Crafting Custom Cocoa Views

Building your own user interface elements

Troy Stephens
Cocoa Frameworks Engineer

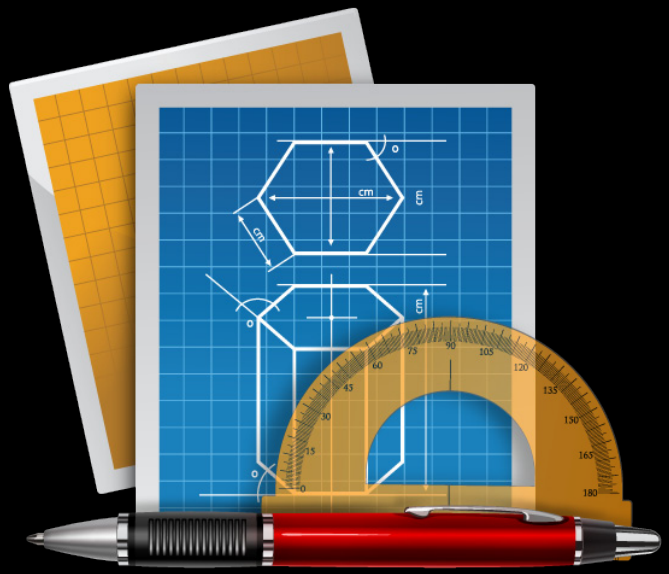
Introduction

- Mac OS X frameworks provide many standard controls, but…
- Sometimes you need to invent…something new
- How to do the job right?
  - *A checklist sure would be handy!*

Take Home Info

- Custom view implementor's checklist
- New code sample

Craftsmanship



- Attention to detail
- Worth doing = Worth doing well
- Robustness
- Functional completeness
- Simplicity + Power = Elegance
- Use-appropriate design

Crafting Views

The basics

- Layout
- Drawing
- Event-handling
  - Keyboard
  - Mouse
  - Trackpad, tablet, etc.
- Accessibility
- Support standard system features

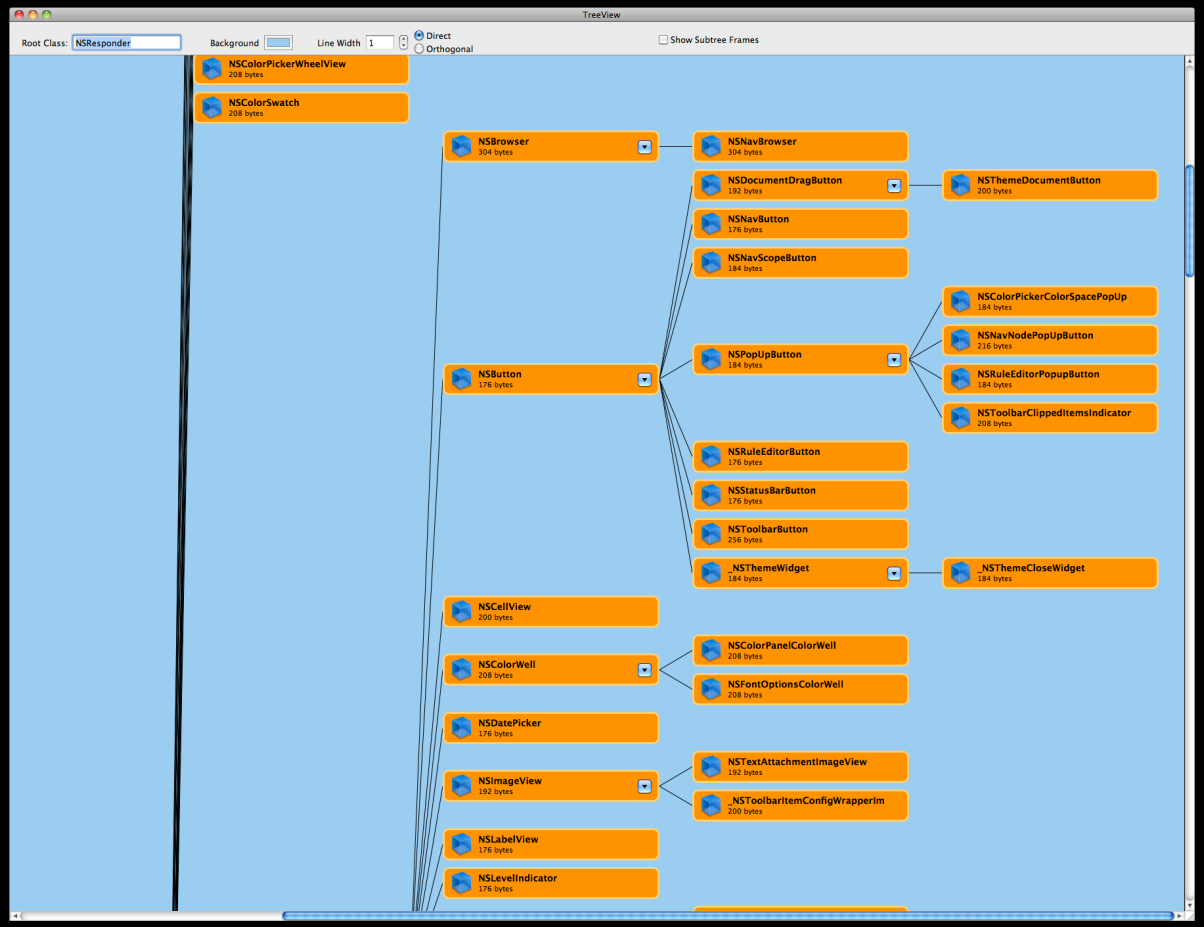
Crafting Views

Refinements

- Appearance
- Planning for animation
- Responsiveness and scalability

Today's Code Sample

TreeView


TreeView
Root Class: NSResponder
Background
Line Width 1
Direct
Orthogonal
Show Subtree Frames
NSColorPickerWheelView 208 bytes
NSColorSwatch 208 bytes
NSBrowser 304 bytes
NSNavBrowser 304 bytes
NSDocumentDragButton 192 bytes
NSThemeDocumentButton 200 bytes
NSNavButton 176 bytes
NSNavScopeButton 184 bytes
NSColorPickerColorSpacePopUp 184 bytes
NSNavNodePopUpButton 216 bytes
NSPopUpButton 184 bytes
NSButton 176 bytes
NSRuleEditorPopupButton 184 bytes
NSToolbarClippedItemsIndicator 208 bytes
NSRuleEditorButton 176 bytes
NSStatusBarButton 176 bytes
NSToolbarButton 256 bytes
_NSThemeWidget 184 bytes
_NSThemeCloseWidget 184 bytes
NSCellView 200 bytes
NSColorWell 208 bytes
NSColorPanelColorWell 208 bytes
NSFontOptionsColorWell 208 bytes
NSDatePicker 176 bytes
NSImageView 192 bytes
NSTextAttachmentImageView 192 bytes
_NSToolbarItemConfigWrapperIm 200 bytes
NSLabelView 176 bytes
NSLevelIndicator 176 bytes

Demo

TreeView
Presenting Tree-Graph Structures
developer.apple.com/wwdc/sessions/details/?id=141

Major Topic Areas

- Designing for animation
- Drawing
- Handling state changes
- Handling interaction

Designing for Animation

Designing for Animation

- Factor content to minimize redraw and relayout during animations
- Consider both layer-backed and window-backed operation
  - Be robust to backing layer tree construction and teardown
  - Leverage `layerContentsRedrawPolicy` New
  - Share/reuse repeated content efficiently
- Plan for scalability

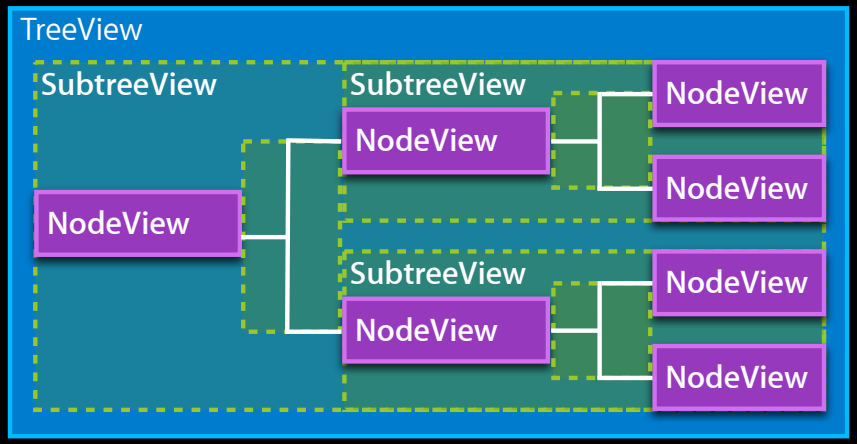
Views or Non-Views?

How to factor your content?

- "Non-View Objects" = your own NSObjects/CALayers/etc.
- Different performance implications for layer-backed *vs.* window-backed
- Non-views *can sometimes* be more lightweight, in terms of
  - Memory usage
  - CPU usage
- *However,* factoring as views has benefits in layer-backed mode
  - Caching of content in separate parts
  - Animation versatility, `[view animator]` move/resize
- Views also provide culling, event-handling, and Accessibility benefits

TreeView Design Choices

- Nested View Subtrees
  - Groups subtrees logically
  - Simplifies relayout animation
  - Caches content when layer-backed



Designing for Animation

- Make your custom view properties animatable where appropriate
  - Override `+defaultAnimationForKey:`

```
+ (id)defaultAnimationForKey:(NSString *)key {
    if ([key isEqualToString:@"borderColor"] ||
        [key isEqualToString:@"borderWidth"]) {
        return [CABasicAnimation animation];
    } else {
        return [super defaultAnimationForKey:key];
    }
}
```

  - Enables use of "animator" syntax with your custom property

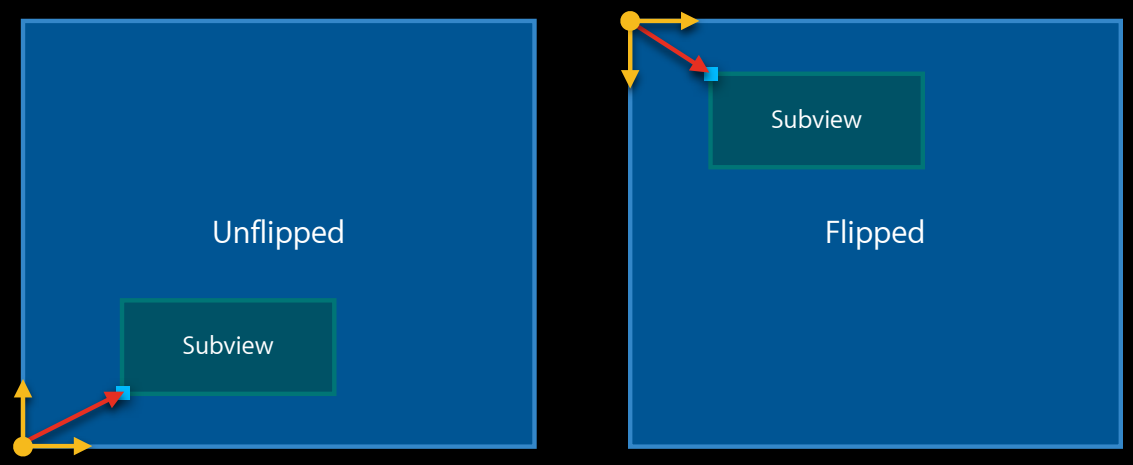
```
[[view animator] setBorderColor:[NSColor blueColor]];
```

Drawing

To Flip, or Not to Flip?

Overriding -isFlipped

- Determines origin and y-axis direction of your bounds (interior)
- Determines meaning/interpretation of your subviews' frame origin



- Nonrecursive (unlike CALayer's "geometryFlipped")

To Flip, or Not to Flip?

How to decide?

- Think about the natural growth direction for your content
- Choose accordingly
- Mostly a matter of convenience
  - Which convention enables you to write simpler code?
- Affects pinning, if the documentView of an NSScrollView

Drawing
(and layout)

- Your basic responsibility: Override `–drawRect:` to draw your content
- Draw *only what you need to*
  - Test for intersection with the NSRect passed to `–drawRect:`
  - Use `–needsToDrawRect:` and/or `–getRectsBeingDrawn:count:`
- Invalidate *only what you need to*
  - Use `–setNeedsDisplayInRect:`

    in preference to `–setNeedsDisplay:`
- Be careful to invalidate the views that actually draw the affected content
  - Important in layer-backed mode!

Layout

Positioning your content and subviews

- Consider using `-viewWillDraw`
  - Allows resizing, addition, and removal of subviews just before draw time
  - If you perform your layout this way, make sure view needs display whenever layout is needed
  - Always call up to `[super viewWillDraw]` (before, after, or in the middle of doing your work)

Opaque View Optimization

Overriding -isOpaque

- Returns `NO` by default
- Override to return `YES` if your view guarantees to cover its entire bounds rectangle with 100% opaque fill
- If your view `isOpaque`, but its `alphaValue < 1.0`, AppKit still does the right thing

Geometry Calculations

- Use compatible units!
- Do the necessary conversions between views, to get compatible values

```
-convertPoint:fromView: nil -> window
-convertPoint:toView:

-convertSize:fromView:
-convertSize:toView:

-convertRect:fromView:
-convertRect:toView:
```

Geometry Calculations

- Perform pixel alignment in "base" space
  - Yields appropriate results for both layer-backed and window-backed operation

```
-convertPointToBase:
-convertPointFromBase:

-convertSizeToBase:
-convertSizeFromBase:

-convertRectToBase:
-convertRectFromBase:
```

Handling Printing (or PDF Output) Specially

Modifying your -drawRect: Method's behavior

```
- (void)drawRect:(NSRect)rect {

    // Draw background fill color only if we're not printing.
    if ([NSGraphicsContext currentContextDrawingToScreen]) {
        [[self backgroundColor] set];
        NSRectFill(rect);
    }

    ...

}
```

Handling State Changes

Be prepared!

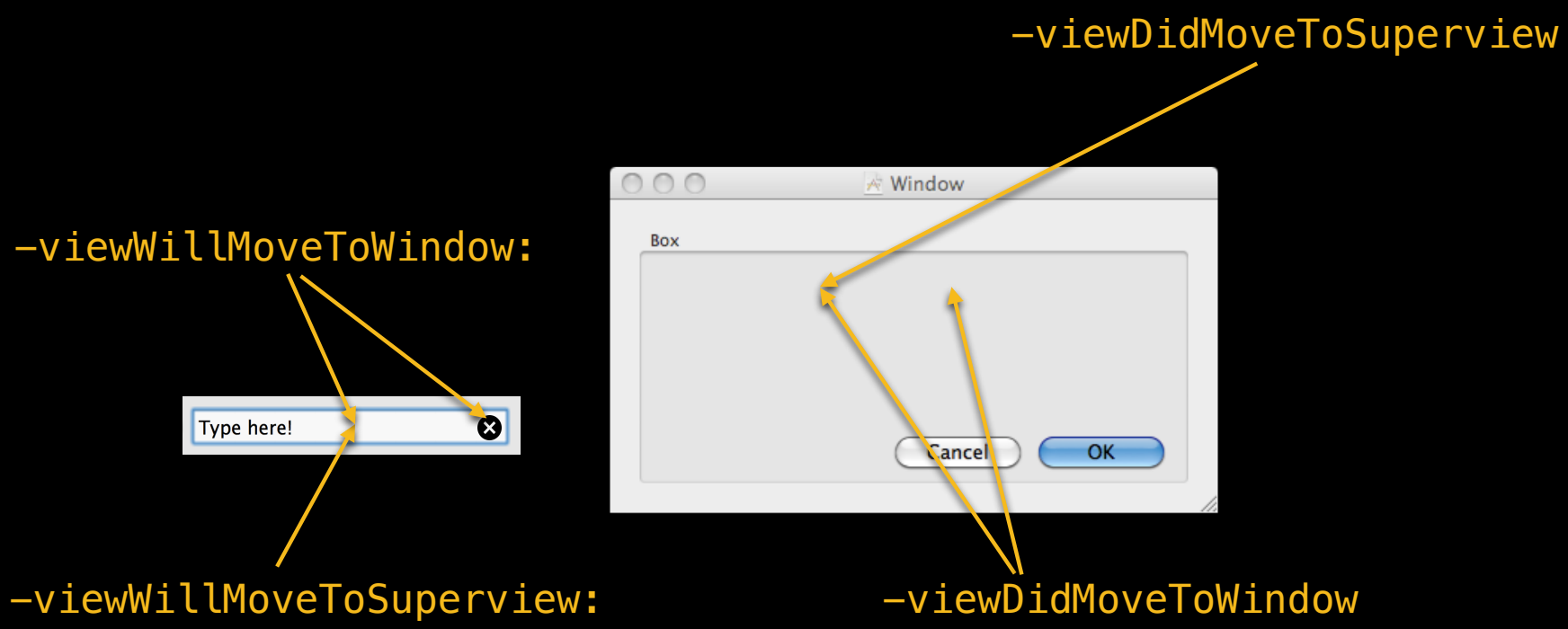
Entering/Exiting Layer-Backed Mode

If you need to react to this, override `-setLayer:`

```
- (void)setLayer:(CALayer *)newLayer {
    [super setLayer:newLayer];
    if (newLayer != nil) {
        // Becoming layer-backed, or
        // just getting a different
        // layer.
    } else {
        // Leaving layer-backed mode.
    }
}
```

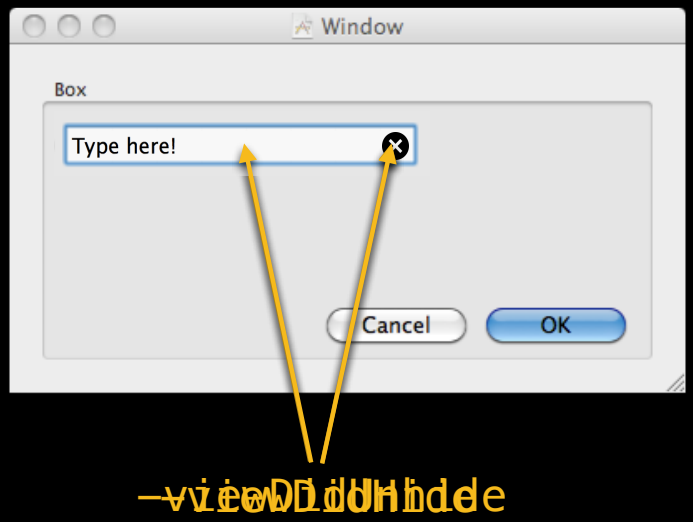
Addition/Removal

…from a superview or window



Being Hidden/Unhidden

Affects entire subtrees



Becoming/Resigning firstResponder

…in the application's keyWindow

NSWindowDidBecomeKeyNotification

NSWindowDidResignKeyNotification

-acceptsFirstResponder ?

-becomeFirstResponder

-resignFirstResponder

Being Resized

- Can override `-setFrameSize:`
  - Always call up to super
- Can override `-resizeWithOldSuperviewSize:` and `-resizeSubviewsWithOldSize:`
  - Make sure `autoresizesSubviews` is on, if you want to receive these
  - Good practice to call up to super

Being Archived/Unarchived

Enabling use in .xib/.nib files, and copying

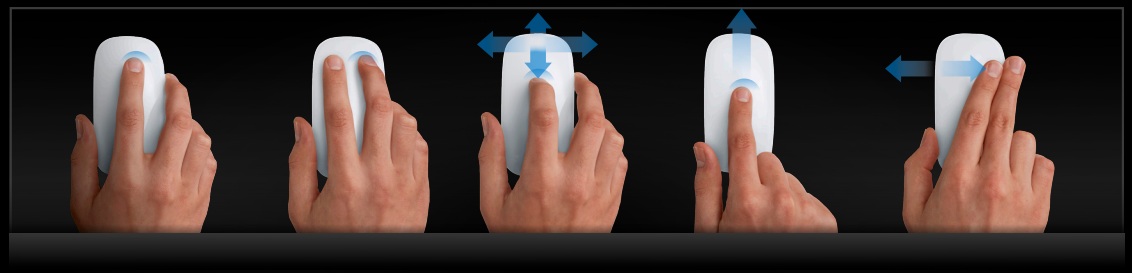
```
- (void)encodeWithCoder:(NSCoder *)coder {
    [super encodeWithCoder:coder];
    if ([coder allowsKeyedCoding]) {
        [coder encodeObject:borderColor forKey:@"borderColor"];
        [coder encodeFloat:cornerRadius forKey:@"cornerRadius"];
        ...
    }
}

- (id)initWithCoder:(NSCoder *)decoder {
    self = [super initWithCoder:decoder];
    if (self) {
        if ([decoder allowsKeyedCoding]) {
            borderColor = [[decoder decodeObjectForKey:@"borderColor"] retain];
            cornerRadius = [[decoder decodeFloatForKey:@"cornerRadius"] retain];
            ...
        }
    }
    return self;
}
```

Handling Interaction

Input/Event Sources

Keyboard



Mouse

Tablet
Pressure sensitivity, erase

Accessibility

Trackpad
Gestures and multitouch events

Supporting Accessibility

- Enables assistive device access for users with disabilities
- Provides for automated user interface testing

Making a Custom View Accessible

- Expose your view to Accessibility
  - Expose substructure to Accessibility (e.g., ContainerView)
- Specify an appropriate NSAccessibility role for your view
- Return appropriate NSAccessibility attribute values for the role
- Support setting attribute values and actions for the appropriate role

Handle Keyboard Input

- If you want key events, ask to accept them
  - Override `–acceptsFirstResponder` to return YES
- Override `–keyDown:`, and optionally `–keyUp:`, to handle key events
- Interesting NSEvent properties are
  - `characters`
  - `charactersIgnoringModifiers`
  - `modifierFlags`
  - `isARepeat`
- Pass any key events you don't handle to super

Handle Keyboard Input

- Might want to respond to changes in modifier key state
  - Override `-flagsChanged:`
  - Inspect the event's `modifierFlags`

Handle Keyboard Input

- Make your parts keyboard-navigable; support arrows, etc.
- Indicate you are focused when firstResponder in keyWindow
  - Use `NSSetFocusRingStyle()` to draw focus around a shape
  - Use `-setKeyboardFocusRingNeedsDisplayInRect:` instead of `-setNeedsDisplayInRect:` when your view is showing a focus ring
    - Important because focus rings can spill outside clip

```
- (void)drawRect:(NSRect)rect {
    NSWindow *window = [self window];
    if ([window firstResponder] == self && [window isKeyWindow]) {
        [NSGraphicsContext saveGraphicsState];
        NSSetFocusRingStyle(NSFocusRingOnly);
        [[NSColor whiteColor] set];
        NSRectFill([self bounds]);
        [NSGraphicsContext restoreGraphicsState];
    }
    ...
}
```

Handling Mouse Button Events

Main mouse button

-mouseDown:

-mouseDragged: → -mouseDragged:

-mouseUp:

Handling Mouse Button Events

```
-rightMouseDown:                                  -rightMouseUp:
        -rightMouseDragged:──→ -rightMouseDragged:
```

Handling Mouse Button Events

Other mouse buttons…

```
-otherMouseDown:                                  -otherMouseUp:
        -otherMouseDragged:──→ -otherMouseDragged:
```

Handling Mouse Events

Main NSEvent Properties of Interest

- buttonNumber
- clickCount
- modifierFlags
- locationInWindow

```
- (void)mouseDown:(NSEvent *)theEvent {
    NSPoint viewPoint =
        [self convertPoint:[theEvent locationInWindow]
                  fromView:nil];
```

Handling Mouse Movement

Mouse motion

```
-mouseMoved:
```

```
[window setAcceptsMouseMoved:YES]
```

Handling Mouse Movement

Consider other ways of responding

- Tool tips
- Cursor rects
- Tracking areas

Handle Gestures and Touch Events

- Gestures
  - Don't need to "opt in"
  - Just override one or more of
    - `-magnifyWithEvent:`
    - `-rotateWithEvent:`
    - `-swipeWithEvent:`
  - Might also be interested in
    - `-beginGestureWithEvent:`
    - `-endGestureWithEvent:`

Handle Gestures and Touch Events

- Touch events
  - More complex, but arbitrarily powerful
  - Opt in using
    - `-setAcceptsTouchEvents:`
    - `-setWantsRestingTouches:`
  - Override all of
    - `-touchesBeganWithEvent:`
    - `-touchesMovedWithEvent:`
    - `-touchesEndedWithEvent:`
    - `-touchesCancelledWithEvent:`
  - Important: Always call up to super!

Handling Tablet Input

Interaction with Inking

- By default, a pen down over your view can start inking
- If you want to handle pen events, override `-shouldBeTreatedAsInkEvent:`
  - Return `NO` when you want to suppress inking
  - NSControl default is `NO`
  - NSView default is `YES`

Handle Tablet Input

- Look for special Tablet Events
  - `-tabletProximity:`
  - `-tabletPoint:`

Handle Tablet Input

- Interesting tablet NSEvent properties
  - `locationInWindow`
  - `absoluteX`, `absoluteY`, `absoluteZ`
  - `pressure`
  - `rotation`
  - `tilt`
  - `tangentialPressure`
  - `buttonMask`
  - `isEnteringProximity`

Handle Tablet Input

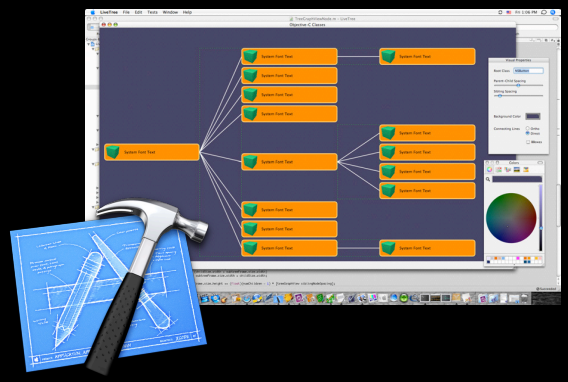
- Interesting tablet NSEvent properties
  - `pointingDeviceType`
  - `pointingDeviceID`
  - `capabilityMask`
  - `vendorID`
  - `tabletID`
  - `systemTabletID`
  - `vendorPointingDeviceType`
  - `pointingDeviceSerialNumber`
  - `uniqueID`

Major Topic Areas Covered

- Designing for animation
- Drawing
- Handling state changes
- Handling interaction

Take-Home Thoughts

- With attention to detail and thought put into the design, you can craft robust, polished custom views that fit naturally in the Mac UI
- Refer to the TreeView code sample, and accompanying checklist, for ideas when working on your own custom views



developer.apple.com/wwdc/sessions/details/?id=141

Related Documentation

developer.apple.com

- Cocoa View Programming Guide
- Cocoa Accessibility Guide
- Cocoa Drawing Guide
- Cocoa Event-Handling Guide

Related Sessions

Key Event Handling in Cocoa Applications	Russian Hill Friday 11:30AM
Usable by Everybody: Design Principles for Accessibility on Mac OS X	Nob Hill Tuesday 9:00AM
Cocoa Tips and Tricks	Marina Tuesday 2:00PM
API Design for Cocoa and Cocoa Touch	Marina Thursday 4:30PM

More Information

Bill Dudney

Application Frameworks Evangelist
dudney@apple.com

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

WWDC10