User Interfaces

Lecture 15

Application Programming on Mac OS

Hamza Bennani
hamza@hamzabennani.com

September 4, 2018

UNIVERSITY of OTAGO
Te Whare Wānanga o Otāgo
NEW ZEALAND

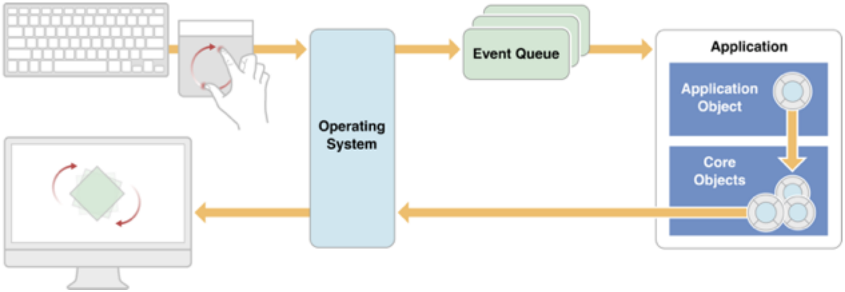
Logistics

- Office hours: Tue/Thu, 2pm to 3pm.
- Office: 250 Geoff Wyvill.
- Acknowledgment: Lech, David, Stephanie.
- Any questions, Feedback, Comments? Email: hamza@hamzabennani.com
- Any suggestions for making the next 12 lectures rock?
- Assignment 2 due on 5th of October at 23:59 the latest.
- Presentations on the 3rd of October at lab time
- https://doodle.com/poll/nbetczehtku2heqa
- The assignment 2 is in pairs (3 pairs so far!!!).
- Labs not updated!
- Tutorials

Mac OS X Application

Definition

An application is a complex system, made of many subcomponents: graphical interface, data processing, event handling, storage, multi-threading

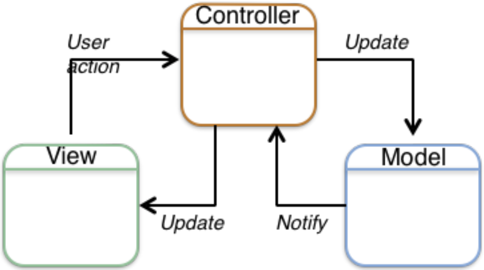


Cocoa Environment

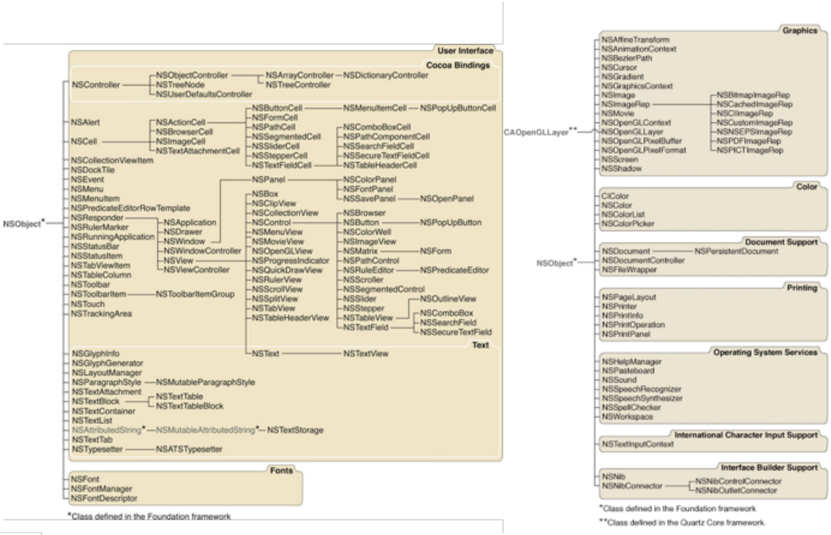
- Cocoa is a Collection of Frameworks & Libraries. Key parts:
- **Appkit**
  - Provides a set of elements for GUI: windows, views, buttons, . . .
  - Provides controllers that glue model & views together
  - Abstracts away most of the logic “under-the-hood” - such as the mouse and keyboard event handling, etc.
- **Core Data**
  - Abstracts away data storage
  - Options for XML, binary files, or SQLite database for storage
- **Foundation Framework**
  - Library for custom logic binding all the other elements together

"One Pattern to Rule them All"

- MVC
  - **M**odel: Information storage
  - **V**iew: Interface that allows the user to interact with the information
  - **C**ontroller: Coordinates interaction between view & model
    Sole purpose: decouple view & model as much as possible
- Cocoa framework heavily utilises the MVC pattern



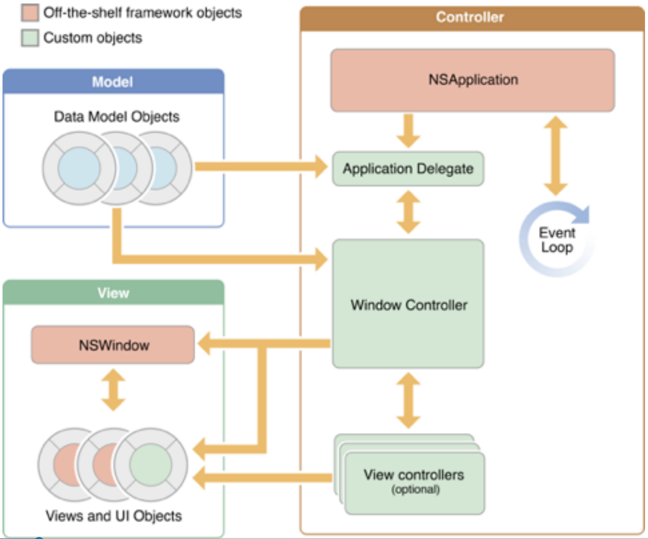
Appkit


User Interface
Cocoa Bindings
NSObject*
NSController
NSObjectController
NSArrayController
NSDictionaryController
NSTreeNode
NSTreeController
NSUserDefaultsController
NSButtonCell
NSMenuItemCell
NSPopUpButtonCell
NSFormCell
NSAlert
NSActionCell
NSPathCell
NSComboBoxCell
NSBrowserCell
NSSegmentedCell
NSPathComponentCell
NSCell
NSImageCell
NSSliderCell
NSSearchFieldCell
NSTextAttachmentCell
NSStepperCell
NSSecureTextFieldCell
NSCollectionViewItem
NSTextFieldCell
NSTableHeaderCell
NSDockTile
NSEvent
NSPanel
NSColorPanel
NSMenu
NSFontPanel
NSMenuItem
NSBox
NSSavePanel
NSOpenPanel
NSPredicateEditorRowTemplate
NSClipView
NSResponder
NSCollectionView
NSBrowser
NSRulerMarker
NSApplication
NSControl
NSButton
NSPopUpButton
NSRunningApplication
NSDrawer
NSMenuView
NSColorWell
NSStatusBar
NSWindow
NSMovieView
NSImageView
NSStatusItem
NSWindowController
NSOpenGLView
NSMatrix
NSForm
NSTabViewItem
NSView
NSProgressIndicator
NSPathControl
NSTableColumn
NSViewController
NSQuickDrawView
NSRuleEditor
NSPredicateEditor
NSToolbar
NSRulerView
NSScroller
NSToolbarItem
NSToolbarItemGroup
NSScrollView
NSSegmentedControl
NSTouch
NSSplitView
NSSlider
NSOutlineView
NSTrackingArea
NSTabView
NSStepper
NSComboBox
NSTableHeaderView
NSTableView
NSSearchField
NSTextField
NSSecureTextField
Text
NSGlyphInfo
NSText
NSTextView
NSGlyphGenerator
NSLayoutManager
NSParagraphStyle
NSMutableParagraphStyle
NSTextAttachment
NSTextTable
NSTextBlock
NSTextTableBlock
NSTextContainer
NSTextList
NSAttributedString*
NSMutableAttributedString*
NSTextStorage
NSTextTab
NSTypesetter
NSATSTypesetter
Fonts
NSFont
NSFontManager
NSFontDescriptor
*Class defined in the Foundation framework
Graphics
NSAffineTransform
NSAnimationContext
NSBezierPath
NSCursor
NSGradient
NSGraphicsContext
NSImage
NSBitmapImageRep
NSImageRep
NSCachedImageRep
NSMovie
NSCIImageRep
NSOpenGLContext
NSCustomImageRep
CAOpenGLLayer**
NSOpenGLLayer
NSNSEPSImageRep
NSOpenGLPixelBuffer
NSPDFImageRep
NSOpenGLPixelFormat
NSPICTImageRep
NSScreen
NSShadow
Color
CIColor
NSColor
NSColorList
NSColorPicker
Document Support
NSDocument
NSPersistentDocument
NSObject*
NSDocumentController
NSFileWrapper
Printing
NSPageLayout
NSPrinter
NSPrintInfo
NSPrintOperation
NSPrintPanel
Operating System Services
NSHelpManager
NSPasteboard
NSSound
NSSpeechRecognizer
NSSpeechSynthesizer
NSSpellChecker
NSWorkspace
International Character Input Support
NSTextInputContext
Interface Builder Support
NSNib
NSNibControlConnector
NSNibConnector
NSNibOutletConnector
*Class defined in the Foundation framework
**Class defined in the Quartz Core framework

Applications types

- Single-window utility app
- Single-window library-style app
- Multi-window document-based app

Single-window app



App Development

Three Ways

- Storyboards: Visual Tool for multiple application views and transitions; (latest)
- XIBs/NIBs: one XIB for one single view; (old-school way)
- Custom Code: no GUI, programmatically.

	Prototyping	Merge Conflicts	Reusability	Auto Layout
StoryBoards	✓	✗	✗	✓
XIBs/NIBs	✓	✗	✓	✓
Custom Code	✗	✓	✓	✗

NSApplication Class

- Every Cocoa application runs exactly one instance of an NSApplication object - **manages the lifecycle** of an application
  - NSApplication is a singleton
  - Instantiated and run from the main function of your program executable
  - NSApp is a global reference for the NSApplication object instance
- Handles the **loading of GUI** at the start and keeps track of windows
  - For instance, which window has the focus, in terms of user input

NSApplication Class

- Runs **the main event loop**
  - Collects and dispatches application events, such as user input
  - Handles redrawing
- Your **program becomes a delegate** of the NSApplication object instance, called after the application is loaded

NSApplication Class

- To get a reference to the running application

```
let application = NSApplication.shared()
```

```
let app = NSApp as NSApplication
```

- Has methods for:
  - Terminating the application
  - Maximising/minimising/hiding windows
  - Updating windows
  - Managing menus

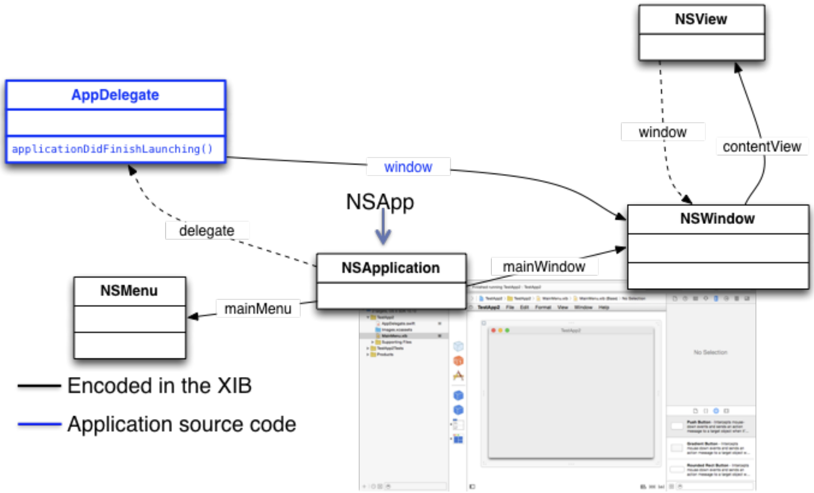
XIB and NIB files

- **XIB: "Xcode/XML Interface Builder"**
  - The XML file in your Xcode project that describes all the visual components added in the Interface Builder
  - The interface shown in the Interface Builder is a rendering corresponding to the contents of this file
  - No need to edit this file directly-when you modify your app's interface in the Interface Builder, the XIB will change
- **NIB**
  - The compiled code corresponding to the XIB file
  - This is a binary file that saves all the objects corresponding to the AppKit's classes specified in the XIB file
  - Becomes part of your application bundle, but you don't access it directly-the application will load it at startup

Application Bundle Contents

- Executable file that starts the application
  - macOS hides that the bundle is actually a directory
  - Clicking on .app runs the executable code in the bundle
- NIB file
  - This file stores all the graphical elements from Xcode's Interface Builder that are part of your application
  - When your application starts, the NIB file is one of the first things to get loaded
- Other files that you included in your Xcode project
  - Images, media, etc.
  - You can access these resources by loading them from the main bundle - the path of the bundle can be found using the NSBundle object corresponding to your application bundle

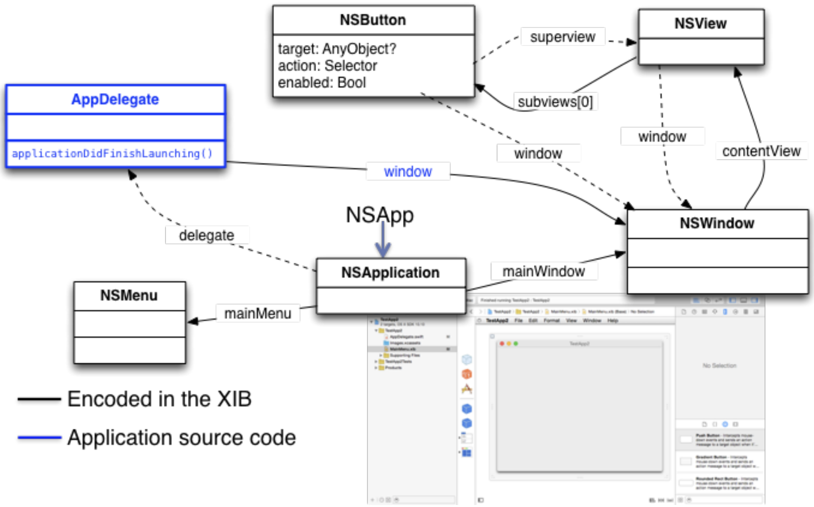
Default Cocoa Application



— Encoded in the XIB

— Application source code

Custom Cocoa Application



Encoded in the XIB

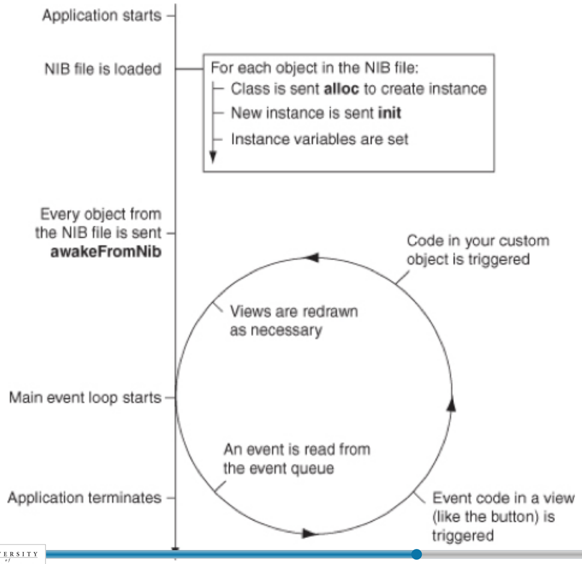
Application source code

Application Delegate

- Code to control your application goes in the AppDelegate.swift file
- Your application delegate serves the **NSApplication** object
- A.D. subscribes to the **NSApplicationDelegate** protocol, which contains optional methods for:
  - launching, terminating, managing the active status, and hiding your application
  - managing windows and dock actions associated with your application
  - opening and printing files
- Your code should be placed in the **applicationDidFinishLaunching**: method, which gets invoked by **NSApplication** after it has completed loading the GUI from NIB

Application Execution

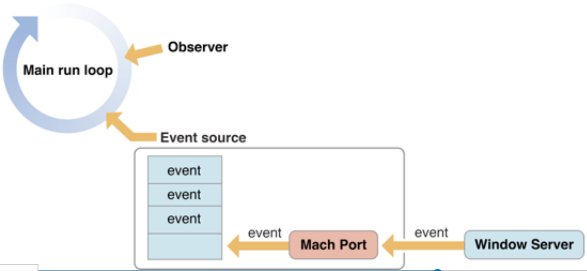
After each object is unarchived from the NIB file and connected (via actions and outlets), it is sent an **awakeFromNib** message.



Application starts
NIB file is loaded
For each object in the NIB file:
Class is sent **alloc** to create instance
New instance is sent **init**
Instance variables are set
Every object from the NIB file is sent **awakeFromNib**
Main event loop starts
Application terminates
Code in your custom object is triggered
Views are redrawn as necessary
An event is read from the event queue
Event code in a view (like the button) is triggered

Main Event Loop

- Waits for events from the OS and dispatches them to appropriate handlers
- The autorelease pool is drained after each pass through the event loop
- When the application terminates, your objects are destroyed



NSRunLoop Class

- You can get a reference to the main loop running in the **NSApplication** instance as follows:

```
let loop = RunLoop.current
```

- Reference to the current loop is useful for adding timers and communication ports

Interface Builder

- Xcode's GUI for creating Cocoa Applications
- You can design the look of your application by dropping various visual elements in the application window
- You can connect graphically various visual elements to your application

Targets and Action

connect controls to code that is invoked when user interacts with the control

Outlets

references in your code to various visual elements, so that they can be controlled programmatically

Outlets

- How do you reference in your code the objects corresponding to the UI elements created in the Interface Builder?
- In Cocoa, these references are called outlets
- In the interface definition for the class, which is going to contain a reference to a given UI object, define a **weak var** preceded by the **@IBOutlet** annotation
- The **@IBOutlet** annotation does not change anything in terms of the program, except for being a special marker for the Interface Builder for keeping track of outlets
- In the Interface Builder you can control-click a UI element and connect it to an outlet

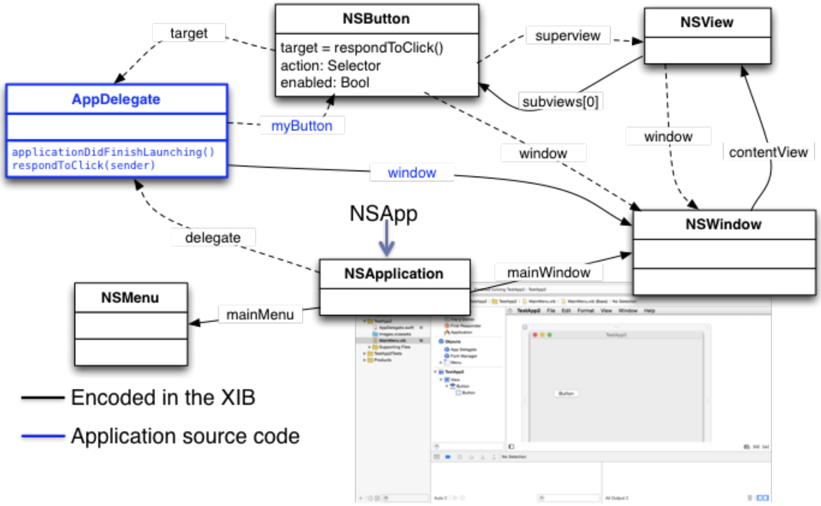
Targets and Actions

- What do you want to happen when a user clicks on a button, or a slider, or a checkbox, or other UI control element?
- Create an **action** - it is a method that implements the logic in response to a user interacting with a control element
- An action method is any method that returns nothing and accepts one parameter (identifying the sender)
- The object implementing the **action** for some control element is referred to as the **target**

Targets and Actions

- Connection of UI elements to corresponding targets and actions can be done graphically in the Interface Builder
- In the interface for the target class specify an action method using the **@IBAction** annotation
- The **@IBAction** annotation is used by the Interface Builder to indicate an action method
- In the Interface Builder control-click a UI element and connect it to a specific **action**

Custom Cocoa Application



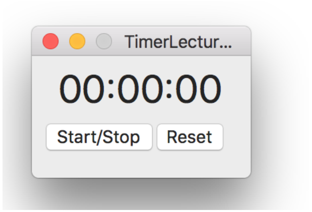
—— Encoded in the XIB

—— Application source code

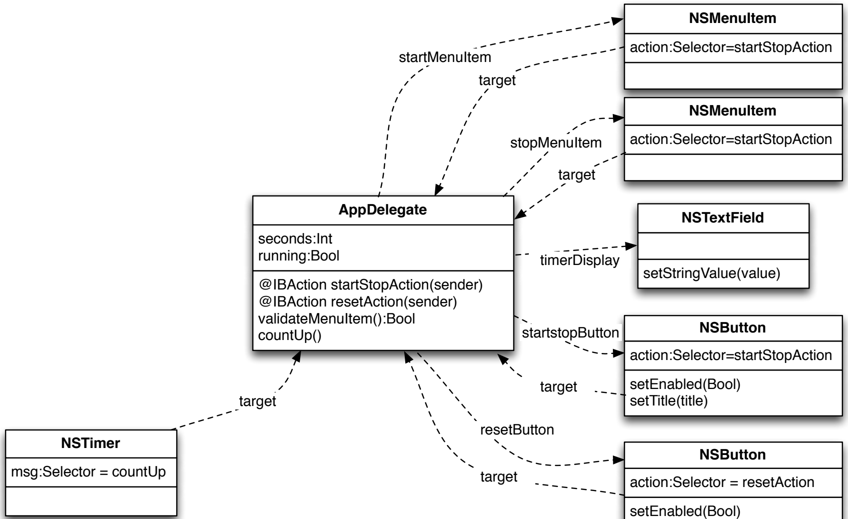
Summary

- Anatomy of the default Cocoa application.
- **NSApplication**: singleton class that runs your application
- **NSApplicationDelegate**: protocol for application delegate with methods for handling various application events
- **XIB**/**NIB**-file storing the visual elements
- **RunLoop**-event loop class, useful for running timers
- The **outlet**, **target** and **action** mechanism which connects each GUI object to the code that dictates their behaviour.
- **@IBOutlet**-annotation that lets Interface Builder know that following pointer is a reference to a GUI element
- **@IBAction**-annotation that lets Interface Builder know that following definition is a an action method to be invoked when a user interacts with a GUI control element

Timer App



Timer App


NSMenuItem
action:Selector=startStopAction
NSMenuItem
action:Selector=startStopAction
startMenuItem
target
stopMenuItem
target
AppDelegate
seconds:Int
running:Bool
@IBAction startStopAction(sender)
@IBAction resetAction(sender)
validateMenuItem():Bool
countUp()
NSTextField
setStringValue(value)
timerDisplay
startstopButton
NSButton
action:Selector=startStopAction
setEnabled(Bool)
setTitle(title)
target
resetButton
target
NSButton
action:Selector = resetAction
setEnabled(Bool)
target
NSTimer
msg:Selector = countUp

Code
IBOutlets & Properties

```
@IBOutlet weak var window: NSWindow!
@IBOutlet weak var outletLabel: NSTextField!
@IBOutlet weak var outletReset: NSButton!
@IBOutlet weak var outletStartStop: NSButton!
@IBOutlet weak var outletStartStopMenu:
        NSMenuItem!
@IBOutlet weak var outletResetMenu:
        NSMenuItem!
var seconds: Int = 0 ;
var running: Bool = false;
```

Code
IBActions

```
@IBAction func actionStartStop(_ sender: Any) {
    if ( running == true){
        stopTimer();
    } else { // running == false!!
        startTimer();
    }
}
@IBAction func actionReset(_ sender: Any) {
    resetTimer();
}
```

Code

Helpful Functions

```
func updateLabel() {
    var secondsLocal = seconds;
    let hour = secondsLocal/3600;
    secondsLocal %= 3600;
    let min = secondsLocal/60;
    secondsLocal %= 60;
    outletLabel.stringValue = String(
        format:"%021d:%021d:%021d",
        hour,min,secondsLocal);
}
```

Code
Helpful Functions

```
@objc func countUp(_ theTimer: Foundation.Timer){
    if ( running == true){
        self.seconds += 1;
        updateLabel();
    } else {
        theTimer.invalidate();
    }
}
```

Code
Main Functions

```
func startTimer(){
    let theTimer = Foundation.Timer(timeInterval:
        1,      target: self, selector:
        #selector( AppDelegate.countUp(_:)),
        userInfo: nil, repeats: true);
    let loop = RunLoop.current;
    loop.add(theTimer, forMode:
    RunLoopMode.commonModes);
    self.running = true;
    outletStartStop.title = "Stop";
    outletStartStopMenu.title = "Stop";
    outletReset.isEnabled = false;
}
```

Code
Main Functions

```
func stopTimer() {
    self.running=false;
    outletReset.isEnabled=true;
    outletStartStop.title = "Start";
    outletStartStopMenu.title = "Start";
}

func resetTimer(){
    self.seconds = 0;
    updateLabel();
 }
```

Code
Main Functions

```
override func validateMenuItem(_ menuItem:
        NSMenuItem) -> Bool {
    if ( menuItem == outletStartStopMenu){
        return true;
    } else if (menuItem == outletResetMenu){
        return !running;
    } else {
        return true;
    }
}
```