWindowController` object that's part of a storyboard is required to have a `contentViewController`—a requirement that the Xcode storyboard editor helps you to fulfill when you drag a window controller (`NSWindowController`) into the storyboard canvas.

NSViewController

`NSViewController` now has a suite of methods for presentation hooks:

- `viewWillAppear`
- `viewDidAppear`
- `viewWillDisappear`
- `viewDidDisappear`

See the header comments in `NSViewController.h` for more details.

`NSViewController` adds a `–viewDidLoad` method that can be overridden for instances that want to know when the view has been loaded in order to perform additional setup work.

`NSViewController` adds the following new features, all commented in great detail in `NSViewController.h`:

- Parent/child container view controller methods
- View controller presentation options
- View controller transition options

NSVisualEffectView

The new class `NSVisualEffectView` enables you to create translucent backgrounds and support vibrancy. `NSVisualEffectView` is automatically used by the system in many places, including `NSPopover` objects, `NSTableView` "source lists" and sidebars, and sheets. Please see the header for detailed information.

Gesture Recognizers

Gesture recognizers are now available in AppKit. The API is nearly identical to the UIKit version. See `NSGestureRecognizer.h`. There is a `NSGestureRecognizer (NSSubclassUse)` category that is explicitly provided for subclassers to override and call. Non-subclassers should never directly access these category methods and properties.

NSImage

In Mac OS X v10.10, named images are now retained. Previously, named images had been weakly referenced, and sometimes cached. Retention allows applications to register a named image using `-[NSImage setName:]` and retrieve it later using `+[NSImage imageNamed:]`. To achieve the same pattern on prior releases, it was necessary to hold an extra reference to the image.

AV Foundation Framework Changes

AV Foundation framework adds support for a broad cross-section of audio functionality at a higher level of abstraction than Core Audio. The new AV Foundation audio capabilities are available in both OS X and iOS and include a new API that allows manual control of the camera focus, white balance, and exposure settings. In addition, bracketed exposure captures allow automatic capturing of images with different exposure settings.

You can now capture metadata over time while shooting video. This capability allows arbitrary types of metadata to 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.

Other new capabilities in AV Foundation include:

- Audio recording and playback
- Audio file parsing and conversion
- Audio units that allow for the creation of sound effects, filters, audio distortion, and reverberation effects
- Pitch and playback speed management
- A built-in equalizer and mixer that you can use in your applications
- Stereo and 3D audio environments
- MIDI compatibility
- Automatic access to audio input and output hardware

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

Core Data Framework

The Core Data framework (`CoreData.framework`) enables you to update multiple records in a store without first loading the record data into memory.

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

Core Image Framework

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

- Accessing the second GPU in the Mac Pro is significantly easier. Use the `offlineGPUCount` method to find the number of GPUs that are not currently driving a display and can be used for image processing. Use the `offlineGPUAtIndex` method to create a `CIContext` based on an offline GPU index.
- You can use core image detectors to 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`) now allows you to determine which floor the device is on if the device is in a multistory building.

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

Foundation Framework Changes

The following sections highlight changes and enhancements in the Foundation framework.

NSString Encoding Detection

NSString adds an API that can be used to detect the string encoding of an array of bytes. This API is used to detect the string encoding of raw data and can also do lossy string conversion. It converts the data to a string in the detected string encoding.

Date Interval Formatter

The `NSDateIntervalFormatter` class is new in OS X v10.10. It's used to format a date interval in a locale-sensitive way. A date interval is defined by two—for example, September 1st, 2013 - October 1st, 2013. In different countries and different regions, people use different languages and different formats to express a date interval. `NSDateIntervalFormatter` is designed to easily format a date interval into a localized string.

Date Components Formatter

Foundation now provides localized formatting of durations and quantities of time using the new `NSDateComponentsFormatter` class. The header file `NSDateComponentsFormatter.h` is extensively commented with examples, information, and usage guidelines.

Unit Formatters

Three new unit formatters have been added in OS X v10.10: `NSMassFormatter`, `NSLengthFormatter`, and `NSEnergyFormatter`. These formatters can format a combination of a value and a unit into a localized string. They can also be used to get a localized string of a unit and to determine whether the unit is singular or plural, based on the given value. These formatters do not support either parsing or unit conversion.

NSXPCConnection Progress Reporting

In OS X v10.10, `NSXPCConnection` and `NSProgress` have been integrated to work together seamlessly. To receive progress updates from work done in another process, make an `NSProgress` object current before calling out to your `remoteObjectProxy` object. If the other side supports reporting progress, the `NSProgress` object in your process is updated as work is completed in the remote process.

NSFileCoordinator

`NSFileCoordinator` has a new reading option that facilitates uploading user documents to web services, such as web pages or mail servers, that only understand regular files and not file packages.

To use this API, request a coordinated read on any user document, the same that you would for normal reading, but use `NSFileCoordinatorReadingForUploading`. Before your accessor block is invoked, `NSFileCoordinator` determines whether the file is a directory. If it is, it creates a zip archive of that directory in a temporary folder that your app can access. If the file is not a directory, then it is copied to that directory. The URL to the newly created temporary file is accessible within your accessor block.

Game Kit Framework

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

- Features that were added in iOS 7 are now available on OS X 10.10, making it easier to use these features in a cross-platform game.
- The new `GKSavedGame` class makes it easy to save and restore a user's progress. The data is stored on iCloud; Game Kit 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.

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

Javascript Core Framework Changes

The Javascript Core Framework introduces APIs to expose and aid in debugging JavaScript in JavaScriptCore JSContexts and WebKit WebViews.

Scene Kit Framework Changes

The Scene Kit framework includes several enhancements to support developing casual 3D games and apps with 3D content:

- **Integration with Sprite Kit.** Use the `overlaySKScene` method to render a Sprite Kit scene above the 3D content in a Scene Kit view—this option is useful for adding a heads-up display to a game. You can also use Sprite Kit textures (see the `SKTexture` class) or entire animated scenes (see the `SKScene` class) in Scene Kit materials, mapping their contents onto 3D objects.
- **Game development features.** Use the `SCNAction` class for a declarative way to animate your scene (matching the `SKAction` programming model in Sprite Kit). You can also use the `SCNAction` class to implement new methods in the `SCNSceneRendererDelegate` protocol to manage your game's scene graph in a traditional update/render loop.
- **Physics.** Scene Kit physics simulation models the effects of forces and collisions on objects, and it updates the displayed node tree automatically. Use the `SCNPhysicsBody` class to add physics characteristics to nodes. Use the `SCNPhysicsWorld` object attached to your scene to control global attributes of the physics simulation.

  Advanced physics features include fields that apply forces to all bodies in their area of effect (see the `SCNPhysicsField` class); joints for connecting physics bodies (see the `SCNPhysicsBehavior` class); and vehicle behaviors that easily and efficiently simulate cars or similar vehicles (see the `SCNPhysicsVehicle` class).
- **Visual effects and animations.** Use the `subdivisionLevel` property on `SCNGeometry` objects to create smooth surfaces from low-resolution geometry. The `SCNLight` class supports new shadow rendering options. The `SCNSkinner` class can now create and edit skeletal animations for 3D models. The `SCNTechnique` class enables you to build custom shader-based post-processing techniques such as color grading, motion blur, and screen-space ambient occlusion. The `SCNParticleSystem` class implements animated 3D particle effects that you can add to a scene, such as fireworks, smoke, and rain. Use the Scene Kit Particle System editor in Xcode 6 to experiment with particle system settings.
- **Asset management.** All Scene Kit objects support the `NSSecureCoding` protocol, so you can save and restore entire scene graphs—including Scene Kit features such as physics, particle systems, and constraints—with the `NSKeyedArchiver` and `NSKeyedUnarchiver` classes. Scene Kit can now load scene content from files in Alembic (ABC) format, used widely in the 3D authoring industry, in addition to existing support for reading and writing files in Collada (DAE) format. Use `.scnassets` folders in Xcode 6 to optimize 3D assets for quick loading and smooth rendering at runtime.

In addition, the default behaviors of several Scene Kit classes change for apps built with the OS X v10.10 SDK or later:

- Animations loaded from scene files automatically play repeatedly using the system time. To change this behavior, use the `SCNSceneSourceAnimationImportPolicyKey` option.
- Custom GLSL programs (`SCNProgram`) are assumed to render opaque fragments. Use the `opaque` property to tell Scene Kit when your program requires alpha blending.
- Lights that cast shadows use forward rendering. This option improves performance but disables shadow coloring. Use the `shadowMode` property to enable deferred shadowing (the default on OS X v10.9 and earlier).
- The default type for newly created lights is `SCNLightTypeOmni`.
- The `locksAmbientWithDiffuse` property of materials defaults to `YES`.
- The `shininess` property of materials no longer has a maximum of `1.0`.

Scene Kit is also available in iOS 8, so you can build cross-platform games or create custom tools for OS X to aid development of your own 3D games for iOS.

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

Sprite Kit Framework Changes

The Sprite Kit framework (`SpriteKit.framework`) adds new features to make it easier than ever to support advanced game effects. These features include built-in support for custom OpenGL ES shaders and lighting, integration with Scene Kit, and support for advanced new physics effects and animations.

Xcode 6 also incorporates many new Sprite Kit editors, including 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*.

Store Kit Framework Changes

The Store Kit Framework adds support to the In-App Purchases API for a deferred transaction—a transaction for which there is an indeterminate delay before the transaction reaches its final state of succeeded or failed.

WebKit Framework Changes

The WebKit framework introduces a unified, extensible API for iOS and OS X and exposes the new underlying multiprocess architecture.

Deprecated Frameworks and APIs

Periodically, 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 for the specified API. Instead, it is the beginning of a grace period for transitioning from that API and to newer and more modern replacements. Deprecated APIs typically remain present and usable in the system for a reasonable 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, 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 your existing code 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. You can use those warnings to track down and remove all references to those APIs.

The following frameworks and APIs are deprecated in OS X v10.10:

- The IOAudioFamily framework
- Methods and properties in Game Kit that use player identifier strings
- The `NSGarbageCollector` class (execution support for garbage collection is not affected)
- The `NS_DEPRECATED_MAC` attribute has been added to many previously soft-deprecated AppKit methods. Searching AppKit headers for `NS_DEPRECATED_MAC` and filtering for 10_10 will yield a list of affected symbols.

Removed Frameworks

The following frameworks are no longer part of the OS X SDK as of version 10.10:

- AppleShareClientCore
- RubyCocoa

App Features

Notable in OS X v10.10 are changes to Safari.

Safari

Safari includes a variety of enhancements, such as:

- **WebGL**. Safari support for WebGL allows developers to create 3D experiences that work natively in the browser.
- **IndexedDB**. The IndexedDB API allows web developers to store structured data for web applications that work online or require large amounts of data to be cached client side.
- **JavaScript Promises**. Safari enables JavaScript authors to more naturally work with asynchronous programming patterns.
- **CSS Shapes and Compositing**. Using CSS, websites can now easily flow text around images and geometry shapes, and perform image compositing operations on DOM elements.
- **SPDY**. Safari supports SPDY, an open networking protocol that websites can adopt to reduce page load latency and improve security.
- **HTML5 Premium Video**. Websites that provide video can now take advantage of EME to deliver encrypted, energy-efficient video in the browser.

Xcode Tools Enhancements

Xcode 6 adds numerous features that support OS X v10.10, notably:

- **Swift.** Swift is a new object-oriented programming language for iOS and OS X development. Swift is modern, powerful, expressive, and easy to use.
- **View debugging.** The View Debugger makes debugging an app's appearance as easy as debugging the lines of code. A single button click pauses your running app and "explodes" the paused UI into a 3D rendering, separating each layer of the stack of views. The View Debugger immediately shows why an image may be clipped and invisible, and the order of the graphical elements becomes clear.
- **Live rendering.** Interface Builder displays your custom objects at design time, appearing as they do when your app is run. When you update the code for your custom view, the Interface Builder design canvas updates automatically with the new look you just typed into the source editor, with no need to build and run.
- **Performance testing.** The enhanced XCTest framework now supports performance tests and an asynchronous testing API, fully integrated into both Xcode and Xcode Server. Asynchronous testing allows you to test code that is distributed in asynchronously executing blocks. With performance tests, you can quantify the performance of each part of an application.
- **Extensions support.** Add an extension target to any iOS or OS X app to expose your app's functionality more deeply in the OS. Xcode connects to the extension when launched, debugging the extension as it runs in the safe, embedded OS context.

- **Storyboards for OS X.** Storyboards come to OS X with Xcode 6, taking advantage of the new View Controller APIs in App Kit. Storyboards make it easy to wire together multiple views quickly and define segue animations without writing code.

For more information about Xcode enhancements, see *What's New in Xcode*.

OS X Mavericks v10.9

This article summarizes the key technology changes and improvements available in OS X v10.9. The information about these changes is organized into sections by technology area.

- Major Features (page 21) describes important features that impact developers.
- Framework-Level Features (page 22) describes changes to system frameworks.
- App Features (page 29) describes changes in built-in apps.
- Xcode Tools Enhancements (page 30) describes changes in Xcode.
- BSD and Kernel Features (page 30) describes changes to the UNIX/POSIX portions of OS X.

For a detailed list of API changes, see *OS X v10.9 API Diffs*.

Please file any bug reports about this release or this documentation at http://bugreport.apple.com/.

Major Features

The following sections highlight OS X v10.9 features that span multiple technology areas or that are otherwise of particular importance to most developers.

App Nap

App Nap reduces power consumption by completely suspending your app's execution when it meets certain criteria. This ensures that your app does not periodically wake up to do unnecessary work. An app is considered to be a candidate for sleep if:

- It is not visible—if all of an app's windows are either hidden by other windows or minimized in a hidden dock, and the app is not in the foreground
- It is not audible
- It has not explicitly disabled automatic termination
- It has not taken any power management assertions

When all of these conditions are met, OS X may put the app to sleep. While asleep, the app is placed on a scheduling queue that rarely gets actual time on the CPU.

The app wakes up automatically when the user brings the app to the foreground or when the app receives a Mach message or Apple event.

To support activities that should not be suspended, the `NSProcessInfo` class has three new methods—`beginActivityWithOptions:reason:`, `endActivity:`, and `performActivityWithOptions:reason:block:`—to tell OS X that your app is actively doing something important. These methods do two things:

- Allow your app to temporarily suspend idle sleep, display sleep, sudden termination, and automatic termination for the duration of a particular operation
- Allow your app to temporarily increase the scheduler timer precision while your app is performing latency-critical operations

---

Note: If you experience problems with App Nap, you can temporarily disable it for a particular process by typing:

```
defaults write <app domain name> NSAppSleepDisabled -bool YES
```

Immediately file a bug describing the app and explaining how to reproduce the problem.

For more information, see *Foundation Release Notes for OS X v10.10*.

Tagging

In the Finder, the user can now add arbitrary tags to files and folders, providing additional groupings independent of the hierarchical folder structure. The user can later search for items by their tags and can assign highlight colors to files based on their tags.

As an app developer, your app can query, add, and remove tags associated with a particular URL by getting or setting the `NSURLTagNamesKey` property on the URL. In addition, your app can specify whether a save panel should show the tag names field, and can set an initial default set of tags on the `NSSavePanel` object, which the user can then modify.

For more information, see *Foundation Release Notes for OS X v10.10*.

Framework-Level Features

The following sections highlight changes to frameworks and technologies in OS X v10.9.

New Frameworks

The following frameworks have been added in OS X v10.9:

- **AV Kit** (`AVKit.framework`). AV Kit is a modern API for incorporating media playback capabilities into apps. AV Kit provides view-level services for media playback, complete with user controls, chapter navigation, and support for subtitles and closed captioning. Built on the most modern OS X media technology, AV Kit is an ideal starting point for developers looking to transition their QuickTime-based applications to AV Foundation.

  For more information, read *AVKit Framework Reference*.

- **Sprite Kit** (`SpriteKit.framework`). Sprite Kit is a powerful graphics framework for developing 2D games such as side-scrolling shooters, puzzle games, and platform games.

  When you use Sprite Kit, you set various sprite attributes such as position, size, rotation, gravity, and mass. Built-in support for actions and physics makes animations look real, and particle systems let you create essential game effects such as fire, explosions, and smoke. Sprite Kit's OpenGL-based renderer then efficiently animates 2D scenes based on the parameters you specify.

  To assist you in developing games based on Sprite Kit, Xcode provides support for texture atlas creation and includes a particle editor.

  For more information, read *SpriteKit Programming Guide*.

- **Map Kit** (`MapKit.framework`). Map Kit provides an interface for embedding maps directly into your App Store app's windows and views. It also provides support for annotating a map, adding overlays, and performing reverse-geocoding lookups to determine placemark information for a given set of map coordinates.

  For more information, read *MapKit Framework Reference*.

- **Game Controller** (`GameController.framework`). Game Controller provides a high-level Objective-C API for accepting input from Made-for-iPhone/iPod/iPad (MFi) game controllers in games. The MFi game controller specification provides a standard way for accessory developers to build controllers with analog d-pads, buttons, triggers, and thumb sticks. Game controllers declare support for a standard or extended profile that indicates which controls they provide. Games can support either profile or both, and you can tailor their behavior according to which controls are available.

  For more information, read *Game Controller Programming Guide*.

- **Media Accessibility** (`MediaAccessibility.framework`). Media Accessibility provides support for getting and setting global user preferences that control the behavior and appearance of captions and subtitles when playing movies.

- **Media Library** (`MediaLibrary.framework`). Media Library provides support for accessing media provided by iLife apps, Aperture, Final Cut Pro, Logic, and the user's Movies folder from within sandboxed apps.

  For more information, read *Media Library Framework Reference*.

- **iTunes Library** (`iTunesLibrary.framework` in `/Library/Frameworks`). iTunes Library provides support for accessing media provided by iTunes from within sandboxed apps. This framework is considered to be a public API as of iTunes 11. For more information, see *iTunes Library Framework Reference*.

Deprecated Frameworks

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 some 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, you should update that code as soon as possible. Fortunately, the compiler generates warnings whenever it spots the use of a deprecated API in your code. You can use those warnings to track down and remove all references to those APIs.

The following frameworks are deprecated in OS X v10.9:

- Instant Message framework
- QuickTime framework
- QTKit framework (except for `QTMovieModernizer`)

The Instant Message framework is superseded by the Social framework. Because the Messages application no longer provides functionality that the Instant Message framework requires, in OS X v10.9 and later this framework always returns an empty buddy list.

The QuickTime and QTKit frameworks are superseded by AV Kit and AV Foundation. You can learn more about transitioning to AV Kit and AV Foundation by reading *Transitioning QTKit Code to AV Foundation*.

The newly added `QTMovieModernizer` class in QTKit provides support for converting media encoded using older codecs into modern formats that are forward-compatible with AV Foundation.

> **Note:** The QuickTime file format (`.mov`) is **not** deprecated. It is the primary media format in OS X and iOS, and is well supported by AV Foundation.

Removed Frameworks

The following frameworks are no longer part of the OS X SDK as of version 10.9:

- `Message`—Use Apple events to ask the Mail application to send mail instead.
- `ServerNotification`—Use push notifications instead.

AppKit Framework Changes

The following sections highlight changes and enhancements in the AppKit framework. For detailed information about changes in the AppKit programming interfaces, see *AppKit Release Notes for OS X v10.10*.

Improved Multiple Monitor Support

In OS X v10.9, multiple monitor support is enhanced to allow you to pin a full-screen app on a screen while continuing to use other apps on a second display. To support this feature, AppKit has added two new `NSWindow` delegate methods: `customWindowsToEnterFullScreenForWindow:onScreen:` and `window:startCustomAnimationToEnterFullScreenOnScreen:withDuration:`.

Responsive Scrolling

Responsive scrolling is an AppKit enhancement that makes scrolling smoother. This involves two significant changes to the way your app draws content:

- Scroll views ask their child views to draw extra content outside their normal view area so that the content can be immediately made available for scrolling purposes. This additional window backing is stored in purgeable memory to minimize additional paging.
- The scrolling thread attempts to redraw the view at 60 frames per second, but it backs off if the app is unable to keep up.
- Scrolling events are processed on a background thread.

Most apps automatically receive this responsive scrolling behavior. However, some views must explicitly opt in, including layer-backed views, custom scroll view or clip view subclasses that override `drawRect:`, `NSSurface`-based document views, transparent document views, and document views that override the `lockFocus` method.

For views in which responsive scrolling is automatically enabled, the behavior change should be entirely transparent to you as a developer. However, if your app exhibits any unusual behavior while scrolling, please file bugs.

> **Note:** You can temporarily disable responsive scrolling for testing purposes by choosing File > Get Info on your app in Finder. After changing the setting, you must quit and relaunch your app.

For more details, see *AppKit Release Notes for OS X v10.10*.

Stack Views

`NSStackView` is a new Auto Layout–based view that creates and manages the constraints needed to create horizontal or vertical stacks of views. It dynamically adds and removes its constraints when views are removed or added to its stack. With customization, it can also react and influence the layout around it—implicitly growing and shrinking when views get removed or added, dropping views from its stack when its size becomes too small, and so on.

Media Library Browser Controller

`NSMediaLibraryBrowserController` is a new controller class that can provide browsing of and drag-and-drop from the media libraries provided by the Media Library framework.

Core Image Framework Changes

Core Image now takes advantage of OpenCL on the GPU to perform image processing operations on recent Macs. It also adds additional standard filters (`CIFilter`) that perform common low-level operations that are used across a wide array of image processing tasks.

Core Services Framework Changes

The following APIs have been removed in the OS X v10.9 SDK:

- Open Transport
- Power management (`Power.h`)

Disk Arbitration Framework Changes

The `DAApprovalSessionRef` type has been replaced by `DASessionRef` throughout this framework. (The two types are functionally equivalent.)

Foundation Framework Changes

The following sections highlight changes and enhancements in the Foundation framework. For detailed information about changes in the Foundation programming interfaces, see *Foundation Release Notes for OS X v10.10*.

NSCalendar Provides More Sophisticated Date and Time Computation

The `NSCalendar` class adds the `enumerateDatesStartingAfterDate:matchingComponents:options:usingBlock:` method for iterating forwards or backwards through dates that match a particular pattern until the callback block tells it to stop. For example, an `NSCalendar` object might call your block for every Saturday beginning from the current date.

Also, existing `NSCalendar` methods that convert between date and date components now use stricter parameter checking to prevent unreasonable combinations of day-identifying calendrical units. The allowed combinations are:

- Era Year Month Day-of-month
- Era Year Month Weekday-ordinality Weekday
- Era Year Month Week-of-month Weekday
- Era Year-for-week-of-year Week-of-year Weekday

along with various harmless subsets of the above, in combination with hour, minute, and so on. For example, using the `NSYearForWeekOfYearCalendarUnit` and `NSYearCalendarUnit` units in the same call to `components:fromDate:` now returns an error.

NSURL Lets You Examine URL Components

The `NSURL` class and the `NSURLComponents` class provide significantly expanded support for obtaining and working with portions of URLs, including path components, host, port, query strings, and so on. This class supersedes the path component functionality in `NSString`.

In addition, the `NSURL` class provides support for getting and setting tags on files and directories.

NSProcessInfo Now Provides Power Management Support

The `NSProcessInfo` class provides the `beginActivityWithOptions:reason:`, `endActivity:`, and `performActivityWithOptions:reason:block:` methods (and associated constants) for temporarily suspending sudden termination, automatic termination, throttling (App Nap), and idle and display sleep.

NSProgress Provides Progress Notifications

The `NSProgress` class provides a standard mechanism that can be used for providing notification of the progress of long-running operations—downloads, for example—to interested parties.

NSNetServices Supports Listening For Connections

The `NSNetService` class now provides a lightweight way to listen for incoming connections. If you set the `NSNetServiceListenForConnections` option flag in the options value passed to `publishWithOptions:`, OS X automatically handles all of the connection management for you.

When a client connects, the `NSNetService` object calls its delegate's `netService:didAcceptConnectionWithInputStream:outputStream:` method, passing it a pair of `NSStream` objects that represent the newly established connection.

NSData Base64 Support

The `NSData` class now provides the following methods for converting to and from Base64 encoding:

- `initWithBase64EncodedString:options:`
- `base64EncodedStringWithOptions:`
- `initWithBase64EncodedData:options:`
- `base64EncodedDataWithOptions:`

NSURLSession Provides Background Downloading and Improved Configurability

The `NSURLSession` class is a new API for making HTTP requests in the context of a browser or in other situations where multiple requests are related. It provides the ability to resume downloads and uploads in the background, with notification when those downloads are complete. It also provides support for custom cookie stores, authentication, and caching in a manner that provides greater transparency and that gives your app greater control compared with earlier `NSURL` APIs.

JavaScript Core Framework Changes

The JavaScript Core framework (`JavaScriptCore.framework`) now provides cross-platform (OS X and iOS) 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 header files.

Open Directory Framework Changes

The Open Directory plug-in API has been redesigned for OS X v10.9. Existing Open Directory plug-ins must be rewritten to conform to the new API. The new plug-in scheme is significantly less complex than the previous scheme, and Xcode templates are available to make creating these plug-ins easier. For more information, see *Open Directory Release Notes*.

OpenGL Framework Changes

OS X v10.9 supports the OpenGL 4.1 Core Profile on Macs with GPUs that are capable of supporting its feature set. Read *OpenGL Programming Guide for Mac* to learn about choosing a profile.

OpenCL Framework Changes

OS X v10.9 supports OpenCL 1.2 using the GPU on recent Macs, including those with Intel HD Graphics 4000.

Security Framework Changes

The Security framework (`Security.framework`) adds support for syncing passwords between user's devices via iCloud. Apps can mark their keychain items for iCloud via a new keychain attribute. For more information about this attribute, see the framework header files. For general information about the keychain, see *Keychain Services Programming Guide*.

In addition, a number of trust-related iOS APIs, such as `SecTrustCopyExceptions`, are now available in OS X.

Store Kit Framework Changes

OS X v10.9 supports auto-renewable subscriptions for In-App Purchases.

WebKit Framework Changes

In OS X v10.9, secure cookie storage is no longer shared between Safari and WebKit.

App Features

Notable in OS X v10.9 are changes to Messages.

Messages

The video chat functionality has been removed from the Messages app in OS X v10.9, in favor of the standardized, iOS-compatible video chat functionality built into the FaceTime app. This change potentially affects developers in two ways:

- The Instant Message Framework is no longer supported, and all methods return an empty list. Use the Social framework instead.
- Multiway video chats are no longer supported.

Xcode Tools Enhancements

Xcode 5.0 adds numerous features that support OS X v10.9, notably:

- The LLDB debugger now supports kernel extension debugging; to learn more about transitioning from GDB to LLDB, read LLDB Command Map and *LLDB Quick Start Guide*.
- The GDB debugger has been removed.

For more information about Xcode enhancements, see *What's New in Xcode*.

BSD and Kernel Features

The BSD, driver, and kernel layers of OS X have been enhanced as follows:

- OS X supports remote kernel and kernel extension debugging using LLDB.
- The OS X v10.9 kernel debug kit now includes kext macros for LLDB.
- Developers producing kernel extensions for OS X v10.9 should sign their extensions with a Developer ID certificate and install them in `/Library/Extensions/`. Extensions in `/Library/Extensions/` are not loaded unless they are properly signed.

  OS X v10.9 also warns the user when an unsigned or improperly signed kernel extension in `/System/Library/Extensions/` is loaded into the kernel or added to a kext cache.

Important: The code signing technique used for kernel extensions changed in OS X v10.8.3. Kernel extensions signed with earlier versions of OS X are treated as *unsigned* by OS X v10.9.

Kernel extensions signed using the newer signing tools are incompatible with OS X v10.7.5 and earlier. If you are producing a kernel extension that must support versions of the operating system prior to version 10.8, you must install an unsigned copy in `/System/Library/Extensions`, in addition to any signed copy in `/Library/Extensions`. Versions of OS X prior to version 10.9 will ignore the signed copy in `/Library/Extensions`. If the user upgrades to OS X v10.9 or later, the user's system will prefer the signed copy in `/Library/Extensions`.

OS X Mountain Lion v10.8

This article summarizes the key technology changes and improvements available in OS X v10.8. The information about these changes is organized into sections by technology area.

- Major Features (page 32) describes important features that impact developers.
- Framework-Level Features (page 35) describes changes to system frameworks.
- App Features (page 43) describes changes in built-in apps.
- BSD and Kernel Features (page 44) describes changes to the UNIX/POSIX portions of OS X.

For a detailed list of API changes, see *OS X v10.8 API Diffs*.

Please file any bug reports about this release or this documentation at http://bugreport.apple.com/.

Major Features

The following sections highlight OS X v10.8 features that span multiple technology areas or that are otherwise of particular importance to most developers.

Game Center

Game Center on OS X v10.8 accesses the same social-gaming network as on iOS, allowing users to track scores on a leaderboard, compare their in-game achievements, invite friends to play a game, and start a multiplayer game through automatic matching. Game Center functionality is provided in two parts:

- The Game Center app, in which users sign in to their account, discover new games and new friends, add friends to their gaming network, and browse leaderboards and achievements.
- The Game Kit framework, which contains the APIs developers use to support live multiplayer or turn-based games and adopt other Game Center features, such as in-game voice chat and leaderboard access.

> **Note:** To support Game Center in a game developed for OS X, you must sign the app with a provisioning profile that enables Game Center.

Apple supports Game Center with the online Game Center service, which performs player authentication, provides leaderboard and achievement information, and handles invitations and automatching for multiplayer games. You interact with the Game Center service only indirectly, using the Game Kit APIs.

In your game, use the Game Kit APIs to post scores and achievements to the Game Center service and to display leaderboards in your user interface. You can also use Game Kit APIs to help users find others to play with in a multiplayer game.

To learn more about adding Game Center support to your app, see *Game Center Programming Guide*.

iCloud Enhancements

OS X v10.8 introduces the iCloud Document Library, which gives users an easy way to access the app-specific documents they've stored in iCloud. When users start an app that participates in iCloud storage, the iCloud Document Library appears and allows them to open, share, duplicate, and organize their documents.

> **Note:** An app indicates that it can participate in iCloud storage by having the appropriate entitlement. (An *entitlement* confers specific capabilities or security permissions on your app.) For more information, see Note in *iCloud Design Guide*.

If your app is `NSDocument` based, you can take advantage of the following things:

- The iCloud Document Library view in the Open window
- The appropriate menu items added to your app's File menu and to a document's Versions menu
- A dialog that helps users resolve conflicts (when they open a document that they changed using another device)

To learn more about integrating iCloud into your app, see *iCloud Design Guide*.

Notification Center

Notification Center provides a way for users to receive and view app notifications in an attractive, unobtrusive way. For each app, users can specify how they want to be notified of an item's arrival; they can also reveal Notification Center to view all the items that have been delivered.

When you use the `NSUserNotification` and `NSUserNotificationCenter` APIs introduced in OS X v10.8, you can configure the user-visible portions of a notification item, schedule items for delivery, and find out when items have been delivered. You can also determine whether your app has launched as a result of a notification and, if it has, whether that notification originated locally or remotely.

Important: The existing `NSNotificationCenter`, `CFNotificationCenter`, and `CFUserNotification` APIs are *not* associated with Notification Center in OS X v10.8.

To learn more about integrating the Notification Center into your app, see *NSUserNotification Class Reference* and *NSUserNotificationCenter Class Reference*.

Sharing Service

The sharing service introduced in OS X v10.8 provides a consistent user experience for sharing content among many types of services. For example, a user might want to share a photo by attaching it to an email or a Twitter message, or sending it to another Mac user via AirDrop.

The new AppKit `NSSharingService` API allows you to get information about available services and share items with them directly. As a result, you can display a custom UI to present the services. You can also use the `NSSharingServicePicker` API to display a list of sharing services (including custom services that you define) from which the user can choose. When a service is performed, the system-provided sharing window is displayed. There the user can comment or add recipients. To learn more about the sharing service APIs, see *NSSharingService Class Reference* and *NSSharingServicePicker Class Reference*.

To allow your custom mail app to participate in sharing services, add the `MailSharingSupported` key to your `Info.plist` file and register a custom event handler, specifying the `mail` event class and the `shim` event ID. Then, in your event handler implementation, use the `shud` keyword to retrieve the event's descriptor data.

Gatekeeper

In OS X v10.8 users can turn on Gatekeeper, which allows them to block the installation of software that does not come from the Mac App Store and identified developers. If your app is not signed with a Developer ID certificate issued by Apple, it will not launch on systems that have this security option selected. If you plan to distribute your app outside of the Mac App Store, be sure to test the installation of your app on a Gatekeeper-enabled system so that you can provide a good user experience.

Xcode supports most of the tasks that you need to perform to get a Developer ID certificate and code sign your app. To learn how to submit your app to the Mac App Store—or test app installation on a Gatekeeper-enabled system—read *Tools Workflow Guide for Mac*.

Objective-C Enhancements

OS X v10.8 includes the following enhancements to Objective-C:

- Default synthesis of accessor methods for declared properties
- Object literals for `NSArray`, `NSDictionary`, and `NSNumber`
- No need for forward declarations for methods that are used only within `@implementation` blocks
- Streamlined object subscripting
- Type-safe enums

> **Important:** Beginning in OS X v10.8, garbage collection is deprecated. Use ARC (Automatic Reference Counting) instead. To learn more about ARC, see *Transitioning to ARC Release Notes*.

64-Bit Compatibility

OS X v10.8 requires a Mac that uses the 64-bit kernel. Additionally, OS X v10.8 does not support 32-bit kernel extensions (KEXTs). Both 32-bit and 64-bit apps can run in OS X v10.8.

Framework-Level Features

The following sections highlight changes to frameworks and technologies in OS X v10.8.

New Frameworks

The following frameworks have been added in OS X v10.8:

- **Accounts** (`Accounts.framework`). This framework provides a single sign-on model for supported account types. 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. To learn more about the Accounts API, see *ACAccount Class Reference*.
- **Audio Video Bridging** (`AudioVideoBridging.framework`). This framework supports Audio Video Bridging (AVB) and implements the IEEE P1722.1 draft standard. AVB enhances the quality of service and provides guaranteed latency and bandwidth for media streams over an AVB network. The Audio Video Bridging API gives you access to the Entity discovery and control protocols of IEEE P1722.1 over an Ethernet network.

- **Event Kit** (`EventKit.framework`). The Event Kit framework provides an interface for accessing a user's calendar events and reminder items. You can use the APIs in this framework to get existing events and to add new events to the user's calendar. Events that are created using Event Kit APIs are automatically propagated to the CalDAV or Exchange calendars on other devices, which allows your app to display up-to-date calendar information without requiring users to open the Calendar app. (Calendar events can include configurable alarms with rules for when they should be delivered.)

  > **Note:** The Calendar Store framework (`CalendarStore.framework`) is deprecated in OS X v10.8. Use the Event Kit framework instead.

  You can also use Event Kit APIs to access reminder lists, create new reminders, add an alarm to a reminder, set the due and start date for a reminder, and mark a reminder as complete. To learn more about the Event Kit APIs, see *EventKit Framework Reference*.

- **Game Kit** (`GameKit.framework`). As described in Game Center (page 32), the Game Kit framework provides APIs that allow your app to participate in Game Center. You can use Game Kit APIs to display leaderboards in your game and to give users the opportunity to share their in-game achievements and play multiplayer games. To learn more about using Game Kit APIs in your app, see *GameKit Framework Reference*.
- **GLKit** (`GLKit.framework`). The GLKit framework provides libraries of commonly needed functions and classes that can help reduce the effort required to create an OpenGL app. In addition, the GLKit framework includes APIs that perform several optimized mathematical operations, reduce the effort in loading texture data, and provide standard implementations of commonly needed shader effects. To learn more about using GLKit APIs in your app, see *GLKit Framework Reference*.
- **Scene Kit** (`SceneKit.framework`). The Scene Kit framework provides a high-level, Objective-C API that you can use to efficiently load, manipulate, and render 3D scenes in your app. Scene Kit allows you to import Digital Asset Exchange files (`.dae` files) that are created by popular content-creation applications and gives you access to the objects, lights, cameras, and geometry data that define a 3D scene. Using an approach based on scene graphs, Scene Kit makes it simple to modify, animate, and render your 3D scenes. For more information about using Scene Kit APIs in your app, see *Scene Kit Programming Guide*.
- **Social** (`Social.framework`). The Social framework provides an API for sending requests to supported social networking services that can perform operations on behalf of your users. You can also use the Social framework to retrieve information for integrating a user's social networking accounts into your app. To learn more about using the Social framework in your app, see *Social Framework Reference*.
- **Video Toolbox** (`VideoToolbox.framework`). The Video Toolbox framework comprises the 64-bit replacement for the QuickTime Image Compression Manager. The Video Toolbox APIs provide services for video compression and decompression, and for conversion between raster image formats stored in Core Video pixel buffers.

AppKit

The following sections highlight changes and enhancements in the AppKit framework. For detailed information about changes in the AppKit programming interfaces, see *AppKit Release Notes for OS X v10.10*.

Layer-Backed Views

The AppKit framework introduces new `NSView` APIs that enhance the implementation and performance of layer-backed views. When you use layer-backed views in your app, Core Animation can perform view animations asynchronously, which can result in better, smoother animations. To benefit from the enhancements to layer-backed views in OS X v10.8, set the `layerContentsRedrawPolicy` property on your custom views to `NSViewLayerContentsRedrawOnSetNeedsDisplay`. If you do any custom drawing, refactor it to use subviews as much as possible (you can add subviews to a view at design time, or you can allow subviews to be added lazily in the `layout` method).

In addition, you can use other new `NSView` APIs to update a view's layer declaratively. For example, you can return `YES` in `wantsUpdateLayer` and then implement `updateLayer` to set the layer's properties. Typically, you set the layer's contents to an `NSImage` object and set the `contentsCenter` property to specify the proper way to stretch the image. Or you can set simple layer properties, such as `backgroundColor` and `borderColor`. To learn more about the new `NSView` APIs, see *NSView Class Reference*.

Because a layer object often works with `CGColorRef` objects, you might need to convert between an `NSColor` object and a `CGColorRef` object frequently. To help you easily perform these conversions, the `NSColor` class provides the new `CGColor` and `colorWithCGColor:` methods. To learn more about new `NSColor` methods, see *NSColor Class Reference*.

In OS X v10.8, text rendering in transparent layers is enhanced. For example, `NSTextField`—in addition to other AppKit UI controls—can properly render text with font smoothing on a transparent layer background. At the same time, AppKit explicitly disables font smoothing in custom-rendered transparent layers to avoid drawing artifacts.

Gestures

New AppKit APIs make it easier for you to adopt modern gestures in your app and to provide a better zoom experience without redrawing your content. For example, `NSScrollView` now has built-in support for the smart zoom gesture (that is, a two-finger double-tap on a trackpad). When you provide the semantic layout of your content, `NSScrollView` can intelligently magnify the content under the pointer. In addition, you can use this new API to respond to the lookup gesture (that is, a three-finger tap on a trackpad). To learn more about the new `NSScrollView` APIs, see *NSScrollView Class Reference*.

Page-Based UI

The new `NSPageController` class, which controls swipe navigation between views or view content, helps you implement a multipage book or browsing history user experience in your app. You can use `NSPageController` and its delegate to manage navigation through a predefined set of content views (such as the pages in a book) or a history of content snapshots, which changes when the user visits different views. To learn how to use `NSPageController`, see *NSPageController Class Reference*.

Text Alternatives

The new `NSTextAlternatives` class stores a list of alternatives for a specific piece of text and can notify your app when the user chooses an alternative. To support dictation, for example, you might use `NSTextAlternatives` to present a list of alternative interpretations for a word or phrase the user speaks. If the user chooses to replace the initial interpretation with an alternative, `NSTextAlternatives` notifies you of the choice so that you can update the text appropriately.

Auto Layout

Enhancements to various AppKit classes improve their support for Auto Layout. For example, you can use the new `NSSplitView` holding priority API to specify how the panes of a split view should react to changes in the split view's size. To allow the height of a fixed-width text field to grow when its content increases, you can use the new `NSTextField` method `setPreferredMaxLayoutWidth`. If you use `NSMatrix` objects in your app, you can use the new `setAutorecalculatesCellSize` method to cause the object's cell contents to determine the cell size.

Block-Based Drawing for Offscreen Images

The new `NSImage` method `imageWithSize:flipped:drawingHandler:` allows you to create an image that delegates its drawing to a block. Because the block is invoked at draw time, the drawing can be adjusted to suit the destination's pixel density, color space, and other properties. The `imageWithSize:flipped:drawingHandler:` method can be a good alternative to using the `lockFocus` and `unlockFocus` methods, especially when you need to support moving images between displays that have different properties. To learn how to optimize your app so that it looks great on a Retina display, see *High Resolution Guidelines for OS X*.

Auto Save

In OS X v10.8 some parts of the Auto Save user experience changed. If your document-based app participates in iCloud, the system automatically saves untitled documents in iCloud as soon as the user begins editing. (If your app doesn't participate in iCloud, untitled documents are saved in the user's Documents folder.) If the

user provides a title in the Save dialog, the new title is used instead. To duplicate a document, the Duplicate menu item now allows users to supply a name for the document copy. When users close a duplicated document that has not been saved explicitly, they're asked whether they want to keep the document or delete it.

When a user edits an old document, the default setting that governs whether an alert appears has changed from on to off. Specifically, the alert that states "The document is locked because you haven't made any changes to it recently" does not appear unless the user turns on the setting in Time Machine preferences. By default, a document is considered old if it has not been changed in at least 2 weeks.

If a document-based app that adopts Auto Save is in the foreground when the user starts Time Machine, the Versions feature displays previous versions of the document.

AV Foundation

The AV Foundation framework includes the following enhancements:

- The `AVAsset` and `AVPlayerItem` classes include new properties that help you find media-resource options that accommodate language preferences, accessibility requirements, and other needs, and then select them for playback.
- New methods extend the `AVPlayer` class so that you can schedule playback to start at an arbitrary host time and can ensure that it remains synchronized with another time-based operation, such as audio playback that's performed using Audio Toolbox APIs.
- The new `AVSampleBufferDisplayLayer` class allows you to display a sequence of video frames in a Core Animation layer. You can also use this class to control timing and optional synchronization with an audio device clock.
- The new `AVPlayerItemOutput` class, in conjunction with new `AVPlayerItem` methods, allows you to get decoded video frames during playback so that your app can process them.

Core Animation

Core Animation introduces the new `drawsAsynchronously` property for a `CALayer` object. When the value of this property is `YES`, the graphics context object that is passed to the layer's `drawInContext` method can queue its drawing commands so that they are executed asynchronously. When drawing commands are executed asynchronously, drawing operations can be done more efficiently.

When you use the `drawsAsynchronously` property, make sure to handle the memory management of your app's context resources (such as shadings, images, and functions) in accordance with the requirements of the Quartz 2D API. In particular, a resource must avoid referencing transient memory (such as data on the stack) because the resource might persist after the function returns to its caller.

Core Graphics

The Core Graphics framework introduces streaming APIs for capturing updates to the display in a real-time manner and for providing scaling and color-space-conversion services. The framework also adds support for viewing and modifying metadata for popular image formats.

Core Location

The Core Location framework introduces the `CLGeocoder` and `CLPlacemark` classes to give users information about a location. For example, you can use a geocoder object to convert latitude and longitude (that pinpoint a location) to a user-friendly description of that location. The user-friendly description—which typically includes street, city, and country information—is stored in a `CLPlacemark` object. You can also use a `CLGeocoder` object to convert an address dictionary from Contacts or a human-readable address string—such as "1 Infinite Loop, Cupertino, CA 95014"—into geographical coordinates.

The Core Location framework also enables region monitoring. In particular, you can use the `startUpdatingForRegion` and `stopUpdatingForRegion` methods to receive callbacks when the user enters or exits a region that you specify. To learn more about the new classes in the Core Location framework, see *Core Location Framework Reference*.

Core Media

The Core Media framework introduces the `CMSync` API, which defines two fundamental media time services objects:

- `CMClock`, which represents a source of timing information such as `mach_absolute_time` or an audio device
- `CMTimebase`, which represents a timeline that is under app control (that is, a timeline on which you can set the time and the rate at which the media plays)

> **Note:** These objects are analogous to the QuickTime Clock and Timebase objects.

The Core Media framework also introduces the `CMAudioDeviceClock` and `CMMemoryPool` APIs. The `CMAudioDeviceClock` API provides a `CMClock` object that's synchronized to an audio device. The `CMMemoryPool` API maintains a pool of recently deallocated memory blocks so that subsequent allocations of the same size can be made more quickly. For example, clients of the Video Toolbox video compression services can get an allocator from a `CMMemoryPool` object and pass it to a `VTCompressionSession` object. As a result, the video encoder can benefit from pooling behavior when the encoder allocates memory for compressed video frames.

Core Text

The Core Text framework includes support for optical glyph bounds and unregistered data fonts. It also introduces:

- AAT (Apple Advanced Typography) text features that help you access common OpenType layout tags
- String attributes that allow you to use baseline alignments
- The `CTLineGetBoundsWithOptions` function, which gives you finer-grained control of line metrics

> **Note:** The ATS (Apple Type Services) framework is deprecated in OS X v10.8. Use Core Text APIs instead.

To learn more about using Core Text in your app, see *Core Text Programming Guide*.

Foundation

The Foundation framework introduces the following enhancements:

- The new `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 v4 random bytes, or you can base the UUID on an existing string. To learn more about the `NSUUID` class, see *NSUUID Class Reference*.

  > **Note:** `NSUUID` is not toll-free bridged with `CFUUID`. If you need to convert between `CFUUID` and `NSUUID`, use UUID strings. (For more information about `CFUUID` objects, see *CFUUID Reference*.)

- The new `NSXPCConnection` class provides a Cocoa interface to the XPC library for communicating between processes, and allows you to use your own objects and classes. To learn more about the `NSXPCConnection` class, see *NSXPCConnection Class Reference*.
- New `NSFileManager` API allows you to discover the currently active iCloud account and detect when the user logs into or out of it.

For detailed information about changes in the Foundation programming interfaces, see *Foundation Release Notes for OS X v10.10*.

OpenGL

In OS X v10.8 three functions related to OpenGL context copying are deprecated:

- `CGLCopyContext`
- `aglCopyContext`
- `copyAttributesFromContext:withMask:`

Store Kit

The Store Kit framework introduces the ability to make your In-App Purchase content available in the Mac App Store. The content can be uploaded to iTunes Connect via Xcode or Application Loader and downloaded in your app using the following Store Kit APIs:

- The `SKPaymentQueue` class has new methods to start, pause, and cancel downloads.
- The new `SKDownload` class represents the status of a download from the Mac App Store and provides methods that allow you to get information about the download, including the percentage complete and an estimate of the time remaining.

System Configuration

The System Configuration framework introduces the captive network API, which allows apps to provide a better user experience when interacting with a captive network. (A *captive network*, such as a public Wi-Fi hotspot, requires user interaction before providing Internet access.) Captive network support (CNS) improves the user's connectivity experience by identifying whether a network is captive and, if it is, looking for another app that might be able to handle the connection. If no such app can be found, CNS then attempts to log in with credentials the user has saved. If neither of these solutions works, CNS displays the network's login webpage so that the user can log in to the network. Without CNS, users might not get prompted to log in to a captive network, and so they might assume that the resulting delay is the app's fault.

You can use the captive network API to tell CNS to mark an interface as online or offline. You can also use it to provide information about the SSIDs (service set IDs) that CNS should relinquish control of in favor of apps that can handle captive network connectivity.

Carbon Core API Deprecations

In OS X v10.8, most of the APIs in the Carbon Core framework are deprecated (Carbon Core is a subframework of the Core Services umbrella framework). In many cases, there are alternative APIs you can use, such as APIs in the Core Foundation, Foundation, and Disk Arbitration frameworks.

Some of the deprecated APIs are high-level wrappers around functions in the POSIX or BSD layers, such as `malloc`, `pthread`, and `sysctl`. In place of using a deprecated wrapper function, you should use the appropriate lower-level API directly. In some cases, you can use GCD (Grand Central Dispatch) instead. Finally, some of the

deprecated APIs are no longer needed or recommended, such as the bit-operation functions in `ToolUtils.h`. (The bit-operation functions were needed in Pascal development, and C provides bit operators.) To learn more about the Carbon Core APIs that are deprecated, see *Carbon Core Deprecations*.

App Features

Notable in OS X v10.8 are changes to Safari.

Safari

Safari 6.0 includes the following new features and enhancements:

- **SVG filters**. SVG (scalable vector graphics) filters combine filter-primitive elements and light source elements into a single sophisticated filter, which you can then apply to any SVG element.
- **Web notifications**. The web notifications API sends a notification from a Safari extension or webpage. Users receive these notifications in Notification Center, along with other notifications they choose to receive.

  Note: Web notifications are available only in Safari on OS X v10.8.

- **Redesigned Web Inspector**. The Web Inspector in Safari 6.0 features a navigation sidebar—for displaying information about resources, storage, and instruments—and a details sidebar—for quickly and easily editing the CSS for any DOM element. The new Web Inspector also includes a more contextual JavaScript evaluation area that is separate from the error log, streamlined instrumentation and profiling tools, and a more accessible page source view.
- **The Web Audio API**. JavaScript Web Audio API synthesizes and processes audio in web apps having complex audio requirements, such as a modern game audio engine or an interactive audio production app.
- **HTML5 media controllers**. Using an HTML5 media controller, you can coordinate the playback of multiple HTML5 media elements. For example, you can use a media controller to synchronize a sign-language-interpretation track with a video track.
- **HTML5-timed text tracks**. You can use an HTML5-timed text track to specify the timing at which text (such as captions or subtitles) appears within an HTML5 video element.
- **CSS filters**. You can use CSS filters to apply pixel effects, such as invert and blur, to an image or webpage element. You can also combine CSS filters, and you can use CSS transitions and animations to animate changes to a filter.

To learn more about Safari 6.0, read Safari and WebKit Release Notes (login required).

Xcode Tools Enhancements

Xcode 4.4 adds numerous features that support OS X v10.8, notably:

- Compiler support for the Objective-C enhancements described in Objective-C Enhancements (page 35)
- The ability to open a project in multiple workspaces
- Enhanced internationalization support, including the ability to add a base locale to your project and define a strings file that contains localized strings

For more information about Xcode enhancements, see *What's New in Xcode*.

BSD and Kernel Features

Grand Central Dispatch (GCD) and XPC (interprocess communication) objects support Automated Reference Counting (ARC) in Objective-C. Using GCD and XPC with ARC requires a minimum deployment target of OS X v10.8 (this functionality is unavailable if you have a 32-bit Intel machine).

OS X Lion v10.7

This article summarizes the key technology changes and improvements that are available beginning with OS X Lion v10.7. The information about these changes is organized into sections by technology area:

- Major Features (page 45) describes overarching changes that span multiple technology areas or are otherwise of particular importance.
- Framework-Level Features (page 50) describes changes to system frameworks.
- BSD And Kernel Features (page 55) describes changes to the UNIX/POSIX portions of OS X and to the OS X kernel, device drivers, and kernel extensions.
- App Features (page 59) describes changes in the behavior of built-in apps such as Safari.
- OS X Server (page 60) describes changes to OS X Server.

For a complete, detailed list of API changes, read *OS X v10.7 API Diffs*.

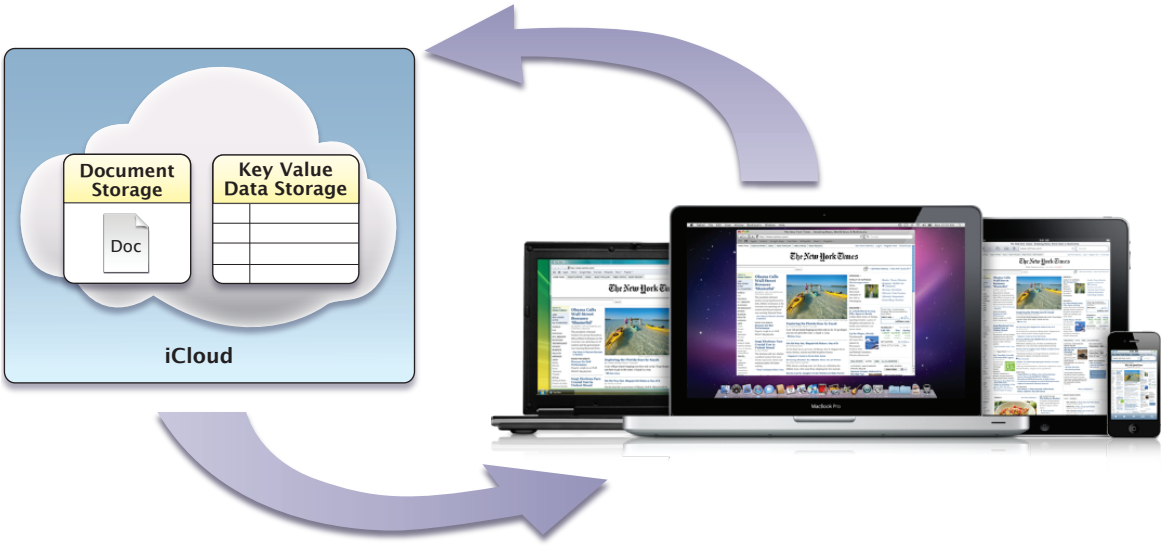
Please file any bug reports about this seed or this documentation using http://bugreport.apple.com/.

Major Features

The following sections highlight features that span multiple technology areas or that are otherwise of particular importance in OS X v10.7.

iCloud Storage APIs

When you use the iCloud storage APIs, your app writes user documents and data to a central location. 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.



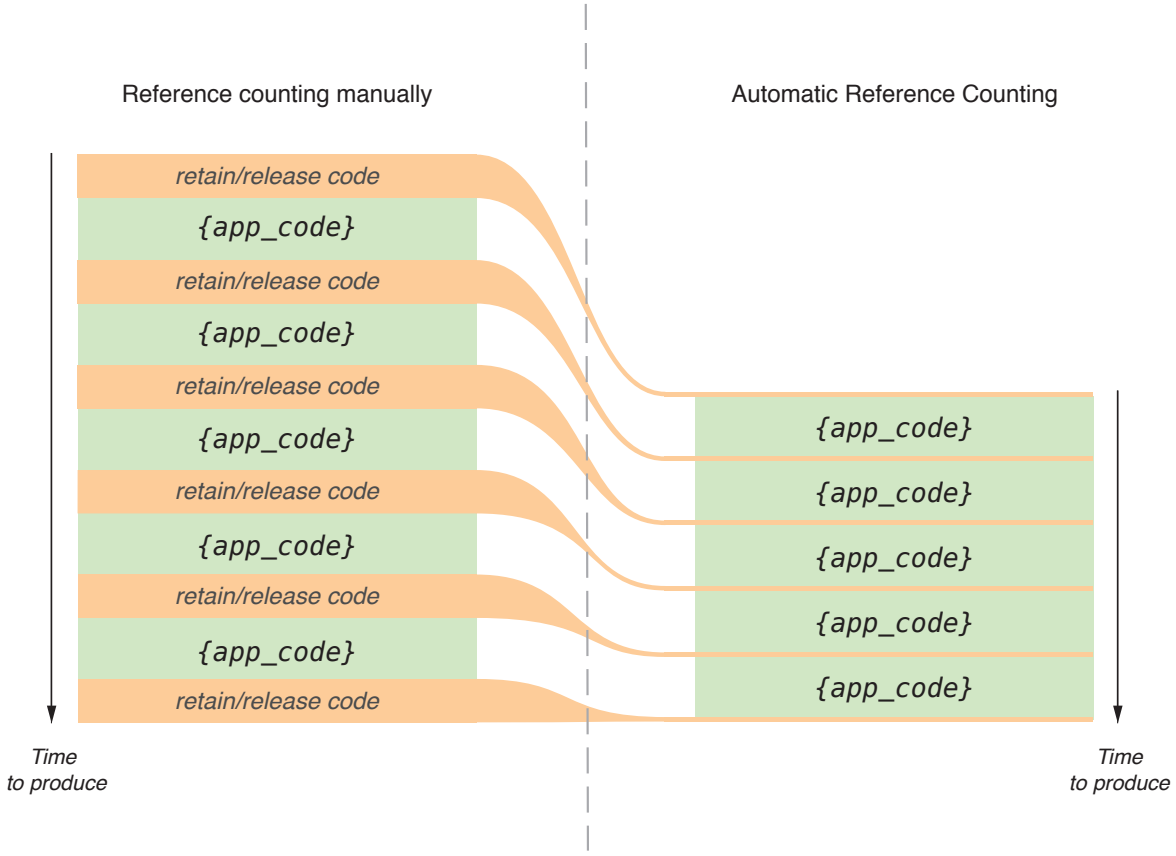
There are two types of iCloud storage that your app can take advantage of:

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

To learn more about adding iCloud storage to your app, read iCloud Storage in *Mac App Programming Guide*.

Automatic Reference Counting

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



ARC replaces the traditional managed-memory-model style of programming found in earlier versions of OS X. Any new projects you create automatically use ARC. And 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*.

Resume

Resume is a systemwide enhancement of the user experience, introduced in OS X v10.7, that supports app persistence. A user can log out or shut down the operating system, and on next login or startup, OS X automatically relaunches the apps that were last running and restores the windows that were last opened. If your app provides the necessary support, reopened windows have the same size and location as before; in addition, window contents are scrolled to the previous position and selections are restored.

When an app that supports sudden termination goes unused or has no open windows, OS X may terminate it behind the scenes. When a user wants to use the app again, it usually relaunches instantly. Users who want to quit apps manually can still do so, but it is no longer necessary.

App persistence requires you to implement the ancillary programmatic features of automatic and sudden app termination, user interface preservation, and automatic data saving.

To learn more about the elements of Resume, read Support the Key Runtime Behaviors in Your Apps in *Mac App Programming Guide*.

Cocoa Auto Layout

Cocoa Auto Layout, a feature introduced in OS X v10.7, improves upon the "springs and struts" model previously used to lay out the elements of a user interface. **Auto Layout** is a rule-based system designed to implement the layout guidelines described in *OS X Human Interface Guidelines*. It expresses a larger class of relationships and is 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 information about this feature, see *Auto Layout Guide*.

File Coordination

File coordination, introduced in OS X v10.7, eliminates file-system inconsistencies due to overlapping read and write operations from competing processes. When you use the `NSDocument` class from the AppKit framework, you get file coordination with very little effort. To use file coordination explicitly, you employ the `NSFileCoordinator` class and the `NSFilePresenter` protocol, both from the Foundation framework.

Use file coordination only with files that you think users might share with other users, not (for example) with files written to cache or temporary locations. File coordination is an enabling technology for automatic document saving, App Sandbox, and other features introduced in OS X v10.7. For more information on file coordination, see Coordinating File Access with Other Processes in *Mac App Programming Guide*.

Full-Screen Mode

Starting in OS X v10.7, you can enable full-screen mode for your app by calling a single method. Enabling full-screen mode adds an Enter Full Screen menu item to the View menu or, if there is no View menu, to the Window menu. When a user chooses this menu item, the frontmost app or document window fills the entire screen.

You enable and manage full-screen support through methods of the `NSApplication` and `NSWindow` classes and the `NSWindowDelegate Protocol`. To find out more about this feature, read Implementing the Full-Screen Experience in *Mac App Programming Guide*.

Popovers

Starting in OS X v10.7, AppKit provides support for popovers by way of the `NSPopover` class. A popover provides a means to display additional content related to existing content on the screen. The view containing the existing content—from which the popover arises—is referred to in this context as a *positioning view*. You use an anchor to express the relation between a popover and its positioning view.

You configure a popover in terms of appearance and behavior. One important part of this task is to specify which user interactions cause the popover to close. By implementing the appropriate delegate method, you can configure a popover to detach itself and become a separate window when a user drags it.

For more information, see *NSPopover Class Reference* and *NSPopoverDelegate Protocol Reference*. For guidelines on using popovers, see Popovers in *OS X Human Interface Guidelines*.

App Sandbox

Introduced in OS X v10.7, App Sandbox provides a last line of defense against stolen, corrupted, or deleted user data if malicious code exploits your app. App Sandbox also minimizes the damage from coding errors. Its strategy is twofold:

1. App Sandbox enables you to describe how your app interacts with the system. The system then grants your app the access it needs to get its job done, and no more.
2. App Sandbox allows the user to transparently grant your app additional access by using Open and Save dialogs, drag and drop, and other familiar user interactions.

You describe your app's interaction with the system by setting entitlements in Xcode. For details on all the entitlements available in OS X, see *Entitlement Key Reference*.

Some app operations are more likely to be targets of malicious exploitation. Examples are the parsing of data received over a network and the decoding of video frames. You can improve the effectiveness of the damage containment offered by App Sandbox by separating such potentially dangerous activities into their own address spaces.

For a complete explanation of App Sandbox and how to use it, read *App Sandbox Design Guide*.

Framework-Level Features

The following sections highlight changes to frameworks and technologies in OS X v10.7.

AV Foundation

Introduced in OS X v10.7, the AV Foundation framework (`AVFoundation.framework`) provides services for capturing, playing, inspecting, editing, and reencoding time-based audiovisual media. The framework includes many Objective-C classes, with the core class being `AVAsset`; this class presents a uniform model and inspection interface for all forms and sources of audiovisual media.

For most audio recording and playback requirements, you can use the `AVAudioPlayer` and `AVAudioRecorder` classes.

AV Foundation is the recommended API for all new development involving time-based audiovisual media. AV Foundation is also recommended for transitioning existing apps that were based on QuickTime or QTKit.

> **Note:** The AV Foundation frameworks for OS X and iOS are almost identical.

For more information about the classes of the AV Foundation framework and how to use them, see *AV Foundation Programming Guide*.

AppKit

Font Collections

Starting in OS X v10.7, the `NSFontCollection` class provides all of the functionality provided by the `CTFontCollectionRef` opaque type, plus a few higher-level features to support font user interfaces. The two APIs are toll-free bridged.

For more information, see *NSFontCollection Class Reference*.

Multi-Image Drag and Drop

OS X v.10.7 introduces new classes and protocols to support multi-image drag and drop in your app. This feature allows fine-grained control over the appearance of multiple dragged icons as a drag proceeds.

When multiple items are dragged, they "flock" together. The source and destination apps can each influence the appearance of the icons at appropriate points during a drag.

For more information, see *AppKit Release Notes for OS X v10.10*, the `NSDraggingSession`, `NSDraggingItem`, and `NSDraggingImageComponent` classes, and the `NSDraggingSource` and `NSDraggingDestination` formal protocols.

Overlay Scrollbars

OS X v10.7 introduces overlay scrollbars similar to those in iOS. Unless the user overrides scrollbar appearance using System Preferences, the following behavior occurs:

- If all of a user's pointing devices support both horizontal and vertical touch scrolling, the scrollbars are hidden during normal use. They appear as an overlay on top of the window's content while the user is scrolling and remain visible briefly to allow scrollbar dragging.
- If a user has at least one external pointing device that does not support scrolling, the scrollbar is displayed at all times and the usable space in the window is reduced, as in previous versions of OS X. (These permanent scrollbars are referred to as *legacy scrollbars*.)
- If a user has no external pointing devices attached, the trackpad settings control the scrollbar behavior; if a user has disabled scrolling for the trackpad in System Preferences, legacy scrollbars are used.

Compatibility Note: There are three situations in which legacy scrollbars are used regardless of hardware or preferences:

- An `NSScrollView` with accessory views inside its scroll track
- Any subclass of `NSScroller` that does not declare itself to be overlay-scroller compatible
- Any `NSScroller` that is not managed by an `NSScrollView`

In addition, the scrollbars no longer contain scroll arrows (in overlay mode or in legacy mode).

Because these changes may affect your app layout, you should test your apps with both standard and scrolling pointing devices.

For more information, see *AppKit Release Notes for OS X v10.10*.

Scrolling Behavior

The scroll views in OS X v10.7 allow developers additional control over scrolling behavior with respect to the Magic Mouse device and trackpads. Specifically, new methods allow the developer to control the stretching behavior of `NSScrollView` objects (auto or on/off in horizontal and vertical axes) and provide content hints to prevent unintentional scrolling drifts.

Additionally, new `NSEvent` events have been added to support mouse momentum in scroll views.

For more information, see *AppKit Framework Reference*.

Text Autocorrection

OS X v10.7 provides expanded UI for text autocorrection, similar to the functionality in iOS, with the opportunity for the user to accept or reject a proposed correction. As in iOS, the spell checker learns from the user's responses, such as whether a correction was explicitly rejected, implicitly accepted, accepted at first and then revised, and so on.

For more information, see *AppKit Framework Reference*.

View-Based Tables and Outlines

`NSTableView` and `NSOutlineView` in OS X v10.7 now support the use of `NSView` objects instead of `NSCell` objects for content. This extension allows apps to easily provide a much richer interface for the content of cells within a table view.

OS X v10.7 also adds new functionality to provide animation of rows as they are added and deleted from the table view, providing a more context-based user experience.

For more information, see *AppKit Framework Reference*.

Window Support

Safari, Xcode, and other apps support an in-window find. In OS X v10.7, this functionality has been extended to allow it to be used in views other than `NSTextView`.

In addition, user-editable window titles allow the developer to programmatically begin editing the window title, typically in response to a user interaction.

For more information, see *AppKit Framework Reference*.

Core Data

OS X v10.7 adds the following enhancements:

- Core Data formalizes the concurrency model for managed object contexts by taking advantage of Grand Central Dispatch. When you create a context, you specify the dispatch queue association option it should adopt: confinement, private queue, or main queue. Contexts should always use the confinement pattern, just as they have prior to OS X v10.7. (In other words, you must not use the context on any thread other than the one on which you created it.) When sending messages to a context associated with a different queue (or thread), you must dispatch your block using the new `performBlock:` or `performBlockAndWait:` method.
- 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
  - Managing discardable edits, such as in an inspector window or view
- 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 information, see *Core Data Framework Reference*.

Foundation

Index Set Range Enumeration

OS X v10.7 adds support for enumerating ranges within an index set.

`NSIndexSet` is an abstraction of a set of indexes. Developers often also view it as a set of noncontiguous ranges, which is a natural extension of this abstraction. As a result, developers often want to be able to enumerate the contents of an `NSIndexSet` object in terms of ranges, instead of individual indexes. Beginning in OS X v10.7, `NSIndexSet` includes convenience methods to provide this enumeration functionality.

For more information, see *NSIndexSet Class Reference*.

JSON Serialization

In OS X 10.7, the Foundation framework adds a new class, `NSJSONSerialization`, to support conversion of JSON data (http://www.json.org/) into Foundation types and vice versa.

For more information, see `NSJSONSerialization` and *Foundation Framework Reference*.

Linguistic Tagging

OS X v10.7 introduces a linguistic tagging class, `NSLinguisticTagger`, that 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 the English language. The API provides a method to find out what capabilities are currently available in other languages as well.

For more information, see *NSLinguisticTagger Class Reference*.

NSFileWrapper Now In Foundation

In OS X v10.7, the `NSFileWrapper` class is in the Foundation framework instead of the AppKit framework.

For more information, see *Foundation Framework Reference* and *AppKit Framework Reference*.

Ordered Sets Support

OS X v10.7 now provides support for ordered sets. An `NSOrderedSet` object contains each element at most once, and the elements can be accessed by a compact integer index space, numbered `0 .. (N–1)`. It thus combines two of the most salient features of the `NSSet` and `NSArray` classes.

The `NSOrderedSet` class similarly borrows a lot from both `NSArray` and `NSSet` (and the mutable subclasses of each), but it is closer to `NSArray` than `NSSet`.

The main client for `NSOrderedSet` is Core Data.

For more information, see *NSOrderedSet Class Reference*.

Regular Expressions

OS X v10.7 adds a Unicode-compatible regular expression class, `NSRegularExpression`, along with related data detectors. This class is used to represent and apply regular expressions to Unicode strings. An instance of this class is an immutable representation of a compiled regular expression pattern and various option flags. The pattern syntax currently supported is that specified by International Components for Unicode (ICU).

The fundamental matching method for `NSRegularExpression` is a block iterator. This method allows clients to supply a block object, which will be invoked each time the regular expression matches a portion of the target string. There are additional convenience methods for returning all the matches as an array, the total number of matches, the first match, and the range of the first match.

For more information, see *NSRegularExpression Class Reference*.

XML Streaming Parser

Prior to OS X v.10.7, `NSXMLParser` always read the entire XML document into memory. Beginning in OS X v10.7, the class allows you to stream the data into memory as required.

For more information, see *NSXMLParser Class Reference*.

OpenGL 3.2

OS X v10.7 now provides OpenGL 3.2 on hardware capable of supporting its feature set. Read *OpenGL Programming Guide for Mac* to learn about selecting the OpenGL 3.2 Core profile.

Quick Look Thumbnails

Beginning in OS X v10.7, Quick Look provides a simple synchronous public method to create a Quick Look thumbnail or icon for a file.

Quick Look also now provides a public API for the Quick Look Preview panel, allowing the developer to display the panel.

For more information see *Quick Look Framework Reference for Mac*.

Store Kit

The Store Kit framework, previously available in iOS, is now also part of OS X. Classes of the Store Kit framework allow you to request payment from a user to purchase additional functionality or content from the Mac App Store. For more information about Store Kit, read *In-App Purchase Programming Guide* and *StoreKit Framework Reference*.

BSD And Kernel Features

The following sections highlight changes to the UNIX/POSIX portions of OS X, including command-line functionality and libSystem. They also highlight changes to the kernel, device drivers, and kernel extensions.

64-Bit Kernel Support

The list of computers that boot K64 by default has grown to include the following hardware:

- Mac Mini (early 2009 and later)
- MacBook (Aluminum; late 2008 and later)
- MacBook Air (late 2008 and later)
- MacBook Pro (mid 2007 and later)
- iMac (mid 2007 and later)
- XServe (early 2008 and later)

Folder Permissions and Ownership

A number of folders in the System and Local file system domains now have different ownership and permissions. Specifically:

- Many folders in the System domain that were previously owned by the `admin` group are now owned by the `wheel` group.
- Permissions for the root directory (`/`) are now mode 755 (writable only by root) instead of mode 775 (writable by the admin group).
- Permissions for `/Applications/Utilities` are now mode 755 (writable only by root) instead of mode 775 (writable by the admin group).
- Permissions for `/Library` are now mode 755 (writable only by root) instead of mode 775 (writable by the admin group), and are no longer sticky.

  All subdirectories within `/Library` now have mode 755 (writable only by root) permissions instead of mode 775 (writable by the admin group) *except:*

  - `/Library/Caches`
  - `/Library/Fonts`
  - `/Library/Java`
  - `/Library/QuickTimeStreaming`
  - `/Library/Receipts`
  - `/Library/Tomcat`

  The subdirectories listed above have the same permissions as in previous versions of OS X (usually mode 775, sometimes with the sticky bit set).
- Permissions for `/Network/Applications` and `/Network/Library` are now mode 555 (unwritable even by root) instead of mode 755 (writable only by root).

- Permissions for `/var/log/DiagnosticMessages` are slightly more lax, with mode 770 (writable by the admin group, unreadable by non-admin users) instead of mode 750 (writable only by root, unreadable by non-admin users).

FileVault

In OS X v10.7, FileVault provides built-in support for encrypted root volumes. When you enable encryption, the system's root volume is encrypted in the background and you can continue to use the system normally. Once encryption is enabled, on reboot or wake from sleep, you must log in before you can access any data on the drive.

Although this feature has limited developer impact (there is no public API for the underlying storage system), there are two things to notice:

- Assumptions that "boot = root" are much more likely to be wrong. (You should be allowing OS X to handle KEXT caches, so in general, this change should not cause problems.)
- The OS X UI may disallow enabling encryption if you have a particularly exotic driver stack. For example, if it is not possible to map a volume-relative block number to a physical block number.

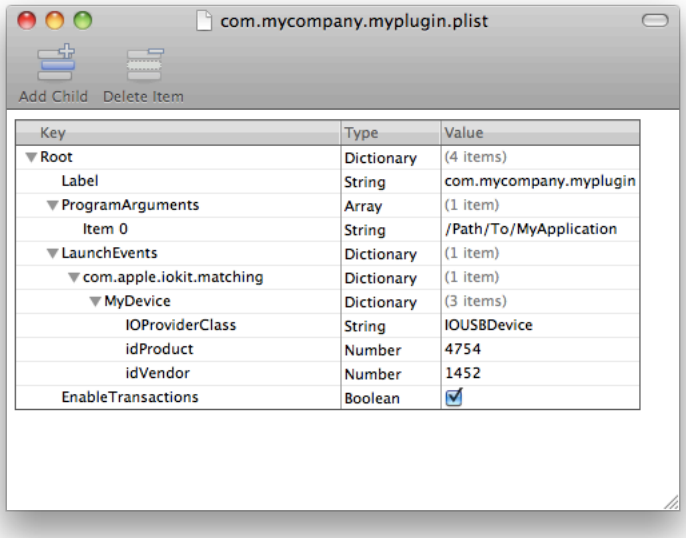
IT developers should also be aware that the FileVault UI is disabled.

In general, if your driver worked correctly with AppleRAID in previous versions of OS X, it should also work correctly with Core Storage.

I/O Kit–launchd Integration

Beginning in OS X v10.7, you can use `launchd` to automatically launch an app whenever a specific device is detected by I/O Kit.

To use this feature, add a property list file to the `/Library/LaunchAgents` folder. The property list has the standard structure of a `launchd` property list file (see `launchd.plist`), with one additional item: an I/O Kit matching dictionary.



The I/O Kit matching dictionary contains a separate dictionary for each distinct device that you want associated with your app. Each of these device dictionaries should provide the product number, vendor number, and provider class for its device.

The I/O Kit matching plug-in watches for `IOService` objects that match any of the device dictionaries you provide and launches your app when at least one is detected. If your app terminates when there is at least one matching `IOService` object, the I/O Kit matching plug-in relaunches it automatically.

Although your app is automatically launched when a corresponding device is detected, it is not automatically terminated when a device is removed. Therefore, you should configure your app to listen for notifications and respond appropriately when a device is removed. See the *USBPrivateDataSample* sample code for more information.

If your app is already running when a corresponding device is attached, no action is taken.

In-Kernel Video Capture

OS X v10.7 provides a new kernel-level programming interface for writing video capture device drivers, I/O Video, that is intended to replace the QuickTime sequence grabber API as the primary means of getting video into OS X.

I/O Video consists of (1) the `IOVideoDevice` class on the kernel side (along with various related minor classes) that your actual driver should subclass and (2) a user space device interface for communicating with the driver.

For more information, see the header file in the Kernel framework.

App Features

The following sections highlight changes to the behavior of built-in apps such as the Finder and Safari.

Safari

New Process Architecture

Beginning in OS X v10.7, Safari includes a new process architecture that separates its rendering process from its app process. As a result, Safari is more responsive, stable, and secure.

Plug-in Support in Safari

In Safari on OS X v10.7, all browser plug-ins run in their own process, improving browser stability and security. Netscape plug-ins continue to work in Safari with no modification. However, Safari does not support WebKit plug-ins. The WebKit plug-in API is not compatible with this new process architecture and is being deprecated. Plug-in developers currently using the WebKit plug-in API should adopt the Netscape plug-in API in order to be compatible with Safari on OS X v10.7. You can find the documentation at *WebKit Plug-In Programming Topics* .

> **Note:** Some WebKit plug-ins can also be replaced by Safari Extensions. To learn about Safari Extensions, read Safari Extensions Development Guide in the Safari Reference Library.

Full Screen API

The WebKit framework's `Element` and `Document` classes now provide methods to enable full-screen mode for web content and to control keyboard behavior while in full-screen mode.

The Finder

In OS X v10.7, the following files and folders are now hidden:

- `/lost+found`
- `$HOME/Library`

OS X Server

The following sections highlight changes to OS X Server.

Database Server Replacement

Beginning in OS X v10.7, OS X Server ships with PostgreSQL as its database server instead of MySQL. If you are using other software that requires MySQL, you must install it yourself.

You can find downloads and installation instructions for MySQL at the MySQL Community Edition site, http://www.mysql.com/downloads/mysql/.

In addition to the directions linked to above, you must manually reconfigure PHP if you use it to work with MySQL databases. Previous versions of PHP obtained their default values for `mysql.default_port` and `mysql.default_socket` from the `mysql-config` command-line tool. Because this tool is no longer available, you must explicitly define these values in `/etc/php.ini`.

The default port for MySQL is traditionally `3306`, and the traditional socket on OS X was `/var/mysql/mysql.sock`. For more information about the available directives, see http://us3.php.net/manual/en/mysql.configuration.php.

Profile Management

OS X now supports configuration profiles similar to those on iOS-based devices. As part of this functionality, OS X Server provides a full mobile-device-management (MDM) server. This configuration service provides you with the ability to administer both OS X clients and iOS clients.

To enable the MDM server, launch Server Center and enable the Device Config service. Then manage devices using the device-configuration web interface. There is no need to use the iPhone Configuration Utility when working with this service.

Although iOS configuration profiles are mostly compatible with OS X clients, there are some minor differences. Specifically, Exchange accounts behave differently because iOS uses ActiveSync whereas OS X uses Exchange Web Services. Also, the following features are not currently supported in OS X:

- Certain advanced network settings, such as GPRS
- Passcode policies
- Calendar subscriptions (iCal)
- The Clear Passcode command
- Find My Mac

As with iOS mobile device management, all managed computers must be able to reach the Apple Push Notification servers in order to receive push notifications.

OS X Snow Leopard v10.6

This article summarizes the key technology changes and improvements that are available beginning with OS X Snow Leopard v10.6. The information about these changes is organized into sections by technology layer:

- System Level
- Framework Level (page 69)
- Application Level (page 80)

System Level

This section introduces changes to the UNIX-level layers in OS X v10.6.

Cache Management with libcache

Aggressive caching is an important technique in maximizing application performance. However, when caching demands exceed available memory, the system must free up memory as necessary to handle new demands. Typically, this means paging cached data to and from relatively slow storage devices, sometimes even resulting in systemwide performance degradation. Your application should avoid potential paging overhead by actively managing its data caches, releasing them as soon as it no longer needs the cached data.

In the wider system context, your application can now also help by creating caches that the operating system can simply purge on a priority basis as memory pressure necessitates. OS X v10.6 includes the low-level `libcache` and framework-level `NSCache` APIs to create these purgeable caches.

The libcache API

The `libcache` API is a low-level purgeable caching API. This API uses callbacks and other concepts that should be familiar to UNIX programmers.

Using the `libcache` API is fairly straightforward. First, your application allocates a cache using `cache_create`. Next, your application stores a pointer to a block of data into the cache by calling `cache_set_and_retain`. For example:

```
cache_t *mycache
```

```
/* Attributes contain a series of callbacks for handling
   comparisons, retain/release, and so on.
 */
cache_attributes_t attrs = {
    .version = CACHE_ATTRIBUTES_VERSION_1,
    .key_hash_cb = cache_key_hash_cb_cstring,
    .key_is_equal_cb = cache_key_is_equal_cb_cstring,
    .key_retain_cb = copy_string,
    .key_release_cb = cache_release_cb_free,
    .value_release_cb = image_release_callback,
};

char *key;
void *data;
uint64_t data_len;

cache_create("com.mycompany.mycache", &attrs, &mycache);
cache_set_and_retain(mycache, key, data, data_len);
```

Note: The above example includes two callbacks specific to the data type: the key retain callback (`key_retain_cb`) and the value release callback (`value_release_cb`). The remaining callbacks are standard callbacks provided in the header file `cache_callbacks.h`.

Next, your application should release the region using `cache_release_value`. At this point, OS X can release the underlying memory if it is needed elsewhere. For example:

```
void *data;
...
cache_release_value(mycache, data);
```

When your application needs to obtain potentially cached data, it must first check to see whether the cached data exists and is still valid. It does this by calling `cache_get_and_retain`.

If the `cache_get_and_retain` operation succeeds, the cached region is still available and your application can use the cached data.

If the `cache_get_and_retain` operation fails, the cached data either was never in the cache or became unavailable while your application was not retaining it. Thus, your application must compute the data or fetch it from the original source. For example:

```
char *key;
void *data;
...
if (cache_get_and_retain(image_cache, key, &data) == 0) {
    // Do something with the value now stored in "data".
} else {
    // Recompute the value.
}
```

You can learn more about the UNIX-level API in *libcache Reference*.

Block objects

OS X v10.6 provides support for block objects in C, C++, and Objective-C. A block object (in casual parlance often referred to simply as a *block*) is a mechanism you can use to create an ad hoc function body as an expression. In other languages and environments, a block object is sometimes called a *closure* or a *lambda*. You use block objects when you need to create a reusable segment of code but defining a function or method might be a heavyweight (and perhaps inflexible) solution—for example, if you want to write callbacks with custom data or if you want to perform an operation on all the items in a collection.

For details, see *Blocks Programming Topics*.

Grand Central Dispatch

Grand Central Dispatch (GCD) is a new BSD-level infrastructure. With this simple and efficient API, application developers can achieve concurrency by avoiding blocking APIs, contended memory access, and synchronization with locking and explicit thread management. It provides a POSIX-layer task-queuing API and a dispatch source API for handling events.

Dispatch Queues

The GCD queue API provides dispatch queues from which threads take tasks to be executed. Because the threads are managed by GCD, OS X can optimize the number of threads based upon available memory, number of currently active CPU cores, and so on. This shifts a great deal of the burden of power and resource management to the operating system itself, freeing your application to focus on the actual work to be accomplished.

To provide greater control over concurrency, GCD provides two different types of queues: concurrent queues and serial queues. Both types are FIFO, but although a task in a serial queue waits until the previous task finishes executing, the task at the front of a concurrent queue can start executing as soon as a thread becomes available.

Dispatch Groups

A dispatch group allows your application to block a thread until one or more tasks finish executing. For example, after dispatching several tasks to compute some data, you might use a group to wait on those tasks and then process the results when they are done.

Dispatch Sources

Calls into the kernel or other system layers can be expensive. GCD dispatch sources replace the asynchronous callback functions typically used to mitigate the expense of handling system-related events. The GCD dispatch source API consolidates event sources into a single run-loop mechanism. This model allows you to specify the events you want to monitor and the dispatch queue and code (in the form of a block or a function) to use to process those events. When an event of interest arrives, the dispatch source submits your block or function to the specified dispatch queue for execution. GCD offers the following types of dispatch sources:

- Timer sources for periodic notifications
- Signal sources for UNIX signals
- Descriptor sources for file and socket operations
- Process sources for significant process events
- Mach port sources for Mach-related events
- Custom sources that you define and trigger

Dispatch Semaphores

You can use a dispatch semaphore to regulate the number of tasks allowed to simultaneously access a finite resource. For example, each application is given a limited number of file descriptors to use. If you have a task that processes large numbers of files, you do not want to open so many files at one time that you run out of file descriptors. Instead, you can use a semaphore to limit the number of file descriptors in use at any one time by your file-processing code.

A dispatch semaphore works like a traditional semaphore, except that when the resource is available, it takes less time to acquire a dispatch semaphore. The reason is that Grand Central Dispatch does not call into the kernel for this particular case. It calls into the kernel only when the resource is not available and the system needs to park your thread until the semaphore is signaled.

For more information about GCD, read *Concurrency Programming Guide* and *Grand Central Dispatch (GCD) Reference*.

64-Bit Kernel

OS X v10.6 includes a 64-bit kernel. Although OS X allows a 32-bit kernel to run 64-bit applications, a 64-bit kernel provides several benefits:

- The kernel can better support large memory configurations.

  Many kernel data structures such as the page table get larger as physical RAM increases. When using a 32-bit kernel with more than 32 GB of physical RAM, these data structures can consume an unmanageably large portion of the 4 GB kernel address space.

  By moving to a 64-bit kernel, the 4 GB kernel address space limitation is eliminated, and thus these data structures can grow as needed.

- The maximum size of the buffer cache is increased, potentially improving I/O performance.

  A 32-bit kernel is limited in its ability to cache disk accesses because of the 4 GB kernel address space limit.

  With a 64-bit kernel, the buffer cache can grow as needed to maximize the use of otherwise unused RAM.

- Performance is improved when working with specialized networking hardware that emulates memory mapping across a wire or with multiple video cards containing over 2 GB of video RAM.

  With a 32-bit kernel, if the combined physical address space (video RAM, for example) of all of your devices exceeds about 1.5 GB, it is impossible to map them fully into a 32-bit kernel address space at the same time. To work around this limitation, driver writers must map smaller apertures of that physical address space into the kernel's address space.

  When such a driver needs to write to or read from an unmapped address on the device, it must unmap an existing region, then map in the new region. Depending on how the mappings are managed, this extra process may cause a performance penalty, particularly for clients that exhibit low locality of reference.

  With a 64-bit kernel, the entire device can be mapped into the kernel's address space at once. This improves performance by removing the extra overhead of mapping and unmapping regions of memory. It also removes the burden of managing these mappings in your driver code, thus making the drivers simpler and less likely to generate panics by unmapping the wrong memory at the wrong time.

You must make your driver 64-bit-capable for OS X v10.6 because the 64-bit kernel does not support 32-bit drivers. Fortunately, for most drivers, this is usually not as difficult as you might think. For the most part, transitioning a driver to be 64-bit capable is just like transitioning any other piece of code. Go through and look for potential problem areas such as:

- Casting pointers to integer types
- Assigning `long` values to `int`
- Storing non-Boolean values in a `bool` variable
- Using size-variant types such as `long` or pointer types in data structures that are shared or transmitted between 32-bit and 64-bit address spaces

If you see any of these potential problems, either change the code so that they never occur, change the padding of data structures (or add unions) so that fields after size-variant types do not change positions, or use magic numbers to support different versions of data structures for 32-bit and 64-bit code and provide translation routines to convert between them.

The document *64-Bit Transition Guide* describes things to look for when checking your code and provides helpful techniques for updating your code to be 64-bit clean.

In addition to being familiar with general porting issues, you should also make sure that your driver avoids use of the `IOMemoryCursor` class and the `getPhysicalAddress` method. Use the `IODMACommand` class instead. (You should do this for 32-bit drivers on Intel-based Macs as well.)

Here are a few other 64-bit changes specific to drivers:

- Only KPI symbol sets are available. The legacy `com.apple.kernel.*` symbol sets are not exported for 64-bit KEXTs.
- The `clock_get_uptime` operation is not available (use `mach_absolute_time` instead).
- Various data structures associated with time calls have changed to allow larger time values.
- `IOCreateThread` and `IOExitThread` are not available.
- The function `kernel_thread` is not available. Use `kernel_thread_start` instead.
- The `kextload` tool now exclusively loads kernel extensions. Additional KEXT management debugging functionality that used to be in `kextload` is now in the `kextutil` tool.
- Many functions in `com.apple.unsupported` are unavailable. Consult the reference documentation and headers for further details.

These changes apply only to the `x86_64` portion of your universal KEXT binary, not to existing `i386` or `ppc` code.

In addition, kernel extension property lists now support architecture-specific properties to allow you to specify different property values for 64-bit versions of your extension. For example, you might specify the property key `OSBundleLibraries_x86_64` to specify a different value for the `OSBundleLibraries` key for the `x86_64` architecture.

> **Note:** Architecture-specific property variants are supported only for properties that begin with `IO` or `OS`. These property variants are not supported for `CFBundle` properties or any Apple-reserved properties.

Improved Shutdown

To improve the user experience in OS X v10.6, API improvements help your application contribute to shorter shutdown times.

To support improved shutdown, your application needs to mark itself as "dirty" or "clean," depending on whether it has unsaved changes and needs to do work before quitting, or can be terminated without further notice. When the system shuts down, clean applications are terminated (via `SIGKILL`) without further interaction.

Supporting Improved Shutdown in Applications

You support improved shutdown in your application by calling the `enableSuddenTermination` and `disableSuddenTermination` methods in `NSProcessInfo`. These are intended to be used as paired calls. Call `disableSuddenTermination` when you have work that must be done before quitting, and `enableSuddenTermination` when that work is done.

It's recommended that you have your application begin in the clean state by including the following entry in the plist for your application:

```
<key>NSSupportsSuddenTermination</key>

<string>YES</string>
```

This essentially launches your application with a call to `enableSuddenTermination`. Make calls to `disableSuddenTermination` when there is work that must be done before quitting, such as when you begin an export operation, and make subsequent calls to `enableSuddenTermination` when the work is complete. Note that even though these calls are in balanced pairs, they can be called on separate threads. For example, you can can disable sudden termination when you schedule an export operation on a background thread, and enable sudden termination when the operation ends, without affecting a foreground process that disables sudden termination when the user begins editing and enables it when the user saves his or her work. Sudden termination is not enabled until all calls to disable it have been balanced by calls to enable it.

For many kinds of work, you do not have to disable sudden termination explicitly. The operating system automatically disables sudden termination if an `NSDocument` object has unsaved changes, for example, or when `NSUserDefaults` or `CFPreferences` has been changed but not synchronized.

Your application should avoid data management and cleanup code that runs lazily during termination. You should carefully examine your code for work you do in `applicationWillTerminate`, overrides of `[NSApplication terminate:]`, and `[NSWindow close]`. Other places to consider are in the handlers `atexit()` and `cxa_atexit()`, and in C and C++ destructors. You should plan to rework such code as soon as possible; in the meantime, you must disable sudden termination to ensure that it will execute.

You generally do not need to perform any deallocations before quitting. The operating system can be relied on to do this for you when it terminates your application.

For more information, see *NSProcessInfo Class Reference*.

Supporting Improved Shutdown in Agents and Daemons

For agents and daemons, there is a UNIX-level function in `vproc.h` (`vproc_transaction_begin`), which corresponds to `disableSuddenTermination`, and another function (`vproc_transaction_end`), which corresponds to `enableSuddenTermination`.

> **Note:** When using the `vproc` API, note that setting the `vproc_t` parameter to `NULL` identifies the process as `self`.

Add the following to your `launchd .plist` file:

```
<key>EnableTransactions</key>

<true/>
```

You should refactor your code as necessary to avoid executing maintenance code lazily in certain places: your `SIGTERM` handler, as well as any `atexit()`, `cxa_atexit()` handlers, module finalizers, and C++ destructors. In the meantime, you must add begin and end transaction calls where appropriate to ensure that any such code continues to run.

Framework Level

This section introduces changes to the framework-level layers in OS X v10.6. Some of the more salient Cocoa enhancements are listed here. Some are described briefly later in this section; each is described in detail in the relevant ADC Reference Library documents.

- **Concurrency:**
  - Concurrent enumerations, searching, and sorting in collections
  - `NSNotificationCenter` supports concurrent notification posting
  - `NSView` supports concurrent drawing
  - `NSDocument` supports concurrent document opening
  - Enhancements and performance improvements to `NSOperation` and `NSOperationQueue`.
  - New `NSBlockOperation` class for block-based operations. See Concurrency with Operation Objects (page 72).
- **File handling and efficiency:**
  - `NSFileWrapper`: `NSURL` and `NSError` APIs, enhanced implementation
  - `NSFileManager`: `NSURL` and `NSError` APIs
  - `NSURL`: Much more complete support for file properties, along with caching for improved performance and path utilities for managing file names. See File-System Efficiency (page 72).
  - New `NSBundle` APIs using `NSURL`
- **Caching and purgeable memory:**
  - New `NSPurgeableData` data class to implement the `NSDiscardableContent` protocol.
  - New `NSCache` class. See The NSCache API (page 71).
- **Text:**
  - Improvements to bidirectional text editing
  - Improvements to text checking in `NSText`, as well as low-level APIs that can be used without `NSText`
  - New `NSOrthography` class to describe the linguistic content of a piece of text, typically used for checking spelling and grammar
  - New `NSTextInputContext` class for dealing with input management systems
- **Other Enhancements:**
  - Better-organized services menu, providing more context-sensitive and user-selectable services
  - `NSPasteboard`: Support for multiple items and a more general and flexible API
  - `NSImage`: cleaner API, better impedance match with CGImage for performance, and support for rotated images from cameras
  - Gesture and multitouch event support
  - `NSEvent` event monitoring
  - `NSCollectionView`, `NSTableView`, and `NSBrowser` enhancements
  - Ability to set desktop images

- Cocoa support for bringing up Dictionary application and Spotlight window in Finder
- New `NSRunningApplication` class to provide information about running applications
- New `NSUserInterfaceItemSearching` protocol to implement searching custom help data in applications
- More control over color spaces in `NSWindow`
- Use of formal protocols for declaring data source and delegate methods
- Block-based sheet APIs
- Block-based enumerations for lines, words, and the like in `NSString` and `NSAttributedString`
- New `NSPropertyList` APIs with better error handling and performance
- APIs and support for sudden termination. See Improved Shutdown (page 68).
- Core Data integration with Spotlight. See *Core Data Spotlight Integration Guide*.
- New `presentationOptions` API for controlling some behaviors of system UI elements (Dock, menu bar, task switcher, and so on)

The NSCache API

In addition to the UNIX-level `libcache` API described above, OS X v10.6 also provides an Objective-C API, `NSCache`. This API is conceptually similar to the `libcache` API, but `NSCache` also offers autoremoval policies to ensure that the memory footprint of the cache doesn't get too large.

To use the API, first create a new `NSCache` object. Next, add new objects for keys using `setObject:forKey:` or `setObject:forKey:cost:`. When you want to retrieve the object again, call `objectForKey:`. If the object is still available in the cache, you will get it back. Otherwise, you must recreate the object and its contents.

You can remove an item from the cache by calling `removeObjectForKey:` or clear the cache by calling `removeAllObjects`.

If desired, you can also specify advisory limits for the maximum number of items in the cache and the maximum total cost of items in the cache. You can also add your own delegate conforming to the `NSCacheDelegate` protocol. If you do, its `cache:willEvictObject:` method will be called before an object is evicted to allow you to clean up as needed.

For more information, see *Caching and Purgeable Memory*.

Purgeable Memory

The `NSCache` API is not the only way to minimize the memory footprint of your application. Cocoa also provides the `NSPurgeableData` class to help your application use memory efficiently and possibly improve performance. Paging is a time-intensive process. With memory marked as purgeable, the system avoids the expense of paging by simply reclaiming that memory. The tradeoff, of course, is that the data is discarded, and if your application needs that data later, it must recompute it.

The `NSPurgeableData` class does not have to be used in conjunction with `NSCache`; you can use it independently to get purging behavior.

For more information, see *Caching and Purgeable Memory*.

Concurrency with Operation Objects

A Cocoa operation is an Objective-C–based object designed to help you improve the level of concurrency in your application. You use it to encapsulate a task that you want executed asynchronously. You can submit operation objects to a queue and let the corresponding work be performed asynchronously on one or more separate threads.

The Foundation framework in OS X version 10.6 includes a new class, `NSBlockOperation`, for executing one or more blocks concurrently. Because it can execute more than one block, a block operation object operates using a group semantic; only when all of the associated blocks have finished executing is the operation itself considered finished.

For more information about concurrency via operation objects, read *Concurrency Programming Guide*.

File-System Efficiency

OS X v10.6 maximizes file-system efficiency by using URLs to consolidate access to files, file properties, and directory contents. This allows a single mechanism to be used for accessing all files, file properties, and directories and eliminates internal translations between paths, URLs, and `FSRefs`.

To maximize file efficiency in your application, you should use `file:` scheme URLs as file accessors, instead of using `FSRefs`, `FSSpecs`, File Manager routines, or POSIX file routines (`stat`, `lstat`, `getattrlist`, `chmod`, `setattrlist`, `chflag`) to access file properties or directory contents. In this way, your file-system calls can be processed with a minimum of format translations.

Methods that accept URLs instead of paths have been added to the `NSFileManager` and `NSFileHandler` classes:

```
copyItemAtURL:toURL:error:
linkItemAtURL:toURL:error:
```

```
moveItemAtURL:toURL:error:
removeItemAtURL:error:
fileHandleForReadingFromURL:error:
fileHandleForUpdatingURL:error:
fileHandleForWritingToURL:
```

These methods work just as the corresponding methods that accept file system paths.

Similarly, delegate methods that accept URLs have also been defined for `NSFileManager`:

```
fileManager:shouldCopyItemAtURL:toURL:
fileManager:shouldLinkItemAtURL:toURL:
fileManager:shouldMoveItemAtURL:toURL:
fileManager:shouldProceedAfterError:copyingItemAtURL:toURL:
fileManager:shouldProceedAfterError:linkingItemAtURL:toURL:
fileManager:shouldProceedAfterError:movingItemAtURL:toURL:
fileManager:shouldProceedAfterError:removingItemAtURL:
fileManager:shouldRemoveItemAtURL:
```

File properties have been added to `CFURL` and `NSURL` to provide the same flexibility and services that formerly required use of `FSRef`. Your application can identify properties using the familiar mechanism of keys and values, where file properties are identified by keys (string constants) that are paired with values (`CFType`s for `CFURL` and objects for `NSURL`).

For example, to obtain the modification date of a file using `NSURL`, you could ask for the property associated with the key `NSURLContentModificationDateKey`. For the file size, there is `NSURLFileSizeKey`. For the icon normally associated with this type of file, there is `NSURLEffectiveIconKey`. To obtain the file's parent directory, there is `NSURLParentDirectoryURLKey`. For the available file space on the parent volume, use `NSURLVolumeAvailableCapacityKey`.

There are over 40 file, directory, and volume properties that can be read this way. And because OS X v10.6 has consolidated the file property API, it can use intelligent caching for file properties, often allowing you to obtain multiple properties without additional file I/O.

To facilitate this, new methods have been added to `CFURL.h`:

```
CFURLCopyResourcePropertyForKey
CFURLCopyResourcePropertiesForKeys
```

`CFURLSetResourcePropertyForKey`

`CFURLSetResourcePropertiesForKeys`

And similar methods have been added to `NSURL.h`:

`getResourceValue`

`resourceValuesForKeys`

`setResourceValue`

`setResourceValues`

For the most part, these methods replace the file properties API in the File Manager, `NSFileManager`, and POSIX file access calls (`stat`, `lstat`, `getattrlist`, `chmod`, `setattrlist`, `chflag`).

In addition, you can now create two kinds of `file:` scheme URLs—path-based and file-ID based—so you no longer need to rely on paths when files are moving around or directories have been renamed. File-ID based URLs contain the volume ID and file ID of the target, and can be used to specify a file or directory (or a bundle, which is a directory by another name).

The `CFURLCreateFileIDURL` function converts a path-based URL to a file-ID URL, and `CFURLCreateFileURL` converts a file-ID URL to a path-based URL. Generally speaking, you can use a file-ID URL wherever you can use a `file:` scheme URL.

The following APIs will also operate correctly with file-ID based URLs:

`-[NSURL path]`

`CFURLCopyFileSystemPath`

`CFURLGetFileSystemRepresentation`

There is also a new Bookmark API which creates a data object (`NSData`) or `CFDataRef` (`CFURL`) containing a representation of a URL (a `CFURLRef` or an `NSURL`). This provides the services formerly supplied by aliases, and replaces the Alias Manager. Bookmark data can be stored persistently—in your application's files or database, for example—and later resolved to create a URL locating the bookmarked file, even if that file has moved. The API also supports fetching resource properties directly from bookmark data.

These methods have been added to `CFURL.h`:

`CFURLCreateBookmarkData`

`CFURLCreateByResolvingBookmarkData`

`CFURLCreateResourcePropertyForKeyFromBookmarkData`

`CFURLCreateResourcePropertiesForKeysFromBookmarkData`

And these methods have been added to `NSURL.h`:

`–bookmarkDataWithOptions:includingResourceValuesForKeys:relativeToURL:error:`

`–initByResolvingBookmarkData:bookmarkData:options:relativeToURL:bookmarkDataIsStale:error`

`+URLByResolvingBookmarkData:options:relativeToURL:bookmarkDataIsStale:error:`

`+resourceValuesForKeys:fromBookmarkData:`

See *CFURL Reference* and *NSURL Class Reference* for additional details.

Image Support

The imaging APIs in OS X v10.6 have been simplified to improve your ability to handle images in your application. Image classes are more consistent, and the subsystems in OS X v10.6 provide more efficient bridging between the classes. These improvements bring you several benefits:

- Less uncertainty in choosing among image types
- Less conversion code for you to write and maintain
- Improved performance in decoding and rendering large images with additional level-of-detail support in lower-level image abstractions
- Reduced memory usage through more efficient caching of image data

OS X v10.6 also offers efficient partial decoding of image files, providing support for your application to access subrectangles of images for tiling.

Clients of `NSImage`, `CIImage` and `CGImageRef` generally don't need to make code changes to benefit from the improved imaging pipeline. If you're using Icon Services to get icons, you should switch to `NSWorkspace`.

QTKit Framework

If your application needs to play back iPod media, including content purchased through iTunes, you can take advantage of a new, lightweight, and more efficient media playback capability provided in QuickTime X and OS X v10.6. You gain access to these media services through the QTKit framework.

The new QuickTime X media services offered in OS X v10.6 allow applications to perform asynchronous movie opening, thus avoiding operations from getting blocked for lengthy periods of time when a media file is opened. Note, however, that asynchronous opening is not available for all media types that QTKit supports. Asynchronous loading can also be cancelled at any time.

In OS X v10.6, maximum backward compatibility with existing QTKit client applications is also preserved. This is accomplished by minimizing the number of changes made to the existing QTKit API and by defaulting to completely backward-compatible behaviors. For example, clients of QTKit must explicitly grant QTKit permission to try to use the new private media services for a particular media file, by including a new attribute in the list of attributes passed to `-[QTMovie initWithAttributes:error:]`.

In OS X v10.6, QTKit support for the new media services provided in QuickTime X is primarily limited to media playback. You can:

- Open a media file
- Gather information about the playback characteristics of the movie, such as its duration, the codecs used, and thumbnail images
- Display the movie in a view
- Control the loading and playback of that movie

In particular, movies handled using the new media services in QuickTime X will not be editable or exportable. If you attempt to edit a `QTMovie` object that was opened as a playback-only movie, an exception is thrown. (Note that this is the current behavior for applications that attempt to edit QTMovie objects that are marked as uneditable.)

Among the enhancements provided by the media services in QuickTime X, two new and important movie attributes are defined:

```
NSString * const QTMovieOpenAsyncRequiredAttribute
NSString * const QTMovieOpenForPlaybackAttribute
```

The first attribute indicates whether a `QTMovie` object must be opened asynchronously. Set this attribute to `YES` to indicate that all operations necessary to open the movie file (or other container) and create a valid `QTMovie` object must occur asynchronously. That is, the methods `movieWithAttributes:error:` and `initWithAttributes:error:` will return almost immediately, performing any lengthy operations on another thread. Your application can monitor the movie load state to determine the progress of those operations.

The second attribute indicates whether a `QTMovie` object will be used only for playback and not for editing or exporting. Set this attribute to `YES` to indicate that you intend to use movie playback methods, such as `play` or `stop`, or corresponding movie view methods such as `play:` or `pause:` in order to control the movie, but do *not* intend to use other methods that edit, export, or in any way modify the movie. Specifying that you need playback services only may allow `QTMovie` to use more efficient code paths for some media files.

For a simple and more efficient movie playback application, you can open a movie file and attach it to a `QTMovieView` object using the following snippet of code:

```
- (void)windowControllerDidLoadNib:(NSWindowController *) aController
   {
       [super windowControllerDidLoadNib:aController];
       if ([self fileName]) {
          NSDictionary *attributes = [NSDictionary dictionaryWithObjectsAndKeys:
                                      [self fileName], QTMovieFileNameAttribute,
                                       [NSNumber numberWithBool:YES],
                                       QTMovieOpenForPlaybackAttribute, nil];
           movie = [[QTMovie alloc] initWithAttributes:attributes error:NULL];
           [movieView setMovie:movie];
           [movie release];
           [[movieView movie] play];
       }
   }
```

Because the attributes dictionary contains a key-value pair with the `QTMovieOpenForPlaybackAttribute` key and the value `YES`, QTKit uses the new media services provided in QuickTime X, if possible, to play back the media content in the selected file.

OpenCL

OS X v10.6 enables your application to leverage not only multiple CPUs and multiple cores, but the high-performance parallel processing power of GPUs built into many systems. For many applications, this can result in a significant speed-up.

Applications can make use of this new capability by calling the OpenCL (Open Computing Library) framework, an Apple-proposed open standard for parallel data computation across GPUs and CPUs. OpenCL provides an abstraction layer so that you can write your code once and it can then run on any hardware that supports the

OpenCL standard. Before OpenCL it was possible to write general-purpose applications that ran on GPUs, but you had to translate your calculations into vector arithmetic and write your code in a different proprietary language for each compute device.

The OpenCL language is optimized for execution on graphics processors without being tied to any particular hardware or architecture. It is a general-purpose computer language, not specifically a graphics language. It results in the largest performance gains when used for data-parallel processing of large data sets. There are many applications that are ideal for acceleration using OpenCL, such as signal processing, image manipulation, or finite element modeling. The OpenCL language has a rich vocabulary of vector and scalar operators and the ability to operate on multidimensional arrays in parallel.

OpenCL programming involves writing compute kernels in the OpenCL-C language, and calling OpenCL framework APIs to set up the context in which the kernels run and to enqueue the kernels for execution. Before an OpenCL compute kernel can run on a particular compute device, the kernel must be compiled into binary code specific to that device. Therefore, to make OpenCL programs portable, the OpenCL compiler and runtime are included in the OpenCL framework and the kernels can be compiled dynamically at runtime. After the program is installed on a system, application load time can be minimized by compiling the program for that system and then saving and running the compiled version in subsequent invocations of the application. OpenCL routines can be linked to and called as C functions from Cocoa, C, or C++ applications.

Multiple instances of a compute kernel can be run in parallel on one or more compute units, such as GPU or CPU cores. OpenCL allows you to query the current device to determine how many instances of a kernel can be executed in parallel, for optimization on the fly. OpenCL syntax makes it easy to describe problems in terms of multi-dimensional arrays, often the most natural way to design a program. Multiple kernels can be linked together into larger OpenCL programs.

To obtain an overview, a description of the process of writing an OpenCL program, and code samples, see *OpenCL Programming Guide for Mac* .

64-Bit Plug-Ins Required

In OS X v10.6, a plug-in must match its host application in terms of both processor architecture and address width.

Applications and other executables that ship as part of OS X are being transitioned to 64-bit-capable executables. If you are writing plug-ins for these 64-bit-capable applications (screen savers, for example), your code must be made 64-bit-native in OS X v10.6 or it will not work on 64-bit-capable Macs.

Some examples of affected plug-ins include:

- Printer dialog extensions
- Screen savers

- Audio units
- Spotlight importers
- Dashboard plug-ins
- Safari plug-ins

If you are writing plug-ins that are loaded by Apple-authored applications or daemons, you should immediately start transitioning your plug-ins to add an `x86_64` version to the universal binaries. Plug-ins that are only two-way (32-bit) universal will not be loaded by 64-bit capable applications in OS X v10.6 and later. One current exception is the System Preferences application, which automatically relaunches itself in 32-bit mode if a user selects a legacy 32-bit preference pane. For an optimal user experience, however, you should still update your preference panes to contain an `x86_64` version as soon as possible.

Important: If the host application is built to run garbage-collected, the corresponding build of the plug-in must have garbage collection enabled. In the GCC 4.2 Code Generation section of the project settings in Xcode, use the Objective-C Garbage Collection pop-up menu to select "Supported [-fobjc-gc]".

For binary compatibility reasons, Automator, screen savers, and System Preferences are currently built to run garbage-collected in 64-bit form but not in 32-bit form. Consequently, any plug-ins for these applications must be built to run garbage-collected in 64-bit form but not in 32-bit form.

To learn how to transition your code to 64-bit, read *64-Bit Transition Guide* and then read either *64-Bit Transition Guide for Cocoa* or *64-Bit Guide for Carbon Developers*.

Core Text

The Core Text framework includes a new Font Manager API for registering and activating fonts, managing font descriptors, manipulating font names, and validating font files. For more information, see *Core Text Reference Collection*.

Formal Protocol Adoption

To provide better compile-time type checking, both the AppKit and the Foundation frameworks now use formal protocols for delegate methods. Required protocol methods are marked with `@required`, where possible. The rest are marked with `@optional`.

Gamma 2.2

Digital images can be displayed on a wide variety of devices, including flat-panel displays, CRTs, and printers. These devices transmit and reflect light with different brightness and intensity, in a nonlinear manner. Including gamma information in the color profile helps the system to reproduce images on different devices with the correct luminance. When gamma information is not available, the system must either guess at the correct gamma correction for the image or use the system default.

Historically, the default gamma correction for Mac OS has been a value of 1.8 (a useful value for print professionals). In recent years, television, video, and web standards have all settled on a default gamma of 2.2. In OS X v10.6, the Mac moves to this common standard, with the following ramifications:

- Images without embedded gamma information look the same when created on Mac OS and displayed on other systems, and vice versa.
- Images tagged with gamma information look the same as in previous versions of Mac OS (with minor improvements in some cases, because images created in a 2.2 space are no longer converted).
- OpenGL images and untagged web images have higher contrast. In general, web content displayed will more closely match the luminance seen on other systems, such as PCs running Microsoft Windows.
- Applications that use system UI elements such as Cocoa controls will see little or no change. However, untagged images that are not part of the system UI appear darker. If your application uses custom UI elements, you may need to adjust their brightness to obtain the desired appearance.

It is strongly recommended that your application tag its images with gamma information. A new API has been added to ColorSync to return the gamma value for a given connected display. Programmatically generated images should use the ColorSync framework to automatically output the correct luminance.

Application Level

This section introduces changes to the application-level layers in OS X v10.6.

Exchange Support

OS X v10.6 includes technology to provide Microsoft Exchange support for Address Book, Mail, and iCal. It uses the Exchange Web Services protocol to provide access to Exchange Server 2007.

JavaScript Performance

Safari in OS X v10.6 includes a number of enhancements to improve JavaScript performance:

- A new simplified DOM query API (W3C Selectors API):

`querySelector`

`querySelectorAll`

- New native replacements for commonly used JavaScript library functions:
  `getElementsByClassName`

- A new JavaScript interpreter in WebKit and Safari. Instead of executing scripts using a traditional interpreter, JavaScript scripts are now compiled down to bytecode, then executed using a bytecode execution engine. This results in a significant performance improvement over interpreting scripts on the fly. As a result, Safari's JavaScript performance in OS X v10.6 is four times faster than it is in Leopard.

In addition to these underlying changes, which are described next, there are additional tools for optimizing your JavaScript code.

DOM Query Selector API

The DOM Query Selector API adds two additional document methods, `querySelector` and `querySelectorAll`. Use these methods to obtain a list of elements matching a particular pattern. They work much like the XPath API, but with a syntax that is more lightweight and easier to learn. (The DOM query selector syntax uses the CSS selector syntax that you should already be familiar with as a web designer.)

For example, if you want to get every element of the class `stripedtable`, you could use the following code:

```
var stripetables = document.querySelectorAll(".stripedtable");
```

To obtain the odd-and even-numbered child elements of these tables, you could use code like this:

```
var darkstripes = document.querySelectorAll(".stripedtable tbody:nth-child(even)");
var lightstripes = document.querySelectorAll(".stripedtable tbody:nth-child(odd)");
```

The `querySelector` works similarly, but returns only the first matching element.

In addition to performing these queries on the document, you can also perform them on an individual element. In the previous example, if you want to work with each of the tables individually, you might write code like this:

```
// Obtain the initial list of matching tables
var stripetables = document.querySelectorAll(".stripedtable");
```

```
for (var mytable in stripetables) {
    // Perform further selection on each of the resulting elements.
   var darkstripes = mytable.querySelectorAll(".stripedtable tbody:nth-child(even));
   var lightstripes = mytable.querySelectorAll(".stripedtable tbody:nth-child(odd));
    ...
}
```

For a complete description of this API, see the W3C Selectors API at http://www.w3.org/TR/selectors-api/.

Other Native DOM API Additions

In addition to the DOM Selector Query API, Safari and WebKit provide a new native implementation of the `getElementsByClassName` document method. This function is similar to `getElementsByName` except that it searches the `class` attribute for class names that match. For example:

```
var buttonBarElements = document.getElementsByClassName('fancytoolbar_button');
```

This call returns an array containing all elements that match the specified class. For example, above call returns all of the following elements:

```
<a class="fancytoolbar_button">...</a>
<img class="link_button fancytoolbar_button" />
<p class="fancytoolbar_button new_paragraph_button">...</p>
```

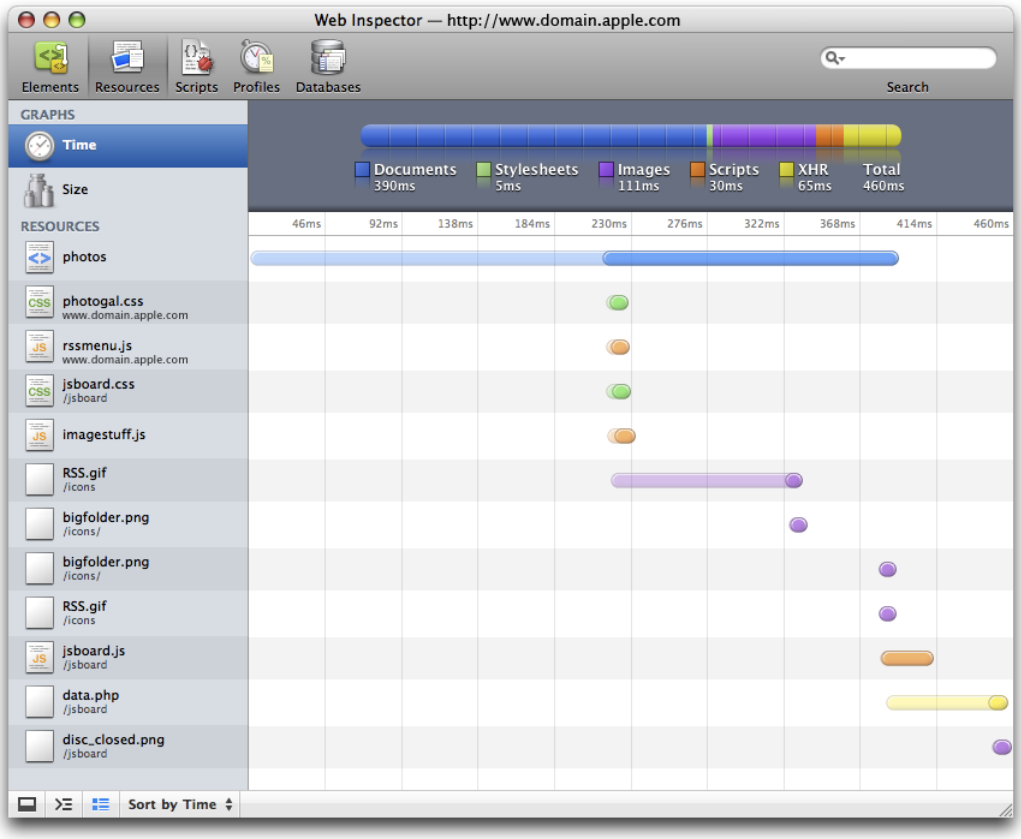
Web Inspector Improvements

Safari in OS X v10.6 provides a number of improvements to the web inspector that help you tune your website for better performance. To use them, choose Show Web Inspector from the Develop menu. (You must first enable the Develop menu in the Advanced pane in Safari Preferences if you have not already done so.) Within the web inspector, you should now see several significant enhancements over previous versions of Safari:

- A graphical resources pane
- A built-in JavaScript debugger
- A performance profiler

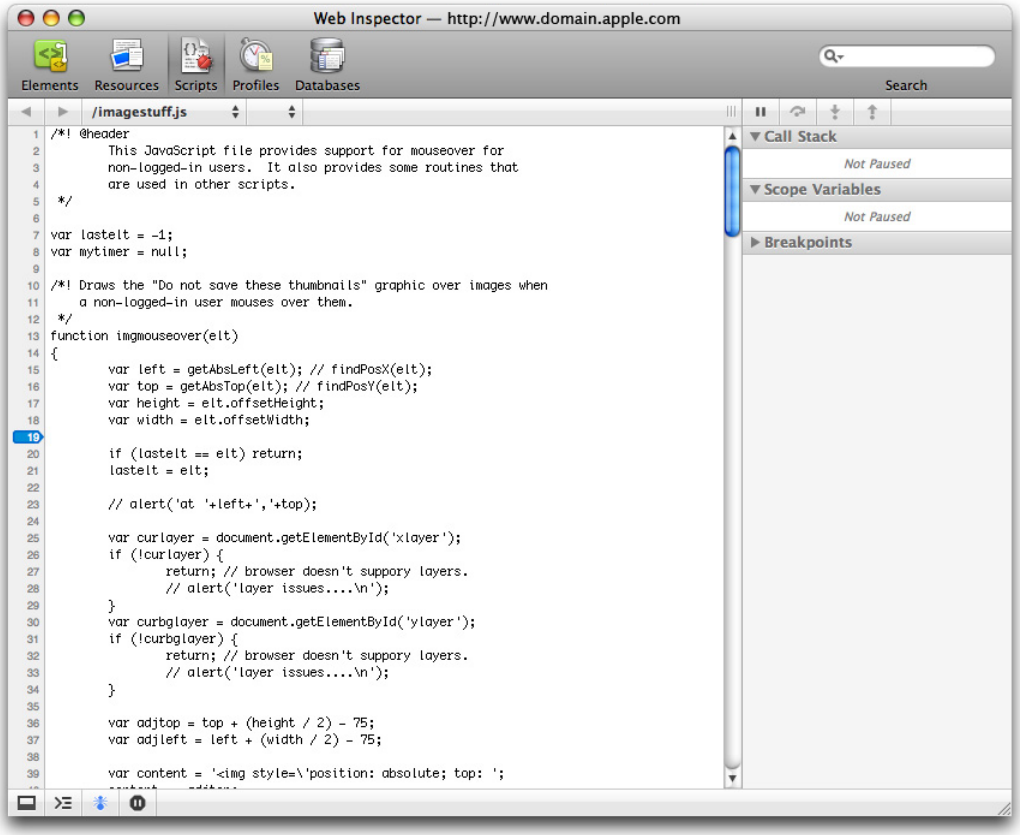
The newly revamped Resources pane now shows detailed graphs of the size of each resource and the amount of time spent loading that resource. By looking at this pane, you can see at a glance what aspects of your web content dominate the load time of the page.

Figure 1 Web inspector Resources pane



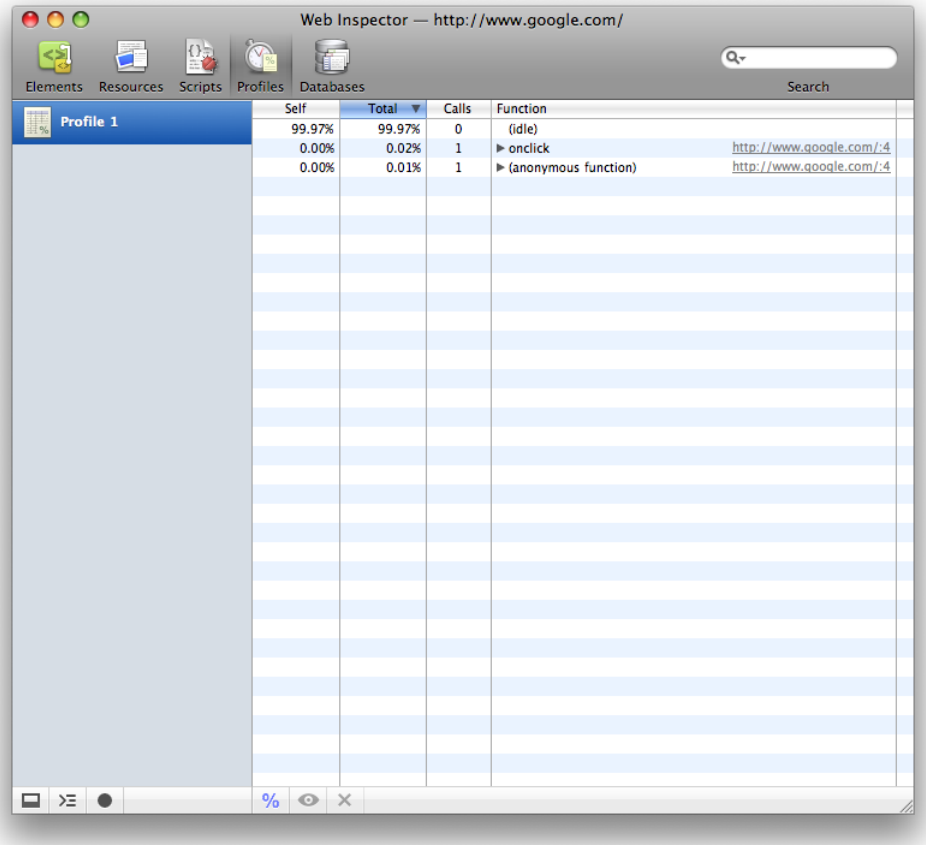
The Scripts pane now provides substantial new debugging functionality. You can pause script execution or set breakpoints to stop automatically, and then step into function calls, step over them, or step out of them. While execution is stopped, you can also examine the values of local and global variables.

Figure 2 Web inspector Scripts pane



Finally, the Profiles pane allows you to test performance on your code. This built-in performance profiling helps you determine which areas of your JavaScript code you should optimize to provide the biggest performance improvements. To use profiling, click the dot icon in the lower-left corner, perform some task on the website, and then stop profiling by clicking the dot again.

Figure 3 Web inspector Profiles pane



After you finish profiling, you can view the profiles by clicking them in the sidebar. Safari then displays the amount of time spent in each function—both in the body of the function itself and in other functions that it calls—and allows you to sort by these values. In this way, you can quickly see which pieces of JavaScript code take the most time and thus are likely to be good targets for optimization.

API Delta Documents

Table 1 lists the documents describing the API changes that were made in system frameworks for OS X v10.6.

Table 1 OS X v10.6 delta documents

Framework	Document
Accelerate	*Accelerate Reference Update*
Address Book	*Address Book Reference Update*
AGL	*AGL Reference Update*
AppKit	*Application Kit Reference Update*
Application Services	*Application Services Reference Update*
Audio Toolbox	*Audio Toolbox Reference Update*
Audio Unit	*Audio Unit Reference Update*
Automator	*Automator Reference Update*
Calendar Store	*Calendar Store Reference Update*
Carbon	*Carbon Reference Update*
Core Audio	*Core Audio Reference Update*
Core Audio Kit	*Core Audio Kit Reference Update*
Core Data	*Core Data Reference Update*
Core Foundation	*Core Foundation Reference Update*
Core Location	*Core Location Reference Update*
Core Services	*Core Services Reference Update*
Foundation	*Foundation Reference Update*
Image Capture	*Image Capture Core Reference Update*
Input Method Kit	*Input Method Kit Reference Update*
Instant Message	*Instant Message Reference Update*
OpenAL	*OpenAL Reference Update*

Framework	Document
OpenCL	*OpenCL Reference Update*
Open Directory	*Open Directory Reference Update*
OpenGL	*OpenGL Reference Update*
QTKit	*QTKit Reference Update*
Quartz	*Quartz Reference Update*
Quartz Core	*Quartz Core Reference Update*
Quick Look	*Quick Look Reference Update*
QuickTime	*QuickTime Reference Update*
Security	*Security Reference Update*
Security Foundation	*Security Foundation Reference Update*
Sync Services	*Sync Services Reference Update*
System Configuration	*System Configuration Reference Update*
WebKit	*WebKit Reference Update*

Document Revision History

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

Date	Notes
2014-10-16	Updated for OS X v10.10.
2013-10-22	Updated for OS X v10.9.
2012-07-23	Added information about features introduced in OS X v10.8.
2012-01-09	Updated to describe additional features in OS X Lion. Updated the links in the OS X Lion v10.7 (page 45) chapter.
2011-07-01	Added information for OS X v10.7.
2009-11-13	Updated to describe new and changed technologies introduced in OS X v10.6.
2007-12-11	Added a description for the Core Text technology and updated the Core Audio description.
2007-10-31	Updated to include new information for OS X v10.5.
2006-03-28	Added reference to ACL man page.
2005-06-04	Added information about GCC 4.0 along with links to a porting document.
2005-04-29	New document that describes features introduced in OS X.

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