

Apple Worldwide Developer Conference 2007 Trip Report

June 11-15, 2007

Lawrence Peng
Directors Office
L-001
Lawrence Livermore National Lab (LLNL)

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

Some Major Take-Home Points

The platform is definitely healthy. Many sessions are packed. Like WWDC 2006, WWDC 2007 focused on Leopard and there is some overlap of information from last year. These notes will try to focus more on the new information.

The first hints have been given that Carbon will be phased out in a future release. The first step in deprecating Carbon is that Carbon UI will not be available in 64-bit. Carbon remains fully implemented in 32-bit.

Java 1.5 is the default on Leopard. Java 1.4.2 officially deprecated, but there for compatibility. Support for Java 1.4.2 post-Leopard is not guaranteed.

Based on the WWDC 2007 beta, minimum Mac requirements for Leopard is an 800 MHz PowerPC G4 or higher, or any Intel Mac. The WWDC beta installer failed on a stock 733 MHz G4 tower.

The first steps towards deprecation of UFS have begun. The Leopard Disk Utility app will no longer format UFS. Leopard will not install on UFS.

Safari 3 is cross platform (Leopard, Tiger, Windows XP and Vista), and is available today as a public beta. Regardless of platform, Safari uses the same technology throughout.

While there was no official SDK introduced for iPhone development, there is an obvious way to develop apps for the device. iPhone has the full Safari engine, so you can immediately start developing apps which support Web 2 plus AJAX. This does not work for everyone, but is the current Apple blessed method.

As first mentioned at WWDC 2006, NetInfo (including the tools) is dead and gone in Leopard. Directory Services tools (introduced with OS X Jaguar) have taken over, and NetInfo is being replaced by a new local data store.

Apple has achieved UNIX 03 certification for Leopard.
(<http://www.opengroup.org/openbrand/certificates/1190p.pdf>)

Keynote Address

Steve Jobs (CEO, Apple, Inc.)

The weblink for the keynote address is:

<http://events.apple.com.edgesuite.net/d7625zs/event/>

The keynote began with a humorous video of actor John Hodgman (as the PC in the Get-A-Mac commercials) impersonating Steve Jobs. In the video, he announces that effective immediately Steve Jobs is quitting and shutting down Apple. This is due to the success of Microsoft Vista and the Microsoft Zune. Vista is doing so well, selling tens of dozens of copies, and of course the Zune is the iPod killer. The Mac appears and calls his bluff, so the PC reverts to plan B by trying to imitate Phil Schiller.

There are more than 5000 attendees, which makes this the biggest WWDC ever. Today there are 950,000 Apple Developer Connection members, which is 200,000 more than last year. There are 159 sessions, 94 hands-on labs, and about 1200 Apple engineers at the conference.

Steve began the keynote by reviewing the big item that Apple accomplished last year, which was completing the Intel transition. He gave recognition to the Apple Engineering teams and third party software developers, but gave special recognition to Intel for their contributions. Paul Otellini was invited to the stage, and Steve presented a special award to Intel.

Next was news about games. Electronic Arts (EA) is coming back to Mac. Bing Gordon (cofounder and Chief Creative Officer) stated that starting July 4th, 2007, four of EA's biggest titles (Command and Conquer 3, Battlefield 2131, Need for Speed Carbon, and Harry Potter and the Order of the Phoenix) will be available. Starting in August, EA will release NFL Madden 2008 and Tiger Woods Pro Tour 2008. Next was news from id Software. John Carmack (Founder and Chief Technical Officer) gave the first public showing which hinted of new stuff coming with id Tech 5.

Now onto Mac OS X. Steve mentioned that there are 22 million active Mac OS X users. About 67 percent is Tiger (15 million), 23 percent on Panther (5 million) and 10 percent running a version of OS X earlier than Panther (2 million).

Today, developers are getting a final look at Leopard before the October release. When Leopard ships in October, it will be 21 months since Tiger on Intel. Leopard contains 300 new features, and Steve will show 10 of them.

Number 1--New desktop

The new desktop is designed with the assumption that users will put a favorite digital photo as the background. The Menu bar adapts to whatever desktop photo is chosen. The dock is more three dimensional, and windows should now have a consistent look throughout, with the active window made more prominent.

The dock now supports a new folder view called Stacks. Stacks are folders in the dock allow rapid access to contents. They can be viewed as a fan or grid. An example of a stack in Leopard is the new downloads folder (in the user's home directory by default) which is being promoted to help clean up desktop.

Number 2--New Finder

The Finder is being further refined in Leopard.

There is a new sidebar which can display devices, recent searches, shared computer resources, etc. Spotlight searches can now be done on remote Macs and servers. Files can now be view using Cover Flow (the same technology which powers iTunes artwork browsing).

Dot Mac subscribers can use a feature called "Back to my Mac". In this case, both the home and mobile machines tell dot Mac what their IP addresses are. Dot Mac then servers as the bridge between the two machines, and encrypts the traffic between the two machines.

Number 3--QuickLook

QuickLook allows for live file previews without having to launch an apps. Out-of-the-box, it works with common file types. It is built using a plugin model so that it can be extended to support additional file types via third party. QuickLook is activated via space bar and can be viewed full screen. Supported document types can be scanned through, or movies can be played.

Number 4--64-bit top to bottom

Steve said that this is the first time that 64-bit goes mainstream (no separate 32-bit or 64-bit versions of OS X). 64-bit goes all the way up the app stack through 64-bit Cocoa. One version of Leopard runs 32 and 64-bit binaries side-by-side. Almost every currently shipping Mac is 64-bit capable.

Although not explicitly mentioned in the keynote, it was later confirmed that 64-bit Cocoa is the way to go for 64-bit user interfaces. The HIToolbox Carbon API is being deprecated going forward. This is a change from WWDC 2006.

Number 5--Core Animation

Core Animation was first introduced in WWDC 2006. Core Animation completes the "core" suite of APIs (others being Core Audio, Core Image, and Core Video). Core Animation is characterized as a 2.5D (2.5 dimensional) API as it deals with scenes of layers, and uses GPU acceleration when possible. The API allows developers to auto animate text, images, video, and OpenGL data with very high production values. As an example, the new Leopard desktop makes heavy use of the Core Animation framework.

Number 6--Boot Camp

As of WWDC 2007 there have been 2.5 million downloads of the Boot Camp public beta, and Boot Camp will be built-in to Leopard. With Boot Camp, you will be able to boot Microsoft Windows XP/Vista on an Intel Mac with complete compatibility and native speed performance. Contrary to the public beta, Boot Camp on Leopard is expected to have the necessary hardware drivers built-in, so you do not have to burn a CD to install the needed drivers.

Boot Camp is not a virtualization solution. You boot into Mac OS X, or boot into Microsoft Windows. Virtualization solutions can be found in commercial packages like VMWare or Parallels, or in other open source alternatives.

Number 7--Spaces

Spaces brings virtual desktops to Mac OS X with an Apple flair. Using Expose functionality (F8 key appears to be the default) you can spread out your spaces to get a bird's eye view of those spaces. You can group apps into separate spaces, and easily move apps between spaces. Setting up the number of spaces and how to switch between spaces is done in a variety of ways by setting up the "Expose and Spaces" preferences in the System Preferences app.

Number 8--Dashboard

Leopard includes two new technologies (WebClip and Dashcode) for Dashboard which were first seen at WWDC 2006. Users make their own widgets without developers using the WebClip feature. Apple is including a new Dashcode API to enable developers to create widgets faster. To date they are more than 3000 widgets available. For Leopard, Apple is adding a Movie widget for checking movie times, previews, purchasing tickets via Fandango, etc.

Number 9--iChat

iChat for Leopard has some significant enhancements. Audio is improved using AAC-LD (low-delay), and photo booth effects can be used to customize your "video look". You can do tabbed chats, and create custom backdrops for video chats. Backdrop demos were shown with George Washington and Steve Ballmer.

Another useful feature is the iChat theater. Here you can participate in making audio-visual presentations on one screen. Any file format that works in QuickLook will work under iChat (e.g. Keynote, JPG, QuickTime, etc.).

Number 10--Time Machine

Time Machine continues to improve since its debut at WWDC 2006. It tries to address the fact that almost nobody does an automatic backup. Time Machine by default is intended to be a one click setup and auto backs up everything. You

can search back in time for lost files, preview them with QuickLook, and restore them with one click. You can restore your entire Mac as well.

The backup drive can be a local hard drive (or USB or Firewire) or network server, and backups can be done over the wire or wirelessly. A backup drive can be used to support multiple Macs as long as it is big enough.

Developers get a Leopard beta today, and Leopard will ship in October. There is one version (not Basic, not Premium, not Business, not Enterprise, not Ultimate) for 129 dollars.

But there is one more thing.....

Safari is Apple's web browser which today has 18.6 million users. Safari's current marketshare is about 4.9 percent; versus 78 percent for IE, 16 percent for Firefox, and 2 percent for everyone else.

Apple wants to expand Safari's marketshare. As of today, Safari 3 is cross platform (Leopard, Tiger, Windows XP and Windows Vista), and a public beta is available today. Regardless of platform, Safari uses the same technology throughout. Google and Yahoo search is built-in. iBench tests shows Safari HTML and JavaScript performance is about twice as fast than IE 7 on Windows XP/Vista, and about 1.6 times faster than FireFox 2 on Windows XP/Vista.

But how to distribute Safari? There are typically 500,000 Firefox downloads a day, but typically one million iTunes a day. To date, there have been more than 500 million downloads of iTunes for Windows. So there is an existing proven method in which to get Safari distributed to customers.

And finally there is one last thing.....

iPhone ships and goes on sale on 6 PM on June 29, 2007.

Developers now have way to develop apps for mobile devices and keep things secure. iPhone has full Safari engine which supports Web 2 plus AJAX. Such apps integrate with iPhone services, run sandboxed on iPhone, and are secure (same as used for transactions with Amazon or banks). Being associated with Safari allow for such apps to have instant distribution, makes them easy to update, and does not require a separate SDK.

As an example, an Apple internal web app was demonstrated which links to Apple's internal Apple Directory (which is on an LDAP database). The app is about 600 lines of code, and took one-person month to develop. The app has the expected appearance of the iPhone contact lists with the usual info of office phone, email mail, office location, etc. The contact information fields are linked with other iPhone services, e.g. office location can be visually seen via internally supplied images, or even Google maps.

OTHER NOTES:

Below are some highlights from some other sessions I attended. These sessions are not broadcast to the general public, so the notes are at a high level.

Mac OS X State of the Union

Successful desktop apps today need a "wow" factor. Just being GUI-based is not enough. Apple sees one of its goals as enabling the creation of amazing apps. Apple does this by leveraging the power of the silicon, a solid foundation on which to quickly build apps and services, and a first class user experience.

At WWDC 2006 only the Mac Pro desktop was 64-bit capable. As of WWDC 2007, most shipping Macs are 64-bit capable.

Why 64-bit? For certain apps, you get increased computational speed. There are optimized instruction sets on both Intel and PowerPC for 64-bit computation. 64-bit supports a much larger address space, and the larger address space can make for simpler programming. Finally, 64-bit provides more registers on Intel, which can help simplify coding.

Tiger had 64-bit at the Unix layer, but with Leopard 64-bit is extended up the stack. A change from last year in that the Carbon-based HIToolbox will not be available in 64-bit. Cocoa UI is the way to go for 64-bit UI. Leopard allows for both 32 and 64-bit apps to run side-by-side at the same time.

The recommended way to support the vector engine is via the Accelerate Framework. The framework supports vector math, linear algebra, and signal and image processing. Complete functionality is encapsulated with industry leading performance. The implementation is platform independent, and designed to be future proof for future revisions of vector engines.

Multicore is the in-thing today, so you need multi-processor (MP) software for multi-processor hardware. You need a scalable OS core, multi threaded graphics, and MP ready apps built on top of a strong foundation like Unix.

CPU utilization in building apps is improved in Leopard. In Tiger you get 40 percent at 8 cores, but Tiger was not designed with this many cores in mind. In Leopard it is 70 percent CPU at 8 cores.

On the topic of multithreaded graphics, we have multithreaded OpenGL in Leopard (not Tiger) and multi threaded core animation. Core Image rendering shows a linear scale up with the number of cores.

For MP ready apps you need to worry about multi-threading. A spinning cursor means that an event thread is non responsive. This can happen during long operations, or if the system is waiting for synchronous events. This prevents concurrency and therefore prevents scaling.

Making an app MP ready means to have it event driven using a run loop, being on the lookout for notifications, and factored to use a MP model. In Leopard there is a new file system notifications API which can watch entire hierarchies, only rescan what changed, and get all changes since last run. This API is key for the Finder and Time Machine. MP model today means using threads. Leopard has new APIs (e.g. NSOperation and NSOperationQueue) to assist in managing threads, rather than the developer trying to do it themselves. The upshot is that you need to scale your app. Multi cores are here so you should get your app MP ready. Minimally you should free up your event thread, use notifications, adapt NSOperation, and test on an 8 core Mac.

On the graphics front, some key difference between the CPU and GPU were reviewed. GPUs tend to have more cores, more gflops (precision can be an issue for some apps) and faster data transfers. The next generation graphics chips are expected to start pushing towards a teraflop. An example given was medical image fusion which combines info from multiple sources. It requires alignment in 3D which is enormous computation.

There are fundamental differences in usage models for the CPU versus the GPU. The CPU is for random access of data which means you want low latency. This means that APIs tend to be implicitly parallel and statically compiled. On the GPU, it is the opposite for common use today. You tend to deal with streaming data access which means you want high throughput. The APIs for the GPU tend to be explicitly parallel and dynamically compiled.

Parallel processing is not transparent to current software, so you should use higher level approaches. For the CPU try using NSOperationQueue, and message passing/Unix. For the GPU, use Core Image and Core Animation when possible.

So new paradigms are coming with continued utilization of the GPU for general purpose tasks. Looking further in time, Apple predicts CPUs to get more cores and GPU to get more programmable. However many of the fundamental differences will remain, so a mission/challenge for Apple is to be able to exploit these differences while keeping app software development easy.

Starting with Intel Macs, Classic is gone and Rosetta is pretty much set. There will be a time when Carbon will fade out, but no timetable was given. Cocoa is the toolbox going forward, which is why Carbon UI is not included for 64-bit.

Cocoa's native language is Objective-C. Apple's focus is to make coding more approachable and write less of it, resulting in code that is faster and more reliable. Garbage collection is part of Objective-C, version 2. Garbage collection is an opt-in per app, and the Cocoa frameworks work either way. Be aware that code with garbage collection may not be faster than without. Enhanced runtime API optimized for 64-bit.

Some of the new frameworks for Leopard are a calendar store API, Imagekit, and Core Animation. ImageKit is a framework for manipulating and rendering images,

utilizing the GPU when possible. Core Animation is a 2.5 dimensional rendering and animation API for both content and user interfaces. Core Animation is integrated with the AppKit and built into Interface Builder. These APIs are leveraged in Leopard client and server. The Calendar Store API (first shown at WWDC 2006) is part of the basis for Leopard Wiki and Group services. Leopard Server retains all of Tiger Server's services, and adds new services for calendaring, easy setup, Spotlight, wiki, podcast, and directories.

The Xcode development toolset is undergoing a major revision. Xcode is Apple's first real big commitment to creating and maintaining tools. Among the new things in Leopard is Xray, DashCode, Interface Builder from scratch, and Automator 2. The drivers for many of the new frameworks and tools are encapsulation and reuse, object orientation, libraries, and tools.

The Leopard single unified windows look is an amalgamation of the best elements of Tiger. The brushed metal theme is gone (to some applause). Among the new elements are shadowing to help define active window and standard panels. Apps which utilize the full screen (e.g. media, immersive, and games) have standard panels with a heads up display (HUD) look. The HUD look is for keeping the user focused on the content, not the panel, but is useable in regular viewing. Apps which are good citizens in using the theme API in Carbon or Cocoa get the new UI elements for free. As mentioned in the keynote, there is a standard download area. If you observe it you get downloads stack membership for free.

In addition to OS X Leopard, Safari 3 is available (public beta) for OS X Tiger and Windows XP/Vista. Safari is based on WebKit and WebKit is used by the help system, dictionary, iChat sessions, mail for composing, dashboard, etc. Regardless of platform, Safari uses the same WebKit engine, same performance, same features, same open source engine, same open standards, etc. As part of open standards support, Safari supports Web 2 and AJAX. So, for example, Google docs and Zillow run just fine. Developers are encouraged to consider web apps development as well as traditional apps. There are about 17000 native apps for OS X, more than 3000 OS X widgets, and more than 100 million websites out there.

More debugging tools are being built into Safari, and Safari on iPhone run the same engine as on the desktop. Safari on iPhone supports Web2 and AJAX, but flash or java is not supported at this time. Safari is the present bridge for app developers. When building web apps for iPhone, developers need to consider standard, screen size and touch interface.

The session closed with a review of what has been the priorities in the 10 years since Steve came back. The first few years was rebuilding the brand and define Mac OS X. The next few years to establish Mac OS X and expand the releases. The last few years was for spending more time on innovative resources.

Development State of the Union

Mac developers need tools which are fast, scalable, tightly integrated and foster innovation. These were some of the drivers for version 3 of Xcode and Interface Builder for Leopard.

Xcode 3 features a variety of enhancements and improvements. Among them are 64-bit support, a new source editor, syntax coloring, code completion, code folding and code focus, increased speed and being tuned for many processor cores. The workflow is more editor centric by having message bubbles, debugger bar and data tips. Debugging is more seamless with the addition of a automatic run/debug mode switch and mini debugger. Objective-C refactoring has been significantly enhanced to be enable a safe refactoring workflow.

Interface Builder 3 features several significant enhancements. It has a new workflow. It supports document window searching. You have a multi-selection inspector, and a new connection panel. Animations and effects are easier with support for Core Animation and Core Graphics. Class information now auto synchronizes of with Xcode and Interface Builder.

Objective-C is the language of choice and the language has undergone a major revision. Version 2 has garbage collection, 32 and 64-bit runtimes. If running as 64-bit only, then you can have a unified exception model with C++.

A new tool for Xcode 3 is Xray. Xray is a tool which monitors multiple performance aspects. It has a timeline structure, and allows you to correlate measurements and drill into the details of events. Xray is partially built on DTrace. DTrace is a scriptable open source technology which provides for system wide tracing of various system behaviors.

The basic workflow of Xray consists of putting instruments together on a timeline. There are about 15 instruments provided out-of-the-box, and more can be custom defined. The timeline shows visualization, and you extract detailed data mining from the recordings. User interface actions are recordable. Documents can be saved as templates which can be invoked right from Xcode.

Apple wants the Mac to be an exploration platform by having developers take advantage of Mac by participating in and leveraging other communities. Many developers are passengers with little control of an entire development stack. Developers have different needs, and want a system which can deal with any build system and any language. Currently, Interface Builder generates source code for Objective-C projects. Projects which involve other common scripting languages typically require using several different apps (e.g. Finder, Terminal, and a text editor), such that the development environment is disjointed.

In Xcode 3, there is a new feature called Xcode organizer. Xcode organizer provides a file browser, and an embedded editor for common scripting languages. It has a scripting infrastructure so you can define scripts for build and run

and more. Xcode organizer recognizes some arbitrary build systems (e.g. autoconf and make). Xcode Organizer means a more integrated experience, quicker access to frequently used files and projects, and scripted operations.

Another set of tools and solution come via automation. App vendors can create custom solutions combining multiple apps using Applescript and Automator. Automator is essential part of scripting community, and there are thousands of downloadable actions.

Welcome to Leopard Server

A principal driver for Leopard server was to take advantage of the recent hardware transition to multi-core processors, 64-bit support, etc.

Among the OS foundation enhancements for Leopard are:

- 1) multi-core scalability (e.g. new scheduler, multithreaded TCP input, more parallel system calls by reducing global locks, virtual memory optimization)
- 2) sendfile support
- 3) POSIX spawn()
- 4) improved 10 GB ethernet performance
- 5) improved HFS+
- 6) NFS server enhancements

A few of the new services mentioned include:

1) iChat Server Federation

connect your iChat server to another company's iChat server. No client side changes. You can envision connecting a small business to another partner company, or to family and friends on Google

2) Time Machine Backup Server

You can manage preferences via Open Directory and backup to servers, Xsan volumes, etc.

3) Spotlight Server

You can search servers. There is no server side setup, smart folder. There is an included Apache module for searching remotely via a web browser. This might be an opportunity for Spotlight importer for iPhone!

4) RADIUS Server (based on freeRADIUS)

You can imagine a typical university network in which you can point base stations to an Open Directory server to regulate access.

5) Web Platform

The usual assortment of web services plus a few new ones (e.g. Apache, MySQL and PHP, Ruby, Ruby on Rails, Mongrel, Capistrano, Perl, Tomcat, WebObjects, etc.). Most of these services are 64-bit now.

Efforts have been made in Leopard Server to make administration and setup out-of-the-box easier. The setup assistant has three default setups (standard, workgroup, and advanced). You can create a custom install as well.

The standard install is for small business. Typically you have no dedicated IT staff, and just want basic services from an all-in-one server. It provides for automatic network setup with no additional setup required, and automatic client setup.

The workgroup install is for departments within institutions and corporations. Typically a few local services are used, and integrated with the larger infrastructure (e.g. centralized directory). The local IT staff have freedom to manage their own server, with easy to setup Kerberos and automatic client setup. The advanced tools are still available.

The advanced install is for advanced administrators for specialized deployments. You have advanced and flexible configurations, and more administration options with advanced tools. You have the option for tiered administration and various scalability enhancements (e.g. server grouping and smart groups)

A new service in Leopard Server is Podcast Producer. There are typically 3 billion podcast downloads per week today. Podcasting is an integral part of iTunes and iTunes University. Though called Podcast Producer, it handles most any type of content. It facilitates content creation and distribution by groups. It aims to provide a automated system which is scalable (e.g. Xgrid and Xsan), and can integrate securely with existing infrastructure.

Podcast Producer is a complete end-to-end solution. The basic process consists of three parts. First is the Podcast Capture app which lets you remotely control various recording systems (e.g. video recorder). Second is the Podcast Producer Server which sends the newly recorded content to a workflow engine for tasks such as encoding, append video, watermarking, etc. Third is publishing servers which could be serving as streams, via iTunes U, via weblog, etc. The modular design permits a large degree of extensibility and customizability.

The capture app is part of Leopard client, and accepts four Podcast sources (video, audio, your local screen, and files). The capture app is available as a command line tool, which means you can create your own front-end if desired.

The workflow engine ships with 18 workflows, and you can create custom ones. Custom effects can be added utilizing technologies like Quartz compositions. The engine performs its distributed processing via Xgrid/Xsan. You can add additional Xservices and Xraids to scale your computational and storage needs.

Delivery of podcasts can be to iTunes, iTunes University, streaming server, web downloads, email notification, archiving, etc.

Management for everyone

1) ARD 3.2 in Leopard--improve it for DSL connections

2) System Imaging

There is a brand new system image utility which includes Automator support. Netboot status is now visible through the Server Admin app.

3) Workgroup manager

--App launch restrictions (apps blocked at the kernel level)

--hierarchical management (set preferences for users, group, computers)

--command line tools (to get info out about managed desktops)

--New managed preferences for Time Machine, Dashboard and Front Row

--external accounts (accounts on external storage)

--one step FileVault

Two additional major new features in Leopard Server are iCal Server and Wiki and Group Services.

1) iCal Server

In a Microsoft world, calendaring means Exchange Server with Outlook. In an iCal server world one can use iCal, Thunderbird, Outlook (need third party adaptors), project management, or a web browser.

The iCal server is built on CalDAV, which has wide vendor support and leverages existing protocols. It features a robust feature set with most everything but the kitchen sink. Apple is working closely with CalConnect which is leading the standards effort.

One open source project to integrate Outlook with CalDAV is Open Connector (<http://openconnector.org/>). It aims to have full CalDAV support from Outlook with a full map message store. It works alongside Exchange and is being tested tested against Outlook 2002 to Office 2007.

Additional iCal Server notes are in the section titled "Leveraging the iCal Server Platform: Standards-based Calendar Services".

2) Wiki and Group Services

For notes on Wiki and Group Services, please see the notes in the section titled "Discovering Wiki and Group Services in Leopard Server".

Discovering Wiki and Group Services in Leopard Server

Existing collaboration tools like iChat, Mail, and iCal (as they stand today) typically provide for one-to-one collaboration. How about collaboration between one to many, or many to many?

Vannevar Bush described the idea of a Memex (Memory Extender) as desk full of microfilm with lots of information which are editable and associable. Apple's Hypercard was a similar idea in the pre-internet days. The Wiki is the internet version of the Memex. Wikis provide a means to increase the value of information via collaboration.

But there are challenges to making Wikis for everyone. Information posted is only useful if people know where it is. Another problem is version control. Most people do not know what version control means. Which do you want to edit? Do you want people to have to understand an obtuse syntax, or to supply a rich experience? Things are screaming for an interactive design. Wikis are designed for the internet, but there are issues making them end-user and IT friendly, as well as integrating with existing infrastructure and access controls.

In Leopard, Apple is introducing Wiki and Group Services. It is a set of integrated services built on open standards to create groups, view directories, create Wikis, blogs and podcasts, setup web-based calendaring and mailing lists. These services are built on top of tags, RSS, search, and directory services. Browser support includes Safari for Mac and Windows, Firefox 1.5 and 2.0 (and Camino), and Internet Explorer 6 and 7. Atom Publishing and meta weblog API's are supported. Apple want Wikis to be for everyone, and not just for nerds.

Apple provides the Directory app for creating new groups, choosing services and determining who can see and write, and as a general directory viewer. The Directory app is written in Cocoa, and ships as part of the client and server. Several attendees requested a cross platform port, citing the lack of a cross-platform version as a possible show stopper for mixed clients.

Wiki design goals

1) Easy to manage and setup

Apple is targeting Wikis for the workgroup at this time. They want anyone to be able to create a Wiki (without burdening IT), but at the same time it should integrate with existing infrastructure.

Access controls are per group, not per page right now. Apple welcomes feedback, and per page permissions were among the first requests.

2) Easy for everyone to contribute, edit, organize

Wiki interaction for users is WYSIWYG-based. Copy-and-paste retains formatting. You can have attachments, and can see the html source.

Information is organize by tags, instead of hierarchical folders. On changed pages, you can do comparisons and view comments. Search capabilities are built-in, and RSS is everywhere.

3) Easy to customize, extend and integrate

The look and feel is controlled by themes. You can choose other basic themes, and can edit themes. Things are as close to CSS zen garden as you can get.

You should be able to just tweak CSS, and could include custom JavaScript and custom sidebar content.

Authentication and authorization are implemented as a Twisted plug-in (www.twistedmatrix.com). This makes it possible to implement other third party schemes like Shibboleth, OpenID, etc. Do not forget the distinction between authentication vs authorization!

XML-RPC is a way to speak to a web service, and its the X in AJAX. It is what Apple's web client uses for the Wiki. There are lots of libraries for doing XML-RPC (Cocoa, AppleScript, JavaScript, Python, Perl,.Net).

During Q&A, Apple folks were asked about tools to get data from other Wikis to Apple Wikis or to an archive. Apple says they had to write tools for their own problems. Requests were made to open source it, or at least make it available.

From Power On to Login: Inside the Mac OS X Leopard Boot Process

This session tries to reduce the mystery of the OS X boot process on Intel Macs. The hope is that this will lead to better debugging and design, and reduce support costs and confusion. It provides an overview of what happens when you turn on your Mac, what can go wrong, and where can you hook in (or even if you should). Much of the following is for debugging, and you should always use supported interfaces. Loadable filesystems are not covered in this session.

Booting in a nutshell involves 4 basic buckets.

- 1) init/ EFI and boot search (ROM)
- 2) load kernel environment and switch (Booter)
- 3) device discovery and root search (Kernel)
- 4) launchd and login (User)

There are some visual cues give hints to where you are in the boot process.

- 1) init means light on from, so EFI is on and initializing.
- 2) chime means EFI is active, and loading EFI drivers and executing them.
- 3) gray screen means you are searching for a boot device
- 4) When you are in the boot loader, then you see the Apple logo
- 5) Once in the kernel, you get the gray spinner
- 6) blue screen means you have entered userland

Turning on your Mac

The SMC (System Management Controller) processor is listening to the power button (and governs the sleep light). You hit the power button, and the SMC sends a signal to turn on the main processor, memory, and other hardware. The first thing the Intel processor does is jump to the reset vector and executes

low level firmware code (pre EFI environment) to initialize the processor, chipset or basic hardware, and memory. Then you enter EFI's DXE (Device Execution Environment) where higher level firmware code runs concurrently (built-in ROM device drivers, add-on ROM device drivers, and boot devices initialization). The boot chime is played asynchronously early in the DXE, as the sound driver is among the first to run.

EFI failure modes

- 1) RAM or hardware check failure signaled with tones and flash hardware light
- 2) failed firmware update results in boot from backup flash area

The current EFI environment based on EFI 1.0 specification. EFI option ROM for PCI devices should work as expected. Developers should use EBC (EFI bytecode compiler) to support 32 and 64-bits. The future direction is to support the UEFI 2.0 specification. Apple is a member of the UEFI consortium. Graphics card developers encouraged to use GOP (Graphics Output Protocol).

EFI lives to boot the system. It is designed load an EFI app, and run it once. An EFI app is normally an OS loader. But it could be a diagnostics program, a hardware utility (e.g. partition utility), a firmware updater, a shell, etc.

When looking for something to boot, the EFI default search path looks on local disks partition (e.g. a blessed file for HFS partitions). Alternatively you can set an NVRAM variable to indicate an explicit boot path. You can boot the system interactively by holding down familiar keys like "OPTION" (something you pick), "N" (network boot), "C" (optical disk) and "D" (diagnostic program).

You get a flash of the "folder with a question mark" (?-Folder) icon each time there is a boot search failure. EFI keeps trying to find something to boot, but ultimately if it cannot find anything it gives you the persistent flashing ?-folder icon.

Booter

The EFI booter is an EFI app delivered with OS X. It is responsible for getting you into the real OS, by loading the OS X kernel (including kexts) and executing them. The booter draws the Apple logo on the screen, which is one way to tell that the booter is running. The booter insulates the kernel from EFI

By default the device the booter was loaded from is the same device from which the the kernel and drivers are loaded. However, a different location could be explicitly specified via an NVRAM variable.

There are three possible way to load the kernel and drivers into memory.

- 1) Prelinked kernel plus drivers as one package (fastest method)
(System/Library/Caches/com.apple.kernelcaches)

The OS makes the prelinked kernelcache after booting. It examines the set of currently running drivers, then it links them together with the kernel on disk.

The resulting kernelcache contains just the drivers needed for booting this machine.

2) Kernel plus driver mkext (loads kernel, then "meta kext")

The multiple kext archive of drivers is Extensions.mkext. It contains drivers from System/Library/Extensions. Loaded kexts have the OSBundleRequired attribute. That is basically all the Leopard boot drivers. This may be more drivers than are needed for this machine to boot, but perhaps not all the drivers needed during the OS lifetime. Drivers which match still must be linked (symbols resolved).

3) Kernel plus individual drivers (which is slowest)

Used if a prelinked kernelcache and mkext are missing or invalid. You load the kernel and then check each driver individually for the needed OSBundleRequired attribute.

Safe sleep (a special case) is the same as hibernation. "Booting" looks just like regular booting except that a special nvram flag is set. The booter sees this and loads the hibernation (memory)image. It draws the unhibernate progress graphic, moves some EFI runtime memory pages around, and jumps to the loaded kernel to finish loading hibernation image. Most I/O Kit kexts see this as wake from sleep.

Netbooting (another special case) is similar to local booting. The booter must use slightly different EFI APIs to load files over the network. To reduce the number of files loaded, the booter defaults to loading only the boot.plist file, kernel, and mkext. It does not try to load individual drivers.

Booter Control

The booter looks at nvram variables (first choice by default).

- 1) boot-args can contain verbose and other kernel flags
- 1) efi-boot-kernelcache-data = path to kernelcache
- 2) efi -boot-file-data = path to load kernel
- 3) efi -boot-mkext-data = path to load mkext
- 4) sometimes:boot-file = path to kernel

The booter will also look at plist file located at:

/library/preferences/SystemConfiguration/com.apple.Boot.plist

Normally the variables are set with bless(8), but you can use nvram(8) if you know exactly what you want.

The Booter can also be controlled by various key combinations. Many of these are already familiar to Macintosh users.

- 1) cmd-s adds "-s" to boot flags for single user mode
- 2) cmd-v adds "-v" to boot flags for verbose mode (switch to text console)
- 3) Shift adds "-x" to boot flags for safe mode (ignore kernelcache and mkext)

- 4) other keys interpreted by EFI before booting (n, c, d, t, option, mousebutton, and eject)

Once the booter is done, it leaves behind notes for the kernel in the device tree. In the notes are the memory map from EFI, a pointer to the ACPI table, driver locations in memory, boot volume-uuid and path the booter came from, and the screen resolution and boot framebuffer location. It then calls `ExitBootServices()` which means the end of EFI, and things jump to the kernel which is the beginning of OS X. The Booter has loaded the correct Mach-O segment out of the mach kernel file and jumped to correct starting address.

Kernel initialization

Visually, kernel initialization begins by drawing the spinning gear (found framebuffer, scheduling and threads started up). Virtual memory is reinitialized, timers started, exception vectors installed, power management initialized, and the I/O Kit launched. The main thread then calls `bsd_init()` to set up the BSD subsystem. If any of these fail, then you get a kernel panic.

Finding a root device

`bsd_init's setconf()` calls `IOFindBSDroot()` in `IOKitBSDinit.cpp`. This in turn makes a call [`service = IOService::waitforservice("device description")`] which waits for a potential root device to appear. If one does, the BSD info about the device is extracted and passed along to the BSD layer. Then `vfs_mountroot()` establishes `"/`, and `devfs_kernel_mount()` creates `/dev`. If something fails then you get the message "still waiting for root device" in verbose mode in every 30 seconds, and the system keeps looking. In an ordinary boot, you see spinning gear for a long time, but eventually get the circle with a slash through it.

Now we have a root device mounted, and want to start up Userland. Get `launchd` started by running `load_init_program()` and using a procedure "similar" to that used to create any other system process. If this fails, then you get a kernel panic with the message that it could not launch PID 1. Contrary to some other Unix systems, on OS X `launchd` is PID 1. It replaces `init()` and `mach_init()`. Once `launchd` does its thing, user land is ready to go. Typically `Loginwindow()` is then started, which then starts the `WindowServers`. Now you can log in!!!

You can specify `"/` to be something other than just the local disk using well known methods. You can modify the `com.apple.boot.plist` file, or use command line arguments. An obvious example of using an alternate `"/` is `NetBoot`.

Boot≠Root (recognizing that booting and rooting are independent)

This technology takes advantage of the fact that booting and rooting are independent. It was first shipped with the Mac Pro with `AppleRAID` booting, and is being taken further in `Leopard`. For example, the Booter must be accessible to firmware (be on `HFS+`), but the firmware does not need to load the `Finder` or

any other "non-firmware app". Firmware looks for little helper partitions which store the boot plist, but the OS can live on an "exotic" volume.

Additional uses for Boot≠Root in Leopard is for AppleRAID for PPC and Intel, and for HFSX on PPC (HFS on Open Firmware does not understand case sensitivity). At this time, rooting from non-mach_kernel filesystems is not supported.

Finally there were developer guidelines for the power-on-to-login process.

1) You should leave System files and file system IDs alone. Do not change the booter or mach kernel. System increasingly self verifying healing so modified system files equal corruption. Files replaced during software updates. Apple_Boot erased on update.

2) You can use the various debugging knobs. You can create on-demand launchd jobs, and write kexts (/System/Library/Extensions) when necessary. You can make EFI apps if you really want to, but talk to Apple about EFI ROM drivers or Boot≠Root for exotic device rooting. Use bless(8) and kextcache -u if needed.

Developing for Unix on Mac OS X

Mac OS X is built on open standards. Starting with Leopard, Mac OS X is certified as UNIX 03 compliant, conforming to the Single Unix Specification (SUS) v3. Almost all of Unix layer is open source (the Darwin project) under various open and free software licenses. Any open source projects brought in under a given license are re-released in Darwin under the same license. Additional open source projects can be built via MacPorts and Fink.

Mac OS X is a dynamic software environment which challenges the historic Unix norm (static IP, static configuration, 24 by 7 operation). OS X users expect to take their machines to different places, meaning network roaming is a given. Many peripherals are plug and play (e.g. USB, Firewire, SATA), which means that the device configuration is always changing. Finally, 24 by 7 operation is not a given as machines (especially portables) will cycle between wake and sleep states. The way to address the above is for the OS and apps to subscribe to notifications, to try again later, and to renegotiate if necessary.

The file system layout is similar to BSD and Linux, but there are other folders which are not typical of UNIX. Most of these non-typical folders are those which are visible in the Finder.

The notion of file system domains helps define the search path for filesystem resources. In general, you search the user space first, then local resources, then the network, and then the local System. The System domain is off limits by Apple request if at all possible.

Bundles are structured directories of related material. They appear as atomic units in the Finder, and users see them as apps, frameworks, or documents.

Apps bundles use relative paths to resources they contain. They can be signed for integrity (new in Leopard). Framework bundles can be viewed as basically dynamic libraries. They may contain multiple versions of resources for compatibility. Document bundles often contain app specific data.

Access to bundles data can be done in an unstructured way via the POSIX API, versus structured access via the Core Foundation and Foundation APIs.

The Xcode development tools are basically built on the GNU tool chain, and include the standard tools plus much more. They are an optional install and freely downloadable. GCC extensions can be used to link with the frameworks.

To set your deployment targets you should use the minimum Mac OS X version GCC flag (e.g. `gcc -mmacosx-version-min=10.4`). This is now the preferred method starting with Leopard. The previous method for setting a deployment target was to set the `MACOSX_DEPLOYMENT_TARGET` environment variable. This still works in Leopard, but is deprecated. When the deployment target is set the compiler hides symbols that were not yet defined, and warns when deprecated symbols are used. This is implemented using `/usr/include/AvailabilityMacros.h`

Mach-O is the Mac OS X executable file format. It is similar to ELF (Executing and Linking Format), but not the same.

- 1) Unlike ELF, Mach-O treats plug-ins and libraries differently.
- 2) Mach-O has a two level namespace to avoid symbol collisions between bundles.
- 3) Universal binaries--one file, same source, same binary, many architectures.

As a general rule, libraries provided by Apple are dynamically linked. This facilitates software updating, and allows third party apps to get immediate benefits of bug fixes. For static linking use GCC linker driver. Do not use `ld` and `ld64` directly. For dynamic linking `dlopen(3)` is now the preferred solution for Leopard (versus being available in Panther and Tiger).

For Leopard, Unix 03 command behavior is default. Apple kept non-standard behavior when it did not conflict or if it is really needed for compatibility. You can set the `command_mode` environment variable to switch between standard (`command_mode="unix2003"`) and non-standard behavior (`command_mode="legacy"`). The system is automatically set for compatibility (unix2003) in two cases. First is for installer pre and postflight scripts for installer packages built on Tiger or earlier. Second is for apps linked to Core Foundation or higher level framework on Tiger or earlier.

For performance reasons, most of the Unix API is in one library located at `/usr/lib/libsystem.dylib`. For compatibility, common libraries are symlinks to `libSystem.dylib`.

For 32-bit source code compatibility, you define `_nonstd_source_` to opt out of UNIX 03. Note that `_nonstd_source_` is NOT available on 64-bit. 64-bit is considered a new platform, so Apple is trying to get a fresh start.

Binaries compiled before Leopard still run. Leopard binaries may not run on earlier systems without using an SDK (this was always true, but with the UNIX 03 changes you are much more likely to see it).

Open Directory handles authentication. There is a PAM API which is effectively a bridge to Open Directory for things like SSH, FTP, and login. Login window (not the same as login) does not use PAM.

Unicode is used to display characters from any language. HFS-plus file names are Unicode (UTF-16). Apple uses UTF-8 encoding with POSIX API. UTF-8 has no embedded null characters, and works with standard C string API (e.g. `strncpy` and `strlcat`). The Terminal app defaults to UTF-8. Apps get UTF-8 input, and should produce UTF-8 output. Higher level APIs for Unicode are `CFString` and `NSString`, and they work well with other CF/NS types. `CFString` is an abstract representation of UTF-16 string.

Mac OS X has an event driven architecture based on the concept of the run loop. Some best practice guidelines are:

- 1) Do not block the main thread. This causes the window server to display the spinning beach ball in GUI apps.
- 2) Keep callback code relatively fast
- 3) Post notifications/queue work for another thread

X11 is being updated to X11R7.2 from the X.Org Foundation. It is optional, but when installed it is launched on demand from the Terminal. X11 on OS X includes 64-bit X11 libraries, and has been moved from `/usr/x11r6` to `/usr/x11`. Apple changes are being recommitted to X.Org repository.

The OS X Security framework has authentication and authorization APIs, and the Keychain API for secure credential storage. The Authorization API is rights-based, prompts for passwords if needed, and allows for administrator control.

There are two security implications/goals with the Authorization API:

- 1) Apple is encouraging developers to try and get away from the historic Unix notion that if a process is running with the same UID as the user everything is fair game. Apple would like to get finer grain controls in place.
- 2) In the long term, Apple wants to avoid using `setuid` binaries. Apple recommends using IPC and `launchd` instead. The IPC message starts a `launchd` on-demand helper. A serialized authorization token sent in an IPC message can be checked against the Security framework Authorization API.
- 3) Protocols are easy to secure. Securing runtime environments is difficult.

Mac OS X Filesystems: A Walkthrough for Developer and IT Professionals

This session provided an OS X file systems overview, and highlighted particular idiosyncrasies associated with OS X. Also reviewed was some new stuff in Leopard, and the current status of ZFS.

Some pertinent points that were made:

- 1) NTFS support rewritten for Leopard, but still read only for now.
- 2) Test your code on a case sensitive file system like HFSX. This helps to catch subtle case-sensitive bugs.
- 3) Extended attributes are supported on all file systems. You should try `tl` preserve extended attributes, even if you do not use them directly.
- 4) Suggest that you not, unless really necessary, use the BSD APIs to modify the file system. This is because the BSD layer generally does not recognize limitations of a given filesystem. So suggest move up one level to use either the "Core Services File Manager" or the `NSFileManager` APIs.

Mac OS X-isms

- 1) Apple double files
 - used to back extended attributes, etc., when no native support available in given file system
 - filtered out by upper level directory enumeration calls
 - currently visible at the BSD level, but you should never access or modify these directly. Managed by kernel.
- 2) `getattrlist()`
 - does not look like Unix function, but it is a full fledged vnode operation
 - can be thought of as a superset of `stat()`
 - `FSGetCatalogInfo()` is best analogy if a Carbon developer
- 3) `MNT_IGNORE_OWNERSHIP`

This is the default for external drives. It is virtually transparent, but you should be aware of certain scenarios.

 - Non-root processes see their own uid when looking at file/directory ownership
 - uid 99 is written out to disk when `MNT_IGNORE_OWNERSHIP` is set
 - uid 99 (on disk) is interpreted as the calling process uid for non-root processes. Root processes always see the real UID.
- 4) VolFS

Do NOT use or depend on it directly!!! No more VolFS in Leopard.
- 5) Deprecation of UFS

It will be a phased deprecation starting with inability to install on or format to UFS via the GUI. The eventual goal is to have read-only UFS. For case sensitivity, use HFSX.

6) UDF

This is completely rewritten for Leopard with read and write support. The new implementation brings support up to version 2.5, and works with block storage devices (thumb drives, portable hard disk drives, etc).

7) Access Control Lists (ACL)

For Leopard, ACL are on by default. The Finder GetInfo panel has been modified to show ACL information.

8) File system Corruption detection—new in leopard

--instead of looking the other way

--dynamic detection performed during system uptime, no need for fsck, etc.

--triggered by physical disk io errors

if have a problem...

--fsck_hfs triggered at reboot as needed

--unrepairable file systems mounted as read only

--where possible, specific filenames are provided in the event of corruption

--also see new -c (block cache size) option for fsck_hfs (improved performance)

9) HFS directory hard links

This was specifically implemented for use by Time Machine and is specific to journaled HFS plus. Very conservative limits are placed on the creation of links. It is NOT intended for general purpose use, and future changes may occur. Those poking around in the backing store should beware!!!

ZFS on OS X--What is ZFS?

-advanced enterprise class file system from Sun Microsystems

-ported to OS X server

-self healing and unmatched data integrity (uses Copy-on-Write)

-pooled storage model

-RAID-Z (improvement over RAID 5)

-built in, live disk scrubbing

-first class snapshots and clones

-built in compression capability

ZFS on OS X server current status

--read only default for WWDC seed

--read-write developer preview coming soon via ADC.

--mostly command line (Disk Utility in Leopard will not format for ZFS)

--not optimized at all for performance

--ZFS not a supported feature for Leopard, but will be in a future release.

Time Machine in Depth

The goal of Time Machine is to provide a seamless and intuitive backup solution with backups always readily available. Version 1 is not meant to be everything to everyone, and is targeted at typical usage for individuals and workgroups. Time Machine runs as a system daemon.

The technology is integrated into Mac OS X in several visible ways. You can browse and retrieve backed up files and folders via the Finder. There is some level of integration between Time Machine and applications. For example, you can retrieve relevant files from within applications like Address Book and iPhoto. The Installer application allows you to backup before you install, and to restore a bootable system from a backup. Similarly, Migration Assistant allows you to migrate your user data from a backup.

Given that a convenient means to back up should be obvious, you can ask the question of why it took so long to come up with Time Machine? Apple's answer is that in order to be practical and useable for the masses, Time Machine needs to leverage technology that is only now becoming available (on Leopard of course).

The basic technologies being leveraged for Time Machine are:

1) FSEvents API

FSEvents is used to quickly determine what changed on disk since the last backup. The API monitors changes to a hierarchy as they happen, or to retrieve a list of changes made in the past. This avoids the need to scan the entire disk each time a backup happens to determine what needs to get copied.

2) HFS+ (there are several aspects of the file system being used)

First is Extended attributes. These are persistently stored key/value pairs attached to files and directories, and are associated with the items, not paths.

Second is Archive Directory Links (new to Leopard). These are hard links to folders used to quickly represent a hierarchy that has not changed since the last backup. They make the backup browsable through the Finder and Terminal. Backups are compatible with Tiger (aliases so you can navigate through them).

Third is ACLs (Access Control Lists). These provide fine grained control over who can modify what, and is used to prevent accidental editing backup content. The ACL implementation on Leopard preserves posix permissions and other ACLs.

3) Quick Look

This API is used to provide rich document previews.

4) Core Animation

This API is used to create the intuitive user experience.

By default, all files needed to restore a bootable system are backed up. System data size is typically several GB, and is expected to stay roughly constant since the size of a bootable system does not normally change in a big way.

You can skip system files. This is useful if you always reinstall system from an image. You trade off bootable system restore for additional space. The option is integrated with installer and migration assistant. It speeds up initial backup, but not incremental.

There are several items automatically skipped from backups. These include cache directories, temp directories (/tmp, /Library/Logs, ~/Library/Logs, Trashes, device mounts (data backed up as a single device for each one), and network related mounts. FileVault disk images are also handled separately.

What is new since WWDC 2006???

1) Simplified options for automatic backups. No more on/off switch, and no need to set backup time. Just check back up automatically. For automatic backups, Time Machine backs up your machine hourly. Hourly backups are kept on a rolling basis for 24 hours. The first backup after midnight becomes the daily backup. There are provisions to backup upon on-disk attachment if needed (e.g. external firewire disks).

2) Manual backups

Backups only done when user presses back up now. No hourly backups are done.

3) Stop and Resume

Backup can now be stopped and resumed later. You can now shutdown while a backup is occurring.

4) Automatic one-click configuration (if automatic backup is on)

We detect new disk and do the setup for you

5) Encrypted backups

You can back up to an encrypted HFS+ disk image

protected with single machine wide password (if not know, cannot mount disk)

time machine can automount disk to backup or restore

protects against loss or theft of disk

key written into your keychain

FileVault accounts are only backed up when user is logged in. Each account is backed up to its own encrypted image (master key on top of it). FileVault master key can access backups to restore them.

6) Spotlight

If you do not find it on disk, you can now search backups. If a file version exists in multiple backups, only one shows.

7) Backup to server and airport disk

Multiple machines can back up to a single server. Each machine is backed up to a disk image on server.

The default configuration for Time Machine is that automatic backup is on, system files are backed up, boot volume is backed up, and volumes containing user accounts are backed up.

Backups will fill your disk over time. The size of each daily backup depends upon the size of unique files. User specified backup windows (weeks, month, year) are always kept. You can set the system to selectively delete old incremental backups (if running out of space). As unchanged files may be referenced in multiple backups, deleting a backup only removes files unique to that backup. If run out of room, then may need second drive. For version one of Time Machine, multiple drives are used separately.

For developers, Apple suggests using QuickLook as your integration strategy with Time Machine. This lets users see their backups from the Time Machine UI, the Finder and Spotlight. If necessary, write a plugin for your filetype.

The structure of a backup can be described by a directory path like this:
/backup disk/backups/computer/individual backups/disks being backed up

FileVault user homes are backed up individually one to encrypted disk images. These will be seen as separate backup snapshots in a folder called "FileVault Accounts" alongside individual backups folders.

Once the encrypted disk image is opened, then you see a mounted volumes containing a general structure of: ~/Backups/Individual backups

You can back up to a network volume using AFP, SMB or NFS. Backups are written onto a disk image which can be mounted and unmounted as necessary. The layout on mounted disk image matches local volume layout (network volume to disk image). Backup is still an HFS+ image. If network backup needs user name and password, then no backup unless logged in.

The backup creation process is straightforward. The first backup is a straight copy using the FSCopyObject API. For an incremental backup, the system reads the FSEvent history of each volume since previous backup. Then it traverses the folders whose contents have changed. You then copy new and changed files and folders, and create hard links to unchanged files and folders. The system then reads the FSEvent history for changes that happened during the backup. The copy and link process is repeated until nothing interesting changes on disk while copying. Then the backup is finalized.

For developers who need to tie into Time Machine, Apple had a few suggestions.

1) To copy from a backup, you should use the FSCopyObjet API.

2) Exclude unnecessary files from backups (caches, index files, rebuildable files). Try to avoid huge, constantly changing files and flat folder structures. Support QuickLook by writing a plugin if needed.

3) Put temp files in standard locations if possible. If not, there is an API to use to add their directory path to the exclusion list. Obviously if the contents subsequently move, they may no longer be excluded.

Xcode 3.0: the New Development Workflow

This session tried to highlight the new stuff that will speed up your work.

Xcode 3.0 was designed to provide a smooth transition to new releases. Xcode 3.0 folder is self contained, so it can be taken anyplace. You can install multiple versions, and mix with version 2.5. Each version generally launches and uses its own tools. However, not all Xcode 2.5 tools run on Leopard.

Xcode 3 is interoperable with 2.4 projects. Xcode 3 opens projects from 2.0 forward. Upgrade 1.5 and earlier projects using Xcode 2.4 or 2.5. You setup project compatibility in the inspector. If you make a 3.0 exclusive setting, your choices are upgrade or cancel. If a Xcode 3-ism creeps into a Xcode 2.4 project, you see warning in status bar and can review in the UI.

Prior to Xcode 3, you often had multiple windows (edit, debug, build, etc.) which hid or blocked each other. In Xcode 3 your code is front and center.

Work pretty much from just the editor window where your code resides. Error messages during compile and build are brought straight to the offending line. Fixed errors go away after the next build and compile cycle. When debugging, controls are in a small strip at the top of the editor window. The idea is to be able to easily switch from run to debug in mid-stride (i.e. "eliminate" debug mode). Data tips, while debugging, show the value of variables as you hover the mouse over them. Break point can be set in the editor window, and you can set conditional breakpoints using editor window message bubbles. Finally, for debugging full screen apps or mouse down events, there is a mini debugger (which is a floating window with full debug window over running application).

The Script Menu (new for Xcode 3) has a facility for WYSIWYG editing of shell scripts. You can drag and drop Automator actions, AppleScripts, etc. This type was doable in earlier versions of Xcode if you knew the cryptic procedure.

There are several new things for navigation and research.

1) Project Find (significantly redone, though UI is mostly the same) You can do finds of various sorts such as in cross project references, in uses of symbols, in nib files, and for multiple regular expressions on same line.

2) Xcode News (New)

A center for news and information (including from ADC). It displays a tabbed window. Each tab contains a miniature web view for getting started, Xcode news RSS feeds, documentation, mailing lists, and giving feedback.

3) Research Assistant (New)

The assistant is displayed in a floating window. It shows vital info about the selected API, build settings, etc. The floating window always available, and is much lighter weight than a full doc window. The assistant can flag usage of deprecated APIs (and recommend a replacement).

4) Documentation viewer (New)

This is a set of modules which can be updated individually via RSS feeds.

5) Lifecycle management (New)

You can connect to or browse a repository from within Xcode. Settings for repository management can be set within Xcode.

6) Freedom to experiment---Snapshots (New)

You can make fast, frequent snapshots of your work, and restore from any snapshot at any time. You can see the deltas between the current version and the previous snapshot. Xcode can be set to automatically make a snapshots before critical operation.

7) Other new stuff

The Xcode editor is accessible in the command line via the xed tool.

Getting Started with Xray

Xray is a powerful, convenient, and flexible "meta" process analysis tool with the simplicity of an iApp. It has a unified user interface, and uses the DTrace framework. Xray is being introduced with Leopard, but will not run on Tiger.

Why is Xray useful?

- 1) records and correlates arbitrary data streams over time
- 2) provides ability to visualize and mine gathered data
- 3) streamlines edit/build/analysis cycle
- 4) simplifies and allows customized usage of DTrace
- 5) Xray complements other tools like Shark

DTrace (defined by Sun Microsystems) is a comprehensive dynamic tracing framework for the Solaris operating environment. DTrace provides a powerful infrastructure to permit administrators, developers, and service personnel to concisely answer arbitrary questions about the behavior of the operating system and user programs. DTrace is dynamic, system wide, and scriptable. On Mac OS X, it is always installed by default (user, server, or developer systems).

Key Xray Elements

1) Xray instrument

An instrument gathers data of a particular kind, such as memory, user input, I/O, file activity, etc. Out-of-the-box and custom instruments are stored in the instrument library by default. Xray instruments are built on top of the Leopard frameworks and on top of DTrace. Both the Leopard frameworks and DTrace are built on top of the Darwin core OS.

2) Trace Document hosts a set of instruments and the data they gather within a trace. Instruments get "applied" to trace documents. This is the primary place where you mine the data and analyze problems. Use inspector to tweak instruments

3) Xray trace template consists one or more instruments

Activated from Xray, Xcode, the command line, or Xray QuickStart keys

Xray replaces the tools called Sampler and ObjectAlloc with Xray instruments (of the same name). The instruments do more than the previous tools.

Xray provides various instruments targeted at memory analysis (leaks, fragmentation, dynamic usage, pointers to freed memory, etc). The instruments can be used singly or in combination in Xray.

1) ObjectAlloc (works with garbage collected apps)

Tracks all malloc/free/retain/release/autorelease calls and shows detailed address and pointer histories. Includes views of statistics and call trees.

2) Leaks (not intended for garbage collection analysis)

Checks for unreferenced memory blocks. It gives you a graphical tool to do static memory analysis. This tool works alongside ObjectAlloc.

3) Memory Usage

This tool records high level statistics like vsize, rsize, page faults, etc.

4) DTrace

A powerful way to build customizable instrumentation for specific problems.

Leveraging the iCal Server Platform: Standards-based Calendar Services

iCal server is Apple open source calendaring solution. It is based on open internet standards, and interoperates with various clients. The default CalDAV client for Mac is iCal.

CalDAV Technology

There is a data format called iCalendar (IETF RFC 2445). On top of it is another spec called iTIP (IETF RFC 2446) which makes requests for information

(e.g. available for a meeting?). HTTP is the transport protocol. On top of HTTP is XML, WebDAV and WebDAV ACLs. On top of that is CalDAV access which specifies how to get calendar data and how to write it to the server so that the server can maintain the consistency of its data and its semantics, so that interesting searches can be done on it. On top of all of it is the CalDAV Schedule spec that let you invite someone or update status of meeting and fan out that info to all those concerned.

Standards update (<http://www.calconnect.org/>)

Put together, iCalendar and iTIP combined provide standard calendaring and scheduling services. They are almost 10 years old, and supported by many products. Revisions are currently underway in the IETF.

CalDAV is split into two primary specifications

- 1) CalDAV access is complete (Published March 2007 as RFC 4791)
- 2) CalDAV scheduling is nearly complete

Additional work going on for CalDAV is the server-to-server protocol, in order to implement federated free/busy information and cross-domain scheduling. Today CalDAV works within a single organization or authentication domain.

iCal server extensions by Apple

- 1) masked uid which improves free/busy ui
- 2) CTag improves polling performance
- 3) Proxies (aka delegates)

Proxies are managed via groups, which make them easier to manage than directly editing ACLs, and to discover proxy relationships. There are two group principals for each primary principal (read only group, read/write group).

- 4) Office hours

This extension describes available time. It allows for recurrences.

- 5) Drop box

This allows attachments to be associated with an event. Attachments are not inline with iCalendar data. All attendees (not just organizer) can edit contents, and changes trigger notifications without new invitations.

The iCal server is single threaded. A master process which launches a bunch of slaves equaling number of CPUs. Clients hit a load balancer mediating access to the corresponding slave. Multiple machines (multiple server machines will have one basic file system across machines (e.g. Xsan)

The iCal Server is implemented in Python (requires 2.4 or better), using the Twisted framework. Leopard will ship with Python 2.5. It is a fully compliant CalDAV server, uses SSL/TLS for transport layer security, and authentication and authorization (basic, digest, Kerberos). Server should scale to moderately large organizations, but Apple does not plan to scale massively (Yahoo and Google probably want to do their own thing anyway).

The server binds to a directory service to obtain information about principals (users, groups, location, resources). Multiple directory services are available. On OS X Server the default is Open Directory. But you can use others like XML, Apache user/group files, etc.

iCalendar resources are stored as files. This makes them simpler to diagnose than a database. File properties, including ACLs, are kept as XML in extended attributes. SQLite used for indexing data. These indexes are disposable, and will be rebuilt next time it needs it.

The iCal server is open source (Apache 2.0 license) and more information is available at www.calendarserver.org website. Bugs are tracked in public bug tracker (Trac), with source changes sent via public email list and via Trac RSS. Mailing lists exists for developers and users, and new developers are welcome.

Calendaring services are here for OS X, and the key functionality is in place. There is more we can do, and the Leopard feature set is pretty firm now. iCal Server is ready for use, and some teams at Apple are living on it. It is mission critical for Apple too, and the iCal teams report no deaths yet.

More information about the Twisted framework is at the following URL:
<http://twistedmatrix.com/trac/>

There is another interesting application which ties in CalDAV data with project management. The application is Project X from Marware.

Discover Java on Mac OS X Leopard

Java is a core feature of Mac OS X. It is preinstalled as part of OS X, and available to apps, applets, web start, server, etc. Java is integrated into the OS, and updated through regular software updates. Apps do not need to bundle a JRE. Java support is virtually transparent to users.

The preferred Java version in Leopard is currently J2SE 5.0 (1.5.0_11). Apple normally makes improvements on top of latest technology. Users can choose the previous version in some cases. The previous version is for compatibility, and is eventually phased out. J2SE 1.4.2 is there but deprecated. Do not count on J2SE 1.4.2 after Leopard. J2SE 1.3.1 is gone and removed from Leopard.

New in Leopard

- 1) Improvements and enhancements to the Swing UI, SWT, and embedding support.
- 2) Leopard Java is switchable at the command line. When users switch Java versions in the GUI preferences (web start and double-click apps), it changes it in the command line as well (use to require going in an adjusting full paths).

3) Java performance is better than Tiger. SwingMarks for J2SE 5.0 show Leopard is 41 percent faster than Tiger. SPECjbb2005 results show Leopard as somewhat faster than Tiger (which was already faster than Linux on the same hardware).

4) Resolution independence support

You get it mostly for free. It is pretty simple to update your app's own images to take advantage of resolution independence.

5) Sun 2D renderer now default

On Tiger, Java graphics calls are mapped to the OS X Quartz API. Quartz took better advantage of OS X capabilities. The Sun2d renderer was added as option in subsequent Java updates. The Sun 2D renderer is Tiger did not support all desired functions (e.g. advanced text functions like bidirectional text rendering), so the Quartz renderer was the default on Tiger.

For Leopard, Apple improved the Sun 2D to take advantage of OS X capabilities. The Sun 2D renderer provides a better match in pixel coverage and for comparing graphics performance with other Java platforms. Apple welcomes feedback about how this change is working. The Quartz renderer is still available.

6) J2SE 5 is 64-bit enabled for Intel

All J2SE 5 apps should just work on Intel 64-bit unless you have native code present. You need to make the native code 64-bit.

7) Improved tool support for Java

--Shark is still there (Java 1.4 and above)

--Jconsole from sun is still there (starting with J2SE 5.0)

--Xray (J2SE 5.0 in Leopard) using the Java Threads instrument

--Ant support in Xcode improved (it understands build xml files)

--Ant (updated), JUnit, and Maven now in Leopard

--Jar bundler and applet launcher for build and deploy are still there

--For Profiling do not forget Quartz debug, and DTrace

--Standard Java IDE and tools (Idea, Eclipse, NetBeans) continue to improve.

J2SE 6 (Developer Preview 7) on Leopard is available today, and is for Intel 64-bit only. It runs on the WWDC beta and is in sync with Java 1.6_01. Bundled apps, Java Web Start, and command line Java (including those which launch graphical apps) should work. Apple is asking for bug reports and feedback. Early SciMark benchmark tests show J2SE 6 being 13 percent faster than J2SE 5.

J2SE 6 is planned for 64-bit Intel only right now (no 32-bit Intel, and no PPC), and not currently planned for Tiger. (32 and 64-bit support for J2SE 5). No timeframe given for final J2SE 6 release, but the first release will not be the preferred release. It was noted that the SWT implementation is Carbon-based, and as noted in other sessions there is no Carbon 64-bit UI.

Two other items were mentioned:

- 1) Updated Tomcat server in Leopard
- 2) No plans for advancing QuickTime for Java from what it is at this time.

Deploying PodCast Producer

Podcast producer is intended to be an open and flexible platform. It is designed to streamline content capture and archival storage. It offloads postprocessing from the workstation, giving a simplified interface for end user with consistent results. It can then tie into various auto publishing and notification (blogs, wiki, streaming, iTunes, email, iTunes U, etc). The infrastructure services and needs for Podcast Producer are a solid DNS implementation, Open Directory and Kerberos, a shared file system (e.g. Xsan), and a processing engine (e.g. Xgrid). Encoding on alternate platforms with Xgrid is not supported, but Apple welcomes feedback.

Some scaling factors to consider are storage, processing power and bandwidth. Approximate current storage requirements are 13 GB per hour for archival DV, 1 GB per hour for Apple TV, and 500 MB per hour for the iPod. Processing power needs can be gauged to first order by benchmarking workflows and adding Xgrid nodes to increase capacity. Xsan needs can be scaled to the number of Xgrid nodes and needs for Podcast producer. Finally you need to look at your network bandwidth for submission and distribution.

Podcast Producer Clients (e.g. QuickTime Player, Podcast Capture, third-party app) query Podcast Producer for authorized workflows to access (via access control lists and Open Directory). You choose the files to process and submit the job. Under the hood, the pcast command line utility is used. Podcast Capture is an app wrapper for pcast written in Cocoa. You can bind cameras (headless capture agents) so that the camera agent is controlled by servers.

Podcast Producer Server and Xgrid process Podcast Producer workflows. Workflows are defined in a bundle of resources. System workflows (which you do not modify) are located in /System/Library/PodcastProducer/Workflows. Custom workflows are located in /Library/PodcastProducer/Workflows. Workflows technical specifications are defined in a template.plist. Here you define task specifications and dependencies. One interesting feature is Xgrid scorecarding. Scorecarding allows you to target grid system nodes best for task.

Customer Case studies

Willie Pritchard
Academic Coordinator, Distance Learning
Podcast Producer Project
De Anza College

Current there are over 100 multi-media classrooms currently installed. Each contains a Mac mini and flat screen for the faculty, projector and screen, full audio, document camera, and other equipment and players as needed. However, currently only one classroom can be recorded. The near term plan is to add 2 more that can auto track and record classroom action with video and audio.

It is expected that many more classrooms with auto recording are to come. Ease of use is critical, preferably no need for faculty to do anything. Flexibility in scheduling and the need to have both automatic and manual control are high priorities. They also want auto placement of materials for our students into our iTunes U site and Moodle (Course Management system which is open source). Podcast Producer will be playing a big role in getting to their goals.

Adam Hochman
Webcast.Berkeley.edu
Capture and Delivery System

The current system has been in place for about 7 years. Right now Real Player is needed due to legacy decisions. Today there are 6 webcast and 14 podcast enabled general assignment classrooms. However there continues to be explosive growth in popularity of webcasts/podcasts.

Freshman expect podcasting like email and wifi. In 2006, 82 courses were recorded resulting in over 3000 hours of lecture content delivered. There were 3.5 million unique views, and 10.6 million mp3 downloads. iTunes U and Google Video are major distribution channels.

The goal for the next 3 to 5 years is to increase the number of webcast/podcast capable classrooms to 200. The next generation system must integrate with the Sakai course environment and other campus systems. It must be more self healing and possess redundant systems for failover. It must address the challenges of scalability, accessibility and sustainability. This means that they will rebuild the system from the ground up, and Podcast Producer will play a major role. The new system will have to address branding and copyright, and using Quartz Composer and Quicktime compositing tools may play a role.

The next steps is to create a scheduling utility (iCal format) to generate and consume event data. Berkeley using iCal format, and is considering iCal server. The roadmap calls for a Fall 2007 podcast producer production pilot (8 classrooms), with a Summer 2008 next generation systems rollout.

What's New in HIToolbox

For the past couple of years, the HIToolbox group has said that the future is Cocoa. The big news was the future of the HIToolbox API's. Post-Leopard will see minor enhancements only, and existing bridges to Cocoa will be maintained. Bug fixes and tweaks are all that is planned post-Leopard. Carbon is becoming what it is, and is heading to deprecation status in the future.

One path is to modernize your app via Carbon. Depending on your application, this could be a large or small amount of work. Eliminate QuickDraw and go to Quartz, and adopt HView compositing mode. This lets you get to resolution independence, and use HICocoaView. Modernization via Carbon may be a stop gap measure as there is not going to be a lot added to Carbon going forward.

If modernization via Carbon methods is a lot of work, consider going directly to Cocoa. The transition to Cocoa could be done in stages. You can do a direct implementation of Cocoa windows using NSWindows. If already using compositing via Quartz, then you can add NSViews via HICocoaView. A lot of this will likely require you to factor your app, which may or may not be a lot of work.

There is no 64-bit Carbon UI. So APIs such as Window Manager, HView/Control Manager, Dialog Manager, Menu Manager, and Navigation Services are gone. HIOBJECT API is available to generate placeholders for Carbon Events. 64-bit includes the Carbon event manager, and the new Text Input Sources API for Leopard for handling input methods. For the Carbon Event manager, any event types tied to UI events will not show. UI agnostic events are still available.

Final decisions on what to cut and what stays will be made in the coming weeks. A rule of thumb is that APIs in the Carbon Framework may be affected by the 64-bit question. If an API is outside the framework, then it is not affected.

Some enhancements to the Leopard HIToolbox are designed to help integrate Cocoa technologies into Carbon applications.

In general if you are a good citizen, you get the updated Leopard interface for free. There are tweaks which may need to be done. Some apps may still have the Tiger look because of old app specific workarounds to avoid incompatibilities with Cocoa. It may help to review previous bug reports sent to you from Apple. Apple recommends using a standard document window. Given the new transparent menu bar, be careful not to position windows under the menu bar.

HICocoaView is a new view available to compositing windows. Apple is providing it since today we have two view APIs (HIToolbox and App kit). Cocoa has richer set, and new stuff is represented in Cocoa views.

An app which uses HICocoaView is the Finder. The main content window in a Finder window is a HICocoaView (excluding the sidebar). This is so Apple could integrate Spotlight and the Finder for Leopard. Spotlight UI is in Cocoa.

To utilize HICocoaView you need to do a few things:

- 1) link to Cocoa.framework
- 2) compile modules using NSViews with ObjC or ObjC++ compiler
- 3) call NSAppLoad() to initialize app kit for Carbon apps

Another aid for the Cocoa UI transition is that in Leopard the Navigation Services API is now implemented on NSOpen/SavePanel. Navigation Services no longer calls NavPreviewProc. Instead it now calls the QuickLook API.

Mac OS X Security Configuration Guidance for further system fortification

In this session, security configuration guidance was given to further harden systems and mitigate additional risk. Guidance was given from the viewpoint of Apple and the NSA's System and Networks Analysis Center (SNAC). The purpose of system hardening is to reduce or eliminate as many security risks as possible, given the environment in which the system is functioning. Tools/services to enable these suggestions are provided by Apple and other third parties.

Top 10 suggested steps for system hardening:

1) Start protected by starting with a clean slate

Do a clean install of Mac OS X. Use an image from known organization default image. Do not assume the system is OK.

2) Securing the host

Use a firmware based password (not an account) which locks an OS boot instance to the partition GUID. Firewire drives use a bridge chip (chip==GUID). Turn off hardware ports when not in use in public and private areas. Turn off discoverability beyond setup time.

3) User authentication

Do not operate under an admin account and do not enable the root account. Use two factor authentication (i.e. smart cards). Modify authorization rights/rules as needed (located at /etc/authorization).

4) securing data and using encryption

There are several options for protecting data at rest. You have FileVault for the users home directory, portable encrypted storage containers (e.g. encrypted disk images), secure virtual memory, and encrypted Time Machine backups (Leopard). The encryption used is AES-128 for Tiger and AES-256 for Leopard.

5) securing active services

Disable or even remove all unused services. Use security enhanced versions (e.g. ssh, scp, and srm instead of telnet, cp, and rm). Leverage service ACLs for finer grained control.

6) enable firewall if active services

Enable if you are running any services. Enable stealth mode in Tiger. Limit incoming connections (in Leopard).

7) maintain system/application updates

Heed security notification from Apple and apply security patches. Maintain vigilance with 3rd party apps. Track changes to self installed open source projects. To mitigate possible instabilities being introduced by automatic software updates, the client software update reference is modifiable. The default is Apple, but you can replace it with your own internal reference.

8) baseline your system

Know what is on the system and snapshot your system if possible. Compare questionable system to the baseline.

9) auditing security relevant events

You can continuously audit the relevant events. BSM (Basic Security Module) is built into OS X. There is a Common Criteria Administration guide/security configuration guide. There are a suite of CC_Tools (Common Criteria Tools) provided. At the command line there is audit, auditd, auditreduce, and praudit. There is a GUI Utility for audit log viewing called Audit Log Viewer.

10) Legal Notification of Usage

Have a login banner with legal notifications about computer usage. Historically this is done by modifying the login window .plist with banner text. Another possibility today is to utilize Authorization Plugins and create you own.

A representative from the National Security Agency (NSA) System and Networks Analysis Center (SNAC) took the stage to outline their approach to developing guidance on securing an operating system.

NSA philosophy on securing a system involves policies and user education (a continual process), system monitoring, and system patching and configuration (largest factor). System configuration is difficult as systems are in flux, updates, user needs, patches, system requirements, etc. Administrators and developers must block all holes, and attackers only have to find one.

The NSA Configuration guides were originally all NSA produced and maintained. Eventually NSA began developing vendor relationships, and began to create the guides in collaboration with the vendor. Today, guide development and maintenance is pretty much the responsibility of the vendors.

During guide development by the vendor, NSA provides input and comments. NSA intent is that vendor guide address security guidance needs for US government customers. If NSA concurs with the guide content, guides are posted with endorsement on nsa.gov website. If NSA does not concur, then NSA publishes an

addendum. In the case of Tiger, NSA has no issues with the guide created by Apple (so published no addendums).

General principles of guide usage (NSA emphasizes the most secure situation)

- 1) separation of function--different servers for different services
- 2) separation of rights--separate admin and user accounts
- 3) encrypt where possible, even locally
- 4) minimize services, especially network services, turn off ip6 if not used
- 5) remove unneeded/unwanted kernel extensions (remove after updates if needed)
- 6) stress highest security/most locked down configuration
- 7) explain implications and provide step by step instructions

Security guidance can be customized to fit your needs. First you need to determine what you have. Then determine what level of security you want, and what level of services you need. More guidance is in Apple's server documents (same guide as on NSA site), and at NSA-SNAC (checklists.nist.gov).

Adopting 64-Bit Programming

Leopard supports a 64-bit address space, and 64-bit pointers using the industry standard lp64 model. Higher level frameworks and libraries are supported in 64-bit. A process is either 32 or 64-bit. Leopard is a single install supporting apps as 4-way universal (32 and 64-bit, PowerPC and Intel architectures). New and existing drivers are compatible, and 32 and 64-bit apps can coexist.

In Tiger, 64-bit exists in libSystem and accelerate frameworks which are C/C++ based. This was useful for command line tools, but no GUI apps. There was no 64-bit Objective-C or Java. For Tiger, Apple recommends 64-bit back-end processes with 32-bit GUI.

Very low level differences between 32 and 64-bit is more substantial on Intel than on PowerPC (PowerPC was a 64-bit design from the start). This is largely related to increased number of general purpose registers and subsequent changes in processor calling conventions for 64-bit Intel.

Standard Unix conventions used, which is similar to Linux and other Unix variants. Documentation on the Unix convention is at x86-64.org.

The general 64-bit memory layout for Intel 64-bit is:

- 1) 0-4gb is kernel and 32-bit user space
- 2) 4gb to 128 tb 64-bit libraries and apps
- 3) 128 tb to 0xFFFF800000000000 is not addressable due to hardware limits
- 4) 0xFFFF800000000000 and higher is reserved for future kernel usage
(I presume it is the same or very similar for PowerPC 64-bit)

OS X picks the right architecture from each universal binary, so OS X chooses the 64-bit side on 64-bit Mac. But what if a 32/64-bit app is run on Tiger?

Apple says that if developers build the 64-bit side with OS X deployment target of Leopard, Tiger sees this and chooses the 32-bit side. Sometimes 32-bit is the right choice even for a 32/64-bit apps. This is the case if the app utilizes 32-bit plug-ins or other supporting libraries.

There are a couple of OS keys you can set. LSArchitecturePriority key which is used by launch services (default order of i386, ppc, x86-64, ppc-64) and the LSMinimumSystemVersionByArchitecture key. The Leopard Finder will allow users to set via the Finder for Leopard to force opening in 32-bit mode.

Porting tips

Make your app 64-bit today if you need access to more than 4 GB of memory, and needs every last percent of performance on Intel. Otherwise it is time to start planning, and for Carbon apps to start the transition to Cocoa GUI. A shippable 64-bit app efficiently handles more than 4 GB of data.

Porting to 64-bit can be as simple as adding the 64-bit architecture in Xcode dialog checkbox. Except that for 64-bit debug builds the architecture dialog checkbox set the architecture to "Native Arch Actual" for performance during iterative development builds. This way debug builds do not to native arch actual by default for performance. Change your build configuration to release build, which are set to Universal by default. Or edit the text field for the architecture build setting to override.

Some code conversion will likely be needed. 64-bit is not necessarily source compatible with today 32-bit apps. Apple's intent is for the same source base to generate 32 and 64-bit versions of the binary.

It is a good idea to build warning free before the port. Use additional flags to help catch conversion problems. There will be runtime crashes and iterative debugging. When looking for 64-bit dependencies, use otool -L to see a binary's direct dependencies. Use the file command in Terminal to see what architectures a binary has.

Common Pitfalls

- assuming sizeof(void*)==sizeof(int)
- mismatch in formatting characters
- failing to use sizeof()
- mixing signed/unsigned arithmetic
- failing to specify appropriate type of constant
- wrong preprocessor macro

API changes for Carbon

As mentioned at WWDC 2006, currently deprecated Carbon APIs and those which depend on those deprecated APIs (e.g. older OS 9 based QuickDraw, low-level

QuickTime C-APIs, Window Manager, Appearance Manager, Font Manager, Sound Manager, Drag Manager, ATSUI, etc.) are not supported in 64-bit. For WWDC 2007, there is the additional announcement that the Carbon UI portion (HIToolbox) is not being brought over to 64-bit. Note that non-UI lower level core service APIs used by Carbon UI (Core Foundation, Apple Events, Launch Services, etc) are 64-bit supported. 32-bit Carbon UI is still supported. There is currently no specific plan to phase out Carbon as a whole.

The news that 64-bit Carbon UI is not available is a change from WWDC 2006. Apple says it is investing heavily in features for Cocoa. Modernizing many Carbon apps is as much work as simply going to Cocoa, without the same level of benefit. Header and library changes coming to make this so. The Cocoa transition may be easier than you think. Use the highest level Cocoa API you can. This gives the best impedance match, and more insulated from API changes.

According to Apple, they hear three myths of transition to Cocoa:

1) Objective-C is scary.

Apple contends that Objective-C is easy to learn. It mixes with C++, and it gets some great new feature for Leopard.

2) Cocoa is not as cross platform as Carbon is

Apple contends your Carbon code was platform specific already, and that many apps have Cocoa GUI and C/C++ backends.

3) Cocoa is not customizable.

Apple contends that events, drawing, controls, user experience, etc. can all be managed/customized by you.

Apple provided two scenarios for how to transition to Cocoa:

1) rewrite your GUI only

Cocoa GUI lives nicely on top of a C/C++ cross platform back end. This means you have potentially less code to touch all in one step. Going 64-bit could require significant changes for the rest of your code anyway.

2) rewrite you whole app

Here you gain the development benefits of Cocoa and automatically gain future Apple enhancements to Cocoa. It may not necessarily take a long time to rewrite, but in the short term it may be more difficult to keep identical functionality with the current app version. In this case, let Apple know if critical APIs or functionality are missing.

The Carbon transition has been going on for some years now, and things have become more common with Cocoa as time has moved on. For example:

1) Both can be made Universal

2) Both use NIB files (although the format is different)

- 3) Carbon is procedural; Cocoa is object-oriented
- 4) Carbon uses QuickDraw and Quartz; Cocoa uses Quartz, NSImage, etc
- 5) Carbon uses Carbon Events; Cocoa uses delegations and notifications
- 6) Carbon uses HView; Cocoa uses NSView
- 7) Both can be packaged as App bundles
- 8) Carbon uses Core Foundation API; Cocoa uses Foundation Kit API

There are many basic paradigms that translate over from Carbon to Cocoa. Once you make the transition, you can adopt the newest technologies like Core Animation, Core Data, Sync Services, Quartz Composer, QTKit, Instant Message, Core Image, Address Book, PDF Kit, dot Mac, WebKit, Bluetooth, and more.

In summary the graphical Carbon APIs are not available, but lower level core services API are available. You should transition your UI to Cocoa, and give Apple feedback on missing API functionality.

API changes for Cocoa

The Objective-C runtime has been rewritten for improved speed, general updates, fast enumeration, garbage collection, properties, etc. Some specific Objective-C version 2 features for 64-bit include non-fragile instance vars, faster method dispatch, and an exception model unified with C++.

Basic data type

- 1) nearly all ints replaced with NSInteger and NSUInteger
- 2) enum cleanup because they are not predictably unsigned int
- 3) all graphics related floating point quantities changed from float to CGFloat, which is a double in 64-bit

NSMovieView and NSMovie are dead, so you should use QTKit

NSQuickDrawView is dead. Use Quartz.

NSMenuView is gone. Use NSMenuItem custom view support

Use the keyed archiving. Unkeyed archiving is highly discouraged.

An aggressive top script is provided to help with the conversion. You do not need to make every change it suggests, so change the script to fit your needs.

Other API changes

New CFBundle/NSBundle API for checking bundle architecture

Message framework (for sending emails through mail) not available to 64-bit, so use Apple Events to Mail to send email messages.

Java available in 64-bit for Intel only

OpenTransport is gone, so use CFNetwork or BSD sockets.

AppleTalk is gone, so use TCP/IP plus Bonjour

Printing

- 1) CFPlugin based print dialog extensions gone
- 2) Carbon QuickDraw based printing API gone
- 3) use Cocoa base printing dialog extension API
- 4) to get info from PPD file in 64-bit app, use CUPS instead of PPDLib.

QuickTime C APIs no available to 64-bit, so use QTKit. You cannot get native QuickTime identifiers in 64-bit (old QuickTime API not available).

IOKit

- 1) new and existing driver are Leopard compatible
- 2) use IODMAcommand where available to support more than 4gb physical ram
- 3) New custom user client API in Leopard to support 64-bit client apps.

In closing, this was an opportunity to remove old stuff as you transition to 64-bit. As Leopard fully enables 64-bit apps, update your code to be 64-bit clean. Transition your UI code to Cocoa if you have not already done so. Much of this work is needed anyway to adopt new Leopard technologies.

Introducing Xsan 2

Xsan is the SAN file system for OS X. In traditional file sharing data flows through a single server. In Xsan clients have direct access to data on the raid. Performance scales as you add storage, and redundant controllers are used for high availability.

Leopard server comes with a few out-of-the-box services that either require or are well suited to work with Xsan. They are the mail server cluster, iCal server cluster and Podcast Producer.

Xsan 2 is scheduled to ship in late 2007. New for Xsan 2 is a rewritten Xsan Admin tool, improved performance, MultiSAN for provisioning your Xsan controllers, and Spotlight support.

The new Xsan Admin tool emphasizes streamlined SAN setup and administration. Assistants guide you through complex tasks, along with streamlined presentation and workflow. It is asset oriented, so you can quickly find the info you want. Xsan Admin can optionally manages your users, groups, and network settings.

Among the performance improvements are shared storage optimizations for your workload. This includes pretuned volume workload settings, new data layout for streaming workloads, and more advanced settings for ever greater fine tuning control (hopefully avoid config file editing).

MultiSAN allows you to provision your Xsan controllers. This enables isolation of mission critical volumes, while retaining global volume visibility for easy bridging between SAN "islands".

Xsan 2 supports Spotlight for advanced searching for your Xsan content. Xsan 2 is fully integrated in the Spotlight search experience, and supports standard Spotlight APIs. This does require Leopard. Tiger clients cannot search, but things get indexed.

There is an Xsan 2 Developer Preview today from ADC which supports both Tiger and Leopard. There is also a seeding program available.

Xsan licenses managed as a pool (pool is dynamic, must remove from SAN to return to pool) and not as a true floating license.

Xsan 1.4 admin program not compatible with the Xsan 2 admin program. Use older admin program for older clients. Can have both admin apps on at the same time.

In principle, Xsan can be a general file server. Apple has a Xsan 2 preset, and a test group living on this idea. The preset is experimental as Apple is trying to understand this kind of scenario, versus the established creative markets.

Next Generation Automation AppleScript, Automator, and the Scripting Bridge

AppleScript is upgraded for Leopard. At the low level, changes were made to lay the foundation to move forward in the future. Specifically this includes 64-bit support, improvements in the Cocoa scripting frameworks, and Unicode support, and updated documentation so that the language guide current with release.

AppleScript now has fully integrated Unicode support. Source text is Unicode, and all internal text operations are Unicode.

For 64-bit support, AppleScript is the same as before. There are a few C APIs that are going away/changing:

- 1) OSAGetAppTerminology to OSACopyScriptingDefinition
- 2) ASGet/SetSourceStyles to ASCopy/SetSourceAttributes

Some of the updates to the scripting frameworks include:

- 1) making SDEF (mechanism to define scripting dictionary) dynamic
- 2) new APIs
- 3) better type checking, error sensing and reporting (it has useful info!)

The Application Object Model has been enhanced to provide more ways to generically address an application. The new ways include:

- 1) tell app at a POSIX path
- 2) tell app by bundle ID or signature
- 3) tell app via its intrinsic properties (version, ID, frontmost?, running?)

Additional enhancements include

- 1) Natively read and write OS X property lists (native read-only in Tiger).

- 2) Folder Actions now have their own process, and are more reliable folder.
- 3) Scriptable system preferences (Dock, Network, Accounts, etc)
- 4) More applications are scriptable (e.g. iChat)

The native messaging architecture for OS X automation is AppleEvents. Currently there are 3 options for writing AppleEvents in code.

- 1) Do some application control with an NSAppleEventDescriptor. This method is fast and efficient, but is complicated to implement.
- 2) Run scripts with NSAppleScript. Simpler, but slower, and does not scale. It does not integrate with Objective-C, and you must know AppleScript.
- 3) Edit scripts using the OSAKit

The Scripting Bridge (new for Leopard) is a new mechanism to make it easier to enable automation with AppleEvents. It recognizes that there are other scripting languages besides AppleScript. It is meant to execute fast and efficiently, and still be simple to implement. One driver for the Scripting Bridge was the problem of efficiently running AppleScripts within Unix scripts. AppleScript is still not totally headless operation due to legacy stuff.

You implementing the Scripting Bridge in two steps. First you generate header files using a simple command line utility. Second is to add the headers to your project, and link to ScriptingBridge.framework. Now you can use it in your code to get or set attributes, set or get elements, call a command, etc.

The Scripting Bridge works with Objective-C, Python via PyObjC, and Ruby with RubyCocoa. I presume other languages like Perl can be used with third party help. The existing ways for manipulating Apple Events are still available. You can mix and match scripting languages at will. This means greater access to Apple Events from a wider group of developers. This should lead to more scriptable apps, better quality dictionaries and more creative workflows.

Automator for Leopard has been significantly upgraded. Some of the more significant upgrades include:

- 1) Starting Points (i.e. Workflow assistant)

This is a step-by-step guide to help you get started.

- 2) Streamlined interface for easier access to information

- actions grouped by category instead of app
- description, variables and log windows pop in/out
- built in media picker to use media on your computer
- can hide the entire library of actions if desired
- results embedded in the action window instead of having to add the show results action all the time

3) Workflow variables (store and retrieve data)
--reusing info through the workflow
--generating generic data (paths, user and system info)
--variables can be monitored and edited via the variable window

4) UI Recording and Playback (Virtual user action)
You can create self-contained recordings which are treated as an Automator action. These recordings can involve interaction with dialogs and windows, and work with apps with no AppleScript or Automator support. This feature relies on the OS X accessibility frameworks (not all apps support accessibility equally). You need to activate support for assistive devices.

5) Automator Frameworks (Automator within Apps)
The 3 main frameworks are AMWorkflow, AMWorkflowController, and AMWorkflowView. AMWorkflow and AMWorkflowController allow you to access and manipulate workflows from within your app. AMWorkflowView enables you to put an Automator action into your UI. Apps which support Automator actions can exchange workflows with each other. You now have an app plugin architecture to the whole system.

Fortran Development and HPC on Mac OS X: 3rd-party solutions

This session featured presentations by three third party vendors: Intel, MacResearch.org, and Absoft. Weblinks are given for the vendors.

Intel Fortran Compiler, Professional Edition version 10.0
<http://www.intel.com/support/performancetools/fortran/mac/>

Integrates into Xcode ide and interoperates with gcc/gdb and supports 64-bit OS X. Fully ANSI Fortran 77,90,95,2000 compliant, with some Fortran 2003 support.

One suite of tools for high performance, computation intensive, multithreaded computing (Fortran compiler v10.0, math kernel library v9.1 (MKL), Intel debugger (IDB)). Intel software products website for a tryout

MacResearch.org
<http://www.macresearch.org/>

MacResearch is an online community for scientists using Apple hardware and software in research, and acts as an advocate for the scientific community with Apple. It provides news, reviews, stories, script repository, tutorials, etc.

MacResearch is partnering with the upcoming OpenMacForge. OpenMacForge is a free software hosting (including precompiled binaries with installer packages) and resources for developing open source scientific software on Mac OS X.

Other resources available on MacResearch:

1) OpenMacGrid

This is a community Xgrid cluster. Free use for any approved scientific app.

2) gfortran and Xcode

Up-to-date version of gfortran commissioned and maintained by MacResearch.org (open source). There is an open source plug-in to use compiler within Xcode on PPC or Intel.

3) OpenMP (www.openmp.org)

API for parallel programming (C and Fortran) with shared memory. It has been around for 10 years and now its come to the desktop.

Absoft Corporation

http://www.absoft.com/OSX_Intel.html

New Absoft Pro Fortran v10 (new product, not a port)

Commercially available 32/64-bit compiler for OS X on Intel shipping now on both Tiger and Leopard. IDE common across Mac (PPC and Intel), Windows, and Linux.

Prices starting at \$299. IMSL numerical libraries with special upgrade pricing for existing Absoft customers are available. 30 day trial version available.

Apple Design Awards 2007

<http://developer.apple.com/wwdc/ada/index.html>

http://en.wikipedia.org/wiki/Apple_Design_Awards

Summary

As always, the notes above represent only a sampling of the sessions that were available. The conference had both developer and enterprise IT tracks.