CS193E
Lecture 6

Menus
Responder Chain
User Defaults

Agenda

- Questions on previous material or assignment?
- Responder chain and menus
- User defaults
- Miscellaneous

Demo

TextEdit
Responding to actions

Menu Handling

Menu Actions

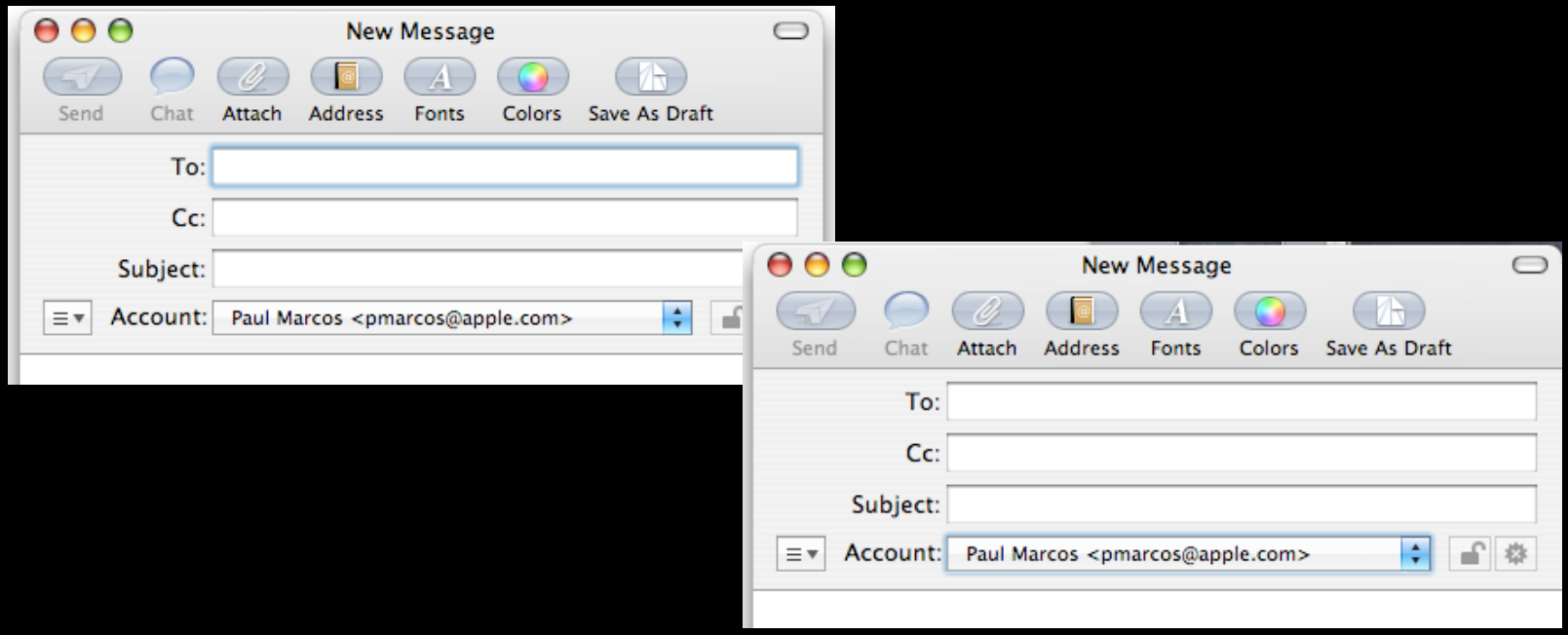
- Menu items have targets and actions just like controls do
- Sometimes menus are wired up like a control, to a single specific target
- Other times, like Copy or Paste, there's no specific target (how could there be?)
- How do menus figure out where actions go?
- Like the military, there's a chain of command

Responder Chain

- The responders in a window get connected to form a list of object to which an action or event is applied
- The responder chain is dynamically updated as the user moves around in an app
- If action or event happens with no specific target, it's "sent up the responder chain"
- Each responder gets a chance to respond to the action, passing to the next if it doesn't

First Responder

- The first object in the responder chain is known as (surprisingly) "the first responder"
- First responder displayed with the blue control outline:



Demo

First Responder and Interface Builder

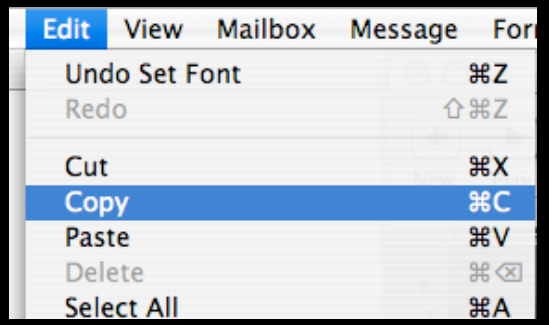
Responder Chain

- In simplest case the order in the chain is as follows:
  1. The main window's first responder
  2. Superviews of the first responder (up to the window's content view)
  3. The main window itself
  4. The main window's delegate
  5. NSApp
  6. NSApp's delegate

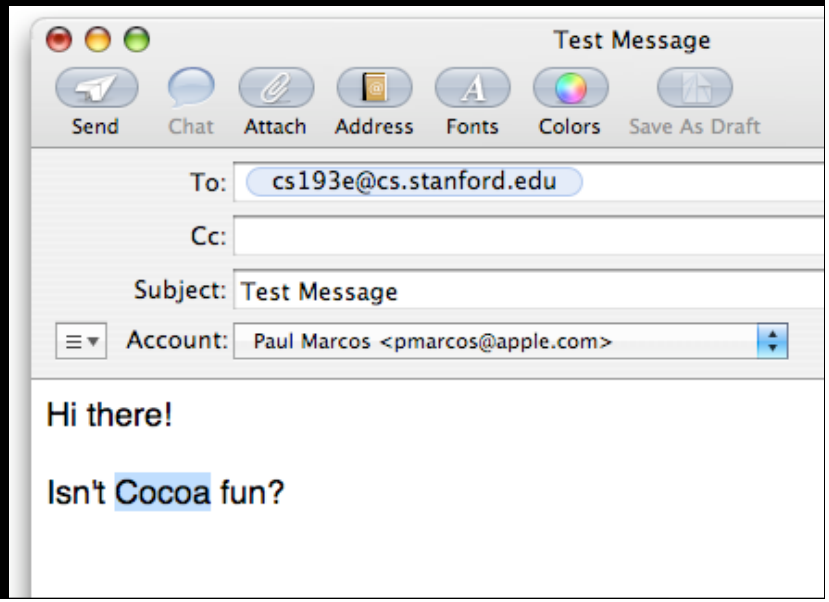
Menu Action Example

User selects Edit > Copy

`copy:` action sent up the responder chain



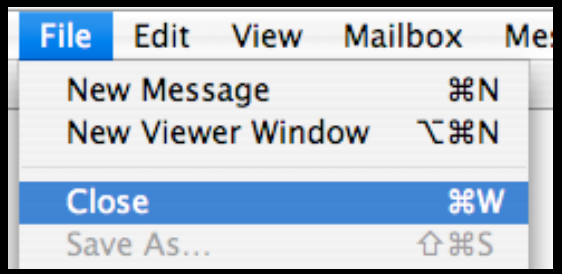
1. The main window's first responder
2. First responder superviews
3. The main window itself
4. The main window's delegate
5. NSApp
6. NSApp's delegate



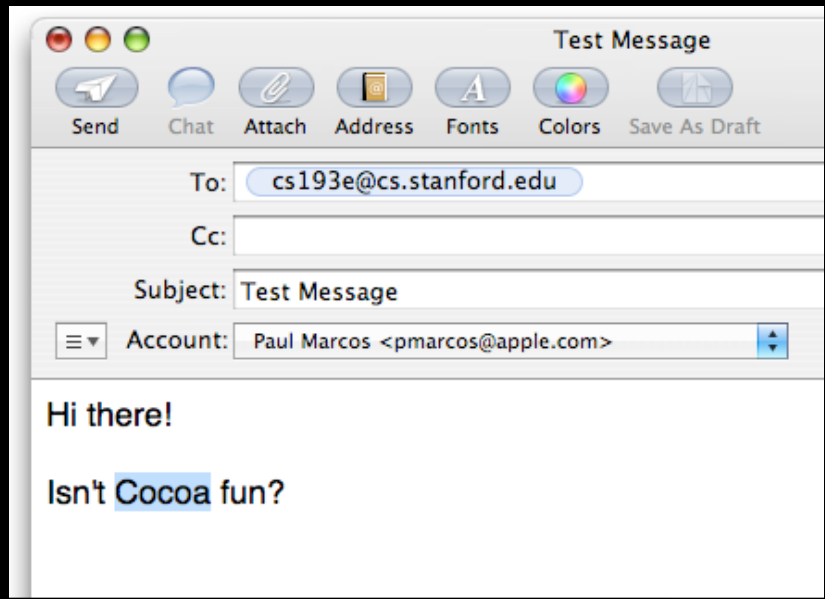
Menu Action Example

User selects File > Close

`performClose:` action sent up the responder chain



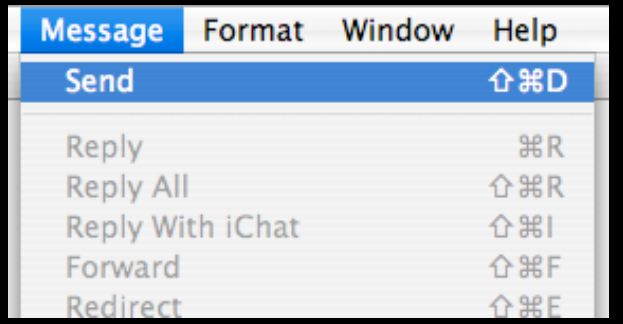
1. The main window's first responder
2. First responder superviews
3. The main window itself
4. The main window's delegate
5. NSApp
6. NSApp's delegate



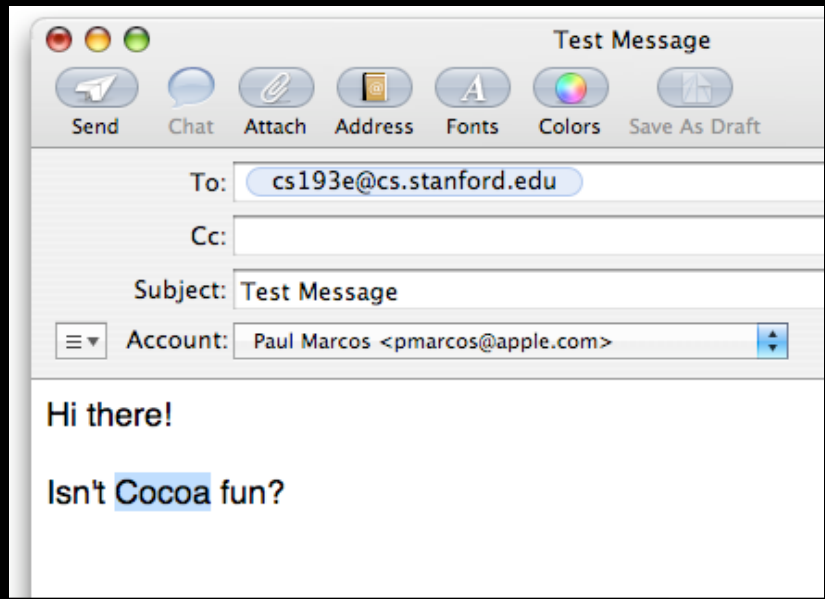
Menu Action Example

User selects Message > Send

`sendMessage:` action sent up the responder chain



1. The main window's first responder
2. First responder superviews
3. The main window itself
4. The main window's delegate
5. NSApp
6. NSApp's delegate



Example Validation

- Validation frequently uses the action of the item to determine if it can be enabled

```
- (BOOL)validateMenuItem:(NSMenuItem *)item
{
    if ([item action] == @selector(saveChanges:)) {
        if (somethingNeedsSaving) {
            return YES;
        } else {
            // Can't cut if there's nothing
            // selected!
            return NO;
        }
    }
    return YES;
}
```

Accepting First Responder

- When mouse clicked in view, Cocoa will try to make that view the first responder
- First it will check to see if it can by calling:
  - (BOOL)acceptsFirstResponder;
- Default implementation returns NO!!!
- Make sure you implement this method and return YES!
- Common cause to "why aren't my menus enabling" question

User Defaults

NSUserDefaults

- Provides means for storing and retrieving user preferences
- Defaults persist between application launches
- Stores key-value pairs
- Has notion of defaults domains for flexibility

Key-Value Pairs

- Key is an NSString
- The value can be
  - Scalar types
    - int, float, BOOL
  - Property list object types
    - NSArray
    - NSDictionary
    - NSString
    - NSNumber
    - NSData
    - NSDate

Getting and setting defaults

```
NSUserDefaults *defaults;
NSString *fileName;
int time;


// Get standard defaults instance
defaults = [NSUserDefaults standardUserDefaults];


// Getting values
time = [defaults integerForKey: @"interval"];
fileName = [defaults stringForKey: @"startupFile"];


// Setting values
[defaults setInteger: time forKey: @"interval"];
[defaults setObject: fileName forKey: @"startupFile"];
```

What about other object types?

Store objects that implement NSCoding as an NSData

```
NSColor *aColor; // assume this exists

// Get standard defaults instance
defaults = [NSUserDefaults standardUserDefaults];

// Archive to NSData and set in defaults
NSData *data =
          [NSKeyedArchiver archivedDataWithRootObject: color];
[defaults setObject: data forKey: @"textColor"];

// Retrieve from defaults and unarchive from NSData
NSData *data = [defaults objectForKey: @"textColor"];
NSColor *textColor =
          [NSKeyedUnarchiver unarchiveObjectWithData:data];
```

Defaults domains

Determine search order and persistence

Domain	Identifier	Persistent
Argument	NSArgumentDomain	NO
Application	Application Bundle Identifier	YES
Global	NSGlobalDomain	YES
Language	Preferred Language Name	NO
Registration	NSRegistrationDomain	NO

NSRegistrationDomain

- Allows the setting of ‘factory defaults’ for your application
- Typically set very early in application launch
- All application user preferences override this domain’s values

Registering defaults early

```
+ (void)initialize {
   if (self == [MyController class]) {
      // get standard NSUserDefaults instance
      NSUserDefaults *defaults
                  = [NSUserDefaults standardUserDefaults];

      // build dictionary of registered defaults
      NSMutableDictionary *dict
                      = [NSMutableDictionary dictionary];
      [dict setObject:@".start" forKey:@"startupFile"];

      // register defaults
      [defaults registerDefaults: dict];
   }
}
```

Miscellaneous

Image Basics
Property Lists

Image Basics

NSImage

- Primary class for handling images in Cocoa
- Encapsulates much of the complexity of images
- Supports a very wide variety of image types
  - Can query class for supported types
- Can be initialized from a variety of sources
- Used for drawing image content
- Can be drawn into to create an image

Creating and using images

```
// From a file
NSString *path; // assume this exists
NSImage *image =
             [[NSImage alloc] initWithContentsOfFile:path];


// From the application bundle or AppKit framework
NSImage *image = [NSImage imageNamed:@"DeleteRecord"];

// Getting image types supported by NSImage
NSArray *types = [NSImage imageFileTypes];

// Using NSImage with NSImageView
NSImageView *imageView;

[imageView setImage: image];
NSImage *image = [imageView image];
```

Property Lists

Property Lists

- File format on Mac OS X for storing common object types
- Commonly called ‘plists’ after the file extension .plist
- Stored in a human readable XML format or a binary format
- Are used on Mac OS X for a great many things
  - Info.plist storing application and bundle data
  - File format used to store preferences
  - Specifying launch criteria and settings to launchd

Working with property list files

- Can be read and edited with Property List Editor tool
- NSPropertyListSerialization class to read and write plists programmatically
- Property lists contain a subset of collection and value classes
  - NSArray
  - NSDictionary
  - NSString
  - NSNumber
  - NSDate
  - NSData

Questions?