Resume and Automatic Termination

In Mac OS X Lion

Session 119

Peter Ammon
Cocoa Frameworks Engineer

David Smith
Cocoa Frameworks Engineer

Resume and Automatic Termination

- On iOS, users love how apps pick up where they left off
- The app may or may not have terminated
- Simpler application model than Snow Leopard
- Mac OS X Lion is moving in the same direction

Resume

Automatic Termination

Resume and Automatic Termination

- Resume
  - Why Resume?
  - API overview
  - Recreating open windows
  - Restoring state within windows
  - Advanced topics and best practices
- Automatic Termination
  - What is Automatic Termination?
  - Benefits of Automatic Termination
  - API overview
  - Future directions

Resume

In Mac OS X Lion

Peter Ammon
Cocoa Frameworks Engineer

Apps simply

Resume

where they left off

after quit, log out, or crash

Demo

Resume and Automatic Termination

- Resume
  - Why Resume?
  - API overview
  - Recreating open windows
  - Restoring state within windows
  - Advanced topics and best practices
- Automatic Termination
  - What is Automatic Termination?
  - Benefits of Automatic Termination
  - API overview
  - Future directions

Why Resume?

Why do I want to do this?

- Simplifying the application model
- Users will expect that the application state is not lost
- Macs will be able to silently restart without the user noticing

Why Resume?

Why should I use the Cocoa APIs for this?

- There is a lot to restore
  - A window has a frame, on a display, on a space, or maybe it's minimized, or full screen…
- It integrates with the rest of the system
  - Inter-application Z-order
  - Shift key
- It's really easy
  - Incremental adoption
  - Complements existing persistence mechanisms
  - You don't have to throw anything away

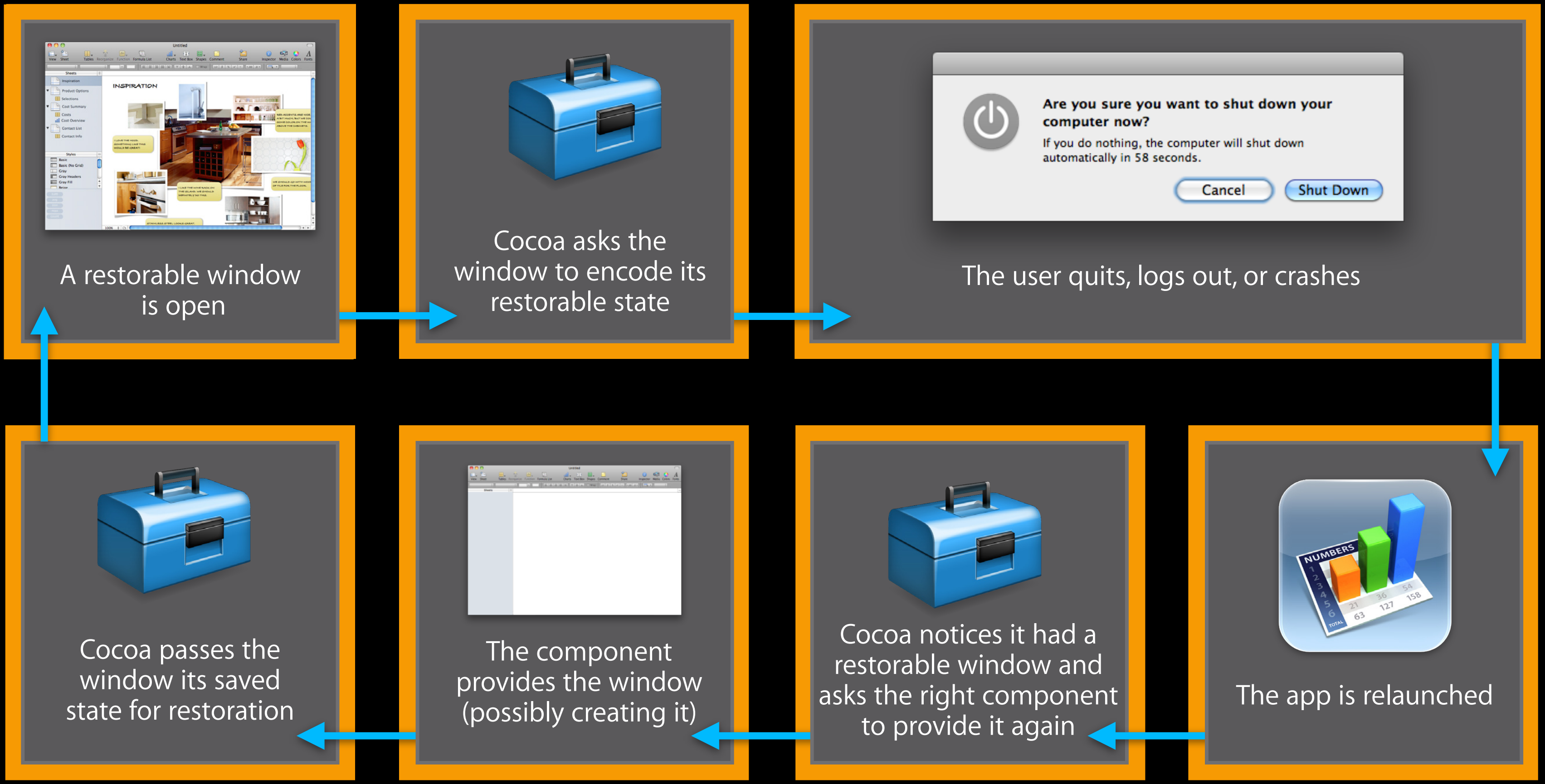
Resume and Automatic Termination

- Resume
  - Why Resume?
  - API overview
  - Recreating open windows
  - Restoring state within windows
  - Advanced topics and best practices
- Automatic Termination
  - What is Automatic Termination?
  - Benefits of Automatic Termination
  - API overview
  - Future directions

Overview of Resume

- Mac OS X has special challenges not found on iOS
    - Multiple windows
    - Documents in many locations
- There is a lot more states that an app can get into
- Additional state contributions from frameworks and plug-ins
- Here is how we meet those challenges

Overview of Resume



A restorable window is open

Cocoa asks the window to encode its restorable state

Are you sure you want to shut down your computer now?

If you do nothing, the computer will shut down automatically in 58 seconds.

The user quits, logs out, or crashes

The app is relaunched

Cocoa notices it had a restorable window and asks the right component to provide it again

The component provides the window (possibly creating it)

Cocoa passes the window its saved state for restoration

Overview of Resume

- Resume API is centered around windows
  - But support for global state, too
- Each component can take responsibility for its own windows
- Two phases
  - Recreating the windows that were previously open
  - Restoring view state within each window

Resume and Automatic Termination

- Resume
  - Why Resume?
  - API overview
  - Recreating open windows
  - Restoring state within windows
  - Advanced topics and best practices
- Automatic Termination
  - What is Automatic Termination?
  - Benefits of Automatic Termination
  - API overview
  - Future directions

Recreating Open Windows

API summary

- Mark a window as restorable

```
[window setRestorable:YES]
```

- Set the restoration class

```
[window setRestorationClass:[SomeClass class]]
```

- Implement the restore method

```
+ (void)restoreWindowWithIdentifier:(NSString *)identifier
        state:(NSCoder *)state
        completionHandler:(void (^)(NSWindow *, NSError *)handler
```

Recreating Open Windows

API summary

```
+ (void)restoreWindowWithIdentifier:(NSString *)identifier
     state:(NSCoder *)state
     completionHandler:(void (^)(NSWindow *, NSError *)handler
```

- Invoke the completion handler with the corresponding window
- The identifier is an easy way to distinguish between different windows restored by the same class
    - Settable in IB
- The state parameter can be used to track even more information necessary for recreating the window
    - More on that later

Restoring Windows

NSFontPanel example

```
[fontPanel setRestorable:YES];
[fontPanel setRestorationClass:[fontPanel class]];



+ (void)restoreWindowWithIdentifier:(NSString *)identifier
        state:(NSCoder *)state
        completionHandler:(void (^)(NSWindow *, NSError *)handler {
           handler([self sharedFontPanel], NULL);
       }
```

This window may already exist!

Restoring Windows

NSDocument integration

- NSDocument sets its windows' restoration class to the NSDocumentController
- NSDocumentController reopens windows by reopening their documents
- Customizable hooks

```
@implementation MyDocumentController
- (void)restoreDocumentWindowWithIdentifier:(NSString *)identifier
        state:(NSCoder *)state
        completionHandler:(void (^)(NSWindow *, NSError *))handler
@end
```

Demo

Restoring Windows

Which windows should be restorable?

- Most windows should be restorable, with some exceptions
- Transient windows
  - Tooltips, shielding windows
- Windows the user does not want to restore
  - Private browsing in Safari
- Windows whose job is done
  - “Install Complete” window
- Windows that the app cannot restore (yet)

Resume and Automatic Termination

- Resume
  - Why Resume?
  - API overview
  - Recreating open windows
  - Restoring state within windows
  - Advanced topics and best practices
- Automatic Termination
  - What is Automatic Termination?
  - Benefits of Automatic Termination
  - API overview
  - Future directions

Restoring State Within Windows

- Each component within the window has its own private state
  - NSView, NSWindow, NSWindowController, NSDocument
  - NSApplication, too
- The component invalidates its state whenever that state changes

```
[self invalidateRestorableState]
```

$\leftarrow$ Fast and inexpensive

- At some point later, the component will be asked to encode its state

```
- (void)encodeRestorableStateWithCoder:(NSCoder *)coder
```

- Upon relaunch, the component will be given its state to restore

```
- (void)restoreStateWithCoder:(NSCoder *)coder
```

Restoring State Within Windows

- Two useful NSWindow delegate methods

```
– (void)window:(NSWindow *)window willEncodeRestorableState:(NSCoder *)coder
– (void)window:(NSWindow *)window didDecodeRestorableState:(NSCoder *)coder
```

- Easy way to do state restoration without subclassing

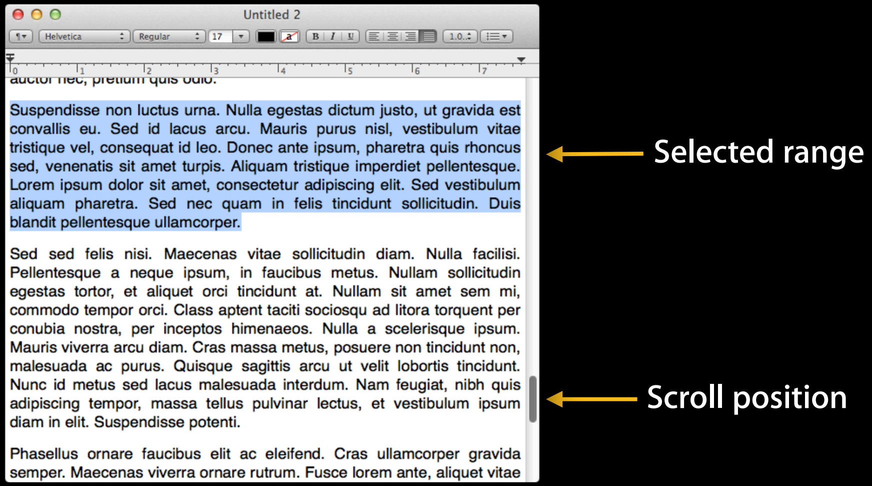
- Useful NSResponder class method

```
+ (NSArray *)restorableStateKeyPaths;
```

- Automatic state restoration of KVO-compliant properties

Restoring State Within Windows

- What state should be restored via this mechanism?
- Restore view and controller state
- Not model state



auctor nec, pretium quis odio.

Suspendisse non luctus urna. Nulla egestas dictum justo, ut gravida est convallis eu. Sed id lacus arcu. Mauris purus nisl, vestibulum vitae tristique vel, consequat id leo. Donec ante ipsum, pharetra quis rhoncus sed, venenatis sit amet turpis. Aliquam tristique imperdiet pellentesque. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed vestibulum aliquam pharetra. Sed nec quam in felis tincidunt sollicitudin. Duis blandit pellentesque ullamcorper.

Sed sed felis nisi. Maecenas vitae sollicitudin diam. Nulla facilisi. Pellentesque a neque ipsum, in faucibus metus. Nullam sollicitudin egestas tortor, et aliquet orci tincidunt at. Nullam sit amet sem mi, commodo tempor orci. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Nulla a scelerisque ipsum. Mauris viverra arcu diam. Cras massa metus, posuere non tincidunt non, malesuada ac purus. Quisque sagittis arcu ut velit lobortis tincidunt. Nunc id metus sed lacus malesuada interdum. Nam feugiat, nibh quis adipiscing tempor, massa tellus pulvinar lectus, et vestibulum ipsum diam in elit. Suspendisse potenti.

Phasellus ornare faucibus elit ac eleifend. Cras ullamcorper gravida semper. Maecenas viverra ornare rutrum. Fusce lorem ante, aliquet vitae

- Beware that model and view state may be out of sync!

Demo

Restoring Windows

Complex cases

- My window needs to know its state before I can even create it!
  - Multiple types of windows or documents

```
+ (void)restoreWindowWithIdentifier:(NSString *)identifier
        state:(NSCoder *)state
        completionHandler:(void (^)(NSWindow *, NSError *)handler;
```

- Combined restorable state of the window, its window controller, and document
- If you can not restore the window, invoke the completion handler with nil
  - But always invoke it

Resume and Automatic Termination

- Resume
    - Why Resume?
    - API overview
    - Recreating open windows
    - Restoring state within windows
    - Advanced topics and best practices
- Automatic Termination
    - What is Automatic Termination?
    - Benefits of Automatic Termination
    - API overview
    - Future directions

Advanced Topics

Inter-view references

- How NSWindow records its first responder:

```
[coder encodeObject:firstResponder forKey:@"NSFirstResponder"]
```

- Encoding an NSResponder archives a reference to it
- Decoding it returns an existing NSResponder, never a new one
- References can cross windows
    - Example: Color and Font panels encode their targets

Advanced Topics

Sandboxed apps

- Encode URLs to files in any restorable state NSCoder
- You automatically get permission to reopen them
  - Even if they have been moved or renamed
  - URL encoding uses bookmarks
- Or use NSDocument

Advanced Topics

Case study: Mail

- Mail already restored its state via state from NSUserDefaults
- Integrating with Resume was ~ 60 LOC
- Idea
  - Gave each window a unique ID
  - Recorded it in both the user default record and the restorable state
  - Restored windows via the existing NSUserDefaults mechanism
  - In the Resume callback
    - Decode the unique ID from the restorable state
    - Find an existing window with that unique ID
    - Invoke the completion handler with that window (or nil)

Best Practices

- Do not assume that if your app is being launched, the user intends to use it immediately
    - Running applications get relaunched at login
    - Your app can be relaunched in the background
    - Splash screens, demanding dialogs, etc. will be perceived as annoying

Best Practices

- Do not create default windows in applicationDidFinishLaunching:
  - Your app may already have restored windows
    - N+1 effect
  - Prefer applicationOpenUntitledFile:
  - One exception: One-window apps like iPhoto

Best Practices

- Be prepared for unexpected changes
  - Screen size, preferences, etc. may change
  - File contents too (remember iCloud)
- Validate that all saved state still makes sense
- Do not forget versioning

Best Practices

- Partial state restoration is OK
  - (Though higher fidelity is obviously better)
- The more you use Cocoa, the less work it will take
  - Example: full screen
  - But if you already restore state, it’s easy to integrate with Resume

Resume Summary

- Resume is about state restoration
- Cocoa tracks the open restorable windows, and asks their restoration classes to recreate them on relaunch
- Each component can recreate the windows it is responsible for
- Each NSResponder can encode and restore its own private state

Automatic Termination

In Mac OS X Lion

David Smith
Cocoa Frameworks Engineer

Resume and Automatic Termination

- Resume
  - Why Resume?
  - API overview
  - Recreating open windows
  - Restoring state within windows
  - Advanced topics and best practices
- Automatic Termination
  - What is Automatic Termination?
  - Benefits of Automatic Termination
  - API overview
  - Future directions

What Is Automatic Termination?

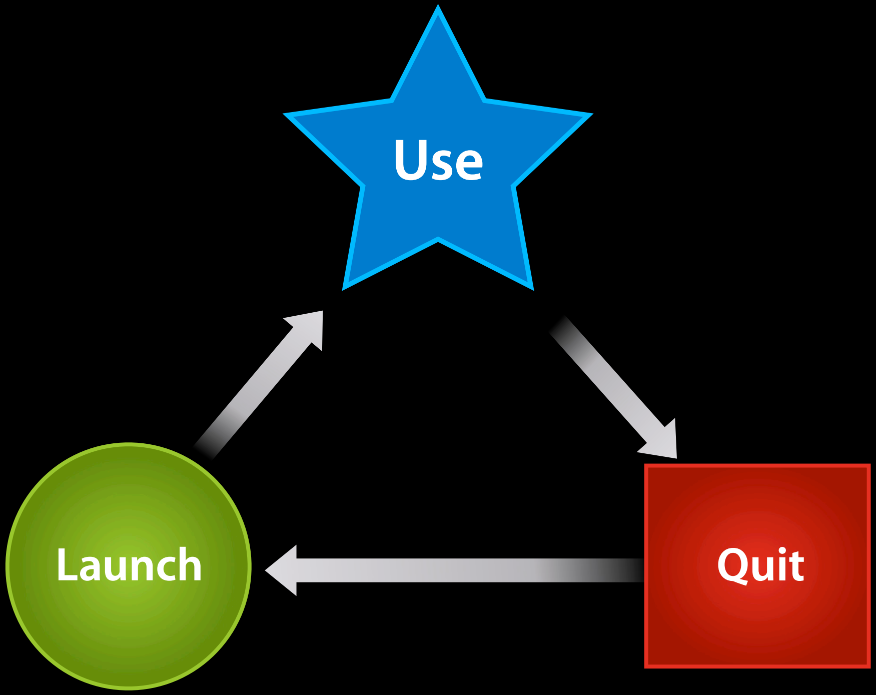
- Decouples apps from processes
  - Open apps might have a process
  - Closed apps might not have a process

What Is Automatic Termination?

- Decouples apps from processes
  - Open apps might have a process
  - Closed apps might not have a process
- A new user model
  - Lets users focus on using apps instead of managing them

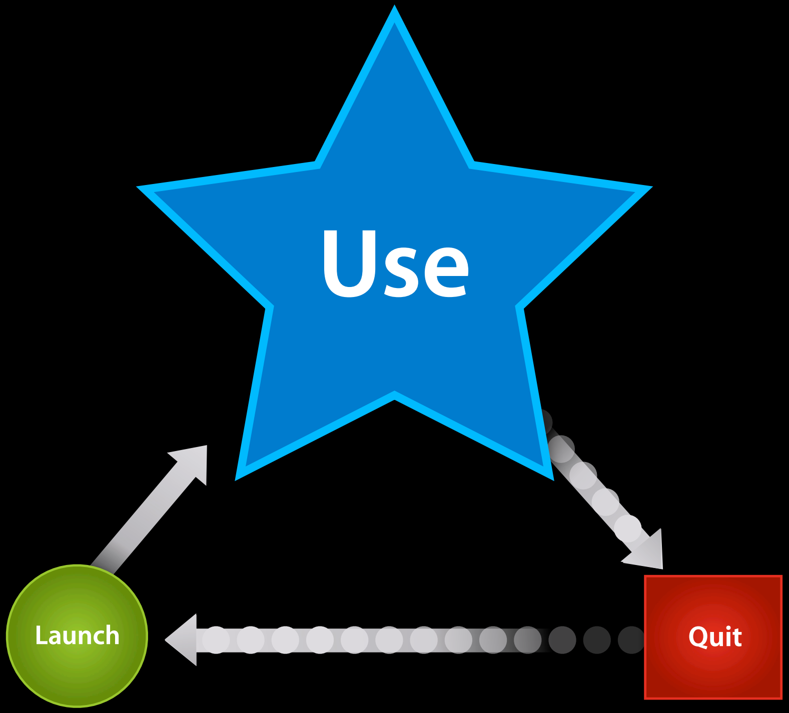
The Application Life Cycle

circa 3000BC (Snow Leopard)



The Application Life Cycle

circa 2011 (Lion)



What Is Automatic Termination?

- Decouples apps from processes
  - Open apps might have a process
  - Closed apps might not have a process
- A new user model
  - Lets users focus on using apps instead of managing them
- iOS-style memory reclamation on the Mac
- Instant relaunch

Resume and Automatic Termination

- Resume
    - Why Resume?
    - API overview
    - Recreating open windows
    - Restoring state within windows
    - Advanced topics and best practices
- Automatic Termination
    - What is Automatic Termination?
    - Benefits of Automatic Termination
    - API overview
    - Future directions

Benefits of Automatic Termination

- Play well with others
- Meet user expectations for new apps
- Relaunch instantly
- Get ready for the future

Resume and Automatic Termination

- Resume
  - Why Resume?
  - API overview
  - Recreating open windows
  - Restoring state within windows
  - Advanced topics and best practices
- Automatic Termination
  - What is Automatic Termination?
  - Benefits of Automatic Termination
  - API overview
  - Future directions

API Overview

- Simple API following the same pattern as Sudden Termination from Snow Leopard

An Important Aside…

- Automatic Termination is not Sudden Termination!
- An application can participate in one, both, or neither
    - Participating in both gets you nice benefits

API Overview

- Simple API following the same pattern as Sudden Termination from Snow Leopard
- Master “on” switch API
    - Set the key “NSSupportsAutomaticTermination” in your Info.plist

```
<key>NSSupportsAutomaticTermination</key>
<true/>
```

    - OR use NSProcessInfo

```
[[NSProcessInfo processInfo] setAutomaticTerminationSupportEnabled: YES];
```

API Overview

- Simple API following the same pattern as Sudden Termination from Snow Leopard
- Temporarily opt out when your app is working…

```
[[NSProcessInfo processInfo] disableAutomaticTermination:@”Reason”];
```

- …and return control to the system when you’re done

```
[[NSProcessInfo processInfo] enableAutomaticTermination:@”Reason”];
```

Running Applications with No Process

- When these criteria are met:
  - No visible windows
  - All open windows are restorable
  - Not the active app
  - No outstanding -disableAutomaticTermination: calls
  - The system is out of available memory
- The kernel may terminate the app's process
  - The app will appear to still be running, and will relaunch transparently if needed

Processes with No Running Application

- When these criteria are met:
  - No open windows
  - Not the active app
  - At least one window has ever been open
  - No outstanding -disableAutomaticTermination: calls
- The app will appear to quit, but its process will remain running
  - This lets it relaunch instantly
  - It will terminate if the system needs to reclaim its resources

Demo

Testing Automatic Termination in Your App

- Simulate system memory pressure to verify expected behavior
    - Use /System/Library/CoreServices/talagent -memory_pressure
    - Only use this for testing! It is not guaranteed to exist

Recap: Adopting Automatic Termination

- Turn it on – Info.plist key or API

```
<key>NSSupportsAutomaticTermination</key>
<true/>
OR
[[NSProcessInfo processInfo] setAutomaticTerminationSupportEnabled: YES];
```

- Wrap activity in paired -disable/-enable calls

```
[[NSProcessInfo processInfo] disableAutomaticTermination:@”Reason”];
[[NSProcessInfo processInfo] enableAutomaticTermination:@”Reason”];
```

- Test with simulated memory pressure

```
/System/Library/CoreServices/talagent –memory_pressure
```

Resume and Automatic Termination

- Resume
  - Why Resume?
  - API overview
  - Recreating open windows
  - Restoring state within windows
  - Advanced topics and best practices
- Automatic Termination
  - What is Automatic Termination?
  - Benefits of Automatic Termination
  - API overview
  - Future directions

Imagine a World…

…that has no “Quit” menu item

…with no need to know if an app is running

…in which your parents never call and say, “My computer is slow”

More Information

Bill Dudney
Application Frameworks Evangelist
dudney@apple.com

Documentation
Mac OS X Dev Center
http://developer.apple.com/devcenter/mac

Apple Developer Forums
http://devforums.apple.com

Related Sessions

What’s New in Cocoa	Presidio Tuesday 10:15AM
Auto Save and Versions in Mac OS X 10.7 Lion	Pacific Heights Tuesday 3:15PM

Labs

Cocoa, Auto Save, File Coordination, and Resume Lab	App Frameworks Lab A Thursday 2:00-4:00PM

WWDC2011