LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

mobile and distributed systems group

Praktikum iOS-Entwicklung

Wintersemester 2015/16

Prof. Dr. Linnhoff-Popien

Florian Dorfmeister, Marco Maier

Agenda

Views

- Hierarchie der Anzeige
- Zeichnen in einer eigenen View

View Controller

- Assoziation und Zusammenwirken von View Controller und seiner View
- Möglichkeiten der Initialisierung

Navigation zwischen Views

- TabBarViewController, NavigationController, ...
- Verwendung von Segues

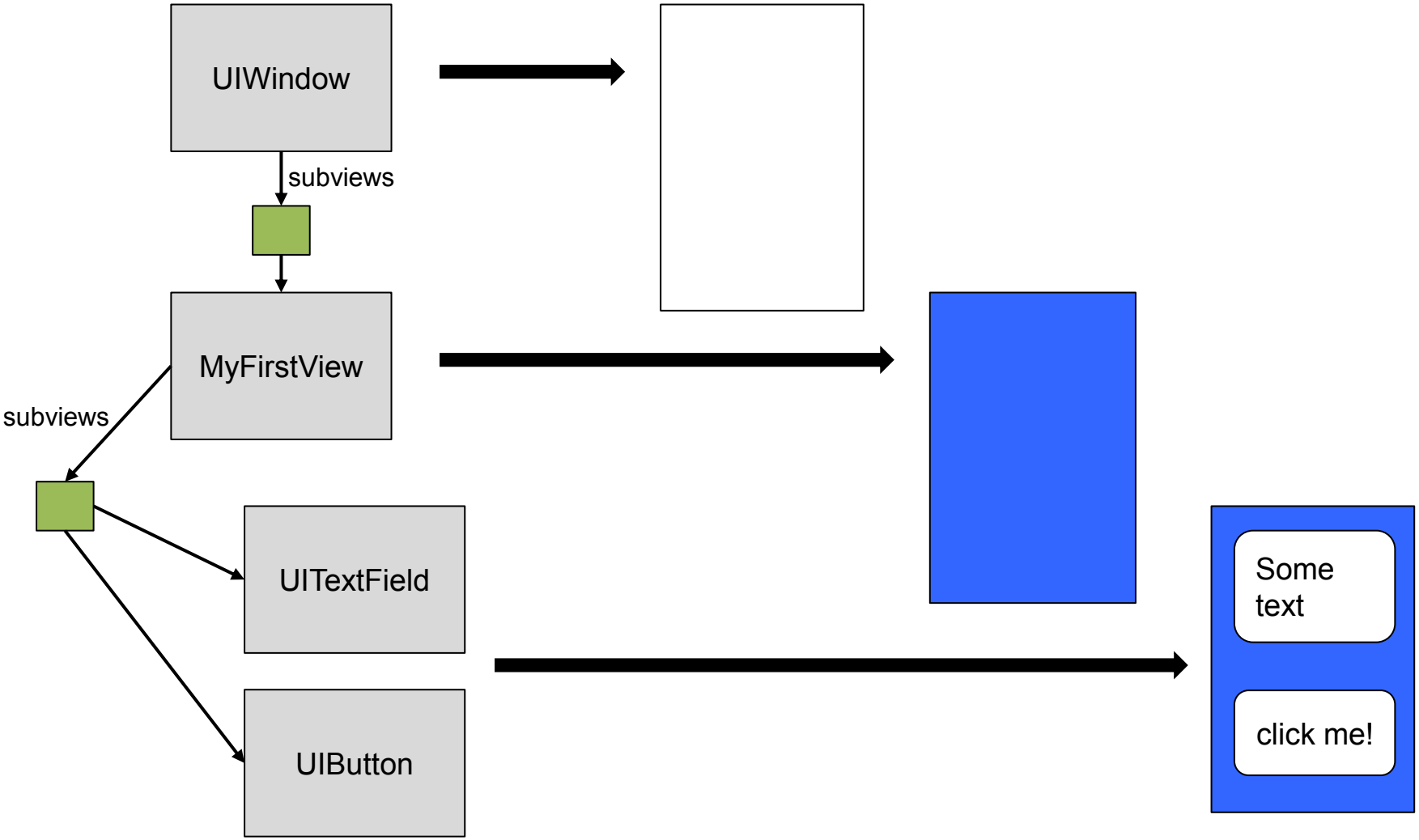
Views

Views in iOS

Was genau ist eine View?

- View im MVC-Konstrukt ist eine Instanz von `UIView` oder einer (selbst definierten) Unterklasse.
- View (`UIView`-Instanz) repräsentiert rechteckige Fläche,
  - die ein Koordinatensystem aufspannt und
  - in der alle Events (z.B. Touch- / Pinch-Gesten) von dieser `UIView`-Instanz behandelt werden.
- View kann sich selbst im Anwendungsfenster anzeigen – das Anwendungsfenster ist eine Instanz von `UIWindow`.
- View existiert immer in baumartiger Hierarchie, dessen Wurzel das Anwendungsfenster ist.
- View nur sichtbar, wenn sie der Hierarchie hinzugefügt wurde!

View Hierarchie



View Hierarchie

Abbildung der Hierarchie mit genau einem Verweis auf Eltern-Instanz (Superview)

```
(UIView *)superview
```

und einer Instanz von `NSArray`, das auf alle Kind-Instanzen (ChildViews) zeigt. (Nachfolgende werden über vorherige Array-Elemente gezeichnet!)

```
(NSArray *)subviews
```

Bei graphischer Erzeugung von Views (XIB-Files / Storyboards) erfolgt die Ausbildung der Eltern-Kind-Beziehungen automatisch

- Achtung: Die Referenz einer Childview auf seine Superview ist vom Typ `__weak`

Custom Views

Verwendung eigener (Custom) Views, um eigene UI-Elemente zu erzeugen

Erzeugung der View erfolgt in Code durch Erben von `UIView` (oder deren Unterklassen)

Achtung: Damit die eigene View verwendet werden kann, muss man sie

- entweder direkt im Code laden (Eltern-Kind-Beziehungen zwischen Views müssen dann manuell hergestellt werden)
- oder im Xib-File / Storyboard als "Custom View" im "Identity Inspector" des entsprechenden UI-Elements korrekt eintragen

Koordinatensystem

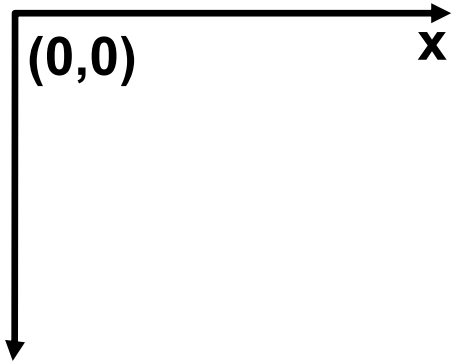
Jede View-Instanz besitzt einen Rahmen (Frame)

View stellt immer ein Rechteck dar

Frame spezifiziert die Größe und die Position einer View in Relation zu seiner Superview

Ursprung des Koordinatensystems ist oben links.

Einheiten der Koordinaten sind Punkte (Tupel der Form (x,y) – und keine Pixel!)



Core Graphics (CG) Framework

Bestimmung von Positionen und Dimensionen von Punkten und geometrischen Figuren erfolgt über Strukturen des Core Graphics Frameworks:

- Z.B. `CGFloat`, `CGPoint`, `CGSize`, `CGRect` (später mehr zu CG-Framework)

Beispiele (Swift verwendet die gleichen CG-Funktionen):

```
// CGPoint: C struct mit zwei CGFloat x & y
CGPoint p = CGPointMake(2.0, 45.2);
p.x += 20;

// CGSize: C struct mit zwei CGFloat width & height
CGSize s = CGSizeMake(100.0, 200.0);
s.height += 50;

// CGRect: C struct mit CGPoint origin und CGSize size
CGRect r = CGRectMake(45.0, 22.2, 300.0, 340.0);
r.size.height -= 40;
r.origin.y += 20;
```

Positionierung und Rotation

Views haben 3 wichtige Properties:

```
@property CGRect bounds;   // Fläche der UIView-Instanz

@property CGPoint center;  // Mittelpunkt der Fläche im Koordinatensystem
                           // der Superview!

@property CGRect frame;    // Fläche im Koordinatensystem der Superview,
                           // die das Rechteck bounds.size vollständig
                           // umschließt.
```

`frame.size` und `bounds.size` sind nicht immer deckungsgleich, da Views transformiert (z.B. rotiert) werden können!

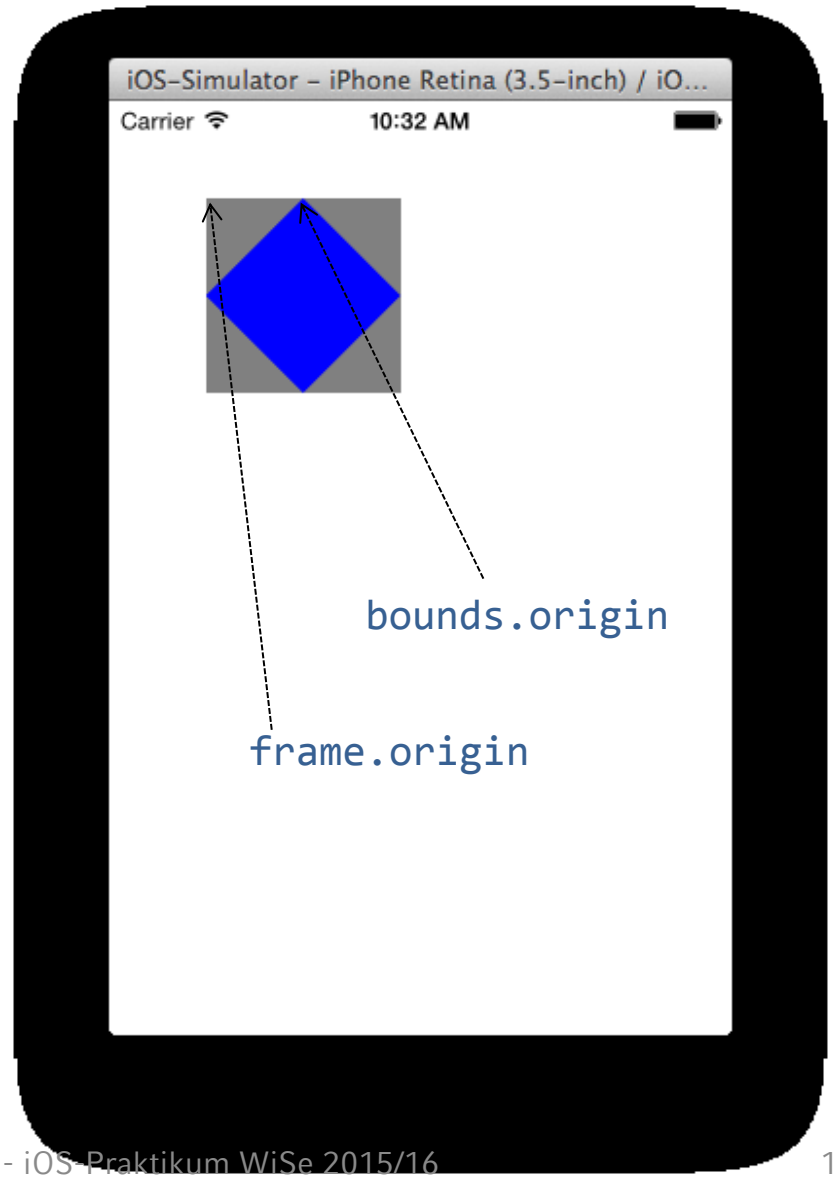
Positionierung und Rotation

View um 45° rotiert:

```
frame[origin=(50,50),size=(100,100)]
bounds[origin=(0,0), size=(71,71)]
```

In der Superview entspricht `bounds.origin` dem Punkt mit den Koordinaten (100, 50)!

`frame` und `center` sollten daher nur von der jeweiligen Superview verwendet werden!



Beispiel: Erzeugen einer eigenen View


```
//  CustomView.h

#import <UIKit/UIKit.h>

@interface CustomView : UIView

@end
```

```
//  CustomView.m

#import "CustomView.h"

@implementation CustomView

@end
```

Beispiel: Erzeugen einer eigenen View


```
//  AppDelegate.m
#import "AppDelegate.h"
#import "CustomView.h”
@implementation AppDelegate

-(BOOL)application:(UIApplication *)application
        didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {
    […]
    CGRect bigRect = CGRectMake(100, 100, 100, 100);
    CGRect smallRect = CGRectMake(50, 50, 50, 50);
    CustomView *bigView = [[CustomView alloc ] initWithFrame:bigRect ];
    CustomView *smallView = [[CustomView alloc ] initWithFrame:smallRect ];
    [bigView setBackgroundColor:[UIColor redColor]];
    [smallView setBackgroundColor:[UIColor blueColor]];
    [self.window addSubview:bigView];
    [self.window addSubview:smallView];
    […]
}
@end
```

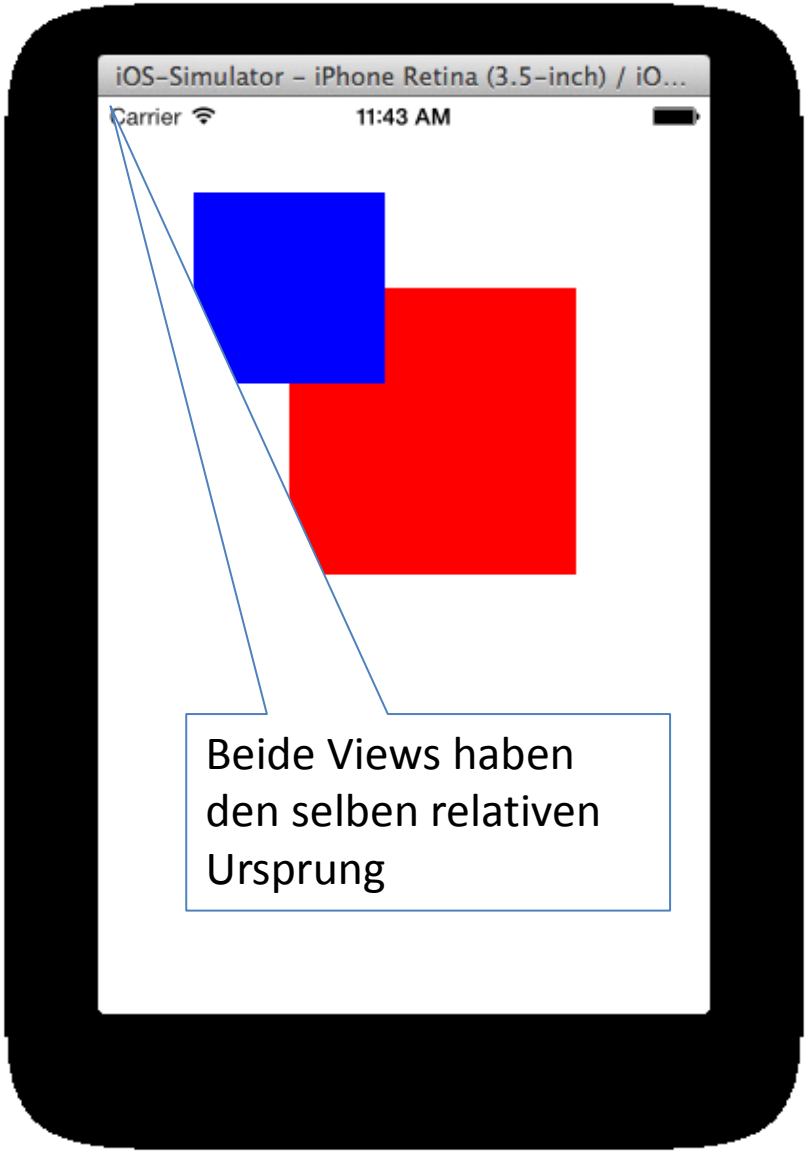
Beide Views werden dem Application Window hizugefügt

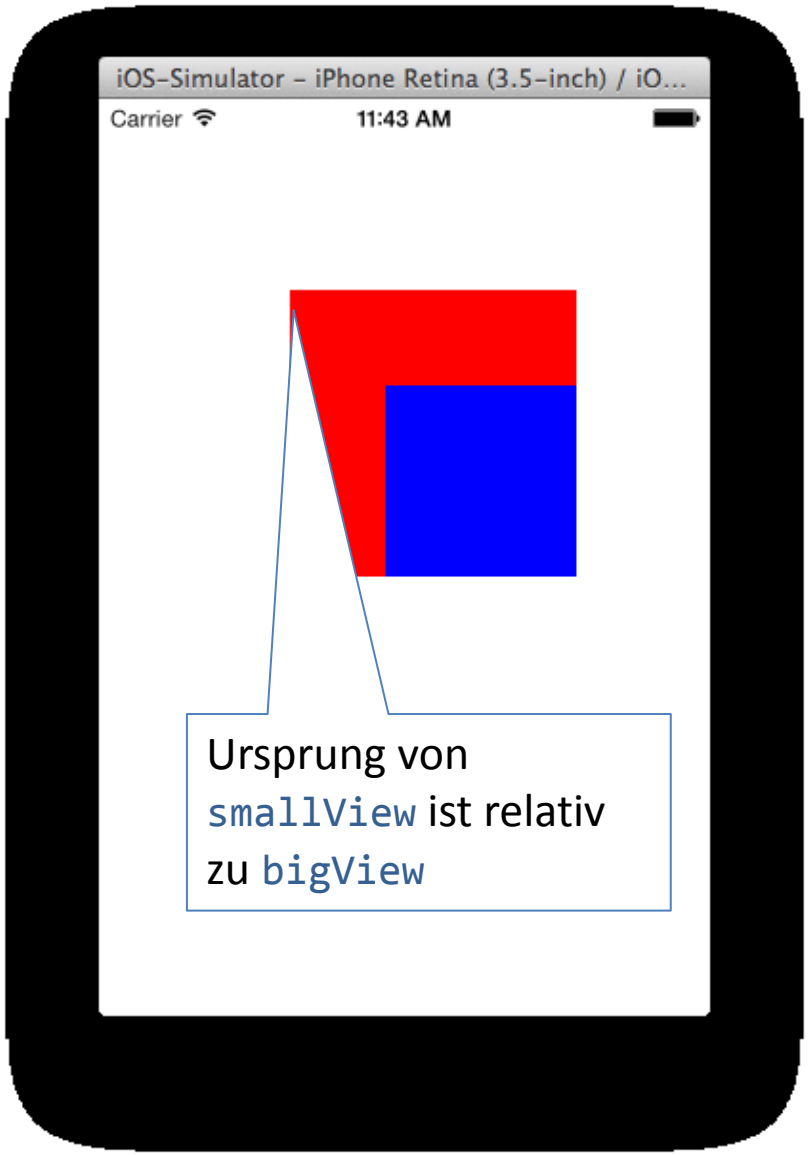
Beispiel: Erzeugen einer eigenen View


```
//  AppDelegate.m
#import "AppDelegate.h"
#import "CustomView.h”
@implementation AppDelegate

-(BOOL)application:(UIApplication *)application
        didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {
    […]
    CGRect bigRect = CGRectMake(100, 100, 100, 100);
    CGRect smallRect = CGRectMake(50, 50, 50, 50);
    CustomView *bigView = [[CustomView alloc ] initWithFrame:bigRect ];
    CustomView *smallView = [[CustomView alloc ] initWithFrame:smallRect ];
    [bigView setBackgroundColor:[UIColor redColor]];
    [smallView setBackgroundColor:[UIColor blueColor]];
    [self.window addSubview:bigView];
    [bigView addSubview:smallView];
    […]
}
@end
```

`smallView` wird Childview der `bigView`

Ergebnis


iOS-Simulator – iPhone Retina (3.5-inch) / iO...
Carrier
11:43 AM
Beide Views haben den selben relativen Ursprung


iOS-Simulator – iPhone Retina (3.5-inch) / iO...
Carrier
11:43 AM
Ursprung von smallView ist relativ zu bigView

Zeichnen in der eigenen View

Überschreiben der Methode `drawRect:`

Vorgehensweise (nur eine Möglichkeit von mehreren!):

- Erzeugen einer Referenz auf den Grafikkontext der View
  `CGContextRef ctx = UIGraphicsGetCurrentContext();`
- Anpassen des Kontext und zeichnen geometrischer Figuren mit weiteren Instruktionen des Core Graphics Frameworks)

```
//  CustomView.m
#import "CustomView.h"

@implementation CustomView
-(void)drawRect:(CGRect)rect {
    CGContextRef ctx = UIGraphicsGetCurrentContext();
}
@end
```

Zeichnen in der eigenen View

`CGContextRef` (definiert als `CGContext*`) ist ein **Zeiger** auf eine Struktur vom Typ `CGContext`

- Suffix `Ref` erleichtert die Unterscheidung zwischen C-Zeigern auf Strukturen und Objective-C-Referenzen
- Es gibt für fast jede CG-Struktur einen entsprechenden `Ref`-Typ (Ausnahmen: "kleine" Strukturen wie `CGPoint`, `CGRect`, ...)

Kontext entscheidet über die Ausgabe der Zeichnungen.

- Z.B. Bildschirm, Bitmap, Drucker, PDF, …

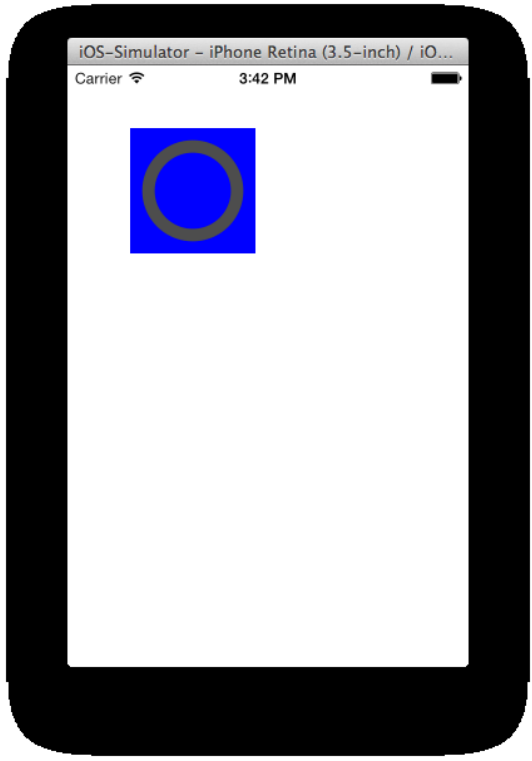
Property `bounds` (nicht `frame` ...!) liefert die Größe der Zeichenfläche

Alle Zeichenoperationen müssen innerhalb von `bounds` erfolgen

- Alles andere wird abgeschnitten!

Beispiel: Kreis in Rechteck


```
//  CustomView.m

#import "CustomView.h"

@implementation CustomView

-(void)drawRect:(CGRect)rect {
    CGContextRef ctx = UIGraphicsGetCurrentContext();

    CGRect bounds = self.bounds;
    CGPoint center =
        CGPointMake(bounds.origin.x + bounds.size.width/2
            bounds.origin.y + bounds.size.height/2);
    float radius = hypot(bounds.size.width, bounds.size.h
    CGContextSetLineWidth(ctx, 10);
    CGContextSetRGBStrokeColor(ctx, 0.3, 0.3, 0.3, 1.0);
    CGContextAddArc(ctx, center.x, center.y, radius, 0.0, M_PI * 2.0, YES);
    CGContextStrokePath(ctx);
}
@end
```



Zeichnen in der eigenen View

Kontext ist nur während des `drawRect:` Aufrufs gültig.

Wichtig: `drawRect:` **NIEMALS** selbst aufrufen (Kontext ist unklar!)

Daher: iOS benachrichtigen, dass View neu gezeichnet werden muss:

```
-(void)setNeedsDisplay;

-(void)setNeedsDisplayInRect:(CGRect)aRect;
```

Core Graphics Framework

Core Graphics bietet zweidimensionales Rendering und stellt Funktionen bereit, um

- Pfade zu zeichnen, zu färben und zu transformieren
- Bilder anzuzeigen und zu erstellen
- PDFs anzuzeigen und zu erstellen…

Core Graphics Funktionen beginnen alle mit CG…

Core Graphics vs. UIKit

Core Graphics ist eine in C implementierte 2D API, d.h. es gibt lediglich Funktionen und Strukturen (keine Objekte und Methoden)

Der Grafik Kontext einer View kann aber auch auf objektorientierte Art und Weise modifiziert werden

- Verwendung von UIKit Klassen und deren Wrapper-Methoden
- Unterschied zur Verwendung von CG-Funktionen:
  - Bei der Verwendung von UIKit kümmern sich die entsprechenden Instanzen selbst um die Allokation, Modifikation und Freigabe des Grafik Kontext.

- Alles was man mit UIKit machen kann, kann man auch direkt mit dem GC-Framework erzielen
- **Umgekehrt gilt dies nicht!**
  - Beispiel: Gradienten

Beispiel: Zeichnen einer roten Linie von a nach b

- mit UIKit

```
[[UIColor colorWithRed:1.0 green:0.0 blue:0.0 alpha:1.0] setStroke];

UIBezierPath *path = [UIBezierPath bezierPath];
[path moveToPoint:a];
[path addLineToPoint:b];

[path stroke];
```

- mit GC-Framework

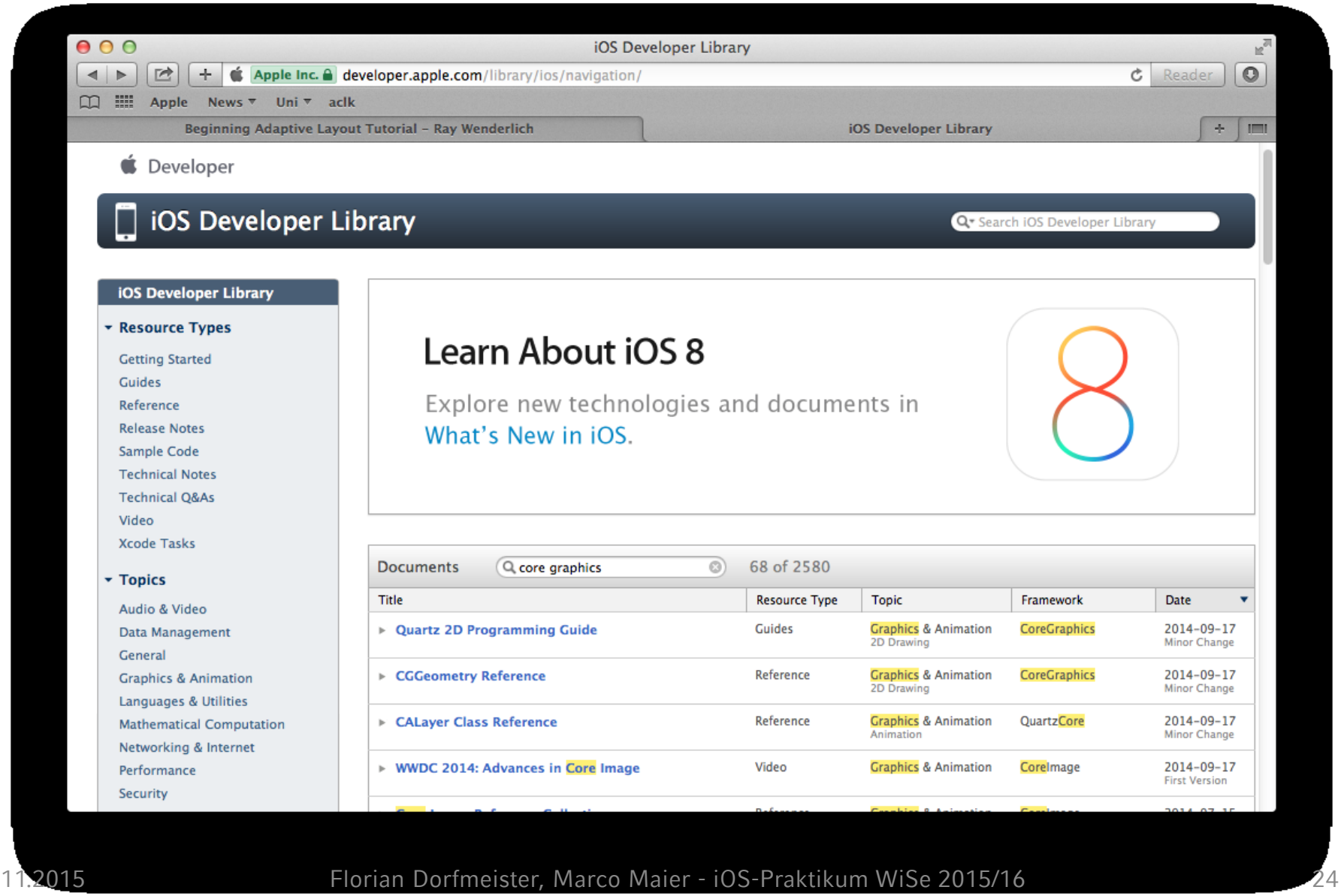
```
CGContextRef currentContext = UIGraphicsGetCurrentContext();
CGContextSetRGBStrokeColor(currentContext, 1, 0, 0, 1);

CGMutablePathRef path = CGPathCreateMutable();
CGPathMoveToPoint(path, NULL, a.x, a.y);
CGPathAddLineToPoint(path, NULL, b.x, b.y);
CGContextAddPath(currentContext, path);

CGContextStrokePath(currentContext);
CGPathRelease(path);
```

Hinweis zur Speicherverwaltung:

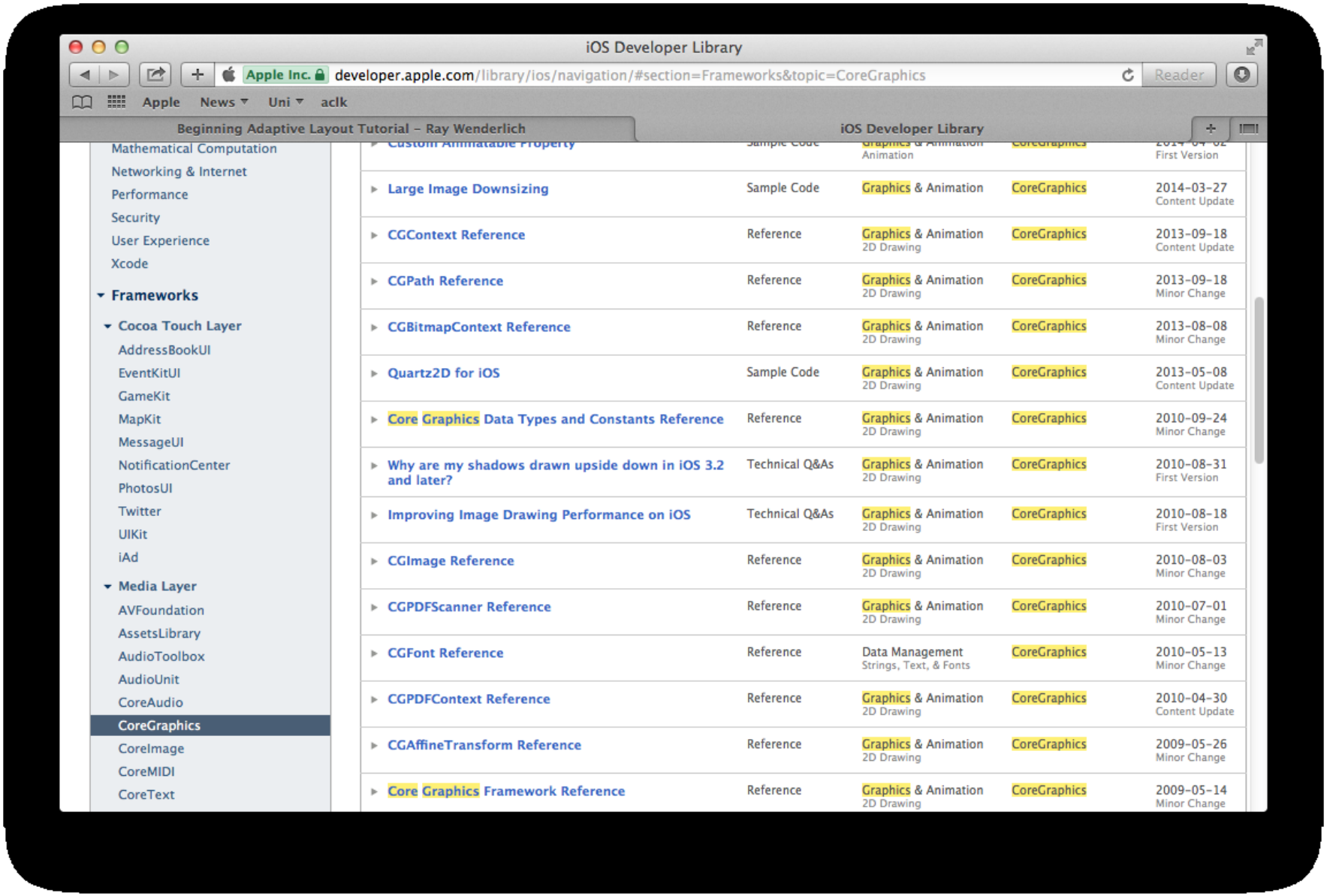
- Wenn man ein Core Graphics "Objekt" mit einer Funktion erstellt, die das Wort "Copy" oder "Create" enthält, dann muss man in **Objective C** (in Swift i.d.R. nicht mehr notwendig) nach Beendigung der Arbeit die dazugehörige Release-Methode aufrufen!
  - Als erstes Argument der jeweiligen Release-Methode übergibt man immer einen Zeiger auf das entsprechende "Objekt"

- Mit den Funktionen `CGContextSaveGState` und `CGContextRestoreGState` kann man den Zustand einer Struktur vom Typ `CGContext` zwischenspeichern
  - Beispiele:
    - Hinzufügen einer Untergruppe von Objekten in einem geclippten Bereich
    - Hinzufügen von Schattierung zu einer Untergruppe von Objekten
    - ...

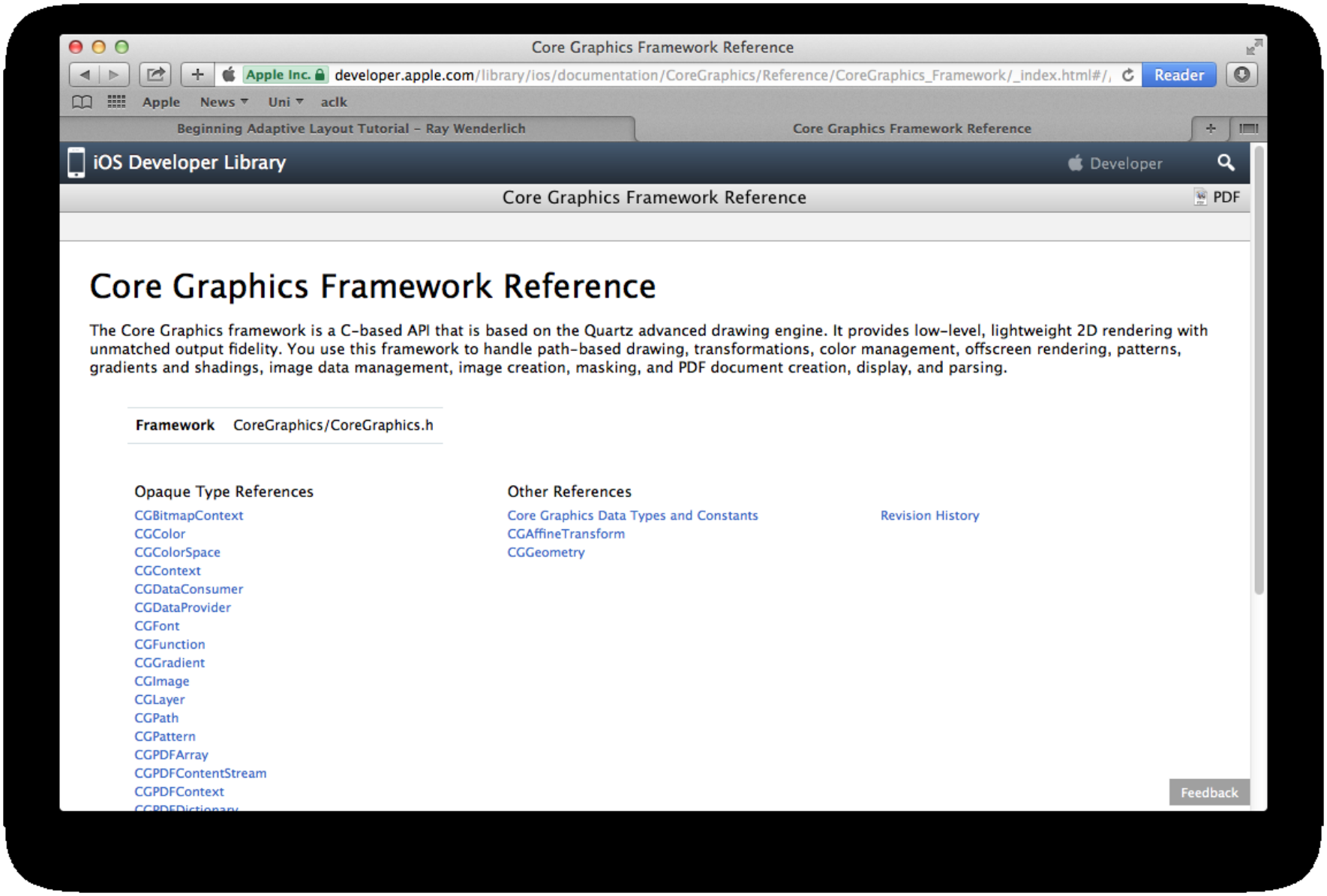
Core Graphics & Dokumentation

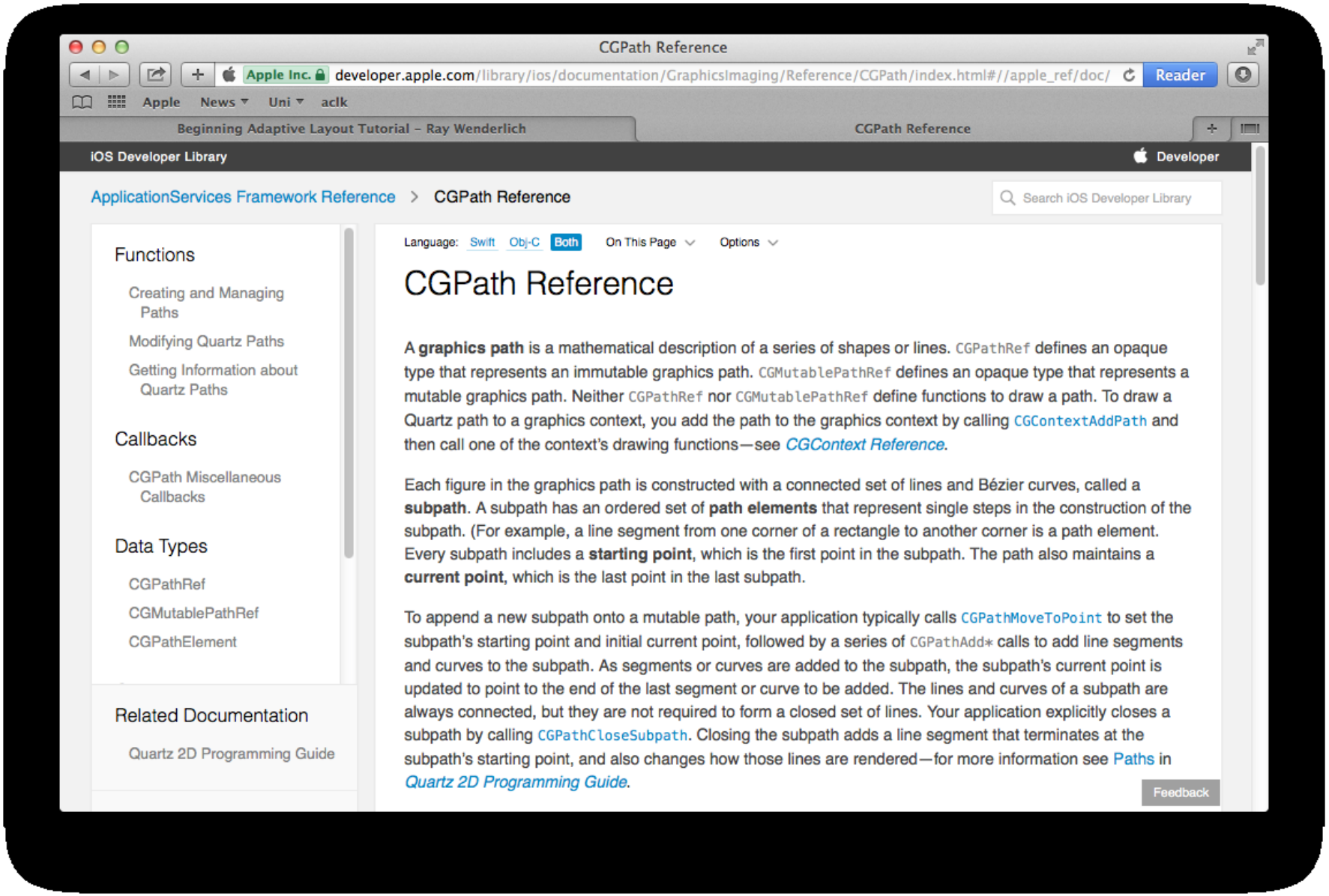


Core Graphics & Dokumentation



Core Graphics & Dokumentation




CGPath Reference
Apple Inc. developer.apple.com/library/ios/documentation/GraphicsImaging/Reference/CGPath/index.html#//apple_ref/doc/
Reader
Apple News Uni aclk
Beginning Adaptive Layout Tutorial – Ray Wenderlich
CGPath Reference
iOS Developer Library
Developer
ApplicationServices Framework Reference > CGPath Reference
Search iOS Developer Library
Functions
Creating and Managing Paths
Modifying Quartz Paths
Getting Information about Quartz Paths
Callbacks
CGPath Miscellaneous Callbacks
Data Types
CGPathRef
CGMutablePathRef
CGPathElement
Related Documentation
Quartz 2D Programming Guide
Language: Swift Obj-C Both
On This Page
Options
CGPath Reference
A graphics path is a mathematical description of a series of shapes or lines. CGPathRef defines an opaque type that represents an immutable graphics path. CGMutablePathRef defines an opaque type that represents a mutable graphics path. Neither CGPathRef nor CGMutablePathRef define functions to draw a path. To draw a Quartz path to a graphics context, you add the path to the graphics context by calling CGContextAddPath and then call one of the context's drawing functions—see CGContext Reference.
Each figure in the graphics path is constructed with a connected set of lines and Bézier curves, called a subpath. A subpath has an ordered set of path elements that represent single steps in the construction of the subpath. (For example, a line segment from one corner of a rectangle to another corner is a path element. Every subpath includes a starting point, which is the first point in the subpath. The path also maintains a current point, which is the last point in the last subpath.
To append a new subpath onto a mutable path, your application typically calls CGPathMoveToPoint to set the subpath's starting point and initial current point, followed by a series of CGPathAdd* calls to add line segments and curves to the subpath. As segments or curves are added to the subpath, the subpath's current point is updated to point to the end of the last segment or curve to be added. The lines and curves of a subpath are always connected, but they are not required to form a closed set of lines. Your application explicitly closes a subpath by calling CGPathCloseSubpath. Closing the subpath adds a line segment that terminates at the subpath's starting point, and also changes how those lines are rendered—for more information see Paths in Quartz 2D Programming Guide.
Feedback

VIEW CONTROLLER

View Controller: Was ist das?

Ein View Controller entspricht dem Controller im MVC Konstrukt

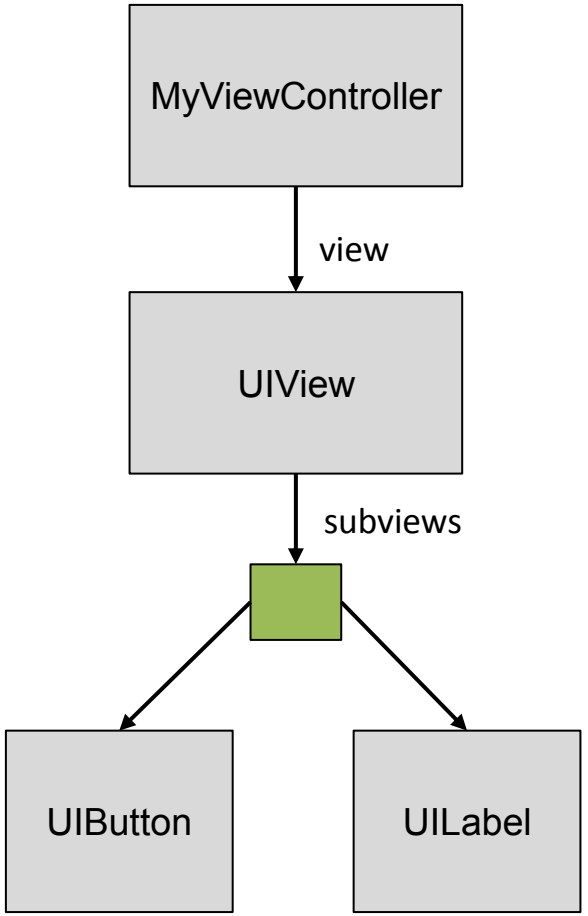
- Verbindung zwischen View und Modell

Eigene (Custom) View Controller Klassen erben immer von der Klasse `UIViewController`.

Ein View Controller verwaltet genau eine View-Hierarchie

- genau eine Reference auf eine Instanz der (Unter-)Klasse (von) `UIView` in `self.view`.

Die View-Hierarchie selbst kann natürlich weiter Subviews beinhalten



Programmatisches Laden einer View

Unterklassen von `UIViewController` können ihre View-Hierarchie programmatisch (also im Code) durch Überschreiben der Methode `loadView:` erzeugen.

```
//  MyViewController.m

#import "MyViewController.h"
#import "MyCustomView.h"

@implementation MyViewController

-(void)loadView {
    // Hier kein Aufruf von [super loadView] notwendig!
    // Wir kümmern uns selbst um die Initialisierung der View
    CGRect frame = [[UIScreen mainScreen] bounds];
    MyCustomView *view = [[MyCustomView alloc] initWithFrame:frame];
    self.view = view;
}
@end
```

Initialisieren einer View mit XIB-Datei

Programmatisches Initialisieren einer View-Hierarchie i.d.R. zu aufwendig

Meist setzt sich die Anzeige (Screen) aus vielen "statischen" vordefinierten UI-Elemente zusammen

Daher: Verwendung von XIB-Dateien bzw. Storyboards zur Initialisierung der UI

Storyboard bzw. XIB-Datei "archiviert" die darin enthaltenen UI-Objekte (XML-Format)

UI wird beim Laden des Storyboards bzw. der XIB-Datei in den Speicher geladen

Ownership und Responsibility

Ein View Controller ist der alleinige Besitzer seiner View (und der Subviews, die er erzeugt)

Ein View Controller ist verantwortlich für das Erzeugung und die Freigabe dieser Views

Bei der Initialisierung der View mittels XIB-Dateien / Storyboards erhält jeder View Controller **seine eigenen Kopie** der View

Bei einer programmatischen Erzeugung dürfen verschiedene View Controller jedoch **niemals dieselbe Instanz einer (Sub-)View** referenzieren!

Initialisieren der Anzeige mit XIB-Datei

Empfohlene Vorgehensweise:

- Benennung der XIB-Datei wie Klasse des dazugehörige Controllers
- Methode `loadView` (von `UIViewController`) sucht dann standardmäßig nach einer XIB-Datei mit dem Namen der Klasse des View Controllers

Alternativ:

- Explizites Laden der View über die Methode `initWithNibName:bundle:` (von `UIViewController`)

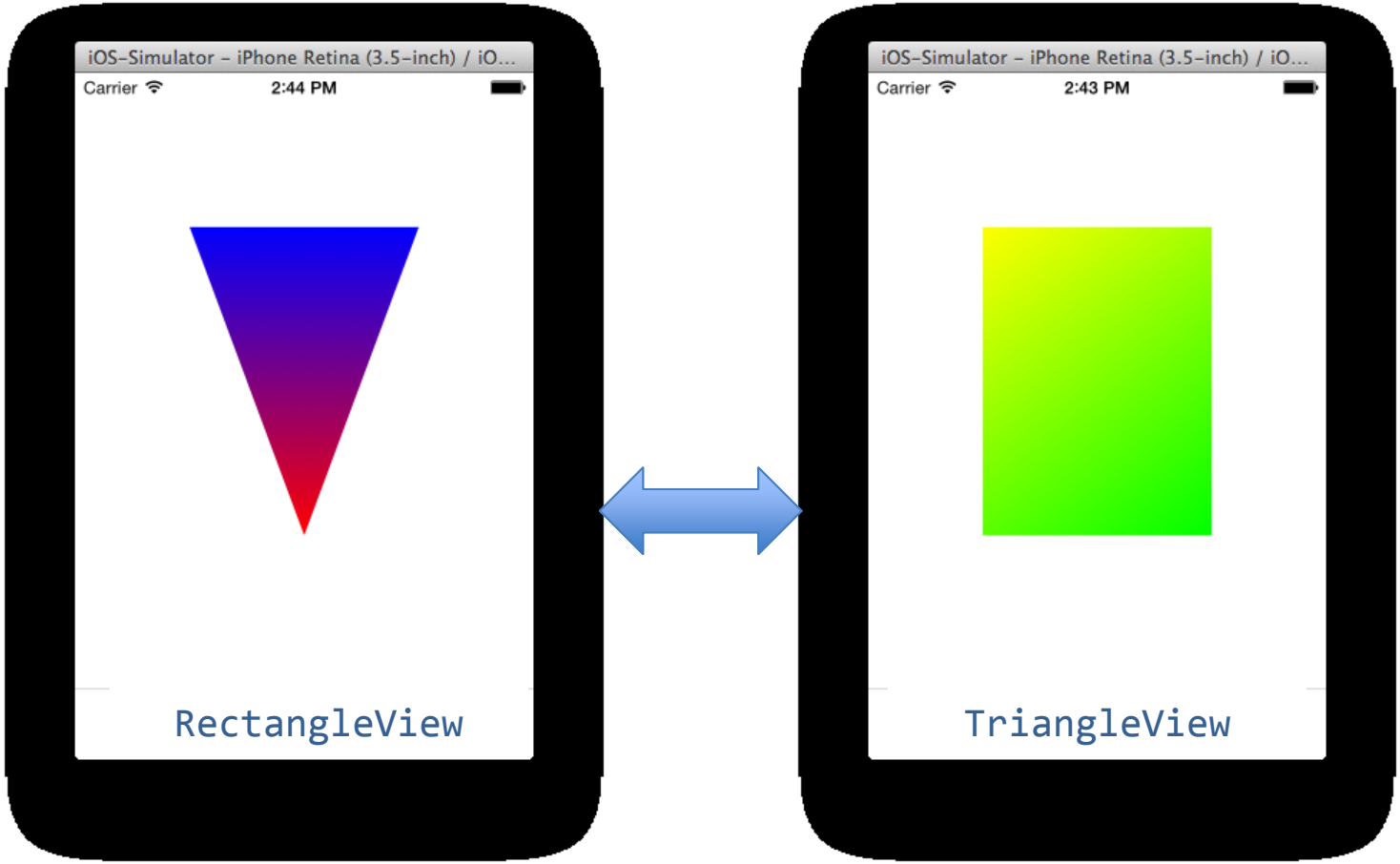
Wichtig:

- Korrekte Assoziation des *File's Owner*
- erzeugt beim Übersetzen die Verbindung mit dem Code

Navigation zwischen Views

Navigation zwischen Views

Wie navigiere ich zwischen zwei Views?



Navigation zwischen Views

Bisher:

- Custom View wird jeweils direkt `self.window` zugeordnet
- Nur eine View pro Anwendung und keine Navigation möglich!

```
#import "GeometriesAppDelegate.h"
#import "RectangleViewController.h" //ODER: TriangleViewController.h

@implementation GeometriesAppDelegate
-(BOOL)application:(UIApplication *)application
         didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {
    self.window = [[UIWindow alloc] initWithFrame:[[UIScreen mainScreen]
bounds]];

    // ENTWEDER ...:
    UIViewController *vc = [RectangleViewController new];
    // ... ODER:
    // UIViewController *vc = [TriangleViewController new];

    self.window.rootViewController = vc;

    [self.window makeKeyAndVisible];
    return YES;
}
@end
```

Navigation zwischen Views

Verwendung von **Container View Controllern** zur Navigation:

- `UITabBarController`
  - zur Verwaltung einer Menge voneinander unabhängiger Screens
- `UINavigationController`
  - zur Verwaltung einer Menge voneinander abhängiger Screens (stacked views)

Zusätzlich für iPad:

- `UIPageViewController`
  - zur Detaildarstellung von Inhalten durch einen eigenen Screen (Navigation wie in einem Buch)
- `UISplitViewController`
  - zur gleichzeitigen Darstellung zweier Screens (Master-Detail-View; jeder wird von einem eigenen View Controller verwaltet)
- `UIPopoverController`
  - zur Darstellung der Details eines Screens in einem eigenen Subscreen (nur iPad)

Navigation zwischen Views

Quelle: https://developer.apple.com

`UITabBarController`

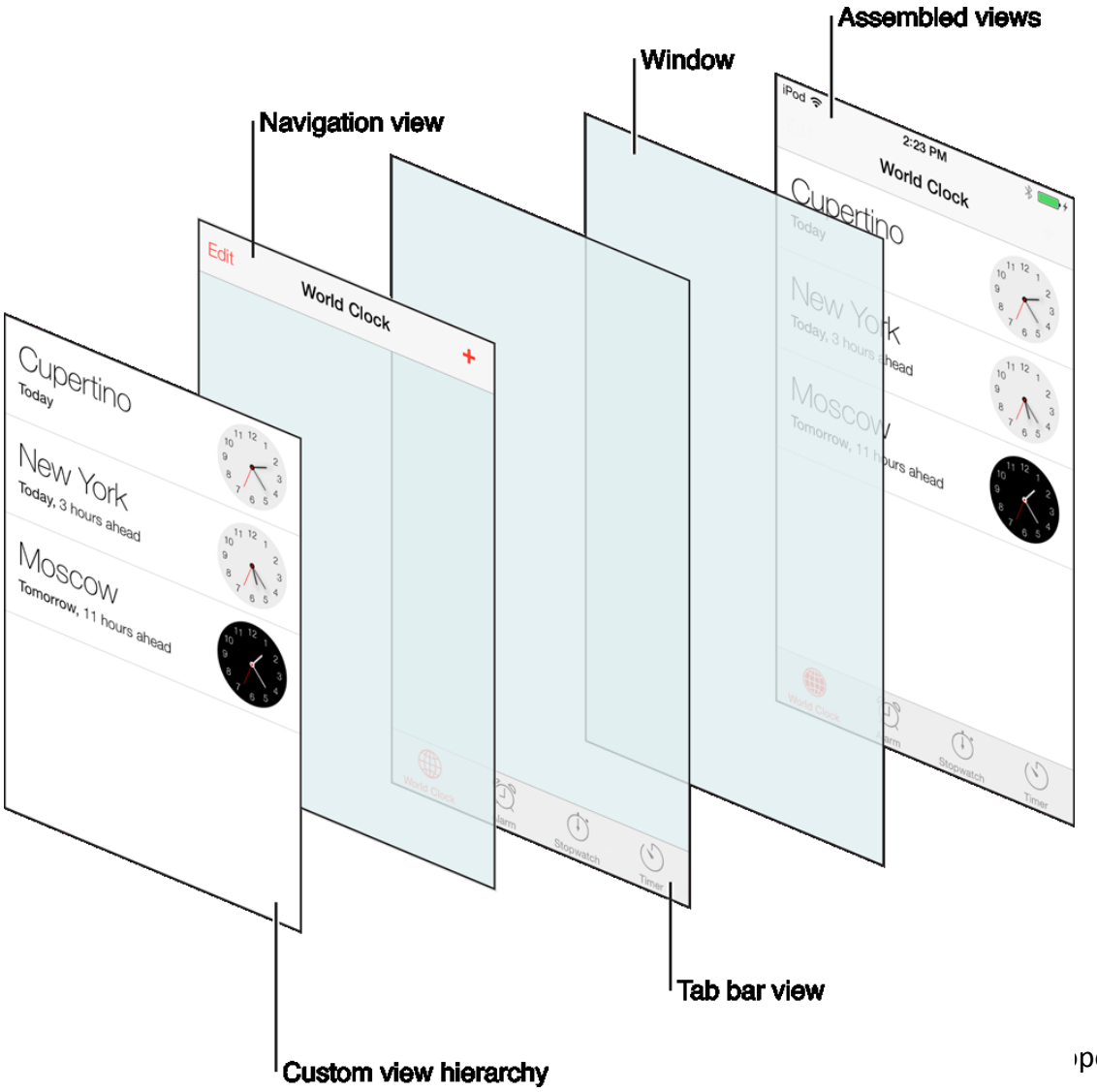
- Root View Controller der gesamten Anwendung
- Verwaltet ein `NSArray` mit allen View Controllern.
  → Alle View Controller müssen beim Starten der Anwendung initialisiert werden!

`UITabBarController` verwaltet zwei Subviews

- `UITabBar`
- `UIView` des aktuellen View Controllers!

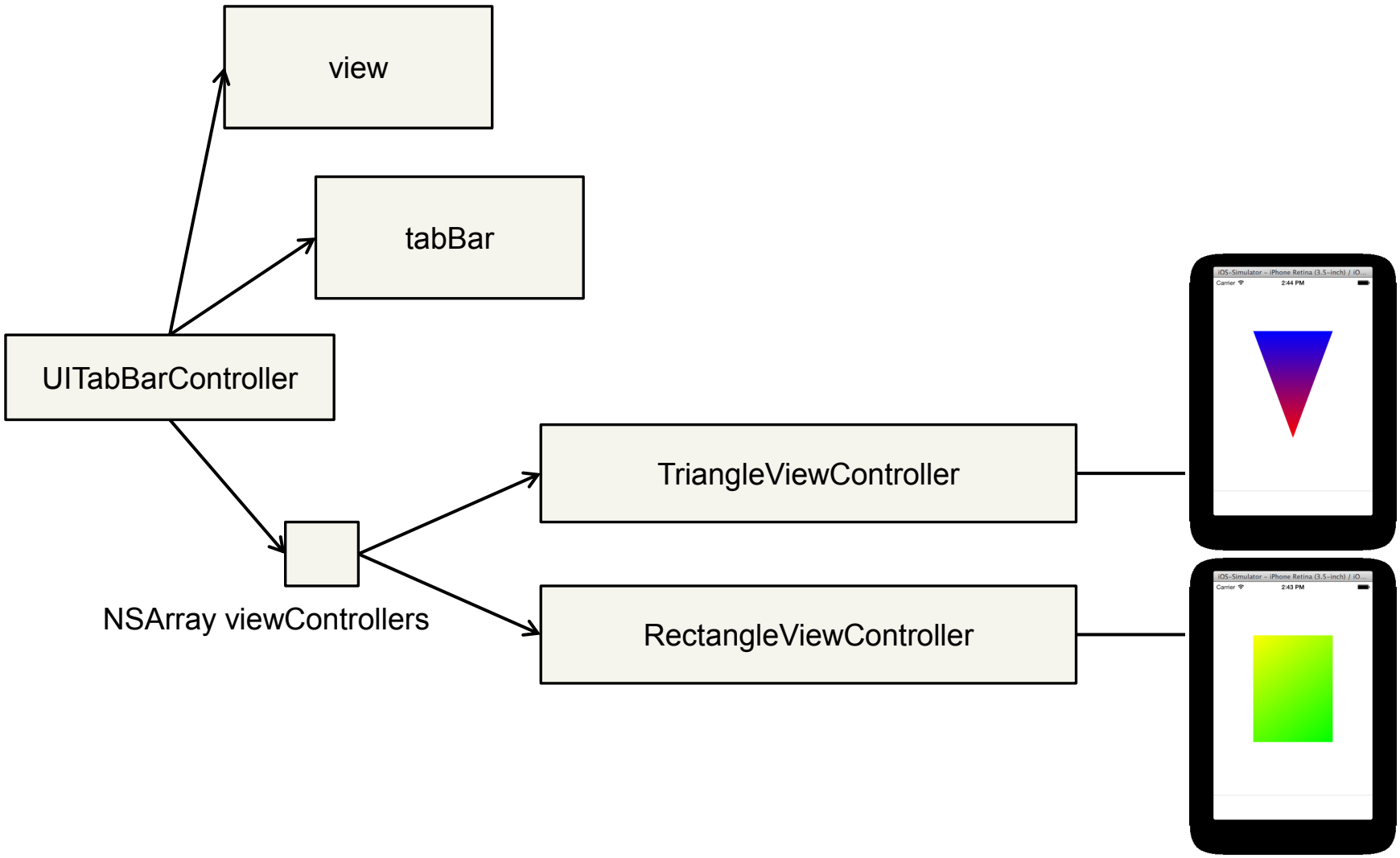
`UITabBarController` verwenden automatisch iOS-Standard-Designelemente

iOS-Standard-Designelemente



per.apple.com

Beispiel: Show Geometries



Beispiel: Show Geometries (Views)


```
// TriangleView.m

#import "TriangleView.h"

@implementation TriangleView

-(void)drawRect:(CGRect)rect {

    // Siehe Hausaufgabe

}
```

```
// RectangleView.m

#import "RectangleView.h"

@implementation RectangleView

-(void)drawRect:(CGRect)rect {

    // Siehe Hausaufgabe

}
```

Beispiel: Show Geometries (View Controller)


```
// TriangleViewController.m

#import "TriangleViewController.h"
#import "TriangleView.h"

@implementation TriangleViewController

// Programmatisches erzeugen der View
-(void)loadView {
    CGRect frame = [[UIScreen mainScreen] bounds];
    TriangleView *triangleView = [[TriangleView alloc] initWithFrame:frame];
    self.view = triangleView;
}

// Ab hier sind sicher alle Referenzen der (Sub-)Views initialisiert
-(void)viewDidLoad {
    [super viewDidLoad];
    // damit Tab Bar die View nicht überlappt (seit iOS7 notwendig)
    self.tabBarController.tabBar.translucent = NO;
}

@end
```

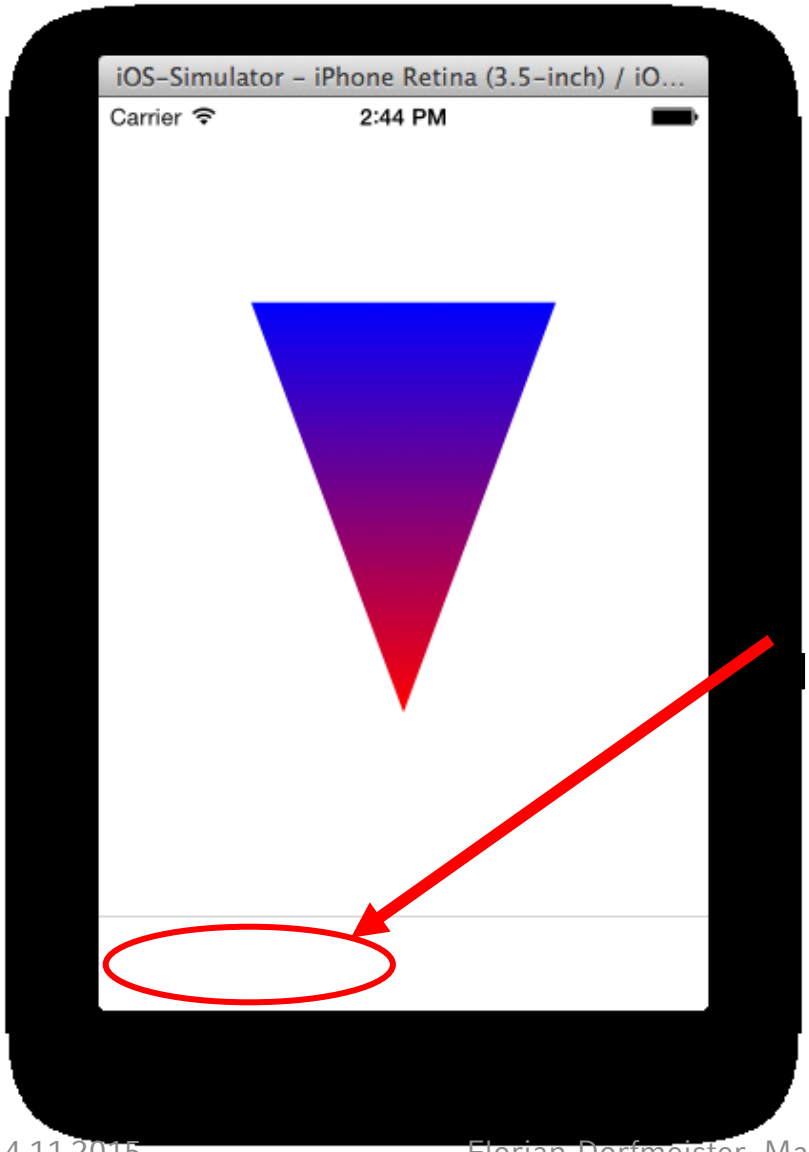
Beispiel: Show Geometries (View Controller)


```
// RectangleViewController.m

#import "RectangleViewController.h"
#import "RectangleView.h"

@implementation RectangleViewController

// Programmatisches erzeugen der View
-(void)loadView {
    CGRect frame = [[UIScreen mainScreen] bounds];
    RectangleView *rectangleView = [[RectangleView alloc] initWithFrame:frame];
    self.view = rectangleView;
}

// Ab hier sind sicher alle Referenzen der (Sub-)Views initialisiert
-(void)viewDidLoad {
    [super viewDidLoad];
    // damit Tab Bar die View nicht überlappt (seit iOS7 notwendig)
    self.tabBarController.tabBar.translucent = NO;
}

@end
```

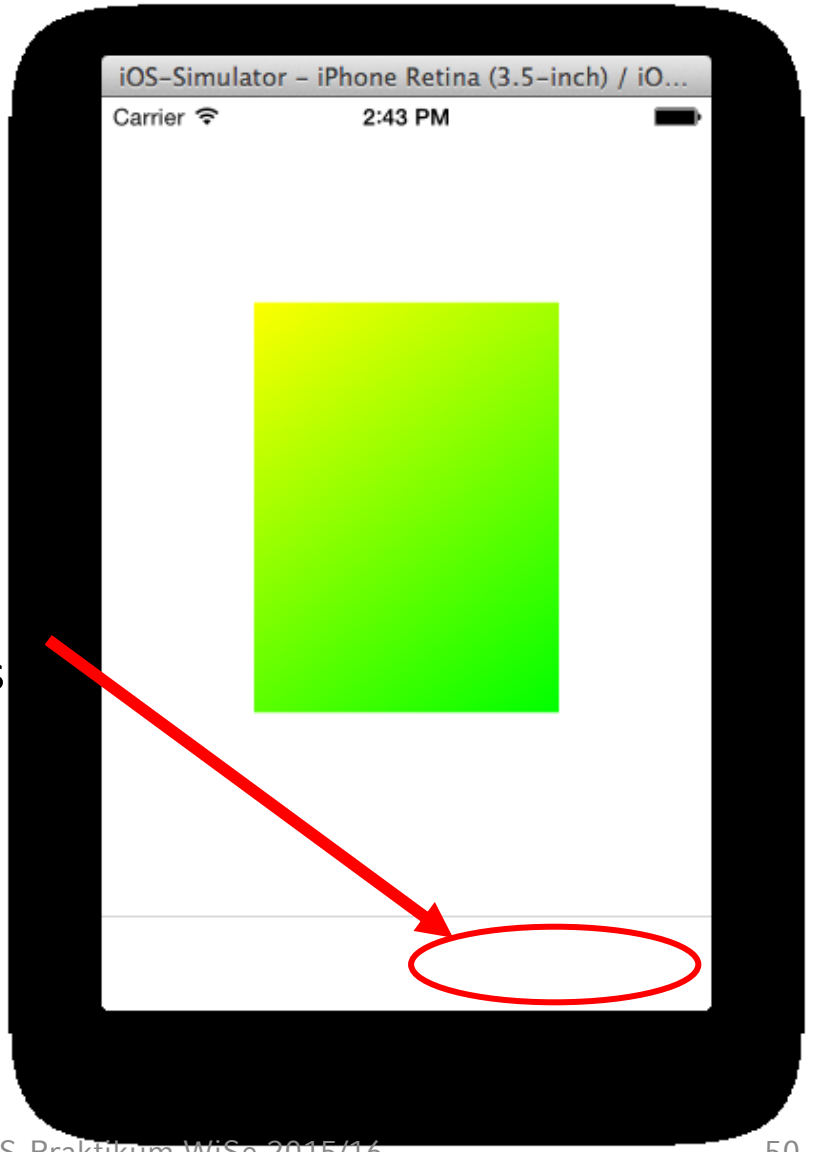
Beispiel: Show Geometries (AppDelegate)


```
#import "GeometriesAppDelegate.h"

#import "RectangleViewController.h"
#import "TriangleViewController.h"

@implementation GeometriesAppDelegate

-(BOOL)application:(UIApplication *)application
    didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {
    self.window = [[UIWindow alloc] initWithFrame:[[UIScreen mainScreen]
bounds]];
    RectangleViewController *rvc = [RectangleViewController new];
    TriangleViewController *tvc = [TriangleViewController new];

    UITabBarController *tabController = [[UITabBarController alloc] init];
    NSArray *viewControllers = [NSArray arrayWithObjects:tvc, rvc, nil];
    [tabController setViewControllers:viewControllers animated:true];

    [[self window] setRootViewController:tabController];

    self.window.backgroundColor = [UIColor whiteColor];
    [self.window makeKeyAndVisible];
    return YES;
}
@end
```

Beispiel: Show Geometries



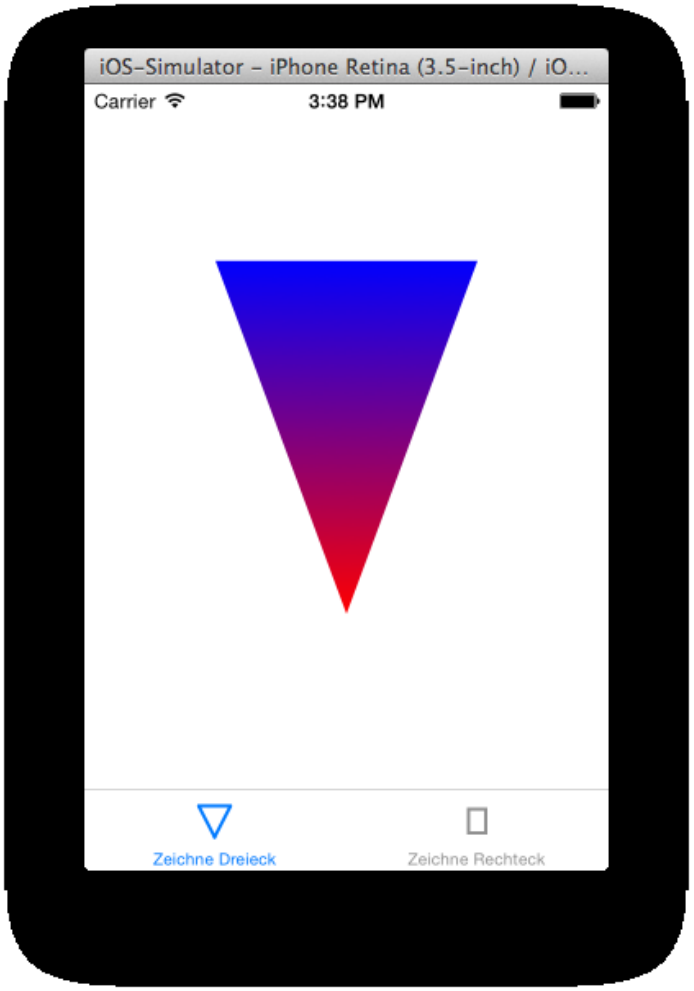
Problem:
leere Tabs

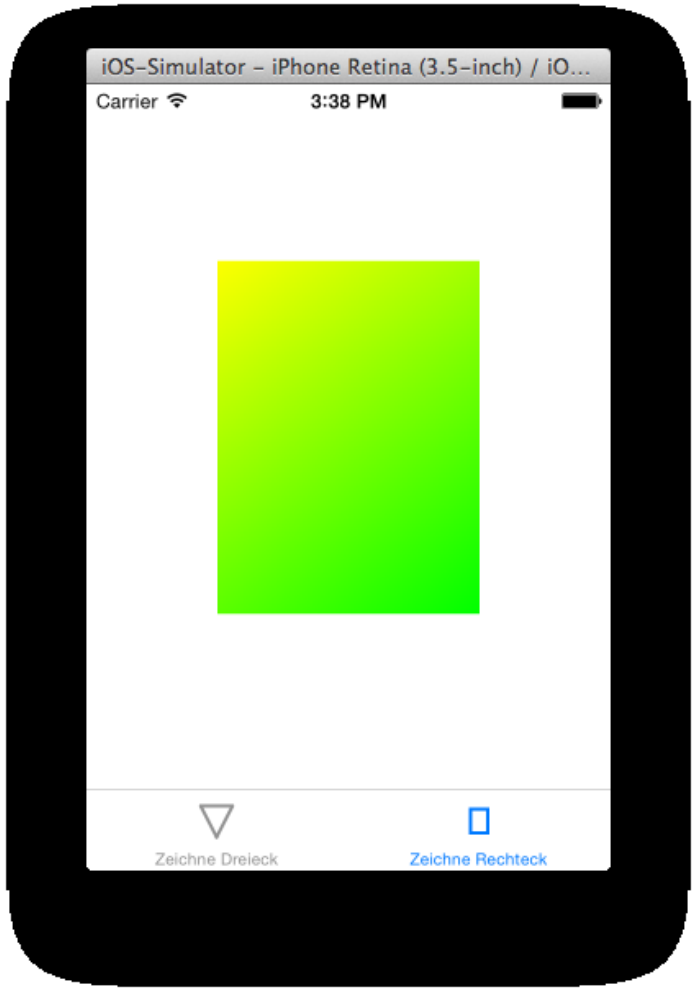


Beispiel: Show Geometries (Weitere Anpassungen)


```
// TriangleViewController.m

#import "TriangleViewController.h"
#import "TriangleView.h"

@implementation TriangleViewController

[...]

-(instancetype)init {
    self = [super init];
    if (self) {
        self.tabBarItem.title = @"Zeichne Dreieck";
        self.tabBarItem.image = [UIImage imageNamed:@"triangle.png"];
    }
    return self;
}
@end
```

Ergebnis





Mehr [illegible] idlines [illegible]
https://developer.apple.com/library/ios/documentation/userexperience/conceptual/mobilehig/

View Controller: Initialisierung

View Controller

- kann verwendet werden, sobald er initialisiert wurde (`alloc/init`)
- designierter Initialisierer: `initWithNibName:bundle:`
  - `nibName`:
    - Der Name der XIB-Datei, mit der der View Controller initialisiert werden soll
  - `bundle`:
    - Das Bundle, das die XIB-Datei enthält
  - Wenn beide Argumente `nil`:
    - Es wird eine XIB-Datei gesucht, die denselben Namen trägt, wie der View Controller
    - Die XIB-Datei wird im Application-Bundle gesucht

View Controller: Initialisierung

Explizites Laden von XIB-Datei

```
[...]

@implementation TriangleViewController
[...]

-(instancetype)initWithNibName:(NSString *)nibNameOrNil
        bundle:(NSBundle *)nibBundleOrNil {
    // anstelle von...
    // self = [super initWithNibName:nibNameOrNil bundle:nibBundleOrNil];

    // ...schreibt man
    NSBundle *appBundle = [NSBundle mainBundle];
    self = [super initWithNibName:@"MyViewController" bundle:appBundle];

    if (self) {
        [...]
    }
    return self;
}
@end
```

View Controller: Lazy Loading

Existierende UI-Elemente verbrauchen Speicher → UI-Elemente sollten

- so **früh wie nötig** und
- so **spät wie möglich**

geladen werden

View Controller lädt seine View erst unmittelbar **vor dem Anzeigen** dieser View (nicht beim Initialisieren des View Controllers).

Daher: **Lazy Loading**:

- Programmatisches Erzeugen der UI nicht im Konstruktor des View Controllers, sondern in der Methode `loadView`

View Controller: Lazy Loading

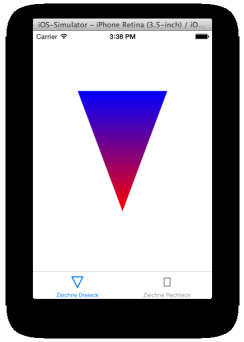
Modifizieren der UI über Callback-Methoden

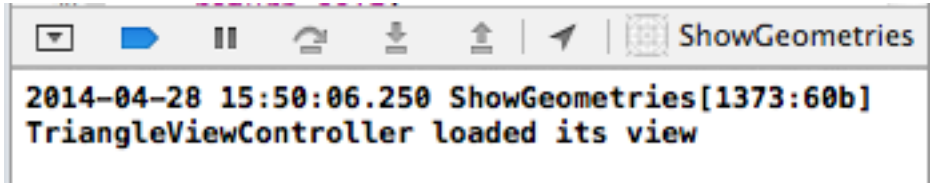
- z.B. wenn Inhalte der View sich seit ihrer letzten Darstellung verändert haben
- Beispiel: `viewDidLoad`

```
// TriangleViewController.m
[...]

@implementation TriangleViewController
[...]

// Wird immer nach dem Laden der View aufgerufen (egal ob Initialisierung
// programmatisch oder über XIB-Datei / Storyboard erfolgt)
-(void)viewDidLoad {
    [super viewDidLoad];
    // Modifizieren der UI
    NSLog(@"%@ loaded its view", [[self class] description]);
}

@end
```

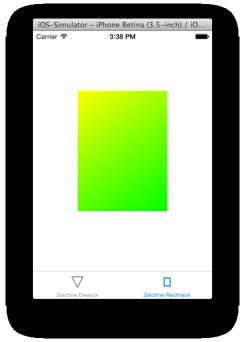
View Controller: Lazy Loading

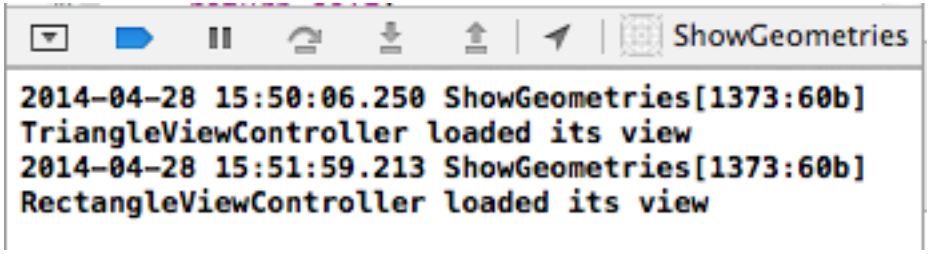
Starten der App



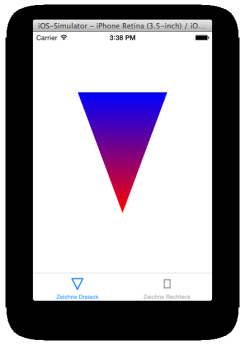


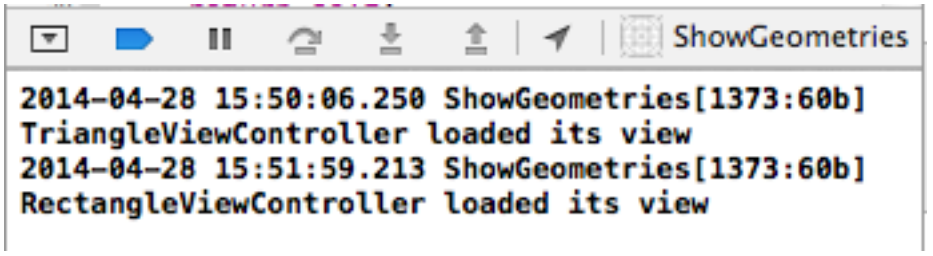
Wechsel des Tab





Wechsel des Tab (keine weitere Ausgabe)





View Controller: Lazy Loading

Hinweis:

- Der Zugriff auf die `view`-Property eines View Controllers innerhalb der Methode `initWithNibName:bundle:` sollte vermieden werden!
- Ein entsprechender Zugriff würd dazu führen, dass die View früher geladen wird als nötig!

View Controller: Lifecycle

Views werden automatisch zerstört, wenn

- der Speicher knapp wird (Low Memory Warning)
- sie gerade nicht benötigt wird (angezeigt wird)

→ Views müssen u.U. neu erzeugt werden

Daher:

- `loadView:`
  - Code für das Erzeugen einer View (**nur wenn View programmatisch erzeugt wird!**)
- `viewDidLoad:`
  - Code zum Modifizieren der View (z.B. Anpassen an Veränderungen des dargestellten Modells)

View Controller: Lifecycle

(Weitere) Callbacks von `UIViewController` zur differenzierten Modifikation der View:

- `viewDidLoad`:
  - Wird unmittelbar nach dem Laden der Xib-Datei aufgerufen
  - Alle View Objekte sind an dieser Stelle initialisiert
  - Hier sollten Konfigurationen getätigt werden, die **genau ein Mal notwendig** sind
- `viewWillAppear`:
  - Wird aufgerufen bevor die View dem Window hinzugefügt wird (also auf dem Screen angezeigt wird)
  - Hier sollten Konfigurationen getätigt werden, die **jedes Mal vor dem Anzeigen** der View erneut ausgeführt werden müssen
- `viewDidAppear`:
  - Wird aufgerufen nachdem die View dem Window hinzugefügt wurde (also wenn sie bereits auf dem Screen angezeigt wird)

View Controller: Lifecycle

(Weitere) Callbacks von `UIViewController` zur differenzierten Modifikation der View:

- `viewWillDisappear`:
  - Wird aufgerufen, bevor die View vom Window entkoppelt wird (also unmittelbar bevor sie vom Screen verschwindet)
- `viewDidDisappear`:
  - Wird aufgerufen, nachdem die View vom Window entkoppelt wurde (also unmittelbar nachdem sie vom Screen verschwunden ist)
- `didReceiveMemoryWarning`:
  - Wird vom Betriebssystem aufgerufen, wenn Speicher benötigt

View Controller: Lifecycle

Je nach Bedarf eignet sich einer der Callbacks, um Modifikationen an der UI vorzunehmen bzw. Zustände zu speichern.

`viewWillAppear`

- eignet sich z.B, um die Orientierung/Farbe einer Statusbar zu ändern

`viewWillDisappear`

- eignet sich z.B. um bestimmte Zustände der View zu speichern

`didReceiveMemoryWarning`

- Um den Speicher unkritischer Objekte freizugeben, die vom View Controller referenziert werden
- Ab iOS 6 kann man diese Methode auch verwenden, um explizit Referenzen zu View-Objekten freizugeben

UINavigationController

- Root View Controller der gesamten Anwendung
- Bekommt einen View Controller als Root View Controller beim Initialisieren übergeben
- Root View Controller enthält die erste Anzeige der App

UINavigationController besteht aus zwei Subviews

- UINavigationBar und
- UIView des aktuellen View Controllers

UINavigationController

- Verwaltet alle View Controller auf einem Stack
- Verwendet automatisch iOS Standard-Designelemente und -animationen

`UINavigationController`

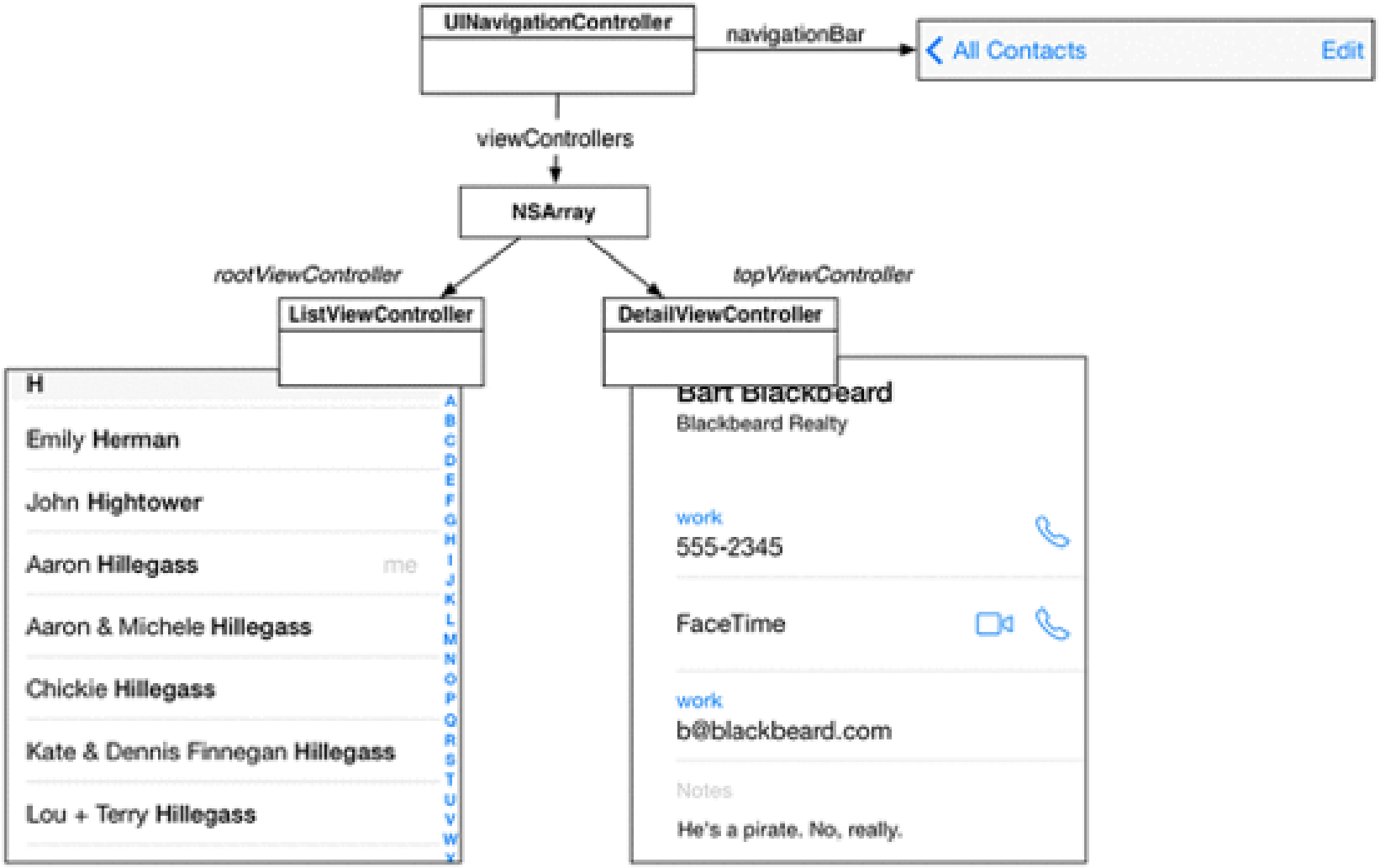
- Root View Controller ist immer das unterste Element auf dem Stack
- Weitere View Controller können auf dem Stack abgelegt (push) / vom Stack entfernt (pop) werden

Ablegen eines View Controllers auf dem Stack führt zur Anzeige seiner View

Stack wird als `NSArray` (`self.viewControllers`) verwaltet (erstes Element ist der Root View Controller)

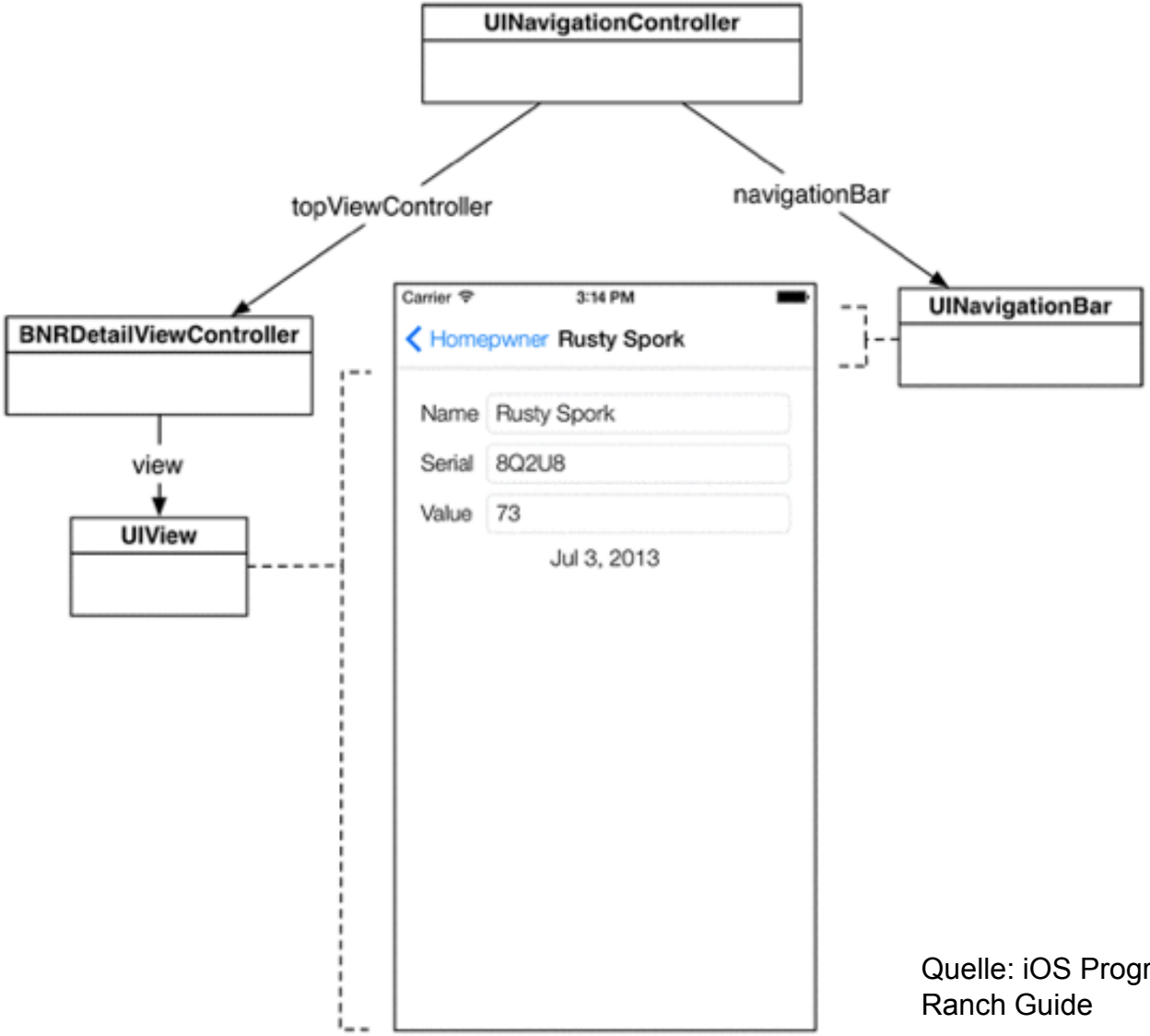
`self.topViewController` referenziert das oberste Element auf dem Stack

UINavigationController: Stack der Controller



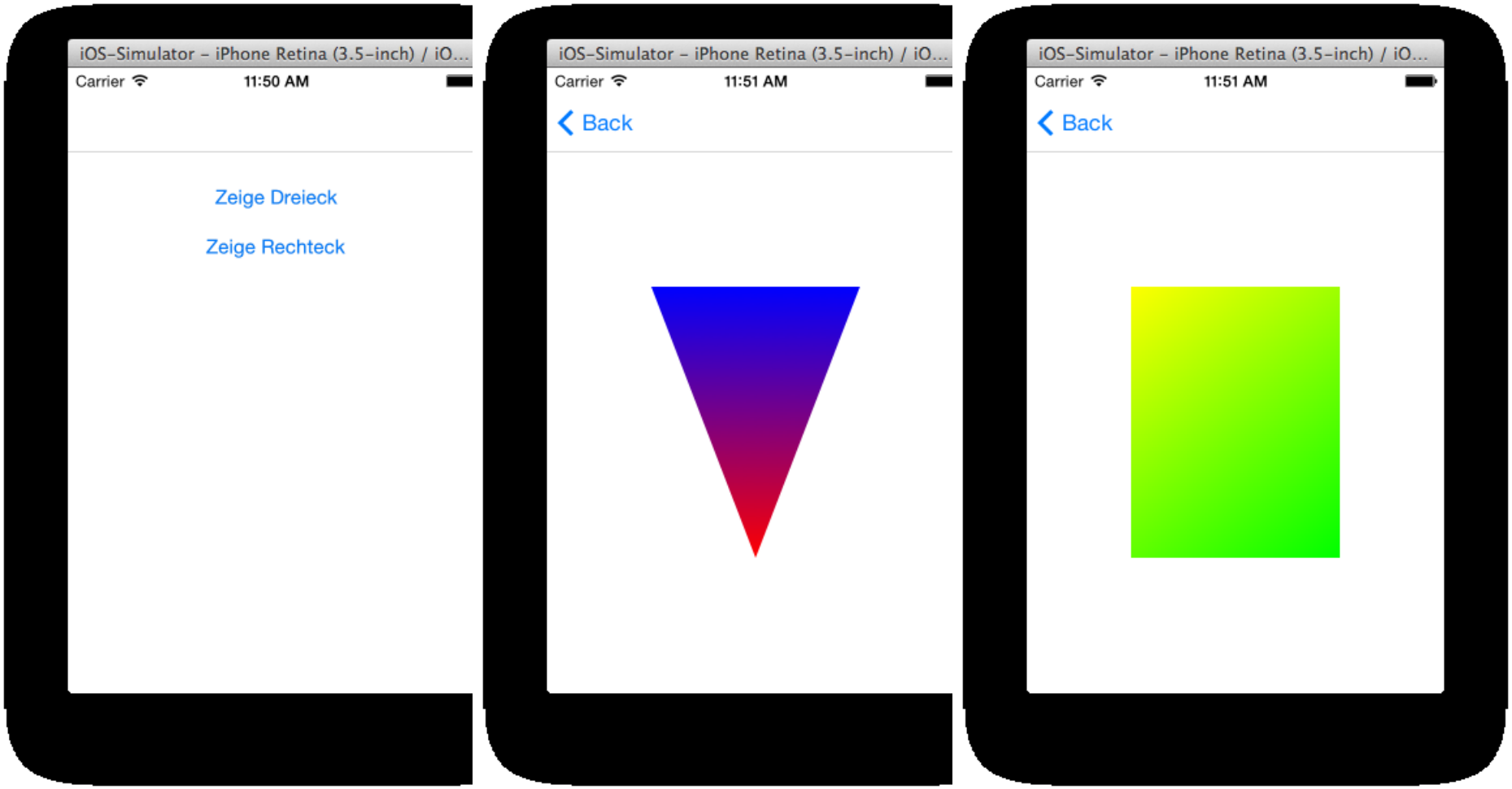
Quelle: iOS Programming, The Big Nerd Ranch Guide

UINavigationController: Assoziation der Views

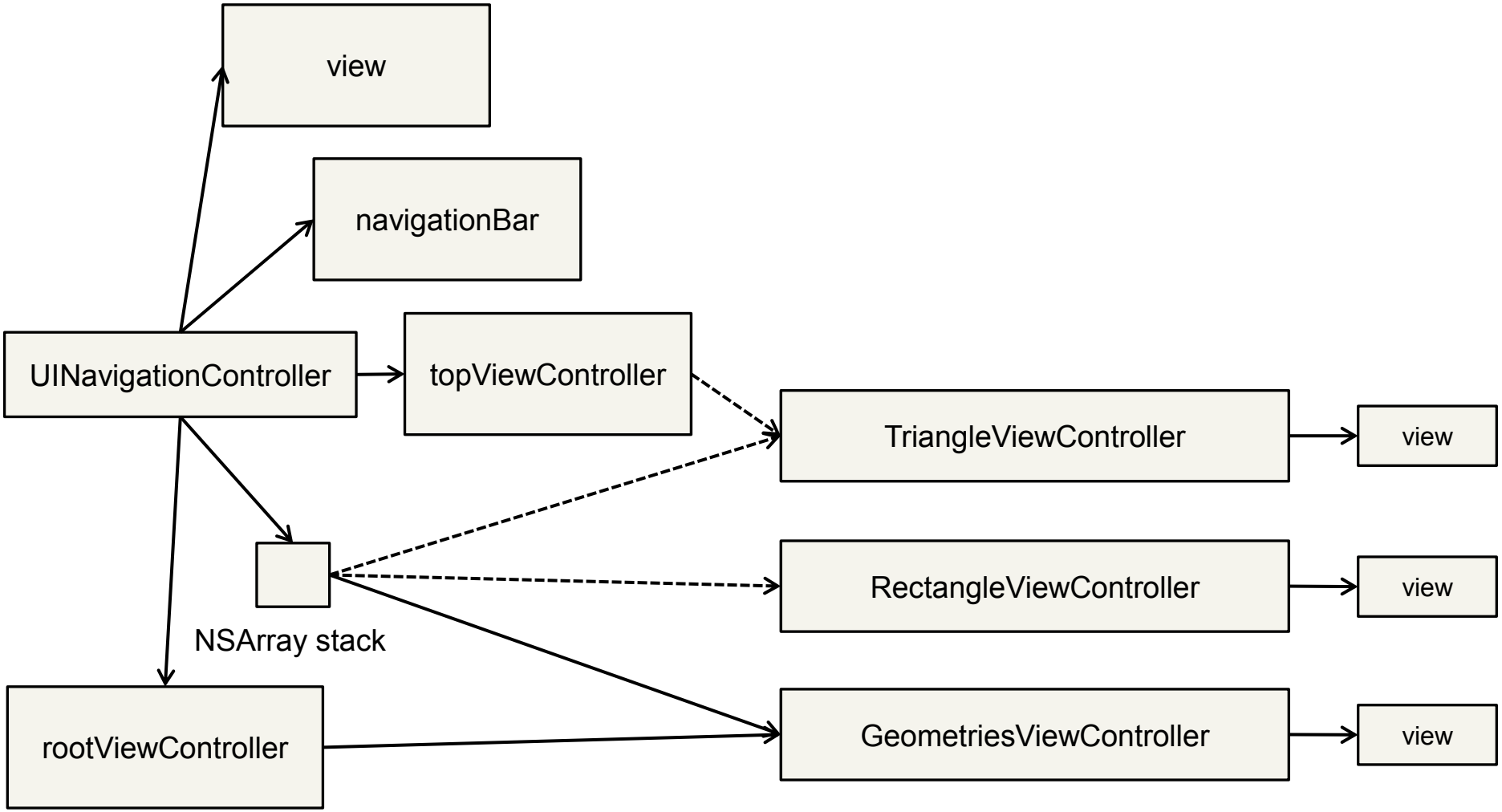


Quelle: iOS Programming, The Big Nerd Ranch Guide

Beispiel: ShowGeometries

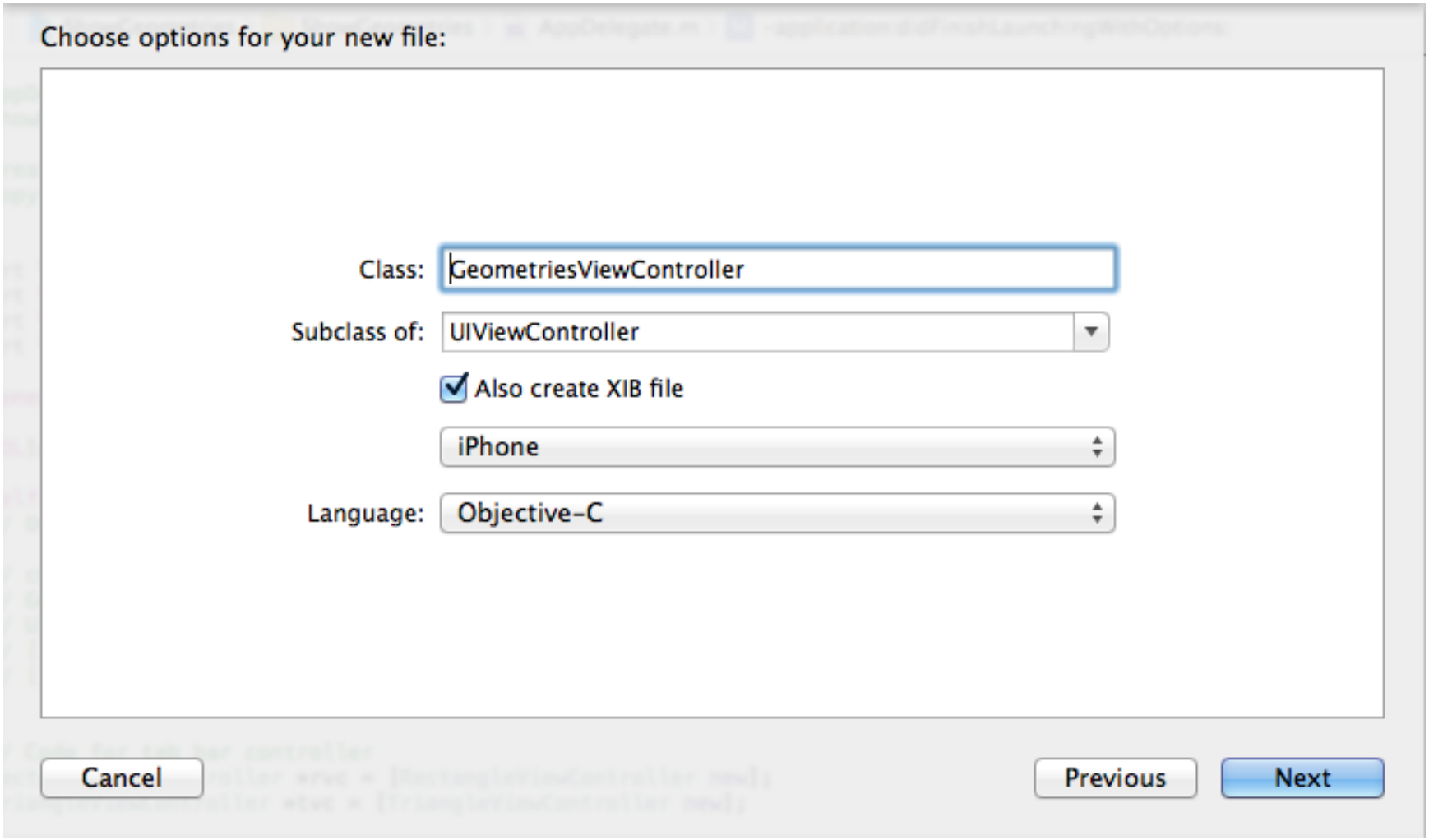


ShowGeometries: Struktur

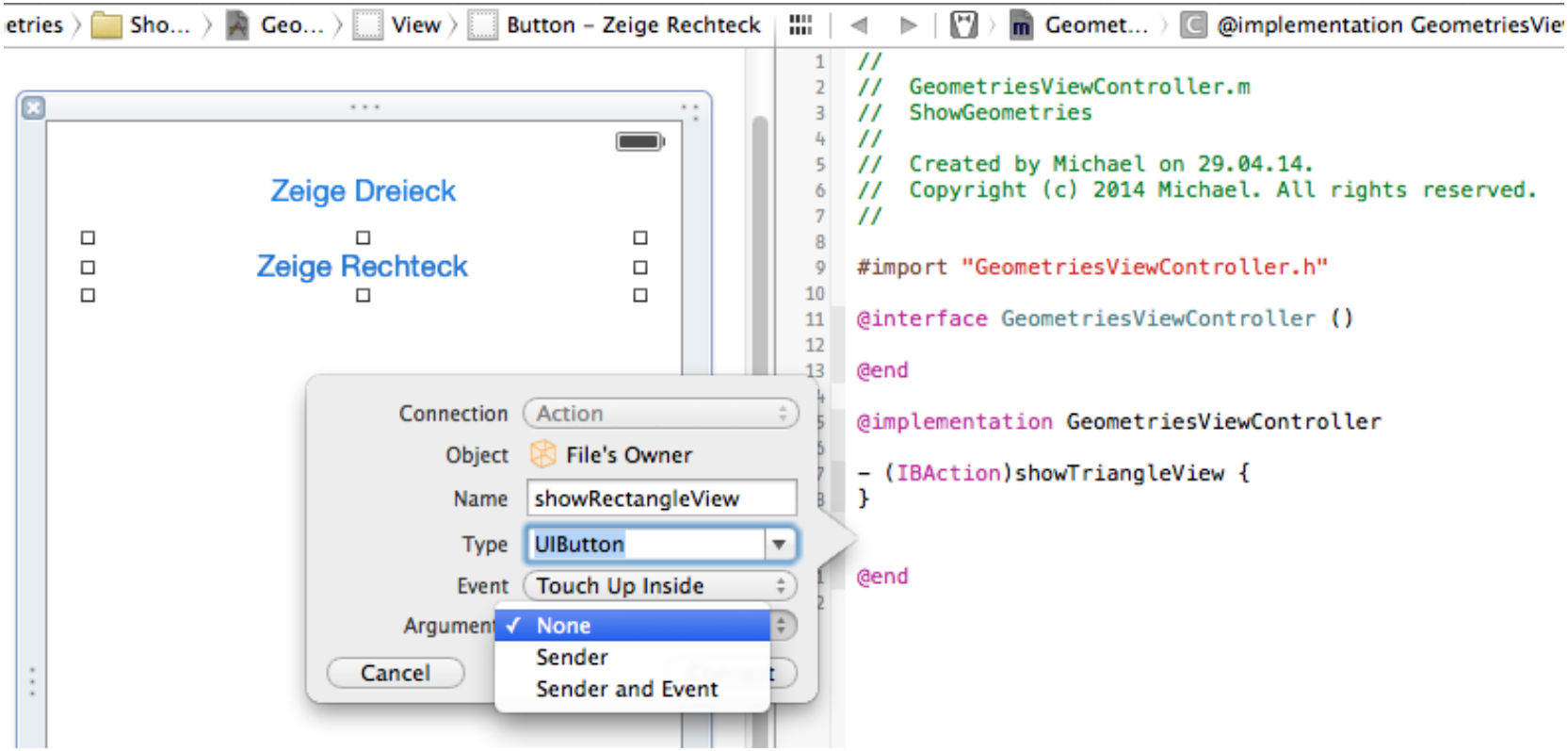


ShowGeometries: ViewController

File → New → File → iOS: Source → Cocoa Touch



ShowGeometries: UI und Action Targets



ShowGeometries: Root View Controller


```
// GeometriesViewController.m

#import "GeometriesViewController.h"
#import "RectangleViewController.h"
#import "TriangleViewController.h“

@interface  GeometriesViewController ()
@property (strong, nonatomic) TriangleViewController *tvc;
@property (strong, nonatomic) RectangleViewController *rvc;
@end

@implementation GeometriesViewController
-(TriangleViewController *)tvc {
    if(!_tvc) _tvc = [TriangleViewController new]; return _tvc;
}
-(RectangleViewController *)rvc {
    if(!_rvc) _rvc = [RectangleViewController new]; return _rvc;
}
-(IBAction)showTriangleView {
    [[self navigationController] pushViewController:self.tvc animated:true];
}
-(IBAction)showRectangleView {
    [[self navigationController] pushViewController:self.rvc animated:true];
}
-(void)viewDidLoad {
    [super viewDidLoad];
    self.navigationController.navigationBar.translucent = NO;
}
@end
```

ShowGeometries: Anpassung des AppDelegate


```
// GeometriesAppDelegate.m

#import "GeometriesAppDelegate.h"
#import "GeometriesViewController.h"

@implementation GeometriesAppDelegate

-(BOOL)application:(UIApplication *)application
        didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {
    self.window =
        [[UIWindow alloc] initWithFrame:[[UIScreen mainScreen] bounds]];
    GeometriesViewController *gvc = [[GeometriesViewController alloc] init];
    UINavigationController *navigationController =
        [[UINavigationController alloc] initWithRootViewController:gvc];
    [[self window] setRootViewController:navigationController];
    self.window.backgroundColor = [UIColor whiteColor];
    [self.window makeKeyAndVisible];
    return YES;
}
@end
```

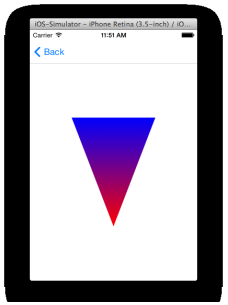
View Controller Memory Management

Beispiel für die Verwendung von `didReceiveMemoryWarning`:

```
// GeometriesViewController.m
[...]

@implementation GeometriesViewController
[...]

-(void)didReceiveMemoryWarning {
    [super didReceiveMemoryWarning];
    NSLog(@"%@ received memory warning", [[self class] description]);
    if(self.tvc != self.navigationController.topViewController) {
        self.tvc = nil;
        NSLog(@"%@ relinquished", [[self.tvc class] description]);
    }
    if(self.rvc != self.navigationController.topViewController) {
        self.rvc = nil;
        NSLog(@"%@ relinquished", [[self.rvc class] description]);
    }
}
@end
```

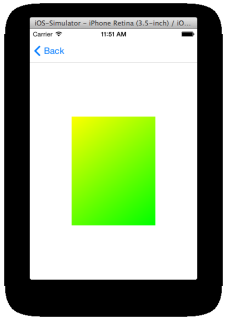
Ausführen der App

Zeige Dreieck



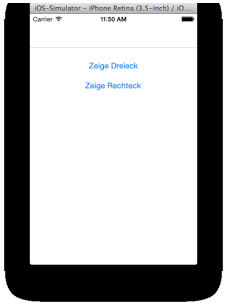
```
2014-04-29 11:54:25.789 ShowGeometries[1925:60b]
TriangleViewController loaded its view
```

Back → Zeige Rechteck



```
2014-04-29 11:54:25.789 ShowGeometries[1925:60b]
TriangleViewController loaded its view
2014-04-29 11:55:42.183 ShowGeometries[1925:60b]
RectangleViewController loaded its view
```

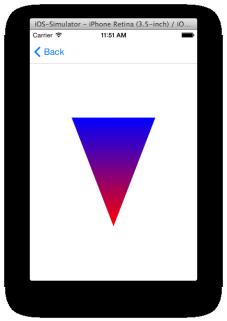
Back (Hardware→ Simulate Memory Warning)

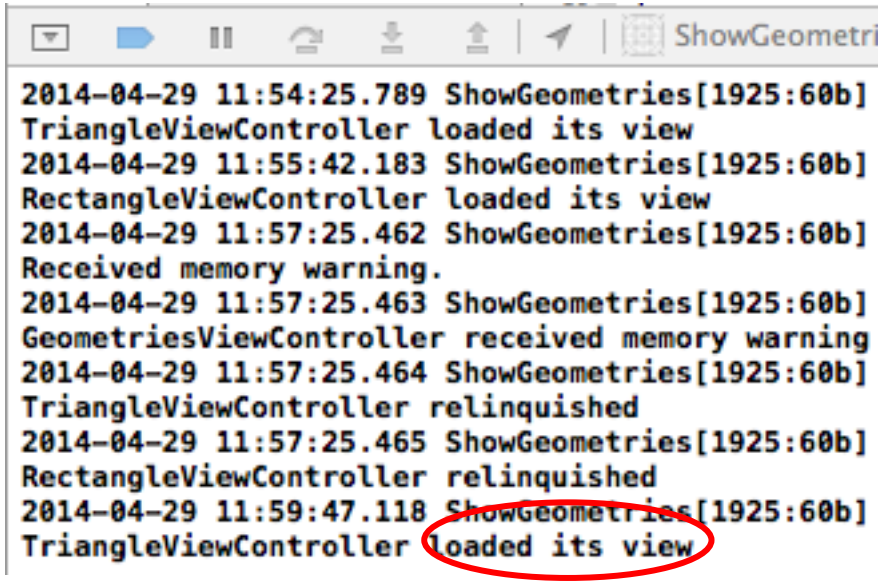



```
2014-04-29 11:54:25.789 ShowGeometries[1925:60b]
TriangleViewController loaded its view
2014-04-29 11:55:42.183 ShowGeometries[1925:60b]
RectangleViewController loaded its view
2014-04-29 11:57:25.462 ShowGeometries[1925:60b]
Received memory warning.
2014-04-29 11:57:25.463 ShowGeometries[1925:60b]
GeometriesViewController received memory warning
2014-04-29 11:57:25.464 ShowGeometries[1925:60b]
TriangleViewController relinquished
2014-04-29 11:57:25.465 ShowGeometries[1925:60b]
RectangleViewController relinquished
```

Ausführen der App

Zeige Dreieck





[...]

Schachtelung von View Controllern

Schachteln von View Controllern ist möglich

Von Kind- zu Eltern-Controller:

- Content View Controller (z.B. Unterklasse von `UIViewController`) kann Kind sein von…
- Navigation View Controller kann Kind sein von…
- Tab Bar View Controller …
- Split View Controller

Zum Weiterlesen:

- https://developer.apple.com/library/ios/documentation/WindowsViews/Conceptual/ViewControllerCatalog/Chapters/CombiningViewControllers.html

STORYBOARDS

Storyboard: Was ist das?

'Storyboarding' ist Teil eines SE-Prozesses und vor allem während der Konzeptionsphase sehr wichtig.

UIs werden entworfen und in eine Reihenfolge gebracht, um User Experience schnell und ohne Programmieraufwand nachvollziehbar zu machen.

Entwürfe werden mit Stift und Papier gezeichnet! (siehe auch: Rapid Prototyping)

Storyboards in iOS

Ein Storyboard ist die visuelle Darstellung der UI einer iOS-Anwendung

Ein Storyboard zeigt Sequenzen verschiedenen Anzeigen (Screens) einer Anwendung und deren Verbindungen (**Segues**) untereinander

Jeder Screen besteht aus einem View Controller und seinen Views

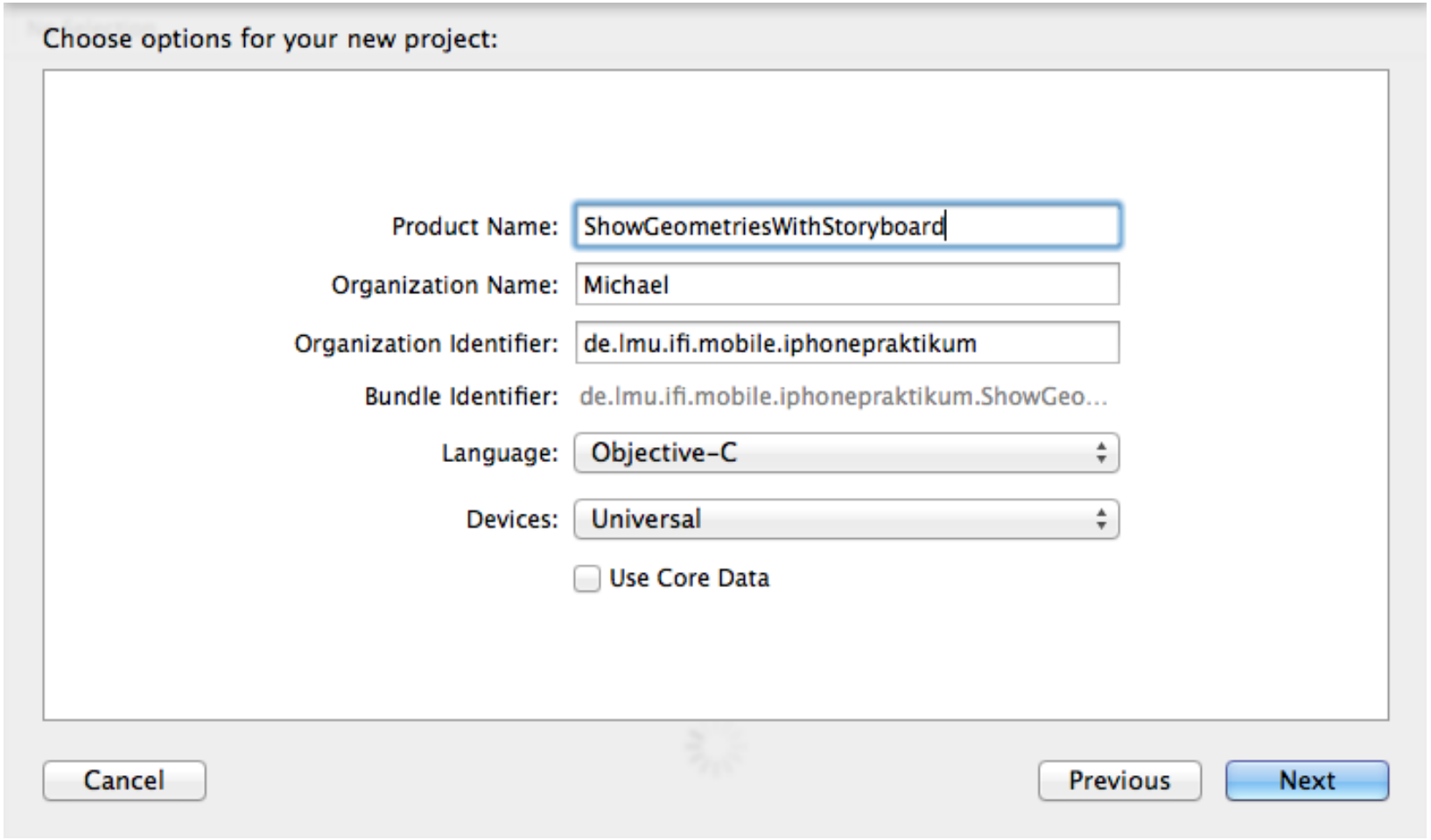
Storyboard wird in XIB-ähnlicher Datei gespeichert und beim Laden der Anwendung in Code-Gerüst übersetzt

Vorgehensweise:

- Anlegen einzelner Screens
- Modellieren der Beziehungen zwischen Screens

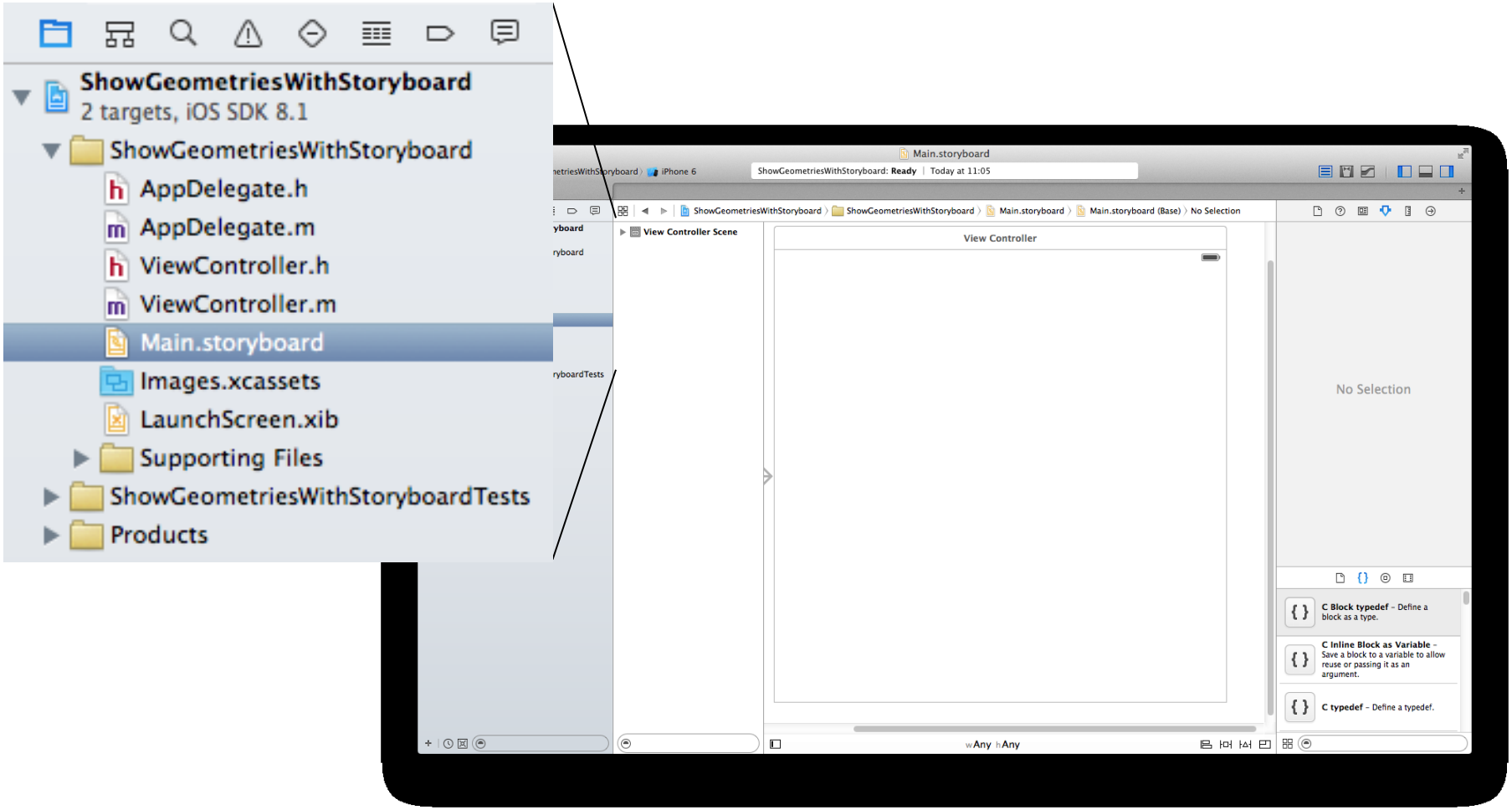
DEMO: ShowGeometriesWithStoryboard

File → New → Project → iOS: Application → Single View Application



DEMO: ShowGeometriesWithStoryboard

Erzeugt automatisch ein Storyboard mit einem View Controller

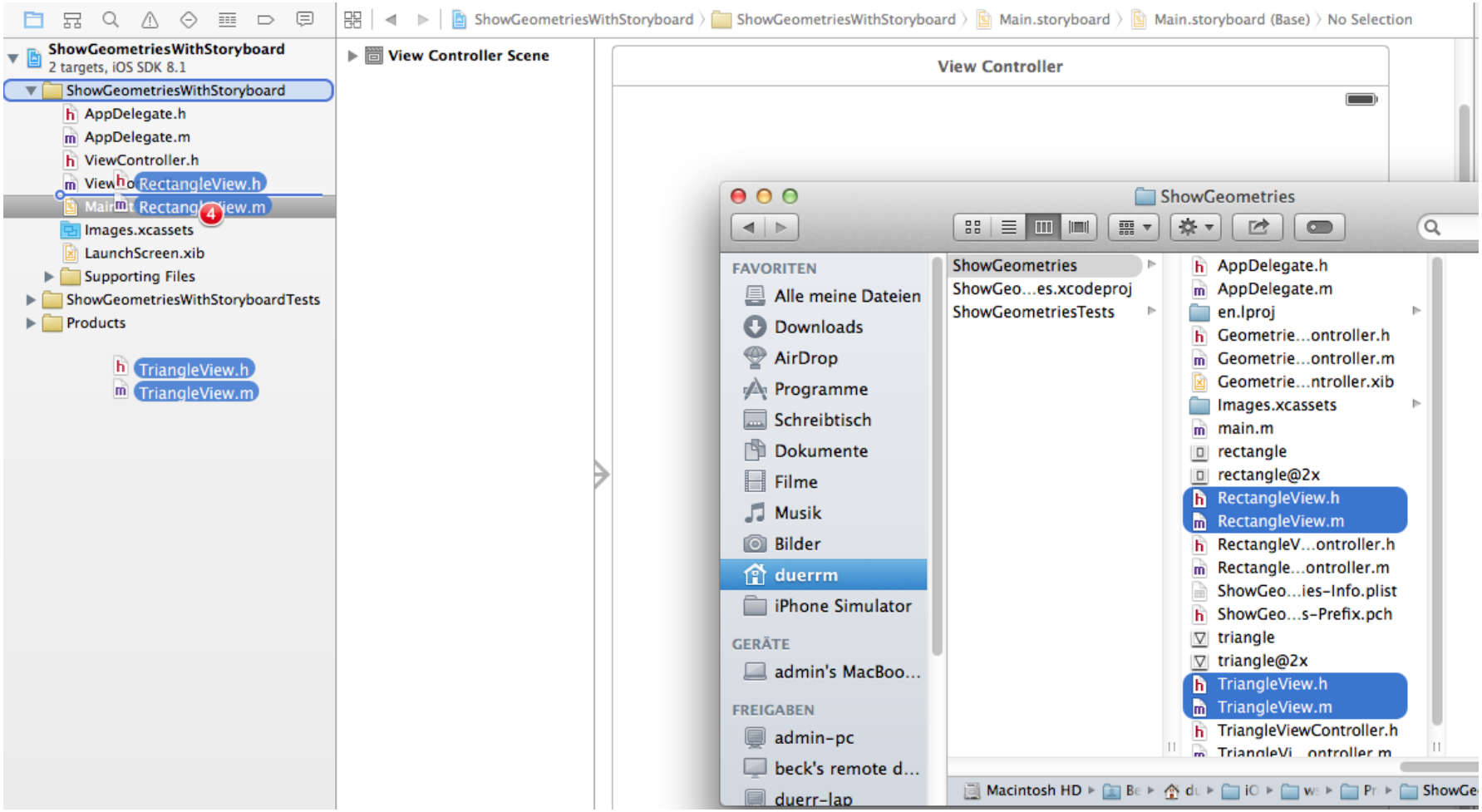


DEMO: ShowGeometriesWithStoryboard

Importieren bereits existierender Dateien für die zu visualisierenden Views:

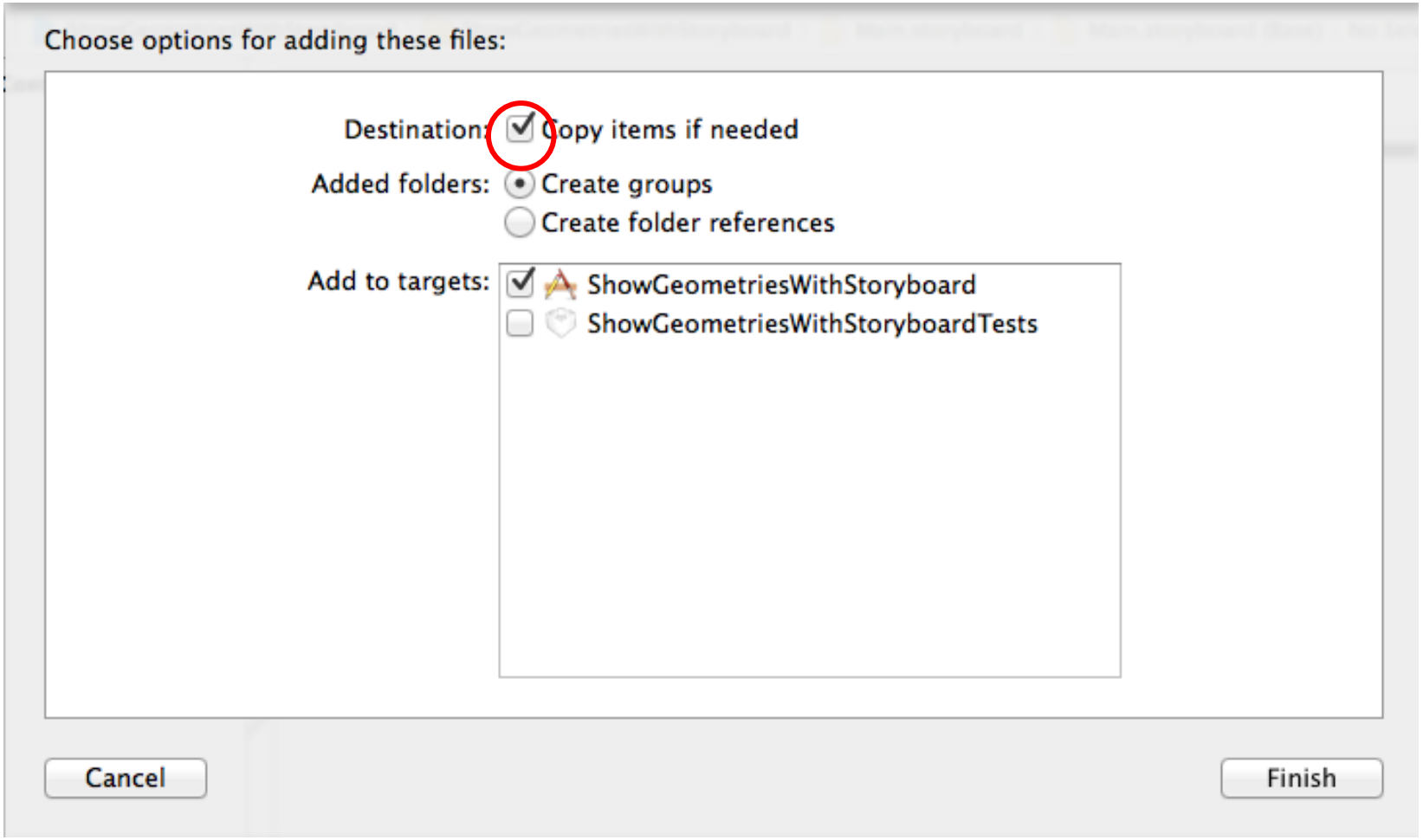
- `ShowTriangle.h`
- `ShowTriangle.m`
- `ShowRectangle.h`
- `ShowRectangle.m`

DEMO: ShowGeometriesWithStoryboard



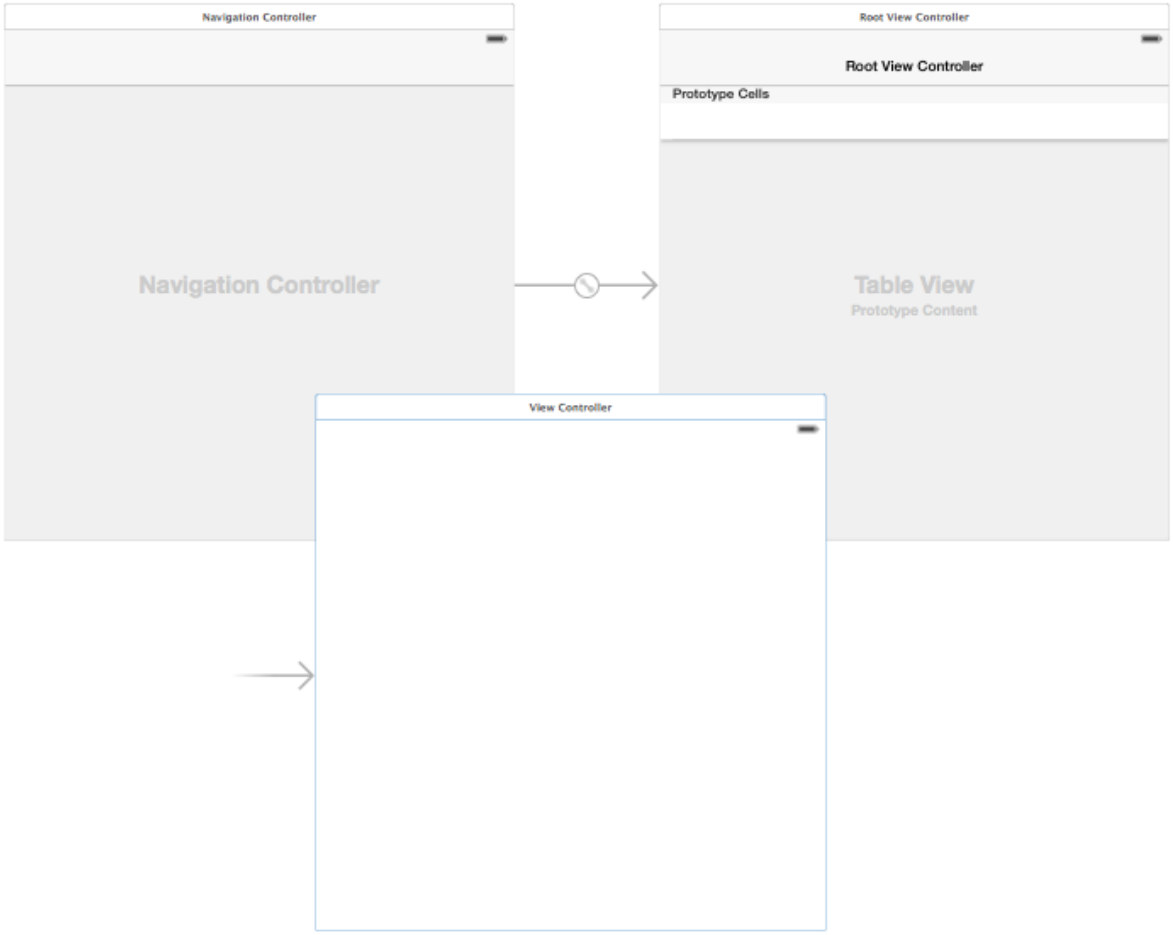
DEMO: ShowGeometriesWithStoryboard

Dateien Kopieren!



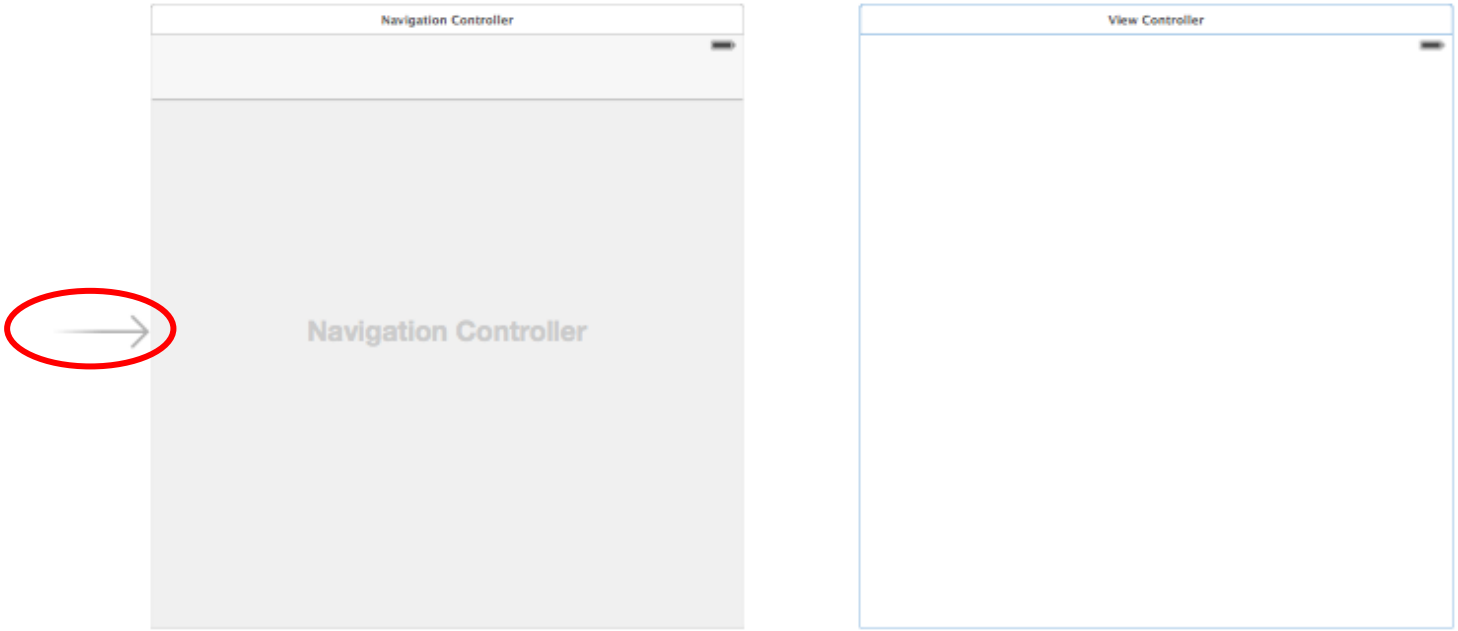
DEMO: ShowGeometriesWithStoryboard

Hinzufügen eines `UINavigationControllers`



DEMO: ShowGeometriesWithStoryboard

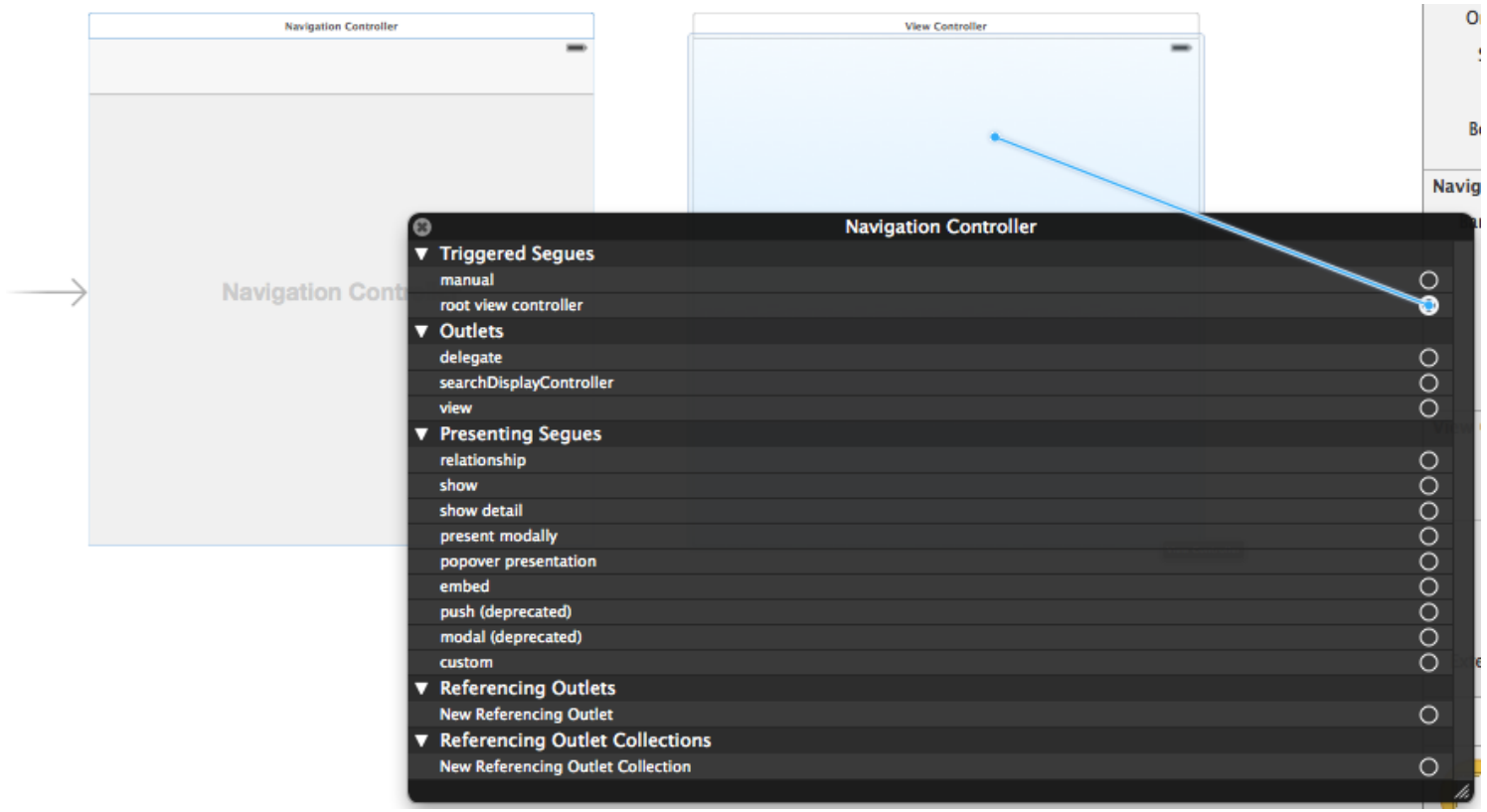
Entfernen des Table View Controllers + Verschieben des Application Entry Points



DEMO: ShowGeometriesWithStoryboard

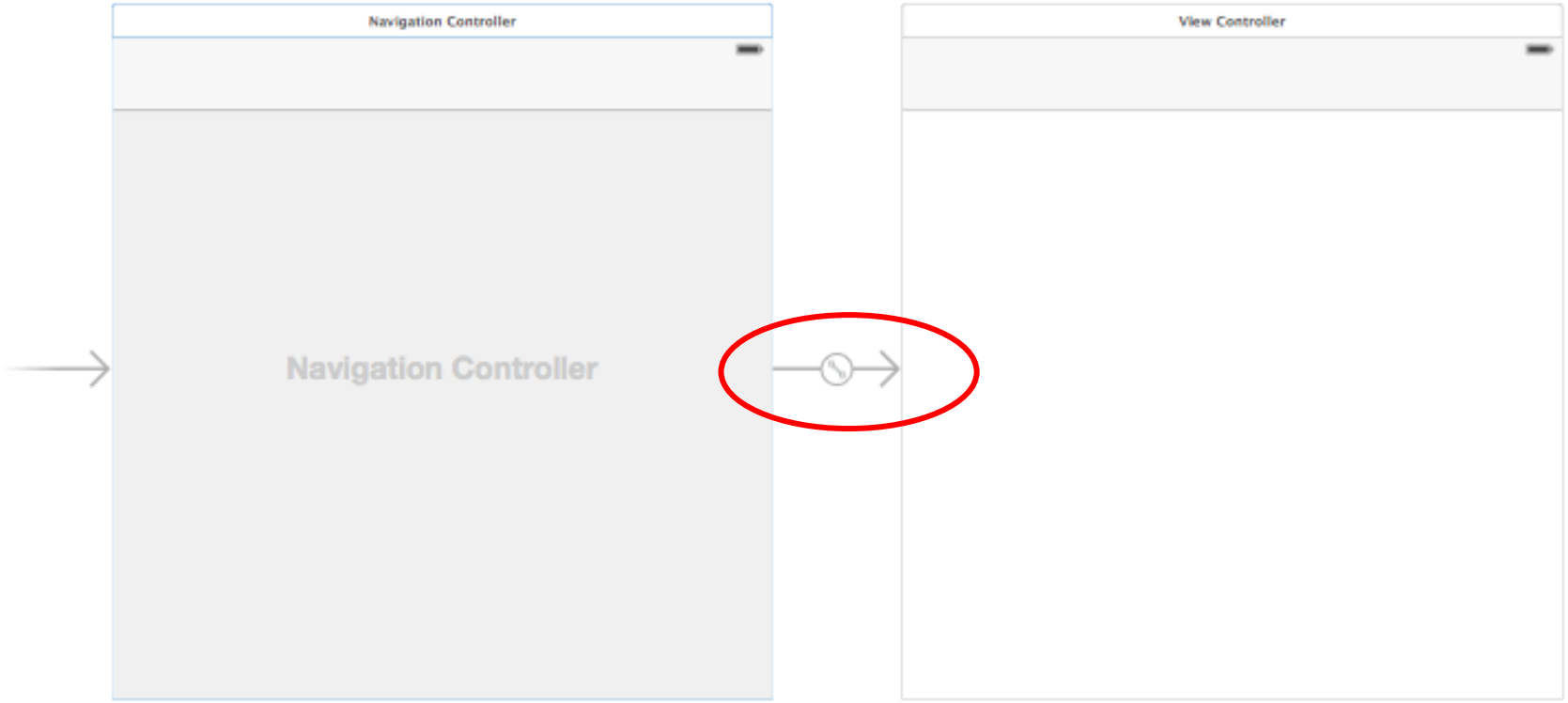
Setzen des `GeometriesViewControllers` als Root View Controller im `UINavigationController`

- **Segue** vom `UINavigationController` zum `GeometriesViewController` (zu Segues später mehr)



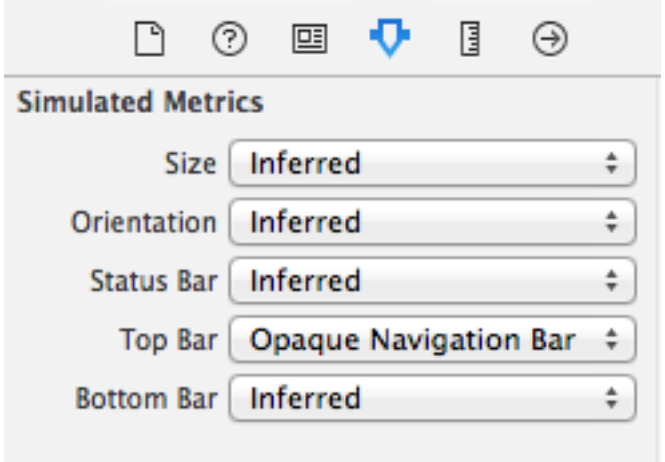
DEMO: ShowGeometriesWithStoryboard

Ergebnis



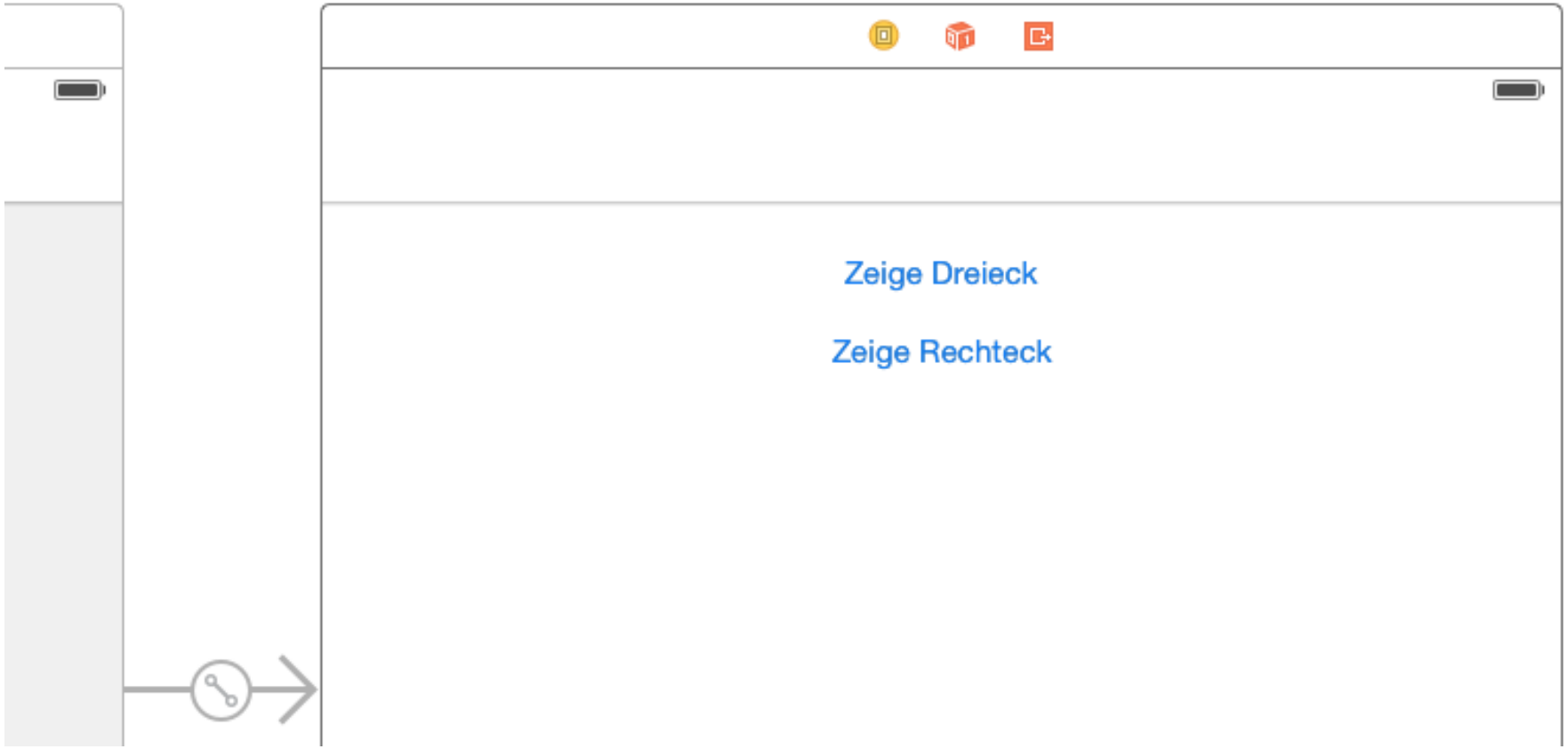
DEMO: ShowGeometriesWithStoryboard

Setzen der Top Bar vom `UINavigationController` als Opaque



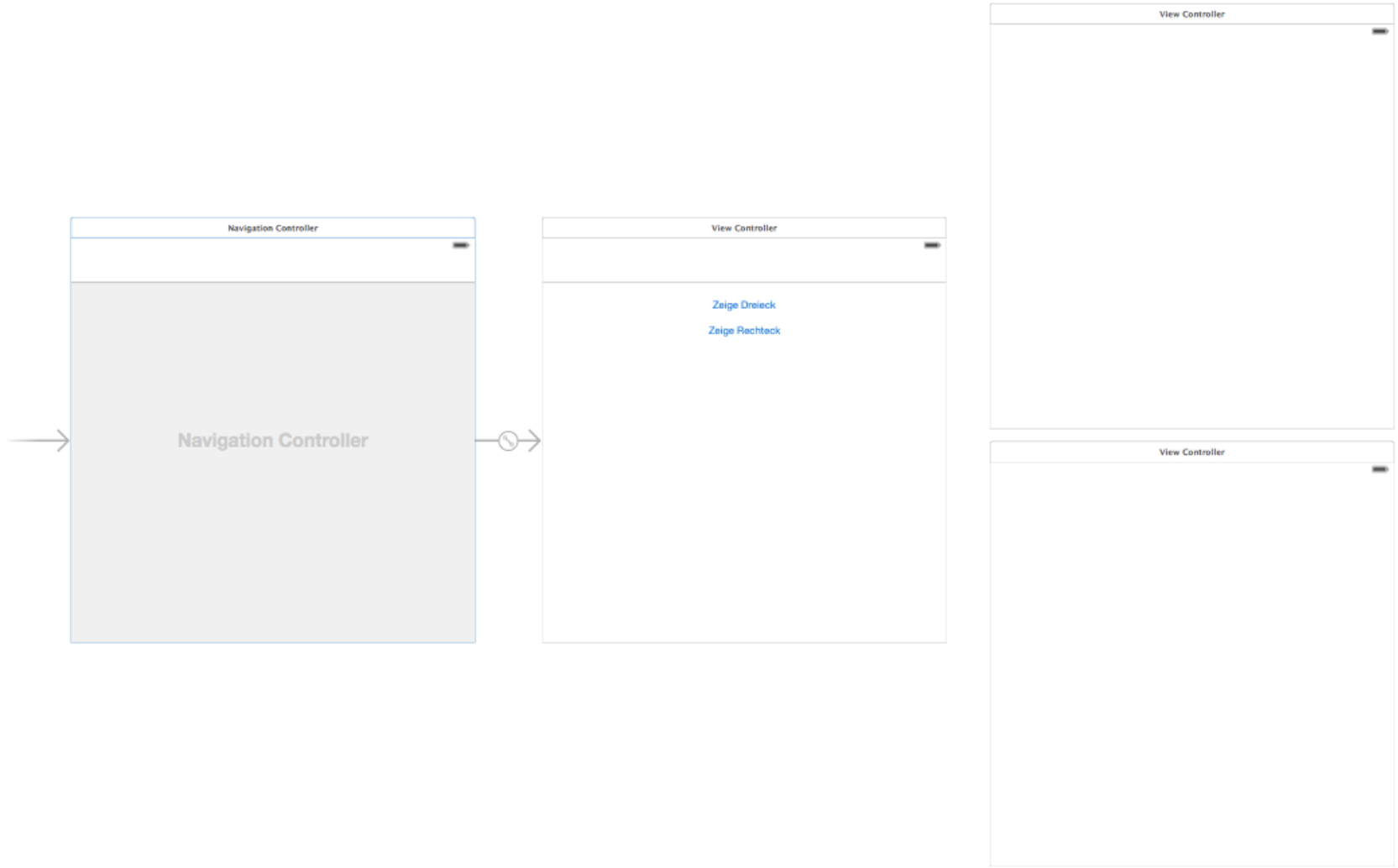
DEMO: ShowGeometriesWithStoryboard

Hinzufügen der UI-Elemente zum `GeometriesViewController`

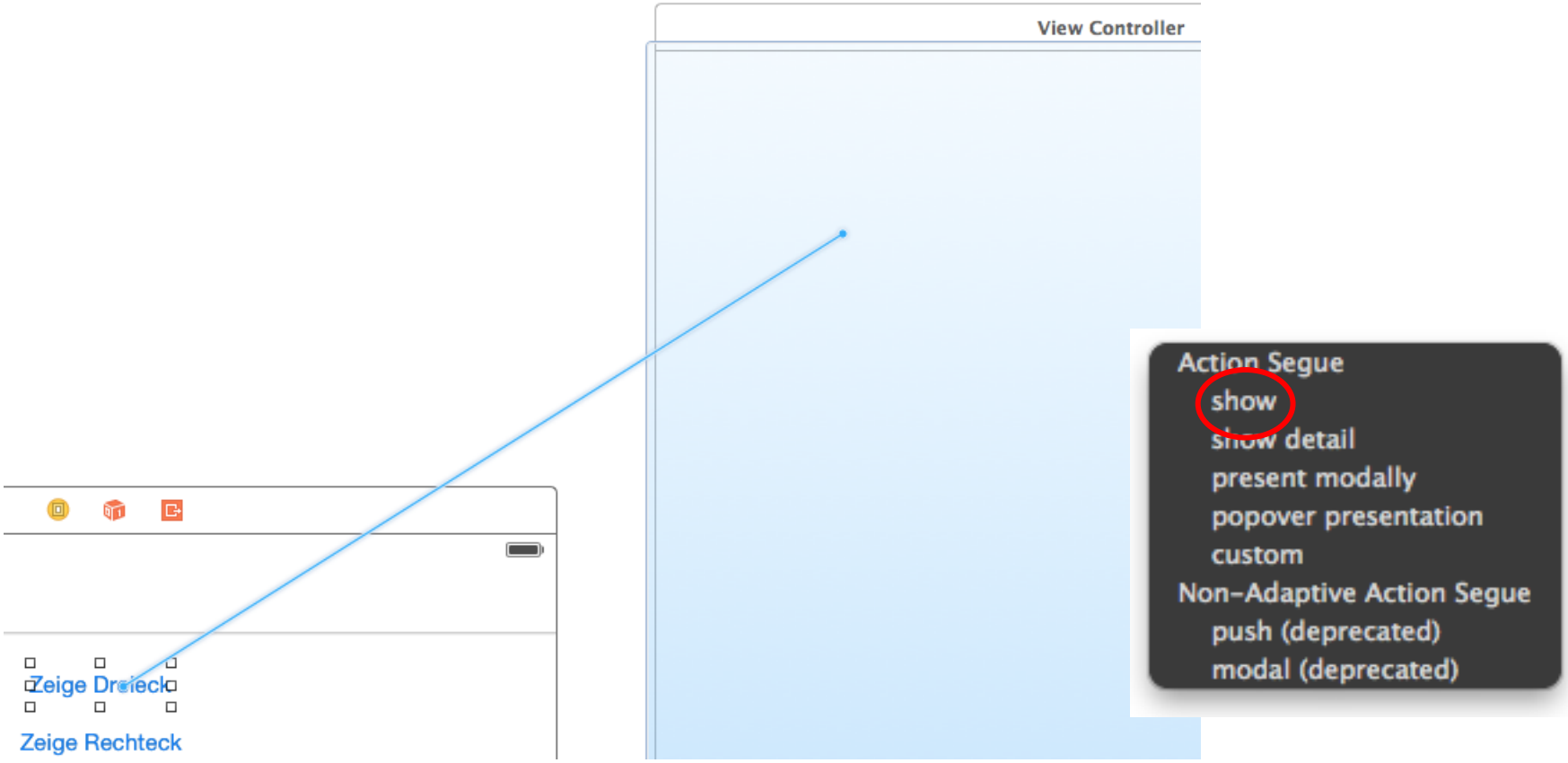


DEMO: ShowGeometriesWithStoryboard

Hinzufügen zweier View Controller für Rectangle und Triangle View

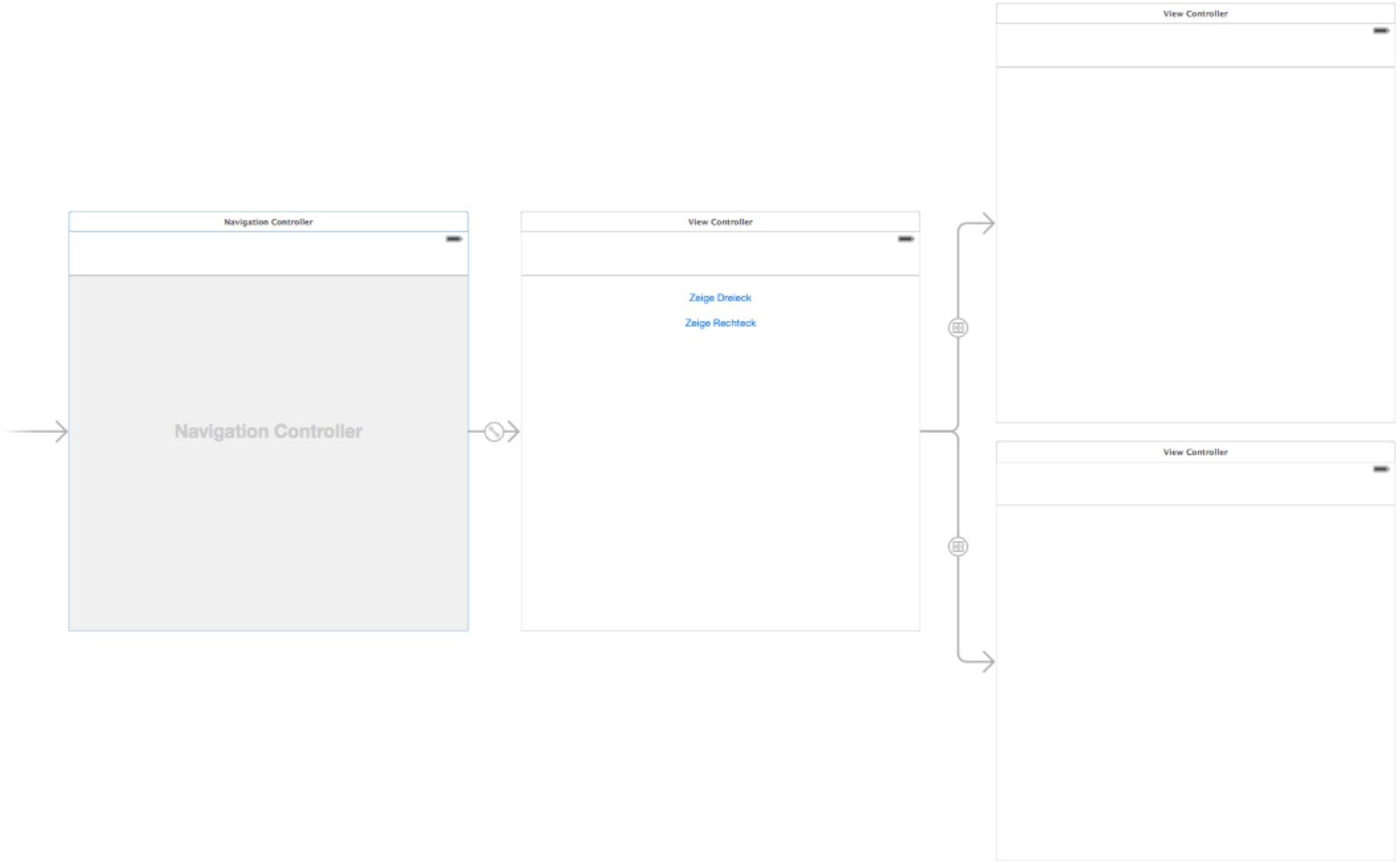


Erzeugen der Segues für Actions (Auswahl: show)



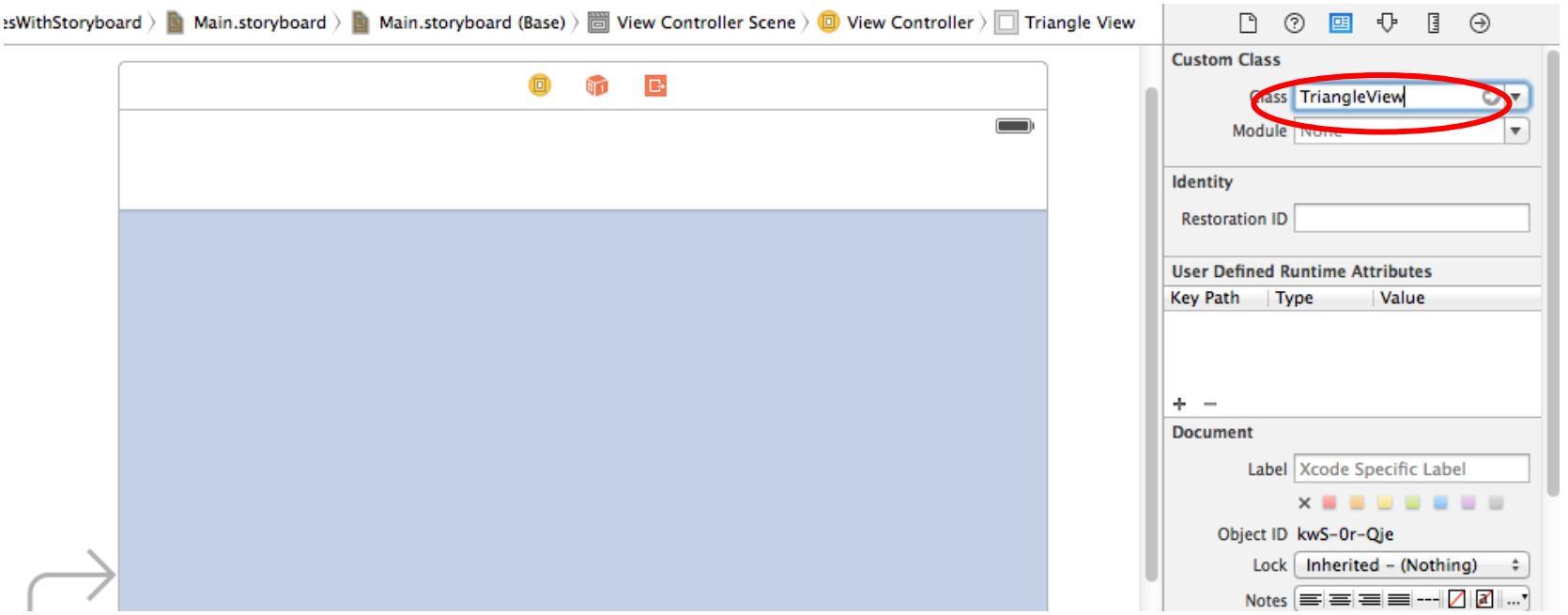
DEMO: ShowGeometriesWithStoryboard

Ergebnis:



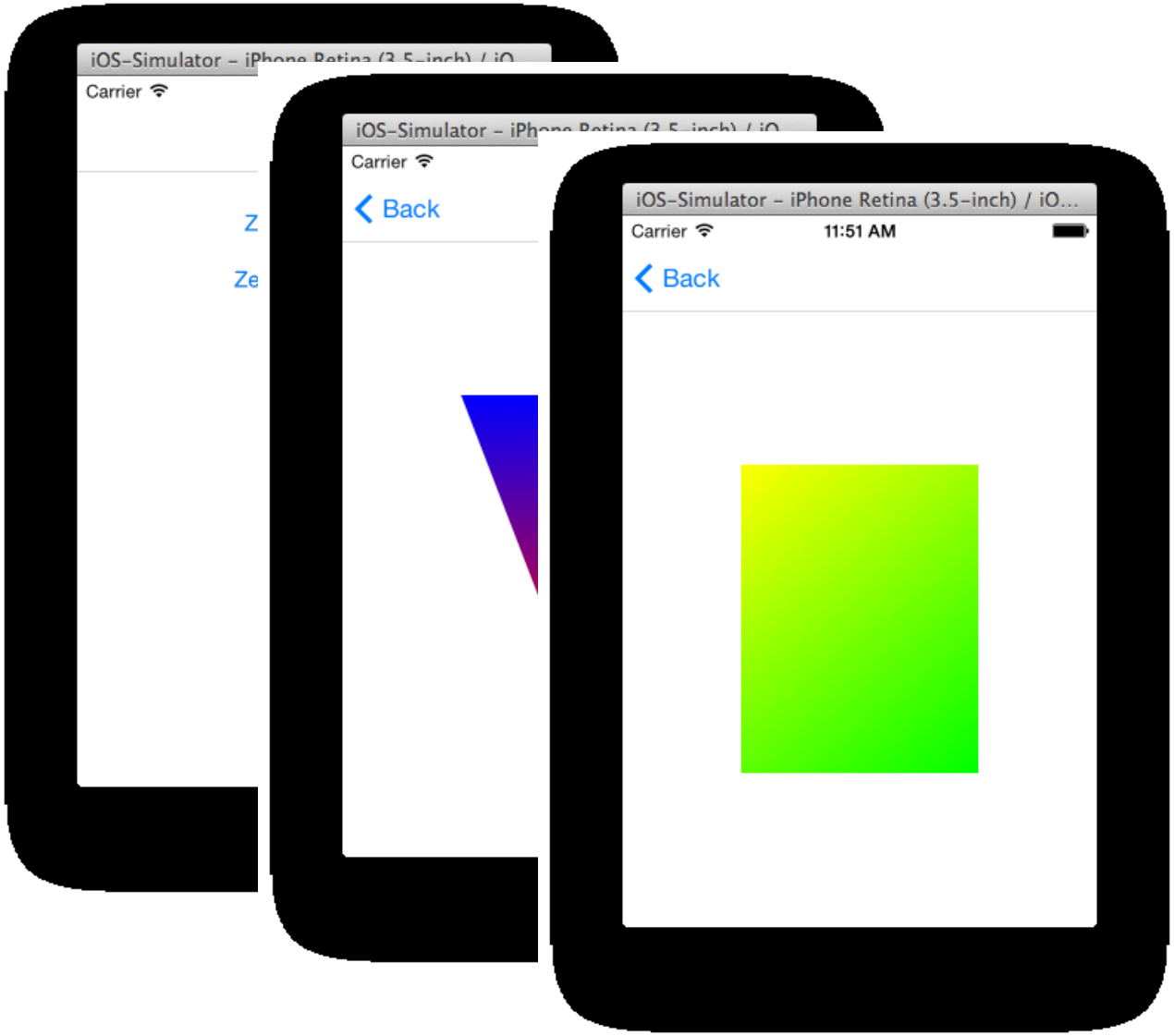
DEMO: ShowGeometriesWithStoryboard

Setzen der Views der beiden Views



DEMO: ShowGeometriesWithStoryboard

Voilà!



DEMO: ShowGeometriesWithStoryboard

Storyboards führen zu sehr wenig Code

```
// GeometriesAppDelegate.m
#import "GeometriesAppDelegate.h"

@implementation GeometriesAppDelegate

-(BOOL)application:(UIApplication *)application
    didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {
    return YES; // Initialisierung: Das ist alles!
}
@end
```

```
// GeometriesViewController.m
#import "GeometriesViewController.h"

@implementation GeometriesViewController
// keine Implementierung notwendig!
@end
```

Storyboards: Weitere Initialisierung

Für alle UI-Objekte, die von einem Storyboard geladen werden, können finale Initialisierungsschritte im zugehörigen View Controller über die Methode `awakeFromNib` vorgenommen werden.

Storyboard: Segues

Segues (sprich /'sɛgweɪ/) modellieren den Wechsel zwischen Screens

Verbindung zwischen zwei UIs werden mit Ctrl+drag angelegt.

Segues haben einen Transitionstyp (Style, z.B. show, modally, custom, ...)

Man kann von Segues erben um eigene Transitionstypen zu modellieren

Segues werden von iOS automatisch erzeugt, wenn ein Übergang von einem Screen zum nächsten ausgelöst wird

Storyboard: Segues

Man kann mit dem Callback `prepareForSegue:sender:` einem Segue Daten übergeben

- wird automatisch im Source View Controller beim Auslösen des Segues aufgerufen
- dient zum Konfigurieren des Destination View Controllers vor Anzeige seiner View
- Kommt üblicherweise zum Einsatz, wenn das Auslösen durch ein UI-Event erfolgt

Programmatisches Auslösen eines Segues möglich:

- Aufruf der Methode `performSegueWithIdentifier:sender:` auf den entsprechenden View Controller

Storyboard: Segues

Jeder Segue wird von einer `UIStoryboardSegue` Instanz repräsentiert und definiert sich über die Kontextinformationen/Properties:

- `style`:
  - wie wird der folgende View Controller angezeigt (show, show detail, present modally, popover presentation, custom; kann man im Attributes Inspector festlegen)
- `sender`:
  - das View Objekt, das den Segue auslöst (z.B. `UIButton`, `UITableViewCell`, ...)
- `identifier`:
  - Eindeutiger Name für Zugriff auf den Segue (kann man im Attributes Inspector festlegen)
- `source-/destinationViewController:`
  - Die beteiligten View Controller Objekte

Erzeugen von Custom Segues durch Erben von `UIStoryboardSegue` möglich

Storyboard: Segues Lifecycle

Ablauf beim Wechsel von einem Screen zum nächsten mit Segues

- Erzeugen und Initialisieren des Destination View Controllers
- Erzeugen des Segue-Objects und Aufruf der Methode `initWithIdentifier:source:destination:` des Segues
  - `identifier:` Eindeutiger Name des Segues (Vergabe im Attributes Inspector)
  - `source/destination:` die beteiligten View Controller Objekte
- Aufruf der Methode `prepareForSegue:sender` des Source View Controllers
- Aufruf der Methode `perform` des Segue-Objekts (Ausführen der Transition, die den Destination View Controller auf dem Screen anzeigt)
- Verwerfen der Referenz (also Löschen) des Segue-Objekts

Storyboard: Erzeugen eines Custom Segues

Erben von der Klasse `UIStoryboardSegue` und Implementieren der Methoden

- `perform` Ausführen der visuellen Transition
- `initWithIdentifier:source:destination:` Initialisierung eigener Variablen der Custom Segue Unterklasse (optional)

Standard-Properties eines Segues können im Attributes Inspector angepasst werden

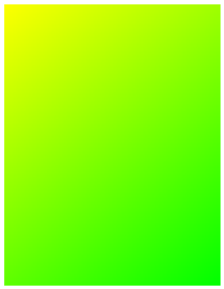
Eigene (custom) Properties müssen in der Methode `prepareForSegue:sender` des Source View Controllers angepasst werden

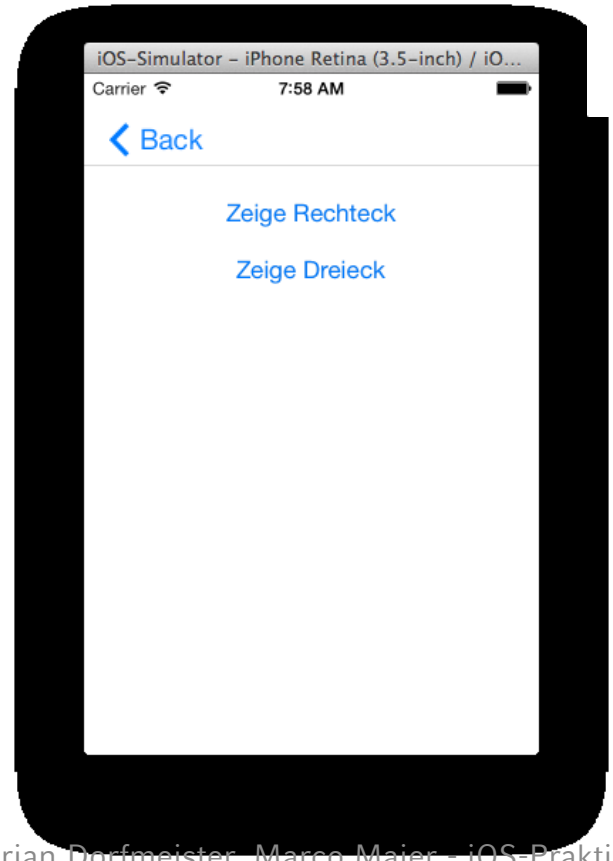
Zum Weiterlesen:

- https://developer.apple.com/library/ios/featuredarticles/ViewControllerPGforiPhoneOS/CreatingCustomSegues/CreatingCustomSegues.html

Beispiel: Custom Segue

Neue View soll von rechts oben nach links unten ,





Beispiel: Custom Segue

Hinzufügen der Klasse `MyCustomSegue` zum Projekt ShowGeometriesWithStoryboard

```
// MyCustomSegue.m
#import "MyCustomSegue.h"

@implementation MyCustomSegue
-(void)perform{
    // dereferenzieren der beteiligten View Controller
    UIViewController *dstCtrl = self.destinationViewController;
    UIViewController *srcCtrl = self.sourceViewController;

    // Unterdrücken der Anzeige der neuen View
    [dstCtrl viewWillAppear:NO];
    [dstCtrl viewDidAppear:NO];
    // Definieren der eigenen Animation
    [srcCtrl.view addSubview:dstCtrl.view];
    CGRect original = dstCtrl.view.frame;
    dstCtrl.view.frame = CGRectMake(
        original.origin.x+original.size.width,
        0-original.size.height,
        original.size.width,
        original.size.height
    );

    // Fortsetzung: nächster Slide
```

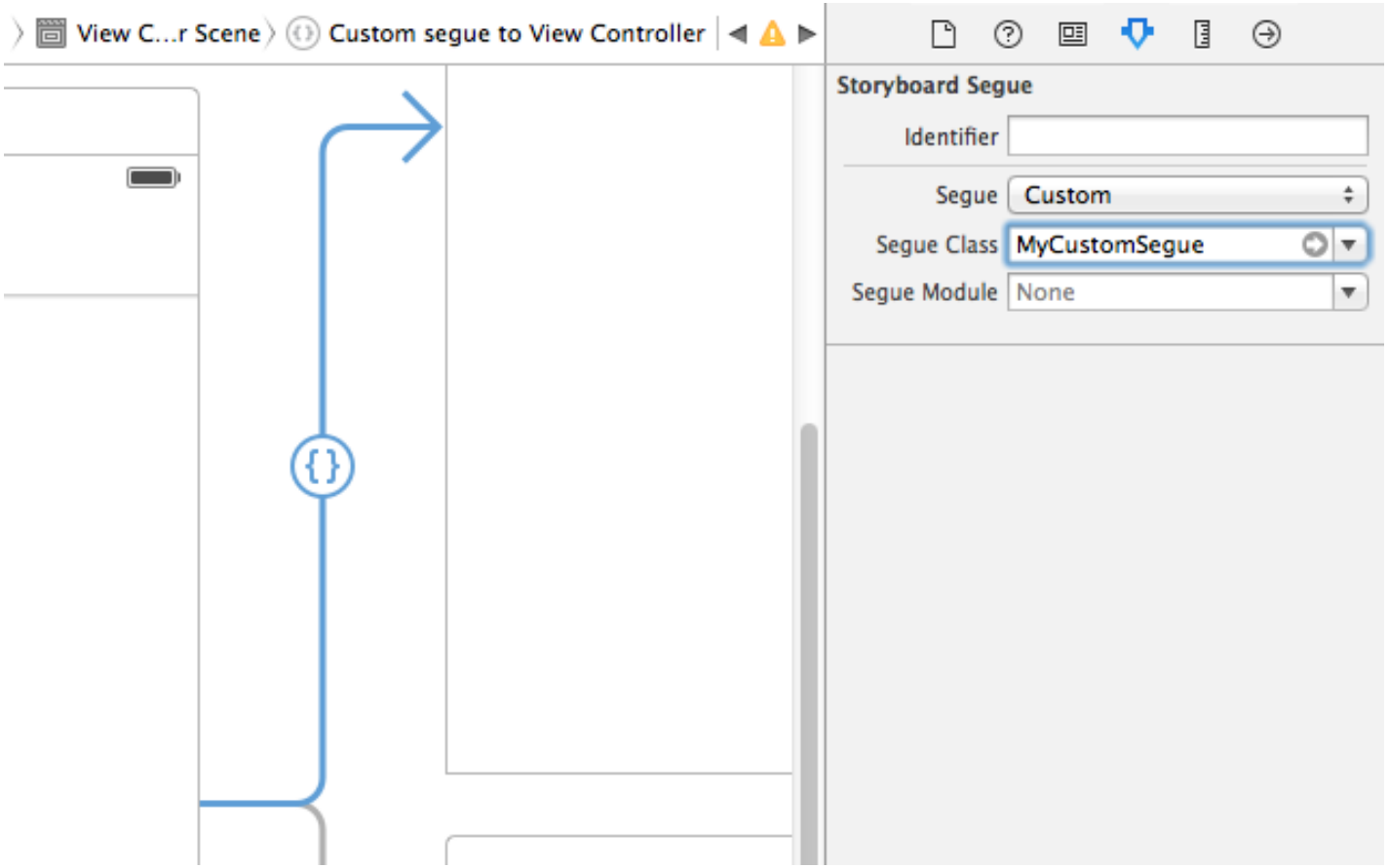
Beispiel: Custom Segue


```
    // Fortsetzung MyCustomSegue.m

    // Starten der Animation
    [UIView beginAnimations:nil context:nil];
    dstCtrl.view.frame = CGRectMake(
        original.origin.x,
        original.origin.y,
        original.size.height,
        original.size.width
    );

    [UIView commitAnimations];

    // Definieren eines Callback (wird am Ende der Animation aufgerufen)
    [self performSelector:@selector(animationDone:)
    withObject:dstCtrl afterDelay:0.2f];
}

-(void)animationDone:(id)vc {
    // Laden des Destination View Controllers
    UIViewController *dstCtrl = (UIViewController*)vc;
    UINavigationController *navController =
        [self.sourceViewController navigationController];

    // Unterdrücken der Standard-Animation
    [navController pushViewController:dstCtrl animated:NO];
}
@end
```

Beispiel: Custom Segue

Ändern des Transitionstyps (Style) auf Custom

Setzen der Segue Class auf MyCustomSegue



Storyboards: Da Allerheilmittel?

Pro

- Einfache Übersetzung des Konzepts in Code
- Simple Darstellung des Interaktionsflusses
- Viel Code kann gespart werden

Contra:

- Kaum geeignet für Team-Arbeit:
  - Typischerweise arbeitet ein Team an einem MVC-Bereich mit eigenem View Controller. Diese Aufteilung ist mit Storyboard nicht mehr möglich
- Keine Unterstützung von Subversion.
  - Konflikte müssen immer manuell behoben werden.
- 'Simplifizierung' kann Hürden bei der Implementierung von komplexeren Apps darstellen
- ViewController werden immer neu instanziiert und können beim Verlassen des Screens nicht zwischen-gespeichert werden

ANHANG

Code für Rotation von Views (Objective C)


```
// AppDelegate.m
#import "AppDelegate.h"

@implementation AppDelegate

-(BOOL)application:(UIApplication *)application
          didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {
     self.window = [[UIWindow alloc] initWithFrame:[[UIScreen mainScreen] bounds]];
     // add first view to window
     CGFloat frameRectViewX = 50;
     CGFloat frameRectViewWidth = 100;
     CGRect frameRect =
          CGRectMake(frameRectViewX, frameRectViewX, frameRectViewWidth, frameRectViewWidth);
     UIView* frameView = [[UIView alloc] initWithFrame:frameRect];
     [frameView setBackgroundColor:[UIColor grayColor]];
     [self.window addSubview:frameView];
     // add second view to first view
     CGFloat boundsViewWidth = frameRect.size.width / M_SQRT2;
     CGFloat boundsViewX = (frameRectViewWidth - boundsViewWidth) / 2;
     CGRect boundsRect =
          CGRectMake(boundsViewX, boundsViewX, boundsViewWidth, boundsViewWidth);
     UIView* boundsView = [[UIView alloc] initWithFrame:boundsRect];
     [boundsView setBackgroundColor:[UIColor blueColor]];
     boundsView.transform = CGAffineTransformMakeRotation(M_PI_4);
     [frameView addSubview:boundsView]; self.window.backgroundColor = [UIColor whiteColor];
     [self.window makeKeyAndVisible];
     return YES;
}
[…]
@end
```

Code für Rotation von Views (Swift)


```
// AppDelegate.swift

import UIKit

@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate {

    var window: UIWindow?

    func application(application: UIApplication,
            didFinishLaunchingWithOptions launchOptions: [NSObject: AnyObject]?) -> Bool {
        window = UIWindow(frame: UIScreen.mainScreen().bounds)
        // add first view to window
        let frameRectViewX:CGFloat = 50
        let frameRectViewWidth:CGFloat = 100
        let frameRect =
            CGRectMake(frameRectViewX, frameRectViewX, frameRectViewWidth, frameRectViewWidth)
        let frameView = UIView(frame: frameRect)
        frameView.backgroundColor = UIColor.grayColor()
        window?.addSubview(frameView)
        // add second view to first view
        let boundsViewWidth:CGFloat = frameRect.size.width / CGFloat(M_SQRT2)
        let boundsViewX = (frameRectViewWidth - boundsViewWidth) / 2
        let boundsRect = CGRectMake(boundsViewX, boundsViewX, boundsViewWidth, boundsViewWidth);
        let boundsView = UIView(frame: boundsRect)
        boundsView.backgroundColor = UIColor.blueColor()
        boundsView.transform = CGAffineTransformMakeRotation(CGFloat(M_PI_4))
        frameView.addSubview(boundsView)
        window?.backgroundColor = UIColor.whiteColor()
        window?.makeKeyAndVisible()
        return true
    }
    [...]
}
```