Apple Worldwide Developer Conference 2009 Trip Report
June 8-12, 2009

by Lawrence Peng, Ph.D.

DISCLAIMER: These notes are the work of the author. They do NOT represent an official position of any other person or organization.

Some Major Take-Home Points

Snow Leopard only supports Intel processors. The end of the line is here for the PowerPC.

Snow Leopard will be a $29 upgrade for Leopard users. $49 for a 5-license family pack.

For Tiger on Intel, the Mac Box Set (Mac OS X Snow Leopard, iLife '09 and iWork '09) will be available for $169 and a Family Pack is available for $229.

Snow Leopard will boot into the 64 bit kernel by default for certain Mac Pro and Xserve configurations. It will boot into the 32 bit kernel by default for all other models. You can modify the default setting as per your requirements and available hardware.

Java 1.6.x will be the default Java install for Snow Leopard. It will run in both Intel 32 and 64 bit. Java 1.4.x and 1.5.x are being removed.

AppleScript Studio is deprecated in Snow Leopard. It is being replaced by the AppleScriptObjC framework (ASOC) to allow Cocoa programming via AppleScript in the same way as with Python and Ruby.

The Directory Services API is deprecated in Snow Leopard. It is being replaced by the Open Directory Framework.

There is a continual increases in code sharing between Mac and iPhone. With Snow Leopard and iPhone 3 you now have Core Location moving to the Mac, and Core Data moving to the iPhone.

With Dashcode 3 you now have bindings for JavaScript very similar to that of Cocoa Bindings.

WWDC sold out again this year.

Keynote Address

Phil Schiller (Senior Vice President of Worldwide Marketing)
Bertrand Serlet (Senior Vice President, Software Engineering)
Scott Forstall (Senior Vice President, iPhone Software)

The weblink for the keynote address is:

http://events.apple.com.edgesuite.net/0906paowdnv/event/index.html?internal=ijalrmacu

The keynote began with an opening commercial that has John Hodgman as PC. PC is frustrated and really struggling to wish us a good WWDC 2009.

There are 5200 developers from 54 countries here. This week there are 129 sessions, 147 labs, and 1000 Apple engineers on-site.

Apple began by showing the growing number of active OS X users (Mac, iPhone and iPod Touch). In the first 5 years since OS X launched, the number of active users in the first 5 years rose steadily to 25 million. In the last two years it has grown to nearly 75 million.

Now moving on to updates, starting with the Mac

Today for many users, the first choice is often new notebook. Phil did a quick review of the unibody notebooks, which started with the MacBook Air.

Today Apple announced a new version of the 15 inch MacBook Pro. It is built on the unibody architecture, and is as thin and as light as the previous model. It has the same built in Li-Polymer battery like the 17 inch MacBook Pro. The lithium polymer battery provides up to 7 hours and up to 1000 charge cycles, and is estimated to last up to 5 years. This is in contrast to 300 charge cycles for older Lithium ion.

In place of the Express card we have the SD card slot. Apple claims the Express card slot was used by less than a single digit percentage of their customers.

The new 15 inch MacBook Pro can be configured with an Intel dual core processor up to 3.06 GHz with 6 MB cache. It supports up to 8 GB 1066 MHz DDR 3 (4 GB is now standard), and can have up to a 500 GB hard drive or a 256 GB solid state drive. Prices start at $1699.

The 17 inch unibody MacBook Pro configuration is being updated, and retains the Express card slot. Both the new 15 and 17 inch MacBook Pros are available today.

The 13 inch MacBook is being updated, and is now called the 13 inch MacBook Pro. It has the same built in battery as the 17 inch MacBook Pro, and an SD card slot (it never had an Express card slot). As with the other MacBook Pros, it supports up to 8 GB 1066 MHz DDR 3 RAM, can have up to a 500 GB hard drive or a 256 GB solid state drive, has a built in backlit keyboard, and has a Firewire 800 port. The new 13 inch MacBook Pro starts at $1199, and is available today.

There MacBook Air has been updated as well. The base price drops to 1499, there is an option for an 128 GB solid state drive, and a faster processor.

All notebook meets EPEAT (Electronic Products Environmental Assessment Tool) gold standard, and Energy Star version 5.

Now moving onto software.

Leopard is best selling release ever, and was well received by critics. Apple took the obligatory swipe at Microsoft by noting that Vista has been a commercial failure. Windows 7 is another version of Vista and still deals with DLL, registry, disk defragmenter, and user account control.

Apple is very proud of Leopard, and the challenge of Snow Leopard is to make a better Leopard. This means refinements, powerful new technologies, and Exchange support. There are more than 1000 software projects in OS X, and for Snow Leopard Apple has updated 90 percent of them. Some examples presented were:

1) The Finder

The Finder has been rewritten in Cocoa. The user interface is effectively the same, but with the Cocoa rewrite you get a few new touches for free.

2) The Dock
Expose is now built into the dock

3) Installation is up to 45 percent faster. Snow Leopard occupies about 6 GB less disk space than Leopard, thanks to technologies like file system compression.

4) Snow Leopard Preview application
Performance is improved in general. Preview does a better job of cut and paste of PDF text through some artificial intelligence to better infer the document structure.

5) Chinese input method
Similar to the iPhone, you can now use the trackpad to enter characters with your finger.

6) Mail
Substantially faster by leveraging new Snow Leopard technologies.

7) Safari
Safari 4 is shipping today for Leopard, Tiger and Windows XP and Vista. 32-bit performance using the SunSpider JavaScript benchmark is up to 8 times that of IE 8. Safari is 100 percent Acid3 compliant. As a point of comparison IE 8 is only 21 percent compliant.

Safari 4 included on Snow Leopard with a couple of extra features. First is crash resistance to rogue browser plug-ins. This was cited as the number one case of Leopard hangs. Snow Leopard with simply display missing plug-in window if a plug-in crash occurs. Second is that you get 50 percent faster JavaScript performance in 64 bit.

8) QuickTimeX
This is a new implementation of QuickTime. It features modern foundations, hardware acceleration, ColorSync, and http streaming.

The UI of QuickTime X player is being modified to put the content at center stage. QuickTime X Player has built in sharing capability.

9) 64 bit
The primary advantage of 64 bit is access to lots of memory. For 32 bit the memory limit is 4 GB, while for 64 bit it is 16 billion GB. In addition, performance of 64 bit math is up to 2x faster. Snow Leopard has all major system components running in 64 bit.

10) Multicore
Multicore is the name of the game today. However, a big problem is how to handle threads. Apple's solution is called Grand Central Dispatch (GCD). It is composed of a language extension, multicore engine, a low level system-wide API, a higher level object oriented framework, and tools.

One example of GCD in action is threads in Mail. When Mail is busy, it generates and uses more threads that Leopard mail. When idle, it uses less threads than in Leopard. Leopard pretty much keeps the same number of threads active.

11) OpenCL
OpenCL is designed to allow developers to use the power of the graphics card beyond just OpenGL graphics. OpenCL provides hardware abstraction, and is based on the C-language. It

does automatic optimization, and has the numerical accuracy needed for scientific applications. OpenCL is an open standard, and all major graphics manufacturers are on board.

12) Exchange
Built-in support into Mail, iCal, and Address Book.
You get the normal OS X functions that you expect, like QuickLook, drag and drop, etc.
Full functionality requires Exchange 2007.

Snow Leopard will be available in September 2009. It will support all available Intel Macs, past and present. Pricing for Snow Leopard will 29 dollars for Leopard users, and there will be a family pack available for 49 dollars. Today, developers will get a near final version developer preview.

Now onto the iPhone OS.

To date there have been more than a million free downloads of the iPhone SDK. There are more than 50K apps on the App Store, and 40 million iPhone OS devices (iPhone and iPod Touch). The App Store reached 1 billion downloads on April 23, 2009.

iPhone OS 3.0 is a major update with a 100 new features. Some of the highlighted features were:

1) Cut, copy and paste
   Works across apps, undo support, developer API, Cocoa touch support for text

2) Landscape
   Enabled to all key apps, like mail notes, and message

3) Message
   MMS, send and receive photos, contacts, audio and location files, and text. 29 carriers in 76 countries with support MMS at launch AT&T support in the USA later this summer.

4) Search
   Calendar, Notes, Email, Messages, and Spotlight support

5) iTunes
   Rent and purchase movies, TV shows, audio books and music videos from the phone.
   Support iTunesU now in the phone

6) Parental controls
   Fine grain control for movies, TV shows, music on the app store.

7) Tethering
   Allows you to share the internet connection of phone with your computer. It works on Mac or PC, and over USB or Bluetooth. It is a seamless experience, with no special software needed on your computer. 22 carriers in 42 countries support at launch. Note there was no mention of AT&T.

8) Safari
   SunSpider JavaScript benchmark greatly improved. For the iPhone 3G with iPhone OS 2.2.1 it took 126 seconds. For iPhone 3G with iPhone OS 3, it took 43 seconds.

   http streaming audio and video. The system will automatically pick the best speed and quality for current connection.

Autofill—optionally remember username and passwords for favorite websites. Can utilize contact information to fill out webforms.

Increased HTML 5 support

9) Languages
Adding more supported languages such as Hebrew, Arabic, Greek, Korean and Thai. Now up to 30 languages in iPhone 3.0 both in portrait and landscape

10) Find My iPhone
Service available to MobileMe customers. Log into MobileMe and MobileMe will try to locate where is the phone (assuming the phone is on, and properly configured for this purpose). You can send the phone a message which plays even in silent mode, and you can send a remote wipe command if it is really lost or stolen.

iPhone OS 3 has over 1000 new APIs

1) In App purchasing
Conditions same as app store, and applies only to purchased apps. Free remains free.

2) Peer to Peer connectivity
Auto find over Bluetooth (no pairing needed)

3) Accessories
Opening up the iPhone to hardware accessory developers. You can talk via dock connectors or Bluetooth, and can use standard or custom protocols.

4) Maps
You can embed Google map service into your apps (including satellite and hybrid). You have access to pan and zoom, custom annotations, locations, geocoding. In addition, developers can come up with turn by turn directions using Core Location. This map serve is what Apple uses for Maps app.

5) Push Notification
Push text alerts, numerical badges, custom alert sounds

iPhone OS 3.0 is free for all iPhone customers, and works with all iPhones. For iPod Touch customers, it will cost $9.95, and works with all versions of the iPod Touch. iPhone OS 3.0 to be available worldwide June 17, 2009.

Paid developers get the GM seed today for download, and are being asked to do 2 things:
1) You need to connect to iTunes and assign parental control age rating.
2) You need to test, and submit update if needed.

The iPhone has raised the expectations of what customers can and want to do with the mobile phone. For example, mobile browser usage is driven mostly by the iPhone (65 percent). There are over 50,000 native apps for the iPhone. This is in contrast to 4900 for Android, 1088 for Nokia (all versions), 1030 for Blackberry and 18 for Palm (all data as of this past weekend).

The iPhone has won JD Power awards for business and consumer markets.

iPhone 3GS is new version where S stands for speed (marketing). The 3GS has the same physical design as previous model 3G. On average the 3GS performs tasks up to 2x faster than the 3G (both running iPhone OS 3.0).

The SunSpider JavaScript benchmark with the iPhone 3GS running iPhone OS 3.0 took 15 seconds. This is nearly 3x faster that the 3G which was nearly 3x that of the of the 3G on the iPhone OS 2.2.1.

iPhone 3GS uses Open GL/ES version 2, and is ready for 7.2 Mbps HSDPA.

The iPhone 3GS has a new camera. Currently iPhone is the leader in uploading photos per day to Flickr (about 3652 users per day), which is 3.5 times that of the nearest competitor. The new camera has 3 Mega pixels with autofocus, auto white balance, and auto exposure. It can take pictures and record video, and APIs are available for both modes. For pictures there is a feature called “Tap to Focus” so that you can bring into focus that element which is central to your photo. Video is recorded as 30 fps VGA with audio, and you can edit and share video with tap of finger.

Another new feature for iPhone 3GS is Voice Control to make phone calls, control the iPod features, etc. You activate it by holding down the home button.

The iPhone 3GS has a new built in digital compass. There is a new compass app with integration with maps, and allows you to orient your heading on a map with the compass. An API is provided for developers.

There are various accessibility enhancements (Voice Over, zoom, white on black, etc.), along with built in support for Nike Plus.

The iPhone 3GS has built in hardware encryption, which allows for nearly instantaneous remote wipe. Backups via iTunes are now encrypted as well.

Battery Life for the iPhone 3GS was touted as equal to or better than the iPhone 3G, and all the major environmental checkpoints were passed (e.g. arsenic free glass, etc.)

Pricing for the iPhone 3GS in the USA was $199 for 16 GB version, and $299 for the 32 GB version. The iPhone 3GS comes in black and white colors.

Apple also wants to make the iPhone even more affordable. So starting today an 8 GB version of the iPhone 3G is available for $99. The iPhone 3GS is available on June 19 in USA, Canada, France, Germany, Italy and Spain. It will launch in 80 countries over the next couple of months.

OTHER NOTES:

Below are some highlights from some other sessions I attended. As these sessions were not broadcast to the general public, I am keeping the discussion at a higher level.

Sessions Kickoff

About 60 percent of attendees are here for the first time. There are many developers who are more focused on the iPhone than on the Mac (and vice-versa). So this year Apple decided to start the afternoon sessions differently than in the past. So the traditional first post-keynote session (called Mac OS X State of the Union) has been broken into two pieces. First was the Sessions Kickoff followed by the Core OS State of the Union.

The sessions kickoff did two things. First was to highlight what is new and different with the iPhone and the Mac since last year. Second was to review the big software principles and constants that Apple uses.

What is new and different

Today (with the Mac, iPhone and iPod Touch) there are 70 million OS X devices in the market. The iPhone in has changed the landscape. There have been more than one billion downloads from the App Store. There is a diverse development community composed of traditional, garage, non-traditional consumers, IT groups, and game developers. The Mac is mainstream and is a good client that fits pretty much anywhere, and continues to grow in popularity.

Three new technologies were highlighted for the iPhone:
1) Location--GPS and magnetic compass

2) Peer to Peer---Bluetooth

3) Push Notification over the cell phone as a system level service

Three new technologies were highlighted for the Mac:
1) Leverage the silicon
   The system is 64 bit, which gives access to more memory, and greater security and performance. No execute heap is the default, stronger heap integrity checks. There are twice the number of registers on Intel. Harder to exploit ABI, as arguments are passed in registers versus the stack.

2) Grand Central Dispatch (GCD)
   A system level service which tries to exploit parallelism in software. Developers will need to learn the required language extension to C and Objective-C.

3) OpenCL
   To enable graphics cards for uses besides graphics. Currently this is where most of the transistors are.

The big constants over time and between the Mac and iPhone were put into 3 categories: structure, APIs and User Interface (UI).

Structure
OS X has some 532514 source files which amounts to about 36 GB in size. Software layering is very important, and Apple invokes several patterns to do so to avoid spaghetti code. This helps to be able to move code to different platforms and architectures, and to adopt new technologies.

Pattern 1—split the front end from the back end. An example is Safari, which separates the interface from the rendering engine (WebKit)

Pattern 2—Abstracting and factorizing to enable common functionality for multiple applications

Pattern 3—Quarantine legacy technology. You want to fence it off, but make sure it works.

APIs are the way to communicate between different layers. Use only be public and supported APIs. Use object orientation to help APIs fit together. Object orientation allows for auto behaviors. It promotes reuse and customization. It also captures intent and is very elegant.

APIs can have a long lifetime. Unix is 40 years, Cocoa is 20 years, and Open GL graphics around 15 years. That is why APIs start as private, then internal review, then external feedback, and finally commit to support.

There is lots of shared APIs between iPhone and the Mac. With Snow Leopard and iPhone 3.0, Core Data is now on the iPhone, and Core Location is now on Snow Leopard.

The UI is often the differentiator. Must be super functional, easy, intuitive and wow!

Core OS State of the Union

The Core OS is the foundation technology for OS X. It is built on the stable foundation of Unix, and is the engine of Mac OS X and the iPhone OS.

Unix is a time proven architecture. It is modular with well defined interfaces between subcomponents which allows for subsystems to take on new challenges. Unix is extensible, scalable, and well suited as a symmetric multiprocessing system.

Apple is a strong user of open source. Over 400 projects in Core OS are open source.

Unix is portable and that has enabled Apple to make the transition from PPC to Intel, and Mac to iPhone in a relatively straightforward manner. Applications are also portable via POSIX.

80 percent of iPhone APIs are shard with Mac OS X (Leopard and iPhone 2.2.x). 1400 more APIs with Snow Leopard and iPhone 3 are shared. 85 percent of Core OS code is shared with Mac OS X and the iPhone OS.

A few technology highlights were given comparing Snow Leopard to Leopard.

1) Time Machine or Time Capsule
   First time backups are 55 percent faster in Snow Leopard (holistic approach to achieve this improvement, so the whole system benefits)

2) Network file systems
   30 percent for AFP (AFPBench), 100 percent faster for NFS (SPECsfs2008) and 130 percent faster for CIFS (SPECsfs2008).

3) File copy (client performance)
   30 percent for AFP, 50 percent faster for NFS, and 60 percent faster for CIFS.

4) Client performance of directory enumeration
   5.6 times better for NFS and 5 times better for SMB

5) VPN and the phone
   VPN on demand--needs Cisco IPSec
   IT department gives you a cert. in your configuration profile

6) hotspots and the phone
   captive networks (e.g. hotspots in airports, hotels, McDonalds)

7) Bonjour and specifically the “Bonjour Sleep Proxy”

Say you have a machine to which your printer is attached, or are streaming media, or have a file you need to access using Back to my Mac. All works great until the machine goes to sleep. One way to solve this is to leave the machine on and awake all the time, but this is wasteful.

The solution is the "Bonjour Sleep Proxy" which utilizes your Airport Extreme (AE) which draws far less power. If the machine goes to sleep it sends its Bonjour service records to the AE (saying I have a printer). AE steals the machine's IP, and as far as the rest of the network sees it, the AE is the computer.

If you need to print (or do something else with the machine) AE sends the IP back to the machine and wakes it up, and now your printer is accessible. Applications will need to use Bonjour APIs to take advantage of this feature.

You can think of this as a combination of Bonjour, MobileMe and Time Capsule. This implies that you could get to any machine, from anywhere, with the machine in any state.

8) GCD

Managing threads is hard. The goals of GCD was to provide a multicore programming model. You break your application into blocks of work, and the OS gets the work done. GCD employs a new multicore engine which provides an ultrafast user space scheduler, optimal thread management, simplified event handling, and simple synchronization based on queues.

GCD is a highly tuned engine with low overhead and lockless scheduling. It utilizes wait free algorithms, and scales thread usage up or down depending on the needs of the application.

GCD is a comprehensive solution. In addition to the multicore engine, it invokes a C-language extension (called blocks), an object oriented framework, a new system wide API and tools. The object oriented framework is called NSOperation. NSOperation is now implemented on top of the new Dispatch framework in Snow Leopard.

There are many system wide APIs which are generating events as well. These event sources and integrated with applications events and processed via GCD.

The developer tools have been upgraded to deal with GCD. For example, Instruments has been extended to handle blocks and provide introspection to see what is going on.

9) 64-bit

Snow Leopard has both 32 and 64 bit applications on the same system, and all major system applications are 64-bit ready. The kernel is 64 bit ready in Snow Leopard.

Why does Apple need to move to 64-bit? As RAM gets larger and more affordable, it will not be long before 96 GB of RAM may not be unusual. It takes 64 bytes to describe each 4K page of RAM, so 1.5 GB virtual space for 96 GB ram. That is about one-third of the 32 bit-kernel virtual address space. This does not include the address space required fo things like VM data structures, process data structures, vnodes, file data you cached, graphics memory mapped, etc. In short, the kernel running out of address space.

The 64 bit kernel has same user kernel interfaces as the 32 bit kernel. This means apps are totally binary compatible with the 32 or 64 bit kernel. With the 64 bit kernel, a system call entry is 250 percent faster, and user/kernel memory copies are 70 percent faster.

64 bit runs by default on some Xserve configuration and Mac Pro configurations, but over time this will be true over a broader set of machines. So Apple kexts are being made 32/64 bit universal. But all kexts need to be 64 bit including third-party kexts.

In Apple's experience, many kexts just recompile, and most of the kext programming interfaces (KPI) are fundamentally unchanged. There are a few KPI removed, but these were already deprecated. A couple of KPI removed were very 32 bit specific, but their replacements are not word size specific. Once all kexts become 32/64 bit universal, the 64 bit story is complete.

So the call to action asked developers to adopt the Bonjour APIs, enable the use of GCD in their software, and to make their kexts compatible with the 64 bit kernel (K64).

Developer Tools State of the Union

The Xcode 3.2 tools suite is shipping and updates all major tools like Xcode, Interface Builder, Instruments, Dashcode, and the iPhone simulator.

For Xcode it was said that project find performance is up to 2.5x times faster, and you can now do full project renaming. With Interface builder you can copy and paste connections and bindings, edit classes in library, and there is support for new iPhone classes and functionalities (e.g. Map Kit and Accessibility).

The Xcode source editor has been refined in the areas of code completion, code folding, syntax coloring, and customizability. There is a new coding font called Menlo. Improvements have been made in the debugger bar and data tips, find and replace, message bubbles, Quick Help, and the Documentation window has been reworked.

Dashcode 3 can now create web applications for multiple deployment targets (Safari on Mac , Windows, and the iPhone). You can have CSS animations and effects. There is now improved support for the HTML 5 specification such as media tags and offline storage. Of particular interest is the new support for data sources and bindings (sort of like Cocoa bindings).

Xcode builds and merges multiple binaries. Applications typically contain multiple executables plus supporting resource files. Executables contain the machine code for the targeted platforms. However supporting resource files are often not duplicated, so total binary is often only 25 percent larger. Thus developers are able to distribute a single app which runs on all Intel and PowerPC Macs. For Snow Leopard, developers should start including an Intel 64 bit binary. The same technology applies for the iPhone OS. Developers should consider targeting both ARM v6 (iPhone, iPhone 3G and iPhone 3GS), and ARM v7 (iPhone 3GS).

The new default compiler is GCC 4.2. Apple added support for blocks (Snow Leopard) and for ARM v7 (iPhone OS 3). In addition, improvements have been made for stronger optimizations and autovectorization.

LLVM
GCC can be thought of as composed of a front end parser, and back end code generator. Last year Apple substituted the GCC back end with LLVM. This year we have Clang to take the place of the GCC parser front end for Objective-C and C.

Apple is moving to a new compiler technology which is Clang and LLVM. LLVM is an open source project, and Apple contributing for almost 4 years. LLVM is about performance, no baggage, and

has a library based design. LLVM library design goals were to be modular, have a small footprint, language independence, and be able to target multi platforms.

Items facilitated by LLVM include OpenCL and OpenGL. LLVM will compile and optimize code at runtime in your app (OpenCL) or produce runtime optimized algorithms (OpenGL). For OpenGL this has advantages over previous customized JIT (faster, flexible, support for new architectures).

Apple is finding LLVM generated code is typically 5 to 25 percent faster than GCC. Clang building is 2.1 times faster for many Apple apps, e.g. Xcode takes 7 minutes under GCC

LLVM and GCC compilers fully binary compatible. The best experience is Clang 1.0 for fastest build time and best code. However Clang still fairly new, so some niceties not yet implemented fully. In those cases LLVM GCC 4.2 is the most compatible, and generates the fastest code.

Applications built with Clang-LLVM which are shipping in Snow Leopard include the Property List Editor, Automator, Xcode, Interface Builder, Dashcode, and AppleScript editor.

Instrument is the flagship application for correlating data sources and performance. There are new instruments for Snow Leopard allow you to monitor GCD, threads, instant off, and zombies. Refinements have been made to many previous instruments. There is a new time profiler instrument which uses the core technology of Shark. So you can use Shark technology to collect data with low overhead, and other instruments to analyze data.

Xcode contains a new tool called the Static Analyzer. Apple mused that with it Xcode might find your bugs before you. The Static Analyzer is based on Clang, and dives deep into your Objective-C and C code. It tries to analyze all possible code paths within one file, while automatically checking for common coding errors. It works for both the Mac and the iPhone.

Errors include memory leaks, proper use of retain and release, over-rooted garbage collected objects, unused instance variables, uninitialized variables, never executed code paths, referencing null points, division by zero, dead stores, use of Foundation calling conventions, and more.

1000 bugs found and fixed by Apple on internal apps like iPhone OS 3, iLife and Snow Leopard over the last several months using the Static Analyzer.

Mac OS X Server State of the Union

Snow Leopard Server will be available in September along with the client. The price is $499 for unlimited clients with no client access fees.

The session began by outlining some improvements with the administration tools.

1) Installation and setup—no longer have to declare ahead of time size and purpose of the server installation.
2) Server Migration—server can migrate data from existing hardware to new server.
3) Airport Configuration—Snow Leopard server can manage your airport base station, so all services accessible from anywhere on internet.
4) Certificate Management into server preferences—integrated SSL certificate management.
5) Server administration for large sites—server admin can manage groups of servers based on any organization you like, and can manage dynamic groups of servers for updating criteria like server load.

With Snow Leopard, Apple aims to complete their transition to a 64 bit OS (client and server). All services were made 64 bit in order to improve available address space, system calls, computation and OS resources. For OS X Server running in 32 bit, you have a maximum of 2, 500 processes. When running in 64 bit, the process limit is essentially dynamic, with up to 30,000 processes with 96 GB and still with room to grow.

Snow Leopard aims to significantly improve OS X Servers multicore scalability. Leopard Server tends to saturate at 3 cores (per core saturation). Snow Leopard uses all available cores. Overall we see a factor of 2x performance boost. Apple touted they now have a highly scalable server from Mac mini to a rack of Xserves.

Some performance numbers cited were (relative to Leopard Server)
-- Web server performance is 1.8 times better
--NFS server 2x better (SPECsfs2008_nfs.v3)
--SMB server is 2.3 x better (specsfs2008_cifs)

Snow Leopard has a brand new mail server engine. Apple is now using the open source project called Dovecot (http://www.dovecot.org/). Dovecot is supposed to be 1.7x better than Sun Java messaging server (SPECmail_Ent2009), and 10x better than Leopard. Dovecot has self healing capability, built in clustering, and server side mail filtering/rules. Apple integrated it with other Apple services (Keychain, Admin tools, Xsan setup, Open Directory, Mobile Access Server, etc.).

iChat server received some updates. First some refinements were made to improve performance. Autobuddy lists were added, and they can be turned on or off on a per group basis. Support was added for AD native digest authentication and Kerberos multi realms.

iCal server is Apple's fully featured calendaring and scheduling solution, and is based on the open standard called CalDAV. It is high performance and has support for delegates. iCal server is completely deployed within Apple itself, so it is considered ready for real world use.

For Snow Leopard, some of the new stuff includes implicit scheduling (server managed instead of client based), push notifications, attendee lookups via CalDAV, and the ability to comment to organizer. Performance compared to Leopard improved around 7-10x in simulations.

There is a web based calendar client in Snow Leopard server. There is enhanced interoperability with non CalDAV systems such as Google calendar beta, Yahoo calendar, and Outlook.

There is a new server service in Snow Leopard called Address Book Server. It is based on the open standard called CardDAV, which was created by the same people who did CalDAV. CardDAV uses vcard as its format.

In Snow Leopard, Address Book client supports CardDAV. Looks and feels just like the way Address Book has always worked. Address Book has offline support and caches things locally. When it connects back to the server it syncs back up automatically. There is support for a Global Address Book via LDAP/Open Directory for Address Book server.

Within Apple, Leopard Wiki server now supports about 500 internal projects/wikis. The availability of this Wiki server was mostly passed by word of mouth. In Snow Leopard, you create a Wiki in three steps with a web based assistant: (1) give it a name, (2) pick a theme, and (3) decide who has access. There is a feature called My Page, which allows you to keep up with wiki projects you care about. Cross-wiki searching is now possible. There a new customization hooks available, and the wiki is optimized for the iPhone to make the iPhone a first class client.

One very nice feature addition to the Wiki is that QuickLook is enabled in the browser. Uploading documents was already easy. With Quicklook you can now view and read many documents without having to download the document and needing a local app installed.

Another new service in Snow Leopard Server is the Mobile Access Server. It is intended to allow remote access to specific types of services on your intranet, and not your intranet in general. It uses strong encryption and authorization, and works on any client OS. No special client software or tokens are needed. Targeted at small businesses who do not (or cannot afford to) deal with the complications and cost of VPN.

Podcast Producer has been updated to version 2. Podcast Producer is an end-to-end solution to create and deliver podcasts to users, and has an open and modular architecture.

Highlighting a few new features

1) Dual source video is supported. You can capture two sources simultaneously and generate picture-in-picture video. There are Apple designed layouts, and automatic transitions.

2) Podcast Composer
   There is a new visual workflow editor, and you can add custom artwork. There is now an option to deploy to server directly.

3) Podcast Library
   You can locally host iTunes U content, and have the option for automatic archiving. There is a built in Atom/RSS feed engine for publishing and delivery of content.

Snow Leopard technology underlies Podcast Producer 2. The new QuickTime X capture API is used for recording and encoding into different format. Quartz Composer can be used for fancy transitions, watermarking, etc. Core Animation is used to enhance user experience. There are many more examples.

Integrating iPhone into the Enterprise

The iPhone is getting traction with hospitals, media companies, and lawyers.

It integrates well with Exchange email, calendar, and contacts. It supports Exchange ActiveSync (Exchange Server 2003 SP2, Exchange Server 2007 SP1) for push email, calendar, contacts, global address list, autodiscovery (Exchange 2007), and remote wipe. The iPhone can be wiped remotely using the mobile administration web tool for Exchange 2003 and Exchange 2007, or the OWA or Exchange management console in Exchange 2007.

New in iPhone OS 3.0 for Exchange is support for certificate authentication (private key to certificate server), and enhanced policies (password and IT administration). Additional Exchange features supported are the search server in 2007, calendar invites, and push to subfolders.

The iPhone has standards based integration. There is IMAP and POP for email, CalDAV in iPhone OS 3.0 (for iCal server, Oracle Beehive, Zimbra, Kerio, etc.), and LDAP in iPhone OS 3 to have lookup integrated with mail, contacts, and SMS.

Network integration includes a built in vpn client (Cisco IPSec, L2TP/IPSec, PPTP) and various means of authentication (digital certificates, two factor tokens, passwords). For iPhone OS 3 there is proxy support for VPN, and the new VPN on-demand feature.

VPN on demand (works for any network based applications, including those from iTunes store) delivering a great user experience, and is seamless to the user. It uses certificate based authentication (no need for tokens), and runs in the background. It is based on defined domains, and is transparent to applications.

For WiFi security the iPhone has WPA/WPA2 enterprise for 802.1x authentication. Included is support for EAP (TLS, TTLS, FAST, SIM), LEAP, and PEAP v0 and v1.

iPhone OS 3 supports captive networks. It detects whether on network that cannot actually access internet, and brings up websheet sheet so that users can see info about that hotspot (hotels, coffee shops).

Integration in Practice of large numbers of iPhones

Since OS X was deployed internally, Apple has been able to adopt a much more self serve approach regarding internal IT support. This contrasts with the much more hands on approach when Apple was running Mac OS 9 internally.

A recent example was when Apple decided to give an iPhone to all their 14,000 employees. Apple IT got emails from Steve Jobs that basically said "I do not want to see any lines, and do it in one day. Do not care how you do it." Recall that the iPhone is brand new, most people at Apple had not seen it yet, the tools we have now did not yet exist, and not everybody is technically inclined.

Apple IT does not have staff to hand hold everyone, so they decided to just give people the phone and point them to an internal web site to help get them up and running. So employees swipe their badge to identify themselves, and get a phone and get a paper with the URL where iPhone help will be. No other questions allowed. The internal Help desk expected to be crushed, so vacations were cancelled and temporary help was setup. But it turns out that although the help desk had a spike in activity, it was well within their ability to handle it. This was considered both an indication of how intuitive the iPhone is to use, and the success of the employee self-serve model (although good and easy to understand documentation is a key component).

Internally Apple does not run Exchange Server, but uses standards based solutions (e.g. IMAP, CalDAV over SSL). For calendaring they have switched over to iCal. Apple is going to certificate based authentication which eliminated the need for tokens for the Mac, and is phasing out tokens for the iPhone.

Integration is only the beginning of the story of what is doing with the iPhone internally. Initially they just did the corporate directory. Then they added tools for dealing with approvals (business purchases, vacation). This was followed by setting up dashboards, to provide quick info for executive decisions. Now they are working on applications for retail business processes.

There are several considerations for deploying iPhones (or any mobile device for that matter). How to activate for cellular service? How to access corporate services? What device policies do you require, and how do you configure your devices to meet those policies?

Many key functions can be done via iTunes. This includes device activation, software updates, in-house app installs, sync media, and backup and restore. You can now do encrypted backups to the host computer with iPhone OS 3.0.

iTunes can be deployed in the enterprise and integrated with desktop management. It can be setup in activation only mode. You can disable network services (iTunes Store, shared media libraries), disable software updates, and disable media/data syncing.

The iPhone Configuration Utility (IPCU) can be used for making configuration profiles, and for provisioning the device. There are several new IPCU features for iPhone OS 3.0.

The IPCU is now at version 2. It is available for both Mac and Windows, and functions identically on either platform. There are additional settings for setting up restrictions and preventing the deletion of profiles. You can sign and encrypt profiles to help secure settings and policies.

Configuration profiles are XML files that contain device settings and policies. You can already configure settings for Exchange ActiveSync accounts, IMAP accounts, VPN settings, and WiFi settings.

For iPhone OS 3 there are additional settings that can be provisioned. These include CalDAV accounts, LDAP accounts and Web Clip installs (e.g. point to documentation, corporate web service). You can put restrictions on access to explicit sites, Safari, YouTube, the iTunes Store, installing apps, camera usage, passcode policies, and credentials that can be used.

Enhance passcode policies include requiring a passcode on the device, allowing simple values, require alphanumeric values, minimum length, minimum number of complex characters, maximum passcode age in days, auto lock in minutes, passcode history, grace period of device lock, and maximum number of failed attempts before the device auto wipes. Note that since shipping the iPhone, there was backup policy about passcodes. Enter the wrong one and it takes longer to be able to enter the next one. At seven attempts it can take up to 15 minutes, and beyond.

The discussion then moved to the use of certificates to enable secure authentication and communication. The goal is to authenticate to Exchange, VPN, WiFi, and other web services using SSL encrypted network communication. You could deploy certificate in a configuration profile, like a configuration profile via email or web, or (for iPhone OS 3) using SCEP (a new certificate delivery mechanism).

Cheryl Madson from Cisco gave a brief talk about SCEP. SCEP stands for Simple Certificate Enrollment Protocol. It is used for certificate enrollment, and as a lightweight mechanism for deployment of certificates (not a full PKI management solution). The protocol was originally developed by Verisign for Cisco, and SCEP is maintained by Cisco as an IETF internet draft. Cisco is looking at using SCEP as they begin to use the iPhone (Cisco IT iPhone Pilot in early phase and testing).

The original problem SCEP was to solve is how to supply certificates for routers running IPSec. A router is not a host with browser software, and a router does not have pre installed CA certificate or trust points. So how to obtain device certificates without pre-established trust relationships?

SCEP is defined on http, and the iPhone runs SCEP over https. The https session is secured by a trust point already known to the iPhone (preinstalled root), and this trust point is not the enterprise CA. The main purposes of all this is to authorizing user requests by establishing client credentials in order to obtain certificates which can be used to get access into an enterprise network.

So what Apple is trying to do with SCEP for the iPhone can be summarized as three steps.

1) User authentication
   Ensure that the user is authorized prior to proceeding with certificate enrollment.

2) Certificate enrollment
   Generates a standard certificate signing request which is also encrypted via the SCEP protocol. Receives the identity certificate in response.

3) Over-the-air device configuration (customer may need to build this part into their infrastructure)
   Delivers an encrypted configuration profile with the settings and credential for iPhone to connect to corporate services. Your Certificate (CA) service needs to support SCEP.

The final topic was regarding development in-house apps.

Apple has tried to provide a secure platform for applications. The iPhone uses signed and sandboxed applications to ensure app integrity, and restricts access to data of other apps. The security framework and keychain services of OS X are available, which safe stores credentials and provides certificate, key and encryption services. The iPhone 3GS has hardware encryption which is done automatically and is transparent to the user. This allow for nearly instantaneous remote wipe by throwing the encryption key away.

The iPhone is about mobility, so you want to tailor mobile apps for mobile workers. You need to define specific tasks for mobile situations since your users are on the move. You want maximum output with minimal input. You need a streamlined workflow which is optimized for ease of use and fast access. Finally you need to be integrated with your data and the network.

As an example of the mobile app for the mobile worker, Dean Moore (Senior Systems Manager and Architect for Sunbelt Rentals) was invited onstage to talk about their company iPhone app called Mobile Sales Pro.

Sunbelt is wholly owned subsidiary of Ashtead Group, PLC. They are the third largest equipment rental company in the USA, and second largest in the world if you include European operations. They have two iPhone developers.

Prior to having an iPhone with Mobile Sales Pro, the sales force was operating in the dark ages. The primary device used by sales force was a Sprint/Nextel two way radio. Communications to the field was done via conference calls. Customer data was printed quarterly as a book. There was no access to operational data while on sales calls, and limited access to new sales lead data.

So the goals were to improve communication within the company and move the sales force to a technology based system so they had live pertinent customer information at the point of sale. As a result they could integrate multiple disparate systems, and provide better customer service. To achieve this they used their existing Exchange system (mail, calendar, and contacts via GAL), put a nice rugged case on the iPhone, and wrote the custom app called Mobile Sales Pro.

Mobile Sales Pro is a complete mobile workflow with multiple components.

1) Planning
   Daily task planner with sales leads and the ability to share them

2) Preparation
   Customer account details such as accounts receivable, rental history and equipment on rent.

3) Fulfillment
   Real time needs analysis with equipment availability. Current rates and equipment location.

4) Service
   Email credit applications, view or email rental contracts and invoices, and request pickup.

The development process involved users early on to get feedback, and to define and agree upon the feature set and milestones. The developers would sometimes accompany the sales force to the field to get a direct feel for what the sales force needed.

Development started in summer of 2008, and by November 2008 had 10 users. They added users to about 80 until the big deployment in April/May of 2009 to 1300 users. Development and application acceptance was facilitated by developing and deploying in phases, and following the UI guidelines for consistency. They tried to program components in an architect once, leverage often way. The program has been very well received, and other groups like the service force wants to do something similar.

Safari and WebKit Overview

Safari is the web browser for OS X and is also available on Windows and iPhone Inside all versions of Safari is the same engine, known as WebKit. Safari's market share since 2005 went from less than 2 to just over 8 percent. This includes both iPhone and the desktop. The latest version of Safari is Safari 4 for the desktop and Safari for iPhone OS v3.

When focus on the mobile phone market, Safari currently accounts for 65 percent of the browser traffic. The next 14 percent are also WebKit based. All others make up the remaining 21 percent.

WebKit is the browser engine (especially for mobiles), and is an industry standard. Many desktop apps make significant use of WebKit. The obvious example are web browsers. Another set of examples are communication apps like iChat, Adium, and AIM, and mail apps like Mail and Entourage. You have development tools like iWeb, Dreamweaver, and Contribute. In addition, apps like Adobe AIR, Aperture, Dashboard, Xcode and Dictionary are users of WebKit.
There are over 100 applications using WebKit, and this does not count the iPhone apps yet.

More than 540 people have contributed to WebKit since 2005. About 75 percent of the contributions are from Apple, and the remaining 25 percent are from non-Apple folks.

Contributors do need to know what is where in WebKit code. At a high level, there are three layers to the WebKit.

1) At the low level is JavaScriptCore
   Contains nitro engine, JIT, VM and runtime data structures

2) At the mid-level is WebCore
   HTML, CSS, DOM SVG, XML, code for loading webpages, platform glue, and rendering engine

3) At the top level is WebKit (yes part of WebKit is also called WebKit)
   public API and platform glue

There was some discussion about new technology in Safari/WebKit.
Standards is how we innovate (W3C, WHATWG, IETF) so everyone benefits. Acid 2, which tests the trickier bits of the CSS standard, was passed first by Safari. Acid3 was first passed by Safari. Acid3 is a 100 different groups of tests which focus on technologies for AJAX and Web 2 apps.

WebKit's rendering performance continues to improve, and the current engine is named Nitro. Using Safari 3 with Leopard as the baseline (with the Sun Spider JavaScript benchmark), Safari 3.1 is 2.7x faster, Safari 4 Preview is 4.2x faster, Safari 4 public beta is 11x faster, and Safari 4 in 64 bit Snow Leopard is 19.8x faster.

Apple demoed Safari 3 on Snow Leopard (modified by Apple to run on Snow Leopard) for a fluid dynamics simulation with JavaScript and the HTML 5 canvas tag gives 7 fps. The same demo with Safari 4 nearly 60 fps

Additional HTML 5 technologies supported as offline applications (applications cache, storage applications data), media elements for audio and video (obviates the need for plug-ins), scalable graphics (web fonts and SVG), and CSS Effects (masks, gradients, reflections, animations).

Finally, it was said that the Snow Leopard demo of safe crashing of browser plug-ins requires running Safari in 64 bit on Snow Leopard.

Transitioning to QuickTime X

QuickTime X is a new technology for displaying time based media. It began on various Apple devices (iPhone, iPod Touch, iPod, AppleTV), and has now migrated to the desktop. QuickTime X was implemented independently of QuickTime 7, and provides significant improvements. You must be using QTKit to get playback advantages of QuickTime X.

There are some other QTKit changes in Snow Leopard. It is now possible to display one QuickTime movie object in multiple views or layers. Due to changes with QuickTime X, the attributes QTMovieCurrentSizeAttribute and QTMovieSizeDidChangeNotification have been deprecated. Their replacements are called QTMovieNaturalSizeAttribute (to get “authored” size) and QTMovieNaturalSizeDidChangeNotification.

In Leopard and earlier, QTKit sat on top of QuickTime 7. The QuickTime 7 stack still had links to older API’s like Sound Manager, QuickDraw, and Carbon. In Snow Leopard QTKit sits on top of both QuickTime 7 and QuickTime X.

QuickTime X is composed a new set of media Services (Apple did not go into specifics), and sits on top of a modern technology stack composed of CoreFoundation, Core Video, Core Audio, and Core Animation.

The QuickTime X media services are designed for extremely efficient playback (as originally for iPhone, iPod and AppleTV), multithreaded, 64 bit native, and support ColorSync throughout. handles modern linear media formats. Loosely speaking QuickTime X plays whatever can be played on Apple devices (e.g. H264, AAC audio. etc). QuickTime X will be learning about more formats over time. There is a very simple public api in Snow Leopard.

The QTKit capture API now uses QuickTime X to process captured movies, and captured movies are color tagged. The QTKit export API now uses QuickTime X to export to device formats.

Not all media files can be handled by QuickTime X. If a file is not suitable, then QTKit falls back to QuickTime 7. You cannot determine which media architecture is being used. When requesting open for playback, you are restricted to certain QTKit APIs that apply to playback only. QuickTime X movies open faster with less CPU time, and take less CPU time when playing.

In QuickTime 7, when you open a media file everything is richly represented internally as a movie In some cases, a lot of work up front is required to build the movie. The advantage of this is that all the required data for further media operations is prepared.

With QuickTime X you declare your intent, and Apple will tailor what they do to match your intention. You use the QTMovieOpenForPlaybackAttribute for this purpose. If you declare nothing, the default is to prepare for anything.

There are usage restrictions when opening media just for playback via QuickTime X. You want to avoid unreliable dependencies. You are not informed which path is being taken. This is an API restriction when opened for playback. There is no editing, saving, archiving, export, or access to QuickTime 7 primitives like Movie, Track, etc, and the restrictions pertain regardless of path taken.

Using the QTMovieOpenForPlaybackAttribute to access QuickTime X brings additional benefits. These include 64 bit native, taking advantage of multiple cores, improved color fidelity, http live streaming, etc. In addition various recent models support H.264 decode in hardware (NVidia 9400M chipsets). You get hardware acceleration if it is not busy, but be aware that it supports a very limited number of simultaneous decoding sessions

For multithreaded apps, developers should consult TechNote 2125 (thread safe programming in QuickTime). You initialize QuickTime movies on the main thread, and use the QTMovieOpenAsyncRequiredAttribute to improve responsiveness by migrating them if necessary to other threads for processing. Success depends on thread safety of underlying decoders, etc, so you should have a fallback ready. Only QuickTime movies attached to the main thread can play.

QuickTime 7 contains a wide variety of components created by Apple and third parties. These include image encoders/decoders, audio encode/decode, movie importers, and movie exporters. Of historical interest is that when QuickTime 1 was introduced in 1991, it only included capabilities for image encode and decode. The intention is to make QuickTime X extensible and secure, but Apple is not ready for plug-in modules with Snow Leopard. For now, you should use the QuickTime 7 methods for now.

So for QuickTime 7 components thread safety is a must from now on. Attempting to display the UI on a background thread can cause crashes, so do not do it. For importers to be thread safe, media handlers and decoders must also be thread safe. Sound manager components are never thread safe. If something is not thread safe, return componentNotThreadSafeErr (-2098).

Compiler State of the Union

Compiler introduction and landscape
You can split the tools Apple ships into 4 basic buckets.

1) iPhone OS 2.x you use GCC 4.0
2) iPhone OS 3.x you use GCC 4.2
3) OS X Leopard you can use GCC 4, GCC 4.2 or LLVM-GCC 4.2
4) OS X Snow Leopard you can use GCC 4.0, GCC 4.2, LLVM-GCC 4.2, or Clang

For GCC 4.2 Apple added support for Blocks (used in GCC 4.2, LLVM, and Clang), 64 bit default builds, and bug fixes.

LLVM is a new compiler strategy for Apple. Currently it complements GCC, but will eventually replace GCC. LLVM is an open source project. While GCC has done well over the years, Apple is choosing not to extend it due to several issues. One is that it does not get the most out of the hardware. Second it does not provide a great developer experience. Third, it cannot be reused to build new applications.

LLVM was intended to radically rethink how compilers are built and used. The idea was to build the compiler as a set of reusable libraries, which can support new applications with shared components. Examples of clients supported by LLVM and Clang are OpenGL, OpenCL, Xcode Static Analyzer, LLVM -GCC and the Clang compiler. The focus of LLVM is to produce fast compiles and great code generation, while using modern techniques and algorithms.

LLVM at Apple has proceeded in stages. The GCC compiler has three pieces. The GCC parser (the front end), and the GCC optimizer and GCC code generator (which together are the back end). Last year, Apple encouraged developers to start using LLVM GCC 4.2. LLVM GCC 4.2 uses the GCC front end, but replaces the back end with LLVM. Today Apple is announcing that developers can now use Clang to replace the GCC front end.

Traditional static compiler relies on separate helpers apps to build executables like the assembler and linker. The “new” part is to take the optimizer and code generator and use as a JIT. OpenGL and OpenCL were cited as examples in Snow Leopard of this “new” method. Other LLVM-GCC built Snow Leopard projects include Core Data, OpenSSL, and Java HotSpot.

For OpenGL, you can think of OpenGL being the front end, and the LLVM optimizer and code generator being the JIT compiler. In Leopard and earlier a very architecture customized JIT was used. But this made it difficult to easily support new hardware architectures. Using traditional interpreters provided flexibility, but were slow.

For OpenCL you have OpenCL kernels parsed by the Clang front end and the LLVM back end. Apple has noticed about a 25 percent speed up versus using the traditional GCC compiler.

LLVM-GCC 4.2 goals were two-fold. First is to be 100 percent source and binary compatible with GCC. Second is to have feature parity, faster compiles, and generate fast (or faster) code. New LLVM –GCC 4.2 feature this year include stack canaries, blocks, OpenMP, and improved debugging. LLVM-GCC 4.2 now supports all major GCC features.

There were some benchmarks comparisons shown between GCC and LLVM-GCC.

Compile time performance when building AppKit which is a large Objective-C framework:
LLVM is 35 percent faster than GCC in release mode.
LLVM is 5 percent faster than GCC in debug mode.

LLVM code generation performance tests focusing on Intel 32 bit versus Intel 64 bit.

OpenSSL 32 bit (for various ciphers)
For MD2, LLVM is 5 percent faster.
For MD5 LLVM is 10 percent faster.
For DES-CBC LLVM is 228 percent faster.

For OpenSSL in 64 bit the difference overall is not as dramatic. LLVM is up to about 20 percent faster on average.

LLVM code generation performance with Link time Optimization (LTO) turned on. LTO performs aggressive optimization across a file.

64 bit OpenSSL with LTO (performance relative to GCC 4.2 -03)
For MD5 LLVM increases to 20-25 percent faster.
For SHA1 LLVM goes from essentially equal to 9 percent faster.

For JPEG decoding time with LTO, LLVM increases from 1 percent to 8 percent faster.

You can enable link time optimization in Xcode by setting a preference. If you are using makefiles, you can use the command line sequence (/Developer/usr/bin/llvm-GCC -03 –flto foo.c –o foo.exe) or in the command line use the -04 switch (/Developer/usr/bin/llvm-GCC -04 foo.c –o foo.exe).

The discussion now shifted to talking more about the open source project called Clang. Clang is a set of libraries supporting both C and Objective-C. It adopted the LLVM design approach. The Clang compiler consists of Clang as the front end, and LLVM as the back end. Clang and LLVM are the Apple compiler strategy. In Xcode, If you have C++ or Objective-C++ code, the Clang compiler falls back to GCC-LLVM automatically. Clang is command line compatible with GCC 4.2.

Clang C++ support is a critical for Apple. Apple is committed to implementing it, and work is underway. Little factoid is that Clang and LLVM are written in C++ so there is some incentive. However, Clang 1.0 has no C++ support.

Apple has tried to make Clang as source compatible with GCC as possible in order to make the transition easy. You are fully binary compatible with GCC and LLVM-GCC. Clang implements GCC extensions, weird GCC quirks, and even some GCC bugs.

As of right now, Clang is missing some specialized and obscure GCC features like OpenMP, nested functions, _label_, etc. If you need these, use LLVM-GCC.

Clang is extreme compatible with Objective-C code. The one caveat is that Apple intentionally broke some specific Objective-C idioms that are incorrect and already deprecated constructs. If you depend on these, you need to fix you code. We do not plan to add them, and the Static Analyzer will not recognize these idioms. The unsupported Objective-C idioms are: cast of super, cast of L-Value, and sizeof Objective-C interfaces (64 bit specific).

Clang defaults to C99 versus GCC defaulting to C89. C99 is a mostly transparent extension to C89, but watch out for link failure due to “inline” differences. If you really want C89, you can set a preference in Xcode.

Compared to GCC, Clang compiler speedup in debug mode is 2 to 3x faster. A full build in debug mode is about 2x faster. Clang is faster than GCC for many reasons having to do with better algorithms, less memory, and engineering design decisions. Two specific examples given which contribute to the improved speed were given. First is the separation of front end and back end, and second was precompiled headers (PCH) and memory usage.

GCC coevolved its C front end with its back end. As a result, over time it developed many tangled interconnections and interdependencies. GCC wastes a lot of time and memory on code generation of headers and declarations that are never actually used in the .m file. LLVM evolved independently of Clang, and the Clang front end purposely kept independent of LLVM back end.

For precompiled headers (PCH) and memory use there were several benchmarks cited. First is that for PCH generation, Clang uses 2x less memory. Second is that using PCH requires 4x less memory with Clang. Third is the PCH file is up to 7x smaller with Clang.

Clang has some other goodies as well. An effort was made to improve error and warnings diagnostics. The compiler detects something is wrong, tries to guess why, and explain it. GCC diagnostics are often not helpful. GCC diagnostics are confusing, poorly worded, and not precise.

Clang is built on the stable and well test LLVM code generator, and is designed to err on the side of rejecting code, not accepting and breaking it. Apple has been working on it for several years. In addition there has been broad testing on all kinds of crazy code:

1) wide range of internal Apple code.
2) countless open source Unix and Mac apps, the Linux kernel, etc.
3) FreeBSD kernel and much of the entire user space.
4) Clang built apps in Snow Leopard that will be shipping in the final product.
5) Xcode (really two apps called Xcode and Xcodebuild).

Xcode is composed of 17 Xcode projects using 18 frameworks and 20 plug-ins. It has a total of 2.5 million lines of code, of which 5% is Objective-C++

Finally, mention was made of the new Clang Static Analyzer which is integrated into Xcode. The Static Analyzer works for both for the Mac and iPhone

At this time the choice of compiler to use on the Mac depends on your situation, what your code dependencies are, etc.

1) Clang
   Gives you fastest compiles and generates fast code. It provides a better developer user experience (e.g. error messages and diagnostics)

2) LLVM-GCC 4.2
   Gives you fast optimized compiles, and generates fast code. It provides full GCC compatibility.

3) GCC 4.2
   This is the current stable compiler for Leopard, and includes support for blocks.

4) GCC 4.0
   Compatible with the Tiger SDK. Otherwise it is time to move forward to at least GCC 4.2.

In-house App Development for iPhone

Mobile is a brave new world. Compared to the traditional desktop (or even laptop) we are in alien territory. We have limited power, smaller keyboard, and a single user system. There are more variables with location and network connections being highly dynamic. In the mobile world time passes slowly, or it certainly seems to for users. What may seen sufficient in the office can feel very slow when your physical surroundings are dynamic.

Users have great expectation. They expect your app to be available anywhere and anytime. Apps are expected to be reliable and trustworthy without users having to do anything special. Finally users expect to be able to use the device securely, without the pain of authentication.

The point is that you should not think of your iPhone apps as miniversions of you desktop app. Think of them as safe and legal performance enhancers. You want to make users better at what they want to do, rather than make them do more things. Users want to feel like they have a million answers in their pocket, at all times, no matter where they are located.

Apple listed 7 habits of highly effective enterprise applications. They should be fast, have familiar design and UI patterns, be formatted for the device, focused, have foresight (certain levels of appropriate integration), forgiving (for user accidents), and fun where appropriate.

With iPhone 1.x, you were restricted to web-based apps. The iPhone OS 2.x introduced the SDK plus expanded web capabilities. In iPhone OS 3.x you have further enhancements to the SDK (e.g. Push Notifications) and to web services (HTML 5, location services, etc). The hidden platform is the hybrid app, which Apple described as a web portal within a native app structure.

Some suggested best practices for architecture and development were given:

1) The iPhone is not a desktop
   The iPhone is a personal device with a different security model. Each app is sandboxed and only one app runs at a time. Network performance is more variable and processing power more limited.

   Thus do not try to directly port desktop apps, but try to overcome the delta between Cocoa Touch and the desktop. You should aim to build reusable statically linked libraries (instead of dynamic that is on the desktop), and consider wrapping LDAP with web services.

2) Build on existing infrastructure
   Leverage existing services and business processes. Employ stateless SOA architectures for scalability. Create RESTful wrappers for SOAP/web services.

3) Engineer for performance
   You need to minimize service invocations, and construct mobile optimized wrappers. Test apps with 3G/WiFi turned off, and take advantage of caching mechanisms. Always try to shrink response times. Capture timings in the client and server, and tune client and server as needed. Reduce the size and number of resources used or needed as appropriate.

4) Test on the device
   Use the simulator to rapidly debug your app. But be aware of differences between the simulator and the real device. This would include things like the security framework, network performance, and processing power.

5) Architect for security
   Always use SSL. Implement secondary encryption on passwords, and use the keychain to persist sensitive data.

6) Create a hybrid portal platform (SDK plus Safari)--web app in a hybrid shell
   Implement authentication centrally. Create stubs to launch web apps within the portal. Access native APIs from JavaScript, and blur the lines between native and web apps. Secure cloud architecture and secure external data access via SSL. No VPN token needed.

   Leverage URL schemes across applications. Launch your app from any app in iPhone OS. Enable inter app messaging, but use with caution to prevent buffer overflows.

7) Customize you deployment plan
   Create a UPP (Universal Provisioning Profile) on the program portal, and use the UPP to sign your apps with Xcode. Create a website to host the UPP and apps. Install apps with iTunes or IPCU.

What's New in Directory Services

This session talked about what is new in Snow Leopard (client and server).

On the client side, there has been major effort put in regarding code cleanup, efficiency and performance, and bug fixes.

Attempts were made to simplify the UI for binding in System Preferences. Basic Directory Utility functionality moved into System Preferences. Go to Accounts and look under login options. There is button for Directory Utility which is in System/Library/Core Services if you need direct access to the app.

Under the hood Directory Services adopted GCD (Grand Central Dispatch), improved caching, etc. DNS lookups now handled by mDNSResponder, instead of Directory Services.

The DS API (Directory Service API) has been deprecated started with Snow Leopard. It is still there, but developers are asked to move to the new OpenDirectory Framework.

Active Directory plug-in was enhanced to include bug fixes, better support for common configurations, support for Windows Server 2008, improved stability, and better mobility behavior. Some of the AD plugin changes include:

1) Now have password expiration notices at login
2) Support for disjointed AD namespaces (DNS name for hostname does not match DNS of AD domain)
3) Improved workgroup manager support. For example editing ad now supports editing groups, computer lists, existing ad records
4) No querying for macAddress (attribute) if its not part of the computer object class
5) Better differentiate between users and contacts records
6) Support for IPv6 AD domains
7) Support for 3 level+ domains (hierarchical)
8) Better kerberos support for win2k8 domains

Some additions were made for supporting AD users needing to access OS X services.

1) Augment records no longer needed (or OS X iCal or Wiki users---but idea still works. The ServicesLocator attribute requirement has been removed.
2) Added AD Digest authentication for some services like Podcast Producer and iChat server.

Now moving to the server side. For Open Directory Server (not limited by no new features rule for Snow Leopard!), a few changes were made with the view always on performance, scalability, and Interoperability.

The Server Setup Assistant no longer talks about options like Standard, Workgroup, Advanced, and Xsan. Now it gives setup options in terms of how to manage users and groups on your server.

Service Access Control Lists (SACLS) are already turned on automatically. SACLS are prominent in the menu bar of Workgroup Manager. SACLS are turned on by default except when the server is manually configured. Users are automatically added to SACLS in Server Preferences, however you must manually add users to SACLS if they are created in Workgroup Manager. If you use custom configurations, we now give option to opt in or out of turning on SACLS. You can turn them on or off as desired.

There has been a change in regard to administration passwords between Leopard and Snow Leopard. In Leopard, the server setup admin account was stored in the Open Directory node. The locadmin was stored in the local DS node. The problem is if something went wrong with your Open Directory master, you lost your main admin account. So it has been reversed for Snow

Leopard. The server setup admin account is stored in the local DS node. The backup directory administrator (called diradmin) with the same password is stored in the Open Directory node.

There were improvements made to LDAP
1) Upgraded to OpenLDAP 2.4.11
2) 2x read performance improvement
3) syncrepl for LDAP replication (no longer slurpd)
4) Removed the 1000 connections limit (this was artificially hardcoded). Apple has tested Snow Leopard with over 8000 connections. The system did well, and probably could handle more. But your mileage may vary. You are basically limited by number of open files allowed, and the rule of thumb is that one connection equals one file. The slapd connection limit is configurable in slapd launchd plist. The default is 8192, but you can change it.

There were improvements made to LDAP database reliability:
1) Updated version of Berkeley DB (BDB)--Sleepycat Software
2) Tuned database parameters
3) f_fullsync is now being used

You can turn fullsyncs on or off by issuing the command “slapconfig -setfullsyncmode yes!no”.
yes = fullsync on, which is normal operations
no = fullsync off, which is suggested during large imports

A few comments regarding f_fullsync were made about reliability versus performance. f_fullsync is great for database reliability, but has the expected slower writes. Fortunately, LDAP operations are mostly reads. Apple does not recommend running with fullsyncs off except when doing large imports. There is little performance benefit, and you put data at risk if off.

PasswordServer has some new features. There is a password synchronization plug-in which can sync to other directory servers. You can create custom password filters (Apple does have a basis set). Password filter hooks provided by Apple. You write a script or binary as a tool, and install it in /Library/Password Server filter.

The Radius Server has been upgraded to FreeRadius 2.1.3. New is authentication for AD users through the Radius server. No special configuration necessary if you are bound to AD. Radius server will autoconfigure to add AD capability.

There is support for IP/hostname changes. Server-wide improvements have been made for changing hostname or IP address. The new tool for changing directory data is called: changeDirData.pl (this is a Perl script).

Finally there were four best practices and tips that were given.
1) Extending the AD schema with OD attributes.

This is a new way to extend the AD schema with OD attributes using Microsoft’s own tools. Use ADAM tools (Active Directory Application Mode) in Windows Server 2003 R2, or the Active Directory Lightweight Directory Services in Windows Server 2008.

Apple has a whitepaper about this:
http://www.seminars.apple.com/seminarsonline/modifying/apple

Microsoft has a link to the ADAM tools here:
http://www.microsoft.com/windowsserver2003/adam/default.mspx

2) Open Directory topology

This is aimed at dealing with slow links in Open Directory replication. A single master can have 32 replicas, and each of the replicas can also have 32 replicas.

Open Directory replication is serial. One slow replication link slows down replication at that level. Thus you should connect replicas with slow connections to a relay (replica), not the OD master.

3) Magic Triangle setup

The Magic triangle describes a setup in which Open Directory (OD) clients are bound to both an Open Directory Master and another Directory Service node (usually Active Directory). The Open Directory master is bound to itself and to Active Directory (AD).

For a Magic Triangle setup, the order of setup is important. First verify you DNS configuration by making sure the forward and reverse lookup for the server hostname is working properly. Then bind the OD clients to AD (or other directory) first, and kerberize services against AD (or other directory). Setup the OD master last.

For the OD master, the Server admin status will show Kerberos stopped. This is normal and refers to the fact that you are not running the KDC on the OD master. Be sure no to duplicate users name, especially admin users.

4) Changing your IP and hostnames on an Open Directory master
Start by making the change in System Preferences. Then verify your DNS configuration. Then run changeip, and then reboot. After the server comes back up, run slapconfig -kerberize if on an Open Directory master. Finally use changedirdata.pl (Perl script)>

What's New in Core Data

Core data on now on the iPhone. It is basically the same as the desktop, save for specific things like Spotlight for the desktop only, or a fetch method for iPhone only.

Documentation starting point
http://developer.apple.com/documentation/Cocoa/Reference/CoreData_ObjC/index.html

Lightweight migration is a new feature in Snow Leopard to streamline the process of moving your application schema to a new application model. No data model ever survives contact with reality. Typically you start with what you thought was the ideal model, but performance issues, user feedback, or feature requests necessitate a change.

In Leopard, schema migration required you to build a mapping model to migrate to new schema. The mapping model needs to be compatible with the old model in order to access the data. You send mapping model to the Migration Manager along with the old data store, and out comes a new data store. For Leopard there was an Xcode assistant to create mapping models. However it often requires hand-tuning, and custom code for complex transformation. All of this is described in the Core Data Model Versioning and Data Migration Programming Guide.

Incrementally changing your data model is considered part of the natural development cycle. So for Snow Leopard, Apple is introducing Lightweight Migration. It is a new automatic migration feature to facilitate simple data migration, which is easy to setup and require zero code.

Lightweight Migration automatically infers the mapping model. It only handles unambiguous changes, but that covers a lot of ground.

In order to use Lightweight Migration, there are 2 things you need to do. First is to keep your old model. Do not directly change your old model. In Xcode, go to the Design menu, then select Data Model, and then Add Model Version. Second is to enable migration options to infer mapping model automatically, and to migrate persistent store automatically.

Also in the Attribute Configuration Inspector of the Xcode Model Design tool, you should set the Renaming Identifier to the old model name.

Lightweight Migration is fast and efficient, and exerts minimal memory pressure. Due to being unambiguous, it can use the SQLite stores to do the migration directly in the database. It uses SQL to alter tables and update rows. Debugging Lightweight migration is only on the Mac OS, and you can set the Desktop user-defaults to enable logging for both the migration and SQL logs.

So in summary Lightweight Migration is for simple migration of simple changes. It is easy to setup, very fast, and uses minimal memory which makes it good for the iPhone. It plays nice with custom mappings, and developers can provide custom mapping model for complex changes.

The discussion now moved to Spotlight search integration (only on Mac OS).

Spotlight search already exists to search for Core Data documents since the initial release with OS X Tiger. Apple provided a document-based app Spotlight importer, and a project template.

However in the case of non-documents, all records are in a single file. You want to be able to search for records, and get at the information in that record. That is was the new Spotlight search integration (via a new project template) is trying to address. It provides a better use experience for non-documents apps, and provides a way to integrate Spotlight into your app in order to better find and open individual records. Apple takes care of maintaining the record files, while you customize the importer and display. This is documented in the Core Data Spotlight Integration Programming Guide.

The discussion now shifted to Fetch performance.

The general fundamentals of fetch performance is to only get what you need, and to decide how much you really need to display.

The is an assortment of existing Fetch options to deal with large data sets. You can fetch less data, managed objects, Object ID's, or just get a count of the number of records.

But what to do when you have apps or a UI that really wants all the data in a single array.
Access to records can be random, and many records may not be displayed at all.
Many records not displayed at alland you can be using way more memory than needed. If you import an array of faults, then you run the risk of excessive fetches to the database.

To address this there is a new capability in Snow Leopard which was described as enabling you to "sip from the fire hose". It provides random access to the entire data set, limits memory use, and amortizes I/O. It can vend large data sets, handle batches and partial faults, and bypass unsaved changes.

Now the discussion moved to talking about Grand Central Dispatch (GCD).

GCD is only on the Mac and is a set of system level frameworks for making concurrent programming easier (NSOperation, libdispatch). Serial or concurrent queues are available, and you submit blocks for execution.

There is only one Core Data Approved technique (because this stuff is tricky), and it is called Thread confinement. Each thread gets its own MOC (managed object context). In GCD this means that anything that might execute concurrently gets its own MOC. You pass object IDs between threads to communicate between them.

For debugging threading issues, you will need the debug version of Core Data framework. This should be downloadable from ADC shortly after Snow Leopard goes GM.

A comment was made if you use the Instant Off feature (new for snow leopard). The hasChanges attribute is now observable, and you should not change the object graph. Apple makes no guarantees if you do.

Programming with Blocks and Grand Central Dispatch

(Documentation--Concurrency Programming Guide at http://developer.apple.com)

Blocks is Apple's name for closures. Blocks (GCC 4.2, LLVM, and Clang) allow you to pass code and data as an expression. The notion of closures are not new. More interpretive languages going back to the 1970's such as LISP, SmallTalk, and Ruby on Rails have had this notion. Apple uses the carat symbol in a function pointer like declaration for blocks.

Blocks is a C language extension and is language interoperable. This means that Objective-CC, C++, and Objective-C++ programmers get to use the feature.

There are a couple of Objective-C specializations. All Objective-C blocks are also always Objective-C objects. As such, they respond to the copy and release messages, and can participate in the add property syntax using the copy attribute.

Blocks are a far better call back API. Over 100 Snow Leopard APIs use blocks.

In Snow Leopard what is really new and interesting is using blocks for asynchronous (nonblocking, multithreaded) uses. This lets you think about concurrent computation on independent data, serialized computation on shared data, and all combinations of the above. That is what GCD is all about.

GCD is all about asynchronous blocks. Apple on the Mac platform is now 100 percent multicore. So we want to leverage the silicon, and reduce the appearance of the spinning beachball. The spinning beachball is the most obvious sign for opportunities to exploit concurrency, you can often use asynchronous design to update the GUI and get rid of spinning beachballs.

GCD sits right on top of the kernel. For some APIs, you can call GCD which can bypass the kernel unless needed. In some cases, bypassing the kernel can be up to 200x faster on 16 virtual cores on a Mac Pro.

GCD is part of libSystem.dylib and is available to all applications (#include dispatch/dispatch.h).

Each GCD object has some polymorphic API. There is reference counted memory management (dispatch_retain and dispatch_release), although no garbage collection at this time. GCD does the right thing by retaining parameters that are passed to it when needed.

There are various types of queues available.

Global Queues are a lightweight list of blocks, and the queue are executed asynchronously. Enqueue and dequeue is FIFO (first-in, first-out). You can have concurrent execution of blocks.

Dispatch main queue executes blocks serially on main thread. It cooperates with the Cocoa main run loop (NSApplicationMain()). It is usable by pure GCD programs (dispatch_main()).

There are Serial queues. Like other queues it is a lightweight list of blocks, and the queue can be executed asynchronously. Enqueue and dequeue is FIFO, and blocks are completed serially.

You can have multiple serial queues. Queues are lightweight, and there are no limits other than no imposed limits other than available memory. It is natural to assign one queue per task to take advantage of the potential for inter task concurrency.

You can create groupings of multiple blocks. You can track multiple blocks as a group (empty group notification). Blocks may run on different queues, and blocks may add more blocks to the group (recursive decomposition).

The Instruments developer tool has added support for blocks. You can track block behavior. This includes latency, track which blocks are enqueued on which queues, and which blocks are executed synchronously. You can also monitor optimizations which include monitoring the longest executing blocks, and the most executing blocks.

There were some final comments. Blocks only available on Snow Leopard at this time. You cannot separate blocks and GCD yet. So no blocks on Leopard without GCD. C++ lambdas cannot be used as callbacks without cheating, so you cannot do GCD with lambdas. Block objects work fine under GC. Part of the garbage collector is being implemented using GCD as well.

Introduction to OpenCL

OpenCL is a compute model and framework. With OpenCL you can
1) leverage CPU and GPU to accelerate parallel computation
2) get dramatic speedups for computationally intensive applications
3) write accelerated portable code across different devices and architectures

OpenCL is designed to give access to all the computing resources in the system. It is a low level framework for high performance, and targets parallel and computationally intensive algorithms (data parallelism in particular). Examples would be physics simulation, image processing, signal processing, video and audio encoding, medical imaging, and financial modeling. All core functions of OpenCL supported by currently shipping video cards in Apple machines, except for a few special extensions. Apple trying to minimize use of extensions. OpenCL is separate from GCD (Grand Central Dispatch), so developers can use either/or, or both.

Loosely speaking, data parallel computing means that you have similar and independent computations across all data. An example given is the box filter for averaging an image. You have the same computation for each output, and all results are independent. So if you have an image that is 1024 pixels equals, at one kernel execution per pixel you have 1048576 executions.

OpenCL computation model is centered on two things. First, kernels are executed across a global domain of work-items (a single kernel execution). Global dimensions define the range of computation (e.g. 1024 x 1024), and there is one work-item per computation, executed in parallel (1024x1024 = 1,048,576). Second is that work-tems are grouped in local workgroups. Local dimensions define the size of the workgroups (e.g. 128 x 128 or 16 x 16). Workgroups are executed together on one processor. Workgroups share local memory and synchronization.

Caveats to the computation model. First is that global work items must be independent. There is no global synchronization. Second is that synchronization can be done within a workgroup. You choose the global and local dimensions that are best for your algorithm.

The OpenCL memory model has several levels.

1) Memory on the machine you are using is called host memory.
2) Memory on the compute device (could be GPU or another CPU) is called the global/constant memory. This memory is not synced with host memory, and is shared by the workgroups.
3) Within a workgroup there is shared local memory (16 KB but is very fast)
4) Finally these is private memory which is per work item.

Memory management is explicit. It is up to you to move data from host to global to local and back.

OpenCL object and type definitions

1) For setting up OpenCL you have:
   Devices which are typically the GPU or CPU.
   Context s which are a collection of devices
   Queues which submit work to the device
2) For memory you have:
   Buffers which are blocks of memory
   Images which are 2D or 3D formatted images
3) For executing things:
   Programs which are collections of kernels
   Kernels which are arguments/execution instances
4) For doing synchronization/profiling :
   You have Events

The basic OpenCL setup procedure is to get the device(s), create a context, and then create command queue(s)

For devices all CPU cores together are a single device. OpenCL executes kernels across all cores in a data parallel manner. Contexts enable sharing of memory between devices. To share between CPU and GPU, both must be in the same context. Queues are the means for submitting work, so each device must have a queue.

Choose the devices depends on your algorithm and hardware available. OpenCL has query methods (e.g. clGetDeviceInfo) to find out what is available.

Memory buffers and simple chunks of memory. Kernels can access however they like whether it be by array, pointers, or structs. Kernels can read and write buffers. Images are opaque 2D or 3D formatted data structures. Kernel can only access via read_image() and write_image(). Each image can be read or written in a kernel, but not both. Reading and writing memory objects is

done by explicit commands to access memory objects. They can operate synchronously or asynchronously.

Compiling kernels requires 3 basic steps.

1. Create a program
   The input is string (source code) or precompiled binary. This is analogous to a dynamic library which is a collection of kernels.
2. Compile the program
   You specify the devices for which kernels should be compiled, pass in compiler flags, and check for compilation/build error
3. Create the kernels
   Compilation returns a kernel object used to hold arguments for a given execution.

To execute kernels you need to set the kernel arguments, and then enqueue the kernel. Note that your kernel is executed asynchronously.

Some hints for performance and debugging.

1) Performance: Overhead
   Compiling programs is very expensive, so reuse programs or precompile binaries. Moving data to or from the GPU is very expensive, so keep data on the GPU. Starting a kernel is expensive, so do a lot of work for each execution. Events on the GPU are expensive, so only use events where needed

2) Performance: kernels and memory
   Large global work sizes help hide memory latency. More than one thousand work items preferred. Divergent execution can be bad on GPU, though it is not a problem on the CPU. All work-items should take very similar control flow paths. Try to handle data reuse through local memory when available. You should access memory sequentially across work-items.

3) Debugging
   Start debugging on the CPU using printf, Shark, and Assembly in GDB. Be very careful about reading and writing out of bounds on the GPU. Use explicit address checks around read and write if a kernel is crashing to locate problems. Play nicely with other apps as the GPU is not preemptive scheduled. You can use extra output buffers/images to record intermediate values. You can set a context call-back function to report API errors.

Adding Authentication, Authorization, and Access Controls with the Open Directory Framework
Directory Service Development in Snow Leopard

Directory Services translates data between the OS and various datastores. Interoperable plugins are used to communicate with datastores like Open Directory, Active Directory, Unix flat files, NIS, eDirectory, and custom directories.

Everybody uses Directory Services. Either as users via applications and other OS services, as developers who develop applications and other services, and as in-house developers and system administrators who configure applications and other services.

For Snow Leopard there is a new framework called the Open Directory Framework. The Open Directory Framework replaces the existing DirectoryService API. The DirectoryService API is still there, but is now deprecated.

The Open Directory Framework is a high level API, and comes in two flavors: Cocoa and CoreFoundation. The Open Directory Framework can be used to read, write, and authenticate to any DS node (Active Directory, Open Directory, ldsocal, etc). It is thread safe and simple to use. The following discussion refers to the Cocoa version, but much is mirrored by the CF version.

NSOpenDirectory Classes (4 major data types)

1) ODSession
   A conversation between your process and directory services
2) ODNode
   A representation of a datastore somewhere. You specify nodes by name and type. Datastores can be flat files, directory server, or a collection of datastores. There are two special nodes:
   Search node---nodes used for authentication
   /Active Directory/All Domains--all configured Active Directory nodes
3) ODRecord
   Some data stored in the node. The data contains a name plus attributes (users, groups, computers, mounts, people, configuration, etc, etc, etc)

   There are record methods to manipulate passwords (verifyPassword, changePassword), and to govern Group memberships. You can check for membership in a group, and to edit group membership. Remember to synchronize the record to commit the changes.

4) ODQuery (query directory service)
   This is used to find users or other attributes, or users with attribute combinations
   Queries can be run with blocking and polling. Queries can be asynchronous. Use ODQueryDelegate for receiving results.

There is a new set of Open Directory Framework error codes in the 4xxx range. Not the old DS - 14xxx codes. Documentation on the developer site.

A few gotchas and tips about using the Open Directory framework. You should check the header doc of NSOpenDirectory to find out which things are autoreleased. Use the search node for reading data rather than specific nodes. Use asynchronous (NS/CF RunLoop) queries to prevent blocking (do not use synchronous queries to block).

There is another new API called the Membership API. The Membership API is well suited for handling group memberships and identity conversions. It has modern user and group functionality. The API is a fast performer and easy to use. Finally the Open directory framework uses the Membership API. So it is about convenience and limited scope.

Group membership rules have changed over time. In the past, parsing a file was sufficient since groups were in flat files, you were limited in the number of groups, and only users were in groups. Today things are more complex. Groups can have thousands of users, users can be in thousands of groups, groups can be nested, computers can be in groups, and groups can span nodes.

Users have changed too from the days of simple username and UID/GID. Users now have more unique and complex identifiers. We have 128 bit GUID/UUID. We have the SID (MS Windows unique identifier). Finally users can have a Kerberos ID.

The Membership API is called by including the header (/usr/include/membership.h). The API

hides the group structure (as the API tries to take care of finding things for you), and translates between UID/GID, GUID, SID, and other names.

If you need to examine user and group records at the command line, do not forget the Directory Service (DS) tools.

1) dscl
   search/look up record, test authentication, read/write data, and interactive and non-interactive
2) dsmemberutil
   flush the membership cache, check group membership, do identity lookups
3) dseditgroup
   manipulate groups from the command line, check membership

For low level troubleshooting info, do not forget the ds debug log. The log is located at /Library/Logs/DirectoryService/DirectoryService.debug.log.

Suggested things to watch for when writing code in order to do Directory Services right. First you should play nice with everyone. Supporting Active Directory (AD) requires you to be aware that AD does things a little differently (naming conventions, auth methods, multi-domain forests, group membership). Fortunately, the AD plug-in shields you from most of this.

Some other do's and don'ts were given (a lot of them driven by AD idiosyncrasies).

1) use the search node
2) use common authentication methods (not all node types support all types of authentication)
3) avoid relying on nonstandard attributes
4) do not try to pull all records
5) avoid triggering events that disrupt DirectoryService
6) check to see if nodes are available
7) do not make assumptions about number/name of Kerberos/DNS realms
8) be careful with parsing, e.g. watch spacing (MYDOMAIN\my group, MYDOMAIN\myuser)
9) use Open Directory framework or Membership API for group membership. Use the Membership API for user identities.

Creating Safari Web Applications in Dashcode

The new Dashcode 3 is part of Xcode, and is only supported on Snow Leopard. You can create apps for both Safari and mobile Safari from one project within one IDE. Dashcode 3 uses a browser simulator to avoid locking up regular Safari. Dashcode 3 supports HTML 5 and CSS3.

Dashcode 3 has several new features:

1) Safari web applications via the Application cache. This allows you to create offline web apps, by caching and syncing data.
2) New templates
3) New UI parts including split layout, grid, and list.
4) HTML5 and CSS3 support. The video tag (similar in concept to the Image tag) is supported. This enables you to use CSS3 effects. You can overlay elements as well. CSS3 support includes reflections, transition, animations and transforms.
5) Binding which allows you to take JSON and XML and put directly into your web app without any code.

Dashcode gives you the ability to create web apps for Safari and the iPhone. Effectively this means create one project, and output two products.

When you are working with remote data you typically fetch user interface stuff and program logic via HTML, CSS, JavaScript. You fetch data with AJAX as XML, JSON. Last year Apple had about 20 slides to illustrate how to fetch data via AJAX. Given that this is a very common activity these days, Apple wants to simplify things.

Apple is introducing data sources. Data Sources are part of your app that talks to the remote server. They encapsulate AJAX requests, are able to parse data (JSON, XML, RSS), and help you visualize data.

So data sources get data from server. Now you need to show data in the UI of your app. Previously you would have to write glue code which moves data back and forth between the view and model of the app. So to simplify this we introduce bindings. Bindings work very much like they do with the Cocoa framework, and are for getting data from model to the view.

Behind the scenes for bindings is the Dashcode JavaScript framework (class-based, and shares lineage with the Apple online store). Core functionality of the JavaScript framework includes key-value observing, bindings, central event handling, and touch support. JavaScript does not actually include support for class based objects, but Dashcode's JavaScript class framework creates the classes for you. When creating web apps Dashcode compresses Apple's JavaScript framework, but not your own code right now. At this time, pushing data back to the server is not supported.

Data sources are an instance of DC.AjaxController. It fetches on load, accesses your data and coverts them to objects, and allows for dynamic updates. Collection data sources are an instance of DC.ArrayController. They provide for options like sorting, filtering, and selection management.

In terms of browser support, Apple optimize for Safari. Firefox 3.5 is supposed to support HTML 5, but who knows about Internet Explorer. But it is just code, and you can modify as you want.

Local Data Storage and Offline Web Applications in Safari

Everything in this talk refers to Safari 4 and iPhone OS 3.

The web has evolved from web page (single page) to web site (bunch of pages) to web applications (structure of pages).

Web apps typically consist of your applications data (state and resources) and the user's data (preferences, settings, and user content). Today the data usually is on a server, but all compute devices have local storage.

Manipulating data on the web today is not the best experience. We have tried coupling data and cookies, but this is a fragile method and limited in scope. We have coupled data with the cloud to a server the application developer controls. But the network is unreliable and slow compared to local operations, and can be complex and expensive. Then there is the combination of data with plug-ins. But plug-ins do not integrate into browser that well, and are not standard web technology.

When manipulating simple data, it should be fast and easy. Complex data needs to be flexible, fast, and work well in the browser. All data needs to be persistent and secure.

Which this data requirements in mind, the discussion moved to talking about HTML 5 Key/value storage. This is about a simple way to manipulate simple data in JavaScript in the browser, and to be able to the application state locally. Data is persistent and accessible to you for a long time and offline with no network required.

For Key/value storage data items are key/value pairs, and are simple to manipulate. They have origin based security which means that documents and web pages from your server are the only ones that get to the data. You can perform operations on Key/value storage items such as Set items, get items, Iterate items, remove items, clear items, and add items. The data comes in two varieties called SessionStorage and LocalStorage, and these two varieties effectively replace plugins and cookies. Finally, there is a basic storage event system.

SessionStorage replaces cookies for session tracking. There is one session per browser window or tab, and it does not get saved to disk. LocalStorage is persistent and global (available to all windows or tabs). The bits get saved to disk.

Storage events keep tabs and windows in sync, and are sent when any item changes. They are sent to documents with the same origin. So for SessionStorage, every frame in the current page is available, while for LocalStorage every frames on every page is available.

For people with lots of complex data, HTML 5 offers HTML Structured databases in the web browser to store complex data offline. The databases have real world SQL transactions built in, are asynchronous and callback based, and have the same origin based security (the web standard).

Structured databases for local storage offers some advantages. The data is available for offline use in the browser. There can be performance improvements with regard to speed, responsiveness, and battery life. Complexity of the system can also be reduced.

What else can be stored offline? That is where the HTML 5 Application cache comes in. Other browsers are starting to implement the idea as well.

HTML 5 Application cache allows you to store different application resources locally, or even store entire application offline. You manifest of resources (you provide list of URL's that go into your app). In addition, there is a mechanism for automated atomic updates (when network available download updates and update at appropriate time). There is a means to detect the online/offline status. Lastly, there is an object you can pulse called window.applicationCache to monitor how the application cache is working.

For the Application cache, you need to create a resource manifest. This is a text file which lists every URL in your application. There are two types of entries: explicit and online. Explicit entries are stored offline for good so that they are available no matter what. Online whitelist entries are those which you go to network if online.

You tell the web browser that you opt into the application cache mechanism by setting the html tag (the html tag finally has an official specified attribute). So for the offLineCheckers demo showed:

<html manifest="offLineCheckers.manifest">

One more thing is that you need to tweak your web server. In your Apache mime.types file, the item called "text/cache-manifest .manifest " must be active on the server.

The loading process for the Application cache is basically 3 steps. First, resources are loaded from the local cache. Second, the manifest file re-fetched in the background. Third, if the manifest file has changed, new resources are fetched as necessary.

When the update is complete you can programmatically arrange to do one or two things. You can swap versions of the application in place, or the new version will be available the next time they visit your online site.

Java SE 6 on Snow Leopard

Java is a core feature of Mac OS X, and has been built in from the beginning. As a core feature, it can be deployed in browser applets, web start apps, bundled double-clickable apps, and as server code. On Mac OS X you can leverage 64 bit, and utilize multiple CPU cores.

Java deployment is integrated into OS X, so that Java apps can "fit in". Swing uses the Aqua look and feel by default. You get access to the OS X font system, double-clickable app support, and to integrate with the dock, menu bar, and scripting services (e.g. AppleScript).

One preferred Java version per version of Mac OS X. There may be more than one version of Java installed, but the preferred version is where the focus of compatibility and enhancements are moved forward over time.

For Tiger:
J2SE 1.4.x was the preferred Java version.

For Leopard:
J2SE 1.4.2 is 32 bit and deprecated. For legacy and transition purposes.
J2SE 1.5.x is preferred on leopard, as 32 bit and 64 bit (Intel only)
J2SE 1.6.x available for Intel 64 bit, but was for developers

For Snow Leopard:
J2SE 1.4.2 and J2SE 1.5.x is gone
J2SE 1.6.x is the preferred version for Intel 32 and 64 bit

On Snow Leopard, Java SE 6 is available for 64 and 32 bit Intel, but has been optimized for 64 bit. There is a higher performance Java 6 VM, and new tools for profiling and debugging.

Now onto some of the new stuff in Snow Leopard.

The JavaApplicationStub is a small piece of Mach-O binary which links to the Java app launching framework. This is the framework which instantiates the proper Java VM for you app. Use the LSArchitecturePriority bit to include only those architectures you support or need (note that "Run in 32-bit mode" works). If you specified a supported architecture using the LSArchitecturePriority bit (and that architecture is available), then that is what gets launched. If you have not specified a particular architecture, then you will not be launched in it. New stubs (PPC, Intel 32/64) will launch in 64 bit by default. Old stubs (PPC, Intel 32) still launch, but continue to run in Intel 32 only.

Bundling you app gives you native behaviors. You get document associations. Because a bundle has a CFBundleIdentifier you have access to many services that OS X apps get. These include access to standard Apple Events, support for Spaces, access to the firewall, and the Get Info box "Run in 32 bit mode" checkbox. If you deploy your Java app using Java WebStart and create a desktop shortcut, then you get the bundle created automatically.

Out of Process Applet plug-in (coming soon)
In order to try this out, you need Safari 4 or Firefox 3.5 (with the Java embedding plug-in disabled).

The Out of Process Applet plug-in allows you to run the Java applet as a separate process increases the stability and security of the system. It represents the next generation app launching using JNLP. You can drag applets out of the browser, create desktop shortcut like any ordinary app, or launch the applet with alternate VM options. It is a developer feature for now, but Apple hopes to enable publically soon.

JNLP is the basic building block of cross platform deployment. This applies to apps, applets, and shared libraries (JavaFX, JOGL, etc.). If your deployed app needs to escape out of the standard Java sandbox, be sure to sign your JNLPs. Place a copy of the jnlp inside of your jar and sign the jar. JNLP outside your jar and inside your jar must match.

The Java VisualVM performance analysis tool from Sun has been integrated, and is new to Java 1.6. It can do stack dumps and head dumps (walks object graphs), look at bytecode via Instruments, is expandable via plug-ins (e.g. Glassfish, Visual GC), etc. It is located at /usr/bin/jvisualvm.

The discussion now moved from new features to performance.

For OS X Snow Leopard server the 64 bit Java SE 6 (using SPECjbb2005 benchmark) is about 19 percent faster on Snow Leopard compared to Leopard. We got this got this by using large pages (UseLargePages), increasing default from 4K to 2 MB. This works only on the 64 bit kernel.

Snow Leopard server should boot 64 bit by default on many Mac Pros and Xserves, but do uname -m to be sure. Large Pages uses wired memory. It will fall back to 4K if large pages fails (it needs contiguous memory).

For OS X Snow Leopard client, the improvement is more dramatic. Most Leopard users are running Java 1.5 in 32 bit mode. When they upgrade, many will be running Java 1.6 in 64 bit mode by default. Using the JVM Benchmark, we see that Java 1.6 on Snow Leopard is about 65 percent faster than Java 1.5 in Leopard.

For the client, 4 things changed to give the 65 percent speed boost. First is the transition from Leopard to Snow Leopard. Second is that we moved from Java 1.5 to 1.6. Third is we changed the default to 64 bit versus 32 bit. Fourth is that we are using the server Java VM (64 bit only).

The server Hotspot VM is great for long running processes. But by default the server HotSpot VM masquerading as the client VM has been tweaked for client apps. If you are a server oriented app, you can get the normal server VM by passing the –server flag.

Apple also implemented GC (Garbage Collection) ergonomics, which is new in Java 1.6 on OS X. It is enabled by default on beefier machines (at least 2 cores with at least 2 GB RAM), and this means the concurrent Mark Sweep (CMS) collector is being used. Apple feels this results in better UI responsiveness. But if you need the default server GC, then pass the –server flag, and this will enable the Parallel GC which is preferred for server apps.

Now the discussion moved to how you can build and tune your app for 64-bit.

All mainstream IDE's are available like Eclipse, JetBrains IDEA, and NetBeans. Xcode can be used for JNI apps as well. Additional tools for tuning and bundling into double clickable app with custom icons (Jar Bundler, Icon Composer, Property List Editor, Instruments, VisualVM) are provided.

Profiling--graphics renderer

Sun2D renderer is default in Java 1.6+. It is cross platform compatible, with a similar performance curve to other platforms. Quartz is default in Java 1.5. It is optimized for OS X native image types. It is faster for some things, and slower for others. Just pick the one that right for you.

Debugging—Dtrace
Over 500 DTrace probe points unique to Java. It is the most flexible/powerful way to inspect things like HotSpot, Libc, and interactions with the kernel. You can use Instruments to visualization DTrace results. Many scripts available online for DTrace just work in OS X (many of them originally written for Solaris).

Debugging--Thread names
Java thread names passed down to POSIX so that thread names (or their corresponding native thread) can be seen in log reports from tools like Sample, GDB, Instruments, and Crash Reporter. Already named threads are not renamed.

Debugging--UnsatisfiedLinkErrors
Now trying to provide more descriptive unsatisfiedLinkErrors

JNI and Cocoa
A new Java JNI library example is provided. It is a few Swing app which calls into the Mac OS X address book. It uses the new JavaNativeFoundation.framework.

The discussion now moved on to an update on the SWT Project.

SWT is basically the GUI toolkit which is part of Eclipse. It began as Carbon 32 bit, Cocoa 64 bit is future of OS X. The effort to port Eclipse to Cocoa began last year, and Cocoa SWT is now an officially supported platform. Cocoa SWT is the default version for Eclipse 3.5 on OS X. Eclipse 3.5 is currently in release candidate 4 status, and should ship on June 24, 2009. SWT Carbon will be deprecated when 3.5 ships.

The move of SWT to Cocoa means that you now get some things for free. Specifically mentioned were 8 new features for SWT Cocoa: mark window modified bit, date picker, sheet support, search fields, native text services, unified toolbar API, drag and drop feedback, and 64 bit.

Some final items regarding Java were mentioned.
1) Web Objects will not be included as feature for Snow Leopard (in contrast to Leopard). More information is supposed to be coming later as to the implications.
2) OpenCL integration is not a priority at this time. Right now they are not using GCD directly.
3) No comment about Java for iPhone.

Image Processing and Effects with Core Image

Core Image is an ultrafast image processing framework. It provides great effects to apply for fun or profit. It works on real time video with Core Video, and on the user interface with Core Animation. It runs optimized on all hardware, has float point accuracy, and a full color managed pipeline. Core Image has a base set of over 125 filters, which can be extended by developers by packaging custom filters as Image Units

Core image is used throughout OS X. It is used not only image editing, but also for great UI and visual effects. A few examples is in theDashboard ripple effect, special effects in iChat and Photobooth, the screen savers and visualizer, and in iPhoto and Aperture.

Filter kernels are written architecture independent language. The language is C-like, has vector types, and is a subset of OpenGL shading language. Core Image can execute filters on GPU or multicore CPU, and leverages OpenGL and/or OpenCL as appropriate.

Filters have some basic characteristics. They perform per pixel operations on an image. Filters can be chained together. Filters can be concatenated to reduce intermediate memory buffers which might slow performance. Finally filters may be structured as filter trees, so that image processing is not required to be linear.

The base set of Core image filters includes 4 geometry adjustments, 12 distortion effects, 7 blurs, 2 sharpens, 6 color adjustments, 10 color effects, 15 stylize filters, 5 halftone effects, 15 tile effect, 7 generators (produces output with no input), 9 transitions, 22 composite ops, 8 reduction operations and more.

Core Image is designed to try and optimize your rendering tree. The runtime contains a just in time optimizing compiler. Optimization is deferred until draw time, and Core Image evaluates just what is needed to display. Filter trees are pruned and concatenated for optimal performance, and code for entire tree is globally optimized.

Core Image contains several constructs known as images, filters. and contexts.
1) CIImage--An immutable object representing a pixelated image from memory or disk, or the result of a filter.
2) CIFilter--A mutable object that represents a recipe to make a CIImage. It typically has key/value input parameters to affect output.
3) CIContext--The destination where Core Image should draw the results. CIContext can be based on OpenGL or Core Graphics contexts.

There are 5 steps to use Core Image in your app.
1) Create your CIImage
2) Create a CIFilter
3) set filter parameters via key value coding
4) Create your CIContext
   To get the best performance using a Core Graphics based CIContext, you can opt-in to QuartzGL acceleration by adding QuartzGLEnable to the Info.plist.
5) Drawing the result
   Get the filter result through key value coding. Tell your CIContext to draw the image. From there you can create a CGImage, or do a Core Graphics render to bitmap.

Writing a CIFilter
There are a couple of reasons why you might write a CIFilter. An obvious reason is that the desired filter currently does not exist. It may be the case that a compound problem can be better described in a custom filter, rather than daisy-chaining existing filters.

Creating a CIFilter is a two step process. First, you write one or more filter kernels in the CI kernel language. This low level code is where the imaging algorithm lives. It produces exactly one output pixel, although it can read from multiple input pixels. Second, you wrap your filter kernels in Objective-C to create a filter or image unit. This high level code enables us to set up samplers to access images. It loads the CIKernels objects, and sets up all the kernel parameters. You define the Domain of Definition (DOD) of output image, and define the Region of Interest (ROI) of input images. You can use Quartz Composer for rapid prototyping, and Xcode for developing Image Units. There is an Image Unit tutorial and CI reference on ADC site.

For DOD and ROI, a more specific definition are the following:

1) DOD is what the filter will render. It defines the rectangle the filter produces. Pixels outside the DOD must be transparent black.

2) ROI is what the filter needs to render. The rectangle enclosing all pixels of an input sampler needed to produce the given output rectangle. Core Image will tile the images for optimized rendering. Without knowing what a filter needs the tiles might not include all of the pixels the kernel want to sample.

Some pitfalls to watch out for.

1) Parameters passed to a kernel must be objects.. So 0.5f has to be passed in as an NSNumber. Colors have to be CIColors.

2) If you need to pass data to a kernel load it into an image, you need to do 2 things. First make sure that the image is created with the colorspace set to nil to avoid colomatching. Second, make sure the kernel is the sampler is tagged with __table sampler.

3) Filter creators must supply Core Image with color components that are premultiplied by the alpha value.

For more complex problems, you can slice the algorithm into smaller sections. Filters can use more than one kernel to produce the output image. A filter can call other CIFilters. An example of these ideas is the lens filter example from Image Unit tutorial. One kernel to produce the lens scale and shine, while a second kernel to produce the programmatic ring image. Both resulting images get combined into one filter using CISourceOverComposting.

Troubleshooting ideas if things do not go as planned.

1) QuartzDebug allows you to check your filter tree when rendering. It also gives you performance data.

2) Check the console when Core Image cannot execute the filter. Look for statements like “ROI not tileable”, or syntax errors in the kernel.

3) Make sure to test your ROI functions with large images.

4) Rotate and scale your input images to flush incorrect sampler transformations.

5) Keep the value range between kernels from -1.0 to 3.0 for compatibility with older hardware.

Some additional information and ideas.

1) Check out the samples in /Developer/Examples (may also be on the ADC website).
   Quartz/Core Image
   Quartz Composer/Compositions/Core Image
   Quartz Composer/Compositions/Core Image Filters

2) Take advantage of the iSight, since you can test your filters on iSight images/video). GPU Gems sample on ADC:CIColorTracking

AppleScript and Cocoa Bridge

Documentation is in the reference library, and in TechNote 1164

Cocoa is the framework for writing OS X applications. It was originally written for Objective-C. However, it can be accessed via other languages too via a technology known as Cocoa bridges. Bridges exist for language such as Python and Ruby, and now for AppleScript.

In Snow Leopard there are a bunch of developer and language improvements to AppleScript.

AppleScript Editor (formerly Script Editor) and is in the Utilities folder. It is multithreaded so that multiple scripts can run at the same time, and the AppleScript Editor remains responsive. There is improved event logging, and finer grained syntax coloring to distinguish event types.

There has been several language upgrades. Many bug fixes and performance improvements. AppleScript itself no longer dependent on the window server, so it can now be run headless. Unicode and locale support has been improved. Full details should be in the AppleScript release notes, which is on the Snow Leopard reference library

There are enhancements aimed at developers. These are principally in regard to 64-bit and thread safety. AppleScript is now 64-bit and thread safe (works on background threads, works on multiple threads, takes care of its own locking). However, scripting additions may not be thread safe (e.g. user interfaces), and in that case the addition will execute on main thread. Scripting additions should add whether they are thread safe to their Info.plist. By default it is assumed that scripting additions is not thread-safe.

AppleScript and Cocoa were first meshed together by using AppleScript Studio (introduced with OS X Jaguar). But it was not quite the same as with Python or Ruby, etc. AppleScript Studio tended to associate a script with a UI element.

AppleScript Studio is not Cocoa in the way other languages know it (e.g. Objective-C). You often cannot use standard Cocoa tools, standard Cocoa documentation, or the standard Cocoa community. Finally you cannot use a lot of Cocoa frameworks (without ugly glue code).

The goal is make AppleScript Cocoa development work like other languages. To do this, two things have been done.

1) AppleScript Studio
AppleScript Studio is still present, but deprecated as of Snow Leopard. Deployed apps still work, and you can continue to develop existing projects. Development features are hidden by default, but you can be turned on if you need it.

2) AppleScriptObjC (ASOC)
This is a new Cocoa bridge for AppleScript, and is built into Snow Leopard. It functions like Cocoa bridge for other languages. You can use standard Cocoa tools, standard Cocoa documentation, or the standard Cocoa community. Finally you can access any Objective-C API. AppleScript is now in line with other languages, and normal Cocoa techniques just work.

NIL in Objective-C is equivalent to "missing value" in AppleScript. In AppleScriptObjc AppleScript terminology and reserved worlds must be escaped using pipes (vertical bars). Classes and constants are treated as elements/properties of current applications.

For data translation AppleScript types bridged to foundation types automatically. Foundation types bridged to AppleScript types manually. AppleScript idioms work on Cocoa objects. When calling NSError** pass "missing value" for nil. You can pass the reference to return the value to get the NSError explanation.

Creating graphics stuff (like line drawing) in the old AppleScript studio was very difficult, if not impossible. With ASOC it is possible via the old Cocoa way. Apple showed a demo which draws dots which you can move, grow, and change color. AppleScript nibs would just like their Cocoa version. So in the demo, we just use the Cocoa nib.

There are few caveats to know

1) This is not a free transition, and you will have to do some rewriting, but a lot of it is throw away. In general they find 30-50 percent less code.

2) In seed, you can access Objective-C functions, but not plain C functions. If you can access it through a supported Objective-C interface, you are OK. May or may not be fixed by GM.

3) Structure bridging only apply to NSRange, NSPoint, NSSize, NSRect. Covers just about all structure types that exist in the kit except for NSDecimalNumber. May or may not be fixed by GM.

4) Seed has a garbage collection bug in 64 bit. Stick with 32 bit for now, but will be fixed in GM.

5) ASOC is Snow Leopard or later only for deployment.

6) Can do basic Core Data stuff, but overriding Core Data objects is a problem.

7) Like Objective-C, ASOC has a single inheritance model. No AppleScript syntax to support Objective-C blocks yet. Known issue.

8) We still use the AppleScript interpreter, just like before it is not compiled. Can mix and match ASOC and pure Objective-C code, just like Python and Ruby.

9) Using the Xcode debugger may still be problematical like with AppleScript Studio.

In summary AppleScriptObjC gives you access to all of Cocoa plus features like bindings, as well as regular AppleScript. AppleScript is future compatible being 64 bit and GCD safe. You can now start to write Cocoa apps using AppleScriptObjC in a manner consistent with other scripting languages you like. AppleScript studio is deprecated.

iPhone Security Best Practices

Real world deployment considerations

This session was intended to review challenges enterprise IT face with mobile devices in general, and to expose IT to security best practices for the iPhone. In doing so, there was discussion about the discuss iPhone system integrity and the iPhone OS 3.0 enhancements.

System integrity means protecting from unauthorized access, use, disclosure, disruption, modification, or destruction. System integrity was broken into 4 categories: physical, data, application, and user.

Security is a constant battle, and involves daily risk management. One needs to identify, assess, and prioritize risks followed by coordinate actions to mitigate unfortunate events.

There are several sources of risk for mobile devices. The device can be lost or stolen, resulting in the loss of physical control. Non-authorized persons may get access to highly sensitive corporate

or personal data. There may be weaknesses with applications, whether they be malicious or accidental.

There are 3 mitigation measures. First is to simply deny use of the device altogether. Second is to let users deal with the device themselves, and hope they know what they are doing. The third, and more practical method, is to enforce organizational policies on the device.

In order to enforce organizational policies on the device, you need to know what are your IT policies on such devices. What are the built in security features you need to manage, and how do you mange the policies for your iPhone devices?

The discussion then moved to the 4 categories for system integrity.

1) Physical--protecting from unauthorized access to phone

The IPCU (iPhone Configuration Utility) is now at version 2.0. A new feature of interest is that it can now prevent removal of profiles once installed (unless device gets wiped).

There has always been a means of locally setting passcode protection. With IPCU v2, you can now do it by policy. With iPhone OS 3 you had additional restrictions you could set locally, and with IPCU v2 you can set restrictions by policy as well.

As a general rule, you should turn off any radio interfaces you do not need. You can also use the setting for airplane mode to turn off all radios if necessary.

You can consider adding a PIN on the SIM card is the SIM contents are considered sensitive. This is different than the device passcode.

SMS Message Preview display can be turned off to block potentially unwanted SMS messages appearing on screen. You cannot control what others say in their messages.

Audio recording may be a complication since there is no current method to lockout microphone while not in phone app.

2) Application integrity

Protecting from modification through verification and trust This is done via technology (code signing and sandboxing), and by Distribution (getting IT apps to your users).

Code signing (only code pages, not supplemental files like nibs and images) is a method for verifying integrity of an application. It ensures that an app has not been modified, and to prevent the app from being modified after install. Code signing limits sources of rogue code, and raises the bar for code injection exploits. It does not prevent an app from misbehaving, does not protect data or communication, and does not ensure you are the authorized user to access corporate data. It does not help in preventing social attacks, or data driven attacks. Nor does it address correctness of code.

App store applications developed by a third party and signed by Apple. This is the App Store way, and it is trusted by every device in the world. Your company apps developed in house and signed with your developer ID, and are trusted by the devices you control via provisioning profiles.

Provisioning profiles instructs a device (or restricted set of devices you select) to trust apps signed with your identity. The profiles are signed by Apple, and obtained through the developer portal.

Provisioning profiles allow you to control which in-house apps run in your organization, and which devices can run the apps.

Sandboxing isolates apps from each other. It is a Mac OS X seatbelt technology that limits access based on the principle of least privilege. It constrains access to only authorized resources, and prevents access to unauthorized resources (prevents one app from accessing another app's data).

There are multiple ways for distributing your apps (that you built, not via the app store).
Both the provisioning profile and the app must be installed. You can use iTunes or IPCU. Then iTunes will auto push both from known locations. If there is no profile, then app should not install on the phone (and definitely should not launch if installed).

If you are running an unmanaged system, you can distribute the .mobileprovision file for the user to drop on to the iTunes app. Alternatively (particularly if you are in a managed environment) you can either push the provisioning profiles (Mac or Windows), or set the profiles via a script, to one of the following locations. The profile can then be auto-installed by iTunes.

For Mac OS X
A) ~/Library/MobileDevice/Provisioning Profiles/
B) /Library/MobileDevice/Provisioning Profiles/
C) Path specified by the ProvisioningProfilesPath key in ~/Library/Preferences/com.apple.itunes

For Microsoft Windows
A) C:\Documents and Settings\username\Application Data\Apple Computer\MobileDevice\Provisioning Profiles
B) C:\Documents and Settings\All Users\Application Data\Apple Computer\MobileDevice\Provisioning Profiles
C) Path specified in the HKCU or HKLM by the Provisioning ProfilesPath registry key SOFTWARE\Apple Computer,Inc\iTunes

In both cases you can push apps to users, and users can drop the apps onto iTunes to get it added to their library to sync.

3) Data Integrity--protecting from the modification or disclosure of data

Apple uses a certificate based trust system (X.509). The iPhone OS trusts the same root certificates as Mac OS X (http://support.apple.com/kb/HT2185). New anchors can be added to the system via configuration profiles (.mobileconfig), email attachments (.cer, .crt, .der), or downloaded via Safari (application/x-x509-ca-cert).

iPhone server certificate must be validated. If it is to be validated against your server, then add the necessary certificates to the iPhone configuration profile. If it is validated against an external server, then users should accept with caution (user training).

Data at rest on computer can be protected from disclosure by encryption. Backups of iPhone data are managed by iTunes, and backups can now be encrypted.

Data at rest on the iPhone can be protected in several ways. The first way is to keeping sensitive data on corporate servers, and only store nonsensitive data locally on device. You can utilize TLS for temporary data access, and for secures transmission of data and credentials. No data stored locally on device if lost or stolen. The second way is for apps to do self encrypting data storage. Alll iPhone apps can encrypt their own data, and have their own keychain. The third way (iPhone 3GS only) is using the new hardware encryption.

The new hardware encryption for the iPhone 3GS is always on, and encrypts all user data and applications using AES 256. This means pretty much everything other than the OS, which is not modifiable anyway. The hardware encryptor is not FIPS certified, but Apple knows that is an issue people are asking about.

From the beginning, there has been local settings for doing a local wipe. Local wipes take about 1 hour per 8 GB. If the battery runs out, wiping continues on power up. With the new hardware encryption, wiping is instantaneous on iPhone 3GS. As of iPhone OS 3.0 you have several remote wipe options. You can register the phone with your MobileMe account, and use MobileMe to issue a remote wipe command. If you have a corporate Microsoft Exchange mail account, then the device can be remotely wiped via Exchange Server 2007 (Exchange Management Console, Outlook Web Access, or Exchange ActiveSync Mobile Administration Web Tool) or Exchange Server 2003 via ActiveSync mobile Administration Web Tool.

4) User integrity

The user integrity problems are we trying to solve involve user credentials and enterprise IT. User credentials are used to authoritatively establish the user identity, and those user credentials must be protected on the device. For enterprise IT, we need to protect services from unauthorized access, and support industry standard verification.

There are several forms of credentials. First is passwords, which are a shared secret between multiple parties. Second is one-time passwords which requires user entry every time its used. Third is X.509 digital identity (certificate and private key), which requires significant infrastructure. The X.509 certificate identifies the key holder and permitted usage. The private key is the secret held to prove identity.

In iPhone 2.x credentials can be deployed to iPhone wrapped as PKCS12 files. The PKCS12 files could be web hosted, emailed as an attachment, or sent via the IPCU. In iPhone 3.x you have the additional choice of using the Simple Certificate Enrollment Protocol (SCEP). With iPhone OS 3.x, there is the new extensible IAP technology for developing other custom authentication methods over Bluetooth or wired protocols.

Finally note that you can have multiple profiles. One profile for public credentials, and another profile/set of profiles for user settings.

Like they do for Mac OS X, Apple is collaborating with NIST and NSA to produce security guides. You can consult other users profiles/experiences at http://www.apple.com/iphone/enterprise/. Note that iPhone guidance is already under development, and a FDCC (Federal Desktop Core Configuration) document will be produced. Interim security guidance for standalone iPhones can be recommended based on standard security principles (similar to those of Mac OS X).