CS193E
Lecture 13

More Cocoa Text
Drag and Drop
Inspectors

Today's Topics

- Questions on the Personal Timeline III assignment?
- Announcements
  - Typo in last lecture's slides
  - Undo behavior and copy/cut/paste
  - Final projects and Leopard
- More Cocoa Text
- Drag-and-Drop
- Inspectors

More Cocoa Text

The field editor
Using a cell to edit

Other Options?

- What if NSTextField isn't enough (or appropriate) and NSTextView is too much?
  - Example: you want to add a text shape to a draw app or edit the node value in a tree?
- You can use pieces of each and do it manually:
  - NSCell: Display and edit text
  - Field Editors: General purpose NSTextView

NSCell

- Mechanism for displaying text and images in views without full NSView overhead
- Useful for displaying text
  - Similar to NSString drawing APIs, but…
- Adds API to support editing

Drawing With a Cell

- Your NSView subclass has a value it wants to display:

```
- (void)drawRect:(NSRect)dirtyRect
{
    NSRect bounds = [self bounds];
    NSCell *cell;

    cell = [[NSCell alloc] initTextCell:string];
    [cell setBordered:YES];
    [cell setFont:[NSFont userFontOfSize:48.0]];
  [cell setStringValue:@”Hello World”];

    [cell drawWithFrame:bounds inView:self];

  [cell release];
}
```

Editing With a Cell

- Relatively simple conceptually, can be a slippery (and frustrating) slope if things go wrong
- Basic approach is:
    1. Configure cell with your value
    2. Tell it to start editing session
    3. Wait until editing session ends
    4 . Retrieve the new value from the cell

1. Configuring A Cell

- Usually the same as when you're drawing

```
NSCell *cell;

cell = [[NSCell alloc] initTextCell:string];
[cell setBordered:YES];
[cell setFont:[NSFont userFontOfSize:48.0]];
[cell setStringValue:@"Hello World"];
```

- Could use NSTextFieldCell instead of NSCell if you need to set text or background color (or need other NSTextFieldCell functionality)

2. Tell It To Start Editing

- Similar to drawing, but with some extra parameters:

```
- (void)editWithFrame:(NSRect)frame
              inView:(NSView *)view
              editor:(NSText *)textObject
              delegate:(id)delegate
              event:(NSEvent *)event;
```

- Where do all these arguments come from?!?

2. Tell It To Start Editing

```
- (void)editWithFrame:(NSRect)frame inView:(NSView *)view
        editor:(NSText *)textObject delegate:(id)delegate
        event:(NSEvent *)event;
```

- Usually invoked from NSView's mouseDown: method in response to a user click (or double click) event
- frame is probably the same as where it was drawn
- textObject is the text object that will actually do the editing

Field Editors

- Every window has a general purpose NSText object tucked in its back pocket
- It's the text editor that NSTextFields use
- Anybody can ask the window for its "field editor"

```
- (NSText *)fieldEditor:(BOOL)create forObject:(id)object;
```

2. Tell It To Start Editing

```
- (void)editWithFrame:(NSRect)frame inView:(NSView *)view
        editor:(NSText *)textObject delegate:(id)delegate
        event:(NSEvent *)event;
```

- textObject is often the window's field editor
- delegate is often the view itself
  - automatically set up as the textObject's delegate
- The delegate methods are how you hear about the editing session ends

3. Wait Until Editing Ends

- Implement (at least) one of the NSTextView delegate methods:

```
- (BOOL)textShouldEndEditing:(NSText *)textObject;
- (void)textDidEndEditing:(NSNotification *)note;
```

- When editing ends (for whatever reason) the delegate is told
- After the delegate is told, field editor is torn down

3. Wait Until Editing Ends

- Editing can end for a wide variety of reasons
  - User hit return in the field editor
  - User clicked into a different view which changed first responder
  - You (or any other view) can manually force editing to end

4. Retrieve New Value

- Retrieve the value from the field editor

```
- (NSString *)string;
```

- Be sure to make a copy of the string instead of directly retaining it!
- Tell the cell that editing is over using:

```
- (void)endEditing:(NSText *)textObject;
```

passing the field editor as the argument

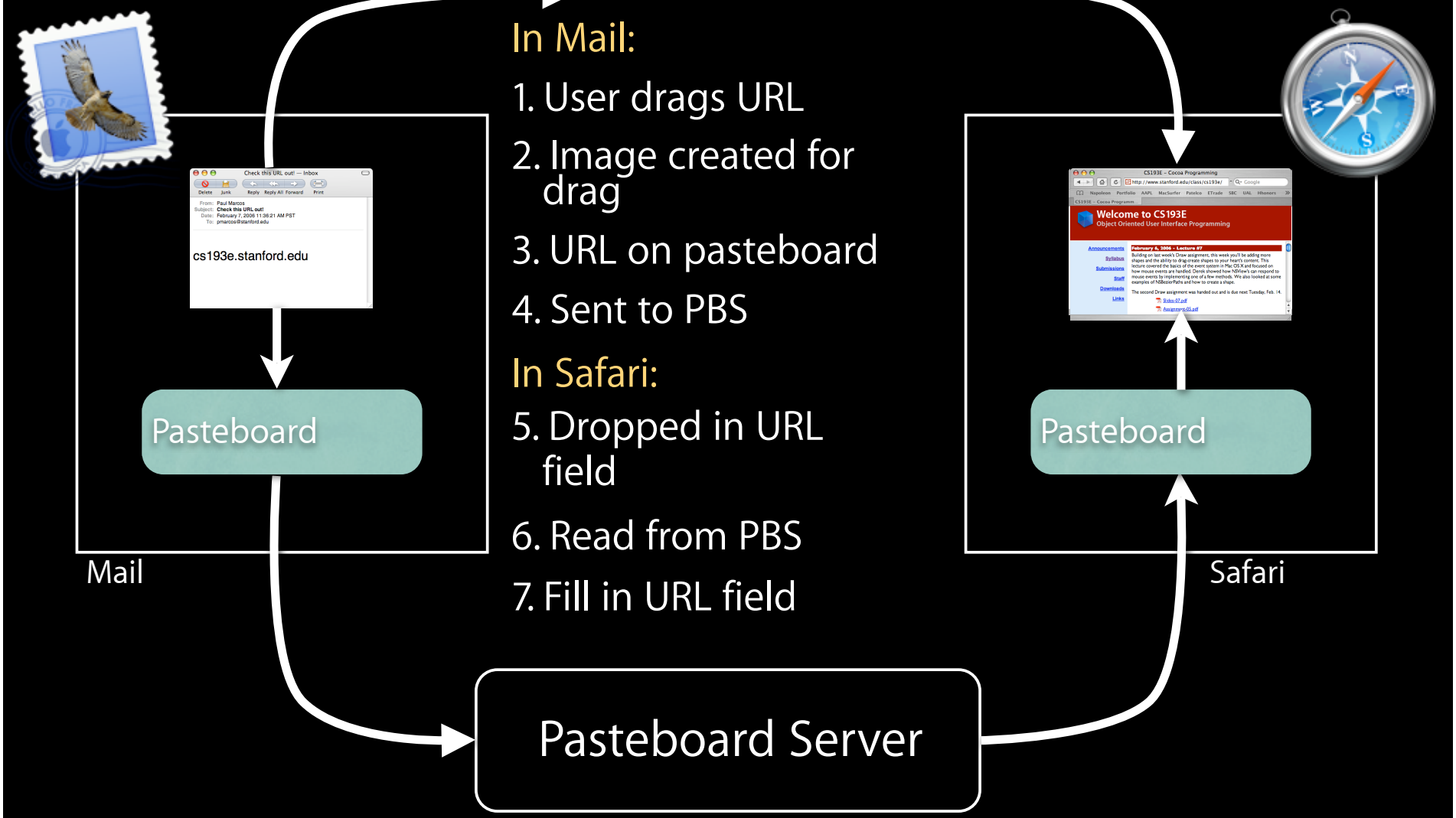
LabelEditor Demo

View Drag-and-Drop

Drag-and-Drop Concepts

- Drag & Drop is copy/paste with some visual pyrotechnics.
- A “Drag Source” starts a dragging session
- A “Drag Destination” receives the data which is “dropped” onto it.
- NSPasteboard is used to shuttle the data from source to destination.

Drag / Drop Flow



Drag Sources

- Your NSView starts the drag session in mouseDown: or mouseDragged:
  1. Put something on the pasteboard
  2. Create an image for dragging
  3. Call

```
- (void)dragImage:(NSImage *)image
    at:(NSPoint)imageLocation
    offset:(NSSize)mouseOffset
    event:(NSEvent *)event
    pasteboard:(NSPasteboard *)pasteboard
    source:(id)sourceObject
    slideBack:(BOOL)slideBack;
```

What's that method again?

```
- (void)dragImage:(NSImage *)image
               at:(NSPoint)imageLocation
           offset:(NSSize)ignored
            event:(NSEvent *)event
       pasteboard:(NSPasteboard *)pasteboard
           source:(id)source /* NSDraggingSource */
        slideBack:(BOOL)slideBack;
```

Where does the image come from?

- You often create it when the drag starts to represent the dragged information
- You can create a new, empty NSImage and draw into it
- Creating an empty NSImage

```
- (id)initWithSize:(NSSize)size;
```

- To draw, lock focus on the image and draw, then unlock focus

```
- (void)lockFocus;
- (void)unlockFocus;
```

Example

```
NSSize size = NSMakeSize(50, 50);
NSImage *image = [[NSImage alloc] initWithSize:size];

[image lockFocus];
[[NSColor blueColor] set];
[NSBezierPath fillRect:NSMakeRect(0, 0, 50, 50)];
[image unlockFocus];
```

NSDraggingSource

Informal protocol

The "source" must implement

```
- (NSDragOperation)
    draggingSourceOperationMaskForLocal:(BOOL)flag;
```

- It may implement other methods

```
- (BOOL)ignoreModifierKeysWhileDragging;
- (void)draggedImage:(NSImage *)image
    beganAt:(NSPoint)screenPoint;
- (void)draggedImage:(NSImage *)image
    endedAt:(NSPoint)screenPoint
    operation:(NSDragOperation)operation;
/* and so on */
```

NSDragOperation

- Determines which operations are permitted
- Can be "or"ed together
- Mostly just determines the look of the cursor

NSDragOperationCopy
NSDragOperationLink
NSDragOperationGeneric
NSDragOperationPrivate
NSDragOperationMove

NSDragOperationDelete
NSDragOperationEvery
NSDragOperationNone

Drag Destinations

- Your NSView subclass declares it accepts certain “dropped” data types

```
- (void)registerForDraggedTypes:(NSArray *)types
```

- The array is an array of pasteboard types

NSDraggingDestination

Informal protocol

```
- (NSDragOperation)draggingEntered: (id <NSDraggingInfo>)sender;
- (NSDragOperation)draggingUpdated: (id <NSDraggingInfo>)sender;
- (void)draggingExited:(id <NSDraggingInfo>)sender;

- (BOOL)prepareForDragOperation: (id <NSDraggingInfo>)sender;
- (BOOL)performDragOperation:(id <NSDraggingInfo>)sender;
- (void)concludeDragOperation:(id <NSDraggingInfo>)sender;
```

NSDraggingInfo

Formal protocol

```
- (id)draggingSource;
- (NSDragOperation)draggingSourceOperationMask;
- (NSPasteboard *)draggingPasteboard;
- (NSPoint)draggingLocation;
- (NSImage *)draggedImage;
- (NSPoint)draggedImageLocation;
- (void)slideDraggedImageTo:(NSPoint)point;
```

Accepting Dragged Colors

- One of the registered drag types should be NSColorPboardType
- NSColor has the nice method

```
+ (NSColor *)colorFromPasteboard:
        (NSPasteboard *)pasteboard;
```

Accepting Dragged Files

- Register for the NSFilenamesPboardType type
- Pasteboard contains an array of filenames via

  [pasteboard propertyListForType:NSFilenamesPboardType]

- Check the array count to restrict drags to a single filename

Table View Drag-and-Drop

Table view drag and drop

- Table view uses the data source
  - To provide drag data
  - To validate and accept drop data
- You configure NSTableView and implement data source methods

Configuring NSTableView

- Registering the types a table view will accept

```
[tableView registerForDraggedTypes: arrayOfTypes];
```

- Configuring locality of drag operations

```
[tableView setDraggingSourceOperationMask:
                        NSDragOperationEvery forLocal:NO];
```

- Not setting the latter is a common cause of ‘dragging only works within my application’ issues

Providing data for a drag

- Implement data source method

```
-(void)tableView:(NSTableView *)tableView
        writeRowsWithIndexes: (NSIndexSet *)indexes
                toPasteboard: (NSPasteboard *)pboard;
```

- This method very similar to a copy method — declare types and set the appropriate data

Validating a drop

- You must implement this method to enable accepting a drag

```
-(NSDragOperation)tableView:(NSTableView *)tableView
               validateDrop: (id <NSDraggingInfo>)info
                proposedRow: (int)row;
      proposedDropOperation: (NSTableViewDropOperation)op;
```

- Returned drag operation refuses drag or returns type of drag
- Proposed row is the index of the drop
- Drop operation is either NSTableViewDropOn or NSTableViewDropAbove, indicating which is proposed.
- You can retarget the drag by sending the table view method

```
-setDropRow:dropOperation:
```

Accepting a drop

- You must implement this method to enable accepting a drag

```
-(BOOL)tableView:(NSTableView *)tableView
      acceptDrop: (id <NSDraggingInfo>)info
             row: (int)row;
   dropOperation: (NSTableViewDropOperation)op;
```

- Retrieve dragging pasteboard from dragging info object
- Pull appropriate data from pasteboard
- Use the row and drop operation to place new data accordingly

Inspectors

Inspectors

- Apps usually have a single inspector panel which inspects the current selection
- Notifications about selection change update the inspector
- Notifications about main window change update the inspector

Questions?