What's New In Mac OS X

Mac OS X

2007-12-11

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

Simultaneously published in the United States and Canada.

Contents

Introduction to What's New in Mac OS X 7

Organization of This Document 7

Mac OS X v10.4 9

Features 9
- 64-Bit Support 9
- Accessibility 9
- ACL Support 9
- Automator 10
- Core Audio 10
- Core Data 10
- Core Image 11
- Core Video 12
- Dashboard 12
- iChat Presence 12
- Installer Enhancements 12
- KEXT API Changes 13
- Network Diagnostics 13
- OpenAL 13
- PDF Kit 14
- Quartz 2D 14
- Quartz Composer 14
- QuickTime 7 14
- QuickTime Kit 15
- Search Kit 15
- Spotlight 16
- SQLite 16
- Sync Services 16
- Web Kit Editing and DOM Support 17
- Xcode 2.0 17
- GCC 4.0 17
- XML Parsing 18

API Delta Documents 18

Mac OS X v10.5 21

Cross-Functional Features 21
- 64-Bit Support 21
- Garbage Collection 22
- Objective-C 2.0 22

Resolution-Independent User Interface 22
Security Enhancements 23
New and Updated Features 24
Address Book Framework 24
AppleScript 24
Automator 25
Calendar Store Framework 25
Carbon 25
Cocoa 26
Core Animation 28
Core Audio 29
Core Data 2.0 30
Core Image 30
Core Text 31
Darwin 31
FSEvents API 32
iChat Instant Message Framework 32
Identity Services 33
Image Kit 33
The Input Method Kit 34
Latent Semantic Mapping Framework 34
Mail Stationery 34
OpenGL 34
Quartz 35
Quartz Composer 35
QuickTime 35
QuickTime Kit 36
RSS/Atom Support 37
Scripting 37
Spotlight 37
Sync Services 38
Time Machine 38
Developer Tools 39
Xcode 3.0 39
Interface Builder 3.0 43
Instruments 44
Dashcode 45
API Delta Documents 46

Document Revision History 49

Figures and Tables

Mac OS X v10.4 9

Table 1 Mac OS X v10.4 delta documents 18

Mac OS X v10.5 21

Figure 1 Xcode application 39
Figure 2 Xcode documentation window 41
Figure 3 Interface Builder 3.0 44
Figure 4 The Instruments application interface 45
Figure 5 Dashcode canvas 46
Table 1 Mac OS X v10.5 delta documents 46

Introduction to What's New in Mac OS X

This document provides developer-level information about features that were introduced in different versions of Mac OS X.

This document is not intended as a complete list of features for each new version of Mac OS X. Instead, it focuses on those features that affect the development of third-party software, providing overviews of each feature along with insight as to 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.

For each release, this document also provides links to "delta" reference documents, which list the new programming interfaces that were introduced with that release.

Organization of This Document

This document includes the following article:

- "Mac OS X v10.4" (page 9) describes the features that were introduced in Mac OS X v10.4.
- "Mac OS X v10.5" (page 21) describes the features that were introduced in Mac OS X v10.5.

Mac OS X v10.4

This article summarizes key features that are available beginning with Mac OS X version 10.4 (also known as "Tiger"). It also lists many of the documents that were created or updated for the release.

Features

Mac OS X v10.4 supports the following new features.

64-Bit Support

The Xcode Tools now support the compilation, linking, and debugging of 64-bit binaries using C or C++. In addition to the tools support, the system also includes 64-bit versions of `libSystem.dylib`, which contains much of the C standard library code, and the Accelerate framework.

Support for 64-bit computing makes it possible to operate on large data sets more efficiently. For more information about creating 64-bit applications, see *64-Bit Transition Guide*.

Accessibility

Mac OS X improves its support for accessibility by including VoiceOver—a full-featured spoken interface to the Macintosh. Built in to the Aqua user interface, VoiceOver uses the accessibility architecture to make it possible for users with visual disabilities to navigate and use the system. VoiceOver complements the existing Universal Access features and promotes collaboration between disabled and nondisabled users on the same computer.

For more information on accessibility technologies in Mac OS X, see *Accessibility Overview*.

ACL Support

Mac OS X v10.4 introduces support for Access Control Lists (ACLs)—a robust system for implementing file-based permissions. ACLs supplement the existing BSD permissions currently used by the Mac OS X file systems. They also offer many improvements over BSD permissions for applications that support them, including the following:

- Support for ownership of files and directories by a group
- Support for multiple owners of a file or directory, each with potentially different permissions
- Enhanced interoperability with Samba and Windows
- More control over a file than just read/write/execute permissions

- Support for static inheritance of file permissions from a parent directory

See the `acl` man page for information and the list of functions you can use to manage ACLs. See also the `unistd.h` and `acl.h` header files.

Automator

Automator lets you automate common workflows on your computer without writing any code. The workflows you create can take advantage of many features of Mac OS X and any standard applications for which predefined **actions** are available. Actions are building blocks that represent tangible tasks, such as opening a file, saving a file, applying a filter, and so on. The output from one action becomes the input to another and you assemble the actions graphically with the Automator application.

In cases where actions are not available for the tasks you want, you can run AppleScript scripts directly with the "Run AppleScript" action or create new actions yourself. Automator supports the creation of actions using Objective-C code or AppleScript commands.

For more information about using Automator, see Automator Help. For information on how to write Automator actions, see *Automator Programming Guide*.

Core Audio

Core Audio includes new tools and functions to improve the audio handling experience of applications. New features include:

- New Audio Units
  - A file-playback audio unit lets you use an existing sound file as an input source.
  - A time and pitch transformation audio unit lets you modify both the pitch and time of audio data.
- AU Lab - a tool that lets you graphically host audio units and examine the results.
- Aggregate device support
- Extended audio file API - a new API for converting files from one format to another
- A new extension mechanism for supporting audio file formats not supported natively by Core Audio
- A clock API for creating and tracking MIDI time formats

For detailed information about the new function calls, see the header files of the assorted Core Audio frameworks.

Core Data

The Core Data framework (`CoreData.framework`) is a new technology for managing the model data of a Model-View-Controller application. Core Data is intended for use in Cocoa applications where the data model is already highly structured. Instead of defining data structures programmatically, you use the graphical tools in Xcode to build a schema representing your data model. At runtime, instances of your data-model entities are created, managed, and made available through the Core Data framework.

By managing your application's data model for you, Core Data significantly reduces the amount of code you have to write for your application. Core Data also provides the following features:

- Storage of object data in mediums ranging from an XML file to a SQLite database
- Management of undo/redo beyond basic text editing
- Support for validation of property values
- Support for propagating changes and ensuring that the relationships between objects remain consistent
- Grouping, filtering, and organizing data in memory and transferring those changes to the user interface through Cocoa bindings

If you are starting to develop a new application, or are planning a significant update to an existing application, you should consider using Core Data. For more information about Core Data, including how to use it in your applications, see *Core Data Programming Guide*.

Core Image

Core Image is an image processing technology that leverages programmable graphics hardware whenever possible. The Core Image application programming interface (API) provides access to built-in image filters for both video and still images and provides support for custom filters and near real-time processing.

Core Image is an extensible architecture for near real-time, pixel-accurate image processing of graphics as well as video. You can perform the following types of operations by using filters that are bundled in Core Image or that you or another developer creates:

- Crop images
- Correct color, including perform white-point adjustments
- Apply color effects, such as sepia tone
- Blur or sharpen images
- Composite images
- Warp the geometry of an image by applying an affine transform or a displacement effect
- Generate color, checkerboard patterns, Gaussian gradients, and other pattern images
- Add transition effects to images or video
- Provide real-time control, such as color adjustment and support for sports, vivid, and other video modes
- Apply linear lighting effects, such as spotlight effects

Core Image is part of the Quartz Core framework (`QuartzCore.framework`). For information about how to use Core Image or how to write custom image units, see *Core Image Programming Guide* and *Core Image Reference Collection*.

Core Video

Core Video provides a modern foundation for delivering video in your applications. It creates a bridge between QuickTime and the GPU to deliver hardware-accelerated video processing. By offloading complex processing to the GPU, you can significantly increase performance and reduce the CPU load of your applications. Core Video also allows developers to apply all the benefits of Core Image to video, including filters and effects, per-pixel accuracy, and hardware scalability.

Core Video is part of the Quartz Core framework (`QuartzCore.framework`). For information on how to use it, see *Core Video Programming Guide* and *Core Video Reference*.

Dashboard

Dashboard provides a lightweight desktop layer for running **widgets**. Widgets are lightweight web applications that display information a user might use occasionally. You could write widgets to track stock quotes, view the current time, or access key features of a frequently used application. Widgets reside in the Dashboard layer, which is activated by the user and comes into the foreground in a manner similar to Exposé. Mac OS X comes with several standard widgets, including a calculator, clock, and iTunes controller.

Creating widgets is simpler than creating most applications because widgets are effectively HTML-based applications with optional JavaScript code to provide dynamic behavior. Dashboard uses Web Kit to render HTML and run JavaScript code. Your widgets can take advantage of several extensions provided by that environment, including a way to render content using Quartz-like JavaScript functions.

For information on how to create widgets, see *Dashboard Tutorial*.

iChat Presence

The Instant Message framework (`InstantMessage.framework`) supports the detection and display of a user's online presence in applications other than iChat. Using this framework, you can find out the current status of a user connected to an instant messaging service. You can then obtain the user's custom icon, status message, or a URL to an image that indicates the user's status. You can then display this information along with other user information in your program. For example, Mail uses the framework to determine if an email is from a user who is currently online; it the person is available, it then displays an appropriate icon next to that person's name.

For more information, see *Instant Message Framework Reference*.

Installer Enhancements

The Installer application supports several new features in Mac OS X v10.4. These features help you deliver a more robust installation experience for your users while ensuring that your software is installed correctly. Among the features added to Installer are the following:

- Distribution scripts offer a flexible and convenient way to author installation packages. By consolidating information that was formerly scattered across many files, a distribution script provides better control for installing packages and metapackages.

- Multiple CD/DVD support. Using distribution scripts, you can support installation from multiple CDs or DVDs and prompt the user for new discs as they are needed.
- Installer plug-ins let you display custom panes in the Installer application.
- File version checking. This feature extends the current "Find File" mechanism to support checks for individual files. This mechanism can ensure that you only replace outdated files on the user's machine, leaving newer files alone.

For more information on creating software installation packages, see *Software Delivery Guide* and *Installer Release Notes*.

KEXT API Changes

The design of the kernel data structures has changed to a more opaque access model. This change makes it possible for kernel developers to write nonfragile kernel extensions—that is, kernel extensions that do not break when the kernel data structures change. Developers are highly encouraged to use the new API for accessing kernel data structures.

For information about writing kernel extensions, see *Kernel Programming Guide*.

Network Diagnostics

Network diagnostics is a way of helping the user solve network problems. Although modern networks are generally reliable, there are still times when network services may fail. Sometimes the cause of the failure is beyond the ability of the desktop user to fix, but sometimes the problem is in the way the user's computer is configured. The network diagnostics feature provides a diagnostic application to help the user locate problems and correct them.

For more information, see the header files of CFNetwork.

OpenAL

The system now comes standard with the Open Audio Library (OpenAL) audio system installed. This interface is a cross-platform standard for delivering 3D audio in applications. OpenAL lets you implement high-performance positional audio in games and other programs that require high-quality audio output. Because it is a cross-platform standard, the applications you write using OpenAL on Mac OS X can be ported to run on many other platforms.

Apple's implementation of OpenAL is based on Core Audio, so it delivers high-quality sound and performance on Mac OS X systems. To use OpenAL in a Mac OS X application, include the OpenAL framework (`OpenAL.framework`) in your Xcode project. This framework includes header files whose contents conform to the OpenAL specification, which is described at http://www.openal.org.

PDF Kit

PDF Kit is a Cocoa framework for managing and displaying PDF content directly from your application's windows and dialogs. Using the classes of PDF Kit, you can embed a PDFView in your window and give it a PDF file to display. The PDFView class handles the rendering of the PDF content, handles copy-and-paste operations, and provides controls for navigating and setting the zoom level. Other classes let you get the number of pages in a PDF file, find text, manage selections, add annotations, and specify the behavior of some graphical elements, among other actions. Users can also copy selected text in a PDFView to the pasteboard.

If you need to display PDF data directly from your application, PDF Kit is highly recommended. It hides many of the intricacies of the Adobe PDF specification and provides standard PDF viewing controls automatically. PDF Kit is part of the Quartz framework (`Quartz.framework`). For more information, see *PDF Kit Reference Collection*.

Quartz 2D

Quartz 2D includes the following improvements:

- Support for the reading and writing of image data (including data that contains multiple images and thumbnail images) using the ImageIO framework
- Layered drawing using CGLayer
- More PDF support
- Additional support for blend modes

For information about new Quartz features, see *Quartz 2D Programming Guide*.

Quartz Composer

Quartz Composer is a development tool for processing and rendering graphical data. Quartz Composer provides a visual development environment built on technologies such as Quartz 2D, Core Image, OpenGL, and QuickTime. You can use Quartz Composer as an exploratory tool to learn the tasks common to each visual technology and then use its application programming interface (API) to programmatically play and control your compositions. In addition to supporting visual technologies, Quartz Composer also supports nongraphical technologies such as MIDI System Services and Rich Site Summary (RSS) file content.

For information on how to use Quartz composer, see *Quartz Composer Programming Guide* and *Quartz Composer Reference Collection*.

QuickTime 7

Mac OS X v10.4 includes QuickTime version 7. This new version includes support for many new features, including the following:

- High-resolution audio
- H.264 support

- Frame reordering video compression
- Support for rendering to OpenGL and the elimination of dependence on graphics worlds (GWorlds)
- A new metadata format
- QuickTime sample table API
- Changes to QuickTime Player and Pro UI

For detailed information list of changes to QuickTime, see *QuickTime 7 Update Guide*.

QuickTime Kit

QuickTime Kit (`QTKit.framework`) is an Objective-C framework for manipulating QuickTime-based media. This framework lets you incorporate movie playback, movie editing, export to standard media formats, and other QuickTime behaviors easily into your applications. The classes in this framework open up a tremendous amount of QuickTime behavior to both Carbon and Cocoa developers. Instead of learning how to use the more than 2500 functions in QuickTime, you can now use a handful of classes to implement the features you need.

Note: The QuickTime Kit framework supersedes the NSMovie and NSMovieView classes available in Cocoa. If your code uses these older classes, you should consider changing your code to use QuickTime Kit.

For an introduction and tutorial on how to use the QuickTime Kit, see *QuickTime Kit Programming Guide*. For reference information about the QuickTime Kit classes, see *QTKit Framework Reference*.

Search Kit

Search Kit in Mac OS X v10.4 contains several improvements over the version introduced in Mac OS X v10.3. Search performance is now significantly faster than before, with indexing up to three times faster than before. By adopting several new function calls, you can also achieve search-as-you-type performance with incremental results. Search Kit also includes the following capabilities:

- The ability to search for text contained within a word or at the end of a word
- Optional proximity indexing and quoted phrase searching
- The ability to combine search patterns to form complex search expressions
- Improved relevance ranking with absolute relevance to support incremental results
- Greater control over indexing and searching options
- Improved search syntax
- Unranked search support, which improves performance

For more information, see *Search Kit Programming Guide* and *Search Kit Reference*.

Spotlight

Spotlight provides advanced search capabilities for applications. The Spotlight server gathers metadata from documents and other relevant user files and incorporates that metadata into a searchable index. The Finder uses this metadata to provide users with more relevant information about their files. For example, in addition to listing the name of a JPEG file, the Finder can also list its width and height in pixels.

Application developers use Spotlight in two different ways. First, you can search for file metadata using the Spotlight search API. Second, you can generate metadata for files that use your custom file formats. If your application defines a custom file format, you should provide a Spotlight importer plug-in to parse files of that format. In addition to writing an importer, your application should also add metadata information to any new files it creates.

Note: Spotlight is intended primarily for searching meta information associated with files. To search the actual contents of a file, use the Search Kit API. For more information, see "Search Kit" (page 15).

For more information on using Spotlight in your applications, see *Spotlight Overview*.

SQLite

The SQLite library lets you embed a SQL database engine into your applications. Programs that link with the SQLite library can access SQL databases without running a separate RDBMS process. You can create local database files and manage the tables and records in those files. The library is designed for general purpose use but is still optimized to provide fast access to database records.

The SQLite library is located at `/usr/lib/libsqlite.dylib` and the `sqlite.h` header file is in `/usr/include`. A command-line interface (`sqlite`) is also available for communicating with SQLite databases using scripts. For details on how to use this command-line interface, see the man page for `sqlite`.

For more information about using SQLite, go to http://www.sqlite.org.

Sync Services

Sync Services extends data synchronization capabilities to all Mac OS X applications. Third-party applications can use Sync Services to synchronize data with system databases, such as those provided by Address Book and iCal. They can also synchronize custom data with other applications and across multiple computers through the user's .Mac account.

Note: With Sync Services, applications can now directly initiate the synchronization process. The iSync application still exists but is now used to initiate the synchronization process for specific hardware devices.

For more information about using Sync Services in your application, see *Sync Services Programming Guide*. For reference information, see *Sync Services Framework Reference*.

Web Kit Editing and DOM Support

Web Kit now includes support for creating text views containing editable HTML. The editing support is equivalent to the support available in Cocoa for editing RTF-based content. With this support, you can replace text and manipulate the document text and attributes, including CSS properties. Although it offers many features, the Web Kit editing support is not intended to provide a full-featured editing facility like you might find in professional HTML editing applications. Instead, it is aimed at developers who need to display HTML and handle the basic editing of HTML content.

Web Kit also includes support for creating and editing content at the DOM level of an HTML document. You can use this support to navigate DOM nodes and manipulate those nodes and their attributes. You can also use the framework to extract DOM information. For example, you could extract the list of links on a page, modify them, and replace them prior to displaying the document in a web view.

For information on how to use Web Kit from both Carbon and Cocoa applications, see *WebKit Objective-C Programming Guide*. For information on the classes and protocols in the Web Kit framework, see *WebKit Objective-C Framework Reference*.

Xcode 2.0

Xcode 2.0 includes the following key new features:

- Workspaces. Xcode 2.0 introduces different project window configurations, called workspaces, that let you choose the layout that best suits your preferred workflow. Xcode 2.0 currently ships with three workspaces:
  - The default (or traditional) Xcode project window, introduced in Xcode 1.2
  - An all-in-one project window that combines views for project management, building, and debugging in a single window
  - A condensed project window workspace that provides a smaller and more compact project window
- A Favorites bar that you can use to hold files, folders, smart groups, and so on
- Support for downloading updates to the ADC Reference Library
- 64-bit support. See "64-Bit Support" (page 9)
- Visual design. Xcode 2.0 includes a visual designer that lets you create class models for C++, Objective-C, and Java classes and persistence models for use with the Core Data framework.

For more information about features and improvements of Xcode, see *Xcode Release Notes* and *Xcode 2.0 User Guide*.

GCC 4.0

In Mac OS X 10.4, GCC 4.0 is the default compiler. If you are creating new projects on the platform, you should naturally be using GCC 4.0 to compile those projects. However, if you are building existing projects using the GCC 3.3 compiler (the default compiler in Mac OS X 10.3), there are also many reasons to upgrade to GCC 4.0, including the following:

- Better compile times
- Better C++ language conformance
- Smaller C++ binaries
- Faster C++ compiles
- Better optimization machinery
- Better error checking and diagnosis

Before you upgrade though, you should understand the changes that have gone into GCC 4.0 and how they might affect your code. In particular, code that compiled cleanly using GCC 3.3 may now generate warnings and errors when compiled using GCC 4.0.

For information on how to port your code successfully to GCC 4.0, see *GCC Porting Guide* and *GCC 4 Release Notes*.

XML Parsing

The NSXML classes offer advanced support for manipulating XML-based documents in Cocoa. You can use these classes to do the following:

- Create new XML documents
- Search the contents of an XML document arbitrarily
- Convert structured documents using Extensible Stylesheet Language Transformation (XSLT)
- Validate XML content
- Create and manipulate document type definitions (DTDs) as logical tree structures

In addition to creating documents, you can also use NSXML to search documents using the XQuery or XPath query languages. These languages let you create complex search expressions to retrieve single nodes, groups of nodes, or atomic values (strings, integers, dates, and so on) from the XML document.

For more information, see *Tree-Based XML Programming Guide for Cocoa* in Cocoa Data Management Documentation.

API Delta Documents

The documents in Table 1 list the API changes that were made in system frameworks for Mac OS X v10.4.

Table 1 Mac OS X v10.4 delta documents

Framework	Document
Accelerate	*Accelerate Reference Update*
Address Book	*Address Book Reference Update*

Framework	Document
AGL	*AGL Reference Update*
AppKit	*Application Kit Reference Update*
Application Services	*Application Services Reference Update*
Automator	*Automator Reference Update* (new framework)
Core Data	*Core Data Reference Update* (new framework)
Core Foundation	*Core Foundation Reference Update*
Core Services	*Core Services Reference Update*
Foundation	*Foundation Reference Update*
Instant Message	*Instant Message Reference Update* (new framework)
OpenGL	*OpenGL Reference Update*
QTKit	*QTKit Reference Update* (new framework)
Quartz Core	*Quartz Core Reference Update* (new framework)
Quartz	*Quartz Reference Update* (new framework)
QuickTime	*QuickTime Reference Update*
Security	*Security Reference Update*
Sync Services	*Sync Services Reference Update* (new framework)
vecLib	*vecLib Reference Update*
WebKit	*WebKit Reference Update*

Mac OS X v10.5

This article summarizes the key features that are available beginning with Mac OS X version 10.5 (also known as "Leopard"). It also lists many of the documents that were created or updated for the release.

Cross-Functional Features

Mac OS X v10.5 supports the features described in this section across system technologies.

64-Bit Support

In Mac OS X v10.5, most system libraries and frameworks are now 64-bit ready, meaning they can be used in both 32-bit and 64-bit applications. The conversion of frameworks to support 64-bit required some implementation changes to ensure the proper handling of 64-bit data structures; however, most of these changes should be transparent to your use of the frameworks. Building for 64-bit means you can create applications that address extremely large data sets, up to 128TB on the current Intel-based CPUs. On Intel-based Macintosh computers, some 64-bit applications may even run faster than their 32-bit equivalents because of the availability of extra processor resources in 64-bit mode.

Although most APIs support 64-bit development, some older APIs were not ported to 64-bit or offer restricted support for 64-bit applications. Many of these APIs are legacy Carbon managers that have been either wholly or partially deprecated in favor of more modern equivalents. What follows is a partial list of APIs that will not support 64-bit. For a complete description of 64-bit support in Carbon, see *64-Bit Guide for Carbon Developers*.

- Code Fragment Manager (use the Mach-O executable format instead)
- Desktop Manager (use Icon Services and Launch Services instead)
- Display Manager (use Quartz Services instead)
- QuickDraw (use Quartz or Cocoa instead)
- QuickTime Musical Instruments (use Core Audio instead)
- Sound Manager (use Core Audio instead)

In addition to the list of deprecated APIs, there are a few modern APIs that are not deprecated, but which have not been ported to 64-bit. Development of 32-bit applications with these APIs is still supported, but if you want to create a 64-bit application, you must use alternative technologies. Among these APIs are the following:

- The entire QuickTime C API (not deprecated, but developers should use QuickTime Kit instead in 64-bit applications)
- HIToolbox, Window Manager, and most other Carbon user interface APIs (not deprecated, but developers should use Cocoa user interface classes and other alternatives); see *64-Bit Guide for Carbon Developers* for the list of specific APIs and transition paths.

Mac OS X uses the LP64 model that is in use by other 64-bit UNIX systems, which means fewer headaches when porting from other operating systems. For general information on the LP64 model and how to write 64-bit applications, see *64-Bit Transition Guide*. For Cocoa-specific transition information, see *64-Bit Transition Guide for Cocoa*. For Carbon-specific transition information, see *64-Bit Guide for Carbon Developers*.

Garbage Collection

Mac OS X v10.5 introduces support for garbage collection in Cocoa applications. Garbage collection is a form of automatic memory management used in many other development environments. When you implement an application using garbage collection, it becomes unnecessary for you to issue `retain`, `release`, and `autorelease` messages to retain or free your Cocoa objects. Instead, the garbage collector keeps track of all in-use objects and automatically frees objects that are not referenced by your application.

Garbage collection in Mac OS X v10.5 is an opt-in feature. You are not required to use it in your applications. For more information about how to support garbage collection in your Cocoa applications, see *Garbage Collection Programming Guide*.

Objective-C 2.0

Objective-C 2.0 is an update to the Objective-C language that adds support for many modern language features, including the following:

- Object properties, which offer an alternative way to declare member variables
- Support for garbage collection; see "Garbage Collection" (page 22)
- A new `for` operator syntax for performing fast enumerations of collections
- Protocol enhancements
- Deprecation syntax

For information about the new features of Objective-C 2.0, see the following documents:

- *The Objective-C 2.0 Programming Language*
- *Objective-C 2.0 Runtime Reference*

Resolution-Independent User Interface

Resolution independence decouples the resolution of the user's screen from the resolution you use to do drawing in your source code. While previous versions of Mac OS X assumed a screen resolution of 72 dots per inch (dpi), most modern screens actually have resolutions that are 100 dpi or more. Content rendered for a 72 dpi screen appears smaller on such screens—a problem that will only get worse as screen resolutions increase. To address this issue, Mac OS X v10.5 supports content scaling for screen-based rendering.

When drawing to a screen where content scaling is enabled, a scale factor is applied to content to render it at the correct size. For content drawn using Quartz and Cocoa primitives, this scaling behavior works seamlessly in most cases to retain your window's crisp graphics. If your code assumes a specific pixel alignment or screen

resolution, however, that code may not render correctly at scale factors greater than or less than 1.0. For example, bitmap images may be automatically scaled during drawing to fit into the larger pixel space, a process that may introduce artifacts.

Although the Mac OS X frameworks handle many aspects related to resolution-independent drawing, there are still things you need to do in your drawing code to support resolution independence:

- Update the images and artwork in your user interface. As the pixel density of displays increases, you need to make sure your application's custom artwork can scale accordingly. That is, your art needs to be larger in terms of pixel dimensions to avoid loss of resolution onscreen at higher scale factors.
- Update code that relies on precise pixel alignment to take the current scale factor into account. Both Cocoa and Carbon provide ways to access the current scale factor.

Content scaling is currently enabled using the Quartz Debug application. For more information about resolution-independence and how to support it in your code, see *Resolution Independence Guidelines*.

Security Enhancements

Mac OS X v10.5 includes several new improvements in the underlying operating system security. The sections that follow describe some of the key features. For general information about the security features available in Mac OS X, see *Security Overview*.

Mandatory Access Control Framework

The Mandatory Access Control (MAC) framework provides a fine-grained security architecture for controlling the execution of processes at the kernel level. This feature enables the “sandboxing” of applications, which lets you limit the access of a given application to only those features you designate. You might use this feature to prevent an application that should not need network access from communicating over the network.

Code Signing

Code signing enables you to attach a digital signature to your applications. Mac OS X v10.5 uses these signatures to check the validity of your application bundle and ensure that it has not been modified. If the contents of your bundle change, perhaps because they have been tampered with, the system alerts the user to this fact. Code signing also results in fewer alert panels after installing system updates because the validity of signed applications can be guaranteed using the signature instead of by prompting the user.

For more information on code signing, see *Code Signing Guide*.

Restrictions on Executing Dynamically Generated Code

One way for malicious programs to bypass system security is to inject some data (such as a carefully crafted image or other seemingly valid data file) into a program’s process space and trick the program into executing the code contained in that data. In earlier versions of Mac OS X, the system placed no restrictions on where code could reside in the process space. In Mac OS X v10.4, a change was made to prevent the execution of code in a program’s stack space on Intel-based Macintosh computers by default. If needed, a program could override this behavior by passing the `allow_stack_execute` option to the `ld` program at link time or by dynamically allowing execution in this space using the `mprotect` system call.

In Mac OS X v10.5, the system is increasing the restrictions on executing code in two ways. First, the system now disallows the execution of stack-based code on both PowerPC-based and Intel-based Macintosh computers by default. Second, for 64-bit programs, the system now disallows attempts to execute code in any portion of the process space unless that portion is explicitly marked as executable. Most developers may not notice these changes because code compiled and linked statically is automatically marked as executable by the `ld` program. If your 64-bit application generates code dynamically, however, you must explicitly mark that code as executable or your program will receive a `SIGBUS` signal and exit when trying to execute that code. As before, you can use the `mprotect` system call with the `PROT_EXEC` option to grant execute permissions to a block of memory containing dynamically generated code. For information on how to use this call, see `mprotect` man page.

For more information, see *Stack Execution Release Notes*.

Quarantine

Applications that download files from the Internet or receive files from external sources (such as email attachments) can use the Quarantine feature to provide a first line of defense against malicious software such as Trojan horses. When an application receives an unknown file, it should add quarantine attributes to the file using new functions found in Launch Services. The attributes associate basic information with the file, such as its type, when it was received, and the URL from which it came. When the user tries to open a file that has quarantine attributes associated with it, Mac OS X inspects the file and automatically prevents known malicious files from being opened. For other files, the system asks the user what to do about the file, providing the user with information found in the quarantine attributes. If the user approves the opening of the file, the quarantine for that file is lifted.

If you are developing a web browser or email program, or if your software somehow deals with files from unknown sources, you should use the Quarantine feature as part of your program's basic security procedures. Quarantine is part of the Launch Services API, which is itself part of the Core Services framework. For more information about the Quarantine API, see the `LSQuarantine.h` header file in that framework.

New and Updated Features

Mac OS X v10.5 supports the following new and updated features, which are listed alphabetically:

Address Book Framework

The Address Book framework adds support for sharing accounts, which are part of the Identity Services feature (see "Identity Services" (page 33)). The Address Book framework includes a new `ABIdentityPicker` class, which posts a user interface for assigning sharing accounts to specific features of your application. For example, a slideshow application could use this feature to restrict access to the slideshows it manages.

For reference information about the Address Book framework, see *Address Book Objective-C Framework Reference* or *Address Book C Framework Reference*.

AppleScript

AppleScript in Mac OS X v10.5 has been updated to support the following features:

- Unicode—script text and internal text operations now use Unicode exclusively
- 64-bit support; see "64-Bit Support" (page 21)
- Easier access to applications in scripts

Mac OS X v10.5 also includes additional scripting support, which is described in "Scripting" (page 37).

Automator

The Automator application includes a new, more intuitive user interface for finding actions and building workflows. It also includes several new features that make it easier to create workflows and dynamically modify actions as they execute. These features include the following:

- Watch Me Do—this feature lets you build an action by recording your interactions with Mac OS X and its applications. The Automator toolbar now includes a Record button for initiating recording of an action.
- Workflow variables—these are placeholders that you can use to represent a piece of text or a value (such as the current date) that can be evaluated at workflow runtime. You can reuse variables throughout your workflow to avoid repetitive data entry.
- Application integration—you can now integrate workflows into your application using the `AMWorkflow` and `AMWorkflowController` classes. These classes give your application the ability to execute workflows and control input and variable definitions programmatically.

For information about the `AMWorkflow` and `AMWorkflowController` classes, see *Automator Framework Reference*.

Calendar Store Framework

The Calendar Store framework provides access to data from the iCal application. You can fetch calendars, events, and tasks from the iCal data storage using several different techniques, including predicate-based queries. You can register for notifications that let you know when calendar, event, and task objects change. You can also make changes to records and save those changes to the user's calendar.

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

Carbon

The HIToolbox includes new functions to make it easier to integrate Cocoa views into your Carbon applications. Using these functions, you can create HIView-based wrappers for your Cocoa `NSView` objects and embed the corresponding `HIViewRef` objects into your windows. Once embedded, you use the standard HIView functions to manipulate the wrapper objects. Wrapped Cocoa views can be used in both composited and noncomposited windows.

The new Text Input Sources API now provides information about and supports the manipulation of text input sources. This API replaces the input source management functions in the Script Manager (such as the `KeyScript` function), Keyboard Layout Access (KL API), and the Text Services Manager (such as the `SetDefaultInputMethod` function).

For information about creating wrappers for Cocoa views, see *HIView Reference*. For general information about Carbon and Cocoa integration, see *Carbon-Cocoa Integration Guide*.

Cocoa

The Cocoa framework includes new classes and methods supporting several new features.

- Support for resolution-independent user interfaces; see "Resolution-Independent User Interface" (page 22)
- 64-bit versions of the Cocoa frameworks; see "64-Bit Support" (page 21)
- Garbage collection; see "Garbage Collection" (page 22)
- Support for animations in windows and views; see "Animation Support" (page 27)
- Improvements to the Cocoa text system; see "Text System Improvements" (page 27)
- Improvements to Cocoa bindings; see "Cocoa Bindings Improvements" (page 27)
- Support for creating implied and explicit animations in `NSView` and some `NSControl` objects
- Support for using views in menus
- A new `NSCollectionView` control for creating matrices using view objects instead of cells.
- A new `NSRuleEditor` control for creating queries and predicates
- A new `NSPathControl` object for displaying file-system paths graphically
- New `NSOperation` and `NSOperationQueue` objects for managing threaded operations and their dependencies
- Exposure of the previously private `NSCondition` class, which can be used to manage the flow of execution in threaded applications
- A new `NSTrackingArea` class to support more robust mouse-tracking behavior
- Improved scriptability support, including new methods for creating scripting objects, enhanced support for sdef scriptability, and improved error sensing and reporting
- Enhanced support for uniform type identifiers (UTIs)
- General user interface improvements to the following:
  - Open and Save panels
  - `NSTableView`
  - `NSOutlineView`
- Behavior changes for the `NSEnumerator` class; see "NSEnumerator Behavior Changes" (page 28)

The following sections describe additional features now available in the Cocoa frameworks. For additional information about new Cocoa classes and methods, see *Foundation Framework Reference* and *Application Kit Framework Reference*

Animation Support

The `NSWindow` and `NSView` classes include support for new animation-based features including features found in Core Animation. These features extend the capabilities of windows and views in the following ways:

- You can use a Core Animation layer as a lightweight backing store for a view and its subviews. Layer-based views take advantage of server-side caching to boost performance and require very few changes to existing drawing code.
- Windows and views support implicit animations when you modify them using an animation proxy object. For example, when you invoke a view's `setFrameSize:` method using an animation proxy, the proxy object animates the view's transition from its current size to the new size.
- Views can now be put into full-screen mode. When doing so, Cocoa uses a Core Image filter to create a smooth transition effect.

You can use these features to simplify your animation code and improve the performance of animated effects in your application. For additional information about Core Animation, see "Core Animation" (page 28).

Text System Improvements

The Cocoa text system includes the following improvements:

- Support for noncontiguous text layout that greatly improves performance and decreases memory usage when dealing with large documents.
- New methods to support the creation of text input sources
- A grammar checking facility
- The ability to edit text attachments inline
- Smart quote support
- Automatic link detection
- Support for copying and pasting multiple text selections

Cocoa Bindings Improvements

The Cocoa bindings technology includes the following new features in Mac OS X v10.5:

- An `NSDictionaryController` class to support binding to dictionaries; see *NSDictionaryController Class Reference*
- An `NSTreeNode` class that works with `NSTreeController` objects; see *Application Kit Framework Reference*
- Additional options for pop-up bindings, including an option for mixing static menu items with dynamically generated items
- Support for bindings in many new Cocoa classes

NSEnumerator Behavior Changes

The `NSEnumerator` class is used to iterate over the contents of a collection. In Mac OS X v10.5, if you are iterating a mutable collection and modify the collection, Cocoa now throws an exception. This behavior affects only those applications that are built against the version of Cocoa found in Mac OS X v10.5 and later, but it does represent a significant change from the way Cocoa used to behave.

The new behavior enforces a more correct (and safer) approach to iteration, which is that you should never modify the collection you are enumerating. Instead, you should modify a copy of the collection or simply gather the list of objects to modify during the iteration and act upon them after the iteration is complete. Alternatively, you can use the new iteration features available in Objective-C 2.0; see "Objective-C 2.0" (page 22).

For more information about collections and iteration, see *Collections Programming Topics for Cocoa*.

Core Animation

Core Animation is a new set of Objective-C classes for doing sophisticated 2D rendering and animation. Using Core Animation, you can create everything from basic window content to Front Row–style user interfaces, and achieve respectable animation performance, without having to tune your code using OpenGL or other low-level drawing routines. This performance is achieved using server-side content caching, which restricts the compositing operations performed by the server to only those parts of a view or window whose contents actually changed.

At the heart of the Core Animation programming model are layer objects, which are similar in many ways to Cocoa views. Like views, you can arrange layers in hierarchies, change their size and position, and tell them to draw themselves. Unlike views, layers do not support event-handling, accessibility, or drag and drop. You can also manipulate the layout of layers in more ways than traditional Cocoa views. In addition to positioning layers using a layout manager, you can apply affine transforms to layers to rotate, scale, skew, or translate them in relation to their parent layer.

Layer content can be animated implicitly or explicitly depending on the actions you take. Modifying specific properties of a layer, such as its geometry, visual attributes, or children, typically triggers an implicit animation to transition from the old state to the new state of the property. For example, adding a child layer triggers an animation that causes the child layer to fade gradually into view. You can also trigger animations explicitly in a layer by modifying its transformation matrix.

You can manipulate layers independent of, or in conjunction with, the views and windows of your application. Both Cocoa and Carbon applications can take advantage of the Core Animation's integration with the `NSView` class to add animation effects to windows. Layers can also support the following types of content:

- Quartz Composer compositions
- OpenGL content
- Core Image filter effects
- Quartz and Cocoa drawing content
- QuickTime playback and capture

The Core Animation features are part of the Quartz Core framework (`QuartzCore.framework`). For information about Core Animation, see *Animation Overview*.

Core Audio

Core Audio includes the following new features in Mac OS X v10.5:

- A set of high-level services for playing and recording digital audio; see "Audio Queue" (page 29)
- Support for OpenAL v1.1; see "OpenAL" (page 29)
- An updated version of the AU Lab application; see "AU Lab" (page 30)
- The new HAL Lab application for testing and debugging audio hardware and drivers.
- New audio unit properties in the Audio Unit framework that let audio units send MIDI data to host applications
- A new set of high-level services (called Audio File Stream Services) in the Audio Toolbox framework for reading non random-access audio streams.
- Four new panner audio units (based on the new `AUPannerBase` class) that support surround sound:
  - The sound field panner unit (`AUSoundFieldPanner`)
  - The spherical head panner unit (`AUSphericalHeadPanner`)
  - The vector panner unit (`AUVectorPanner`)
  - The head-related transfer function (HRTF) panner unit (`HRTFPanner`)

For an overview of Core Audio and its features, see *Core Audio Overview*. For information on how to create audio units, see *Audio Unit Programming Guide*.

Audio Queue

The Audio Queue interface is a new set of high-level services added to the Audio Toolbox framework (`AudioToolbox.framework`). Among the new services provided by this interface are the ability to play and record digital audio, access timing and control information, and support scheduled playback and synchronization. The interface supports both linear PCM and compressed audio formats.

OpenAL

OpenAL is a cross-platform, 3D audio API appropriate for games and audio applications. It provides an environmental context for audio, allowing developers to immerse users in multichannel, directional sound. OpenAL support in Mac OS X v10.5 has been updated to include the following features:

- An audio capture mechanism
- Exponential and linear distance models
- Location offsets
- Additional accessor functions
- Selected control over Core Audio features, including mixer sample rates
- Spatial effects such as reverb and occlusion

For more information on the Mac OS X implementation of OpenAL, see http://developer.apple.com/audio/openal.html.

AU Lab

The new version of the AU Lab application (located in `/Developer/Applications/Audio`) includes the following features:

- A patch manager, supporting live switching among sets of active tracks
- Control over the visibility of tracks based on track type
- A streamlined user interface for MIDI source and audio device reassignment
- Support for up to eight channels in each track, along with support for multichannel panner units
- A new generic panner view that works with Apple and third-party multichannel panner units

Core Data 2.0

Core Data includes the following new features in Mac OS X v10.5:

- An API for creating atomic stores based on custom file formats; see *Atomic Store Programming Topics*.
- Support for schema versioning and data migration; see *Core Data Model Versioning and Data Migration Programming Guide*
- Support for new fetch request configurations, including pre-fetching and fetching just the object IDs
- Faster predicates
- Easier debugging
- Support for 64-bit applications; see "64-Bit Support" (page 21).
- Support for garbage-collected applications; see "Objective-C 2.0" (page 22).

For a list of new and changed classes and methods in Core Data, see *Core Data Reference Update*.

Core Image

Core Image in Mac OS X v10.5 includes extensions to the existing `CIFilter` class and a new class called `CIFilterGenerator`.

The `CIRAWFilter` category extends the `CIFilter` class to support initializing filters with RAW images. The methods in this category let you modify key aspects of the RAW image processing, including the white balance, exposure, sharpening, and boost settings.

The `CIFilterGenerator` class lets you create and reuse complex effects built from one or more `CIFilter` objects. You can create filter generator objects programatically or using an editor view provided by Core Image. Filters inside a filter generator object can be grouped into a filter chain and treated as a single filter. Filter generator objects are registered by their filename or, if present, by an attribute that you supply in its description.

You can save the contents of a filter generator object by writing it out to a filter generator file. A filter generator file contains archived instances of the filter objects associated with a filter generator object. Filter generator files have a unique extension and file format. Copying your saved filter generator files to the Image Units directories on the system causes them to be loaded when any of the `loadPlugins` methods are invoked.

For information on the new features of Core Image, see *Core Image Reference Collection*. For a tutorial on how to create custom image units, see *Image Unit Tutorial*.

Core Text

Introduced in Mac OS X v10.5, Core Text is a C-based API that provides you with precise control over text layout and typography. Core Text provides a layered approach to laying out and displaying Unicode text. You can modify as much or as little of the system as is required to suit your needs. Core Text also provides optimized configurations for common scenarios, saving setup time in your application. Designed for performance, Core Text is up to twice as fast as ATSUI (Apple Type Services for Unicode Imaging), the text-handling technology that it replaces.

The Core Text font API is complementary to the Core Text layout engine. Core Text font technology is designed to handle Unicode fonts natively and comprehensively, unifying disparate Mac OS X font facilities so that developers can do everything they need to do without resorting to other APIs.

For more information about Core Text, see *Core Text Programming Guide* and *Core Text Reference Collection*.

Darwin

The Darwin layer of the operating system includes the following enhancements:

- The Apache web server has been updated to version 2.0.
- More libraries now support 64-bit, including `Libcurl`, `expat`, `libbzip2`, `libxml2`, `libxslt`, `ncurses`, `zlib`, `OpenSSL`, `libiconv`, and `PAM`.
- Darwin now includes Ruby on Rails.
- The Subversion and SVK source-control tools are now included in `/usr/bin/`; in addition, you can now use `mod_dav_svn` in Apache.
- Per-session `launchd` can now be used to manage processes running for an individual user.
- Objective-C toll-free bridging to scripting languages is now supported; this allows bridging between the Foundation and Application Kit frameworks and the Python and Ruby scripting languages.
- The standard C library (`libc`) has been improved, including the addition of support for pthread cancellation points.
- Many command line tools have updated versions, including:
  - apache 2.2.2
  - automake 1.9.6
  - bison 2.3
  - curl 7.15.4
  - emacs 22.0.50
  - fetchmail 6.3.4
  - lsof 4.77
  - netcat
  - perl 5.8.8

- python 2.4
- rails 1.1.2
- ruby
- RubyGems 0.9.0
- subversion 1.3.2
- tcllib-1.8
- tcpdump 3.9.4

- The `lookupd` daemon services have been merged into the `DirectoryService` daemon.

In addition, Apple submitted both Mac OS X v10.5 and Mac OS X v10.5 Server to The Open Group for certification against the UNIX '03 product standard.

FSEvents API

The FSEvents API notifies your application when changes occur in the file system. You can use file system events to monitor directories for any changes, such as the creation, modification, or removal of contained files and directories. Although kqueues provide similar behavior, the FSEvents API provides a much simpler way to monitor many directories at once. For example, you can use file system events to monitor entire file system hierarchies rooted at a specific directory and still receive notifications about individual directories in the hierarchy. The implementation of file system events is lightweight and efficient, providing built-in coalescing when multiple changes occur within a short period of time to one or many directories.

The FSEvents API is not intended for detecting fine-grained changes to individual files. You would not use this to detect changes to an individual file as in a virus checker program. Instead, you might use FSEvents to detect general changes to a file hierarchy. For example, you might use this technology in backup software to detect what files changed. You might also use it to monitor a set of data files your application uses, but which can be modified by other applications as well.

For information on how to use the FSEvents API, see *File System Events Programming Guide*.

iChat Instant Message Framework

The Instant Message framework now includes support for injecting audio or video content into a running conference. You can use this feature to share content from your applications with the active iChat session. Shared content can include still images, audio, and video. Shared audio content is mixed with the user's live microphone to allow for the narration of video content. You render your content to a Core Video buffer through a series of callback functions and that content is automatically encoded into the H.264 video format for distribution to conference users.

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

Identity Services

Introduced in Mac OS X v10.5, Identity Services encompasses features located in the Collaboration and Core Services frameworks. Identity Services provides a way to manage groups of users on a local system. In addition to standard login accounts, administrative users can now create **sharing** accounts, which use access control lists (see "ACL Support" (page 9)) to restrict access to designated system or application resources. Sharing accounts do not have an associated home directory on the system and have much more limited privileges than traditional login accounts.

The Collaboration framework (`Collaboration.framework`) provides a set of Objective-C classes for displaying sharing account information and other identity-related user interfaces. The classes themselves are wrappers for the C-based identity management routines found in the Core Services framework. Applications can use either the Objective-C or C-based APIs to display information about users and groups and display a panel for selecting users and groups during the editing of access control lists.

For more information about the features of Identity Services and how you use those features in your applications, see *Identity Services Programming Guide*.

Image Kit

The Image Kit framework is a new Objective-C framework that makes it easy to incorporate powerful imaging services available into your applications. This framework takes advantage of features in Quartz, Core Image, OpenGL, and Core Animation (see "Core Animation" (page 28)) to provide an advanced and highly optimized development path for implementing the following features.

- Displaying images (`IKImageView` class)
- Rotating, cropping, and performing other image-editing operations
- Browsing for images using the `IKImageBrowserView` class, which supports efficient viewing of large numbers of images, supports drag and drop, and supports the ability to export images to iPhoto
- Taking pictures using the picture taker panel (`IKPictureTaker`). This panel lets you select images by browsing the file system or take pictures using an attached camera, such as a built-in iSight. The panel also supports picture manipulations, such as zoom, crop, and rotate
- Displaying slideshows using the `IKSlideshow` class, which takes images and other input and uses them to create full-screen slideshows
- Browsing for Core Image filters, with each filter displaying a description of the filter behavior and a preview of its effects
- Displaying custom views for Core Image filters. You can create custom views for filters to provide an interface for manipulating the filter parameters.

Prior to the Image Kit, presenting a user interface for a Core Image filter required you to build a view object dynamically based on the attributes of that filter. With the Image Kit, each filter can now provide custom views (subclasses of `NSView`) containing the controls needed to manipulate that filter. Filters can provide different views based on the needs of the target application. For example, filters can provide a basic view (containing a limited number of attribute controls) or an expanded view (containing controls for all attributes). Applications can integrate the views returned by a filter directly into their own windows.

Note: Carbon developers wanting to include the new custom views in their applications can take advantage of the new `NSView` integration feature; see "Carbon" (page 25).

The Image Kit framework is included as a subframework of the Quartz framework (`Quartz.framework`). For more information about the features of this framework, see the framework header files.

The Input Method Kit

The Input Method Kit (`InputMethodKit.framework`) is a new Objective-C framework for building input methods for Chinese, Japanese, and other languages. The Input Method Kit framework lets developers focus exclusively on the development of their input method product's core behavior: the text conversion engine. The framework handles tasks such as connecting to clients, running candidate windows, and several other common tasks that developers would normally have to implement themselves.

For information about its classes, see *Input Method Kit Framework Reference*.

Latent Semantic Mapping Framework

The Latent Semantic Mapping framework (`LatentSemanticMapping.framework`) contains a Unicode-based API that supports the classification of text and other token-based content into developer-defined categories, based on semantic information latent in the text. Using this API and text samples with known characteristics, you create and train maps, which you can use to analyze and classify arbitrary text. You might use such a map to determine, for example, if an email message is consistent with the user's interests.

For information about the Latent Semantic Mapping framework, see *Latent Semantic Mapping Reference*.

Mail Stationery

The Mail application in Mac OS X v10.5 supports the creation of email messages using templates. Templates provide the user with prebuilt email messages that can be customized quickly before being sent. Because templates are HTML-based, they can incorporate images and advanced formatting to give the user's email a much more stylish and sophisticated appearance.

Developers and web designers can create custom template packages for external or internal users. Each template consists of an HTML page, property list file, and images packaged together in a bundle, which is then stored in the Mail application's stationery directory. The HTML page and images define the content of the email message and can include drop zones for custom user content. The property list file provides Mail with information about the template, such as its name, ID, and the name of its thumbnail image.

For information about how to create new stationery templates, see Mail Stationery Release Notes for Mac OS X v10.5.

OpenGL

The AGL framework in Mac OS X v10.5 adds support for the following features:

- Support for multiple OpenGL threads, which increases performance by offloading CPU-based processing onto separate threads where they can be processed by available processor cores.
- Attaching an AGL context to `WindowRef` and `HIView` objects, thus eliminating the need to use a QuickDraw port
- Pixel buffer objects
- Color managed texture images in the sRGB color space
- Improvements in the shader programming API
- Support for 64-bit addressing

In addition to the improvements to the OpenGL support itself, Mac OS X v10.5 includes updates to the OpenGL profiling tools to help you analyze your OpenGL programs and gather performance statistics.

For information about the AGL support, see *AGL Reference*.

Quartz

Quartz 2D includes additional functions for manipulating fonts. These functions are associated with the `CGFontRef` opaque type. They let you retrieve various font attributes, including the ascent, descent, leading, and bounding box of the font. For more information, see *CGFont Reference*.

Quartz Composer

The framework supporting Quartz Composer includes improvements to help you integrate compositions into your applications. In addition to new methods on existing classes, there is a new `QCCompositionLayer` class that loads, plays, and controls Quartz Composer compositions in a Core Animation hierarchy; see also "Core Animation" (page 28). Other Quartz Composer features include the following:

- A system-wide shared repository for compositions
- A composition picker interface for you to use in your application
- Support for writing plug-ins for Quartz Composer

For information about the Quartz Composer framework, see *Quartz Composer Reference Collection*. For information about using the Quartz Composer application, see *Quartz Composer User Guide*.

QuickTime

Mac OS X v10.5 ships with QuickTime 7.2 installed, which builds on the features introduced in QuickTime 7.1. QuickTime 7.1 includes the following features:

- Improved quality for video playback due to the following changes:
  - Improved correction for nonsquare pixels
  - Use of the clean aperture, which is the user-displayable region of video that does not contain transition artifacts caused by the encoding process

    - Support for aperture mode dimensions in QuickTime files helps QuickTime player scale, crop, or preserve content properly

- Improvements to H.264 encoding, including support for transparent alpha layers
- A new aperture mode implementation, including functions to manipulate aperture mode information.
- Functions for manipulating audio in the following ways:
    - changing the pitch while keeping the same playback rate
    - Setting the varispeed and TimePitch AU Render Quality
    - Setting the pitch/rate converter quality from the StdAudio panel, so that exports from scaled edits have improved quality

Note: The C-based QuickTime API supports 32-bit applications only. To support 64-bit applications, you need to use "QuickTime Kit" (page 36) instead.

QuickTime 7.2 adds the following features:

- Improvements in the QuickTime exporter
- Support for new video devices
- Continued improvements in H.264 playback
- Audio and video optimizations
- Security fixes

For information about changes in QuickTime 7.2, see *QuickTime 7.2.1 Update Guide*. For information about changes in QuickTime 7.1, see *QuickTime 7.1 Update Guide*.

QuickTime Kit

The QuickTime Kit framework includes updates to the existing `QTMovie` and `QTMovieView` classes along with several new classes to support professional-grade capture from a number of different devices. New features include the following:

- New classes (`QTCaptureSession`, `QTCaptureView`, and others) to support audio and video capture from one or more external sources, including, but not limited to the following:
    - Cameras (including iSight), microphones, and other USB, FireWire, and DV media devices
    - Raw data from the file system
    - The pixel contents of a computer display
- Support for 64-bit applications
- Support for aperture mode dimensions in the `QTMovie` and`QTTrack` classes
- Record captured media to one or more output destinations, including, but not limited to, the following:
    - A `QTMovie` object

- ❏ A Cocoa view that previews video media captured from the input sources
- ❏ Cameras and other external devices that support recording data transferred from a computer, such as DV cameras

The new classes provide an easy means of extending the QuickTime Kit framework functionality, with particular emphasis on providing the ability to add new types of inputs and outputs. The input and output classes included with the framework provide all components necessary to implement the most common use case for a media capture system: recording from a camera to a QuickTime file. Video capture includes frame accurate audio/video synchronization. You can preview captured content and save it to a file or stream.

For more information about the new classes in the QuickTime Kit, see *QTKit Capture Programming Guide* and *QTKit Framework Reference*.

RSS/Atom Support

The Publication Subscription framework (`PubSub.framework`) is a new framework that provides high-level support for subscribing to RSS and Atom feeds. You can use the framework to subscribe to podcasts, photocasts, and any other feed-based document. The framework handles all the feed downloads and updates automatically and provides your application with the data from the feed.

For information about the Publication Subscription framework, see *Publication Subscription Programming Guide* and *Publication Subscription Framework Reference*.

Scripting

Mac OS X v10.5 includes new bridging mechanisms that make it possible to control scriptable applications from your application using the Python, Ruby, and Objective-C languages. For example, you can generate an interface to iTunes so that your application can access scriptable iTunes features using Objective-C syntax. Bridging is much more convenient than building Apple events programmatically using the older Carbon or Cocoa routines. It is also more convenient, and much more efficient, than using the `NSAppleScript` class to execute scripts from your application. For scripting bridge reference information, see *Scripting Bridge Framework Reference*.

You create the bridging files for an application using the `sdp` command-line tool. For information about using this tool, see the `sdp` man page.

Spotlight

The following improvements have been made to Spotlight and the supporting file-system technologies:

- The File manager now includes the `FSReplaceObject` and `FSPathReplaceObject` functions, which swap the contents of a file while preserving its original metadata. These functions are declared in the `Files.h` header file in the Core Services framework.
- Support for searching metadata on remote servers

- Functions for storing lineage information with a file to track modifications to that file. Lineage information can be used to set, alter, and store data about the relationships between different versions of the same logical file. For more information about working with lineage information, see *MDLineage Reference*.

Sync Services

The following sections describe several new features related to synchronizing data in Mac OS X v10.5.

Sync Services Framework

The Sync Services framework (`SyncServices.framework`) has been updated to include a new session driver class called `ISyncSessionDriver`. This new class greatly simplifies the process of syncing client records with your own applications. For more information about this class, see *Sync Services Framework Reference*.

iSync Plug-in Development

Mac OS X v10.5 also introduces a new application that greatly simplifies the creation of iSync plug-ins for third-party phones and devices. The iSync Plug-in Maker application provides a graphical tool that you use to specify the properties of your device. The application then generates a fully functional plug-in that you can load into iSync and test. For information about using this application to create iSync plug-ins, see *iSync Plug-in Maker User Guide*.

Syncing Application Preferences

In Mac OS X v10.5, it is now possible to synchronize application preferences across multiple machines using .Mac. For third-party applications, the default behavior is to synchronize all user preferences except "ByHost" preferences. If your application contains preferences that should not be synchronized, you can add an array to your application's `Info.plist` file associated with the `com.apple.PreferenceSync.ExcludeSyncKeys` key. The array contains the keys associated with any preferences you do not want to synchronize.

Time Machine

Mac OS X v10.5 includes a new backup feature (called "Time Machine") that protects user data from accidental loss. Included with this feature is a set of programmer-level functions that you can use to exclude unimportant files from the backup set. For example, you might use these functions to exclude your application's cache files or any files that can be recreated easily. Excluding these types of files improves backup performance and reduces the amount of space required to back up the user's system.

For information about the new functions, see *Backup Core Reference*.

Developer Tools

Apple provides a comprehensive suite of developer tools for creating Mac OS X software. The Xcode tools include applications to help you design, create, debug, and optimize your software. This tools suite also includes header files, sample code, and documentation for Apple technologies. You can download Xcode from the members area of the Apple Developer Connection (ADC) website (http://connect.apple.com/). Registration is required but free.

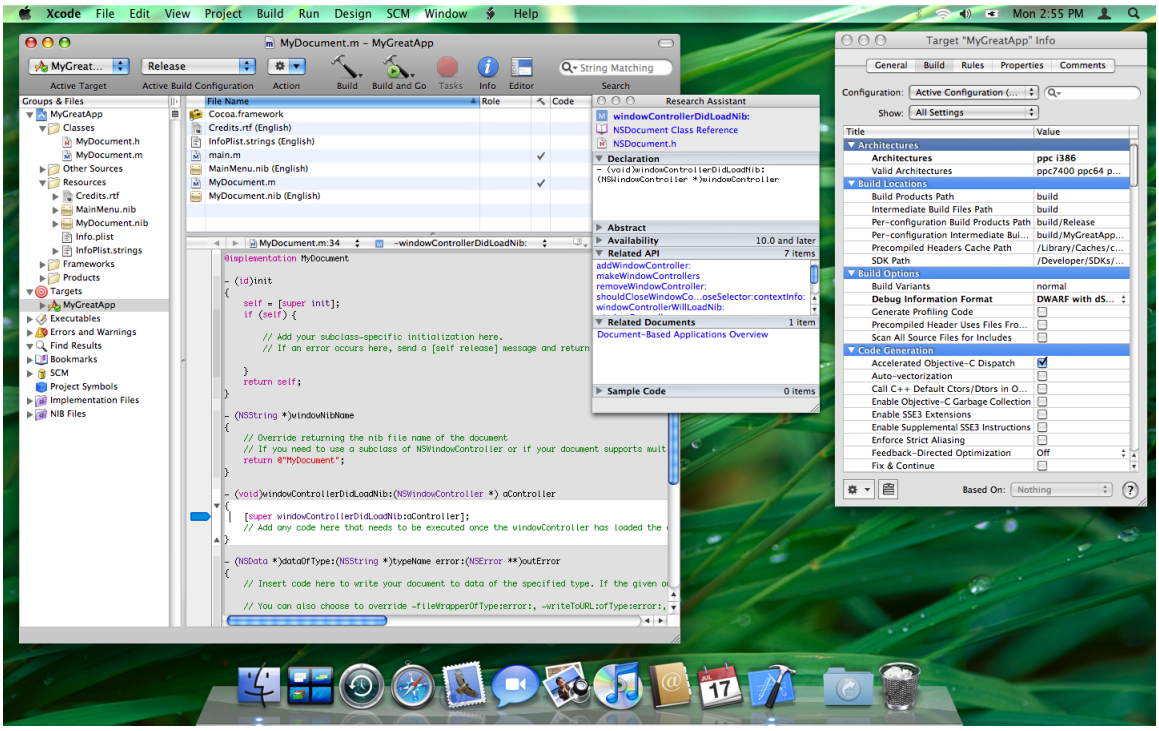
In Mac OS X v10.5 and later, it is possible to install multiple versions of Xcode on a single computer and run the applications and tools from different versions side-by-side. The applications listed in the following sections are installed in *<Xcode>*`/Applications`, where *<Xcode>* is the root directory of your Xcode installation. The default installation directory for Xcode is the `/Developer` directory.

The following sections detail the improvements to the developer tools suite in Mac OS X v10.5.

Xcode 3.0

The Xcode 3.0 application introduces several new features intended to make the development process easier. In addition to basic workflow improvements, Xcode includes new tools to simplify everything from writing your code to debugging your code. Figure 1 shows the Xcode application window, inspector, and Research Assistant.

Figure 1 Xcode application



The following sections provide basic information about the new features. For detailed information about new Xcode features, see *What's New in Xcode*.

Xcode Editor Improvements

The Xcode editing environment includes performance improvements and also offers several new features that go beyond basic text editing to help improve the development workflow. The new features include the following:

- Improvements to the performance of typing, scrolling, and opening files. The Xcode editor now opens and scrolls large source documents up to 10 times faster than before.
- Code annotations, such as errors and warnings, are now shown inline with the code itself, and not just as icons in the gutter. You can control the visibility of annotations using the segmented control in the navigation bar.
- Code folding helps you organize your source files by letting you temporarily hide the content of a method or function in the editor window. You can initiate code folding by holding down the Command and Option keys and pressing either the left or right arrow key. A ribbon to the left of the text shows the current nesting depth and contains widgets to fold and unfold code blocks.
- Improved syntax coloring.
- Changes to Code Sense code completion, including a new interface that is faster and more intuitive and changes to the Code Sense shortcut keys. Code Sense heuristics have also been tuned to provide more accurate completions, along with a "most likely" inline completion as you type. This feature is similar to the auto-completion features found in Mail, Terminal, and other applications.

Streamlined Debugging

In Xcode 3.0, the distinction between "Run" and "Debug" has been eliminated. Now, you simply build your executable and run it. Hitting a breakpoint interrupts the program and displays the debugger window. (You can also debug from any editor window now.) Other features of the new debugging environment include the following:

- Variable tooltips.
- Reorganization (and in some cases consolidation) of toolbar and menu items to improve space usage, while still keeping all the needed tools available.
- Consolidation of the Standard I/O Log, Run Log, and Console log into the Console log window.

Research Assistant

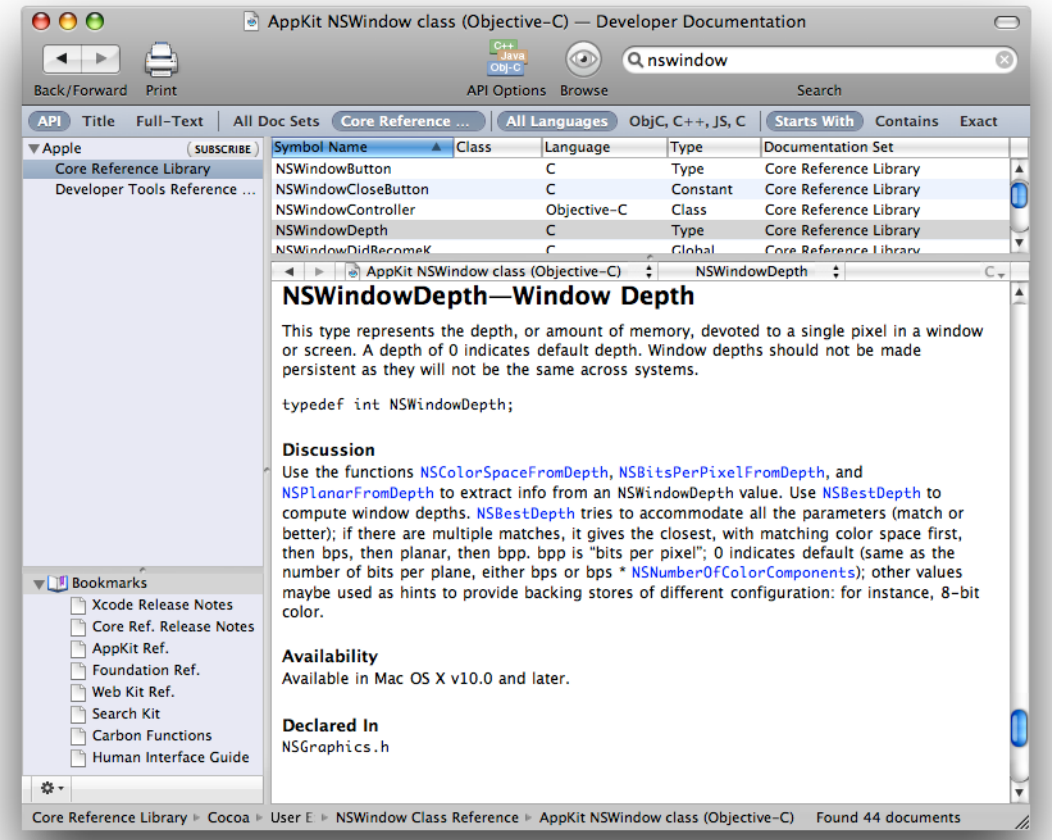
The Xcode documentation window provides an easy way for you to find information quickly, but now there is an even faster way to get the information you need as you type. This feature is called the Research Assistant.

The Research Assistant is an inspector that displays documentation for the currently selected text (see Figure 1 (page 39)). As the selection changes, the Research Assistant updates the information in its floating window to reflect the classes, methods, and functions you are currently using. This window shows the declaration, abstract, and availability information for the selection along with the framework containing the selected identifier, relevant documentation resources, and related methods and functions you might be interested in using.

Documentation Window

The documentation window (Figure 2) in Xcode provides an environment for searching and browsing the documentation. This window provides you with fast access to Apple's developer documentation and gives you tools for searching its content. You can search by title, by language, and by content and can focus your search on the documents in a particular documentation set.

Figure 2 Xcode documentation window



Documentation sets are collections of documents that can be installed, browsed, and searched independently. Documentation sets make it easier to install only the documentation you need for your development, reducing the amount of disk space needed for your developer tools installation. In addition to the Apple-provided documentation sets, third parties can implement their own documentation sets and have them appear in the Xcode documentation window. For information on how to create custom documentation sets, see *Documentation Set Guide*.

Refactoring Tools

Xcode's new refactoring tools add the ability to make large-scale changes to your Objective-C source code. The changes you make are propagated throughout your code base, making sure that the changes do not break your builds. Supported transformations include the following:

- Renaming instance methods
- Creating new superclasses
- Moving methods into a superclass
- Converting accessor methods to support Objective-C 2.0 properties
- Modernizing appropriate `for` loops to use the new fast enumeration syntax introduced in Objective-C 2.0

Build Settings

The Build pane in the inspector now supports the following improvements:

- Per-architecture build settings. You can now set different build settings for each architecture your product supports.
- 32-bit and 64-bit architecture checkboxes.

Project Snapshots

Project snapshots provide a lightweight form of local source control for Xcode projects. Using this feature, you can take a "snapshot" of your project's state at any point during development, such as after a successful build or immediately prior to refactoring your code. If after making subsequent changes you decide those changes were not useful, you can revert your project files back to the previous snapshot state. Because snapshots are local, your intermediate changes need never be committed to source control.

Project Versioning

Xcode projects now have a Compatibility pane in the project inspector that lets you determine whether you want an Xcode 3.0–only project or one that can be used by previous versions of Xcode. Marking a project as Xcode 3.0–only generates an alert whenever you try to use an Xcode feature that is not present in previous versions of the application.

Repository Management

Xcode now supports the management of multiple SCM repositories to allow you to perform tasks such as the following:

- Initial checkout of projects
- Tagging source files
- Branching
- Importing and exporting files

Miscellaneous Improvements

Xcode includes the following additional improvements:

- Dwarf debugging information is now generated by default for new projects.
- Support for the ANT build system, which can be used to build Java and WebObjects projects.
- Core Data modeling tools improvement. For more information, see "Core Data 2.0" (page 30).
- An updated version of the GCC compiler that supports Objective-C 2.0; see "Objective-C 2.0" (page 22)

Interface Builder 3.0

Interface Builder has undergone its first major update in over a decade and includes numerous improvements to the application appearance and workflow. In addition to support for new controls (such as Cocoa toolbars), the new version features major improvements in the following areas:

- Object palettes
- Inspectors
- Creating connections between objects
- Custom object integration
- New support for refactoring your nib files
- Support for adding Core Animation–based behaviors

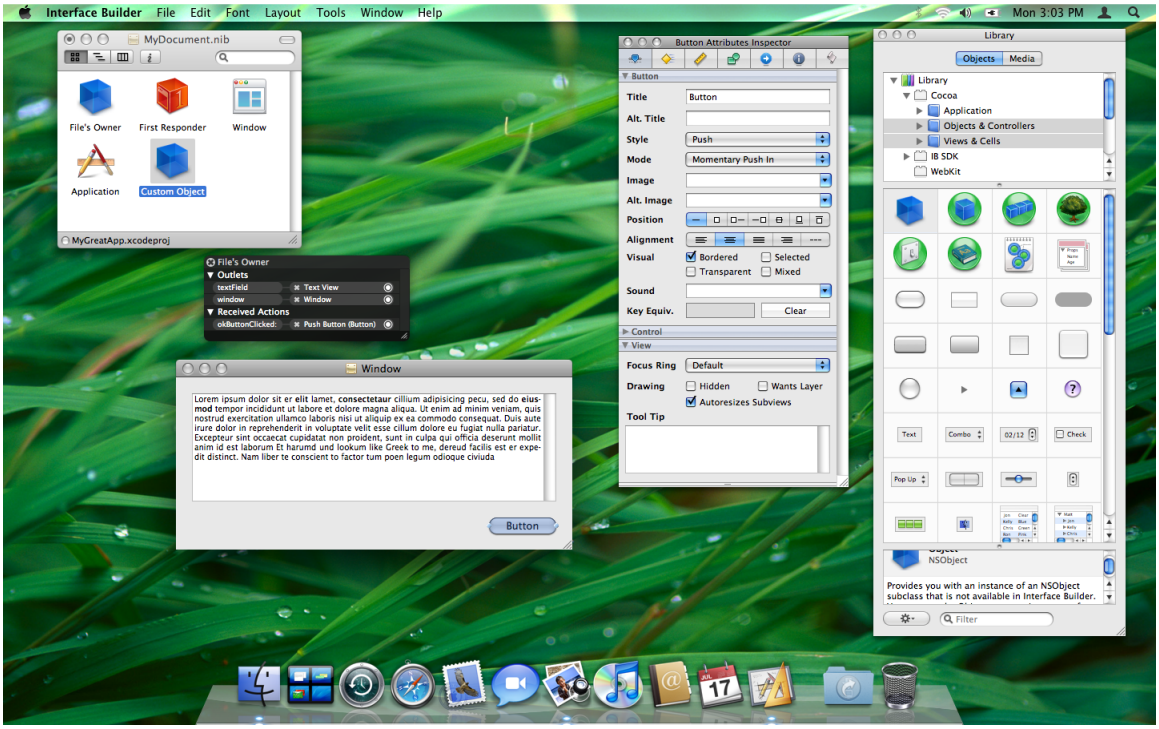
In place of the old palette window, Interface Builder now lets you choose views and controls from an object library. This library groups related controls into categories and lets you either browse for the desired element or search for it by name. The library also provides controls for displaying controls in the size (regular, small, mini) that you want and seeing your selection reflected in the library window.

Thanks to improvements in the underlying infrastructure for managing controls, it is now much easier to integrate your own controls into Interface Builder. Interface Builder includes the Interface Builder Kit framework and an Xcode project template for creating custom plug-ins. Most of the work needed to integrate custom controls now occurs directly in the nib file of your plug-in, with more advanced features requiring only modest amounts of code.

Because of its infrastructure improvements, Interface Builder now supports the ability to inspect multiple objects. The basic attributes inspector breaks up attributes by class, allowing each class to define a separate section for its own content. Selecting multiple objects of different types still displays any attributes inspectors that are common among the selected objects. Other types of inspectors, such as the size inspector also remain active in most cases, allowing you to modify the size of multiple controls all at once.

Figure 3 shows the new look of the Interface Builder application, including the new document, library, and inspector windows.

Figure 3 Interface Builder 3.0



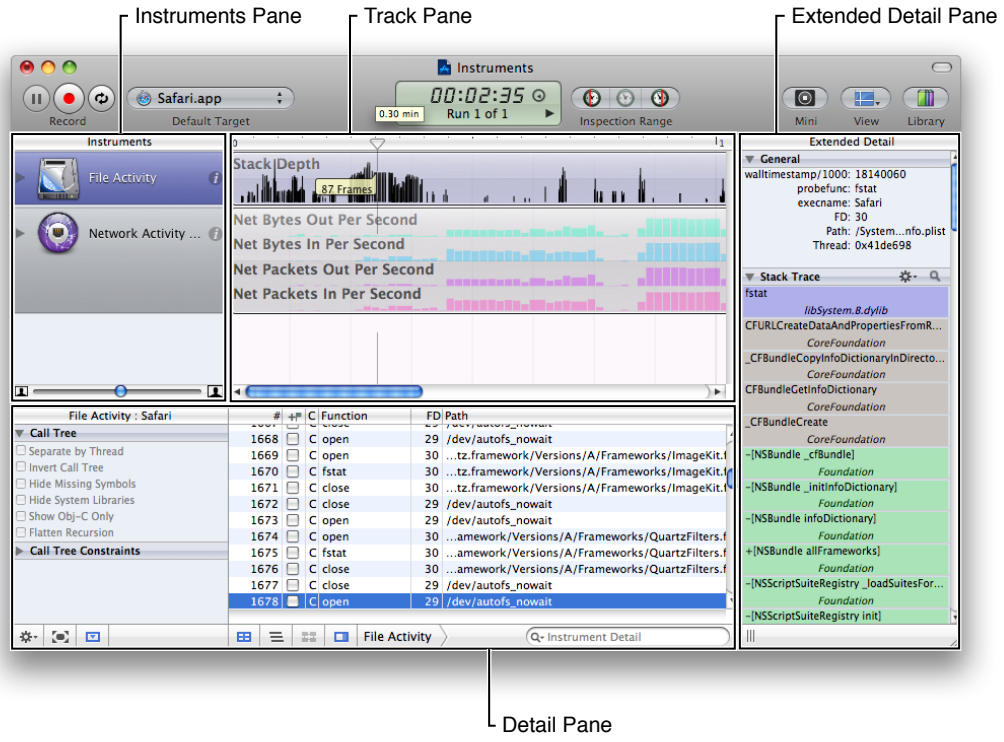
For information about Interface Builder features and how to use them, see *Interface Builder User Guide*. For information about how to integrate your own custom controls into Interface Builder, see *Interface Builder Plug-In Programming Guide*.

Instruments

Instruments is a new application that complements existing performance tools such as Shark. The Instruments application provides an integrated set of tools for debugging and analyzing performance problems in your application.

Figure 4 shows the Instruments user interface. Instead of analyzing different aspects of your program's performance individually, Instruments lets you configure multiple "instruments" to gather several different types of data simultaneously. Instruments let you gather information about CPU usage, disk I/O, memory usage, garbage collection events, Cocoa notifications, and more. The data for each instrument is displayed along a horizontal time axis, giving you the chance to correlate results from different instruments and thus see the "big picture" of your application's behavior.

Figure 4 The Instruments application interface



An important aspect of Instruments is the repeatability of data gathering operations. Instruments lets you record a sequence of events in your application and store them in the master track. You can then replay a sequence of events to reproduce the exact same conditions in your application. This repeatability means that each new set of data you gather can be compared directly to any old sets, resulting in a more meaningful comparison of performance data. It also means that you can automate much of the data gathering operation. Because events are shown alongside data results, it is easier to correlate performance problems with the events that caused them.

For an introduction to Instruments, see *Instruments User Guide*.

Dashcode

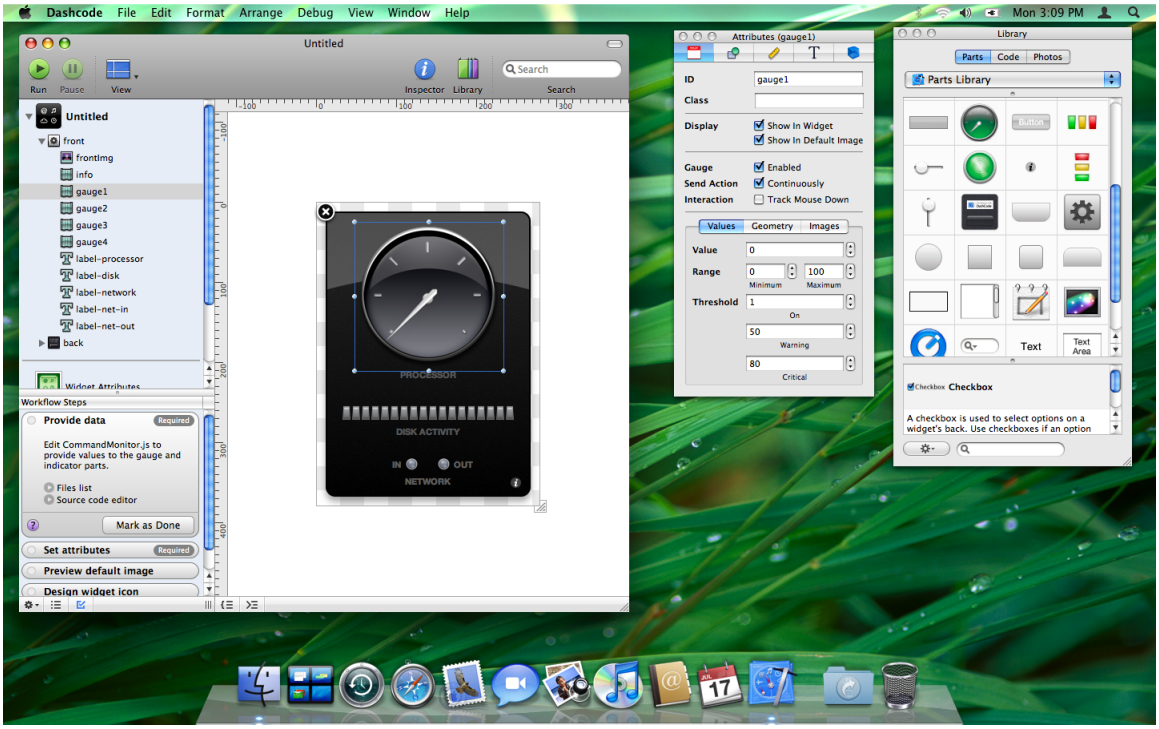
Dashcode is an integrated environment for laying out, coding, and testing Dashboard widgets. Although users see and use widgets as applications, they're actually packaged webpages powered by standard technologies such as HTML, CSS, and JavaScript. Although it is easy for anyone with web design experience to build a widget using existing webpage editors, as a widget's code and layout get more complex, managing and testing of that widget becomes increasingly difficult. Dashcode provides the following features to help simplify the widget design process:

- A project manager to marshall your widget's resources
- Visual tools to design your widget interface
- Tools to set metadata values, specify required images, and package your widget
- A source code editor to implement your widget's behavior

- A debugger to help you resolve issues in your widget's implementation

Figure 5 shows the Dashcode canvas, inspector, and library windows. The canvas is a drag-and-drop layout environment where you lay out widgets visually. Using the inspector window, you can apply style information to the controls, text, and shape elements that you drag in from the library.

Figure 5 Dashcode canvas



For more information about Dashcode, see *Dashcode User Guide*.

API Delta Documents

Table 1 list the documents describing the API changes that were made in system frameworks for Mac OS X v10.5.

Table 1 Mac OS X v10.5 delta documents

Framework	Document
Accelerate	*Accelerate Reference Update*
Address Book	*Address Book Reference Update*
AGL	*AGL Reference Update*
AppKit	*Application Kit Reference Update*

Framework	Document
Application Services	*Application Services Reference Update*
Automator	*Automator Reference Update*
Calendar Store	*Calendar Store Reference Update* (new framework)
Carbon	*Carbon Reference Update*
Collaboration	*Collaboration Reference Update* (new framework)
Core Data	*Core Data Reference Update*
Core Foundation	*Core Foundation Reference Update*
Core Services	*Core Services Reference Update*
Directory Service	*Directory Service Reference Update* (new framework)
Foundation	*Foundation Reference Update*
Instant Message	*Instant Message Reference Update*
OpenGL	*OpenGL Reference Update*
Preference Panes	*Preference Panes Reference Update*
PubSub	*Publication Subscription Reference Update* (new framework)
QTKit	*QTKit Reference Update*
Quartz Core	*Quartz Core Reference Update*
Quartz	*Quartz Reference Update*
QuickTime	*QuickTime Reference Update*
Security	*Security Reference Update*
Security Interface	*Security Interface 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 Mac OS X*.

Date	Notes
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 Mac 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 Mac OS X.