Implementing Visualization Apps on iOS Devices

Elastic Image Software LLC

iOS software development is done using Objective-C an object-oriented superset of C.

It was developed in the spirit of Smalltalk: message-y and introspective.

- Objective-C and C can be intermingled freely.
- Objective-C is simple, approachable, lightweight, and pragmatic. No frills.
- In a nutshell: OOP for C hackers and Unix heads.

Objective-C Supports

- **Class**: Grouping of data + code. The type of an object.
- **Instance**: A specific copy of a class.
- **Method**: A message that an object can respond to.
- **Instance variable (ivar)**: A piece of data belonging to an object

Interface - .h file

```
@interface className : superclassName

@property(nonatomic, retain) Type *propertyForType;

+(return_type)classMethod;

+(return_type)classMethodWithParam:(paramType) paramName;

-(return_type)instanceMethodWithParam1:(param1Type)param1Name
                           andParam2:(param2Type) param2Name;

@end
```

Implementation - .m file

```
@implementation classname

@synthesize propertyForType;

+(return_type)classMethod {
    // do stuff
}

+(return_type)classMethodWithParam:(paramType) paramName {
    // do stuff
}

-(return_type)instanceMethodWithParam1:(param1Type)param1Name
                             andParam2:(param2Type) param2Name {
    // do stuff
}

@end
```

Apple style tends towards long self-documenting method names. This helps you familiarize yourself with other's code.

- instantiation
- property setting
- message passing

```
self.window =
[[UIWindow alloc] initWithFrame:[[UIScreen mainScreen] bounds]];

self.window.backgroundColor = [UIColor whiteColor];

[self.window makeKeyAndVisible];
```

Objective-C Types

Dynamically-typed: `id whoCaresWhatThisIs;`

Statically-typed: `Thang* aThang;`

Selectors identify methods by name

```
@interface Observer : NSObject

@property(nonatomic, retain)  id targetObject;
@property(nonatomic, assign) SEL targetAction;

-(id) initWithTarget:(id)object action:(SEL)action;

@end
```

Selectors identify methods by name

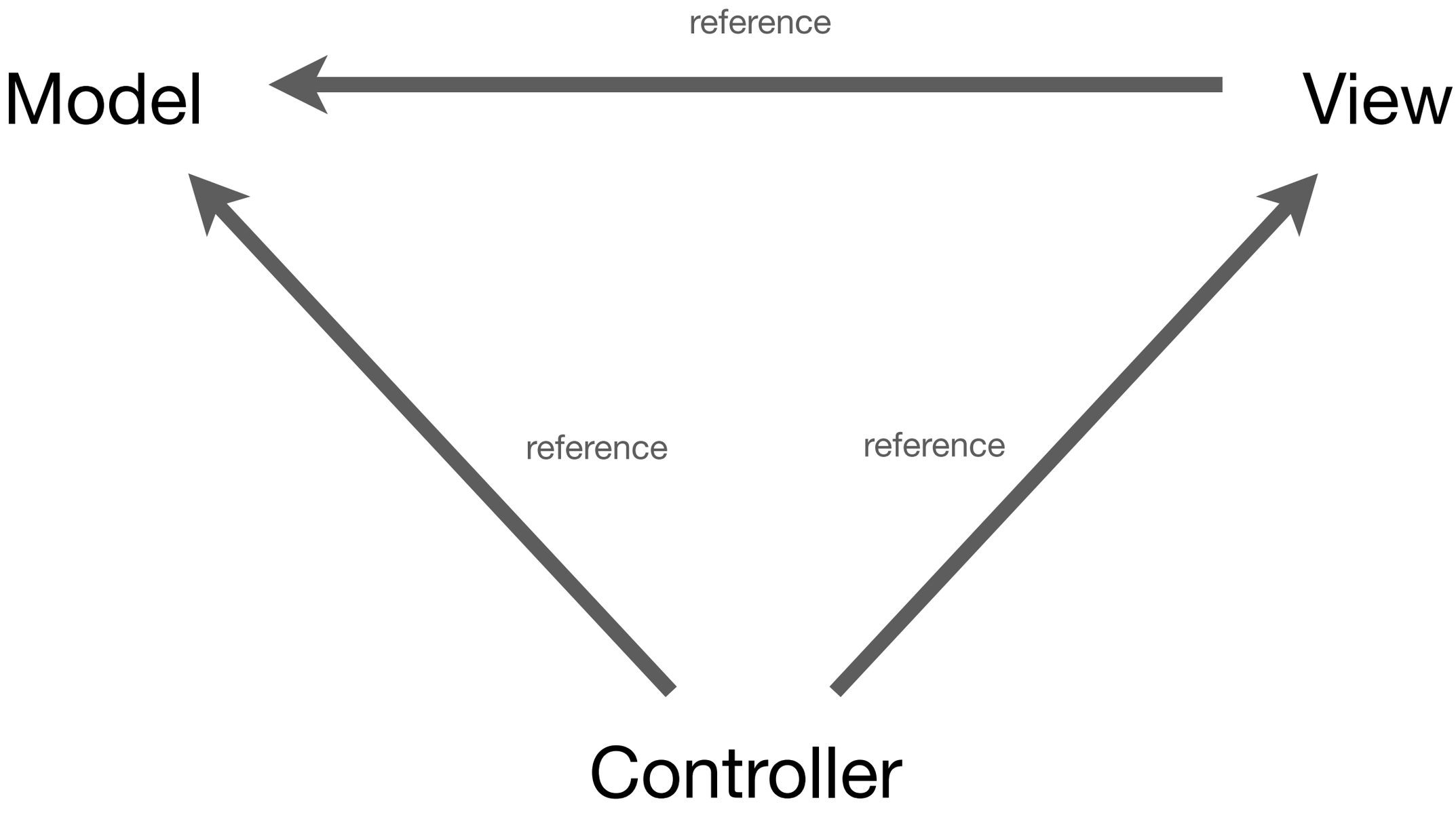
```
observer = [[Observer alloc] initWithTarget:self
                                     action:@selector(updateDisplay:)];
```

Objective Lifecycle

- Create an instance.
- Futz with it (Send messages. Pass it around.)
- Discard it.

iOS is designed around the foundational pattern: Model View Controller.

Much of iOS development - excluding Model development - involves customization and extension of this foundation.


reference
Model
View
reference
reference
Controller

Model

View

Key-value **O**bserving
or
Notification

Key-value **O**bserving
or
Notification

Target - Action

Controller

iOS has rich support for a loose, flat, decoupled style of programming

- Notification
- Target - Action
- Key-value Observing (KVO)
- Block
- Dispatch Queue
- Delegation (Protocols)

Notifications

Notification

Notification respondent

```
[[NSNotificationCenter defaultCenter] addObserver:self
                                         selector:@selector(respondToMyNotification:)
                                             name:MyNotification
                                           object:nil];
```

```
- (void) respondToMyNotification:(NSNotification *)notification {
    // do stuff
}
```

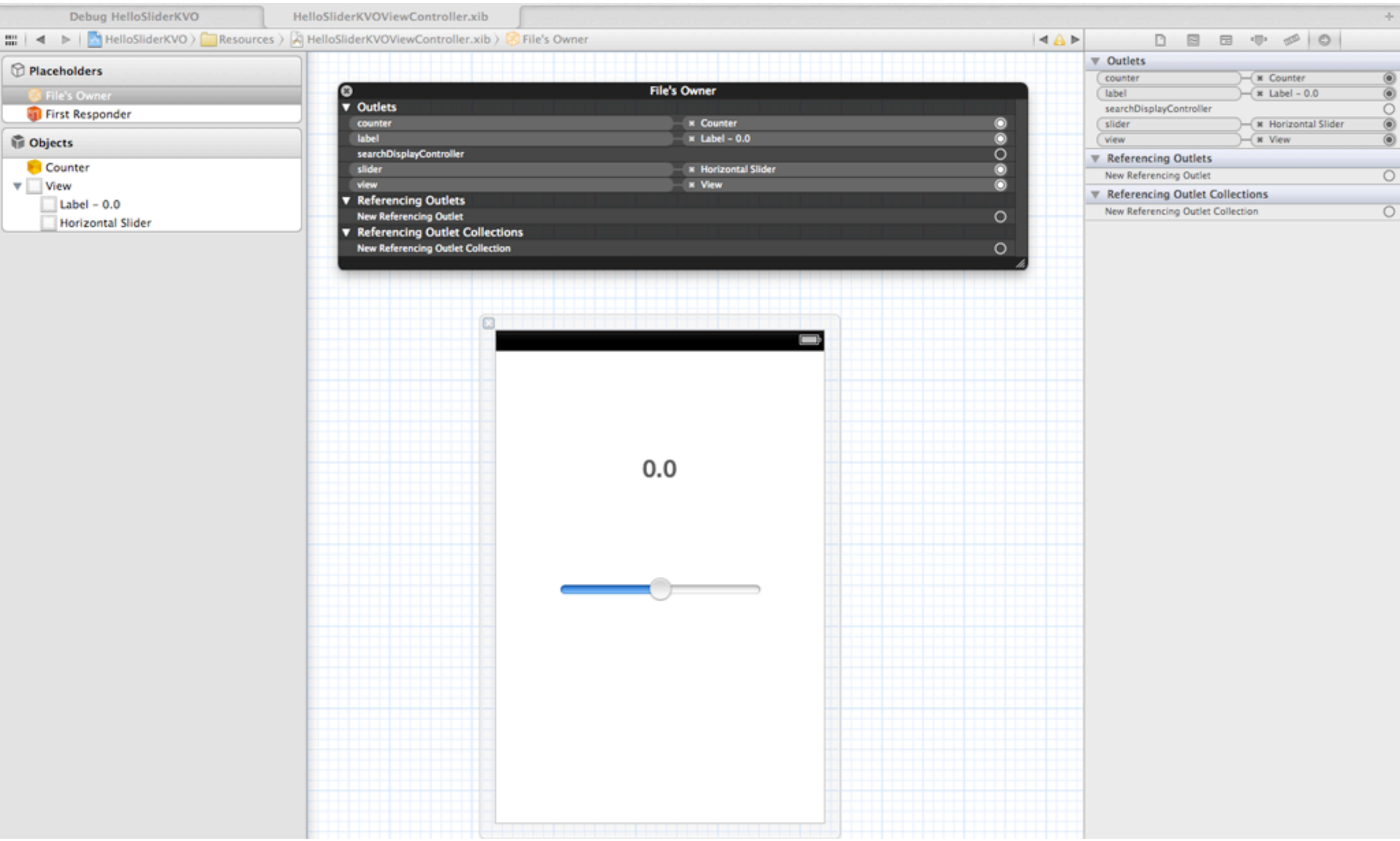
Notification

Notifier

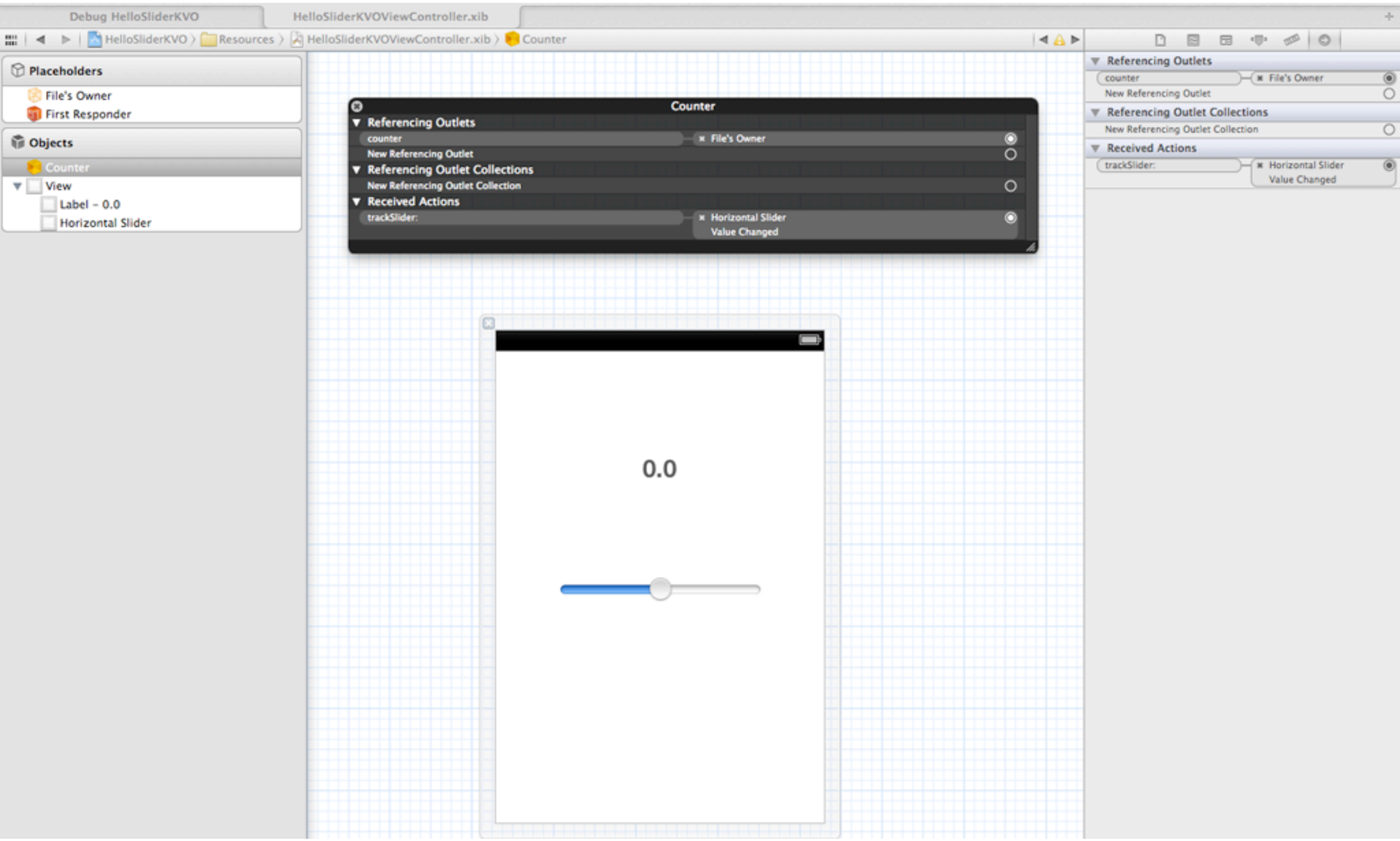
```
[[NSNotificationCenter defaultCenter] postNotificationName:MyNotification object:self];
```

Target - Action

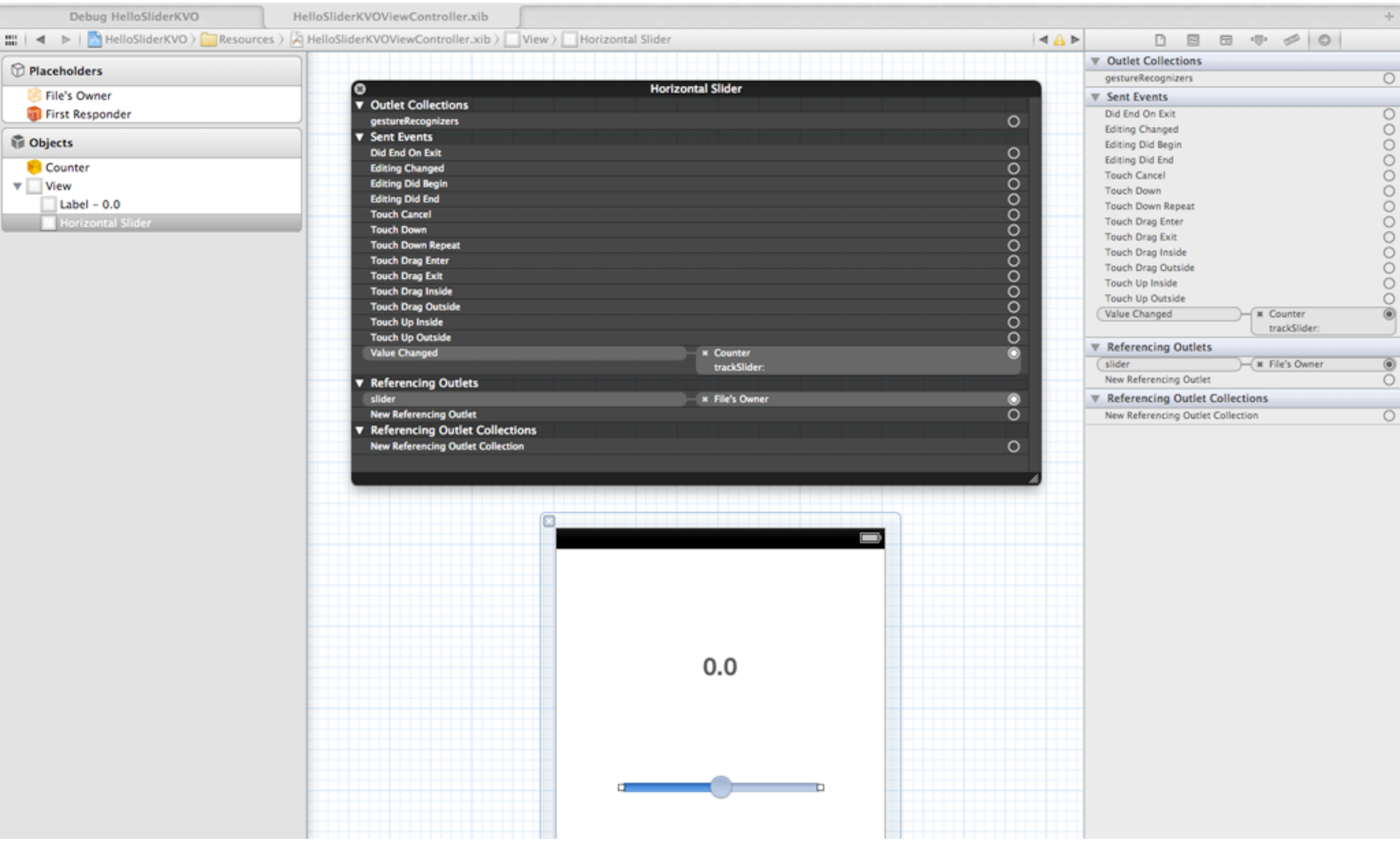
Target - Action



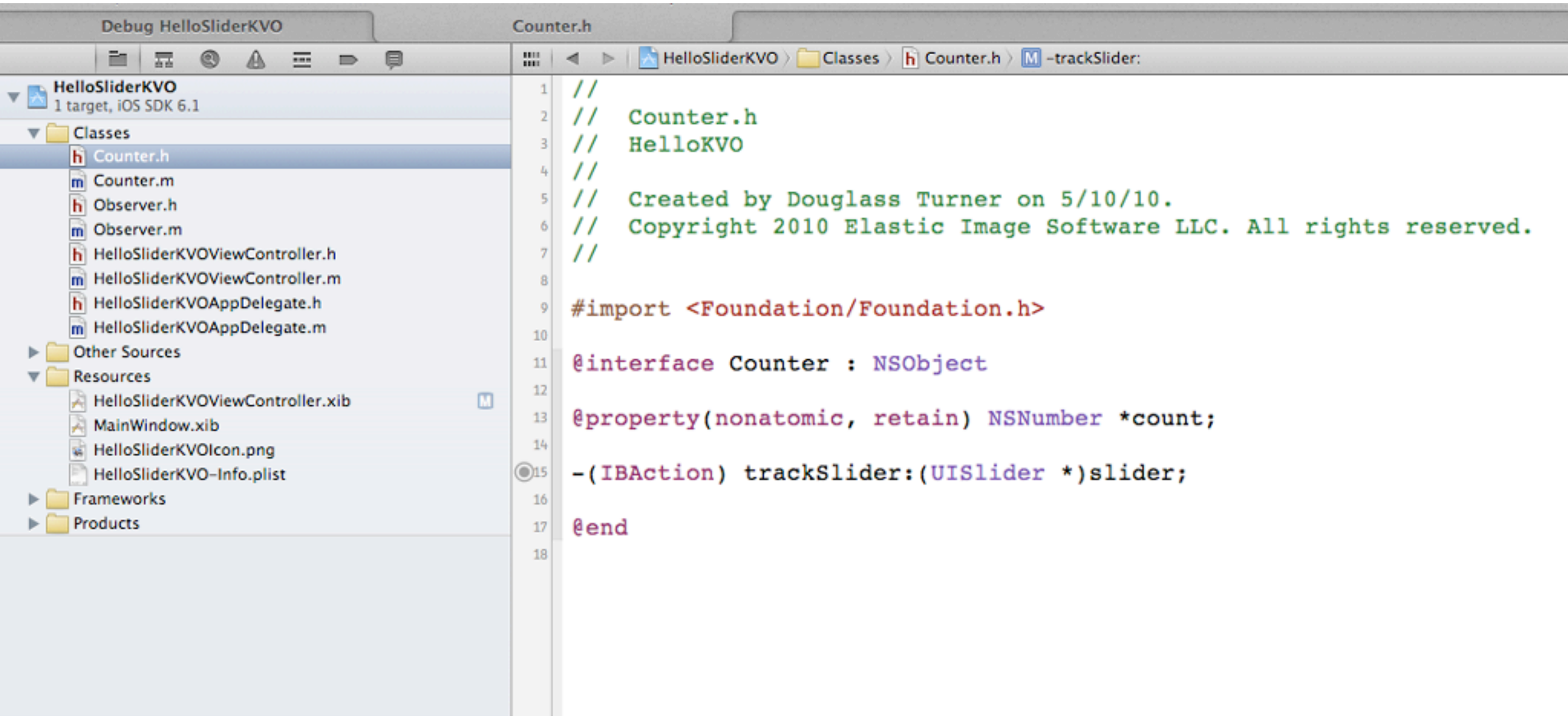
Target - Action



Target - Action



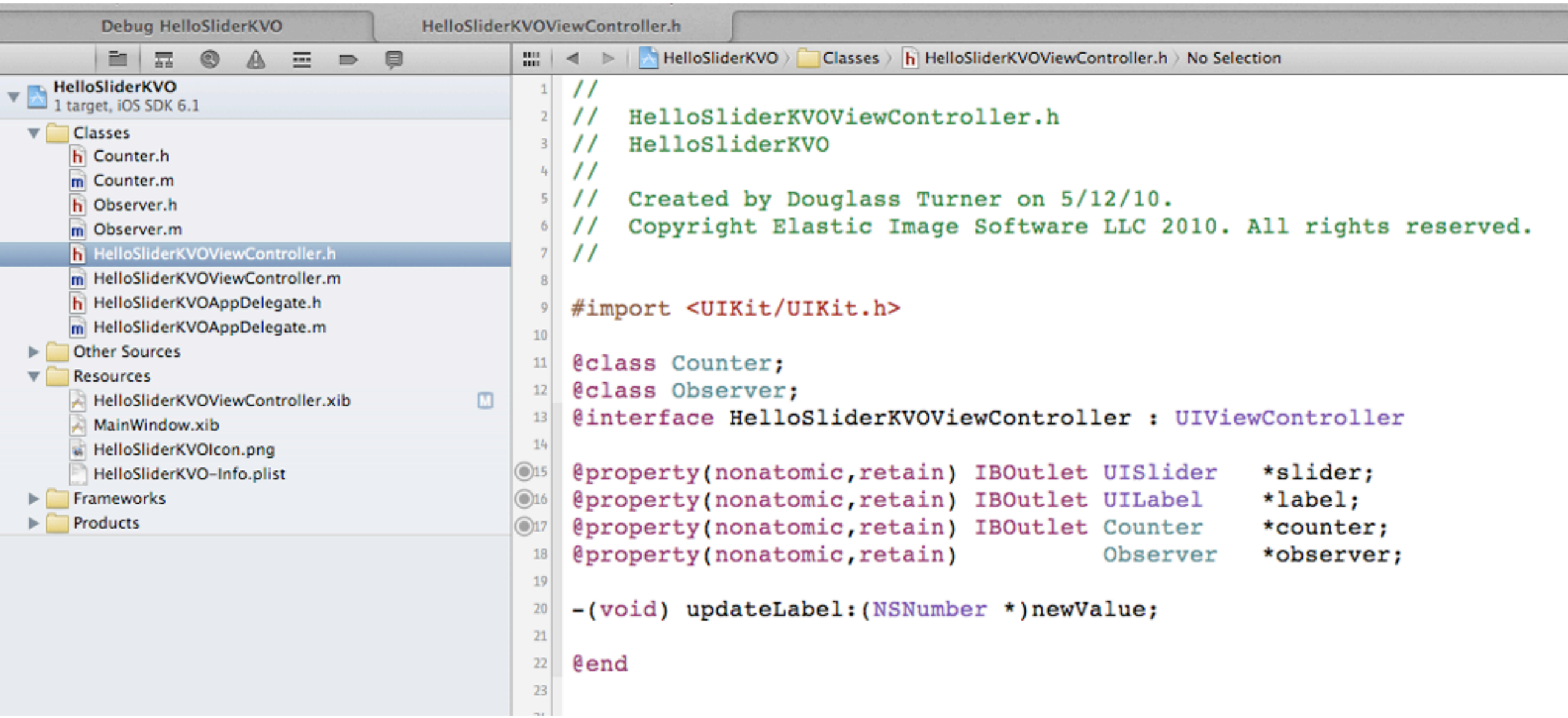
Target - Action



Target - Action

```
@implementation Counter

-(IBAction) trackSlider:(UISlider *)slider; {
    self.count = [NSNumber numberWithFloat:slider.value];
}

@end
```

Target - Action



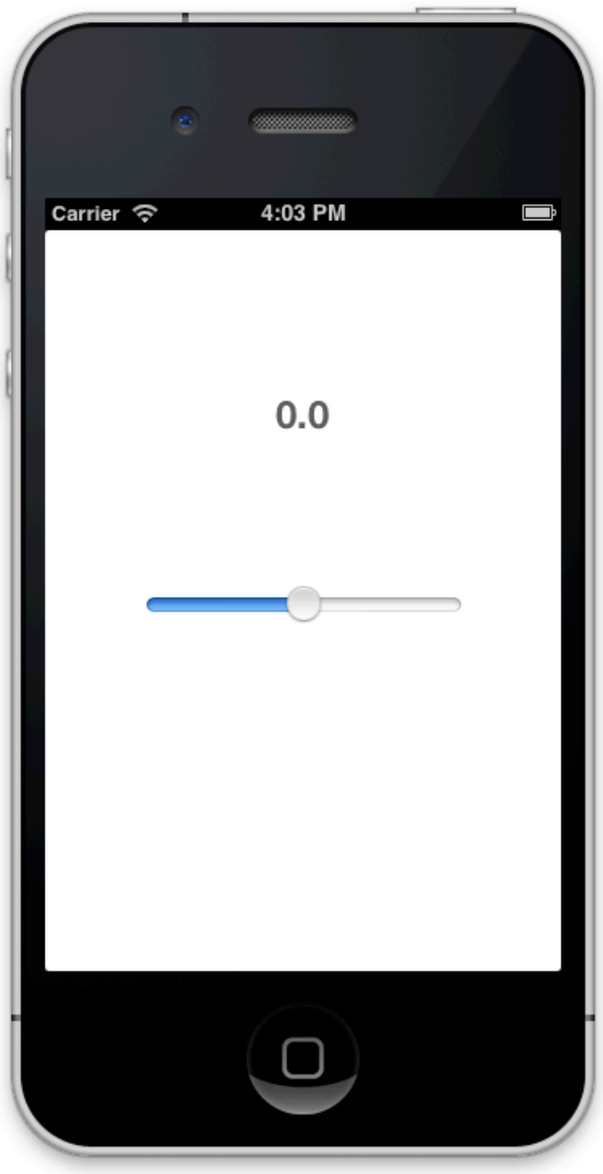
Target - Action

```
@implementation HelloSliderKVOViewController

- (void)viewDidLoad {

    self.observer = [[[Observer alloc] initWithTarget:self
                                               action:@selector(updateLabel:)] autorelease];

    [self.counter addObserver:self.observer
                   forKeyPath:@"count"
                      options:NSKeyValueObservingOptionNew
                      context:NULL];
}

-(void) updateLabel:(NSNumber *)newValue {

    self.label.text = [NSString stringWithFormat:@"%.2f", [newValue floatValue]];
}

@end
```

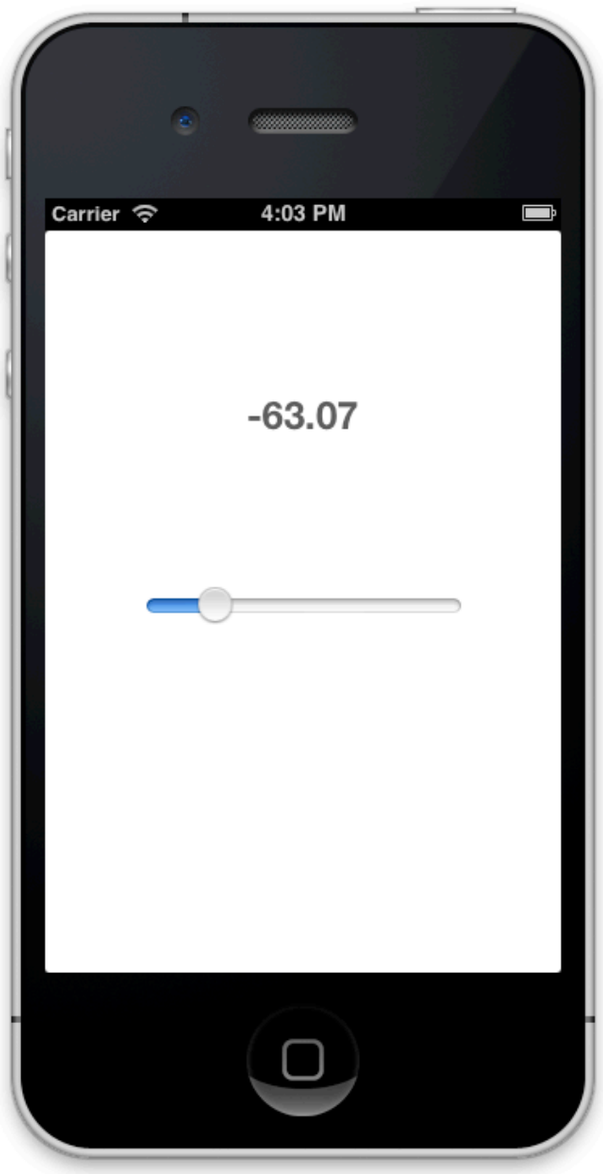
Key-value Observing (KVO)

Any **property** is by default “Key-value Compliant” allowing it to be observed.

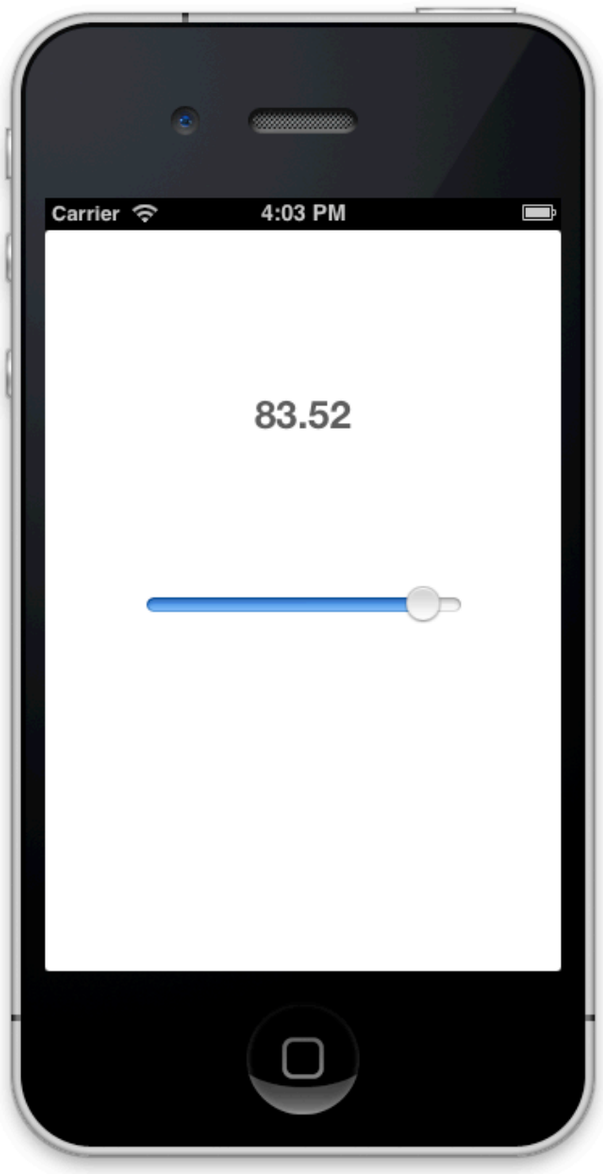
Example: Hello Slider KVO



Example: Hello Slider KVO



Example: Hello Slider KVO



Example: Hello Slider KVO

```
@interface Counter : NSObject

@property(nonatomic, retain) NSNumber *count;

-(IBAction) trackSlider:(UISlider *)slider;

@end
```

Example: Hello Slider KVO

```
@implementation Counter

-(IBAction) trackSlider:(UISlider *)slider; {
    self.count = [NSNumber numberWithFloat:slider.value];
}

@end
```

Example: Hello Slider KVO

```
@interface Observer : NSObject

-(id) initWithTarget:(id)object action:(SEL)action;

@property(nonatomic,retain)id targetObject;
@property(nonatomic,assign)SEL targetAction;

@end
```

Example: Hello Slider KVO

```
// Observer Implementation ...
```

```
-(id) initWithTarget:(id)object action:(SEL)action {
    if (self = [super init]) {
        self.targetObject = object;
        self.targetAction = action;
    }

    return self;
}
```

Example: Hello Slider KVO

```
// Observer Implementation ...


- (void)observeValueForKeyPath:ofObject:change:context: {

    [self.targetObject performSelector:self.targetAction
                            withObject:[object valueForKeyPath:keyPath]];

}
```

Target - Action

```
@implementation HelloSliderKVOViewController

- (void)viewDidLoad {

    self.observer = [[[Observer alloc] initWithTarget:self
                                               action:@selector(updateLabel:)] autorelease];

    [self.counter addObserver:self.observer
                   forKeyPath:@"count"
                      options:NSKeyValueObservingOptionNew
                      context:NULL];
}

-(void) updateLabel:(NSNumber *)newValue {

    self.label.text = [NSString stringWithFormat:@"%.2f", [newValue floatValue]];
}

@end
```

HelloSlideKVO Demo

GitHub: http://bit.ly/VUzM2a

Blocks & Dispatch Queues

Blocks & Dispatch Queues

Block

```
^{ NSLog(@"Doing something"); }
```

Dispatch Queue

```
dispatch_queue_t queue = dispatch_get_global_queue(0,0);

dispatch_async(queue, ^{
 NSLog(@"Doing something");
});
```

Blocks & Dispatch Queues

```
- (void)updateFeaturesWithNotification:(NSNotification *)notification {

    dispatch_async([UIApplication sharedIGVAppDelegate].trackControllerQueue, ^{

        [self updateFeatures];

        dispatch_async(dispatch_get_main_queue(), ^{

            [self.coverageTrack setNeedsDisplay];
            [self.track setNeedsDisplay];

        });
    });

}
```

Delegation (Protocol)

Delegation (Protocol)

A protocol is identical to an interface in Java. A collection of method signatures implemented by the object that “conforms” to the protocol.

The delegate/protocol pattern is ubiquitous throughout iOS.

Delegation (Protocol)

```
@interface UITableViewController:UIViewController <UITableViewDelegate, UITableViewDataSource>
```

UITableViewController inherits from UIViewController and conforms to the UITableViewDelegate and UITableViewDataSource protocols

UITableViewDelegate

```
@protocol UITableViewDelegate<NSObject, UIScrollViewDelegate>

@optional
    - (NSIndexPath *)tableView:willSelectRowAtIndexPath:;
    - (NSIndexPath *)tableView:willDeselectRowAtIndexPath:;

@end
```

UITableViewDataSource

```
@protocol UITableViewDataSource<NSObject>

@required
- (NSInteger)tableView:numberOfRowsInSection:;

@optional
- (NSInteger)numberOfSectionsInTableView:;
- (NSArray *)sectionIndexTitlesForTableView:;

@end
```

The blend of mobility, gesture, and rise of the GPU makes iOS App development an entirely new form of software development.

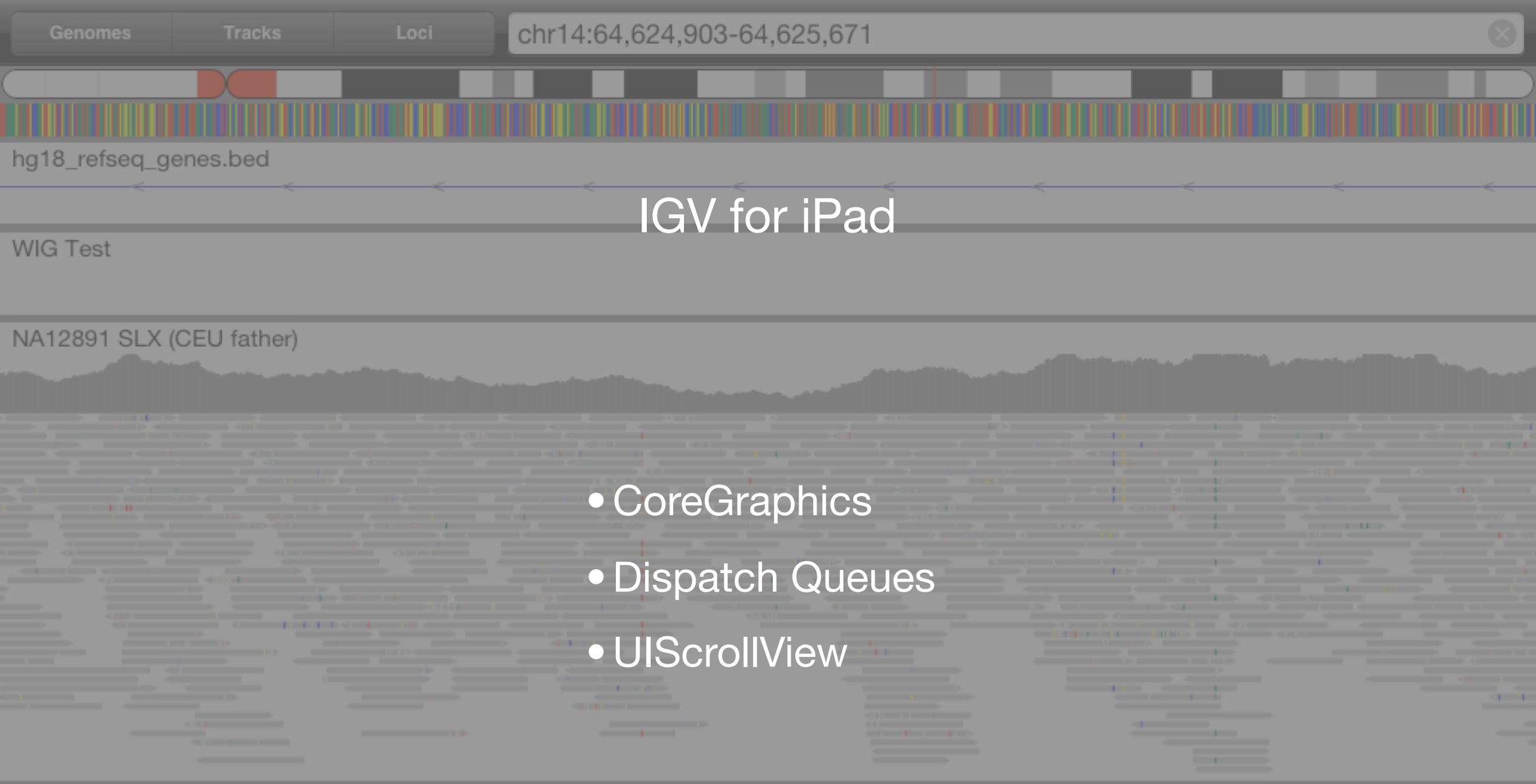
Developers must discard many of their desktop assumptions when developing for iOS

- No mouse
- No interface
- No keyboard
- Arms length interaction
- One handed Interaction

Demos

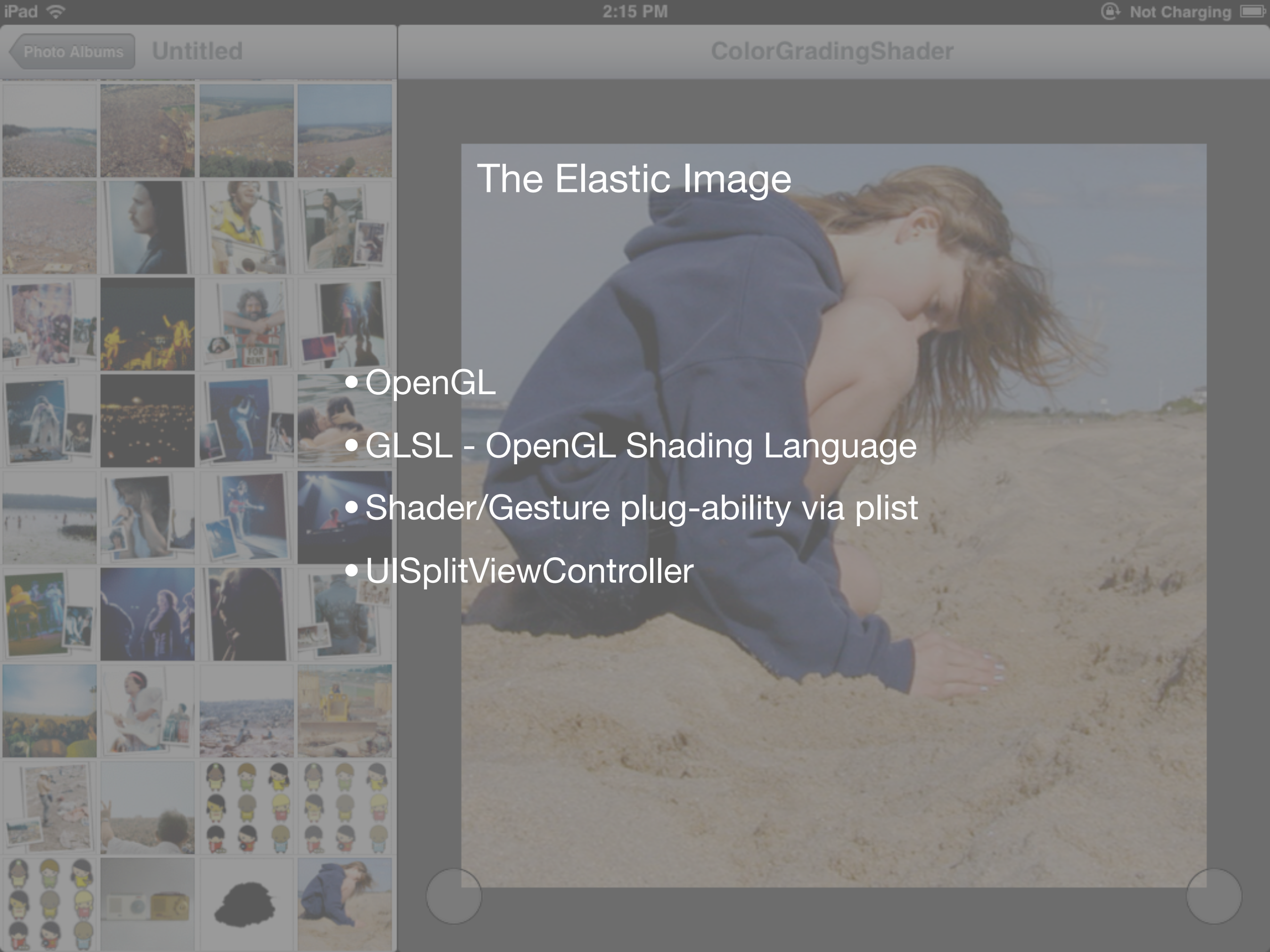
Multi-resolution Image

- CATiledLayer
- UIScrollView
- Amazon Web Services (S3)



IGV for iPad

- CoreGraphics
- Dispatch Queues
- UIScrollView


iPad
2:15 PM
Not Charging
Photo Albums
Untitled
ColorGradingShader
The Elastic Image
• OpenGL
• GLSL - OpenGL Shading Language
• Shader/Gesture plug-ability via plist
• UISplitViewController

The Elastic Image

Property lists enable simple specification of a shader gesture and its handler.

The Objective-C runtimes enables easy conversion from string to class instance

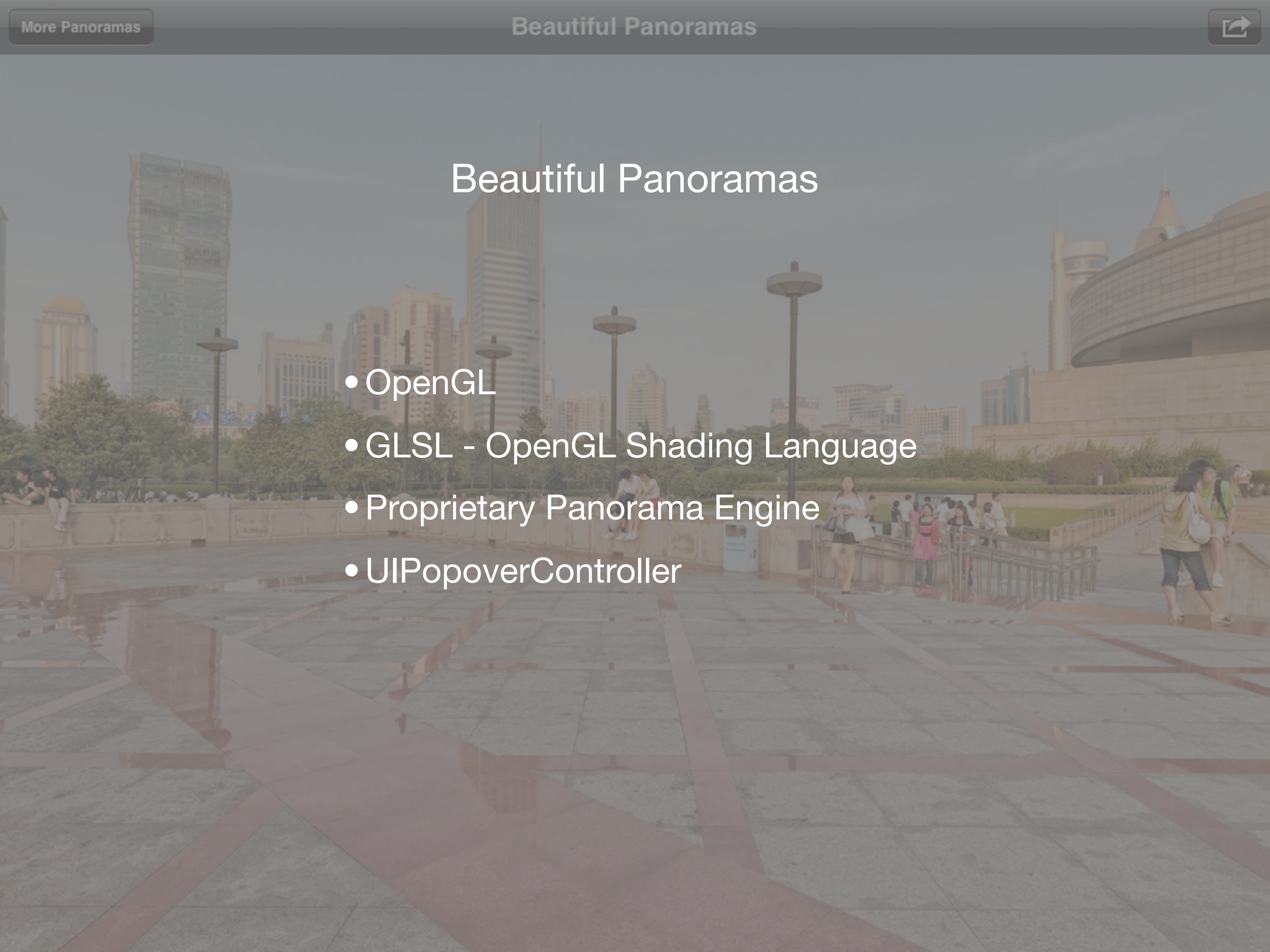
Key	Class	Value
▼Root	Dictionary	⬍ 1 key/value pairs
▼shaderLibrary	Dictionary	⬍ 9 key/value pairs
▼ColorGradingShader	Dictionary	⬍ 4 key/value pairs
▼uniforms	Dictionary	⬍ 5 key/value pairs
▼gestures	Array	⬍ 1 ordered objects
▼0	Dictionary	⬍ 2 key/value pairs
class	String	⬍ UIPanGestureRecognizer
selector	String	⬍ colorGradingShaderGestureHandler:
▶hero	Dictionary	⬍ 2 key/value pairs
▶lut	Dictionary	⬍ 2 key/value pairs
▶mixer	Dictionary	⬍ 3 key/value pairs
▶projectionViewModelMatrix	Dictionary	⬍ 2 key/value pairs
fragment	String	⬍ EISColorGradingShader
vertex	String	⬍ EISColorGradingShader
▶vertexAttributes	Dictionary	⬍ 2 key/value pairs

The Elastic Image

Property lists enable simple specification of a shader gesture and its handler.

The Objective-C runtimes enables easy conversion from string to class instance

```
- (UIGestureRecognizer *)createGestureWithDictionary:(NSDictionary *)gestureDictionary {

    NSString *gestureClassName = [gestureDictionary objectForKey:@"class"];
    Class _gesture = NSClassFromString(gestureClassName);

    NSString *selectorName = [gestureDictionary objectForKey:@"selector"];
    SEL  _selector = NSSelectorFromString(selectorName);

    UIGestureRecognizer *shaderGesture =
    [[[_gesture alloc] initWithTarget:self action:_selector] autorelease];

    shaderGesture.delegate = self.detailController;

    return shaderGesture;
}
```

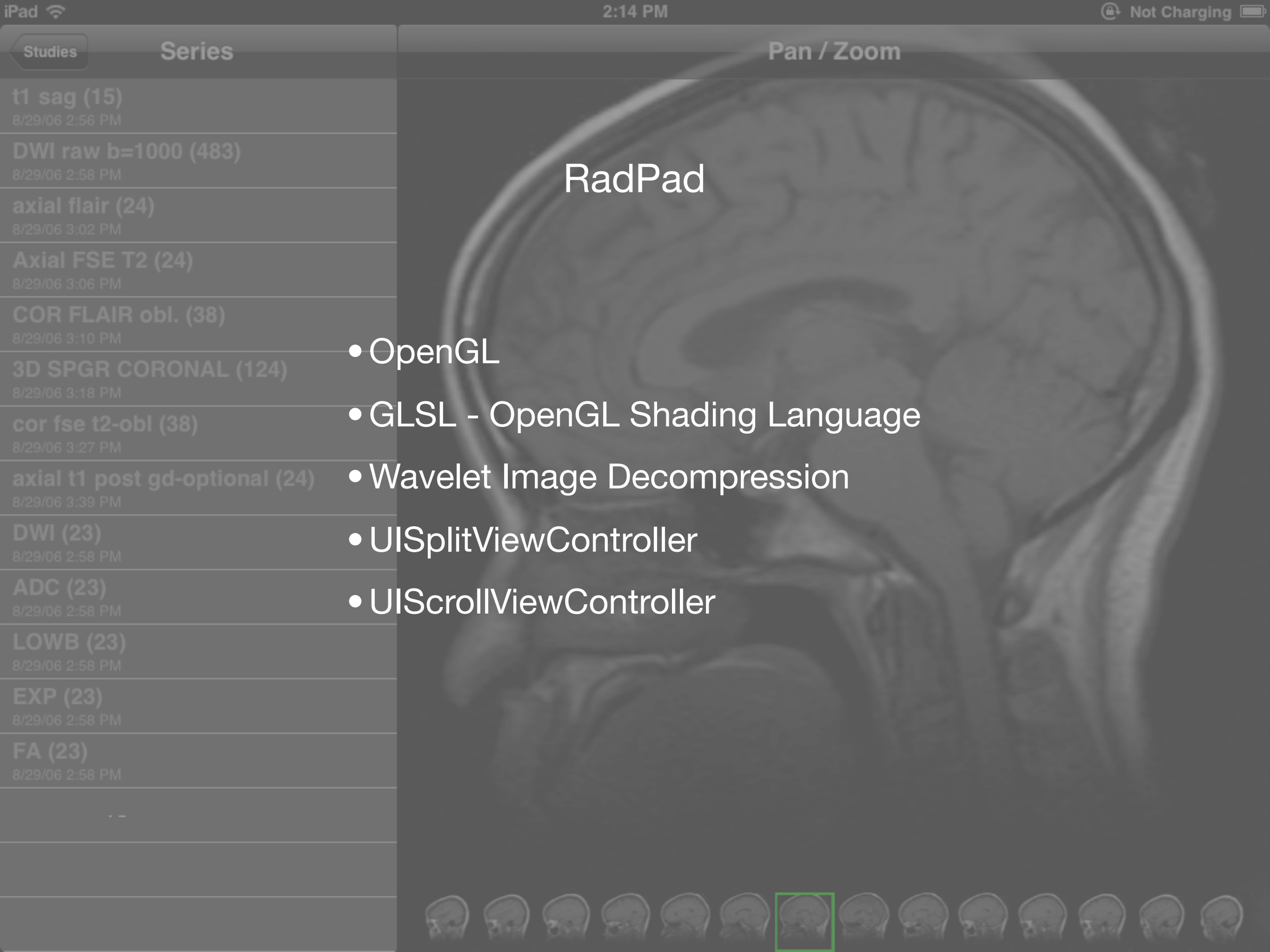


Beautiful Panoramas

- OpenGL
- GLSL - OpenGL Shading Language
- Proprietary Panorama Engine
- UIPopoverController

BMW Interior Panorama

- OpenGL
- GLSL - OpenGL Shading Language
- Proprietary Panorama Engine
- 3D Spatial Picking



RadPad

- OpenGL
- GLSL - OpenGL Shading Language
- Wavelet Image Decompression
- UISplitViewController
- UIScrollViewController