User Interfaces

Lecture 22

Cocoa Preferences

Hamza Bennani
hamza@hamzabennani.com

September 27, 2018

Revision

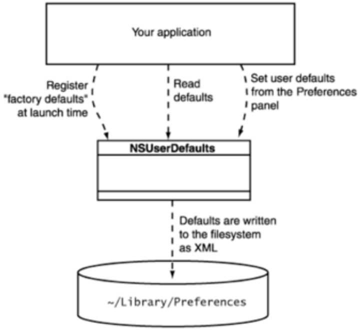
- MacOS Apps: event driven, app life cycle, event loop, apps types . . .
- MVC. . .
- Windows and Views. . .
- Multiple Windows. . .
- Mouse and Keyboards. . .
- Bindings. . .
- Controllers / Undo and Redo. . .
- Preferences. . .

Preferences

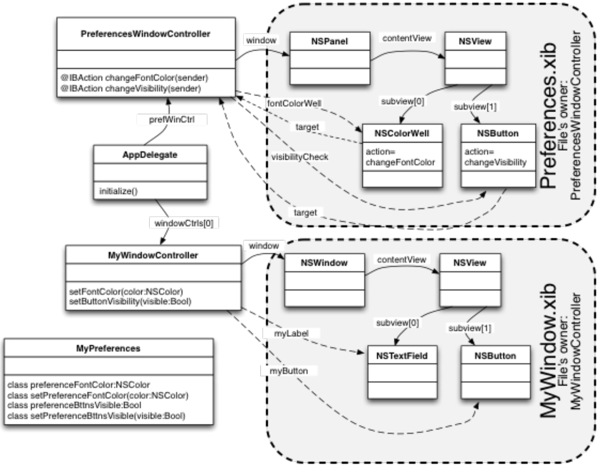
- Most apps. have user-defined preferences
  - Examples are fonts, colours, key bindings, etc.
  - Preferences often are application-wide
- For storing user preferences we use UserDefaults
- For communicating new preference settings to other parts of the application, we use notifications

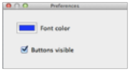
UserDefaults

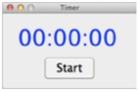
- UserDefaults keeps track of your app’s preferences An application has an instance of UserDefaults for managing user defaults - can fetch a reference to it using: UserDefaults.standard
- Preferences stored under user’s home as a ‘property list’: ~/Library/Preferences (represents a dictionary)
- Process of setting up user default preferences: Register your "factory defaults" Read and use settings Change the settings


Your application
Register "factory defaults" at launch time
Read defaults
Set user defaults from the Preferences panel
NSUserDefaults
Defaults are written to the filesystem as XML
~/Library/Preferences

User Preferences Example





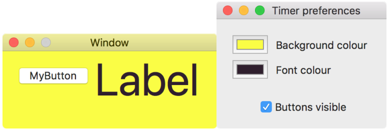


Registering “Factory Defaults”

- When application runs for the first time, the structure with user’s preferences does not exist and it needs to be created
- This needs to happen as early a possible when application starts
  - If we’re an NSObject subclass, can use class method: override class func initialize()
  - (Swift does not natively provide an init equivalent, but see dispatch_once() in init())
- The "factory defaults" setup can be done in the init method of application delegate

Step by Step

- Register Initial Default Settings (Factory Defaults)
- Read the settings
- Use the settings
- Change the settings
- Notifications



Registering "Factory Defaults"

```
// Define archive keys for different settings
let MyFontColorKey = "FontColor"
let MyBttnsVisibleKey = "BttnsVisible"
let BackColorKey = "BackColor";
```

Registering "Factory Defaults"

```
// Initialise the preferences factory defaults (once)
let fontColor:Data = NSKeyedArchiver.archivedData(
    withRootObject: NSColor.red) as Data
let bttnsVisible = NSNumber(value: true)
let defaultValues = [MyFontColorKey:fontColor,
                     MyBttnsVisibleKey:bttnsVisible,
                     BackColorKey:fontColor] as [String : Any]
UserDefaults.standard.register(defaults: defaultValues)
UserDefaults.standard.setPersistentDomain(defaultValues,forName: "~/
    Library/Preferences/userDefaults.plist")
```

Read The settings

Create class methods that read and return each setting from the property list

```
class MyPreferences {
class func preferencesFontColor()->NSColor {
    // Read the colour setting from defaults
    let defaults = UserDefaults.standard
    let color:Data = defaults.object(forKey: MyFontColorKey)
        as! Data
    return NSKeyedUnarchiver.unarchiveObject(with: color)
        as! NSColor
}
class func preferencesButtonsVisible()->Bool {
    let defaults = UserDefaults.standard
    return defaults.bool(forKey: MyBttnsVisibleKey)
}
//[...]
```

Use the Settings

. . . when loading a window where the settings apply

```
@IBOutlet weak var outletLabel: NSTextField!
@IBOutlet weak var outletButton: NSButton!
@IBOutlet weak var window: NSWindow!
```

Use the Settings

. . . when loading a window where the settings apply

```
func applicationDidFinishLaunching(_ aNotification: Notification) {
    // Insert code here to initialize your application
    setFontColor(color: MyPreferences.preferencesFontColor())
    setButtons(visible: MyPreferences.preferencesButtonsVisible())
    setBackColor(color: MyPreferences.preferencesBackColor())
    //[...]
```

Change the Settings

. . . when loading a window where the settings apply

```
func setBackColor(color:NSColor){
    self.window?.backgroundColor = color
}
func setFontColor(color:NSColor){
    outletLabel.textColor = color
}
func setButtons(visible:Bool){
    outletButton.isTransparent = !visible
    outletButton.isEnabled = visible
}
```

Change the Settings

…when loading the preferences window where the settings can be changed

```
class PreferencesController: NSWindowController {
    @IBOutlet weak var backColorWell: NSColorWell!
    @IBOutlet weak var fontColorWell: NSColorWell!
    @IBOutlet weak var visibilityCheck: NSButton!
    override func windowDidLoad() {
        super.windowDidLoad()
        fontColorWell.color = MyPreferences.preferencesFontColor()
        backColorWell.color = MyPreferences.preferencesBackColor();
        if MyPreferences.preferencesButtonsVisible() {
            visibilityCheck.state = NSControl.StateValue(rawValue: 1)
        } else {
            visibilityCheck.state = NSControl.StateValue(rawValue: 0)
        }
    }
```

Change the Settings

. . . Create class methods that save new settings

```
class func setPreferencesFontColor(color:NSColor){
    let colorAsData:Data = NSKeyedArchiver.archivedData(
        withRootObject: color) as Data
    UserDefaults.standard.set(colorAsData, forKey: MyFontColorKey)
```

Change the Settings

. . . Create actions, triggered by the controls in the preference window, which call appropriate preference setters

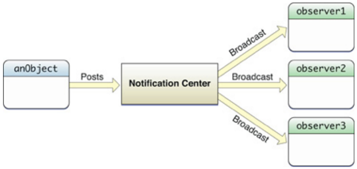
```
@IBAction func changeFontColor(sender: AnyObject?) {
    MyPreferences.setPreferencesFontColor(color: fontColorWell.color)
}
@IBAction func changeVisibility(sender: AnyObject) {
    MyPreferences.setPreferencesButtonsVisible(
        visible: visibilityCheck.state.rawValue==1)
}
@IBAction func changeBackColor(sender: AnyObject) {
    MyPreferences.setPreferencesBackground(color: backColorWell.color)
}
```

Next?

What is left??

Notifications

- Notifications pass around information about the occurrence of events
- Every running application has an instance of NotificationCenter for that application's use
- To use the notification centre:
  - Interested objects register as observers
  - Later, observed object posts notification to centre
  - When notification is posted, observers act
  - What design pattern does this infrastructure correspond to?



Notification

- Notification is a class that encapsulates information carried by a notification
- It contains instance variables for:
  - Notification name, which is a string
  - Reference to an object associated with a notification
  - Reference to Notification, which can carry any other custom information
- Provides methods for creating a new notification with name and associated object as well as getters for reading the name, the object, and custom user information

Predefined Notifications

- Some events you can handle by designating a delegate (for NSApplication or NSWindow), and provide methods that handle various events:
  - Available methods listed in NSApplicationDelegate and NSWindowDelegate protocols
- However, you may also register a class as an observer for a particular event
- AppKit classes post predefined notifications
  - Definitions for these notifications are in the Notification section in the documentation for that class

Predefined Notifications

- Some NSView notifications:
  - NSViewBoundsDidChange,
  - NSViewFrameDidChange, & more. . .
- Some NSWindow notifications:
  - NSWindowDidBecomeKey,
  - NSWindowDidMove,
  - NSWindowDidResize, & many more. . .
- Some NSApplication notifications:
  - NSApplicationDidFinishLaunching,
  - NSApplicationWillTerminate,
  - NSApplicationWillUnhide, & many more. . .

Predefined Notifications

```
var model:MyModel = MyModel()

convenience init() {
    // init parent here...

    // Register us for notifications about preference changes
    let nc = NotificationCenter.default
    nc.addObserver(self,
        selector: #selector(AppDelegate.saveEverything(_:)),
        name: NSNotification.Name.NSApplicationWillTerminate,
        object: nil)
}

func saveEverything(_ note:NSNotification){
    model.saveState()
}
```

Custom Notifications

- You can define your own notifications and register observers for custom notifications
  - Useful for message passing between windows
- All it takes is creating a custom notification string
  - Now you can register observers for that notification string
  - You can send notification associated with that string
- . . . and this is very useful for sending notifications about preference changes

Send Notification

. . . when a preference setting changes

```
class func setPreferencesFontColor(color:NSColor){
    let colorAsData:Data = NSKeyedArchiver.archivedData(
        withRootObject: color) as Data
    UserDefaults.standard.set(colorAsData, forKey: MyFontColorKey)
    // Post a notification about the preference change
    let d = ["color":color]
    let nc = NotificationCenter.default
    nc.post(name: NSNotification.Name(rawValue: MyFontColorKey),
            object: self, userInfo: d)}
class func setPreferencesButtonsVisible(visible:Bool) {
    UserDefaults.standard.set(visible, forKey: MyBttnsVisibleKey)
    let d = ["buttons":visible]
    let nc = NotificationCenter.default
    nc.post(name: Notification.Name(rawValue: MyBttnsVisibleKey),
            object: self, userInfo: d)}
```

Register notification observer

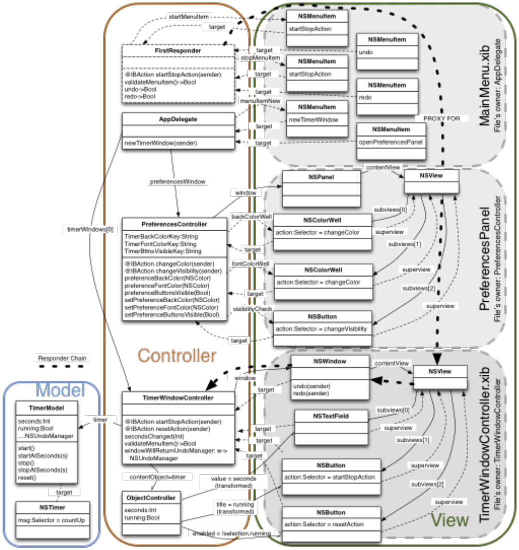
Write handling routines/ Register for notifications with appropriate notification keys

```
setBackColor(color: MyPreferences.preferencesBackColor())
// Register us for notifications about preference changes
let nc = NotificationCenter.default
nc.addObserver(self,selector:
    #selector(AppDelegate.fontChangeHandler(_:)),
                name: NSNotification.Name(rawValue:
                    MyFontColorKey),object: nil)
nc.addObserver(self,selector:
    #selector(AppDelegate.buttonsChangeHandler(_:)),
                name: NSNotification.Name(rawValue:
                    MyBttnsVisibleKey),object: nil)
nc.addObserver(self,selector:
    #selector(AppDelegate.backGroundChangeHandler(_:)), name:
    NSNotification.Name(rawValue: BackColorKey),object: nil)
```

Summary

- Cocoa provides a framework for management of user preferences. These preferences are saved into a .plist file (in XML or binary format) into user's home directory
- **UserDefaults** - the main class that manages user preferences
- **Notification** - a class that encapsulates messages that can be passed between objects in an application
- **NotificationCenter** - class that registers observers interested in notifications

Timer App Preferences



Responder Chain
Model
TimerModel
seconds:Int
running:Bool
...:NSUndoManager
start()
startAtSeconds(s)
stop()
stopAtSeconds(s)
reset()
target
NSTimer
msg:Selector = countUp
timer
Controller
FirstResponder
@IBAction startStopAction(sender)
validateMenuItem()->Bool
undo->Bool
redo->Bool
AppDelegate
newTimerWindow(sender)
preferencesWindow
timerWindows[0]
PreferencesController
TimerBackColorKey:String
TimerFontColorKey:String
TimerBttnsVisibleKey:String
@IBAction changeColor(sender)
@IBAction changeVisibility(sender)
preferenceBackColor(NSColor)
preferenceFontColor(NSColor)
preferenceButtonsVisible(Bool)
setPreferenceBackColor(NSColor)
setPreferenceFontColor(NSColor)
setPreferenceButtonsVisible(Bool)
TimerWindowController
@IBAction startStopAction(sender)
@IBAction resetAction(sender)
secondsChanged(Int)
validateMenuItem()->Bool
windowWillReturnUndoManager: w-> NSUndoManager
contentObject=timer
ObjectController
seconds:Int
running:Bool
value = seconds (transformed)
title = running (transformed)
enabled = !selection.running
startMenuItem
stopMenuItem
menuItemNew
target
window
backColorWell
fontColorWell
visibilityCheck
NSMenuItem
startStopAction
undo
redo
newTimerWindow
openPreferencesPanel
PROXY FOR
MainMenu.xib
File's owner: AppDelegate
contentView
NSPanel
NSView
subviews[0]
superview
subviews[1]
subviews[2]
NSColorWell
action:Selector = changeColor
NSButton
action:Selector = changeVisibility
PreferencesPanel
File's owner: PreferencesController
NSWindow
undo(sender)
redo(sender)
NSTextField
action:Selector = startStopAction
action:Selector = resetAction
View
TimerWindowController.xib
File's owner: TimerWindowController