Fachbereich Automatisierung und Informatik	▲ Hochschule Harz Wernigerode

Vorlesung „iOS mit xcode und Swift“

UI-Skript

Version 13.10.2021

Dipl.-Inf., Dipl.-Ing. (FH) Michael Wilhelm
Friedrichstraße 57 - 59
38855 Wernigerode

Raum: **2.202**
Tel.: **03943/659-338**
Fax: **03943/659-399**
Email: **mwilhlem@hs-harz.de**
Web: **http://www.miwilhelm.de**

Inhaltverzeichnis

Abbildungen

1 Projekt erstellen

- Starten von xcode
- Neues Projekt (Framework-Auswahl):
  - Master-Detail Application (MVC)
  - Paged Bases Application: Editor
  - Single View Application
  - Tabbed Application (Register)
  - StickerView Application
  - iMessage Application
  - Game

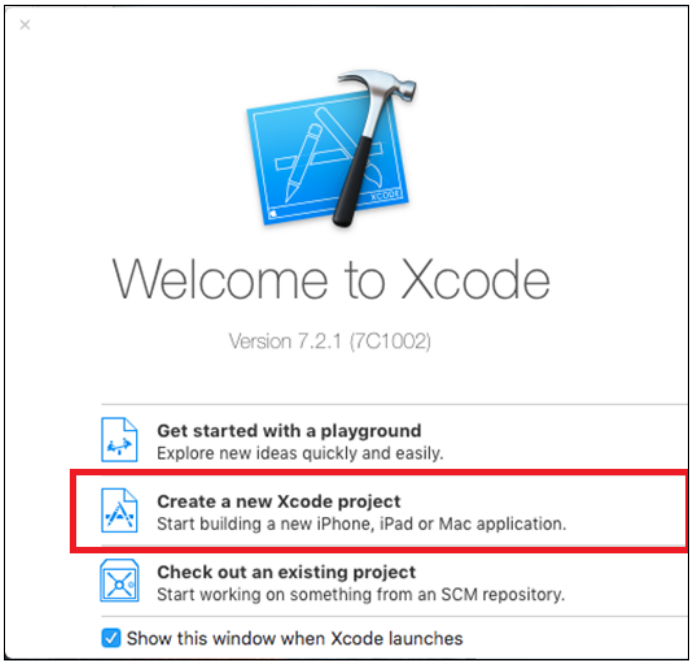


Abbildung 1 Auswahl eines neues Projektes

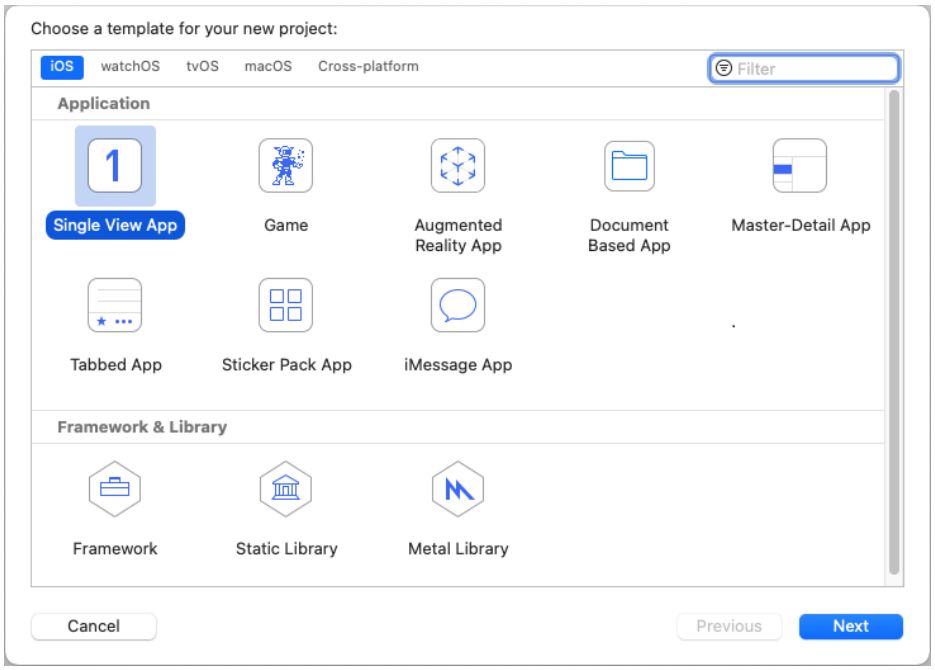


Abbildung 2 Single View Application

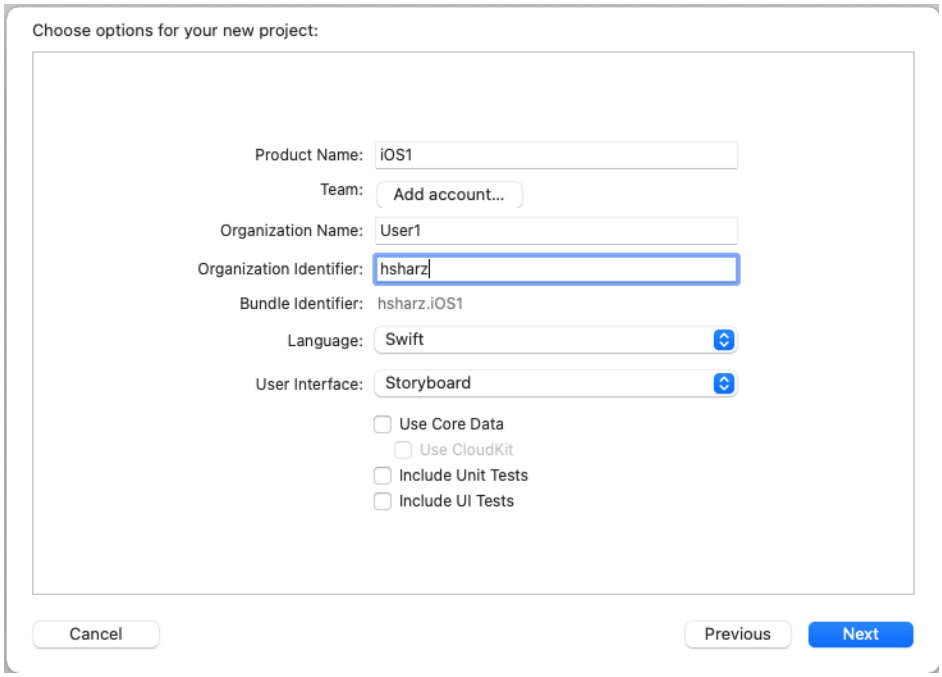


Abbildung 3 Projektdaten

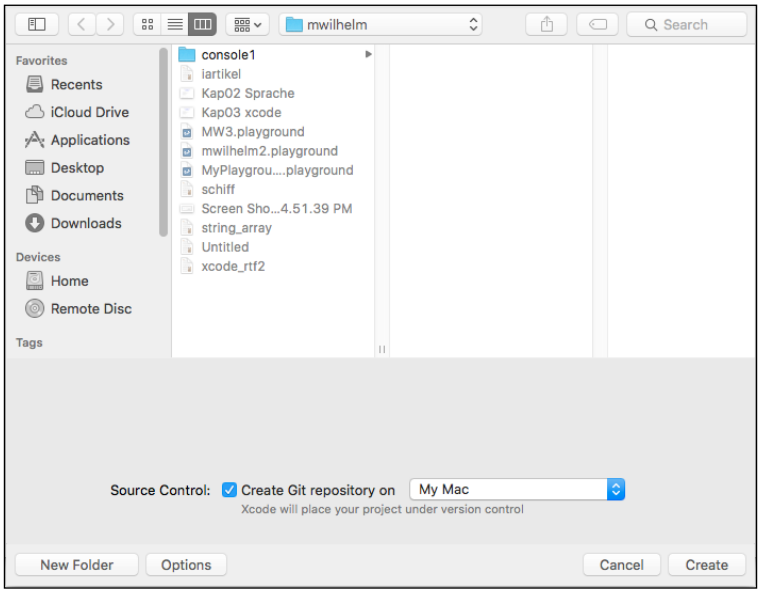


Abbildung 4 Speicherort

1.1 Application Projektdaten

- AppDelegate.swift
  - Startpunkt der App
- ViewController.swift
  - Beinhaltet den Swift-Code eines UI-Views
- Main.storyboard
  - Verwaltet den ViewController oder die ViewControllers.
  - Hier kann man die UI-Elemente einfügen und bearbeiten.
- Assets.xcassets bzw. Images.xcassets.
  - Verwaltet die Logos, Symbole.
- LaunchScreen.storyboard
  - Startfenster, Startlogos.
  - Localisation, Gebrauch mehrerer Sprachen.
- Info.pList
  - Property List Files, auch als Speicherung eigener Daten.

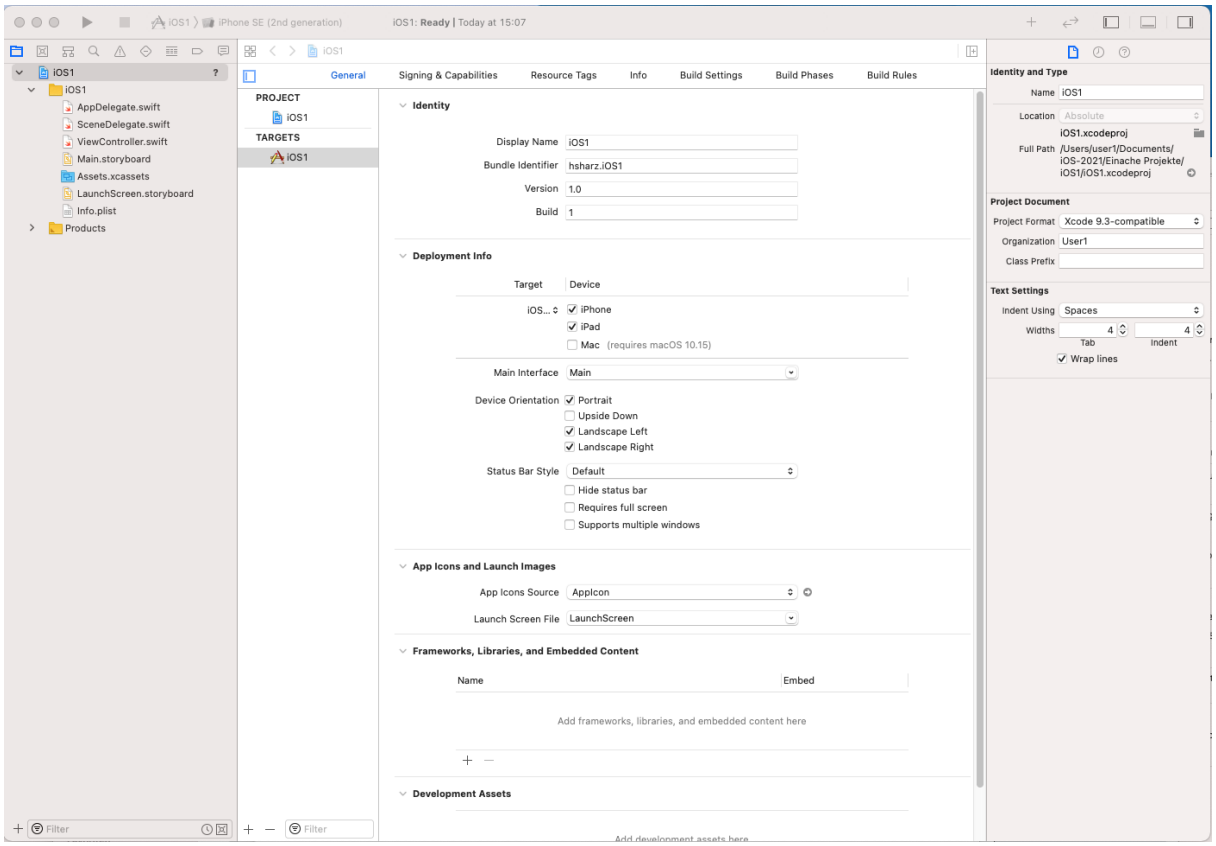


Abbildung 5 SWIFT-IDE

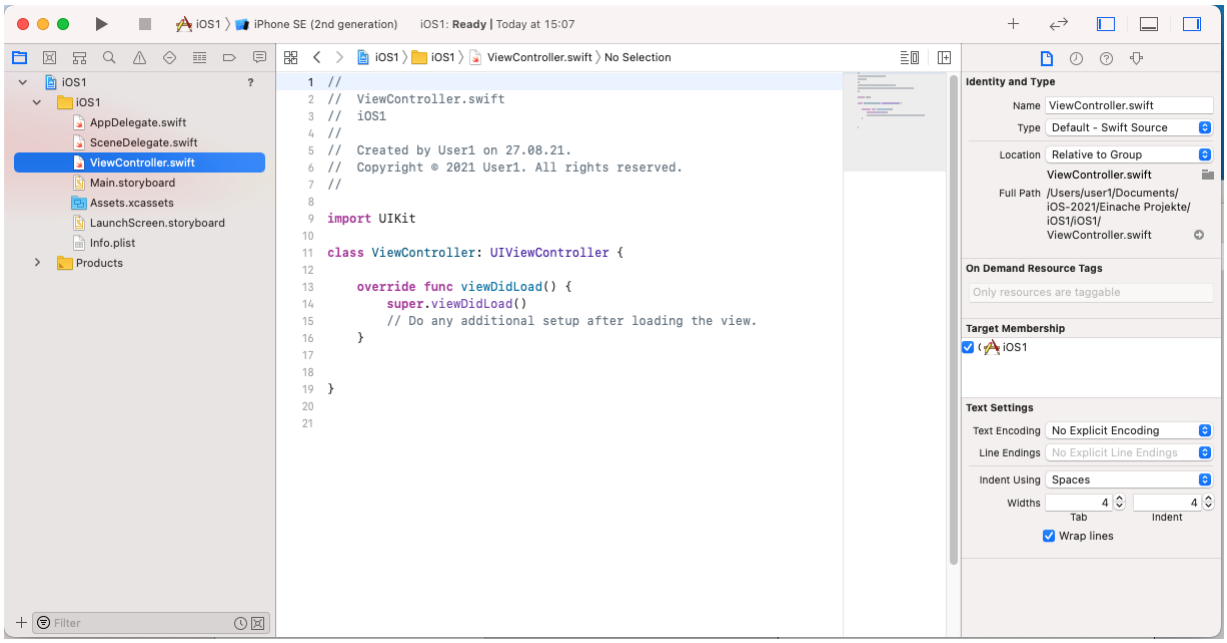


Abbildung 6 SWIFT-IDE

2 UI erstellen

- Öffnen des Fensters „Main.storyboard“.
- Auf der rechten Seite werden die UI-Elemente angezeigt.
- Mit Drag&Drop die Elemente auf den View ziehen.

Mit dem Eintrag „Main.storyboard“ im Projektbaum kann man die zwei Projektansichten aufrufen:

- Anklicken Main.storyboard
- Rechte Maustaste

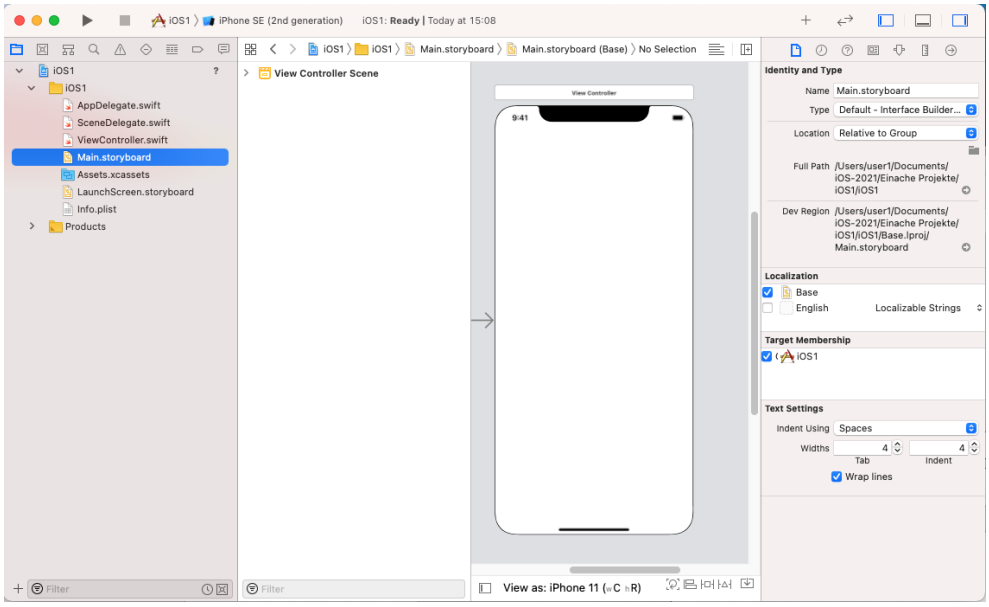


Abbildung 7 Umschalten der Sicht des ViewControllers in der IDE

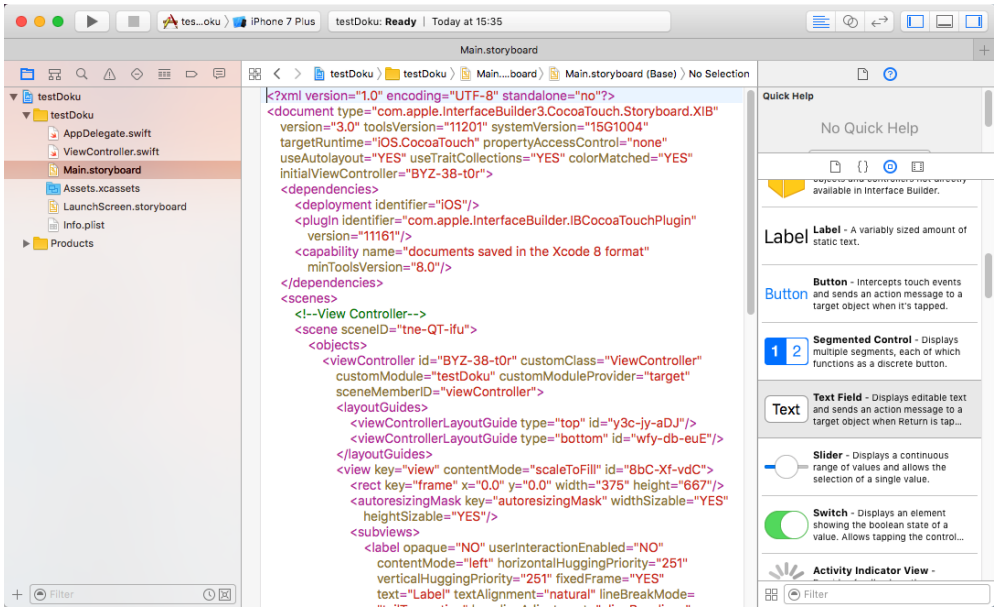


Abbildung 8 ViewController in der XML-Sicht

2.1 Wichtige UI-Elemente

Anzeigen des UI-Listen-Dialogs: Shift-Command-L

Label **Label** - A variably sized amount of static text.	Label-Element: UILabel Anzeige von Texten, readonly **Label haben auch mehrere Zeilen**
Button **Button** - Intercepts touch events and sends an action message to a target object when it's tapped.	Schalter-Element: UIButton onClick-Event
Text **Text Field** - Displays editable text and sends an action message to a target object when Return is tap...	Textfeld: UITextField Einzeilige Texteingabe. Change-Events vorhanden.
1 2 **Segmented Control** - Displays multiple segments, each of which functions as a discrete button.	RadionButton: UISegementedControl Mehrere Schalter in einem „JPanel“ (RadioButton)
Switch - Displays an element showing the boolean state of a value. Allows tapping the control...	An/Aus:UISwitch Schalter, Anzeige eines Zustandes (An,Aus) Eher in Einstellungen zu finden. Benutzt auch als CheckBox.
Slider - Displays a continuous range of values and allows the selection of a single value.	Slider: UISlider Auswahl aus einem Bereich in Zusammenarbeit eines TextField.
Progress View - Depicts the progress of a task over time.	ProgressBar: UIProgressView Progressbar, Fortschrittsbalken, 0,0 bis 1,0
Picker View - Displays a spinning-wheel or slot-machine motif of values.	Liste mit Delegates Anzeige von mehreren Zeilen und Spalten (Uhr)
Date Picker - Displays multiple rotating wheels to allow users to select dates and times.	Datum-Auswahl
Text View - Displays multiple lines of editable text and sends an action message to a target objec...	Mehrzeiliger Editor: UITextView. Change-Events **nicht** vorhanden. Man muss ein Protocol benutzen.

Image View - Displays a single image, or an animation described by an array of images.	Anzeige eines Bildes; UIImageView, Anzeige mit “Aspect to Fit”
Page Control	
Menu Command	
Main Menu	
Sub Menu	
Inline Section Menu	
Title **Navigation Bar** - Provides a mechanism for displaying a navigation bar just below the stat...	Automatisch in der Projekt-App
Navigation Item - Represents a state of the navigation bar, including a title.	
Edit **Toolbar** - Provides a mechanism for displaying a toolbar at the bottom of the screen.	
Item **Bar Button Item** - Represents an item on a UIToolbar or UINavigationItem object.	
Tab Bar - Provides a mechanism for displaying a tab bar at the bottom of the screen.	Benötigt für weitere ViewController in einer TabBar-Application
Tab Bar Item - Represents an item on a UITabBar object.	Benötigt für weitere ViewController in einer TabBar-Application
Search Bar - Displays an editable search bar, containing the search icon, that sends an action messa...	
Search Bar and Search Display Controller - Displays an editable search bar connected to...	
View - Represents a rectangular region in which it draws and receives events.	Entspricht dem Canvas-Element in Java. Zeichnen innerhalb des Elementes

Table View - Displays data in a list of plain, sectioned, or grouped rows.	Tabelle à la JTable: UITableView Man kann eine Erweiterung oder ein Protocol benutzen.
Table View Cell - Defines the attributes and behavior of cells (rows) in a table view.	Benutzt in UITableView

Quelle:
https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIKit_Framework/

2.1.1 UILabel

- Attribute
  - **text**
  - attributedText
  - font
  - textColor
  - textAlignment
  - lineBreakMode
  - enabled
- Größe des Labels verändern
  - adjustsFontSizeToFitWidth
  - allowsDefaultTighteningForTruncation
  - baselineAdjustment
  - minimumScaleFactor
  - **numberOfLines**
- Methoden
  - Keine

2.1.2 UITextField

- Attribute
  - **text**
  - color
  - font
  - aligment
  - placeholder
  - background
  - disabled
  - border style
  - clearbutton
  - min font size
  - capitalization
  - correction
  - spell checking

    - keyboard type
    - appearance
    - return key
- Text Attribute
    - attributedText
    - attributedPlaceholder
    - defaultTextAttributes
    - textColor
    - textAlignment
    - typingAttributes
    -
    - adjustsFontSizeToFitWidth
    - minimumFontSize
- Edit-Verhalten
    - editing
    - **clearsOnBeginEditing**
    - clearsOnInsertion
    - allowsEditingTextAttributes
- Setting
    - borderStyle
    - background
    - disabledBackground
- Views
    - clearButtonMode
    - leftView
    - leftViewMode
    - rightView
    - rightViewMode
- Methoden
    - **ValueChanged**

2.1.3 UIButton

- Attribute
    - **text**
    - shadow offset
    - drawing
    - line break mehrzeilig
    - edge
    - inset Ränder
    - type
    - **StateConfig**
        - **default**
        - **highlighted**
        - **focused**
        - **selected**
        - **isenabled**
    - title Plain oder Format-String
    - image
    - background

- Methoden
    - isenabled lesen und setzen !
    - titleLabel
    - titleForState(_:)
    - setTitle(_:forState:)
        - setTitle("Klick mich", for: UIControl.State.normal)
        - setTitle("Klick mich highlighted", for: UIControl.State.highlighted)
        - setTitle("Klick mich focused", for: UIControl.State.focused)
        - setTitle("Klick mich selected", for: UIControl.State.selected)
        - setTitle("Klick mich disabled", for: UIControl.State.disabled)
    - attributedTitleForState(_:)
    - setAttributedTitle(_:forState:)
    - titleColorForState(_:)
    - setTitleColor(_:forState:)
    - titleShadowColorForState(_:)
    - setTitleShadowColor(_:forState:)
    - reversesTitleShadowWhenHighlighted

    - adjustsImageWhenHighlighted
    - adjustsImageWhenDisabled
    - showsTouchWhenHighlighted
    - backgroundImageForState(_:)
    - imageForState(_:)
    - setBackgroundImage(_:forState:)
    - setImage(_:forState:)
    - tintColor
    -
    - contentEdgeInsets
    - titleEdgeInsets
    - imageEdgeInsets
- Events
    - Touch Up Inside

Weitere Events:

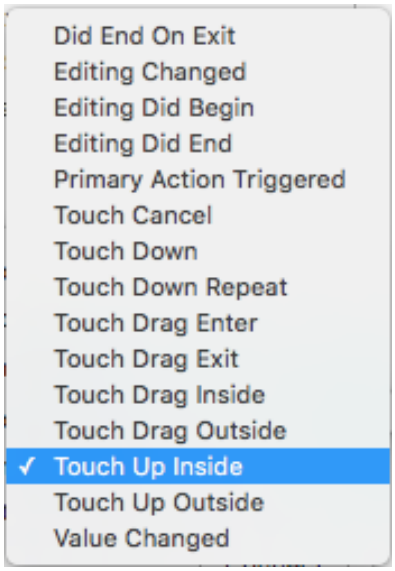


Abbildung 9 Events eines UIButtons

Wichtige Events:

Editing Changed
Touch Up Inside

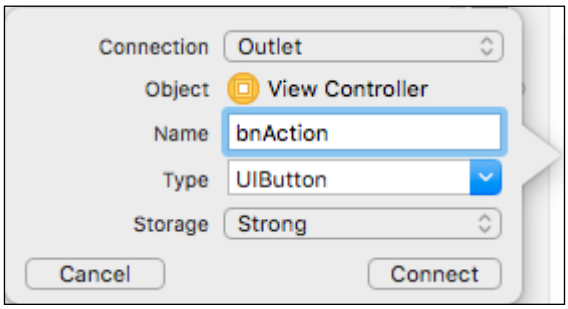


Abbildung 10 Eintragen eines Button als Referenz, um Button zu disabled oder enabled

2.1.4 UISegmentedControl

Abbildung 11 UISegmentedControl, ist auch eine RadioButton

- Attribute
  - **Anzahl Buttons, siehe untere rechte Abbildung**
  - **text**
  - font, size etc.
  - **selectedSegmentIndex**
  - **numberOfSegments**

- Methoden
  - changeValue
  - setImage(_:forSegmentAtIndex:)
  - imageForSegmentAtIndex(_:)
  - setTitle(_:forSegmentAtIndex:)
  - titleForSegmentAtIndex(_:)
  - insertSegmentWithImage(_:atIndex:animated:)
  - insertSegmentWithTitle(_:atIndex:animated:)
  - removeAllSegments()
  - removeSegmentAtIndex(_:animated:)

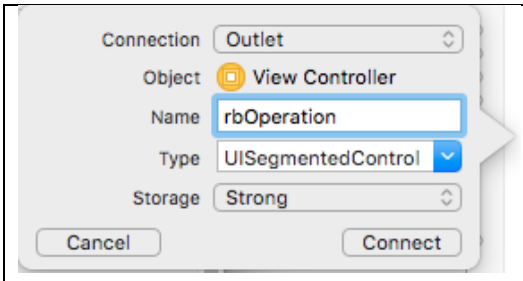


Abbildung 12 Einfügen einer Referenz SegmentedControl

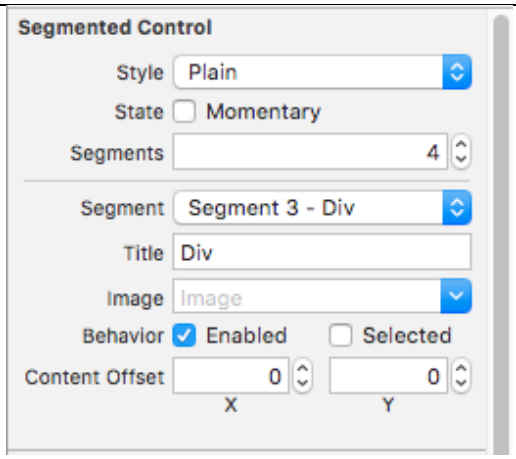


Abbildung 13 Einfügen von neuen Buttons in SegmentedControl

2.1.5 UISwitch

Abbildung 14 UISwitch, funktioniert auch als CheckBox

- Attribute
  - **on**
    - if uiSwitch.on {
    - }
  - font, size etc.
- Methoden
  - **ValueChanged**
  - **setOn(_:animated:)**
- Verhalten ändern
  - onTintColor
  - tintColor
  - thumbTintColor
  - onImage

- offImage

2.1.6 UISlider

Abbildung 15 UISlider

- Attribute
  - value Float
  - setValue(_:animated:)
  - minimumValue Float
  - maximumValue Float
  - font, size etc.
  - continuous:Bool Dauerfeuer
- Methoden
  - ValueChanged
- Anpassbarkeit:
  - minimumValueImage
  - maximumValueImage
  - minimumTrackTintColor
  - currentMinimumTrackImage
  - minimumTrackImageForState(_:)
  - setMinimumTrackImage(_:forState:)
  - maximumTrackTintColor
  - currentMaximumTrackImage
  - maximumTrackImageForState(_:)
  - setMaximumTrackImage(_:forState:)
  - thumbTintColor
  - currentThumbImage
  - thumbImageForState(_:)
  - setThumbImage(_:forState:)

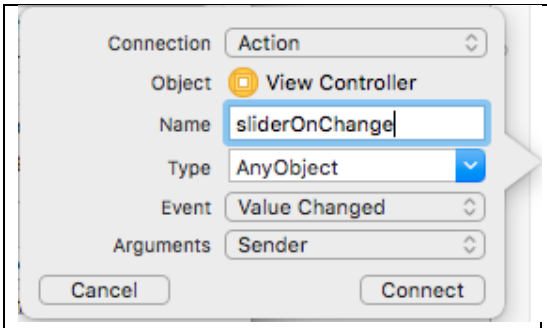


Abbildung 16 Einfügen einer Referenz Slider

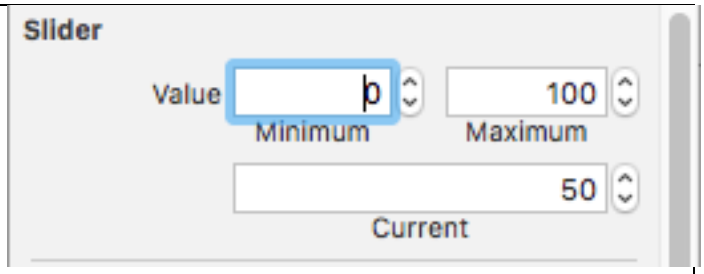


Abbildung 17 Ändern der Grenzen eines Sliders

2.1.7 UIStepper

Abbildung 18 UIStepper, funktioniert auch als ComboBox mit einem TextField und Array

- Attribute
  - **value** **Aktueller Wert, Datentyp ist DOUBLE**
  - **minimumValue**
  - **maximumValue**
  - **continuous**
  - **autorepeat**
  - font, size etc.
  - stepValue
  - wraps
- Wichtige Methode
  - ValueChanged
- Anpassen mit
  - tintColor
  - backgroundImageForState(_:)
  - setBackgroundImage(_:forState:)
  - decrementImageForState(_:)
  - setDecrementImage(_:forState:)
  - dividerImageForLeftSegmentState(_:rightSegmentState:)
  - setDividerImage(_:forLeftSegmentState:rightSegmentState:)
  - incrementImageForState(_:)
  - setIncrementImage(_:forState:)

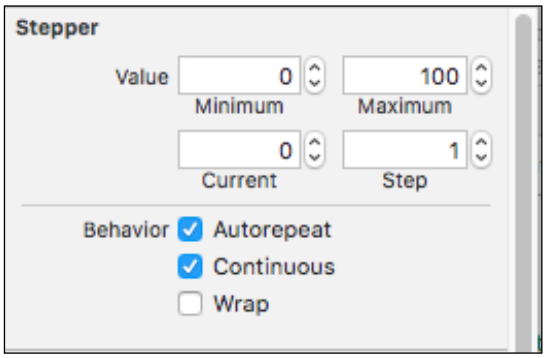


Abbildung 19 Ändern der Grenzen eines Steppers

2.1.8 UIProgressView

- Attribute

    - **progress**
    - **setProgress(_:animated:)**
    - **minimum immer 0,0**
    - **maximum immer 1,0**
    - font, size etc.
    - observedProgress
- Methoden
    - ValueChanged
    - progressViewStyle
    - progressTintColor
    - progressImage
    - trackTintColor
    - trackImage

2.1.9 UIPickerView

Hiermit wählt man aus einer Liste einen Wert aus. Man kann aber auch zwei unabhängige Spalten definieren (siehe Beispiel Uhr oder Binary2).

- Attribute
    - font, size etc.
    - numberOfComponents
    - numberOfRowsInComponent(_:)
    - rowSizeForComponent(_:)
    - **delegate**
    - dataSource
- Selected-Methoden
    - selectRow(_:inComponent:animated:)
    - selectedRowInComponent(_:)
- Methoden
    - **ValueChanged**

Beispielcode:

```
class ViewController: UIViewController, UIPickerViewDelegate,
UIPickerViewDataSource {

    let staedte = ["Brüssel", "Mailand", "Moskau", "Kairo", "Paris",
"Wernigrode", "Hamburg", "New York",
                   "Tokio","London", "Madrid","Rom"]

    // outlets
    @IBOutlet var pickerview: UIPickerView!
    @IBOutlet var tIndex: UITextField!
    @IBOutlet var tValue: UITextField!

    @IBAction func bnAktionClick(_ sender: UIButton) {
        let row = pickerview.selectedRow(inComponent: 0)
        tIndex.text = String(row)
        tValue.text = staedte[row]
    }
```

```
    func numberOfComponents(in pickerView:UIPickerView)-> Int{
        return 1
    }

    func pickerView(_ pickerView:UIPickerView, numberOfRowsInComponent
component:Int) -> Int{
        return cities.count
    }
    func pickerView(_ pickerView:UIPickerView, titleForRow row:Int,
forComponent component:Int) -> String?{
        return staedte[row]
    }

    func pickerView(_ pickerView: UIPickerView, didSelectRow row: Int,
inComponent component: Int) {
        // selected Index
    }

    override func viewDidLoad() {
        super.viewDidLoad()
        self.pickerview.delegate=self
        pickerview.dataSource = self
    }


    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
    }

}
```

Für die PickerView benötigt man vier Methoden aus dem Protocoll bzw. Interface

Name der Methode	**Beschreibung**
numberOfComponents(in pickerView:UIPickerView)-> Int	Die Anzahl der Spalten
pickerView(_ pickerView:UIPickerView, numberOfRowsInComponent component:Int) -> Int	Die Anzahl der Zeilen. Über den Parameter „component“ kann man die Anzahl pro Spalte definieren (siehe zweites Beispiel).
func pickerView(_ pickerView:UIPickerView, titleForRow row:Int, forComponent component:Int) -> String?	Der Inhalte für die Spalte und Zeile. Der Parameter „component“ seletioert die Spalte.
pickerView(_ pickerView: UIPickerView, didSelectRow row: Int, inComponent component: Int	Das „ChangeEvents“ der PickerView

Beispiel in einem View:

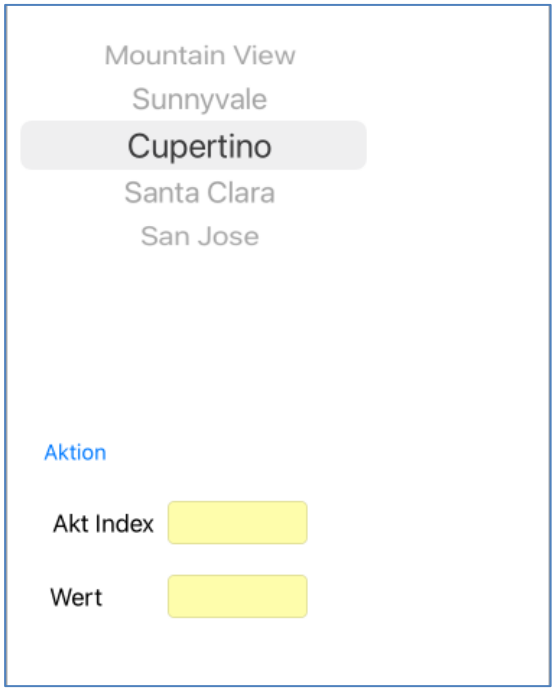


Abbildung 20 PickerView für das Beispiel ConvertNumbers1

Man kann PickerView auch mit mehreren Spalten darstellen. Beim iPhone wäre das zum Beispiel die Uhrzeit.

Beispiel ConvertNumbers2:

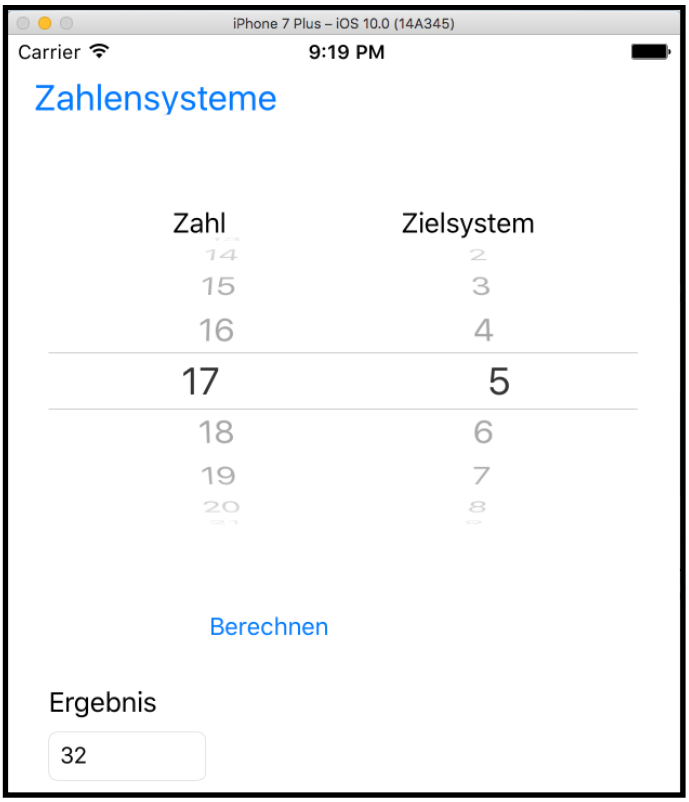


Abbildung 21 Ein PickerView mit zwei Spalten

Aufgabe:

- Auswahl einer Zahl (im Zehnersystem)
- Auswahl eines Zielzahlensystems
- Umrechnen der Eingabezahl

Eigenschaften:

- Zwei Eingaben per PickerView
  - Dezimalzahl in der ersten „Rolle“
  - Basissystem in der zweiten „Rolle“

- Globale Variablen:
  - let minZahl:Int=0
  - let maxZahl:Int=3000
  - let minSystem:Int=2
  - let maxSystem:Int=30

- Delegate-Prinzip eines PickerViews
  - numberOfComponentsInPickerView
  - pickerView (Anzahl der Spalten)
  - pickerView (Stringwert in Abhängigkeit der Spalte)

Quellcode:

```
import UIKit

class ViewController: UIViewController, UIPickerViewDelegate,
UIPickerViewDataSource {


    let minZahl:Int=0
    let maxZahl:Int=300
    let minSystem:Int=2
    let maxSystem:Int=30

    // Refrenzen
    @IBOutlet var tergebnis: UITextField!
    @IBOutlet var pickerview: UIPickerView!


    override func viewDidLoad() {
        super.viewDidLoad()
        self.pickerview.delegate=self
        pickerview.dataSource = self
    }

      // Anzahl der Spalten
    func numberOfComponents(in pickerView:UIPickerView)-> Int{
        return 2
    }

      // Ausgabe der Anzahl der Zeilen
    func pickerView(_ pickerView:UIPickerView, numberOfRowsInComponent
component:Int) -> Int{
        switch (component) {
            case 0: return maxZahl-minZahl+1
            case 1: return maxSystem-minSystem+1
            default: tergebnis.text = "Falsche Spalte in pickerView
getRowCount(column)"
                    return 0
        }
    }


      // Rückgabe der aktuellen Zelle, man beachte die switch/case
    func pickerView(_ pickerView:UIPickerView, titleForRow row:Int,
forComponent component:Int) -> String?{
        switch (component) {
        case 0: return String(minZahl+row)
        case 1: return String(minSystem+row)
        default: tergebnis.text = "Falsche Spalte in pickerView getCell"
                return "xxxxx"
        }
    }

      // Events für die aktuelle Position
    func pickerView(_ pickerView: UIPickerView, didSelectRow row: Int,
inComponent component: Int) {
        // labelErgebnis2.text = staedte[row]    //
```

```
    }



      // Berechnen der neuen Zahl
    @IBAction func bncalcclick(_ sender: UIButton) {
          // holen der beiden Werte aus den Spalten
        let row1:Int = pickerview.selectedRow(inComponent: 0)
        let row2:Int = pickerview.selectedRow(inComponent: 1)

        let zahl:Int=row1+minZahl
        if zahl<0 {
            tergebnis.text="Fehlerhafte Eingabe in Eingangszahl"
        }

        let system:Int=row2+minSystem
        if system<2 || system>30 {
            tergebnis.text="Fehlerhafte Eingabe im Ausgangszahlensystem"
        }

        let result:String = base_convert2(number: zahl, base: system)
        tergebnis.text=result

    }

    func base_convert2(number:Int , base:Int) -> String {
        let hex : [String] = ["0" , "1" , "2" , "3" , "4" , "5", "6" , "7"
, "8" , "9" , "A" , "B" , "C" , "D" , "E" , "F" , "G" , "H" , "I" , "J" ,
"K" , "L" ,
                              "M" , "N" , "O" , "P" , "Q" , "R" , "S" , "T"
, "U" , "V" , "W" , "X", "Y", "Z"
        ]
        var erg : String = ""
        var zahl : Int = number
        var rest : Int = 0;
        while (zahl>0) {
            rest = zahl % base
            erg = hex[ rest ] + erg
            zahl =  zahl / base
        }
        if  erg.count<2 {
            erg = "0" + erg
        }
        return erg;
    }

}
```

2.1.10 UITextView

Dieses UI-Element ist ein Multiline-Editor.

- Attribute
  - font, size etc.
  - text
- Methoden
  - ValueChanged

Man sollte immer eine ScrollView einfügen:

- Erst den ScrollView einfügen.
- Dann den TextView in den ScrollView einfügen. Kontrollieren kann man das über den Projektbaum.
- Das Verfahren entspricht dem eines JScrollPane und einem JTextArea in Java.

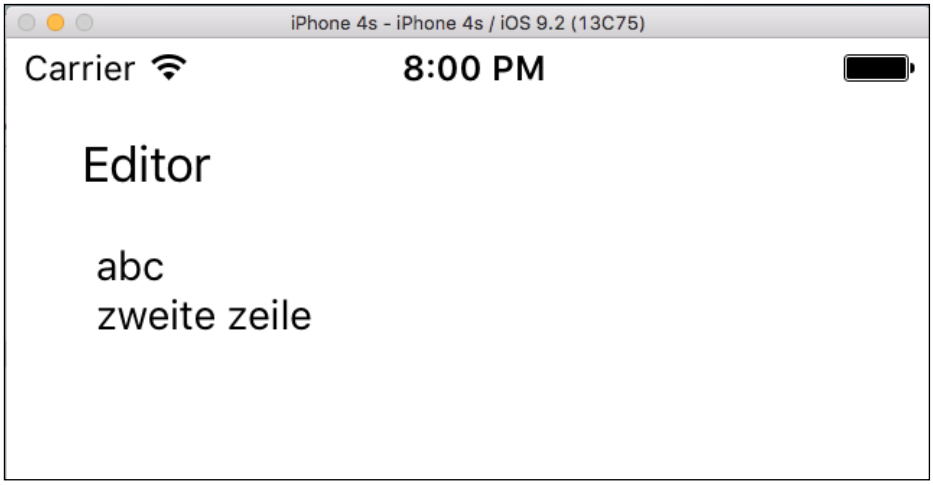


Abbildung 22 Beispiel für einen TextView

2.1.11 UIDatePicker

- Attribute
  - calendar
  - date
  - locale
  - setDate:animated
  - timeZone
- Konfiguration des Date Picker Modus
  - datePickerMode
- Konfiguration der Zeit-Attribute
  - maximumDate
  - minimumDate
  - minuteInterval
  - countDownDuration

Beispiel:

- labelErgebnis.text = uiDatePicker.date.description

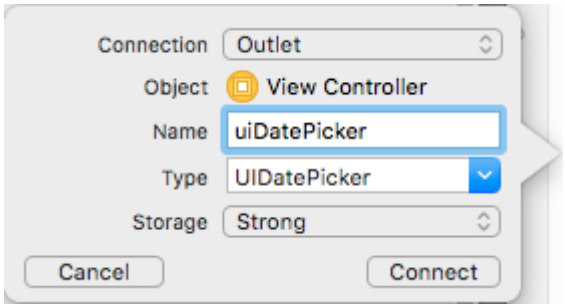


Abbildung 23 Outlet eines DatePickers eintragen

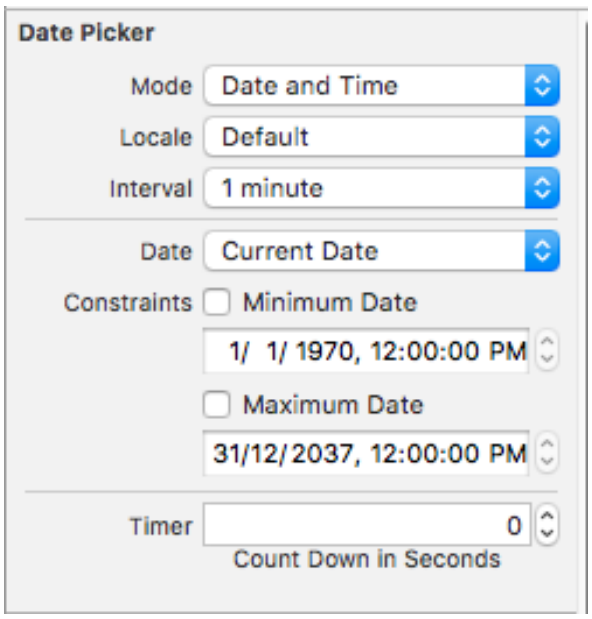


Abbildung 24 Eigenschaften eines DatePickers

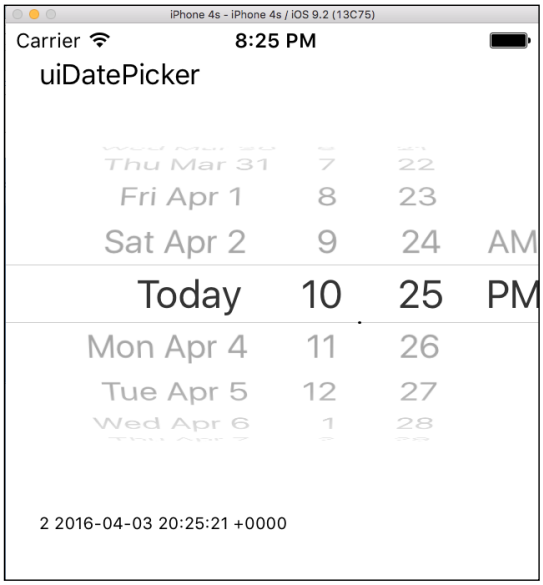


Abbildung 25 Beispiel eines DatePicker mit Anzeigen des aktuellen Wertes

Abfragen der Werte:

- let calendar = Calendar.current
- let jahr = calendar.component(.year, from: datepicker.date)
- let monat = calendar.component(.month, from: datepicker.date)
- let tag = calendar.component(.day, from: datepicker.date)
- labeldatum.text = datepicker.date.description // toString-Methode

Setzen der Styles und Mode eines DatePickers:

- datepicker.datePickerMode = UIDatePicker.Mode.date
- datepicker.datePickerMode = UIDatePicker.Mode.dateAndTime
- datepicker.datePickerMode = UIDatePicker.Mode.time
- datepicker.datePickerMode = UIDatePicker.Mode.countDownTimer
- datepicker.datePickerMode = UIDatePicker.Mode.date

- datepicker.preferredDatePickerStyle = .inline
- datepicker.preferredDatePickerStyle = .automatic
- datepicker.preferredDatePickerStyle = .wheels
- datepicker.preferredDatePickerStyle = .compact
- datepicker.preferredDatePickerStyle = .automatic

Abbildung 26 compact-Style eines DatePickers

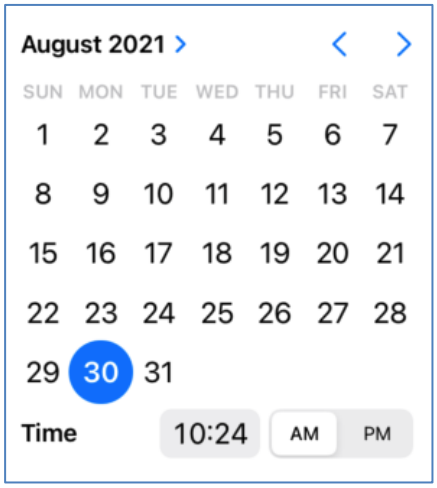


Abbildung 27 inline-Style eines DatePickers

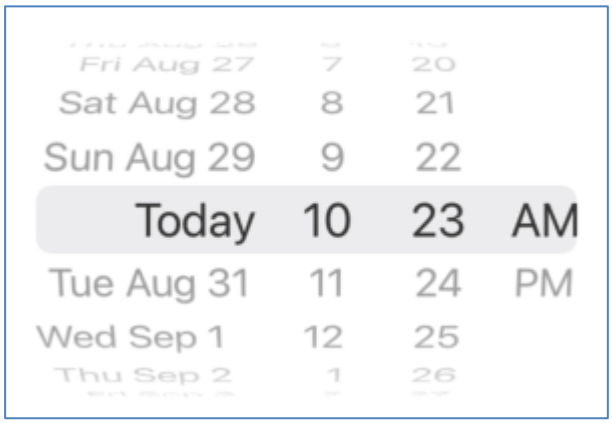


Abbildung 28 wheel-Style eines DatePickers

Setzen der Min- und MaxDate eines DatePickers:

- let calendar = Calendar.current
- var components = DateComponents()
- components.day = 1
- components.month = 1
- components.year = 2017
- components.hour = 0
- components.minute = 0
- let minDate = calendar.date(from: components)
- datepicker.minimumDate=minDate

- components.day = 31
- components.month = 12
- components.year = 2021
- components.hour = 0

- components.minute = 0
- let maxDate = calendar.date(from: components)
- datepicker.maximumDate=maxDate

2.1.12 UIActivityIndicator

- Methoden
    - **startAnimating()**
    - **stopAnimating()**
    - isAnimating()
    - hidesWhenStopped

Beispiel:

```
uiActivityIndikator.startAnimating()
  // Aktion
uiActivityIndikator.stopAnimating()
```

2.1.13 uiTableView

Arbeitet ähnlich der JTable. Man benötigt ein grafisches Element uiTableView, aber des Weiteren auch ein Protokoll mit vier Funktionen und zwei Delegate-Anweisungen.

Eigenschaften eines uiTableViews:

- Anzeige von Arrays in Spalten und Zeilen
- Jede Zeile ist eine Struktur bzw. ein Object
- Jede Spalte kann unterschiedliche grafische Elemente beinhalten.
- Die Zeilen können **gruppiert** werden.
- Die TableView holt nun durch eine Funktion JEDE Zelle.

Weitere UI-Elemente:

- UINavigationController:
  Es erscheint ein Pfeil zur weiteren Navigation am Ende der Zeile

Unterschied zwischen TableView und TableViewController:

- Ein TableViewController nimmt den kompletten Raum des Views an.
- Man kann keine weiteren Elemente in die View einfügen.

Ablauf:

- Erzeugen eines Projektes
- Einfügen eines uiTableView in den Start-ViewController
- Eine Referenz, Outlet, im Quellcode erzeugen
- Dass Protokoll „UITableViewDataSource“ eintragen.
    - class meinTableView: UIViewController, **UITableViewDataSource** {

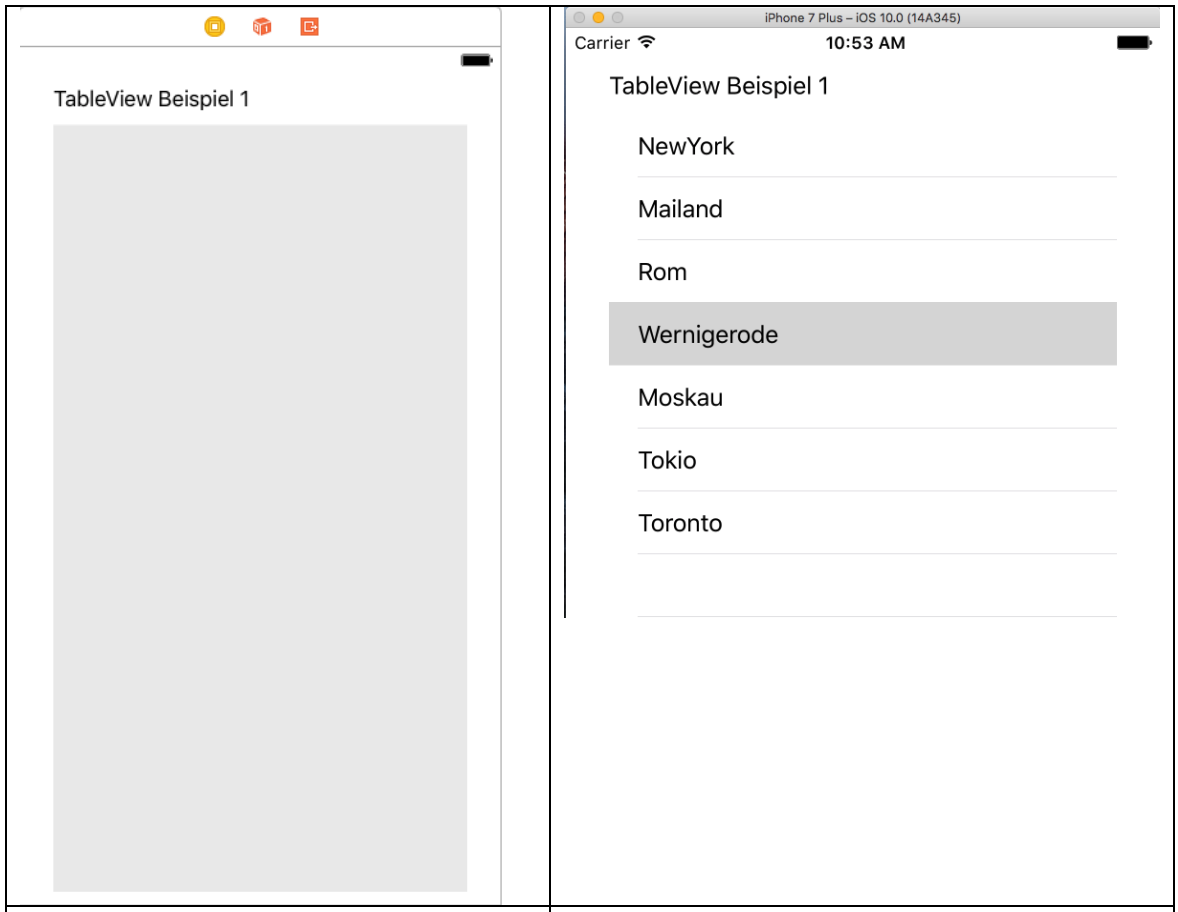
Wichtige Funktionen des Protokolls „**UITableViewDataSource**“

- `func numberOfSections(in tableView: UITableView) -> Int`
  `Anzahl der Spalten, normalerweise 1`

- `func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int`
  `Anzahl der Spalten`

- `func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell`
  `Erstellen einer Zelle, holen des Inhaltes jeder Zelle`

2.1.14 TableView1



Entwurf im ViewController	Starten des Programms

Ablauf:

- Einfügen eines UITableView-Elementes
- Eintragen der drei Protokolle (Interfaces) für UITableViewDataSource
- Eintragen des Daten-Arrays
- Vervollständigen der Methoden

ViewController:

```
class ViewController: UIViewController, UITableViewDelegate,
UITableViewDataSource {

  let cities = ["NewYork", "Mailand", "Rom",
    "Wernigerode","Moskau", "Tokio", "Toronto"]

  // outlets
  @IBOutlet var uiTableView: UITableView!


  func numberOfSections(in tableView: UITableView) -> Int {
    return 1        // Anzahl der Spalten
  }

  // Anzahl der Elemente für jede Gruppe, ev, Parameter gruppe
  func tableView(_ tableView: UITableView,
          numberOfRowsInSection section: Int) -> Int {
    return cities.count     // Anzahl der Zeilen
  }


  // holen des Inhaltes, Aufbau einer UITableViewCell
  func tableView(_ tableView: UITableView,
      cellForRowAt indexPath: IndexPath) -> UITableViewCell  {
    let cell:UITableViewCell=UITableViewCell(
    style: UITableViewCellStyle.default, reuseIdentifier: "cell")

    let row = (indexPath as NSIndexPath).row
    let city = cities[ row ]
    cell.textLabel!.text=city
    return cell

  }


    override func viewDidLoad() {
        super.viewDidLoad()
        uiTableView.dataSource = self
        uiTableView.delegate=self
    }

}
```

Mit der obigen Methode wird jedesmal ein neues Cell-Objekt erstellt. Diese Technik ist natürlich Speicherverschwendung.

Bessere Alternative:

```
func tableView(_ tableView: UITableView, cellForRowAt indexPath:
IndexPath) -> UITableViewCell {
```

```
    // kann ich eine Zelle wiederverwenden?
   var cell = tableView.dequeueReusableCell(withIdentifier: "cell")
   if cell == nil {
     cell = UITableViewCell(
       style: UITableViewCellStyle.default, reuseIdentifier: "cell")
   }
   let row = (indexPath as NSIndexPath).row
   let city:String = cities[ row ]
    cell?.textLabel!.text = city  // .name
   return cell!
}
```

2.1.14.1 Typen der cell-Instanz:

Insgesamt gibt es vier vordefinierte Formate:

Typ	Erläuterung
UITableViewCellStyle.default	Bild links, Titel rechts
UITableViewCellStyle.subtitle	Bild links, Titel rechts, darunter Zusatzinformation
UITableViewCellStyle.value1	Bild links, Titel in der Mitte, rechts Zusatzinformation
UITableViewCellStyle.value2	Zusatzinformation links, rechts Titel

Mögliche Elemente in der Zelle:

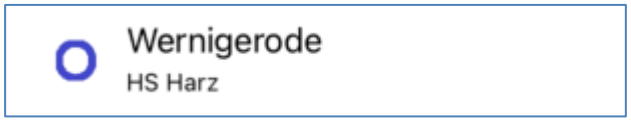
Typ	Zuweisung
textLabel	cell?.textLabel!.text ="City"
detailTextLabel	cell?.detailTextLabel!.text ="City"
imageView	cell?.imageView!.image = UIImage(named: "Bild6")

Beispiele:

default:

value1

value2



subtitle:

2.1.14.2 Selbstdefiniertes cell-Format

Vorgehensweise:

- Neues Projekt erstellen
- Eintragen einer TableView
- Eintragen eines Outlets für die TableView
  - @IBOutlet var tableview: UITableView!
- Eintragen der Delegates
  - UITableViewDelegate, UITableViewDataSource
- Die Verwaltungsklasse erstellen (Cmd-N, Klasse Student, NSObject)
- Liste für die Klasse erstellen
  - var liste = [Student]()
- Eintragen der
- Eintragen der drei Methoden:
  - func numberOfSectionsInTableView(tableView: UITableView) -> Int {}
  - func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {}
  - func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell{}
- Eintragen der Referenz in viewDidLoad
  - tableview.delegate=self
  - tableview.dataSource=self
  - tableview.tableFooterView = UIView(frame: CGRect.zero)
- **Definieren eines Prototypes:**

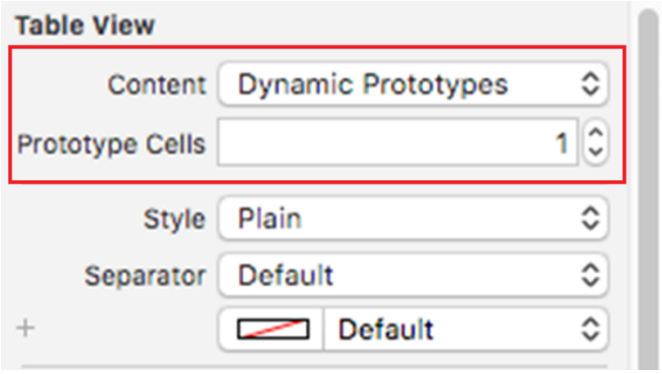


Abbildung 29 Definieren eines Prototypes

Nun erscheint in der TableView im MainStoryboard ein Rechteck, in dem man die UI-Elemente einfügen kann.

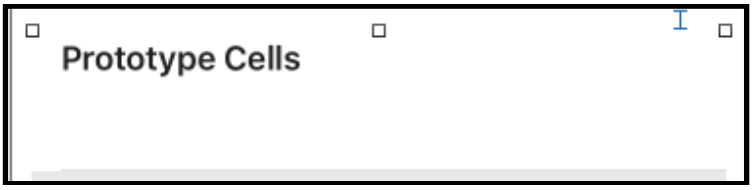


Abbildung 30 Prototyp Cells Bereich

- **Eintragen des Namens**

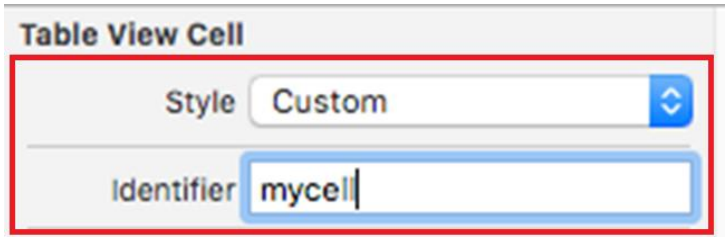


Abbildung 31 Eintragen eines Namens für die Protypcell

Im Projektbaum wird nun die Zelle dargestellt:

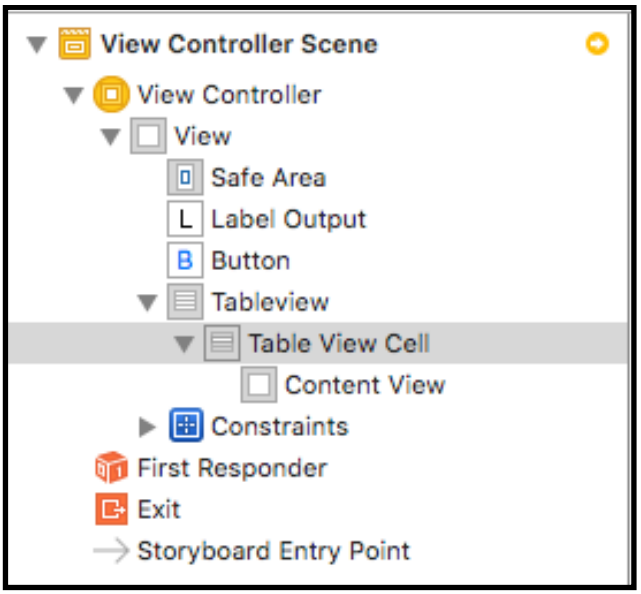


Abbildung 32 Anzeige des Projektbaum bezüglich der Prototyp-Zelle

- **Einfügen der UI-Elemente:**

Abbildung 33 Ein ImageView und zwei LabelView für Prototyp-Zelle

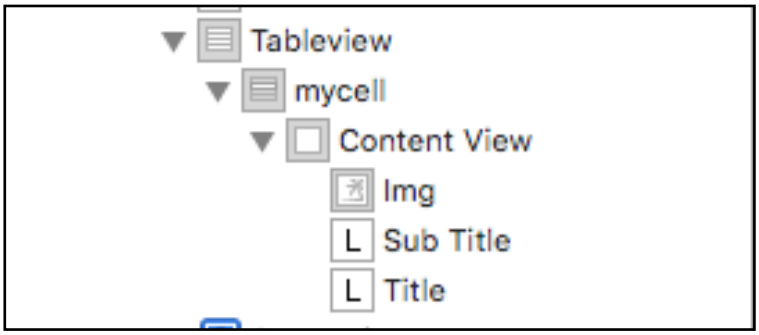


Abbildung 34 Ein ImageView und zwei LabelView für Prototyp-Zelle im Projektbaum

Um die Elemente anzusprechen, benötigt man jeweils einen Namen (à la subtitle). Dazu muss man eine Klasse definieren und Outlets hinzufügen.

- **Definition einer Klasse für die Prototyp-Zelle:**
  - **Cmd+N**

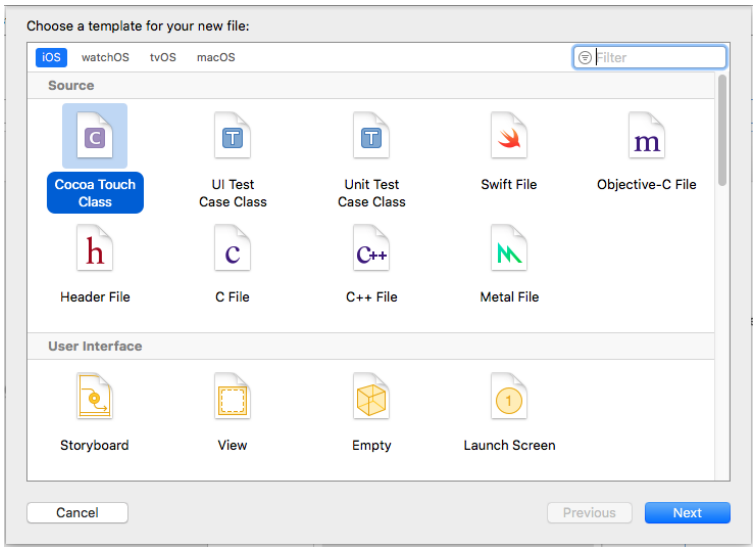


Abbildung 35 Auswahl des Template für die neue Klasse

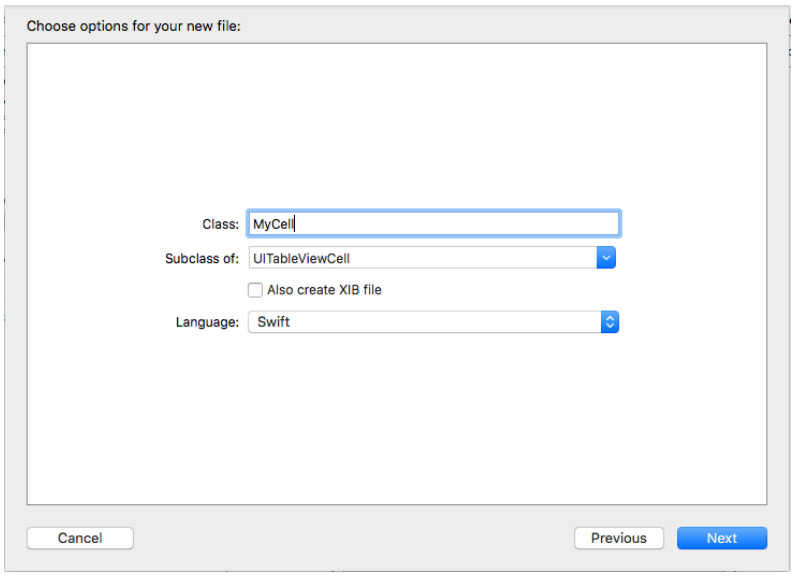


Abbildung 36 Klasse für die Prototyp-Zelle

Die Oberklasse ist nicht NSObject, sondern UITableViewCell.

- **Zuweisen der Klasse zur Prototyp-Zelle**

Dieser Schritt weist der Prototyp-Zelle die neue Klasse zu:

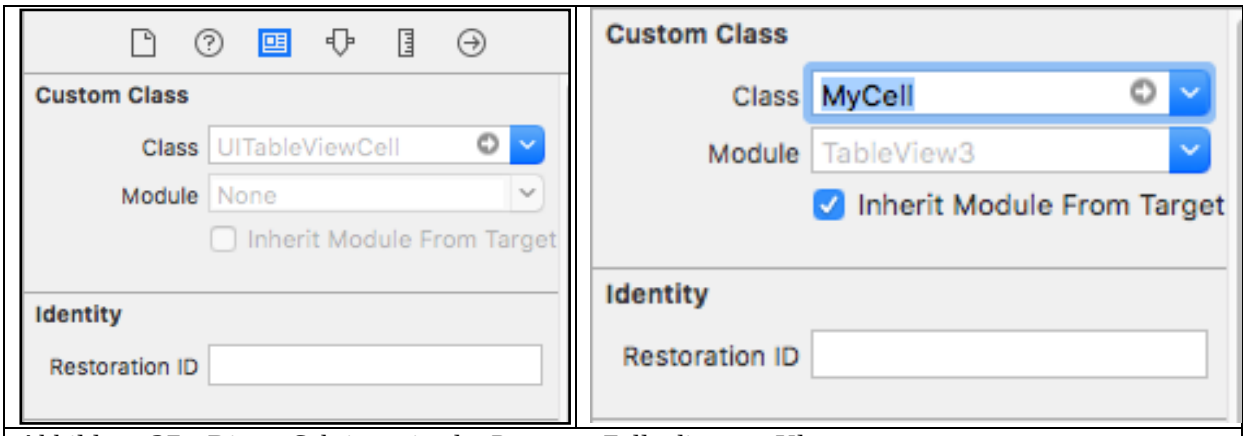


Abbildung 37 Dieser Schritt weist der Prototyp-Zelle die neue Klasse zu:

- **Eine Klasse MyCell erstellen**
  - MyCell ist abgeleitet von NSObject

Quellcode der Klasse MyCell:

```
import UIKit

class MyCell: NSObject {
  @IBOutlet var img: UIImageView!
  @IBOutlet var title: UILabel!
  @IBOutlet var subTitle: UILabel!

}
```

- **Einbau der "draw-cell-Methode**

```
func tableView(_ tableView: UITableView, cellForRowAt indexPath:
IndexPath) -> UITableViewCell {

   let cell = tableView.dequeueReusableCell(
            withIdentifier: "mycell", for: indexPath) as! MyCell

   let row = (indexPath as NSIndexPath).row
   let student:Student = liste[ row ]
   cell.title!.text = student.name
   cell.subTitle!.text=String(student.mnr)
   switch student.semester {
     case 1:
        cell.img!.image = UIImage(named: "sem1")
     case 2:
        cell.img!.image = UIImage(named: "sem2")
     case 3:
        cell.img!.image = UIImage(named: "sem3")
     case 4:
        cell.img!.image = UIImage(named: "sem4")
```

```
        case 5:
            cell.img!.image = UIImage(named: "sem5")
        case 6:
            cell.img!.image = UIImage(named: "sem6")
        default:
            cell.img!.image = UIImage(named: "bachelor")
    }  // switch

    return cell
}
```

2.1.14.3 TableView mit einer Detailansicht

Ablauf:

- Anzeige der Accessory: Details

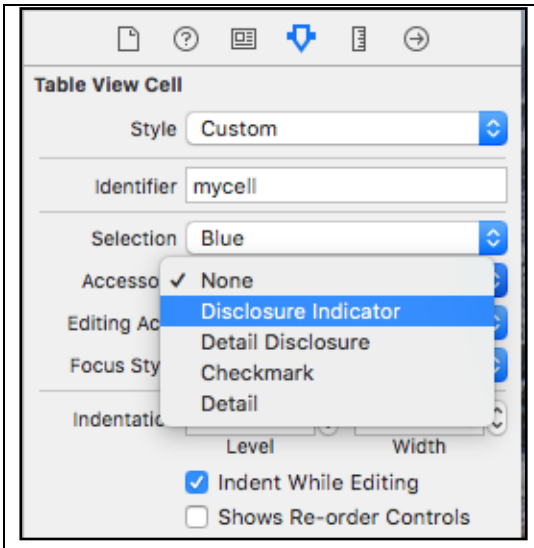
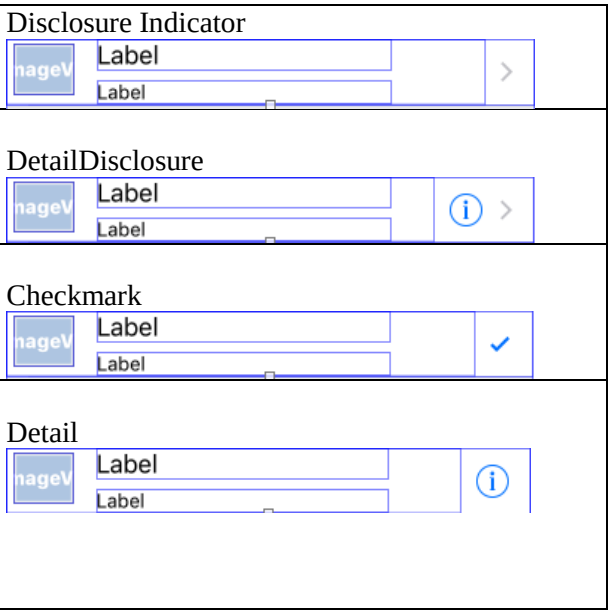


Abbildung 38 Anzeige der Accessory



Die mit diesen Symbole signalisiert man dem Anwender, dass e seine Details-Ansicht gibt. Mit dem "Klick" auf ein Element wird die Detailsansicht geöffnet.

- Einfügen eines Navigationskontroller

Am einfachsten mit folgendem Menüeintrag:

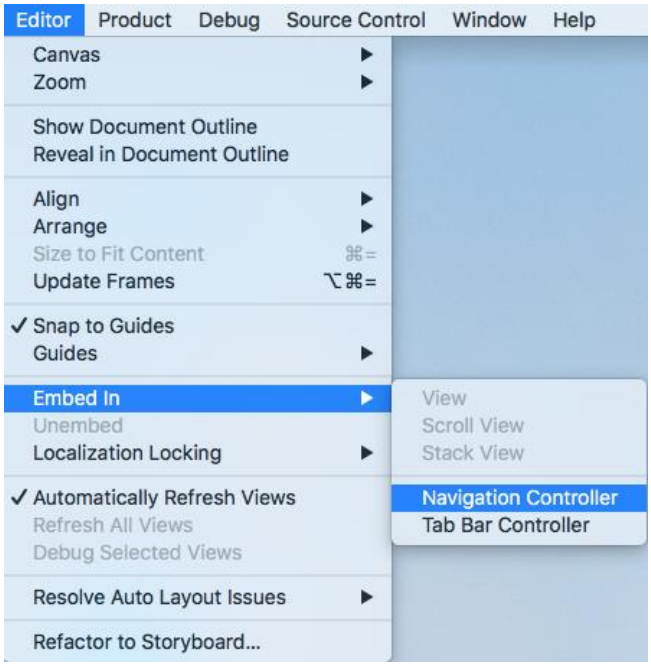


Abbildung 39 Einfügen eines Navigationskontrollers

Nun

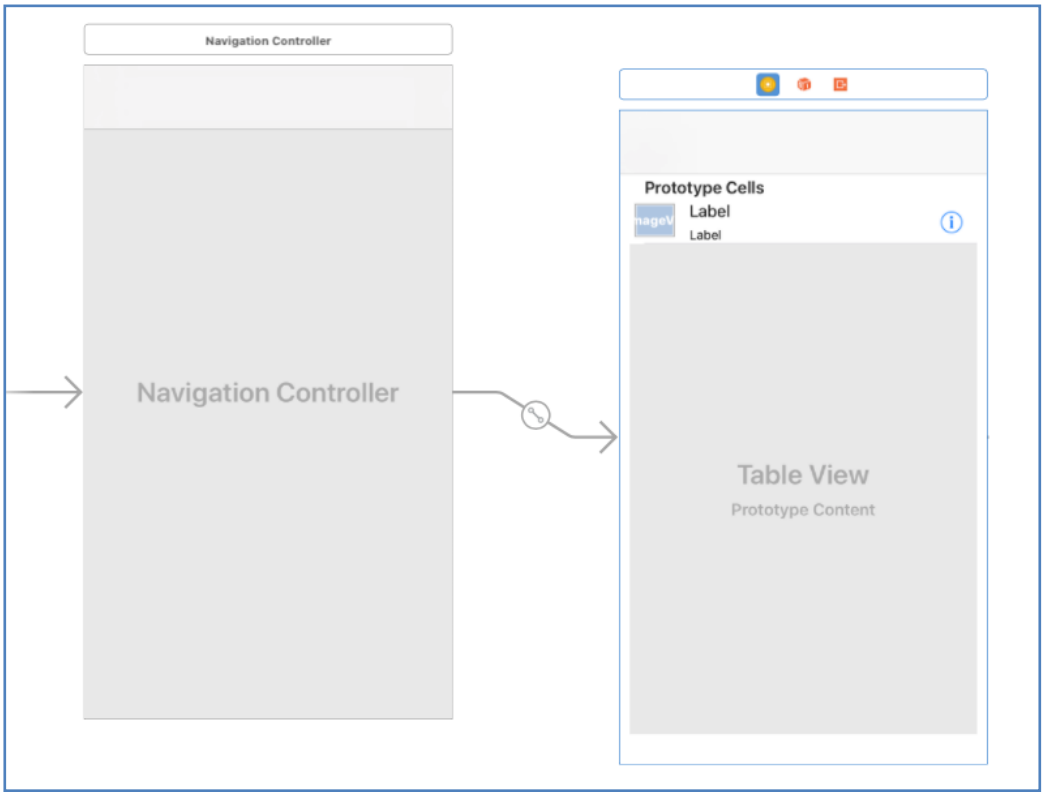


Abbildung 40 Anzeige des Navigations-Controllers mit dem TableView

Die Verbindung der beiden Controller wird automatisch erzeugt.

- Einfügen eines Detail-ViewController

Nun wird aus der UI-Liste ein ViewController in das Storyboard eingefügt. Danach wird im Projektbaum der Name des neuen ViewControllers auf “Detail” gesetzt.

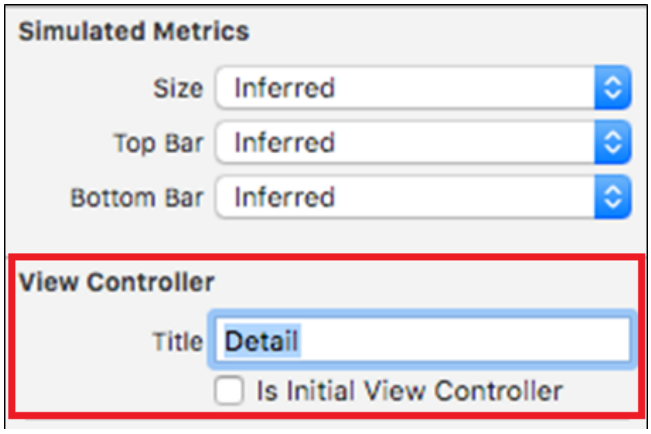


Abbildung 41 Benennen eines neuen ViewControllers

- Erstellen einer ViewController-Klasse

Der StandardViewController hat schon am Anfang eine zugeordnete Klasse “ViewController.swift”. Für neue ViewController muss jeweils eine Klasse erstelle werden (abgeleitet von UIViewController).

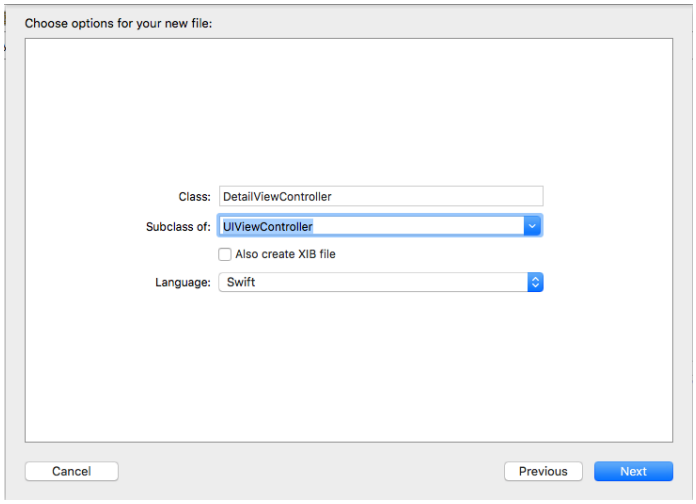


Abbildung 42 Eine Klasse für einen ViewController

Mit dieser Klasse können dann die Outlets der UI-Elemente verknüpft werden.

Dazu muss aber der ViewController mit der Klasse verbunden werden:

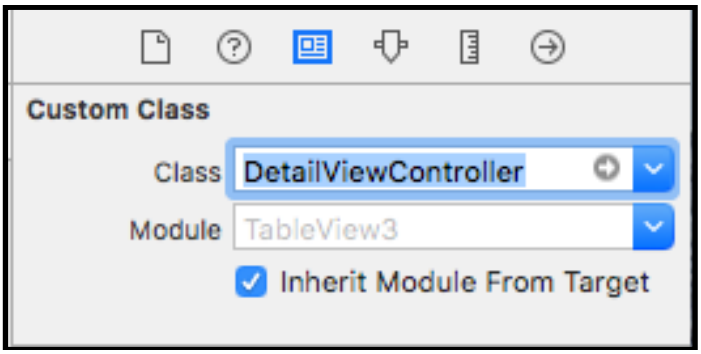


Abbildung 43 ViewController mit einer Klasse verbinden

- Verknüpfen der Zelle mit dem Controller (Drag&Drop)

Ablauf:

- ✓ Anklicken ViewCell
- ✓ Ctrl-Taste drücken
- ✓ Drag&Drop zum neuen ViewController
- ✓ Auswahl „Show“

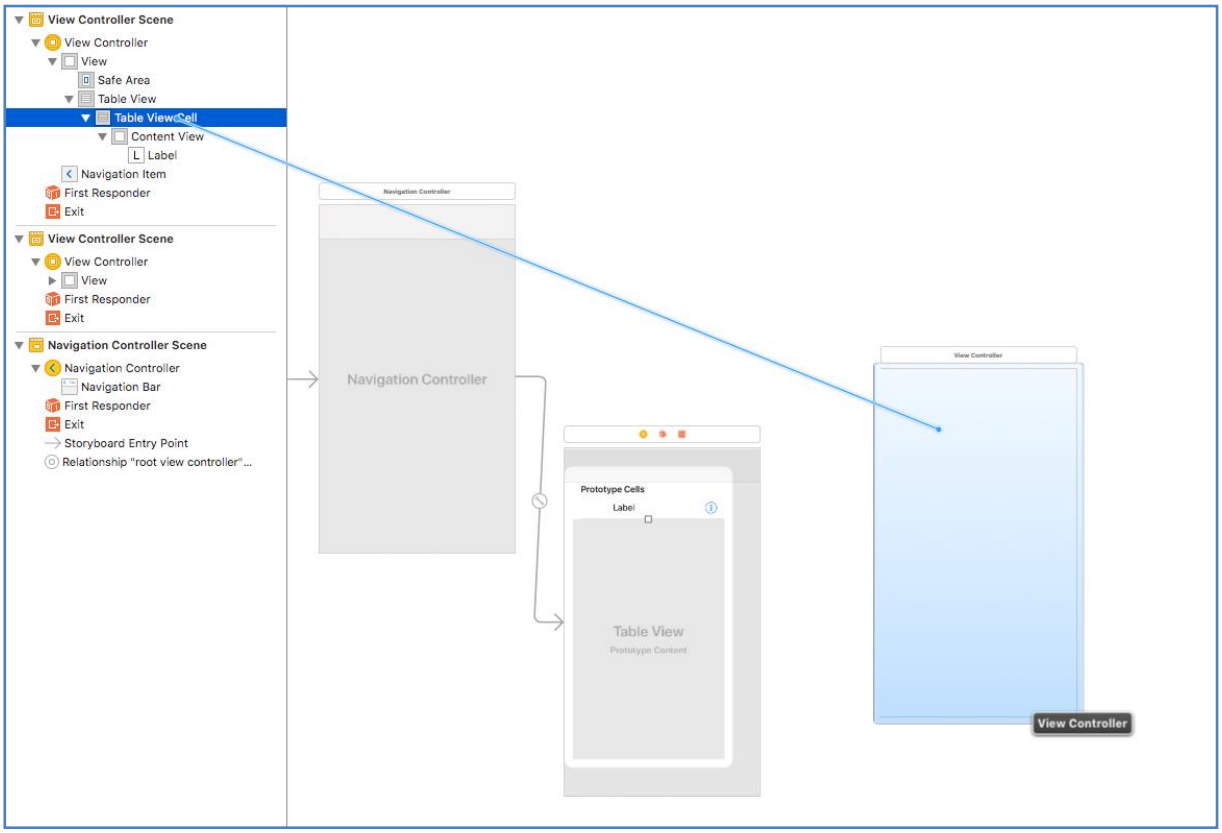


Abbildung 44 Verknüpfen der TableCell mit dem neuen DetailsViewController

Mit dem obigen Drag&Drop wird das “Ziel” verknüpft. Nach dem Loslassen des Mauskursors erscheint ein Dialog, in dem man den Eintrag “Show” auswählt.

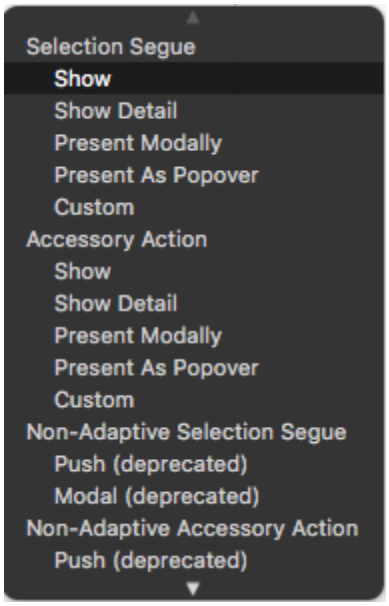


Abbildung 45 Drag & Drop zum DetailViewController

Im MainStoryboard erkennt man nun den Aufruf:

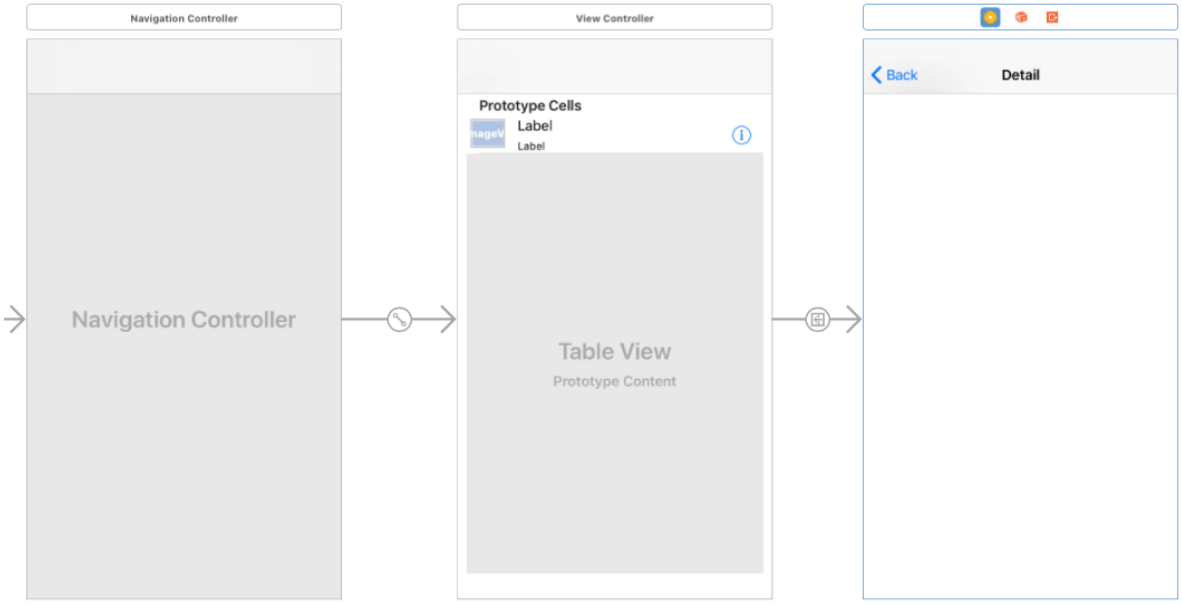


Abbildung 46 Verknüpfung der drei Controller.

In den letzten Schritten müssen die UI-Elemente eingetragen warden und das aktuellen Objekt (“Student”) muss übergeben werden.

- UI-Elemente eintragen:

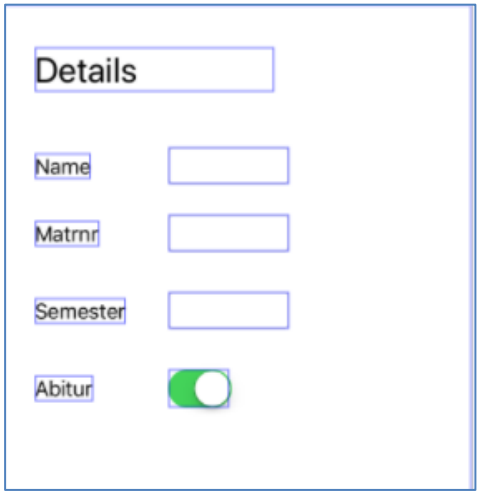


Abbildung 47 Anzeige der UI-Elemente des Details-ViewControllers

Quellcode des DetailsViewControllers

```
//
class DetailViewController: UIViewController {

    // public Data
    var studentData:Student!

    // Outlets
    @IBOutlet var tName: UITextField!
    @IBOutlet var tMatrnr: UITextField!
    @IBOutlet var tSemester: UITextField!
    @IBOutlet var switchAbitur: UISwitch!




    override func viewDidLoad() {
        super.viewDidLoad()
        tName.text = studentData.name
        tMatrnr.text = String(studentData.mnr)
        tSemester.text = String(studentData.semester)
    }

}
```

- Im letzten Schritt muss noch im HauptController der Datentransfer zum DetailsViewController eingetragen warden:

```
override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
   if let dest = segue.destination as? DetailViewController,
      let cell = sender as? MyCell,
```

```
        let indexPath = tableview.indexPath(for: cell) {

              dest.studentData = liste[ indexPath.row ]

    }  // if
}
```

Bemerkungen:

- ✓ Die erste Abfrage bezieht sich darauf, ob das Ziel, der DetailsViewController, korrekt erzeuigt wurde.
- ✓ Die zweite Abfrage dient der Unterscheidung, welcher Prototyp als Quelle dient. Es ist ja möglich, mehrere Prototypzellen zu definieren. Beim obigen Beispiel könnte man auch diese Abfrage löschen, da wir keinerlei Informationen aus den einzelnen Zellen holen.
- ✓ Die letzte Abfrage dient dazu, die aktuelle Zeile zu erfassen.

2.1.15 uiTableViewController

Arbeitet ähnlich der JTable. Man benötigt ein grafisches Element uiTableView, aber des Weiteren auch ein Protokoll mit vier Funktionen und zwei Delegate-Anweisungen.

Eigenschaften eines uiTableViews:

- Anzeige von Arrays in Spalten und Zeilen
- Jede Zeile ist eine Struktur bzw. ein Object
- Jede Spalte kann unterschiedliche grafische Elemente beinhalten.
- Die Zeilen können **gruppiert** werden.
- Die TableView holt nun durch eine Funktion JEDE Zelle.

Weitere UI-Elemente:

- UINavigationController, ein Pfeil am Ende der Zeile, zur weiteren Navigation

Unterschied zwischen TableView und TableViewController:

- Ein TableViewController nimmt den kompletten Raum des Views an.
- Man kann keine weiteren Elemente in die View einfügen.

Ablauf:

- Einfügen eines uiTableView
- Eine Referenz, Outlet, im Quellcode erzeugen
- Dass Protokoll „UITableViewDataSource“ eintragen.
    - class meinTableView: UIViewController, **UITableViewDataSource** {

Funktionen des Protokolls „**UITableViewDataSource“**

2.2 Ändern der Größe / Positionen der UI-Elemente

- UI-Elemente mittels Maus verschieben
- UI-Elemente mittels Maus verkleinern, vergrößern

2.3 Ändern der Eigenschaften eines UI-Elementes

Menü „Views“, Eintrag „Utilities“ kann man die Eigenschaften ändern.

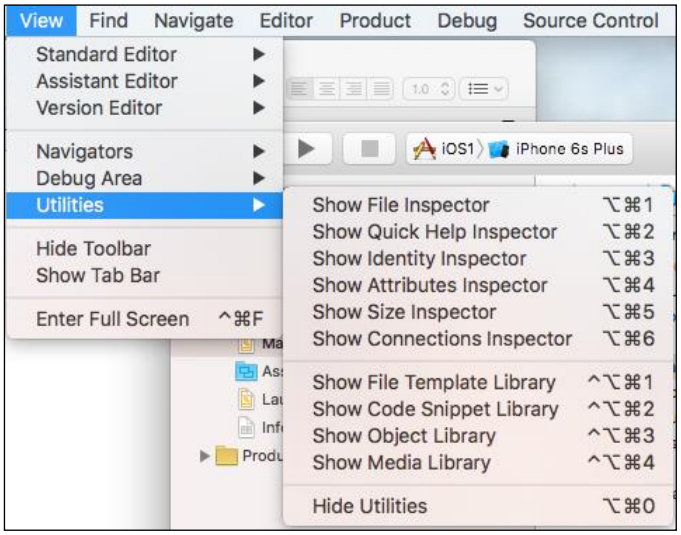


Abbildung 48 Properties

Wenn die „storyboard“ in der IDE eingetragen wurde, gibt es den Eintrag „Show Attributes Inspector“ nicht. Dann muss man rechts oben das blaue Symbol anklicken:

Abbildung 49 Property-Dialog anzeigen

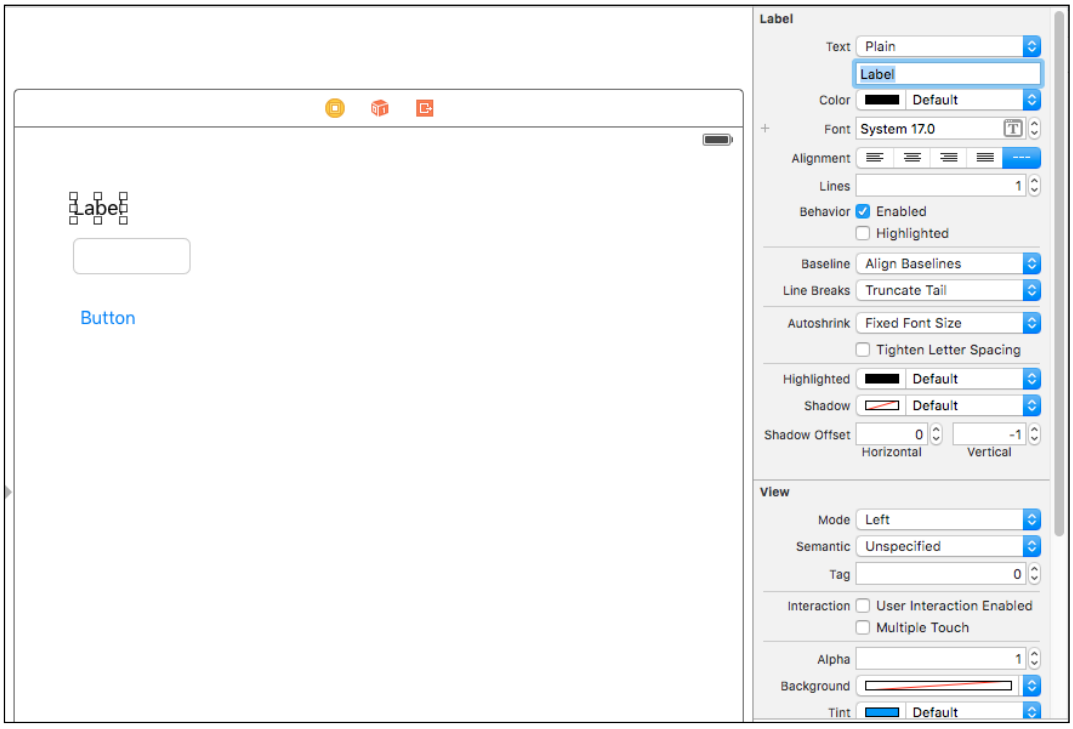


Abbildung 50 Property-Dialog

2.4 Zielgerät

Mit dem Menü “Product”, Eintrag “Destination” kann man das Zielgerät auswählen.

Abbildung 51 Zielgerät

Sinnvoll wäre iPhone 5. Es hat eine nicht zu hohe Auflösung.

3 UI-Variablen

Mit folgender Methode erhält man eine Referenz eines UI-Element in den Quellcode:

- View und Editor müssen sichtbar sein.
- Ctrl-Taste drücken.
- Linke Maustaste über dem UI-Element drücken und zum Editor verschieben.
- Oben, nach class { „einfügen“.

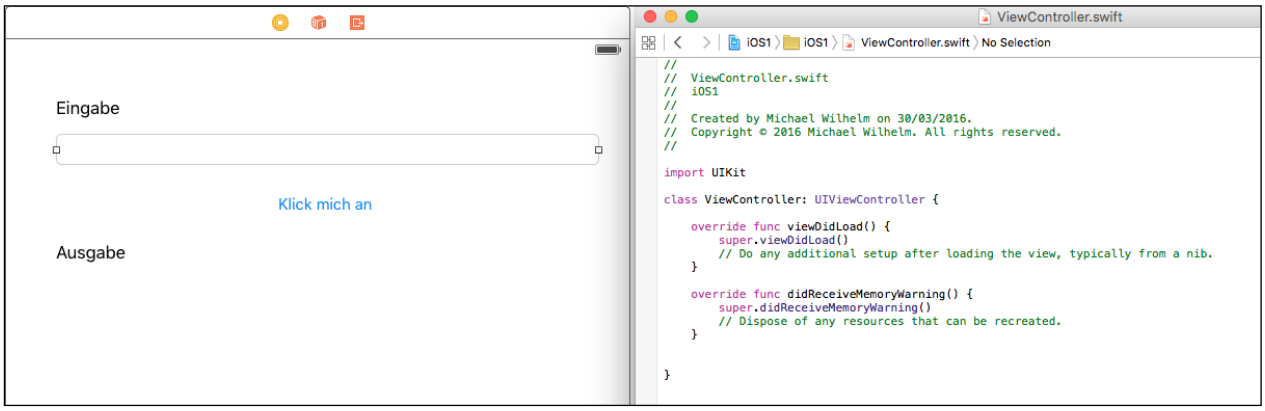


Abbildung 52 UI-Variable eines TextField eintragen

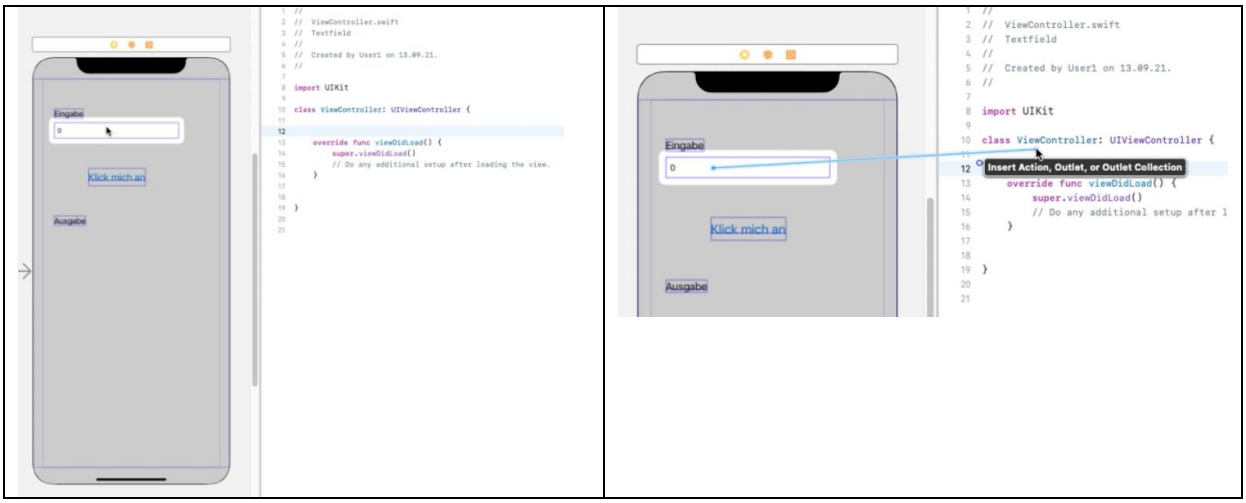


Abbildung 53 UI-Variable eines TextField eintragen

Nach dem Drag&Drop erscheint folgender Dialog:

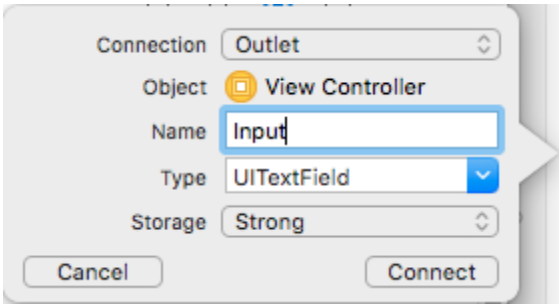


Abbildung 54 Outlet, Referenz eintragen

4 Action-Methoden (onClick-Events)

Bei einem Button benötigt man keine Referenz, Outlet, sondern ein Action-Event. Die Methode ist aber weitgehend identisch. Nach dem Drag&Drop erscheint das bekannte Fenster:

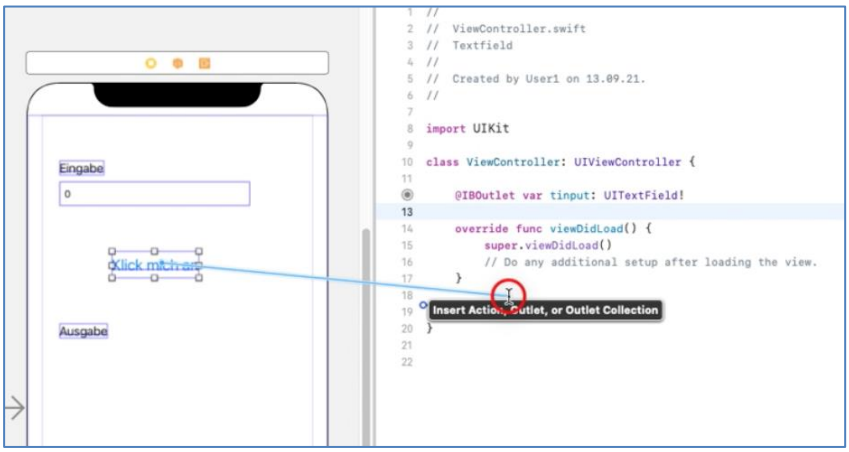


Abbildung 55 Drag&Drop für ein ButtonClick-Event

Abbildung 56 Button-Event

Beispielcode:

```
ViewController.swift: class ViewController: UIViewController {
    // Attribute
    var nr:Int32=0
    @IBOutlet var bnAction: UITextField!
    @IBOutlet var tInput: UITextField!
    @IBOutlet var lblOutput: UILabel!

    override func viewDidLoad() {       }

    @IBAction func bnActionClick(sender: AnyObject) {
        nr++
        let input:String=tInput.text!
        self.lblOutput.text=input+" "+String(nr)
    }
}
```

5 Layout-Erstellen

- Jedes Element kann mit Abständen nach oben, unten, links und rechts positioniert werden.
- Markierte Elemente können gleiche Breiten, Höhen zugeordnet werden.
- Ein GridbagLayout gibt es leider nicht.
- Alternativen:
  - Horizontal StackView
  - Vertical StackView

Wichtig:

- Es gibt zwei Abläufe um ein Layout zu erstellen.

- **Die UI-Elemente werden nacheinander eingefügt**
  - In dieser Version kann man jedes Mal den Menüeintrag „UpdateFrames“ aufrufen.

- **Die UI-Elemente sind vorab alle im ViewController eingefügt**
  - In dieser Version darf man den Menüeintrag „UpdateFrames“ **erst am Ende** aufrufen, da die UI-Elemente, die noch kein Constraints haben, oben im View liegen.

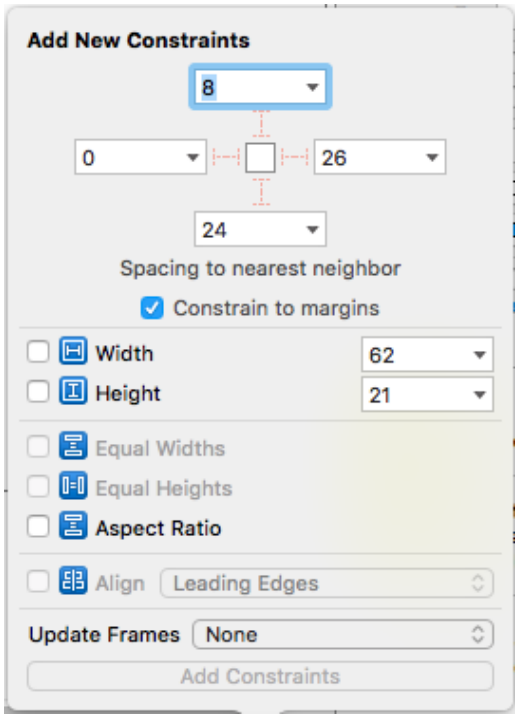


Abbildung 57 Constraints-Dialog

Ablauf:

- Projekt erstellen.
- Ein erstes ui-Element eintragen und positionieren.
- Anklicken des ersten Elementes
- Anklicken des Layout-Schalters 
- Die aktuelle Position wird mit den vier Abständen angezeigt

- Top
  - Abstand zum View
  - Abstand zum Rahmen
  - Abstand zum oberen ui-Element
- Left
  - Abstand zum View (linker Rand)
  - Abstand zum Rahmen (linker Rand)
  - Abstand zum linke ui-Element
- Right
  - Abstand zum View (rechter Rand)
  - Abstand zum Rahmen (rechter Rand)
  - Abstand zum rechten ui-Element

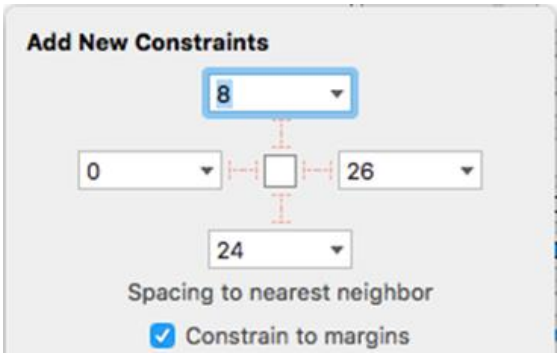


Abbildung 58 Constraints eintragen

- Nun alle gestrichelten roten Linien (top, left, right, bottom) anklicken, die mit ein Layout-Constraint haben sollen.
- Nun die Abstände eingeben.
- Bei gleicher Höhe für ein uiLabel und ein uiTextfield gilt:
  - uiLabel: 10 Pixel
  - uiTextfield: 6 Pixel
- Schalter „Add x Constraints“ betätigen

- Im View wird dann die errechnete Position mit roten Linien angezeigt

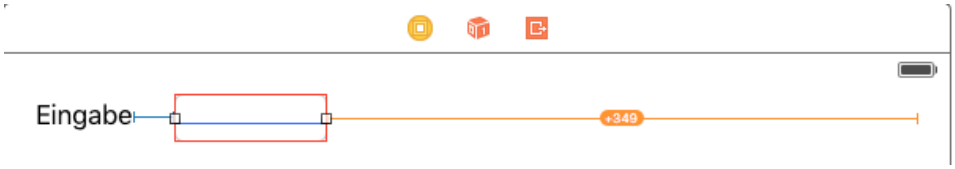


Abbildung 59 Vorschau eines Constraints

- Schalter „Update Frames“ betätigen 
- Dann den oberen Menüeintrag „Update Frames“ aufrufen.

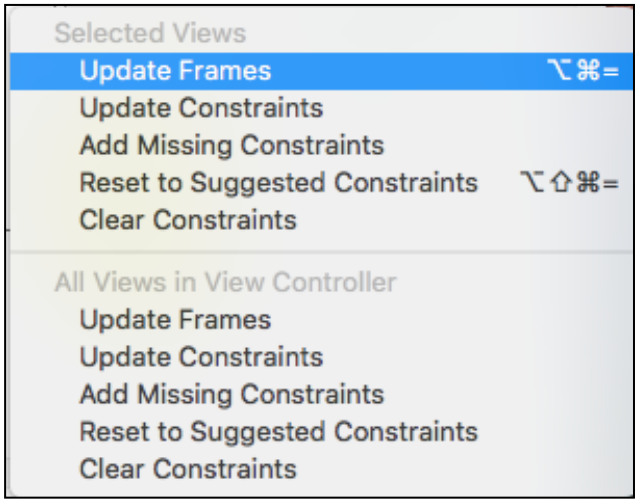


Abbildung 60 Update des Frames

5.1 Aligments Constraints

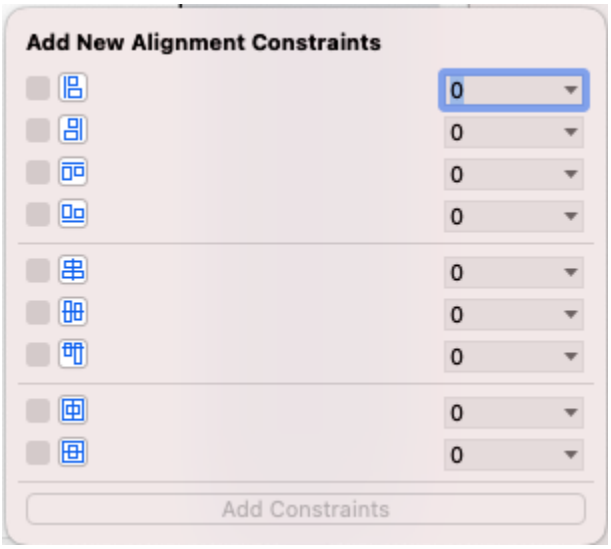


Abbildung 61 Aligments Constraints

5.1 Beispiele

5.1.1 Layout 1

- Label-Feld
- TextField darunter

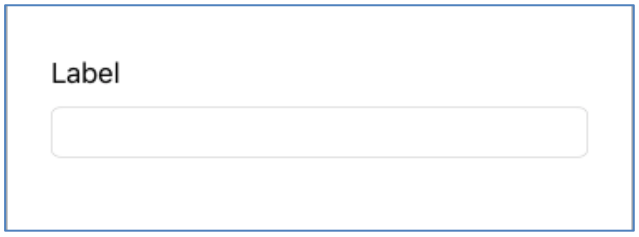


Abbildung 62 Layout in xcode

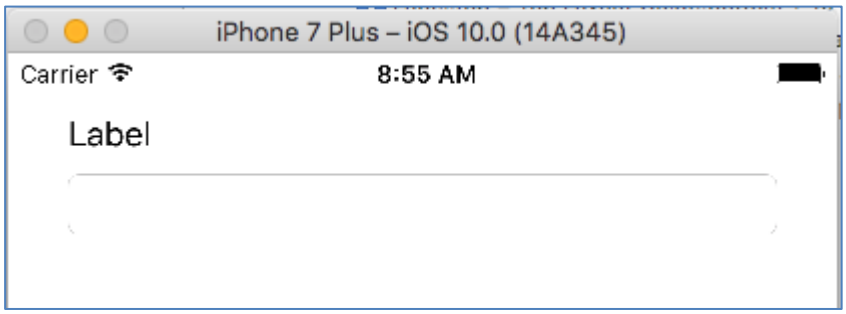


Abbildung 63 1. Layoutbeispiel: Label mit Textzeile

Abbildung 64 1. Layoutbeispiel: Label mit Textzeile (Querformat)

Positionsdaten:

- **Label-Feld**
  - Top: 10 Pixel
  - Left : 10 Pixel
- **TextField**
  - Top: 10 Pixel
  - Left : 10 Pixel
  - Right: 10 Pixel

Im Projektbaum werden die Contraints angezeigt:

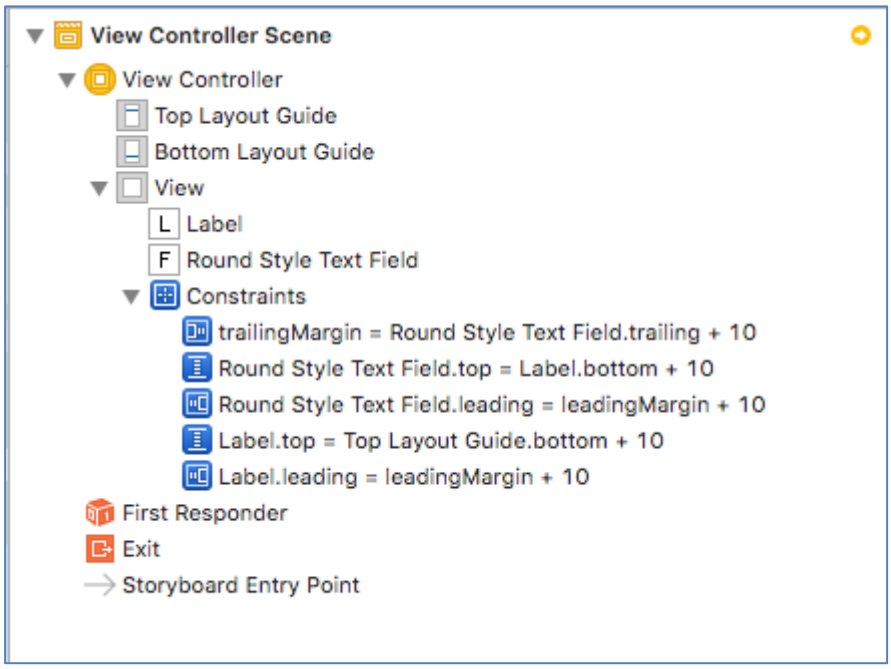


Abbildung 65 Anzeige der Constraints des 1. Layoutbeispiels

Die letzten Einträge beziehen sich auf die Label-Begrenzungen (top und left jeweils 10 Pixel). Der erste und der dritte Eintrag betreffen die Grenzen links und rechts. Im zweiten Eintrag wird der Der Abstand zum Labelfeld definiert.

5.1.2 Layout 2

- Label-Feld
- TextField **daneben**

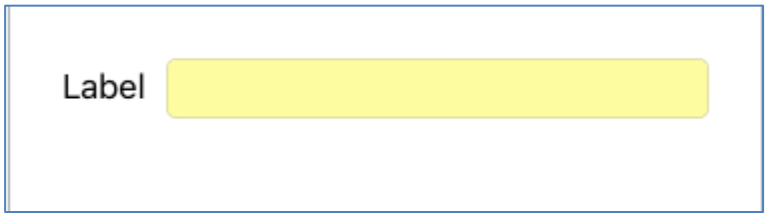


Abbildung 66 Layout in xcode

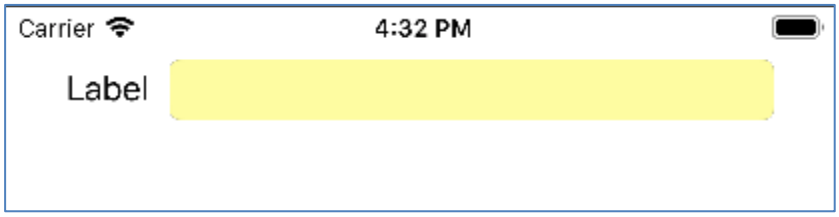


Abbildung 67 2. Layoutbeispiel: Label mit Textzeile

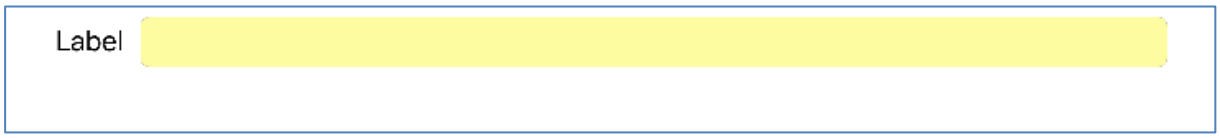


Abbildung 68 2. Layoutbeispiel: Label mit Textzeile (Querformat)

Positionsdaten:

- **Label-Feld**
  - Top: 10 Pixel
  - Left : 10 Pixel
- **TextField**
  - Top: **6 Pixel**
  - Left : 10 Pixel
  - Right: 10 Pixel
- Im View wird dann die errechnete Position mit roten Linien angezeigt

Im Projektbaum werden die Contraints angezeigt:

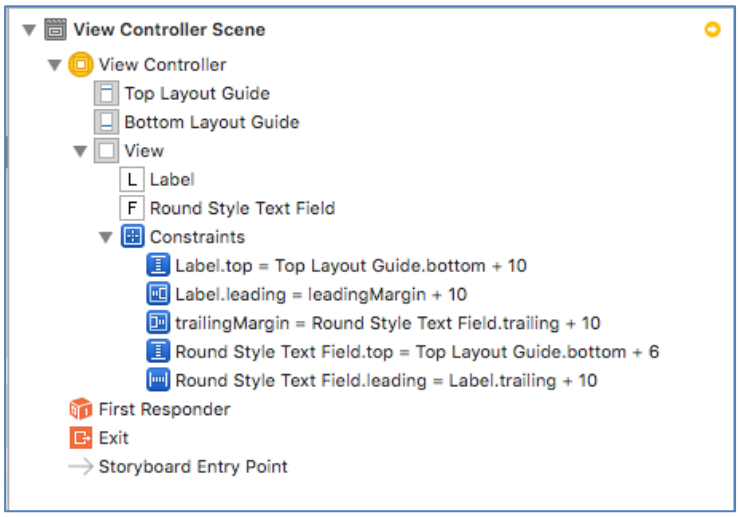


Abbildung 69 Anzeige der Constraints des 2. Layoutbeispiels

5.1.3 Layout 3

- Zwei Labels
- Zwei TextFields
- Manuell mit den Layout-Constraints

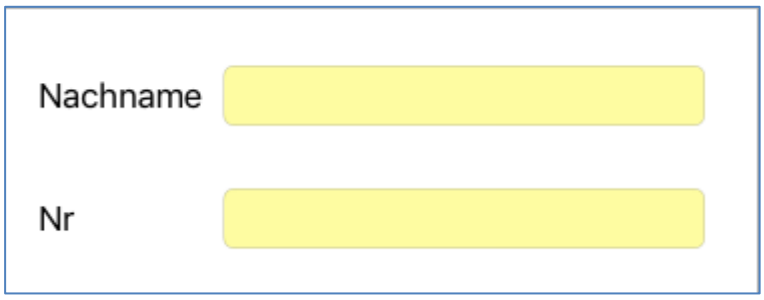


Abbildung 70 Layout in xcode

Damit die beiden Label gleichgroß dargestellt werden, müssen beide markiert werden. Danach werden die Breiten gleichgesetzt (Beispiel siehe unten).

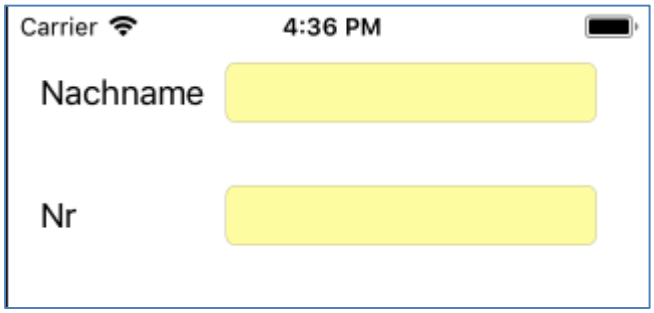


Abbildung 71 3. Layoutbeispiel: Label mit Textzeile

Abbildung 72 3. Layoutbeispiel: Label mit Textzeile (Querformat)

Positionsdaten:

- **Label-Feld Nachname**
  - Top: 10 Pixel
  - Left : 10 Pixel
  - Right: 10 Pixel
- **TextField Nachname**
  - Top: **6 Pixel**

- Left : 10 Pixel
- Right: 10 Pixel

- **Label-Feld UNr**
  - Top: 40 Pixel
  - Left : 10 Pixel
  - Right: 10 Pixel
- **TextField UNr**
  - Top: **36 Pixel**
  - Left : 10 Pixel
  - Right: 10 Pixel
- Im letzten Schritt werden beide Label-Elemente angeklickt.
  - Markieren
    - 1. Label anklicken
    - Ctrl-Taste drücken
    - 2. Label anklicken
  - Aufruf des Contraints-Dialog
    - Anklicken der CheckBox „Equal Widths“

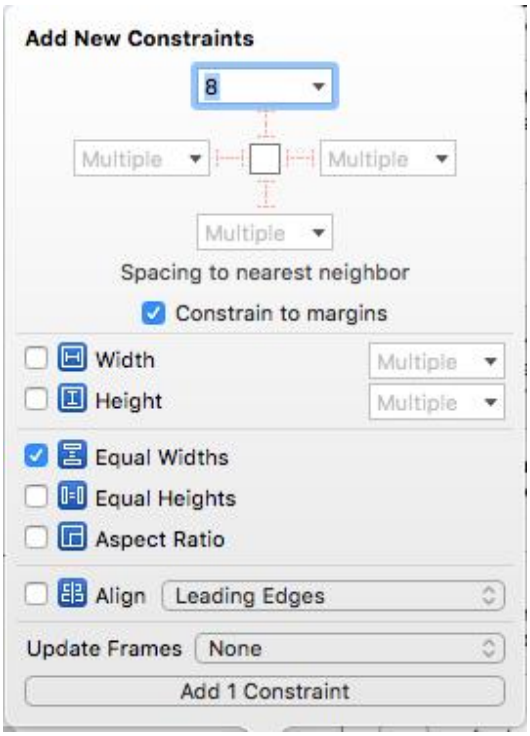


Abbildung 73 Constraint "Equal Widths"

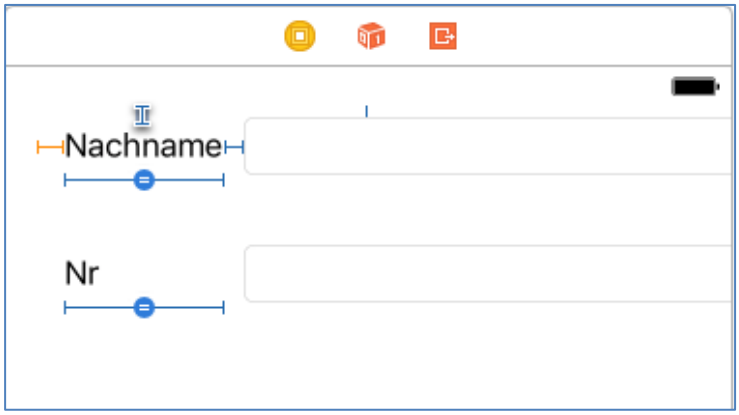


Abbildung 74 Anzeige der Constraints der Nachnamen und Nr im ViewController

Im Projektbaum werden die Contraints angezeigt:

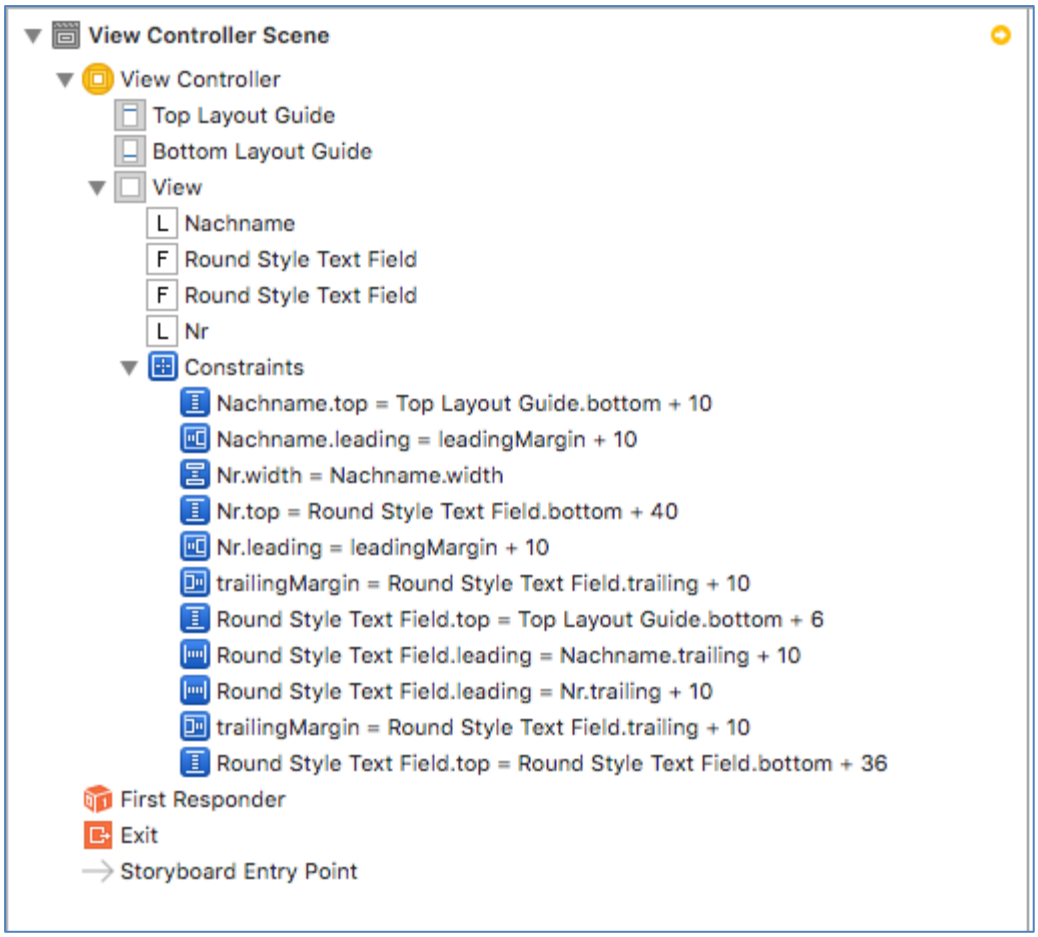


Abbildung 75 Anzeige der Constraints des 3. Layoutbeispiels

5.1.4 Layout 4

- Zwei Labels
- Zwei TextFields
- Manuell mit den Layout-Constraints, Verhältnis 1:3

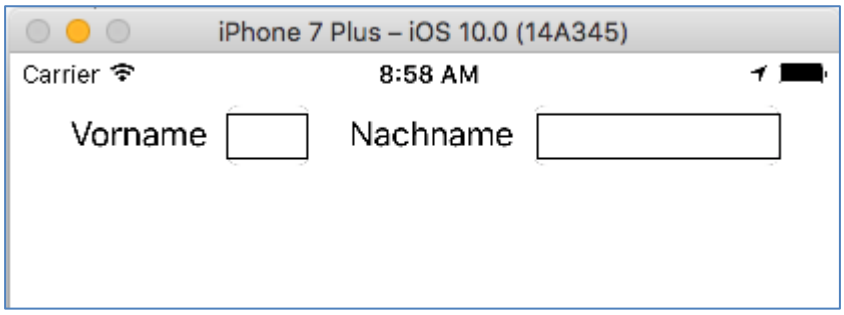


Abbildung 76 4. Layoutbeispiel: Label mit Textzeile

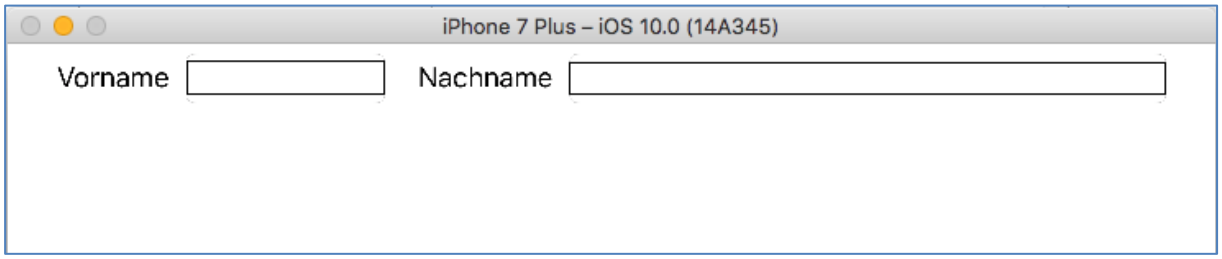


Abbildung 77 4. Layoutbeispiel: Label mit Textzeile (Querformat)

Positionsdaten:

- **Label-Feld Vorname**
  - Top: 10 Pixel
  - Left : 10 Pixel
  - Right: 10 Pixel
- **TextField Vorname**
  - Top: **6 Pixel**
  - Left : 10 Pixel
  - Right: 10 Pixel
- **Label-Feld Nachname**
  - Top: 20 Pixel
  - Left : 10 Pixel
  - Right: 10 Pixel
- **TextField Nachname**
  - Top: **6 Pixel**
  - Left : 10 Pixel
  - Right: 10 Pixel
- Im letzten Schritt werden beide Textfield-Elemente angeklickt.
  - Markieren
    - 1. Textfield anklicken
    - Ctrl-Taste drücken
    - 2. Textfield anklicken
  - Aufruf des Contraints-Dialog

- Anklicken der CheckBox „Equal Widths“

Label Vorname

1) Projekt erstellen, Single View
2) Öffnen des Mainboards (View)
3) uiLabel per Drag&Drop hinzufügen

4) Anklicken des Layout-Schalters 
5) Den oberen und linken Strich anklicken (siehe untere Abbildung)

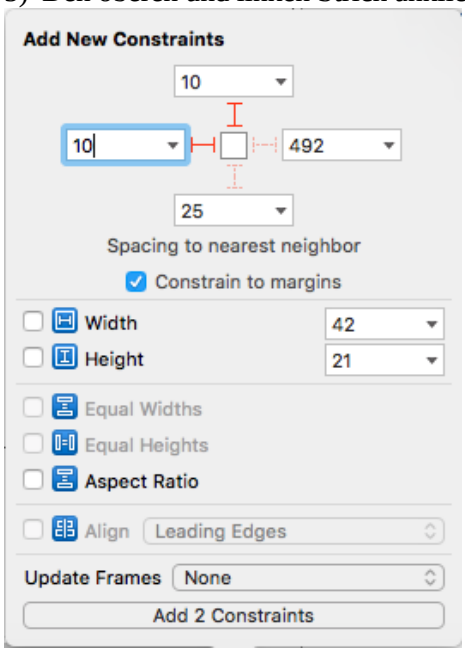


Abbildung 78 Contraints-Dialog

Nun wird das Label immer nach link/oben ausgerichtet.

6) Schalter „Add x Constraints“ betätigen 

7) Im View wird dann die errechnete Position mit roten Linien angezeigt

8) Schalter „Update Frames“ betätigen 
   Dann den Menüeintrag „Update Frames“ aufrufen

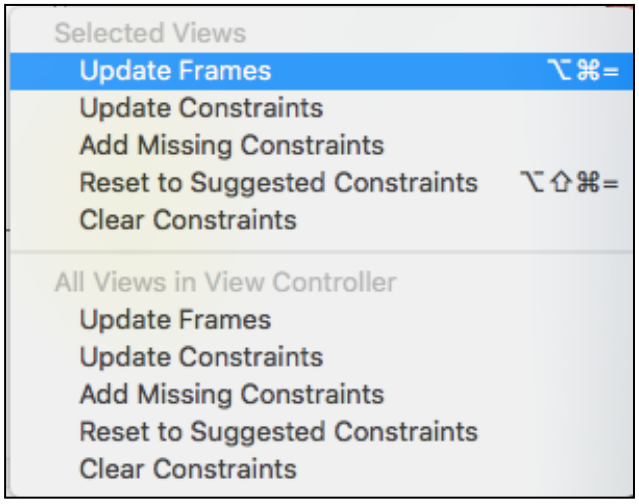


Abbildung 79 Update-Frames

Damit müsste das Label immer korrekt ausgerichtet sein.
Bitte testen (vertikal und horizontal)

uiTextField Vorname

1) uiTextField per Drag&Drop hinzufügen
2) Das Textfeld rechts neben dem Label „Vorname“ positionieren.
   Die Höhe sollte 6 Pixel betragen, bitte die rote Linie beachten
3) Das Label „Nachname“ neben dem Textfeld „Vorname“ platzieren.
   Das ist wichtig, damit die rechte Begrenzung definiert werden kann.

4) Anklicken des Layout-Schalters
5) Den oberen, linken und rechten Strich anklicken (siehe untere Abbildung)
   10 Pixel zum linken Label
   6 Pixel zum oberen Rand
   10 Pixel zum rechten Label

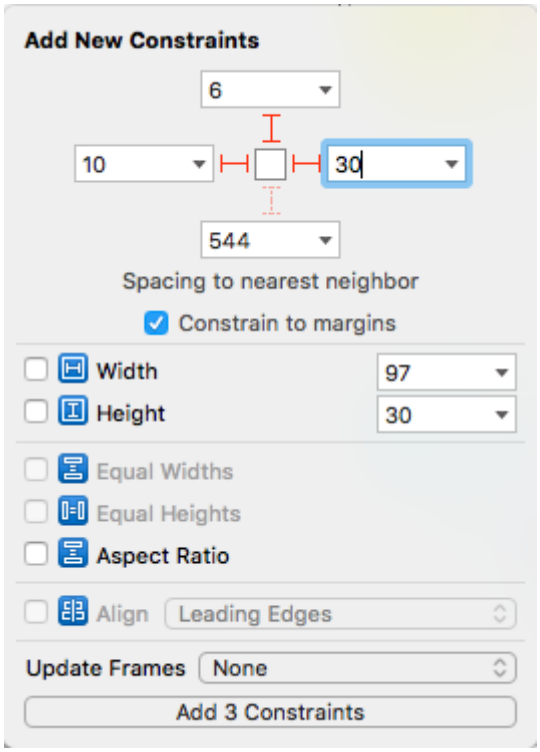


Abbildung 80 Ausrichten des Textfields nach dem linken und rechten Label

Nun wird das Textfeld „Vorname“ immer nach link/oben und rechts ausgerichtet.

6) Schalter „Add x Constraints“ betätigen 

7) Im View wird dann die errechnete Position mit roten Linien angezeigt
8) Wichtig:

   Nun darf man NICHT den Schalter „Update Frames“ betätigen 
   Das Problem ist das zweite Label, welches noch nicht „verdrahtet“ ist.

Label Nachname

1) Das Textfeld „Nachname“ rechts neben dem Label „Nachname“ positionieren.
   Die Höhe sollte 6 Pixel betragen, bitte die rote Linie beachten
   Das ist wichtig, damit die rechte Begrenzung des zweiten Labels definiert werden kann.

2) Anklicken des Layout-Schalters 
3) Den oberen, linken und rechten Strich anklicken
   30 Pixel zum linken Textfeld „Vorname“
   10 Pixel zum oberen Rand
   10 Pixel zum rechten Textfeld „Nachname“

4) Schalter „Add x Constraints“ betätigen 

5) Im View wird dann die errechnete Position mit roten Linien angezeigt

uiTextField Vorname

1) Anklicken des Layout-Schalters
2) Den oberen, linken und rechten Strich anklicken (siehe untere Abbildung)
3) Den oberen, linken und rechten Strich anklicken
   10 Pixel zum linken Label „Nachname“
   6 Pixel zum oberen Rand
   10 Pixel zum rechten Rand
4) Schalter „Add x Constraints“ betätigen
5) Im View wird dann die errechnete Position mit roten Linien angezeigt
6) Schalter „Update Frames“ betätigen
   Dann den Menüeintrag für alle UI-Elemente „Update Frames“ aufrufen und testen.

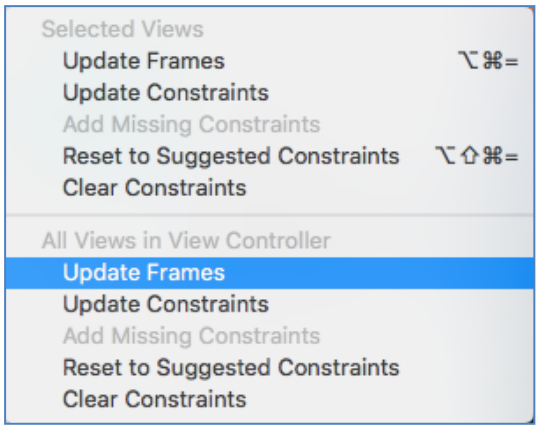


Abbildung 81 Alle ui-Elemente aktualisieren

Aktuelles Ergebnis:

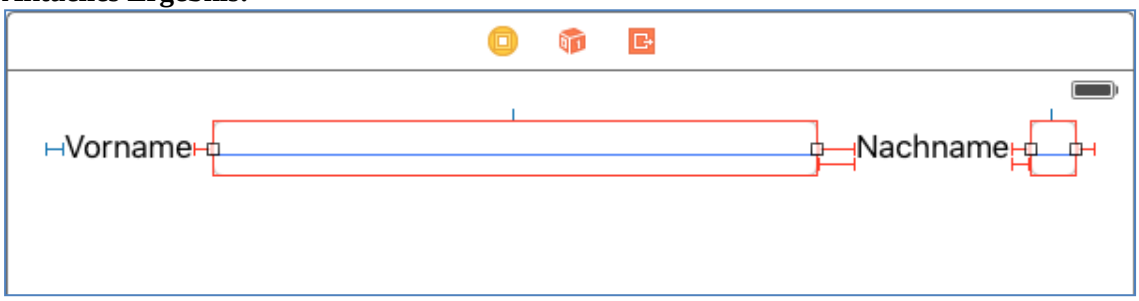


Abbildung 82 Die Textfelder haben nicht die gleiche Größe

Abhilfe
1) Anklicken der beiden Textfelder

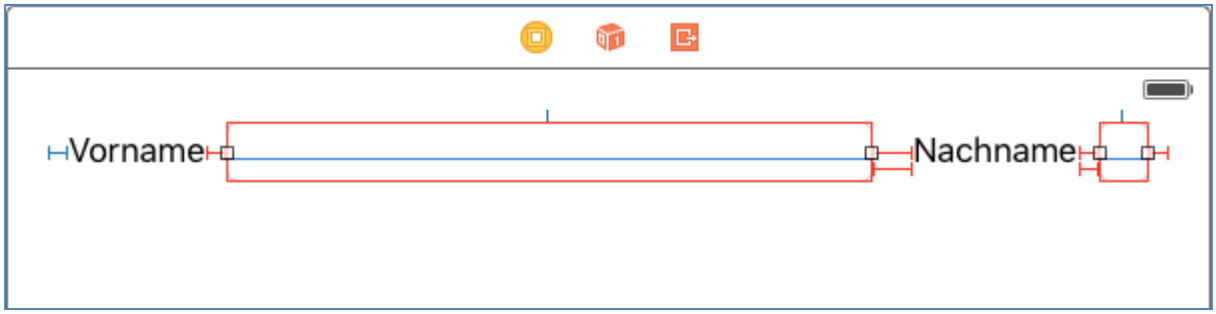


Abbildung 83 Markieren zweier Textfelder

2) Anklicken des Layout-Schalters 
3) Man sieht, dass mehrere Elemente angeklickt sind (Multiple)

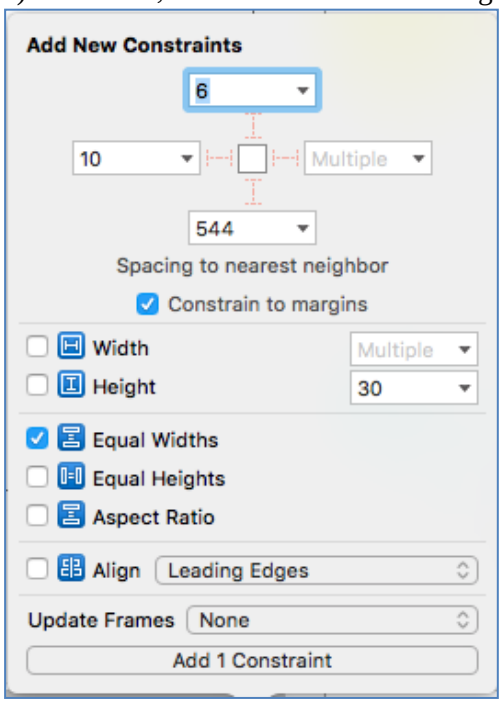


Abbildung 84 Gleiche Breite zweier Textfelder im Layout

4) Anklicken der Eigenschaft „Equals Widths“
5) Schalter „Add 1 Constraints“ anklicken
6) Update **aller** Views

Aktueller Stand im View-Designer:

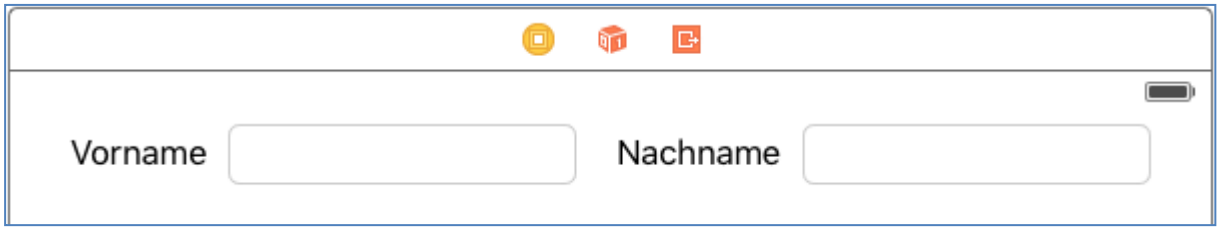


Abbildung 85 Das Ziel ist erreicht

Ergebnis im Testmodus:

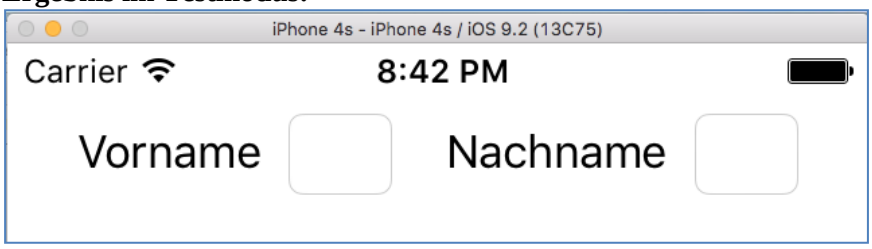


Abbildung 86 Vertikaler Test

Abbildung 87 Horizontaler Test

Nun kann man die Breiten unterschiedlich definieren:

1) Markieren beider Textfelder
2) Doppelklick auf ein blaues Gleichheitszeichen

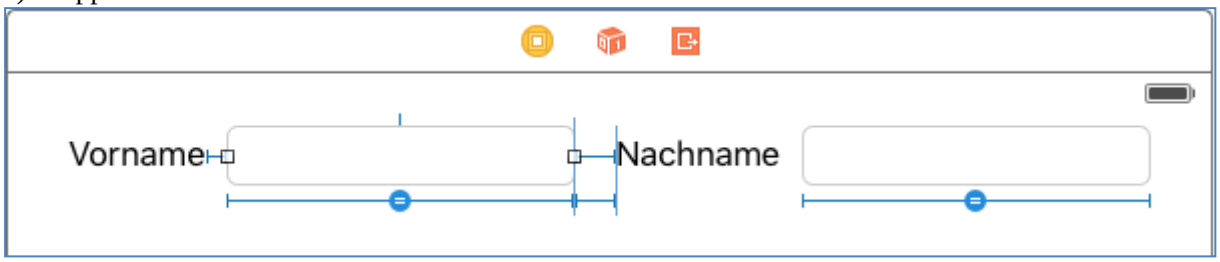


Abbildung 88 Breitenverhältnis zweier Textfields

3) Ein Doppelklick auf eine blaue Linie kann man das Verhältnis der beiden Textfelder definieren.

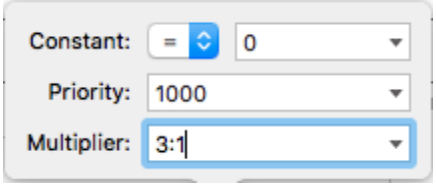


Abbildung 89 Anklicken des rechten Textfeldes, Beenden mit der Return-Taste

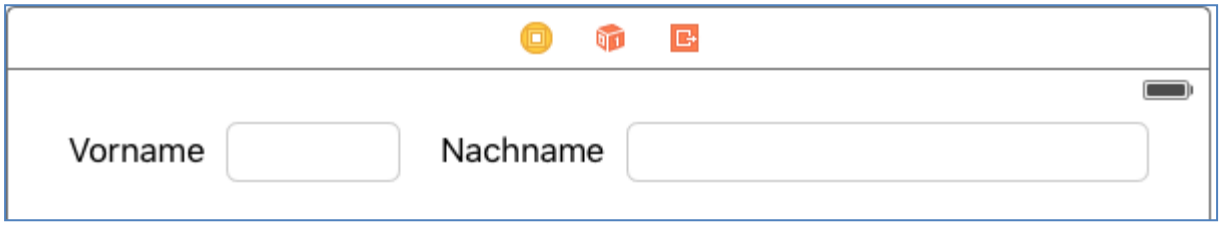


Abbildung 90 Ergebnis des Verhältnisses 1:3

Im Projektbaum werden die Contraints angezeigt:

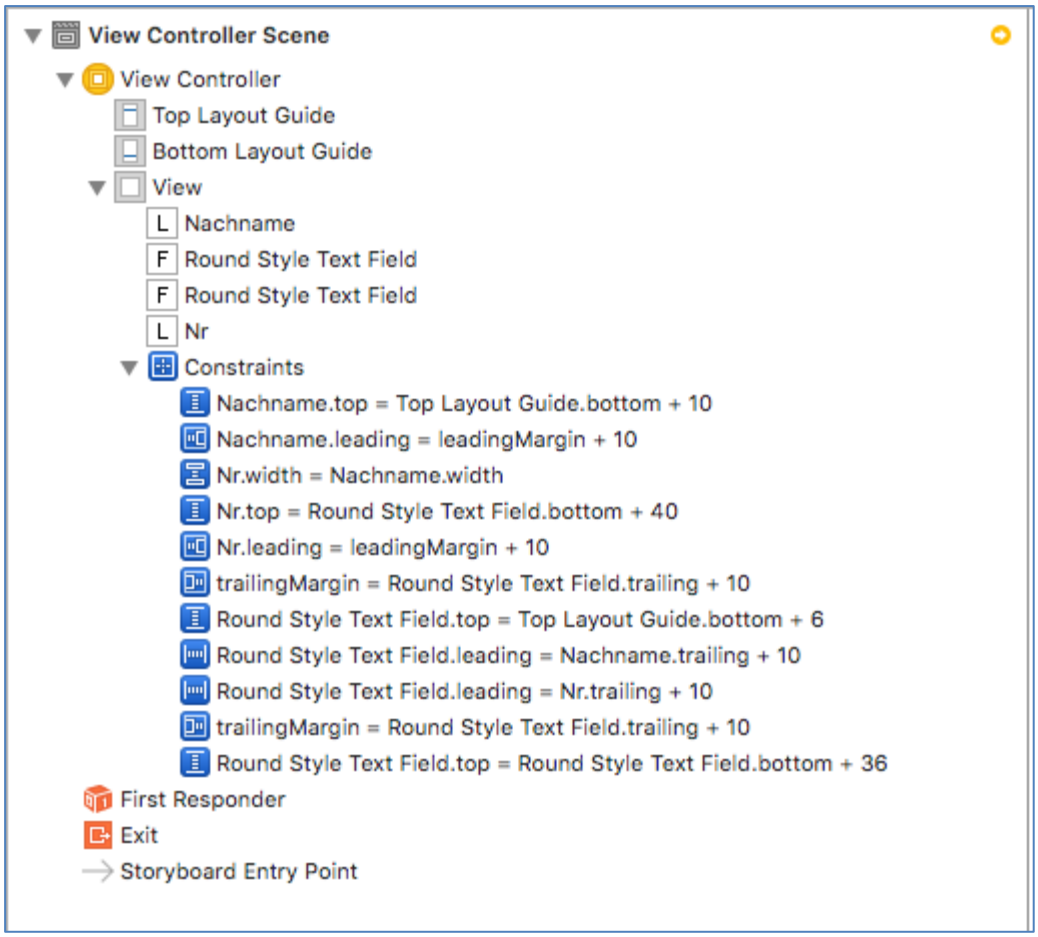


Abbildung 91 Anzeige der Constraints des 4. Layoutbeispiels

5.1.5 Layout 5

- Zwei Schalter, die nebeneinander mit der gleichen Breite liegen
- Manuell mit den Layout-Constraints

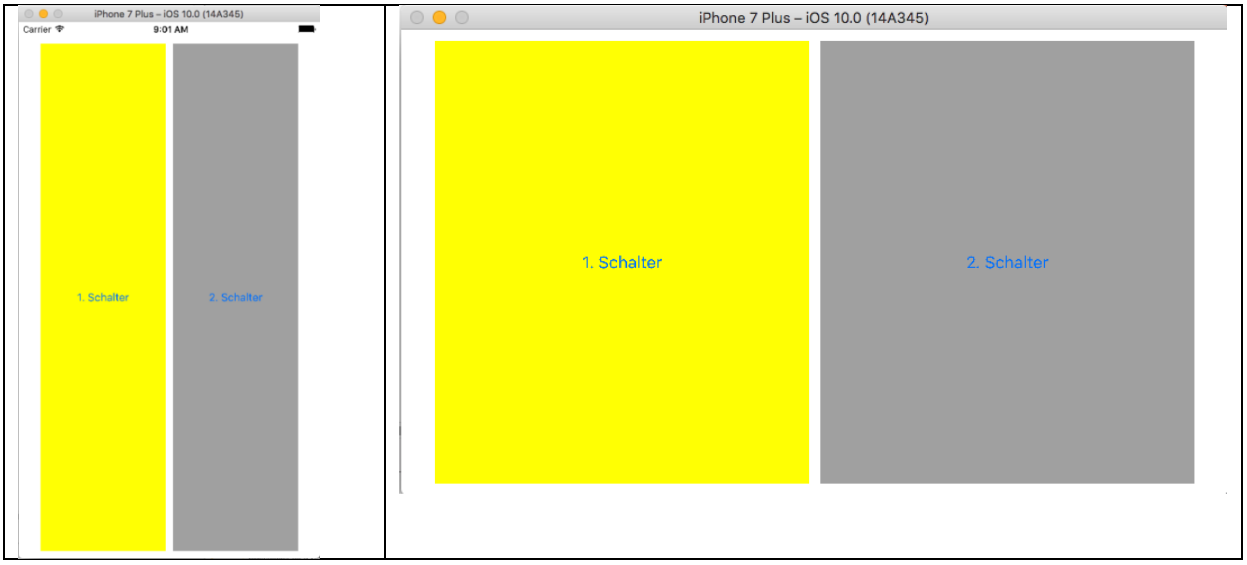
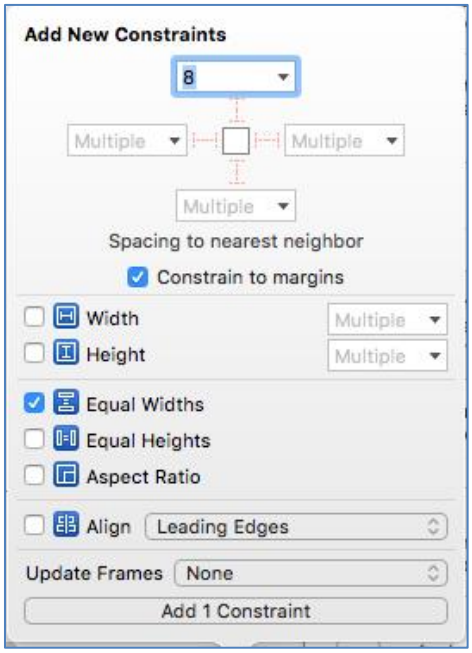


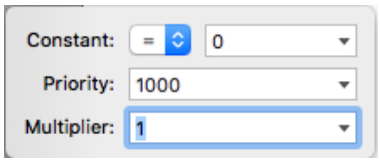
Abbildung 92 5. Layoutbeispiel: Label mit Textzeile

Positionsdaten:

- **1. Schalter**
  - Top: 10 Pixel
  - Left : 10 Pixel
  - Right: 10 Pixel
  - Bottom: 10 Pixel
- **2. Schalter**
  - Top: 10 Pixel
  - Left : 10 Pixel
  - Right: 10 Pixel
  - Bottom: 10 Pixel
- Im letzten Schritt werden beide Schalter angeklickt
  - Markieren (im Grafikmodus **ODER** im Elementtree.
    - 1. Schalter anklicken
    - Cmd-Taste drücken
    - 2. Schalter anklicken
  - Aufruf des Contraints-Dialog
    - Anklicken der CheckBox „Equal Widths“



- Im letzten Schritt wird das Breitenverhältnis eingetragen:
  - Anklicken eines Gleichheitszeichen
  - Eintragen des Breitenverhältnisses im Textfield „Multiplier“.



Im Projektbaum werden die Contraints angezeigt:

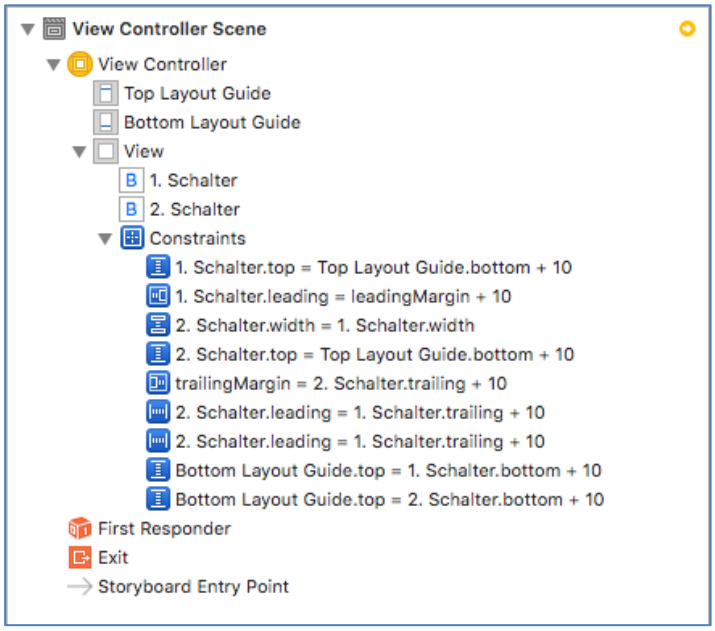


Abbildung 93 Anzeige der Constraints des 5. Layoutbeispiels

5.1.6 StackView

Neben den klassischen Layouts gibt es ab Version 7,0 einen StackView. Apple hat wohl eingesehen, dass Anfänger und Fortgeschrittene eine kleine Hilfe benötigen.

- Ein StackView kann man sich wie ein Regal vorstellen. In jedes Fach kann jedes UI-Element, also auch ein StackView, eingetragen werden.
- Die Zwischenräume können mit „Leerzeichen“ gefüllt werden. Dann aber nur gleichmäßig.
- Das Eintragen geschieht am besten im Projektbaum. Ein blauer Strich zeigt die Hierarchie.
- Separate Breite/Höhen sind auch im StackView möglich.

Abbildung 94 Horizontal StackView, ab Version 7

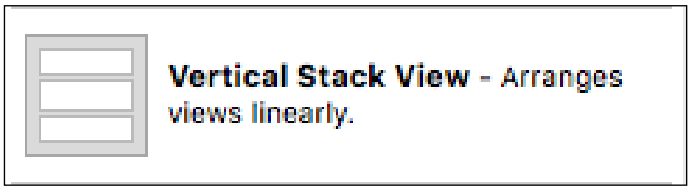


Abbildung 95 Vertikaler StackView, ab Version 7

5.1.7 Layout 6 (Beispiel 3)

- Zwei Labels
- Zwei TextFields
- **Diesmal mit zwei StackViews**

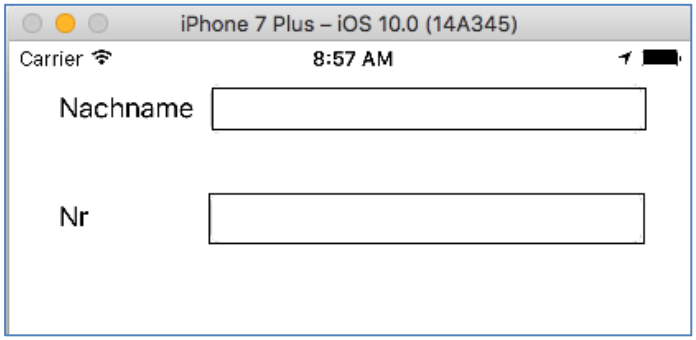


Abbildung 96 3. Layoutbeispiel: Label mit Textzeile

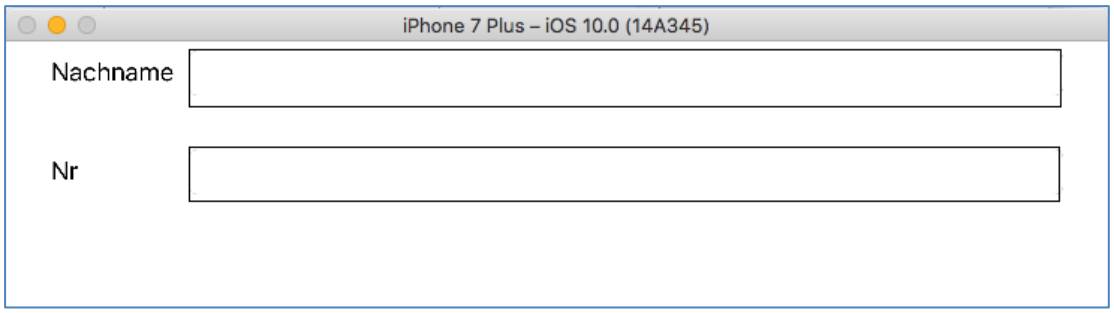
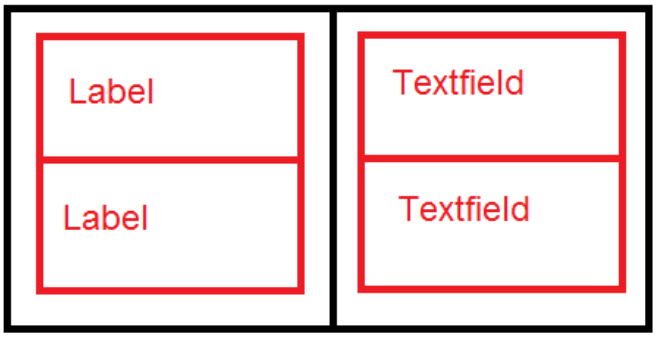


Abbildung 97 3. Layoutbeispiel: Label mit Textzeile (Querformat)

Die Lösung des Problems beinhaltet drei Stackviews. Damit sind alle Label immer gleich lang!

- Zwei Labels
- Zwei TextFields
- **Drei StackView‘s**



Die Hauptbasis ist ein horizontaler StackView (schwarz), der in seinen „Fächern“ jeweils ein **vertikales** StackView beinhalten. Damit sind beide Labels automatisch gleich groß.

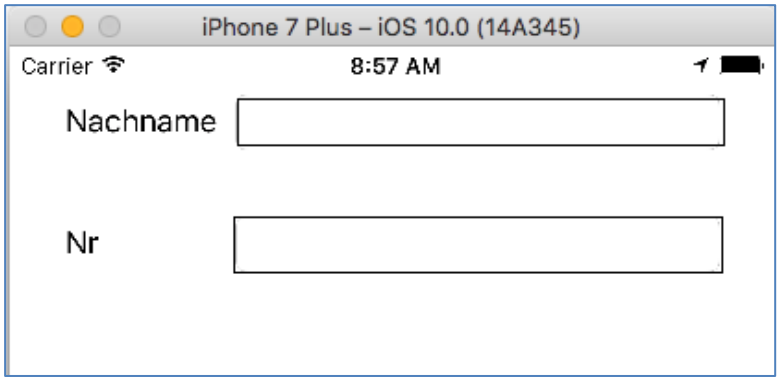


Abbildung 98 7. Layoutbeispiel: Zwei Label mit zwei Textzeilen

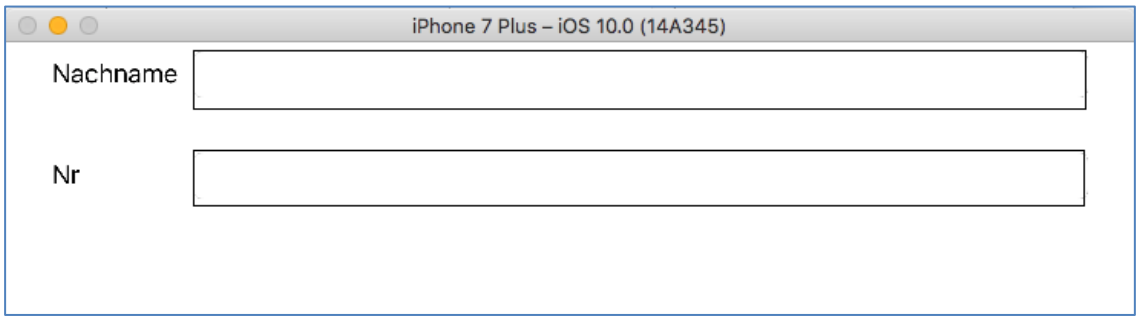


Abbildung 99 7. Layoutbeispiel: Zwei Label mit zwei Textzeilen (Querformat)

Vorgehensweise:

Positionsdaten:

- **1. StackView**
  - Top: 10 Pixel
  - Left : 10 Pixel
  - Right: 10 Pixel
- **2. StackView**
  - Eintragen in den ersten StackView
- **3. StackView**
  - Eintragen in den ersten StackView
- 1. Label “Nachname” in den 2. StackView positionieren
- 2. Label “Nachname” in den 2. StackView positionieren
- 1. TextField in den 3. StackView positionieren
- 2. TextField in den 3. StackView positionieren

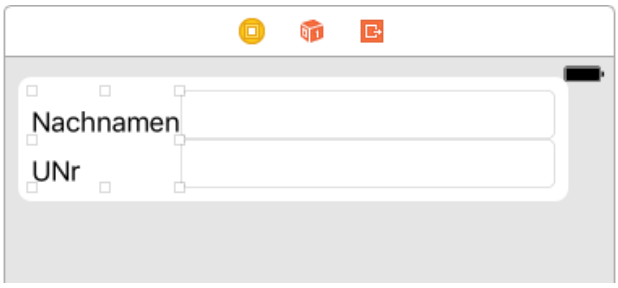


Abbildung 100 Abbildung der Label's und Textfields im ViewController

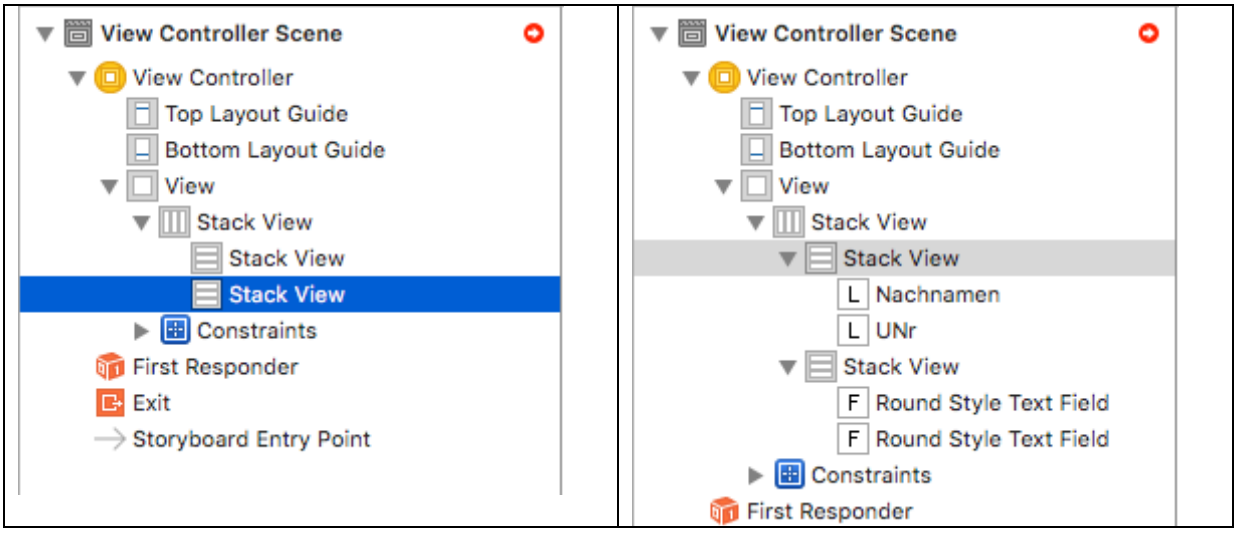
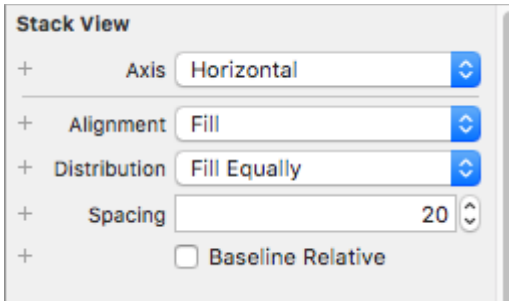


Abbildung 101 Anzeige der StackViews mit den Elementen im Elementbaum

Mit "Spacing" kann man den Raum zwischen den beiden vertikalen StackView eintragen:

- Anklicken des 1. StackView.
- Wechseln zum Property-Fenster.
- Spacingwert "20" eintragen.



Im Projektbaum werden die Contraints angezeigt:

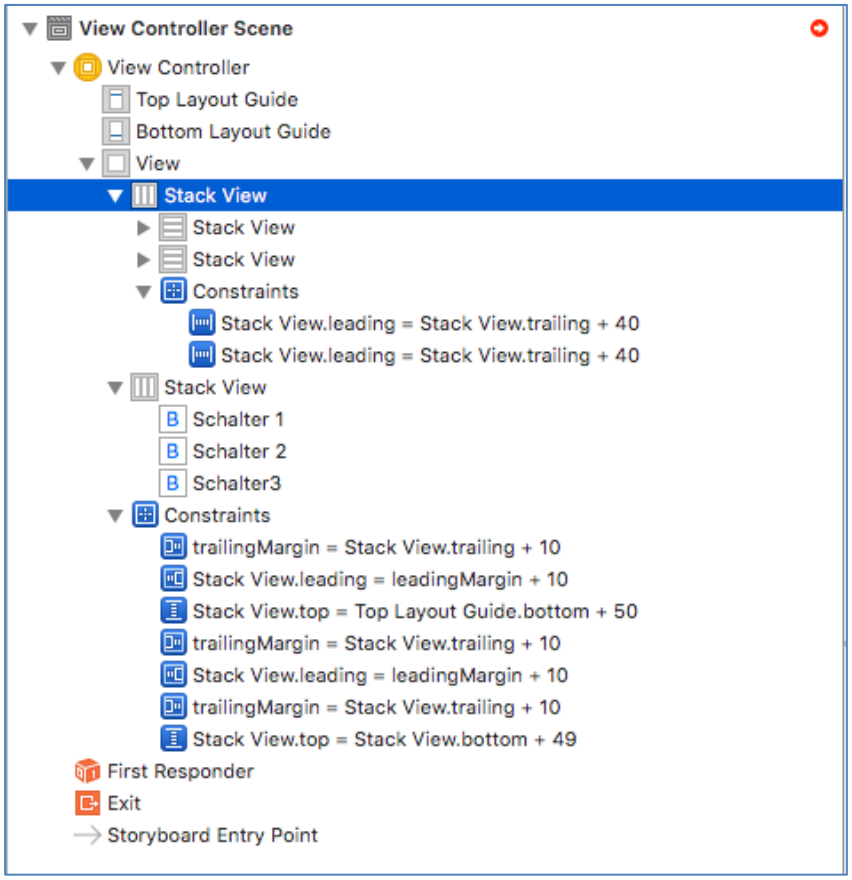
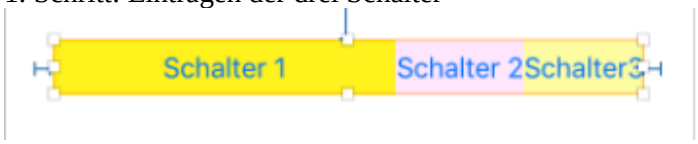


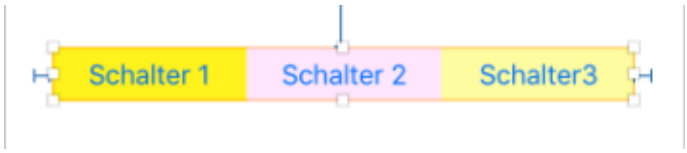
Abbildung 102 Anzeige der Constraints des 7. Layoutbeispiels

Als Übung kann man nun drei Schalter in einem horizontalen StackView platzieren:

1. Schritt: Eintragen der drei Schalter



2. Schritt: Alle Schalter haben die gleiche Größe



3. Schritt: Der Zwischenraum beträgt 20 Pixel

5.1.8 Layout 7

- Ein Label
- Ein TextField
- Ein Schalter
- Ein TextView (Multiline-Editor)

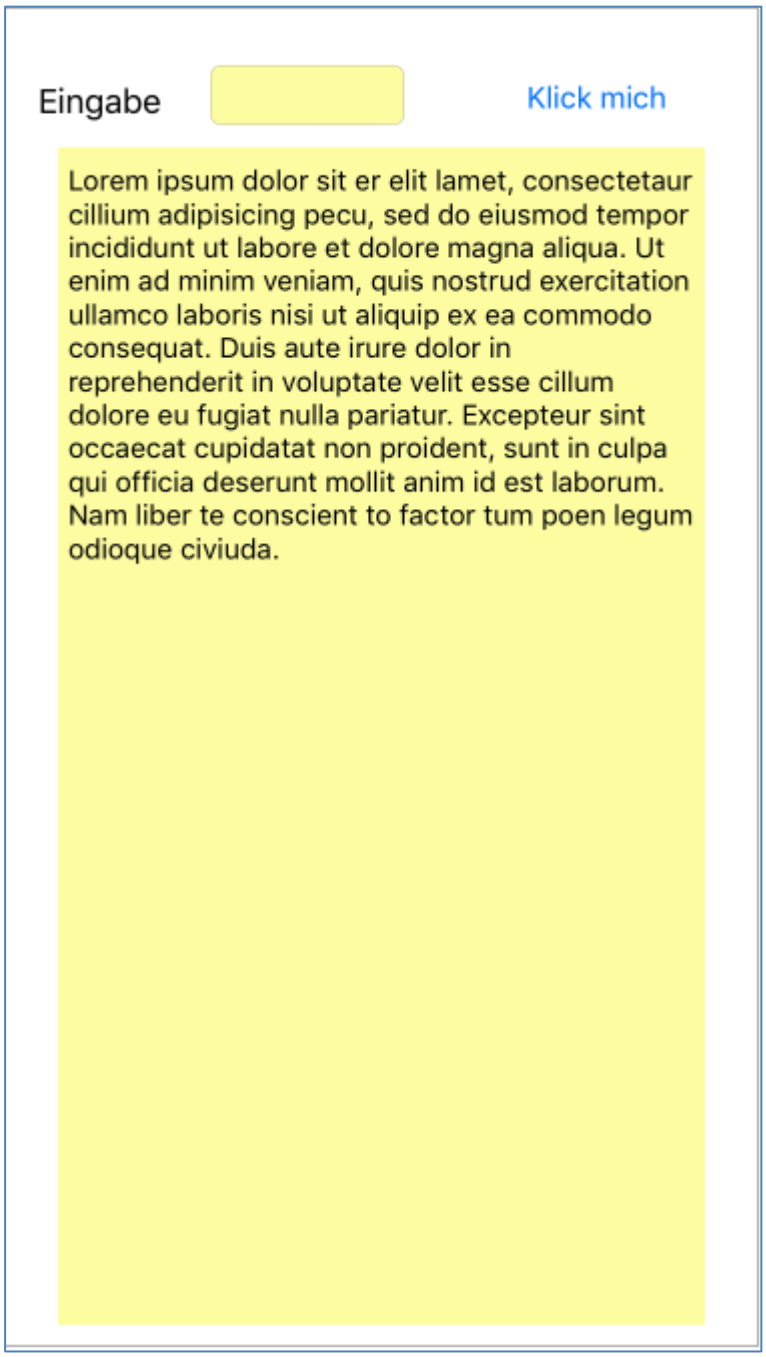


Abbildung 103 3. Layoutbeispiel: Label mit Textzeile

Die erste Zeile sollte mittlerweile leicht von der Hand gehen. Der TextView hat nun die Besonderheit, dass er in **alle** Richtungen (top, left, bottom, right) Begrenzungen hat.

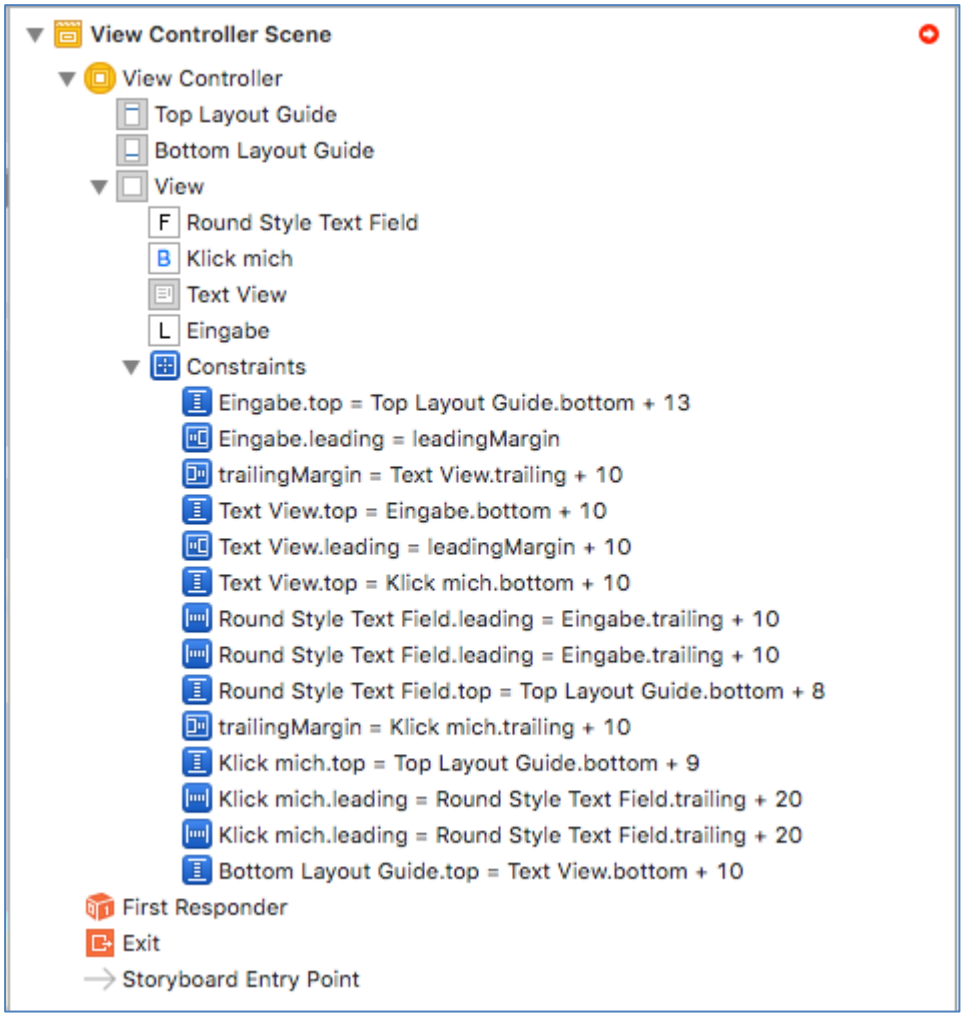


Abbildung 104 Constraints des Beispiels

Vorgehensweise:

Positionsdaten:

- **TextView**
  - Top: 10 Pixel
  - Left : 10 Pixel
  - Right: 10 Pixel
  - Bottom: 10 Pixel

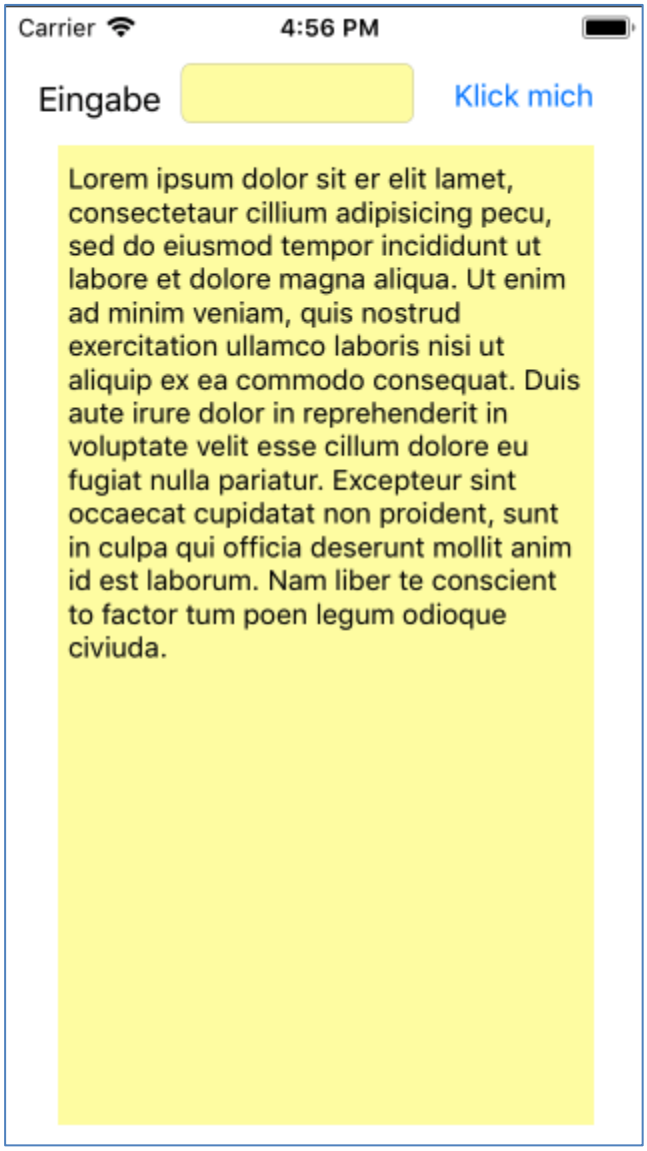


Abbildung 105 8. Layoutbeispiel: Label mit TextView

6 Projekt Klasse

Wenn man eine Klasse für alle View-Controller benötigt, kann man das folgendermaßen erreichen:.

1. Anlegen einer neuen CocoaTouch Class

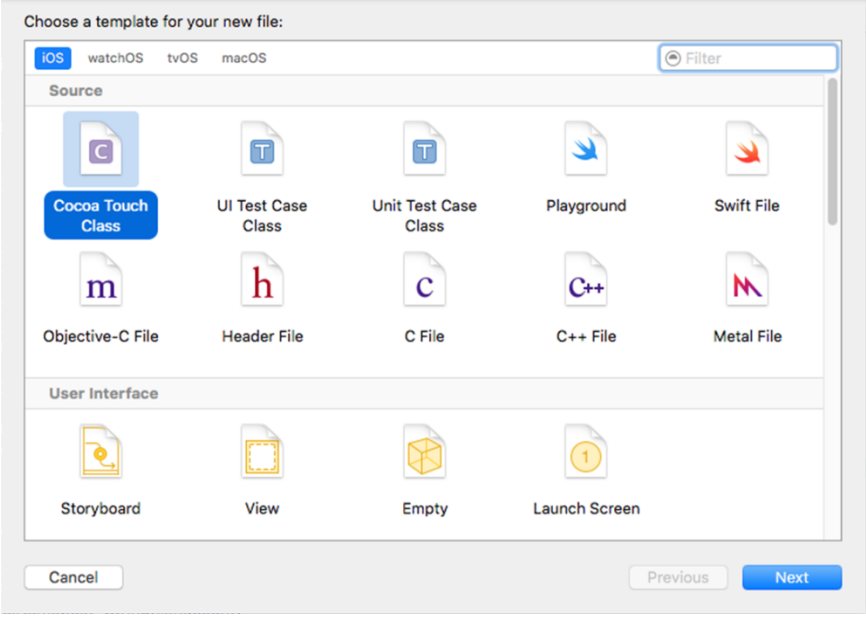


Abbildung 106 Neuer Eintrag "Cocoa Touch Class"

2. Basis-Klasse auswählen: NSObject

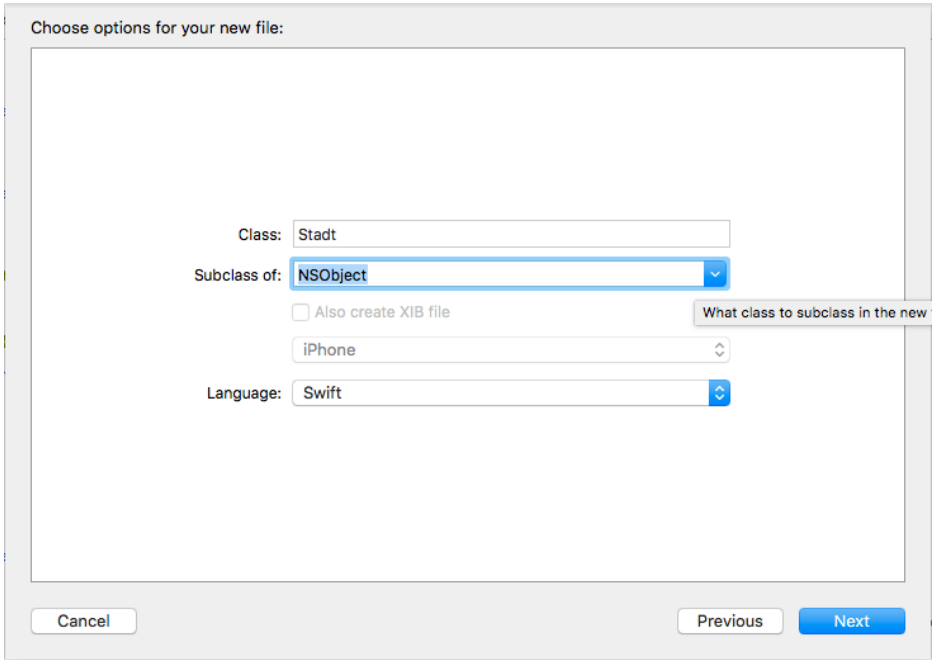


Abbildung 107 Eine neue Klasse abgeleitet von NSObject (Next Step Object)

3. **Speichern unter dem Klassennamen, z. B. City**

4. **Eintragen der Klassenstruktur:**

```
import UIKit


class City: NSObject {
    var continent: Continent=Continent.AFRICA
    var name : String = ""
    var remark : String = ""
    var visited:Visited = Visited.NONE

    init(_ continent: Continent,_ name:String, _ remark:String, _ visited:Visited){
        self.continent = continent
        self.name = name
        self. remark = remark
        self.visited = visited
    }


}
```

Abbildung 108 Projekt-Klasse „City“

Damit kann man die obige Klasse im gesamten Projekt benutzen.

7 Einfache Sofort-Dialoge (alert-Dialoge)

Möchte man Meldungen ausgeben, benötigt man kurze Dialoge. Die „alert-Klasse liefert solche Dialoge. Hier muss der Anwender zum Beenden des Dialogs einen Schalter betätigen. Also eine ähnliche Technik wie die Winwords-MessageBoxen.

Das schöne an der Technik ist, dass man auch individuelle Eingaben programmieren kann:

- Man kann eine beliebige Anzahl von Schaltern einfügen.
  - Das Attribut style bestimmt die Unterscheidung:
    - default
    - destructive (hier wird dann etwas verändert)
    - cancel
- Der „preferredStyle“ bestimmt das Aussehen:
  - Mit „alert“ wird ein normale Dialogbox angezeigt.
  - Mit „actionSheet“ wird die Dialogbox ganz unten angezeigt.
- Zusätzlich kann man nur Texteingeben einfügen. In Android ist es möglich völlig selbstständige Layouts zu entwerfen.
- **Wichtiger Punkt**
  - Der Aufruf der Dialogbox geschieht asynchron! Das heißt, dass sofort nach dem Aufruf im Quellcode weitergegangen wird. Man wartet also **nicht** auf das Ende der Dialogbox. Deshalb benötigt man separate Methoden für jeden Fall. Hier wäre die Alternative, dass man Funktionsparameter übergibt (siehe yes-no und yes-no-cancel)

7.1 Alert-Dialog ok Schalter

Hier wird ein Ok-Schalter eingebaut. Die Methode ist als standalone Methode deklariert. Sie benötigt nur die Caption und den Text.

```
func showOKDialog(_title title:String, _message message:String) {
  //Erstellen eines  UIAlertController
  //setzen des Titel und der message
  let alertController =
UIAlertController(title: title, message: message, preferredStyle: .alert)

  //the confirm action taking the inputs
  let okAction = UIAlertAction(title: "Ok", style: .default) { (_) in
     self.ok()    // optional
  }

  alertController.addAction(okAction)

  // nun anzeigen
  self.present(alertController, animated: true, completion: nil)
}  // showOKDialog

  // Hilfsfunktionen
  func ok(){
    editor.text="Ergebnis\nOk gedrückt"
  }

Aufruf:
showOKDialog(_title: "Info", _message: "Ihre Festplatte wurde gelöscht")
```

Abbildung 109 Alert-Dialog Ok (alert)

Abbildung 110 Alert-Dialog Ok (actionSheet)

7.2 Alert-Dialog ok und cancel Schalter

Hier werden zwei Schalter (positiv und negativ) eingebaut. Die Methode ist als standalone Methode deklariert. Sie benötigt die Caption und den Text.

```
func showOKCancelDialog(_title title:String, _message message:String) {
  //Erstellen eines  UIAlertController
  //setzen des Titel und der message
  let alertController = UIAlertController(title: title, message: message,
preferredStyle: .actionSheet)

  let okAction = UIAlertAction(title: "Ok", style: .default) { (_) in
     self.ok()
  }

  let cancelAction = UIAlertAction(title: "Cancel", style: .cancel) { (_)in
    self.cancel()
  }

  // hinzufügen zum Dialog
  alertController.addAction(okAction)
  alertController.addAction(cancelAction)

          //anzeigen
  self.present(alertController, animated: true, completion: nil)
}  // showInputDialog


  // weitere Methoden
  func ok(){
     editor.text="Ergebnis\nOk gedrückt"
  }
  func cancel(){
     editor.text="Ergebnis\nCancel gedrückt"
  }

Aufruf:
showOKDialog(_title: "Info", _message: "Ihre Festplatte wurde gelöscht")
```

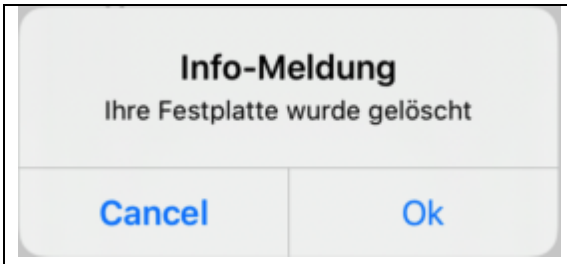


Abbildung 111 Alert-Dialog Ok (alert)

Abbildung 112 Alert-Dialog Ok Cancel (actionSheet)

7.3 Alert-Dialog yes und no Schalter

Hier werden wieder zwei Schalter (positiv und negativ) eingebaut. Diesmal haben sie die Beschrifungen „yes“ und „no“. Die Methode ist als standalone Methode deklariert. Sie benötigt die Caption und den Text.

```
func showYesNoDialog(_title title:String, _message message:String) {
  let alertController = UIAlertController(title: title, message: message,
preferredStyle: .actionSheet)

  let yesAction = UIAlertAction(title: "Yes", style: .default) { (_) in
    self.yes()
  }

  let noAction = UIAlertAction(title: "No", style: .cancel) { (_) in
    self.no()
  }

  alertController.addAction(yesAction)
  alertController.addAction(noAction)

  self.present(alertController, animated: true, completion: nil)
}  // showYesNoDialog

  // weitere Methoden
func yes(){
   editor.text="Ergebnis\nYes gedrückt"
}

func no(){
   editor.text="Ergebnis\nNo gedrückt"
}

Aufruf:
showYesNoDialog(_title: "Datei löschen",_message: "Wollen Sie wirklich die
Datei t.txt löschen?")
```

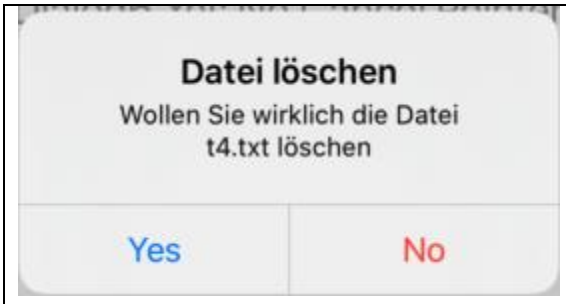


Abbildung 113 Alert-Dialog Yes/No (alert)

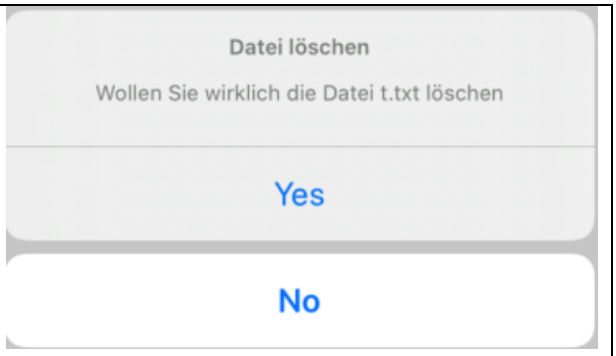


Abbildung 114 Alert-Dialog Yes/No (actionSheet)

7.4 Alert-Dialog no und yes Schalter

Hier werden wieder zwei Schalter (positiv und negativ) eingebaut. Diesmal haben sie die Beschrifungen „yes“ und „no“. Die Methode ist als standalone Methode deklariert. Sie benötigt die Caption und den Text.

```
func showYesNoDialog(_title title:String, _message message:String) {
  let alertController = UIAlertController(title: title, message: message,
preferredStyle: .actionSheet)

  let noAction = UIAlertAction(title: "No", style: .default) { (_) in
    self.yes()
  }

  let yesAction = UIAlertAction(title: "Yes", style: .cancel) { (_) in
    self.no()
  }

  alertController.addAction(noAction)
  alertController.addAction(yesAction)

  self.present(alertController, animated: true, completion: nil)
}  // showYesNoDialog

  // weitere Methoden
func yes(){
   editor.text="Ergebnis\nYes gedrückt"
}

func no(){
   editor.text="Ergebnis\nNo gedrückt"
}

Aufruf:
showYesNoDialog(_title: "Datei löschen",_message: "Wollen Sie wirklich die
Datei t.txt löschen?")
```

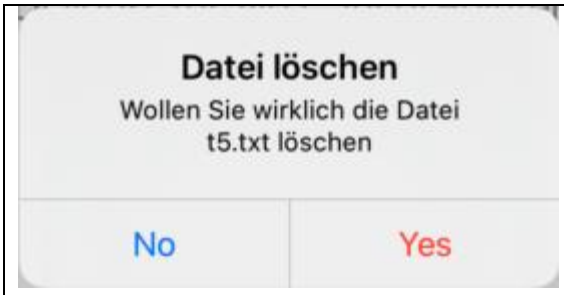


Abbildung 115 Alert-Dialog No/Yes (alert)

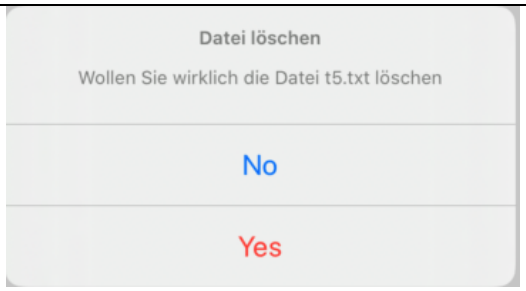


Abbildung 116 Alert-Dialog No/Yes (actionSheet)

7.5 Alert-Dialog yes und no Schalter mit Funktionspointern

Hier werden wieder zwei Schalter (positiv und negativ) eingebaut. Diesmal haben sie die Beschrifungen „yes“ und „no“. Die Methode ist als standalone Methode deklariert. Sie benötigt die Caption, den Text und zwei Funktionspointer. In dieser Variante kann die untere Methode in unterschiedlichen Kontexten aufgerufen werden. **Die „Rückgabe-Methoden“ sind also unterschiedlich**!

```
func showYesNoDialog(_title title:String, _message message:String, _yesFunc
yesFunction : @escaping()->Void, _noFunc noFunction : @escaping()->Void) {

  let alertController = UIAlertController(title: title, message: message,
preferredStyle: .actionSheet)

  let yesAction = UIAlertAction(title: "Yes", style: .default) { (_) in
    yesFunction()
  }

  let noAction = UIAlertAction(title: "No", style: .cancel) { (_) in
    noFunction()
  }

  alertController.addAction(yesAction)
  alertController.addAction(noAction)

  self.present(alertController, animated: true, completion: nil)
}  // showYesNoDialog

  // weitere Methoden
func yes(){
   editor.text="Ergebnis\nYes gedrückt"
}

func no(){
   editor.text="Ergebnis\nNo gedrückt"
}

Aufruf:
showYesNoDialog(_title: "Datei löschen",_message: "Wollen Sie wirklich die
Datei t.txt löschen?")
```

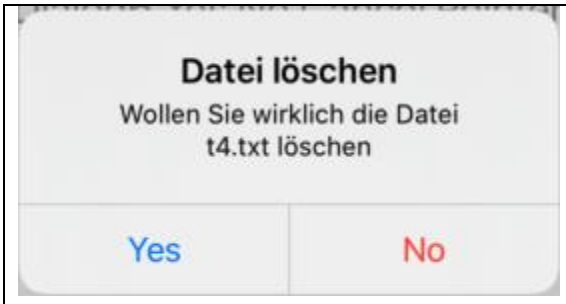


Abbildung 117 Alert-Dialog Yes/No (alert)

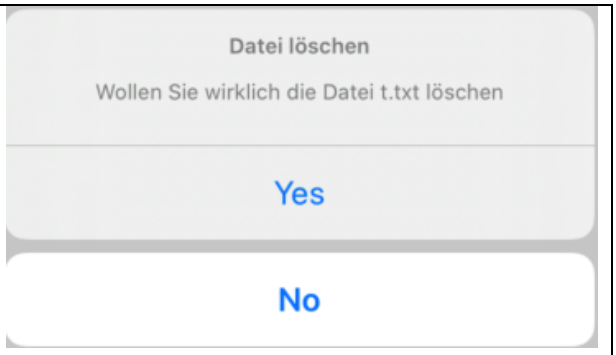


Abbildung 118 Alert-Dialog Yes/No (actionSheet)

7.6 Alert-Dialog yes,no und cancel Schalter

Hier werden wieder drei Schalter (positiv, negativ und neutral) eingebaut. Diesmal haben sie die Beschrifungen „no“ und „yes“. Die Methode ist als standalone Methode deklariert. Sie benötigt ontext, die Caption, den Text und drei Funktionspointer. In dieser Variante kann die untere Methode in unterschiedlichen Kontexten aufgerufen werden. **Die „Rückgabe-Methoden“ sind also unterschiedlich**!

```
func showYesNoCancelDialog(_title title:String, _message message:String,
_yesFunc  yesFunction : @escaping()->Void, _noFunc noFunction :
@escaping()->Void, _cancelFunc cancelFunction : @escaping()->Void) {

  let alertController = UIAlertController(title: title, message: message,
preferredStyle: .alert)

  let yesAction = UIAlertAction(title: "Yes", style: .default) { (_) in
    yesFunction()
  }

  let noAction = UIAlertAction(title: "No6", style: .destructive) { (_) in
    noFunction()
  }

  let cancelAction = UIAlertAction(title: "Cancel", style: .cancel) { (_)in
    cancelFunction()
  }

  alertController.addAction(yesAction)
  alertController.addAction(noAction)
  alertController.addAction(cancelAction)

  self.present(alertController, animated: true, completion: nil)
}  // showYesNoCancelDialog
```

```
// Zusatzmethoden
func yes()->Void{
  editor.text="Ergebnis\nYes6 gedrückt"
}

func no()->Void{
  editor.text="Ergebnis\nNo6 gedrückt"
}
func yes2()->Void{
  editor.text="Ergebnis\nYes6 gedrückt"
}

func no2()->Void{
  editor.text="Ergebnis\nNo6 gedrückt"
}

func cancel()->Void{
  editor.text="Ergebnis\nCancel6 gedrückt"
}


Aufrufe (unterschiedliche yes und no Pointer!):
showYesNoCancelDialog(_title: "Bearbeiten beenden",_message: "Soll die
Datei t6.txt gespeichert werden?", _yesFunc: yes, _noFunc: no, _cancelFunc:
cancel)

showYesNoCancelDialog(_title: "Bearbeiten beenden",_message: "Soll die
Änderungen beim Eintrag „Oldenburg" gespeichert werden?", _yesFunc: yes2,
_noFunc: no2, _cancelFunc: cancel)
```

Abbildung 119 Alert-Dialog Yes/No/Cancel (alert)

Abbildung 120 Alert-Dialog Yes/No/Cancel (actionSheet)

7.7 Alert-Dialog mit drei Eingabezeilen

In diesem Beispiel werden drei Eingabezeilen dargestellt. Auch hier benötigt man eione externe Methode, zum Beispiel ok(vorname:String, nachname:String, matrikelnummer:String).

```
func showInputDialog() {
   let alertController = UIAlertController(title: "Alert Dialog", message:
"Eingaben", preferredStyle: .alert) // hier kein actionSheet !

  let okAction = UIAlertAction(title: "Enter", style: .default) { (_) in
    //getting the input values from user
    let vorname = alertController.textFields?[0].text
    let nachname = alertController.textFields?[0].text
    let matrnr = alertController.textFields?[2].text
    self.ok(vorname, nachname, matrnr)
  }

  let cancelAction = UIAlertAction(title: "Cancel", style: .cancel) { (_)in
         // keine Aktion oder self.cancel()
}

        //adding textfields to our dialog box
   alertController.addTextField { (textField) in
      textField.placeholder = "Vorname"
      textField.keyboardAppearance = .dark
      textField.keyboardType = .default
      textField.autocorrectionType = .default
      textField.placeholder = "Eingabe des Vornamens"
      textField.textColor = UIColor.green
   }

   alertController.addTextField { (textField) in
      textField.placeholder = "Nachname"
      textField.keyboardAppearance = .dark
      textField.keyboardType = .namePhonePad
      textField.autocorrectionType = .default
      textField.placeholder = "Eingabe des Nachnamens"
      textField.textColor = UIColor.green
      // textField.isSecureTextEntry = true  // password
   }

   alertController.addTextField { (textField) in
      textField.placeholder = "Matrikelnummer"
      textField.keyboardAppearance = .dark
      textField.keyboardType = .numberPad
      textField.autocorrectionType = .default
      textField.placeholder = "Eingabe der Matrikelnummer (ohne m)"
      textField.textColor = UIColor.blue
      textField.font = UIFont(name: "AmericanTypewriter", size: 14)
   }

   alertController.addAction(okAction)
   alertController.addAction(cancelAction)

    self.present(alertController, animated: true, completion: nil)
}  // showInputDialog2// Zusatzmethoden

func ok(vorname:String, nachname:String, matrikelnummer:String)->Void{
  // action
}
```

```
func cancel()->Void{
  editor.text="Ergebnis\nCancel6 gedrückt"
}
```

Aufrufe:

```
showInputDialog2()
```

Abbildung 121 Eingabe dreier Zeilen

8 PageBased

Die PageBased-Application ist eine „Vorlage“ zum Anzeigen von Seitenorientierten Daten (in Android heißen die Seiten Fragmente), dazu gehören PDF-Dateien oder eine Auflistung von Kochrezepten. Bis xcode Version 12 gab es einen Wizzard, der die Struktur komplett erzeugt hat. Ab xcode 13 gibt es nur einen PageBased-Controller, der aber ohne die Hilfstruktur implementiert ist.

Dieses Kapitel beschreibt deshalb eine andere Variante. Im Projekt werden Swipe-Gesten angefangen, um damit die Logik der wechselnden Fenster zu simulieren.

Ablauf:

1. Erstellen eines Projektes „Single View Application“
2. Einfügen der Hilfsklassen „City“.
3. Einbau der UI-Elemente.
4. Die Referenzen erstellen
5. Gesture-Logik einbauen

8.1 Projekt erstellen

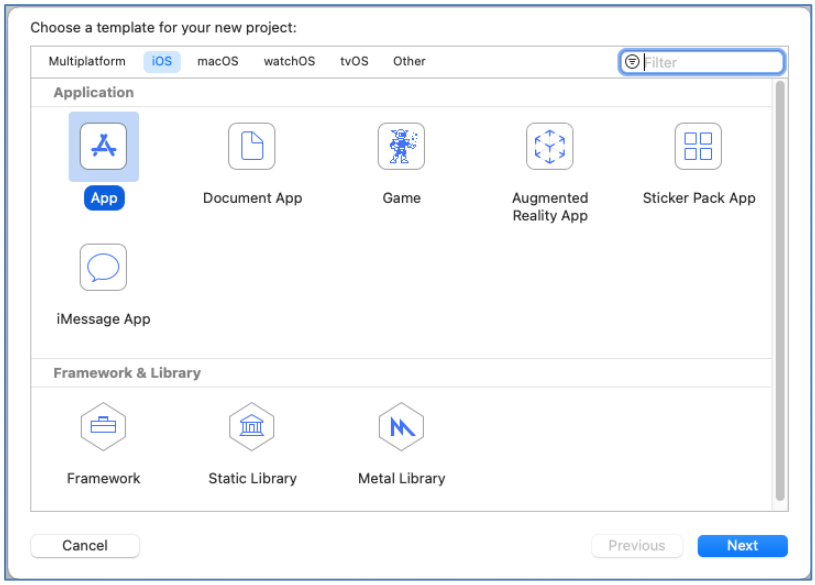


Abbildung 122 Dialog zum Erstellen eines Projektes

Nun wird der Name eingegeben: NaviController07

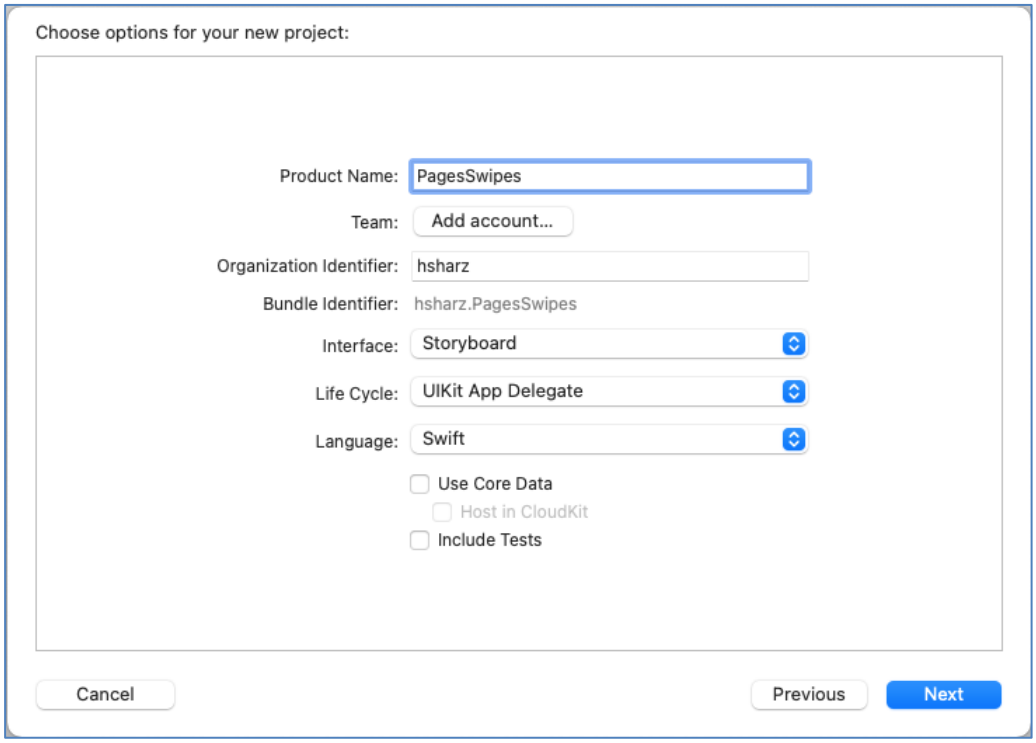


Abbildung 123 Eintragen des Namens

Danach wird das Projekt gespeichert:

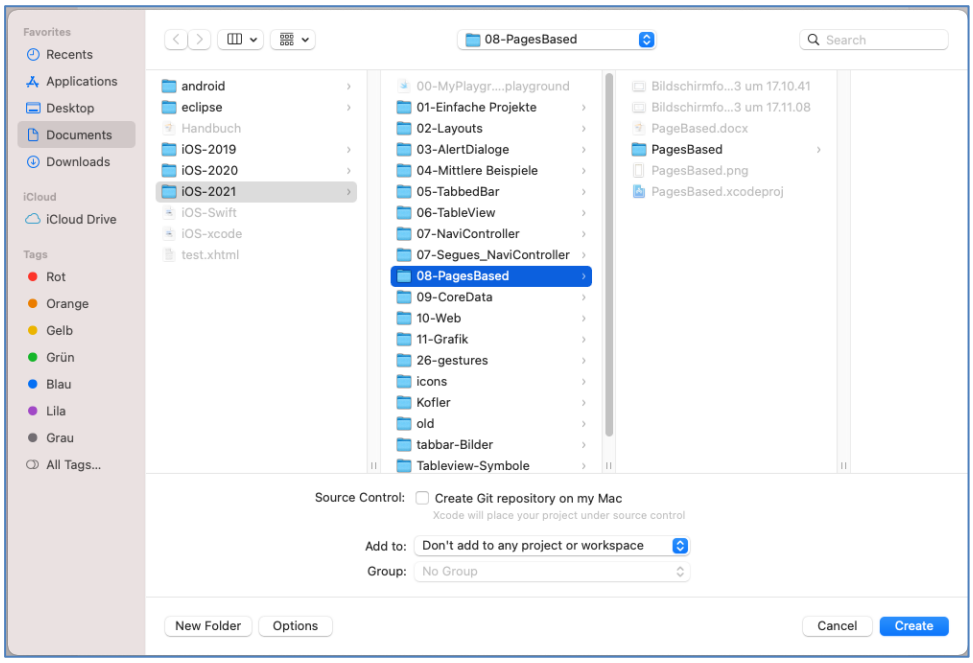


Abbildung 124 Projekt im Ordner speichern

Nun hat man kann bekannte Anfangsszenario.

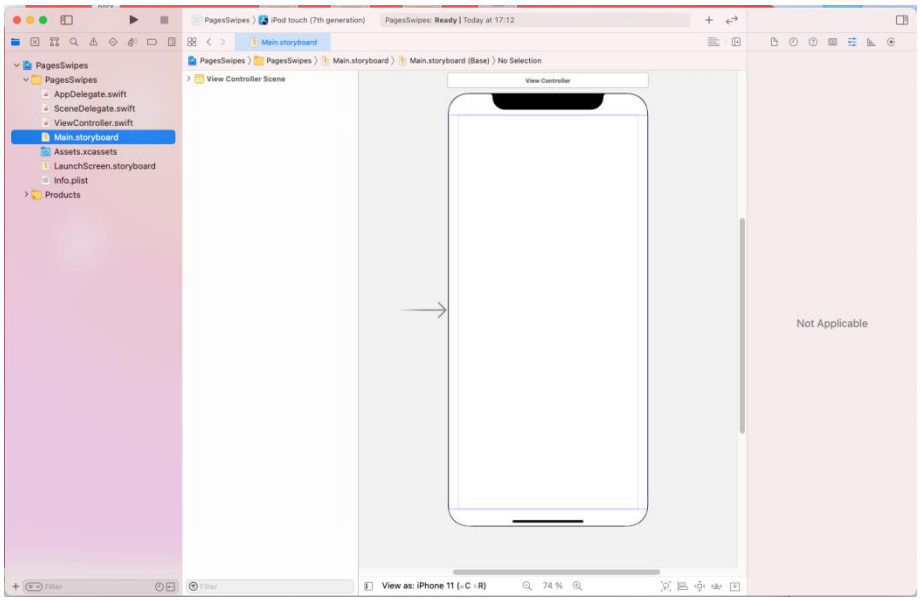


Abbildung 125 Anfangsszenario

8.1.1 Einfügen der UI-Elemente

Der UIViewController hat leider keine Gestensteuerung, deshalb muss als erstes eine View eingefügt werden.

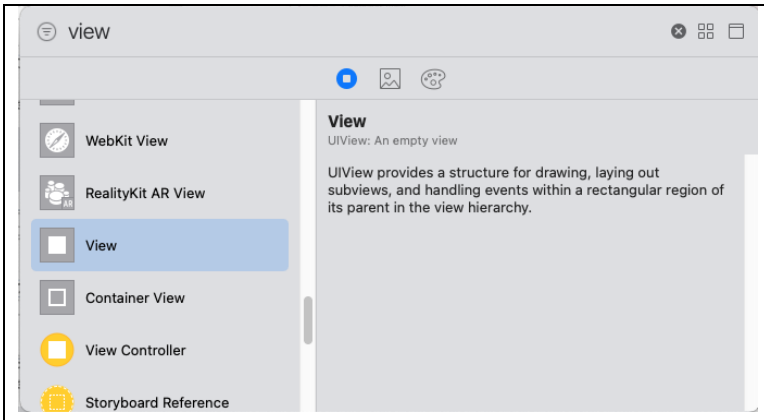
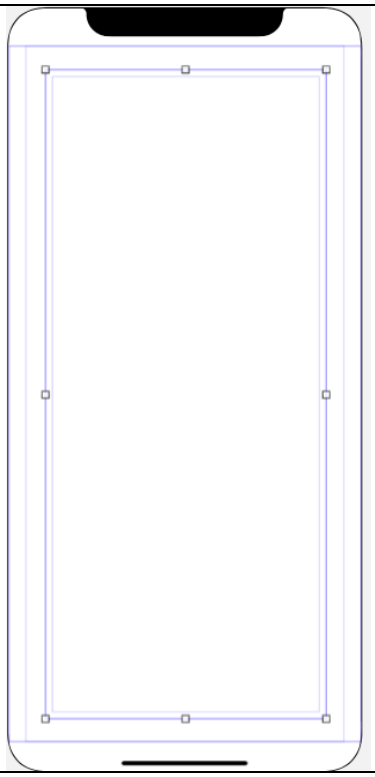


Abbildung 126 View einfügen

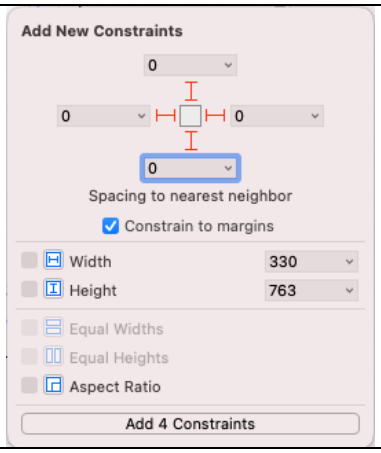


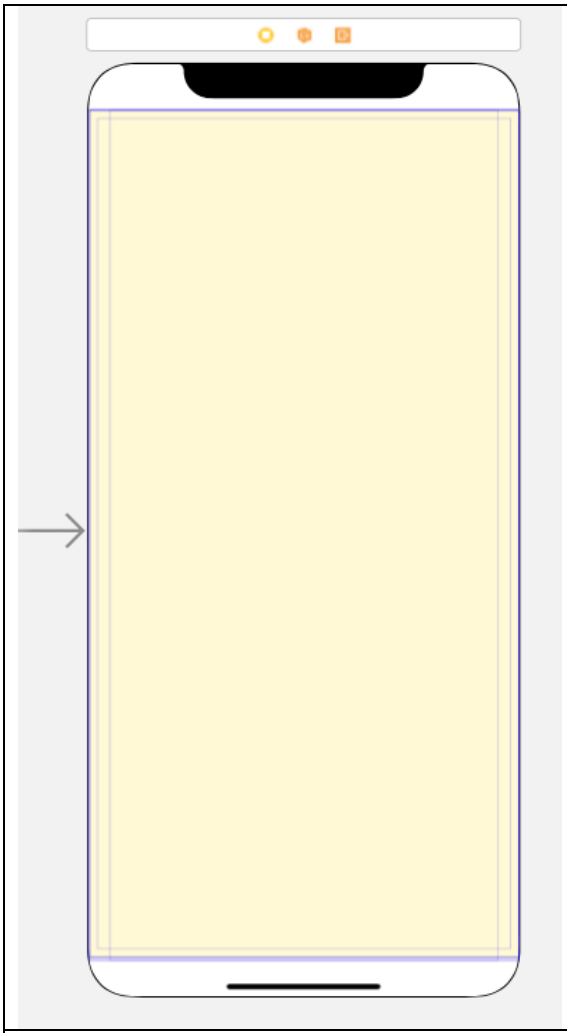
Als Contraints werden viermal die Null gewählt:

Contrainst der View:

- Top:0
- Left: 0
- Right: 0
- Bottom: 0

Nun noch ein helles Gelb als Hintergrundfarbe auswählen.





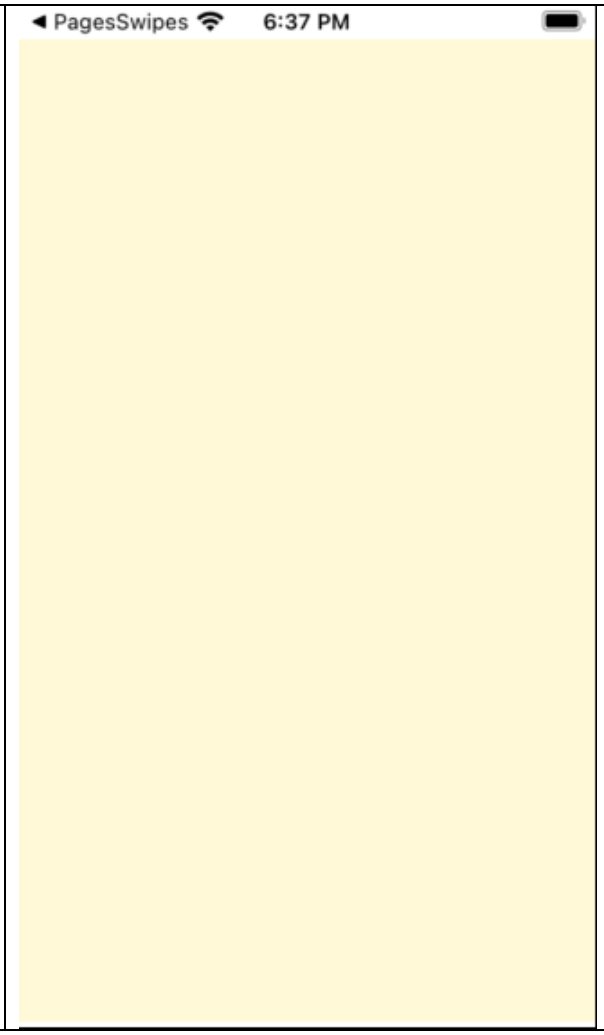


Abbildung 127 PagesSwipe mit einem gelben View

Nun werden die eigentlichen UI-Elemente in dem gelben View eingefügt:

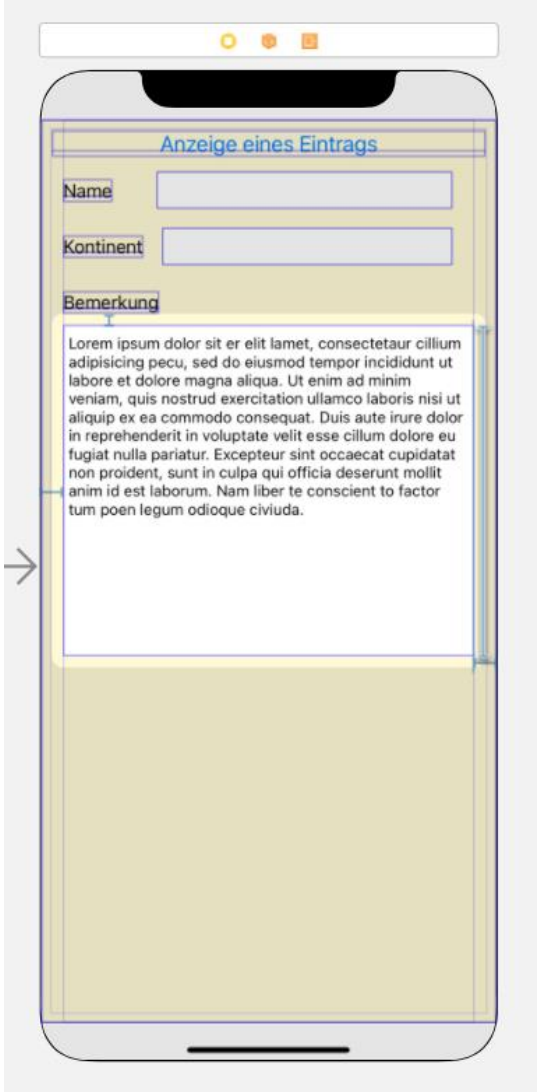


Abbildung 128 UI-Elemente in PagesSwipe

LabelCaption:
- Top: 10
- Left: 10
- Right: 10

LabelName:
- Top: 20
- Left: 20

tName:
- Top: 13
- Left: 40
- Right: 40

LabelKontinent:
- Top: 30
- Left: 20

tKontinent:
- Top: 16
- Left: 12
- Right: 40

LabelBemerkung
- Top: 30
- Left: 20

tBem (TextView):
- Top: 10
- Left: 10
- Right: 10
- Height: 300

8.1.2 Referenzen erstellen

Für folgende UI-Elemente werden Referenzen erstellt:

- **backgroundview: UIView!**
- tname: UITextField!
- var tcontinent: UITextField!
- var tremark: UITextView!

8.1.3 Hilfsklassen erstellen

In dem Projekt sollen Städte angezeigt werden. Dazu brauchen wir eine Klasse „City“ und eine Verwaltungsklasse „CitiesDB“.

Anlegen einer neuen Klasse „city.swift“

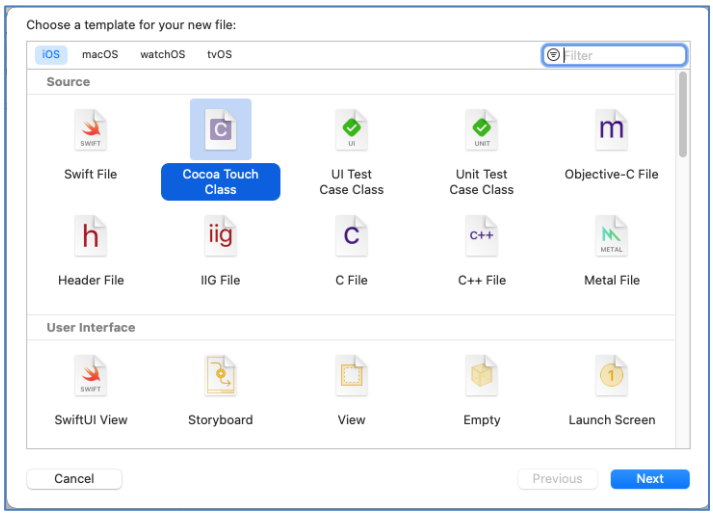


Abbildung 129 Einfüge-Dialog für die Klasse City.swift (Cmd+N)

Nun den Namen und den Typ eintragen:

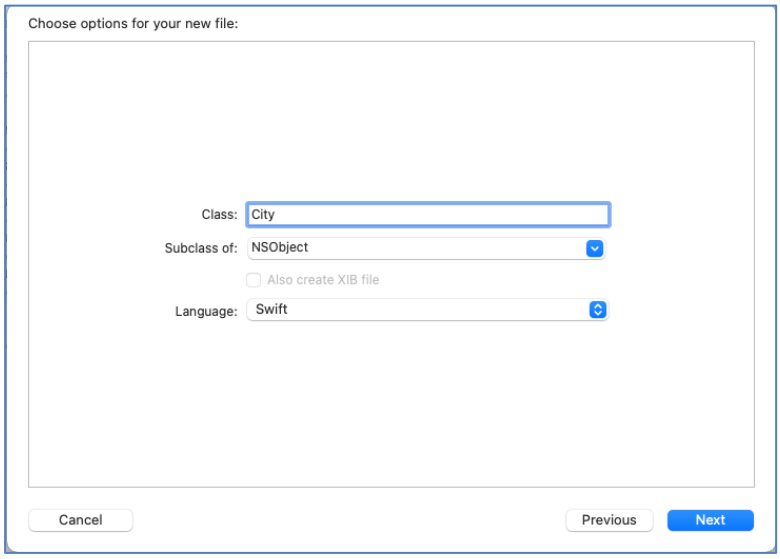


Abbildung 130 Name und Typ der Klasse City

Nun noch eine zweite Klasse „Enums.swift“ und eine dritte Klasse „CitiesDB.swift“ einfügen.
In der Klasse „Enums.swift“ werden die Aufzählungen, Enumeration eingetragen:

```
import UIKit

enum Continent:Int {          // Deklaration
  case  AFRICA=0, AMERICA, ASIA, AUSTRALIA, EUROPE, ANTARTKIS
}

func getContinentText(_ continent:Continent) -> String {
        switch (continent) {
```

```
         case Continent.AFRICA:
            return "Afrika"
            case Continent.AMERICA:
               return "Amerika"
            case Continent.ASIA:
               return "Asien"
            case Continent.AUSTRALIA:
               return "Australien"
        case Continent.EUROPE:
           return "Europa"
        case Continent.ANTARTKIS:
           return "Antartkis"
            default:
                   return "unbekannt"
        }
    }

enum Visited : Int{        // Deklaration
  case  NONE=0, DOWN, AVERAGE, UP
}
```

Quellcode für Enums.swift

Nun die Klasse „City“

```
import UIKit


class City: NSObject {
      // Attribute
   var continent: Continent=Continent.AFRICA
   var name : String = ""
   var remark : String = ""
   var visited:Visited = Visited.NONE

     // Konstruktor
   init(_ continent: Continent,_ name:String, _ remark:String, _ visited:Visited ){
      self.continent = continent
      self.name = name
      self.remark = remark
      self.visited = visited
   }  // init

}
```

Quellcode für City.swift

Am Schluss wird die Verwaltungsklasse eingefügt.

```
import UIKit

class CitiesDB: NSObject {

   public static func loadCitiesIntern()  -> Array<City> {
     var cities=[City]()
     cities.append( City( Continent.AMERICA ,  "New York",  "Manhattan", Visited.NONE)  )
     cities.append( City( Continent.EUROPE , "Mailand", "Mailander Scala", Visited.AVERAGE))
     cities.append( City( Continent.AUSTRALIA ,"Central Australien","Ayers Rock, Uluru",
Visited.NONE)  )
     cities.append( City( Continent.AMERICA ,"Peru",  "Machu Picchu",  Visited.UP)  )
```

```
     cities.append( City( Continent.EUROPE ,"Wernigerode",  "HS Harz",  Visited.DOWN)  )
     cities.append( City( Continent.AUSTRALIA ,"Queensland",  "The Great Barrier Reef",
Visited.NONE)  )
     cities.append( City( Continent.AMERICA ,"Rio de Janeiro", "Karneval",  Visited.DOWN)  )
     cities.append( City( Continent.EUROPE ,"Brüssel",   "Atomium",  Visited.UP)  )
     cities.append( City( Continent.AUSTRALIA ,"Sidney",  "Opernhaus",  Visited.UP)  )
     cities.append( City( Continent.AFRICA ,"Kairo",  "Pyramiden",  Visited.NONE)  )
     cities.append( City( Continent.AMERICA ,"Chile",  "Atacama Wüste",  Visited.UP)  )
     cities.append( City( Continent.ASIA ,"Tokio",  "Shinjuku Gyoen National Garden",
Visited.DOWN)  )
     cities.append( City( Continent.EUROPE ,"Kiel",  "Kieler Woche",  Visited.UP)  )
     cities.append( City( Continent.AFRICA ,"Hunsbergen","Apollo 11 Höhle",  Visited.NONE) )
     cities.append( City( Continent.EUROPE ,"Dresden",  "Das grüne Zimmer",  Visited.NONE) )
     cities.append( City( Continent.AFRICA ,"Marrakesch",  "Botanische Garten Jardin
Majorelle",  Visited.DOWN)  )
     cities.append( City( Continent.AMERICA ,"San Fransisco",  "Golden Gate Bridge",
Visited.UP)  )
     cities.append( City( Continent.EUROPE ,"Halberstadt",  "John Cage",  Visited.NONE)  )
     cities.append( City( Continent.ASIA ,"Peking",  "Verbotene Stadt",  Visited.NONE)  )
     cities.append( City( Continent.EUROPE ,"Venezuela",  "Angel Falls",  Visited.UP)  )
     cities.append( City( Continent.AMERICA ,"Bilbao",  "Guggenheim-Museum",  Visited.UP) )
     cities.append( City( Continent.ASIA ,"Chiang Mai",  "Der weiße Tempel",  Visited.UP) )
     cities.append( City( Continent.AMERICA ,"Rio de Janeiro",  "Zuckerhut",  Visited.NONE))
     return cities
}
```

Quellcode für CitiesDB.swift

8.1.4 Gesten einbauen

Es werden zwei GestureRecognizer, links und rechts erzeugt und dem gelben View hinzugefügt. Jeder GestureRecognizer hat eine Eventmethode, gestureLeft und gestureRight, mit der man die Daten erneuern kann.

```
import UIKit

class ViewController: UIViewController {

    @IBOutlet var backgroundview: UIView!
    @IBOutlet var tname: UITextField!
    @IBOutlet var tcontinent: UITextField!
    @IBOutlet var tremark: UITextView!

    var cities : [City] = []


    var index:Int=0

    override func viewDidLoad() {
        super.viewDidLoad()
        cities = CitiesDB.loadCitiesIntern()
        index=0
        showDetails(index)

        let gestureLeft1 =  UISwipeGestureRecognizer(
          target: self,
          action: #selector(ViewController.gestureLeft(_:)))
        gestureLeft1.direction = .left
        backgroundview.addGestureRecognizer(gestureLeft1)
```

```
        let gestureRight1 =  UISwipeGestureRecognizer(
          target: self,
          action: #selector(ViewController.gestureRight(_:)))
        gestureRight1.direction = .right
        backgroundview.addGestureRecognizer(gestureRight1)

    }

    func showDetails(_ index:Int) {
        if ( (index>=0) && (index<cities.count) ){
            let city = cities[index]
            tname.text = city.name
            tcontinent.text = getContinentText(city.continent)
            tremark.text = city.remark
        }
    }


    // Actions für Gestures, die in viewDidLoad eingerichtet wurden
    @objc func gestureLeft(_ tapGR: UIGestureRecognizer) {
        if ( index<(cities.count-1)){
            index+=1
            showDetails(index)
        }
    } // gestureLeft

    @objc func gestureRight(_ tapGR: UIGestureRecognizer) {
        if ( index>0){
            index-=1
            showDetails(index)
        }
    }  // gestureRight
}
```

9 Tab-Bar-Controller

Der Tab-Bar-Controller ist die einfachste Version für die Verwendung mehrerer ViewControllern. In alten xcode-Versionen konnten man diesen Projekttyp separat auswählen. Aktuelle muss man eine Single-App auswählen und den HauptView durch eine TabbBar-Controller ersetzen. Diese hat dann einen internen ViewController (in Android Fragment genannt).

Neues Projekt erstellen:

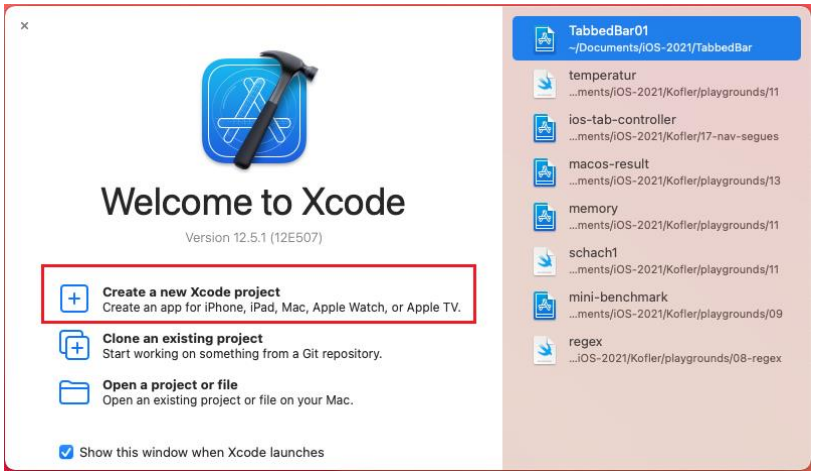


Abbildung 131 Neues Projekt erstellen (hier Single-App)

Im nächsten Fenster wählt man die iOS / Single-App Variante aus:

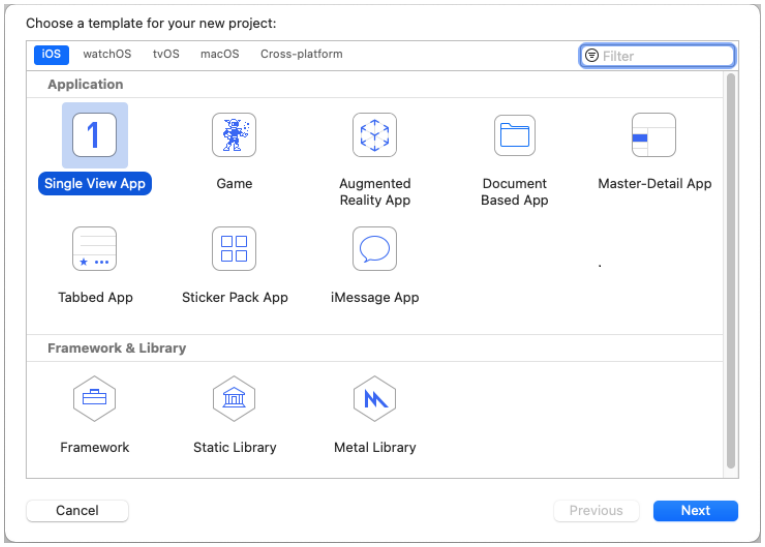


Abbildung 132 Single View Application

Danach muss man den gewünschten Namen eingeben (**TabbedBar01**):

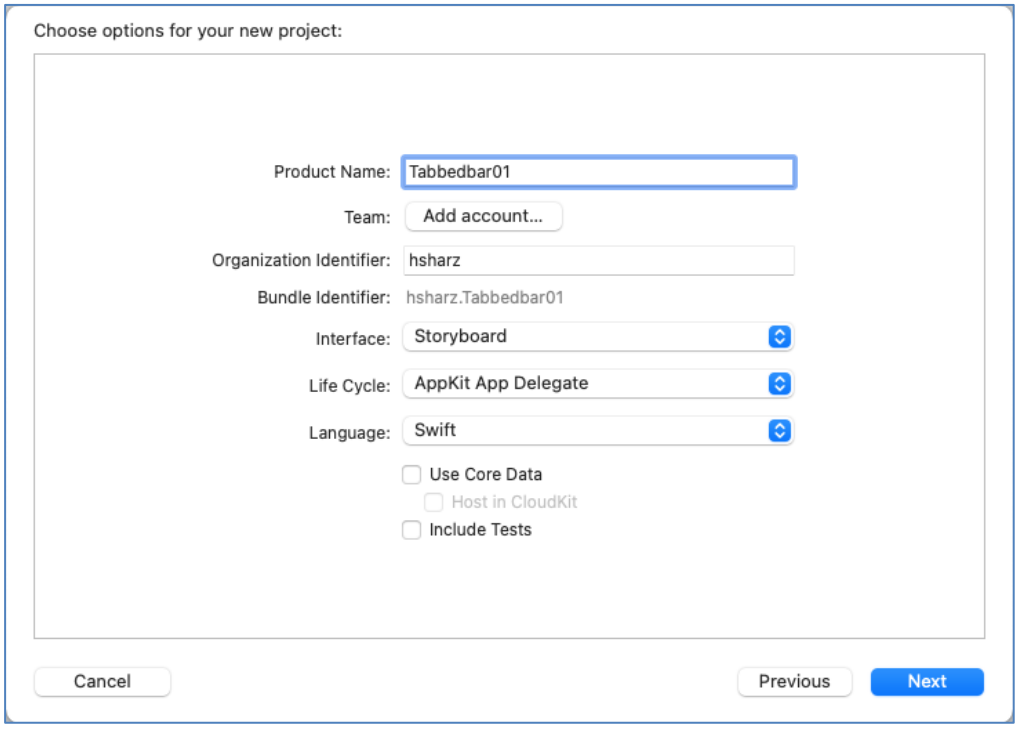


Abbildung 133 Name der App

Danach muss den den gewünschten Ordner auswählen:

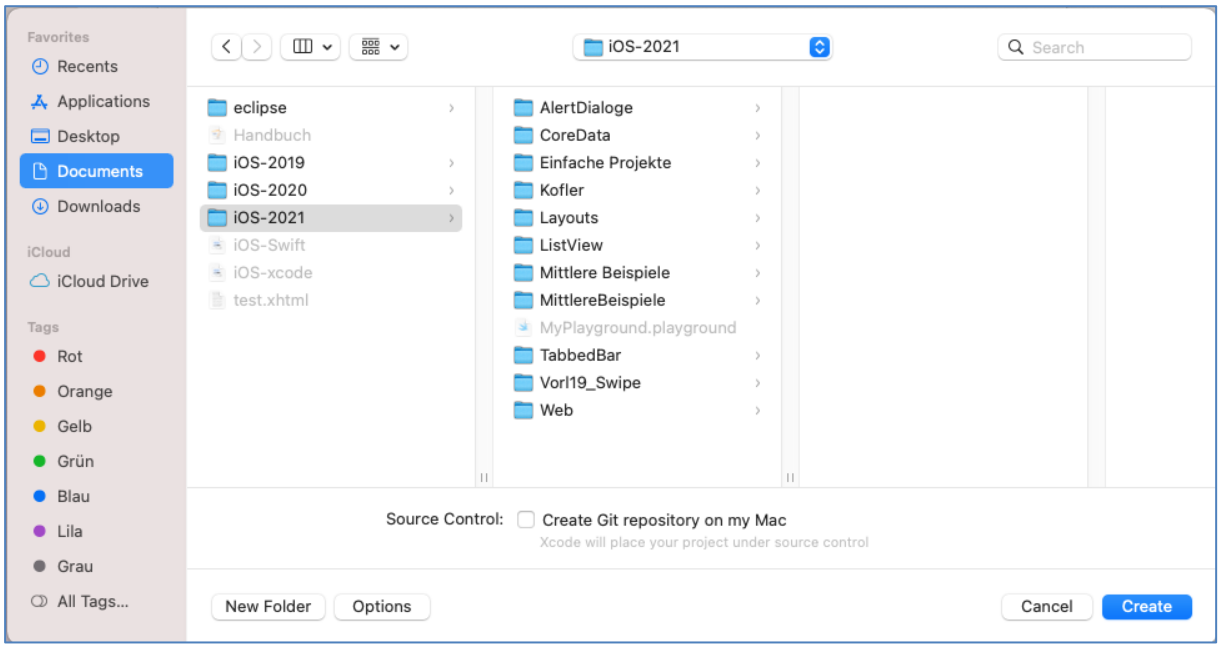


Abbildung 134 Speichern des neuen Projektes

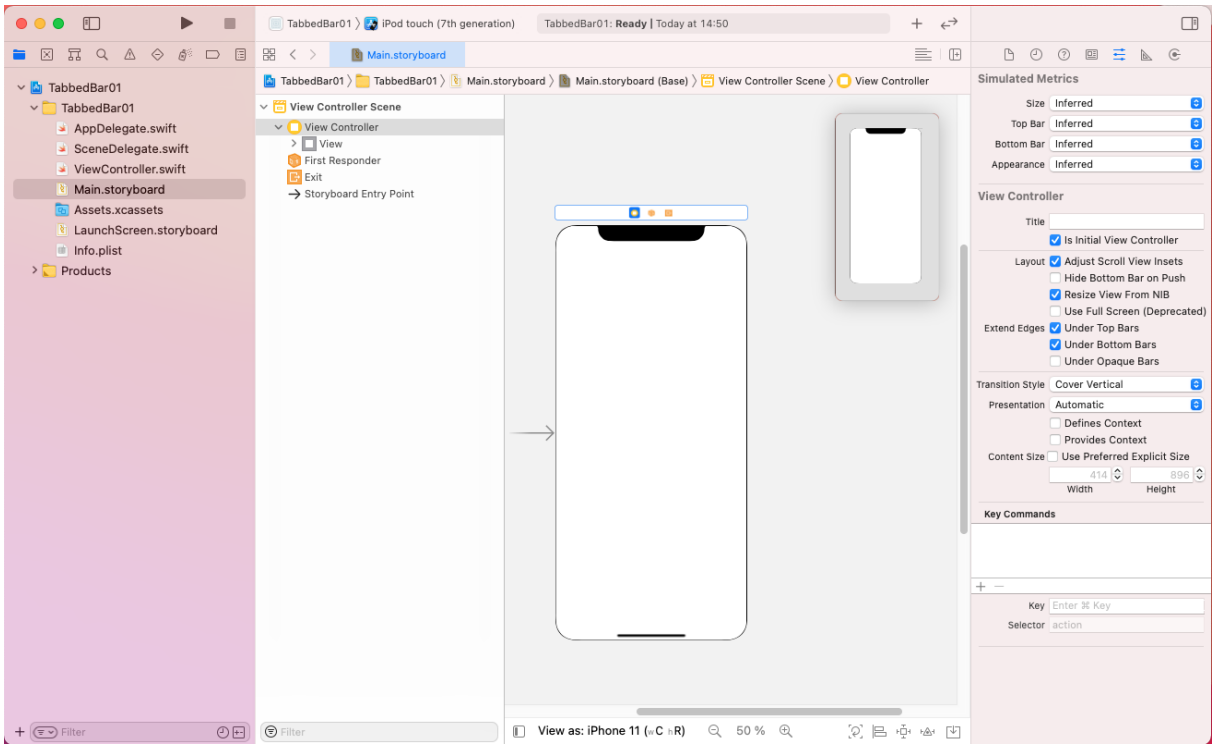


Abbildung 135 Anzeige der normalen App

In den nächsten Schritten wird der einzelne ViewController durch einen TabbedBar-Controller ersetzt. Danach wird ein weiterer ViewController für die TabbedBar eingefügt und beide werden umbenannt (ViewControllerAdd, ViewControllerSub). Am Schluss müssen noch die Symbole eingefügt werden. Danach werden nur noch die bekannten Schritte für die Programmierung benötigt (UI-Elemente einfügen, Code behind).

9.1 Einfügen des TabBar-Controllers

Man klickt in dem vorhandenen ViewController das obere linke Symbol an.Damit signalisiert man, das man diesen ViewController ersetzen will.

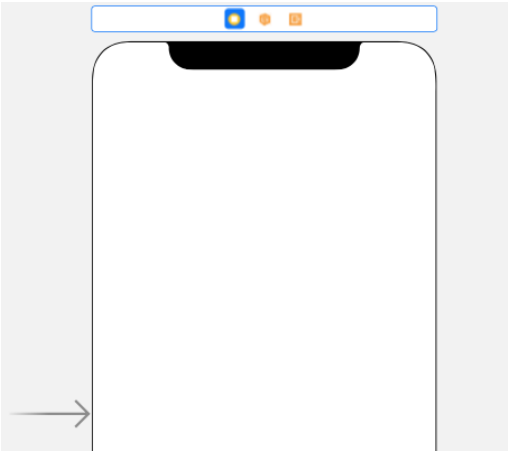


Abbildung 136 Aktivieren des vorhandenen ViewController

Nun ruft man das Menü „Editor“, „Embed In“ und den Eintrag Tab Bar Controller auf.

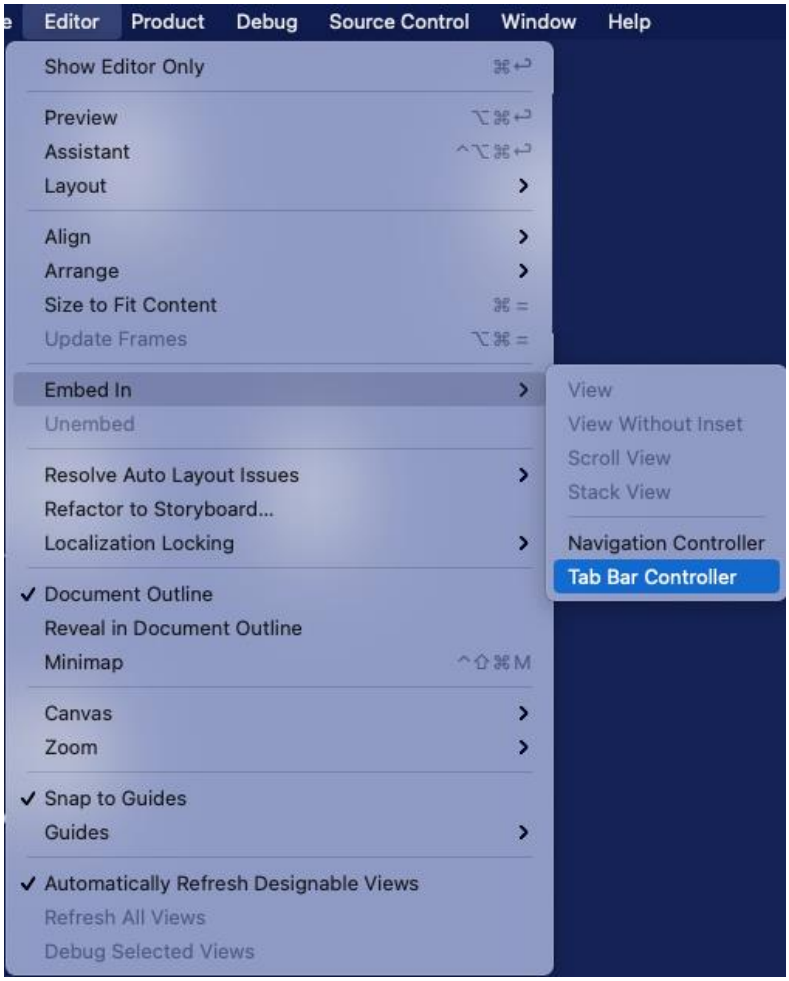


Abbildung 137 Einfügen eines Tab Bar Controllers

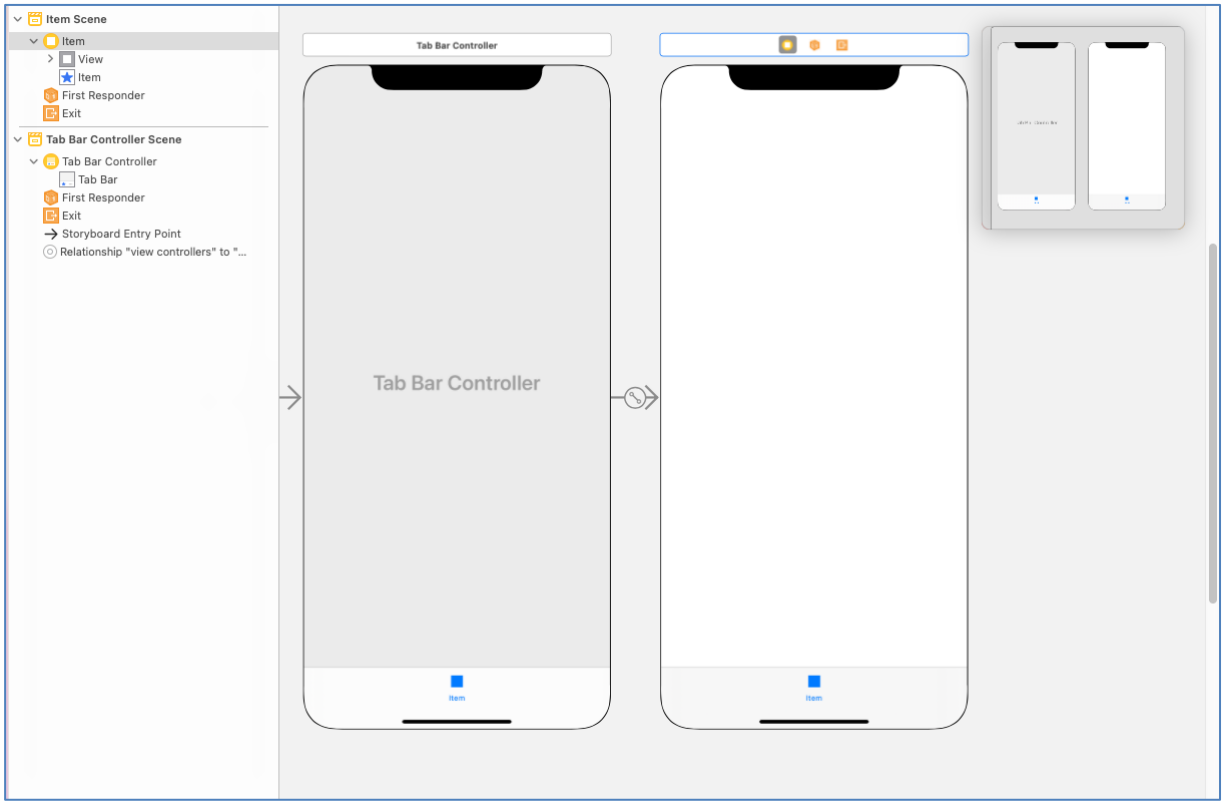


Abbildung 138 DEr Tab Bar Controller

Im nächsten Schritt fügen Sie einen neuen ViewController rechts neben dem ersten ViewController ein.

9.2 Neuer uiViewController

1. Aus der ui-Galerie wird ein uiViewController in den Mainboard.story geschoben. Er wird nicht in den Tab-Bar-Controller geschoben, sondern als „Nachbar“ platziert.

2. Aktivieren Sie den Tab-Bar-Controller . Drücken Sie die Ctrl-Taste und ziehen Sie nun die linke Maustaste auf den neuen ViewController. Damit ist die Verbindung erstellt. Beachten Sie dabei, dass der Zoomfaktor auf 100% stehen sollte, damit man beide Views sehen kann.

3. Nach dem Loslassen des Mauskursors, erscheint ein Dialog, in dem Sie den Typ der Verbindung auswählen müssen. Hier wählen Sie den Eintrag „**view-controllers**“ aus.

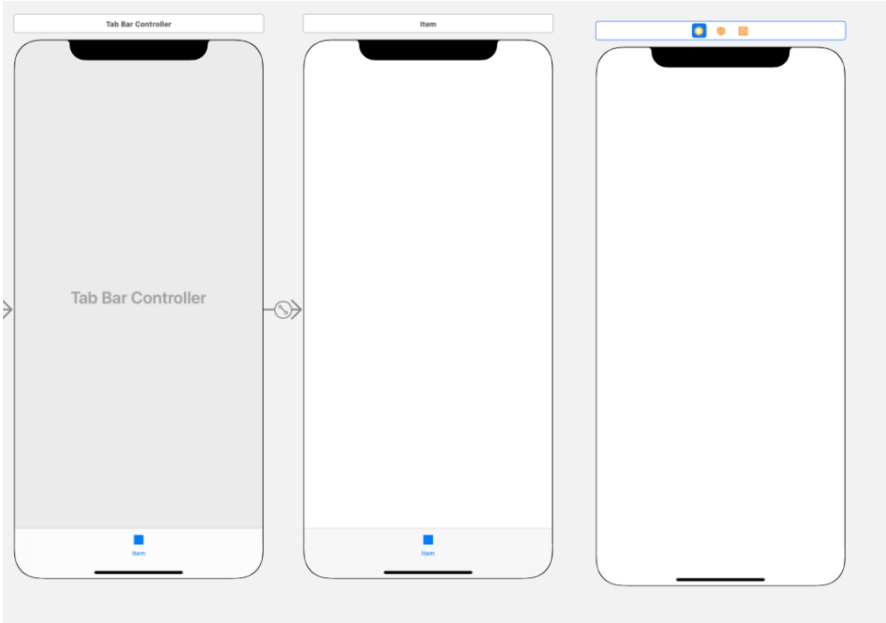
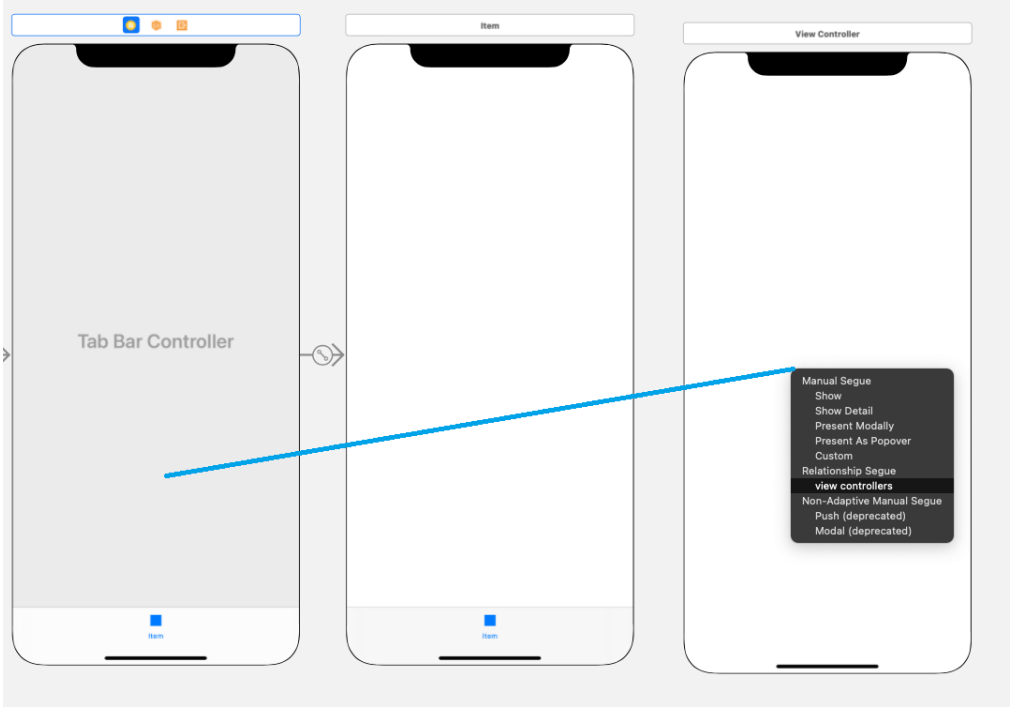


Abbildung 139 Der neue ViewController rechts neben dem ersten.-

Die untere Abbildung zeigt das ziehen zum neuen zum neuen ViewController.



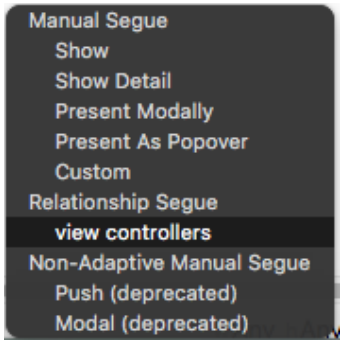


Abbildung 140 Der neue View wird mit dem TabbarController verbunden

Abbildung 141 Anzeige der Verknüpfungen eines Tab-Bars mit zwei ViewControllern

In der unteren Abbildung sieht man die neuen Elemente.

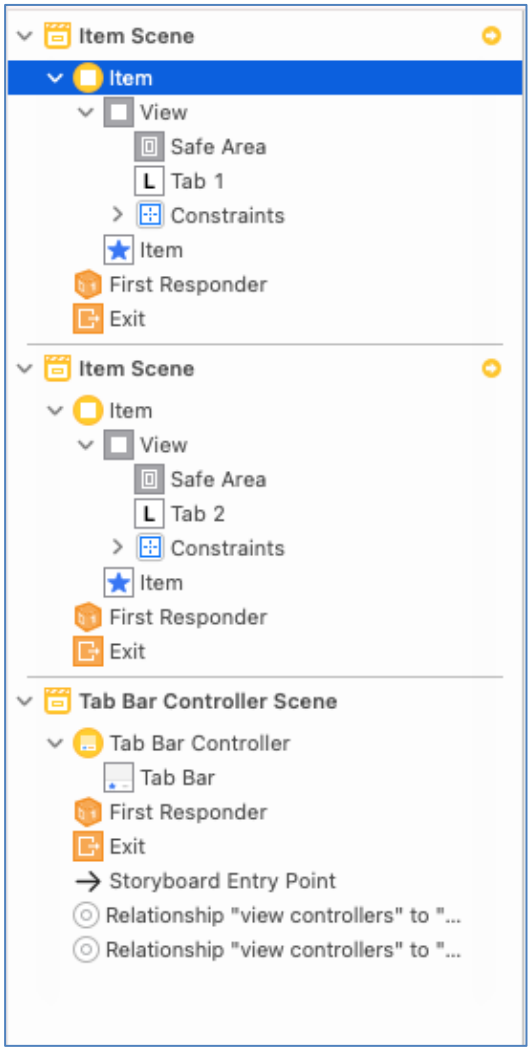


Abbildung 142 Anzeige des Projektes

Nun kann man die Titel ändern (Dazu den passenden View oben anklicken):

- View1 in Addition
- View2 in Subtraktion

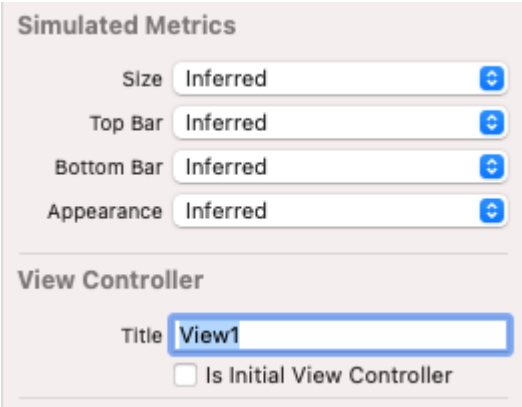


Abbildung 143 Den Titel eines ViewControllers ändern

9.3 Symbole einfügen

Man lädt die Datei „tabbedbarsymbole.zip“ von meiner Homepage

- miwilhelm.de/scripte/ios/ tabbedbarsymbole.zip

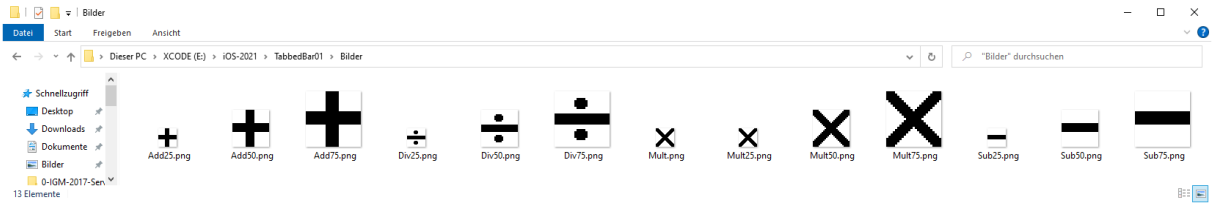


Abbildung 144 Symbole für das TabbedBar-Projekt

Ablauf:

- Öffnen Sie das Fenster für den Projekt-Eintrag „Assets.xcassets“ bzw. Images.xcassets..
- Öffnen Sie den Finder und den Ordner, in dem die Bilder eingetragen sind.
  - **Die Bilder müssen einfarbig sein.**
  - **Die Farbe der Symbole spielt keine Rolle, es wird immer die blaue Farbe verwendet.**
  - **Der Hintergrund muss transparent sein.**
  - **Es sollten drei Bilder vorhanden sein:**
    - 25×25 Pixel
    - 50×50 Pixel
    - 75×75 Pixel
- Ziehen Sie das **erste** Bild in den Assets.xcassets bzw. Images.xcassets.. Der linke Bereich in der oberen Abbildung.
- Ziehen Sie die beiden restlichen Bilder in den Bereichen 2x und 3x.
- Alternativ können Sie auch eine PDF-Datei mit einer SVG-Grafik einfügen. Diese wird dann skaliert.
- Prinzipiell können Sie auch nur ein Symbol verwenden. Dieses wird dann hochskaliert.
- Damit können im Eigenschaftsdialog das/die Bilder auswählen. Beachten Sie, dass man die **untere** Eingabe verwendet.

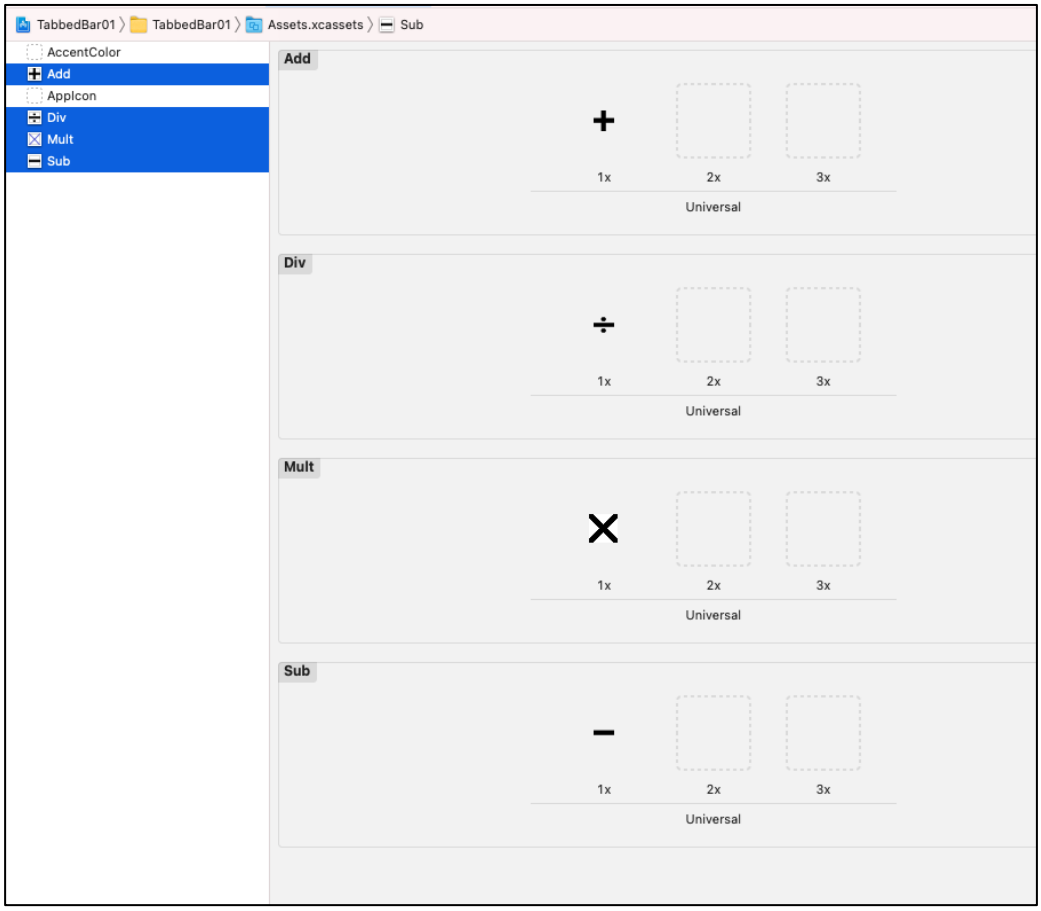


Abbildung 145 Anzeige der eingetragenen Bilder (jeweils eins)

Nun kann man

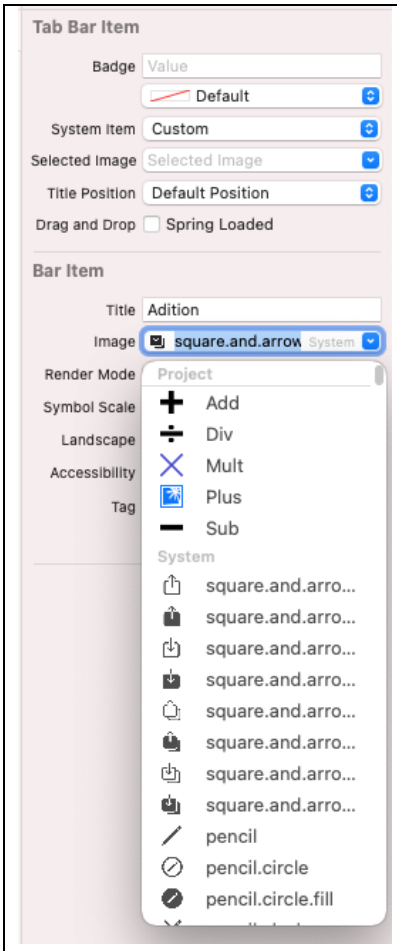


Abbildung 146 Auswahl eines Symbols

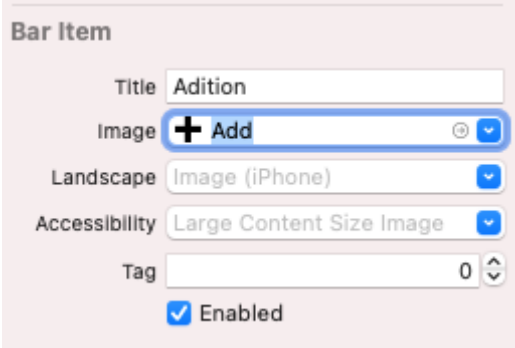


Abbildung 147 Anzeige des Symbols

9.4 Erstellen einer Swift-Klasse für den zweiten ViewController

Mit dem Menü „File“ und Eintrag „File“ wird eine neue Datei erstellt.

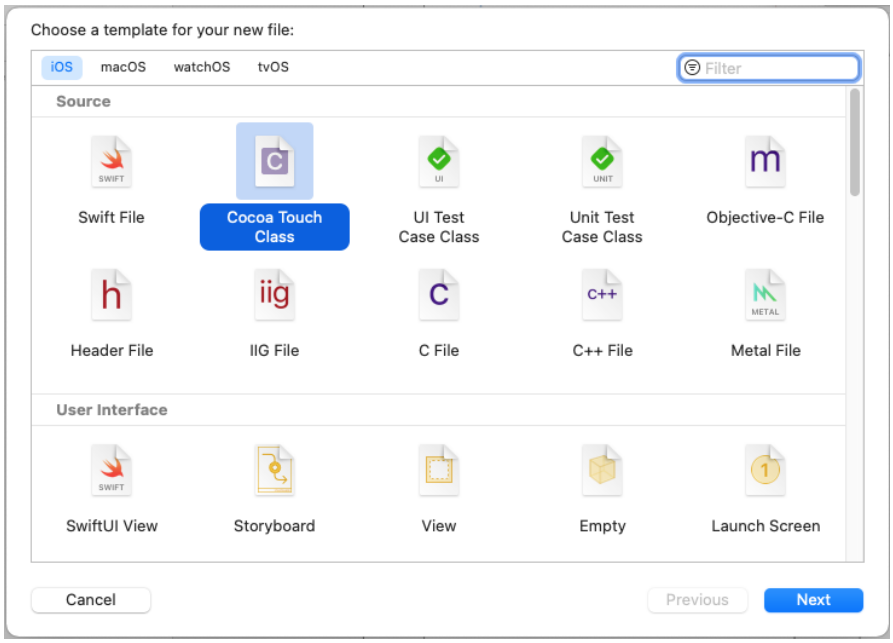


Abbildung 148 Auswahl des Dateityps

Nun den Namen eingeben:

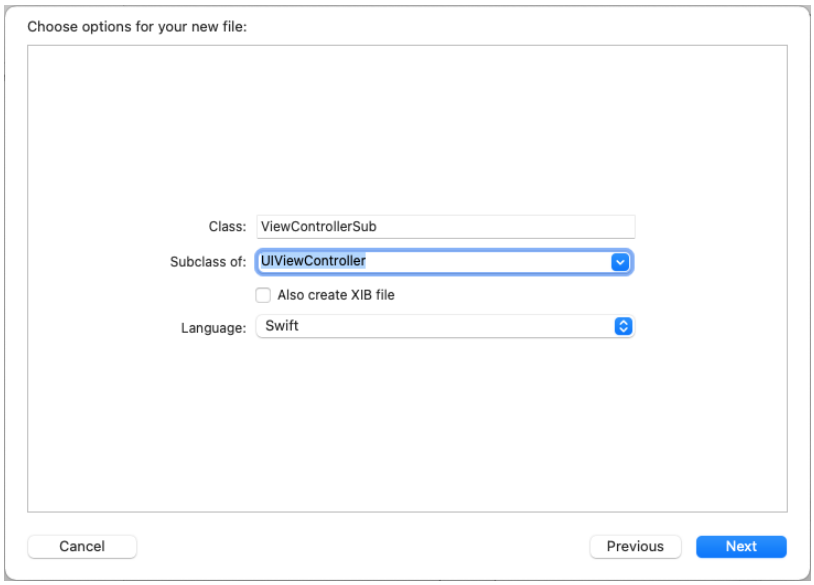


Abbildung 149 Der Name und die Oberklasse für die neue Datei

Nun wird die neue Datei erstellt:

```
class ViewControllerSub: UIViewController {
    override func viewDidLoad() {
        super.viewDidLoad()
    }
}
```

Nun muss, sollte, noch der Name des ersten ViewController in „ViewControllerAdd“ geändert werden. Im letzten Schritt muss noch die Verbindung des zweiten ViewControllers zu der zweiten Datei eingetragen werden. Dazu klickt man in SubController den linken der drei Schalter an.

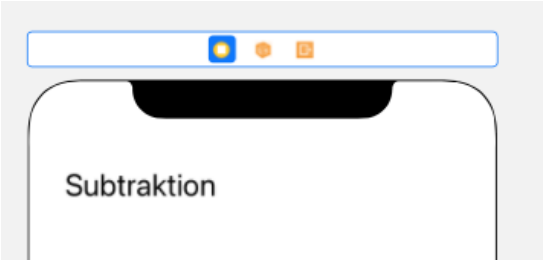


Abbildung 150 Aktivieren des SubControllers

Nun kann man im rechten Property-Fenster die Klasse auswählen.

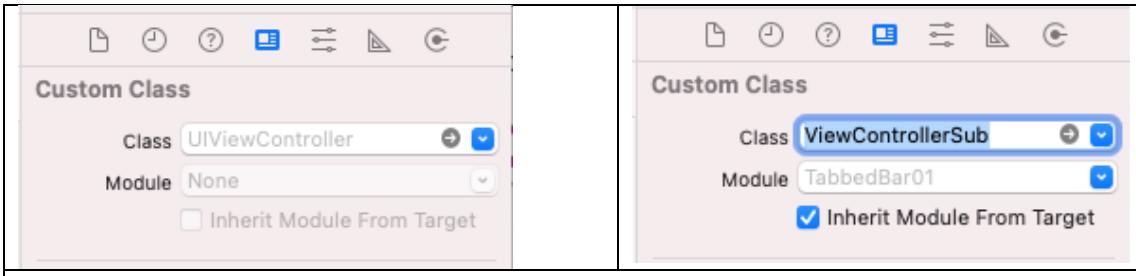


Abbildung 151 Swiftklasse für einen ViewController setzen

Die nächsten Schritte sind die bekannten:

- Einfügen der UI-Elemente
- Referenzen holen
- Events eintragen
- Aktion programmieren

Abbildung 152 Contraints für die Textzeilen (Breite beachten)

Abbildung 153 Layout des 1. Tabs

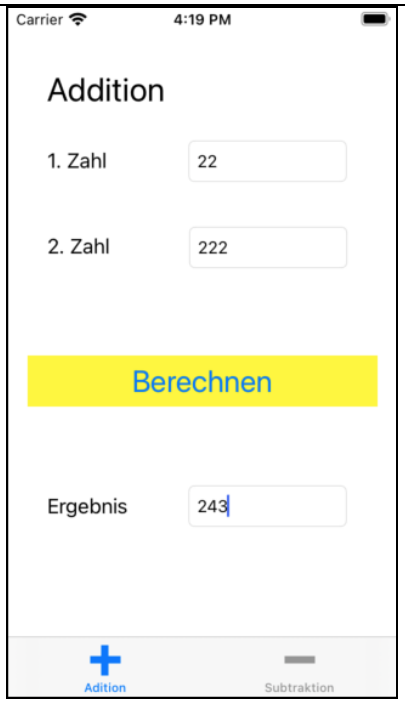


Abbildung 154 Anzeige im Simulator

Quellcode für die Addition:

```
class ViewControllerAdd: UIViewController {

    @IBOutlet var tnumber1: UITextField!
    @IBOutlet var tnumber2: UITextField!
    @IBOutlet var tresult: UITextField!

    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view.
       tnumber1.text = "22"
       tnumber2.text = "12"
    }

    @IBAction func bncalcclick(_ sender: Any) {
        let number1 = Double(tnumber1.text!)
                if (number1==nil) {
                    tresult.text = "Fehlerhafte 1. Zahl"
                    return
                }
                let number2 = Double(tnumber2.text!)
                if (number2==nil) {
                    tresult.text = "Fehlerhafte 2. Zahl"
                    return
                }
                tresult.text = String(number1!+number2!)
    }  // bncalcclick

}
```

9.5 Komplexere TabbedBar Variante

Dieses Kapitel zeigt die Verwendung einer TabbedBar in einer TabbedBar. Damit kann man noch flexibler die Struktur aufbauen.

Ablauf:

- Neues Projekt erstellen
- Einfügen und Ersetzen eines TabbedBar-Controllers
  - Dieser hat die Tab's
    - Addition
    - Substraktion
- Einfügen eines weiteren TabbedBar-Controllers
  - Dieser hat die Tab's
    - Primzahlen
    - Fibonacci
    - Wertetabelle
- Verknüpfen des ersten TabbedBar-Controllers mit dem zweiten TabbedBar-Controllers

9.5.1 Einfügen des TabBar-Controllers

Man klickt in dem vorhandenen ViewController das obere linke Symbol an.Damit signalisiert man, das man diesen ViewController ersetzen will.

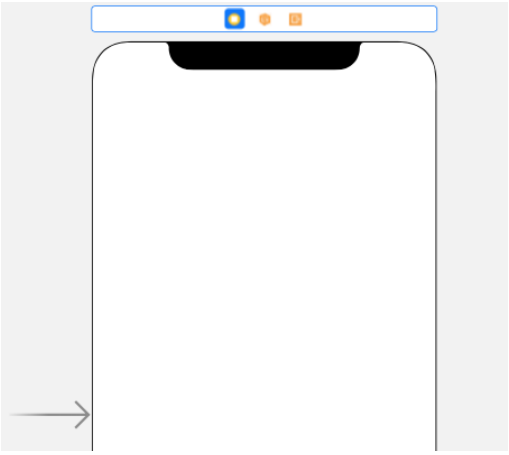


Abbildung 155 Aktivieren des vorhandenen ViewController

Nun ruft man das Menü „Editor“, „Embed In“ und den Eintrag Tab Bar Controller auf.

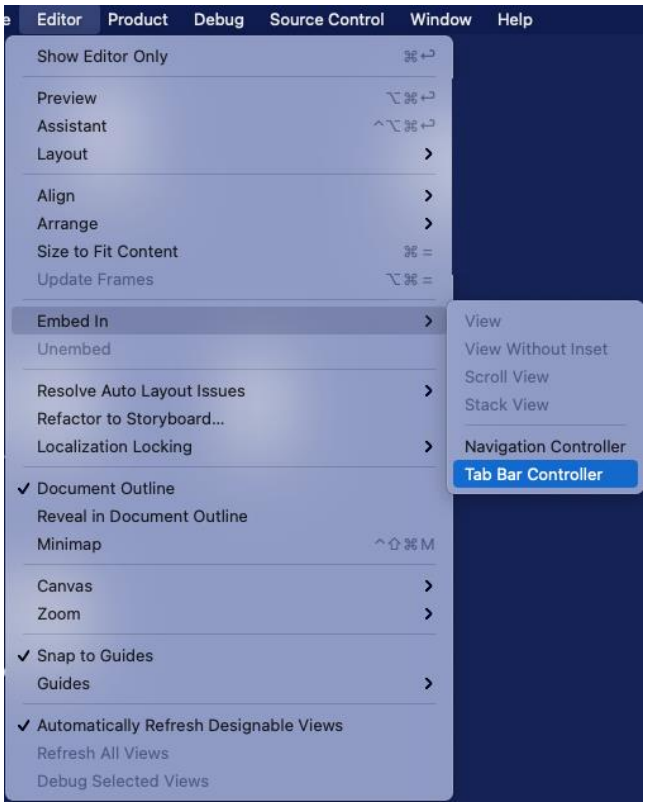


Abbildung 156 Einfügen eines Tab Bar Controllers

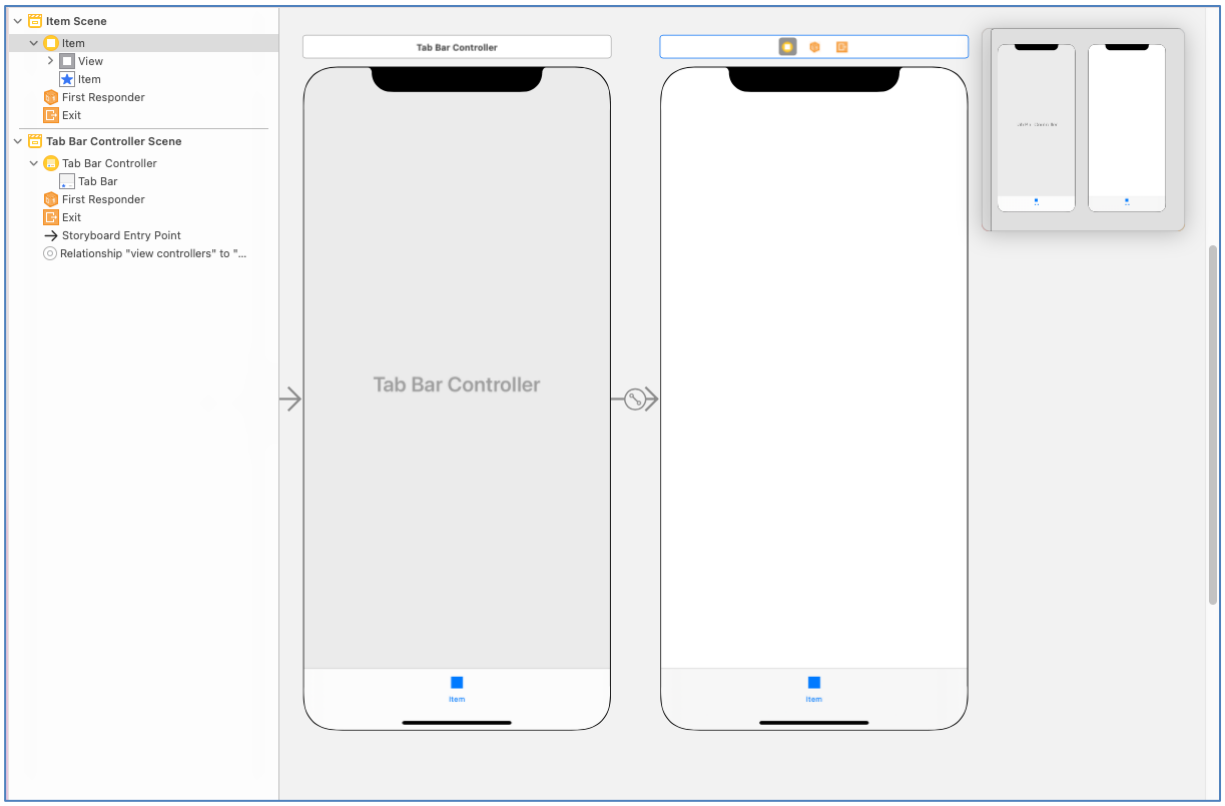


Abbildung 157 Der Tab Bar Controller

Im nächsten Schritt fügen Sie einen neuen ViewController rechts neben dem ersten ViewController ein.

9.5.2 Neuer uiViewController

1. Aus der ui-Galerie wird ein uiViewController in den Mainboard.story geschoben. Er wird nicht in den Tab-Bar-Controller geschoben, sondern als „Nachbar“ platziert.

2. Aktivieren Sie den Tab-Bar-Controller . Drücken Sie die Ctrl-Taste und ziehen Sie nun die linke Maustaste auf den neuen ViewController. Damit ist die Verbindung erstellt. Beachten Sie dabei, dass der Zoomfaktor auf 100% stehen sollte, damit man beide Views sehen kann.

3. Nach dem Loslassen des Mauskursors, erscheint ein Dialog, in dem Sie den Typ der Verbindung auswählen müssen. Hier wählen Sie den Eintrag „**view-controllers**“ aus.

Alternativ kann man auch einen Tab-Bar-Controller per UI-Dialog einfügen. Dann hat man automatisch zwei Childs. Nun muss man aber den StartViewController neu setzen. Dazu klickt man den blauen waagerechten Pfeil an und verschiebt ihn zum neuen Tab-Bar-Controller. Dann kann man den alten ViewController löschen.

Nun fügen Sie die Beschriftungen, die Bilder und die UI-Elemente ein. Danach werden die Swift-Klassen erstellt und der restliche Quellcode eingefügt.

9.6 Zweiter Tab-Bar-Controller

Über den UI-Dialog fügt man nun einen zweiten Tab-Bar-Controller ein. Dann hat man automatisch zwei Childs. Dann fügt man einen weiteren UIViewController hinzu und fügt zum zweiten Tab-Bar-Controller. Per Drag&Drop wird nun der erste zweiten Tab-Bar-Controller mit dem zweiten zweiten Tab-Bar-Controller verknüpft.

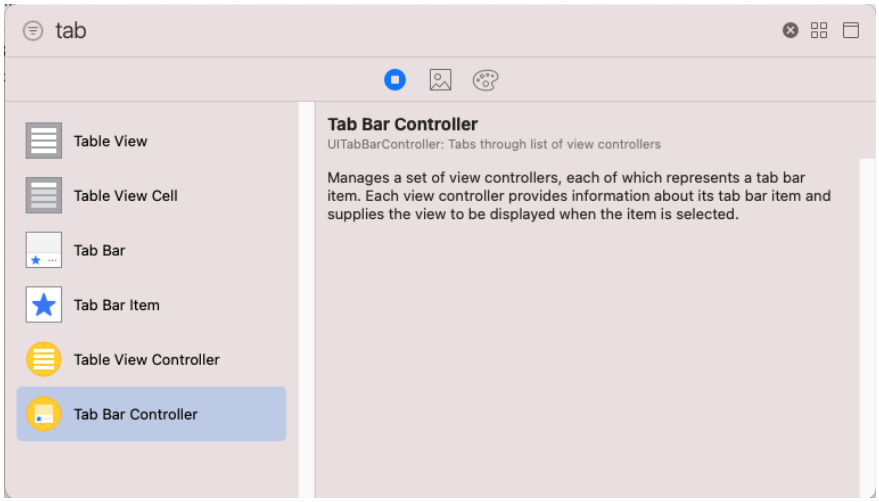


Abbildung 158 UI-Dialog mit einem Tab-Filter

Die untere Abbildung zeigt die ViewController bevor die beiden Tabbed-Bar verknüpft wurden.

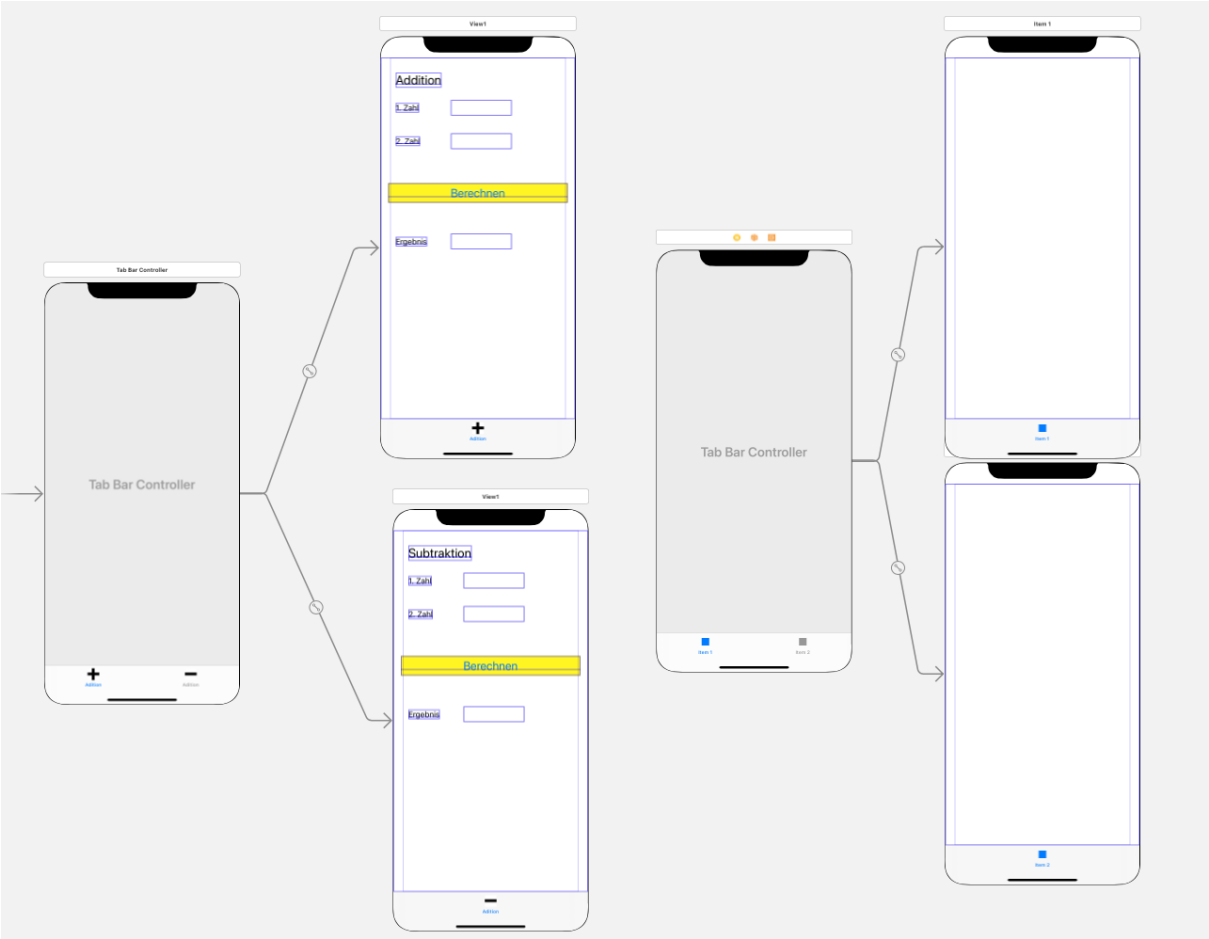


Abbildung 159 Layout der App vor der Verknüpfung der beiden Tabbed-Bars

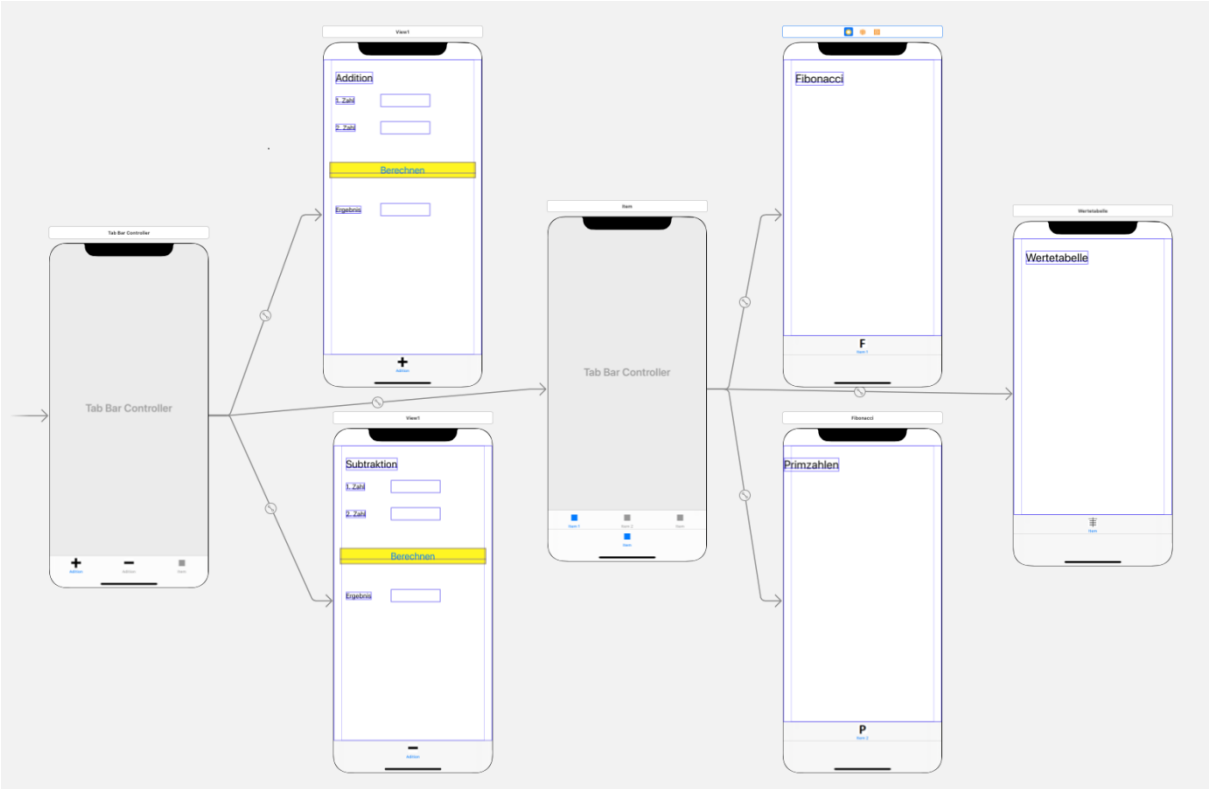
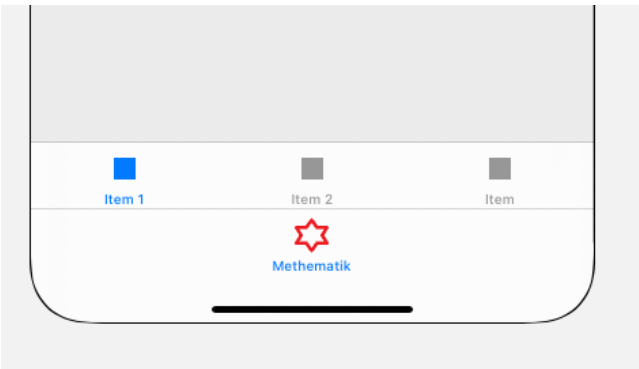
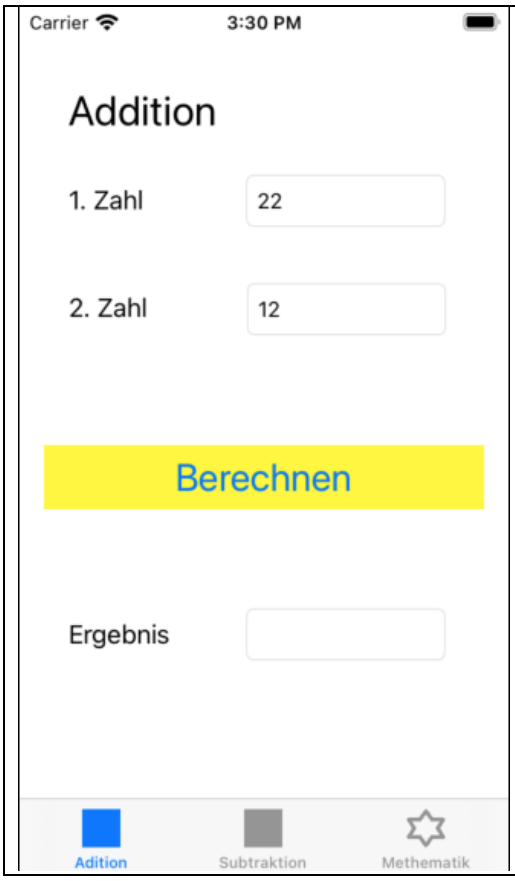


Abbildung 160 Layout der App nach der Verknüpfung der beiden Tabbed-Bars

Man beachte die Linie zwischen den beiden Tabbed-Bars und die zwei Zeilen in der zweiten Tabbed-Bar (siehe untere Abbildung).




Carrier
3:30 PM
Addition
1. Zahl
22
2. Zahl
12
Berechnen
Ergebnis
Adition
Subtraktion
Methematik

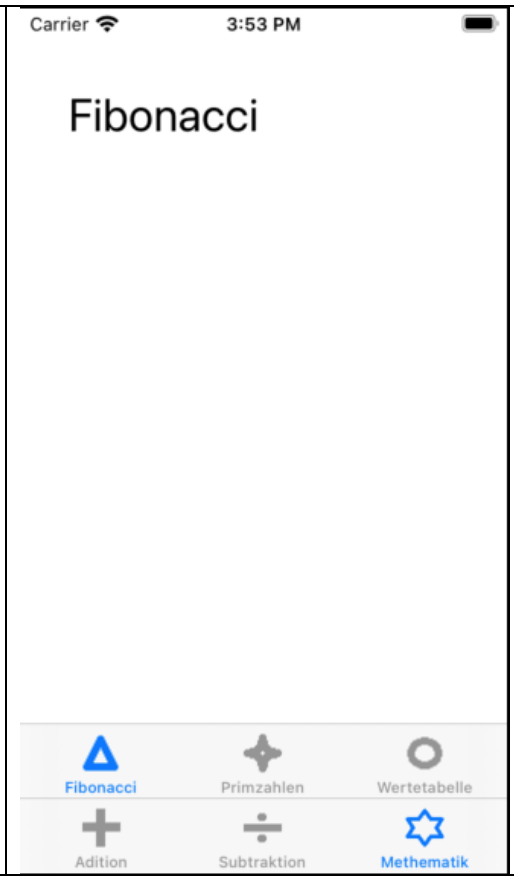

Carrier
3:53 PM
Fibonacci
Fibonacci
Primzahlen
Wertetabelle
Adition
Subtraktion
Methematik

Abbildung 161 Tabbed-Bar mit zehn ViewController

10 Segues

Bei der Verwendung der „Segues“-Technik, Nachfolger, muss man die Verwaltung teilweise selbst vornehmen. Auch die „Zurück-Technik“ ist aufwändiger.

10.1 Anlegen einer App

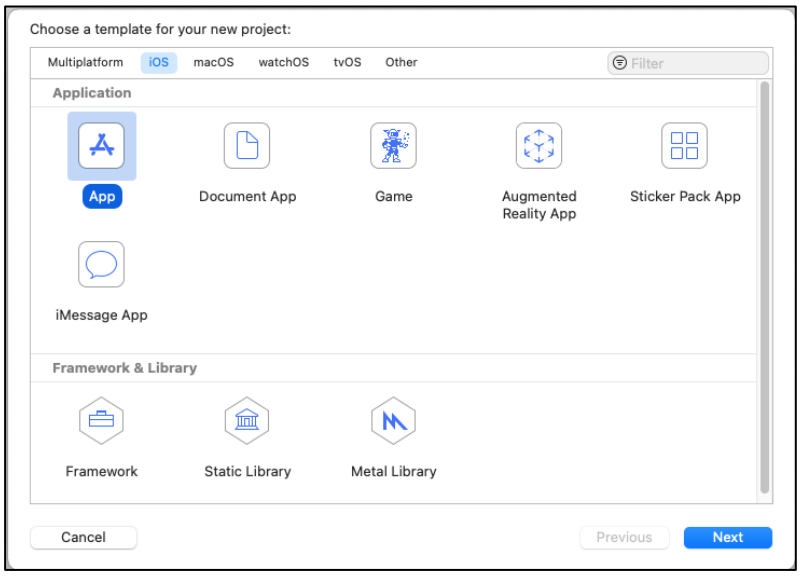


Abbildung 162 Anlegen einer App

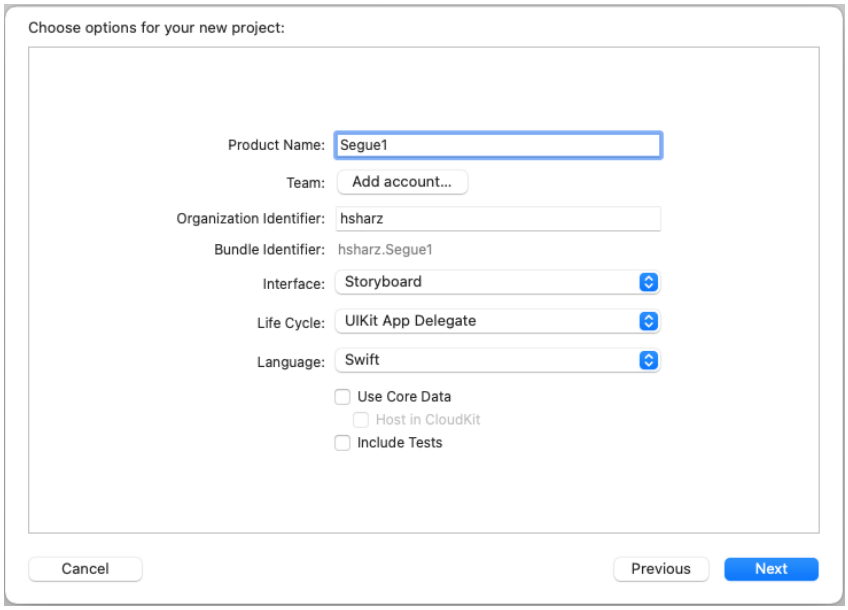


Abbildung 163 Name des Projekts

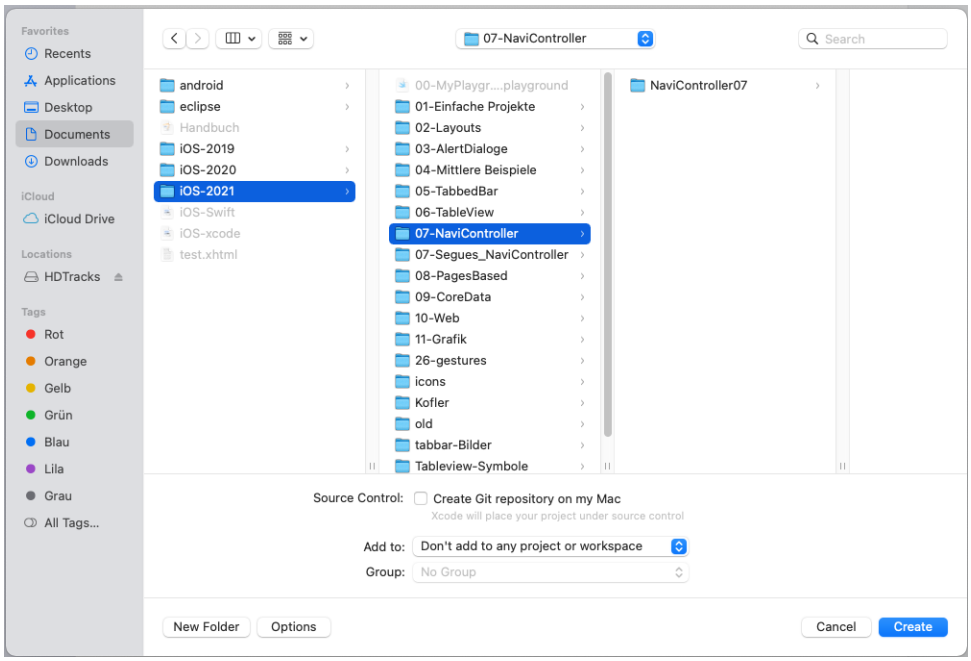


Abbildung 164 Speichern der App in einem Ordner

Nun sollte das Projekt wie in der unteren Abbildung aussehen:

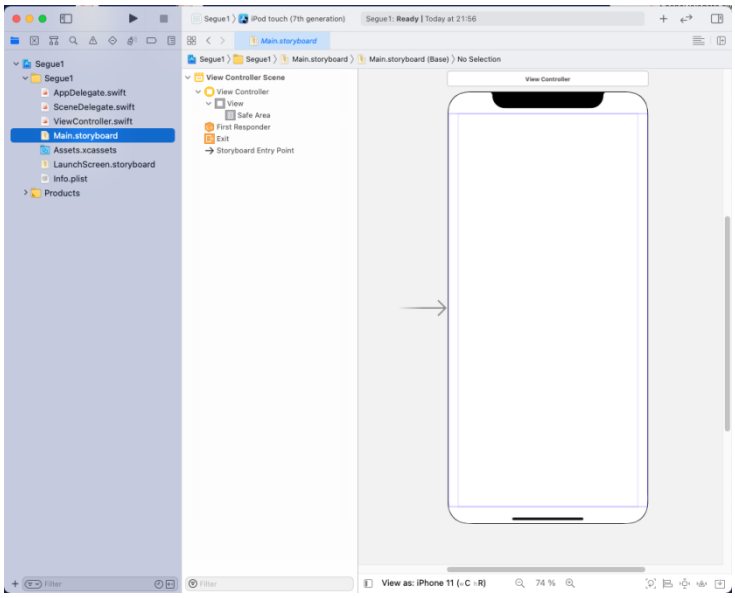


Abbildung 165 Aktueller Stand

10.2 Vier UIViewController erstellen

Nun werden vier UIViewController aus dem UILibDialog in das Main.storyboard eingefügt.

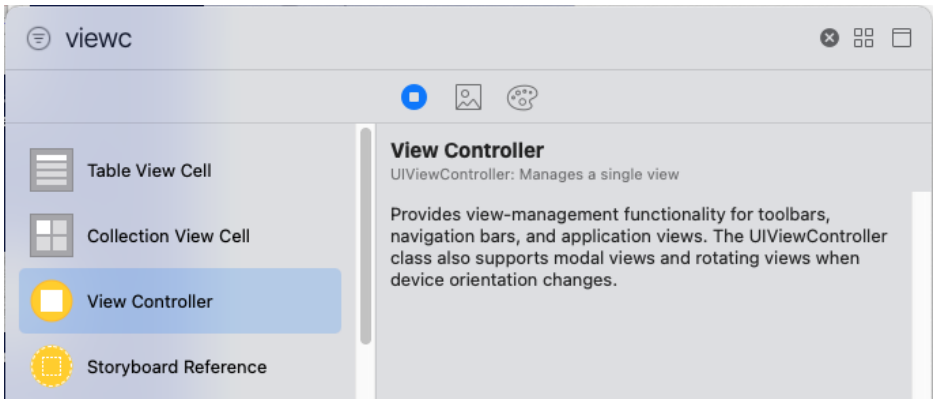


Abbildung 166 Filter für einen UIViewController

Nach dem Einfügen sollte das Projekt jetzt so aussehen:

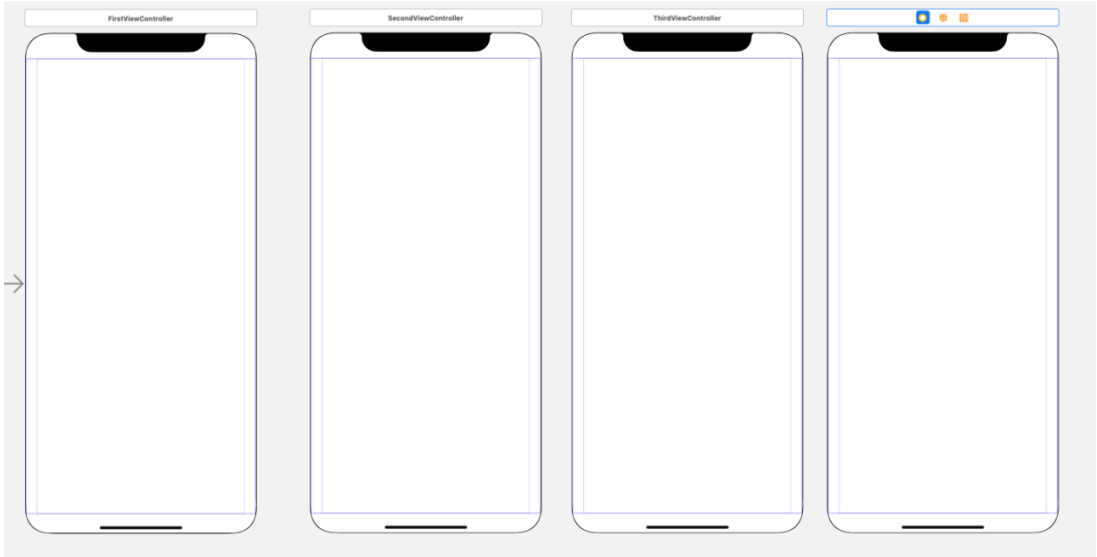


Abbildung 167 Vier UIViewController als Seque

Die UIViewController benötigen nun noch jeweils eine zugehörige Swift-Datei. Der erste UIViewController hat ja schon eine zugehörige Datei.

Als erstes ändert man den Namen des ersten ViewController in“FirstViewController“:

- Anklicken des Controllers
- Rechts oben den Titel ändern
- Nun den Dateinamen und den Klassennamen in FirstViewController.swift ändern.

Für die restlichen drei ViewController benötigt man jeweils eine neue Klasse. Dazu Cmd+N drücken:

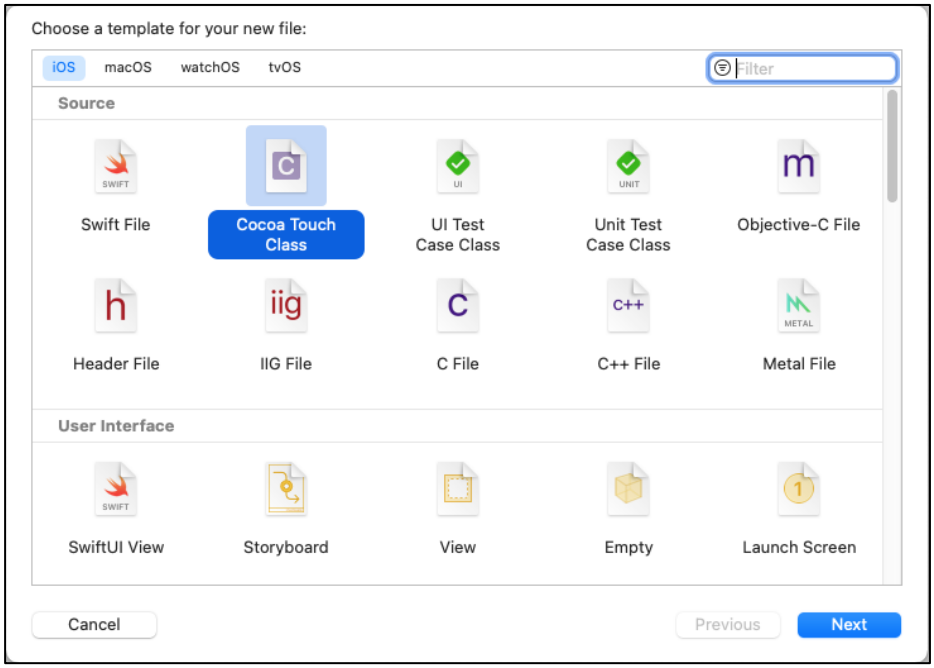


Abbildung 168 Klassendialog

Nun trägt man den Namen „SecondViewController“ ein:

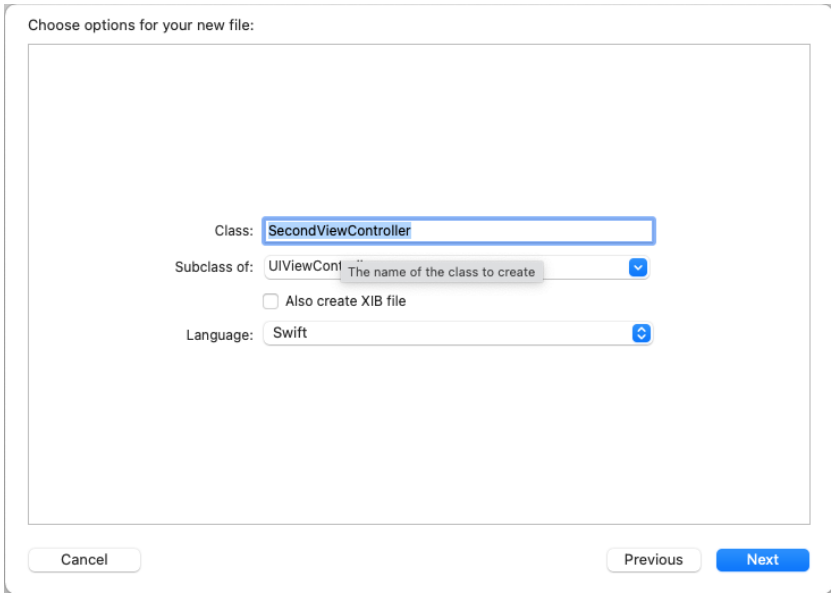


Abbildung 169 SecondViewControllerdatei erstellen

Das gleiche mit dem dritten ViewController:

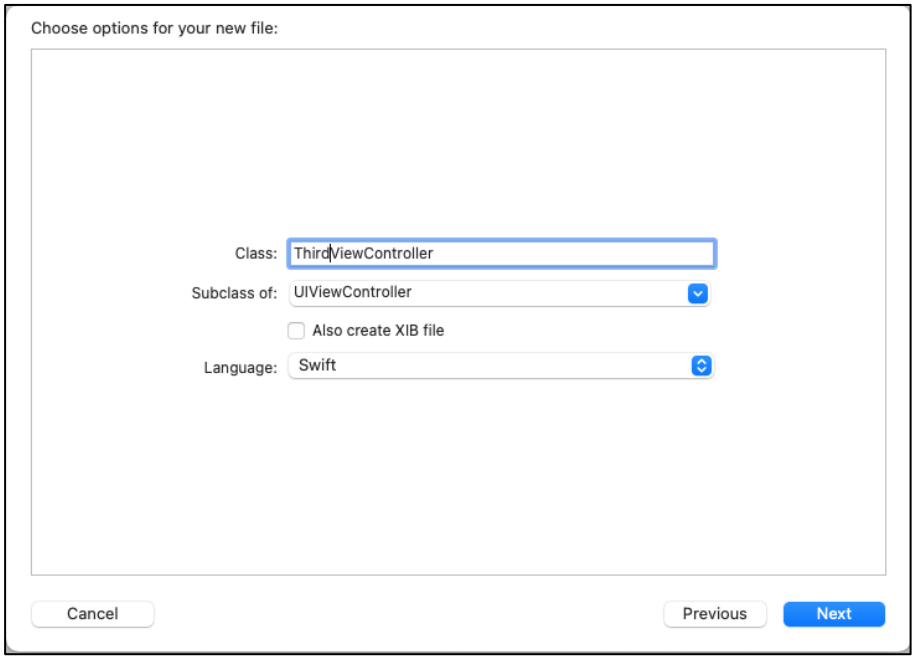


Abbildung 170 ThirdViewControllerdatei erstellen

Und nun die letzte Swift-Datei:

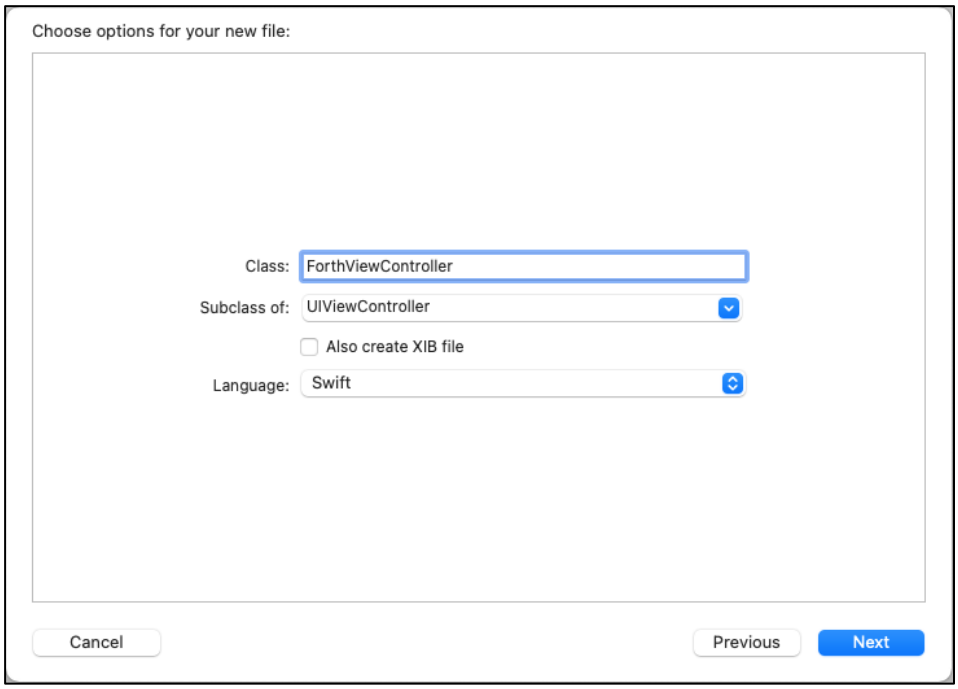


Abbildung 171 ForthViewControllerdatei erstellen

In der unteren Abbildung ist die vierte Datei dargestellt:

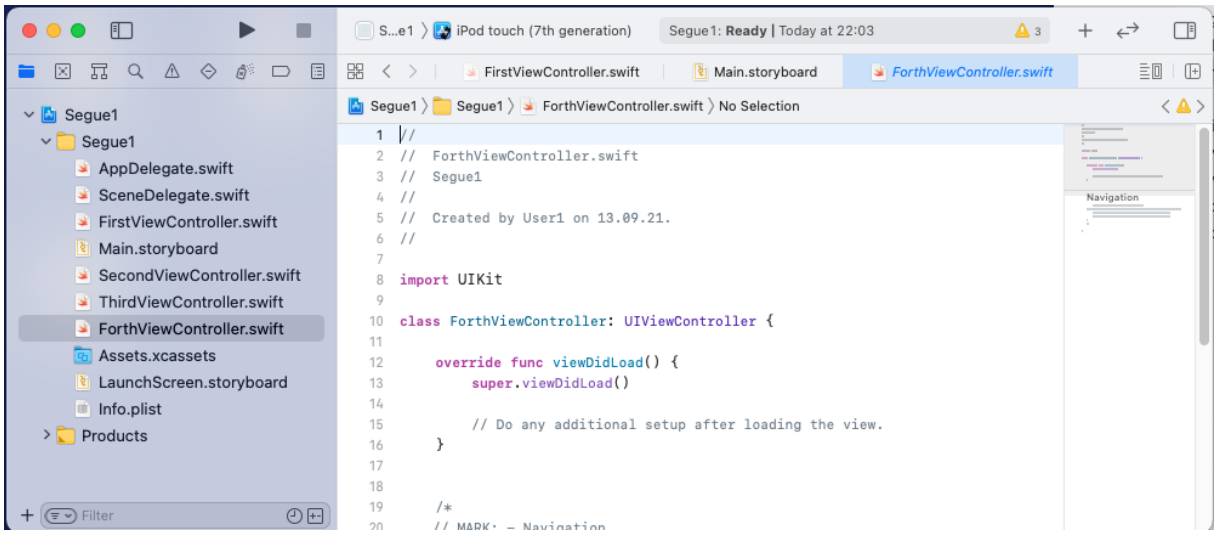
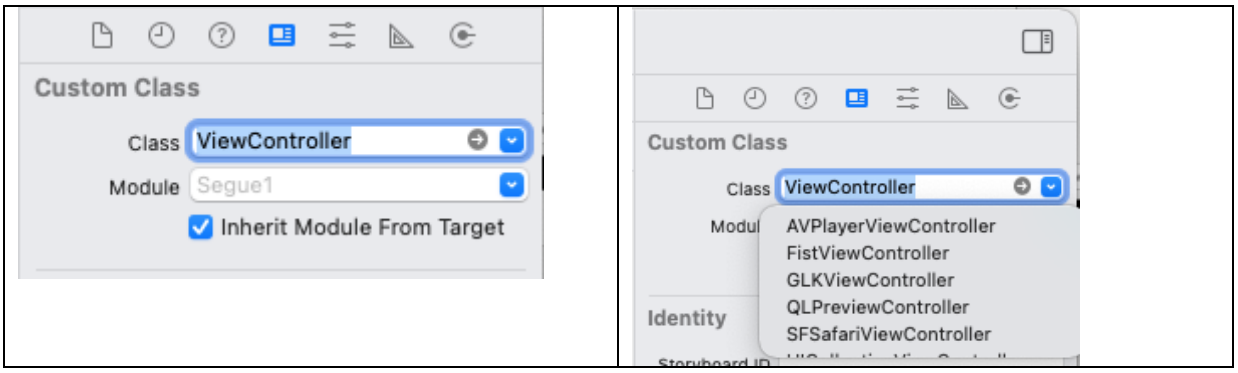


Abbildung 172 ForthViewController.swift

Nun sind alle Swift-Datei erstellt, aber noch nicht den einzelnen UIViewControllern zugeordnet:

- Aktivieren des ersten ViewControllers
- Rechts oben im Propertybaum den Eintrag „ViewController“ in „FirstViewController“ ändern. Man muss nur den Eintrag aus der Liste auswählen.



Das gleiche Schema gilt für den zweiten, dritten und vierten UIViewController.

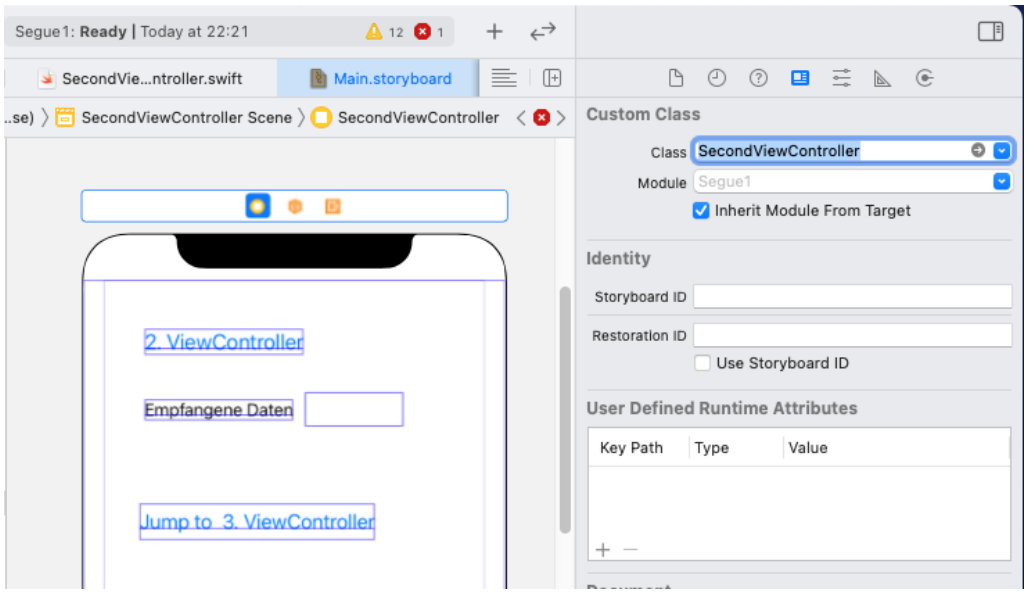


Abbildung 173 Verknüpfung zur Datei „SecondViewController.swift“

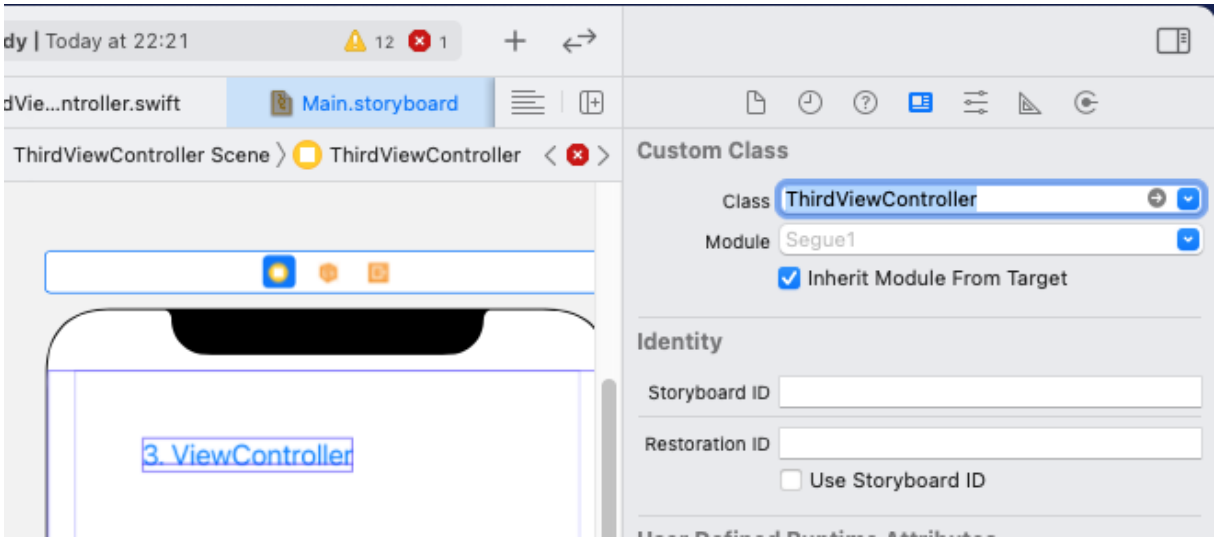


Abbildung 174 Verknüpfung zur Datei „ThirdViewController.swift“

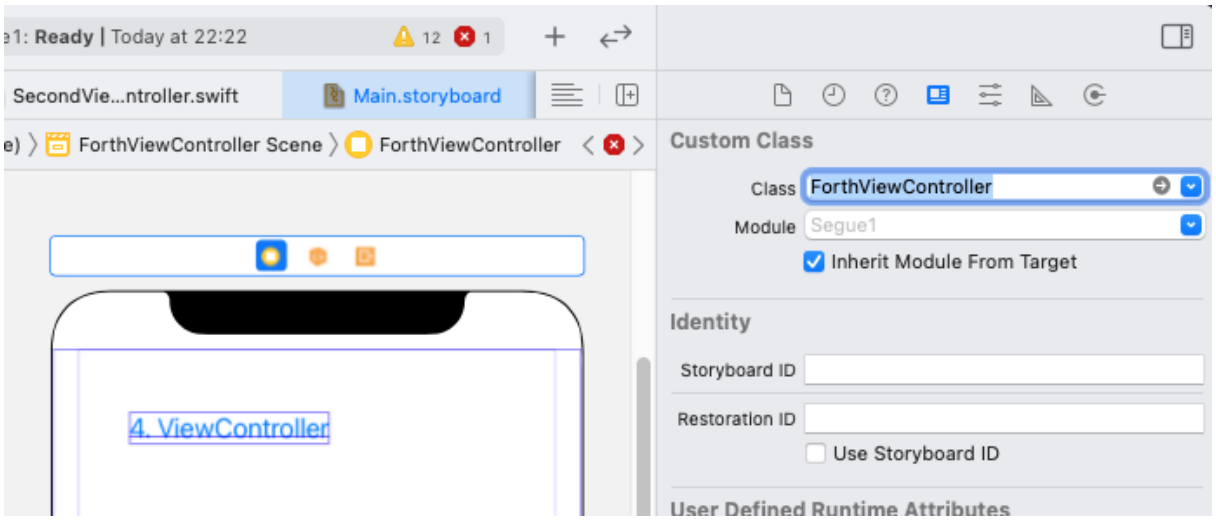


Abbildung 175 Verknüpfung zur Datei „ForthViewController.swift“

10.3 Einfügen der UI-Elemente

Nun werden die UI-Elemente für alle vier ViewController eingefügt.

10.3.1 Aufbau des ersten ViewControllers

Abbildung 176 UIElemente des 1. ViewControllers

10.3.2 Aufbau des zweiten ViewControllers

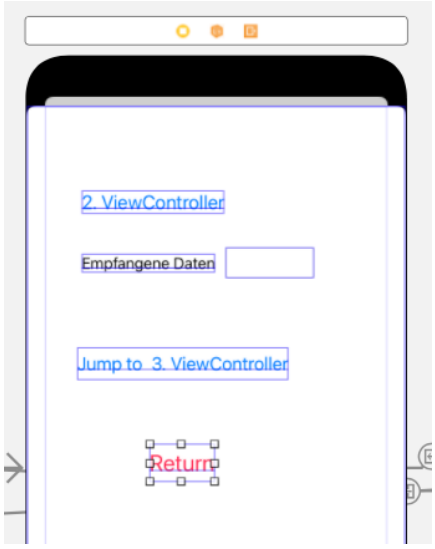


Abbildung 177 UIElemente des 2. ViewControllers

10.3.3 Aufbau des dritten ViewControllers

Abbildung 178 UIElemente des 3. ViewControllers

10.3.4 Aufbau des vierten ViewControllers

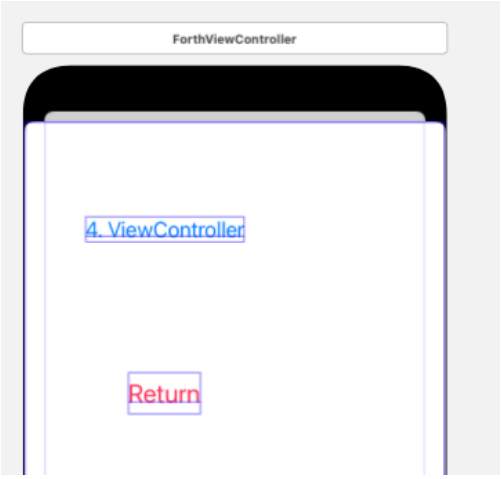


Abbildung 179 UIElemente des 4. ViewControllers

10.4 Referenzen erstellen

Für dieses Beispiel braucht man nur zwei Referenzen erstellen. Man erstellt für jede Eingabezeile eine Referenz:

10.4.1 Referenz des ersten ViewControllers

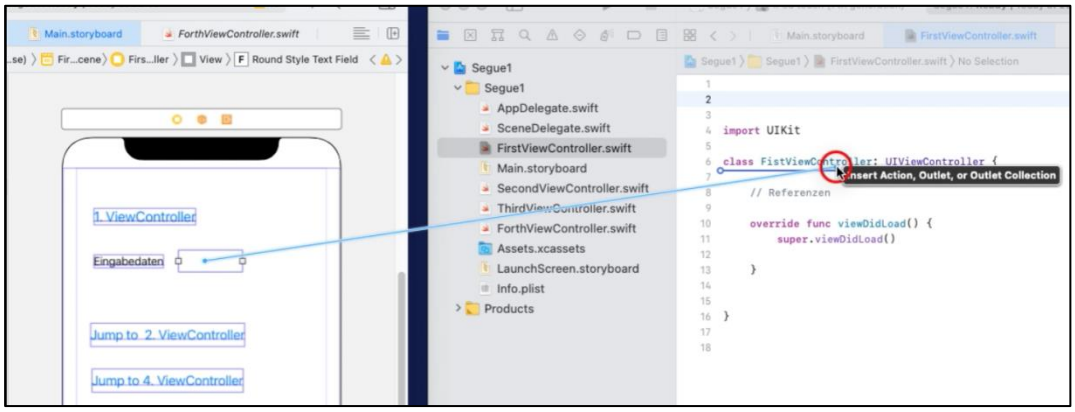


Abbildung 180 Referenz der Eingabezeile im 1. ViewController

Nun noch den Namen vergeben:

Aktueller Stand des ersten ViewController.swift

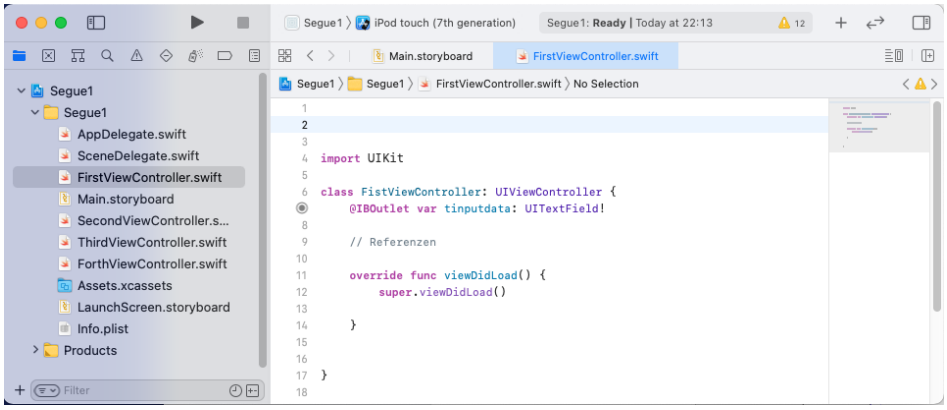


Abbildung 181 Quellcode für den ersten ViewController

10.4.2 Referenz des zweiten ViewControllers

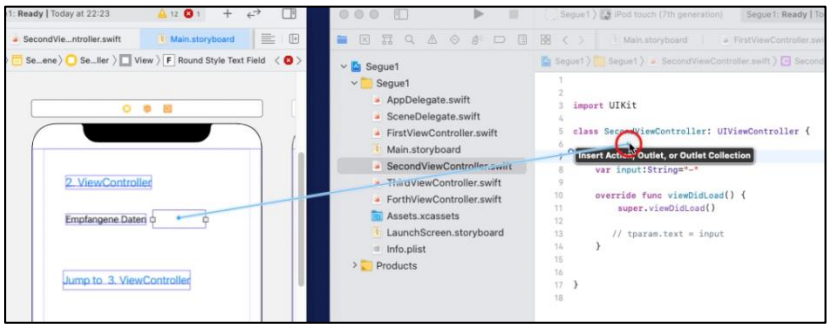
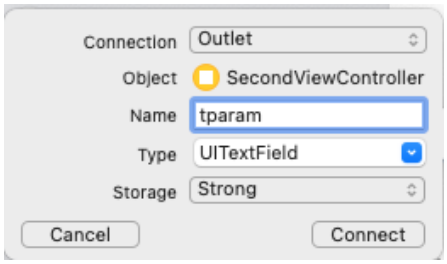


Abbildung 182 Referenz der Eingabezeile im 2. ViewController

Nun noch den Namen vergeben:



Aktueller Stand des zweiten ViewController.swift:

```
class SecondViewController: UIViewController {

    @IBOutlet var tparam: UITextField!

    override func viewDidLoad() {
        super.viewDidLoad()
```

```
        }

}
```

Abbildung 183 Quellcode für den zweiten ViewController

10.5 Interne Variable verarbeiten

Die Navigation geht vom ersten ViewController zum zweiten ViewController. Dabei wird der Inhalt des Textfeldes zum zweiten ViewController mittels einer Variable, **input**, **im** zweiten ViewController übertragen!

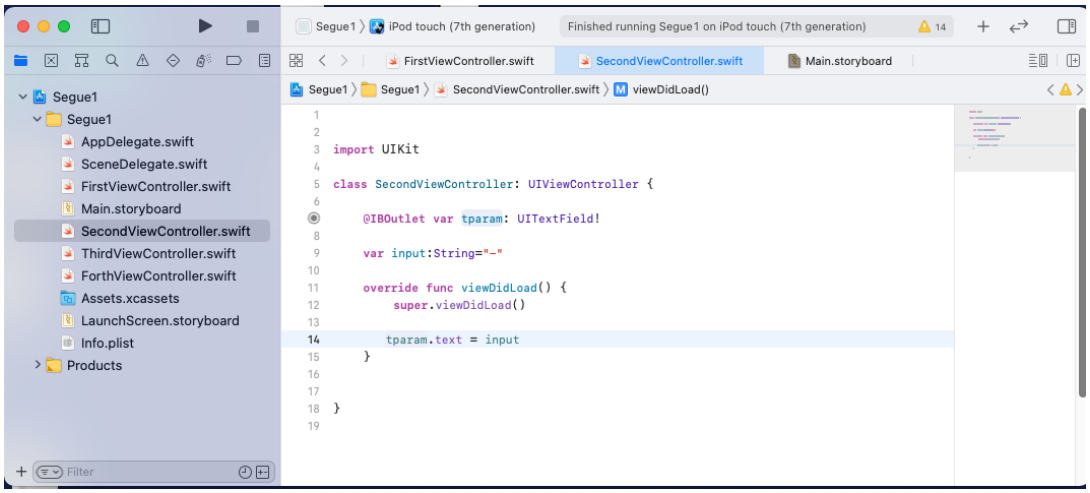


Abbildung 184 Der Parameter wird im 2. ViewController verarbeitet

10.6 Links der Schalter setzen (Segue)

Nun wird das bekannte Drag&Drop für den Aufruf vom ersten zum zweiten ViewController benutzt:

- Anklicken des Schalters im ersten ViewController
- Ctrl-Taste drücken
- Ziehen zum zweiten ViewController
- Loslassen und „Show“ wählen

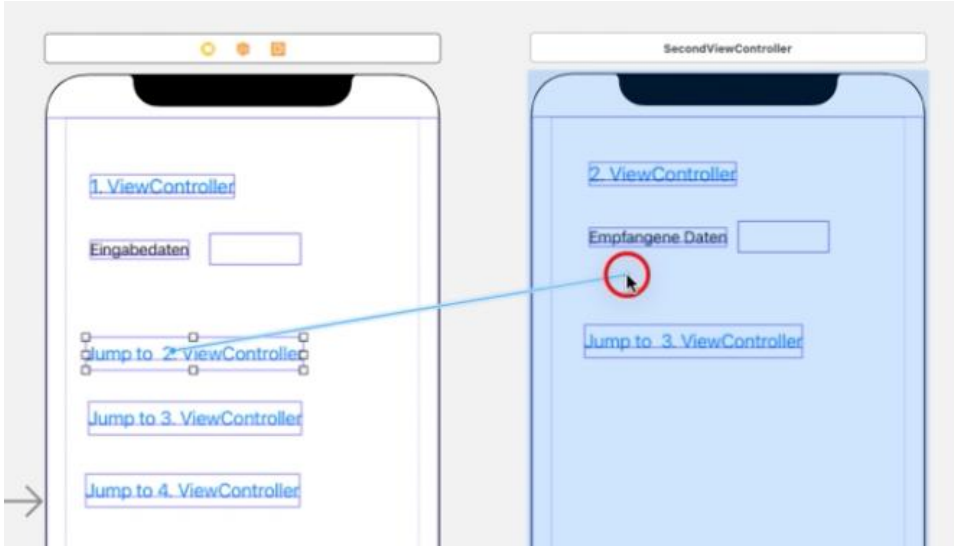


Abbildung 185 Link vom ersten zum zweiten ViewController

Nun die Auswahl des Anzeigemodus:

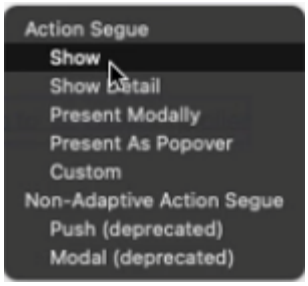


Abbildung 186 Anzeigemodus des zweiten ViewControllers

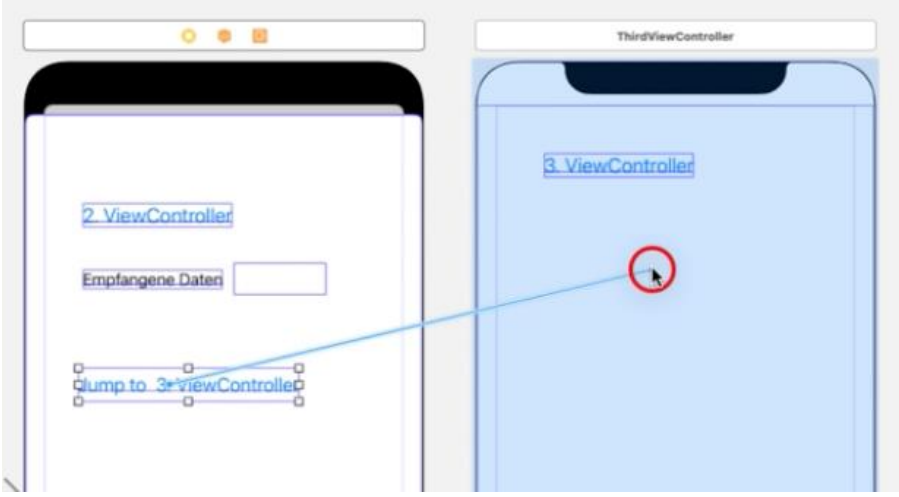


Abbildung 187 Link vom zweiten zum dritten ViewController

Da hier nichts übertragen wird, benötigt man keine prepare-Methode!

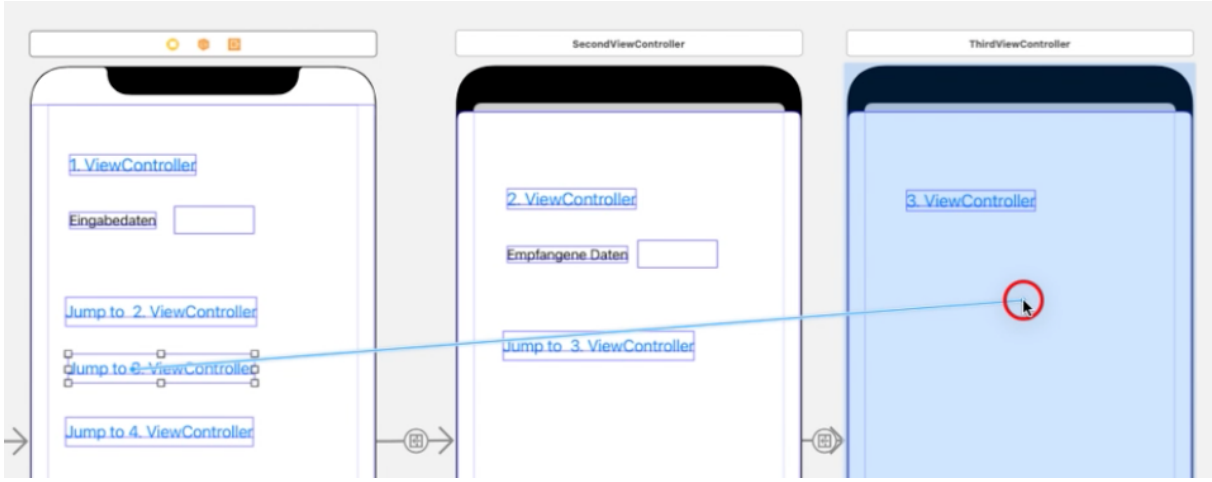


Abbildung 188 Link vom ersten zum dritten ViewController

Abbildung 189Link vom ersten zum vierten ViewController

Bevor der erste Test stattfinden kann, muss noch beim „Link“ zum zweiten ViewController der Inhalt des Eingabefeldes übertragen werden. Dazu gibt es die „prepare“-Methode im **ersten** ViewController.

```
override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
   let input = tinputdata.text
    // Check, correct Controller
   if let dest = segue.destination as? SecondViewController {
      dest.input = input!
   }
}
```

Nun erfolgt die Anzeige der Aufruf-Struktur:

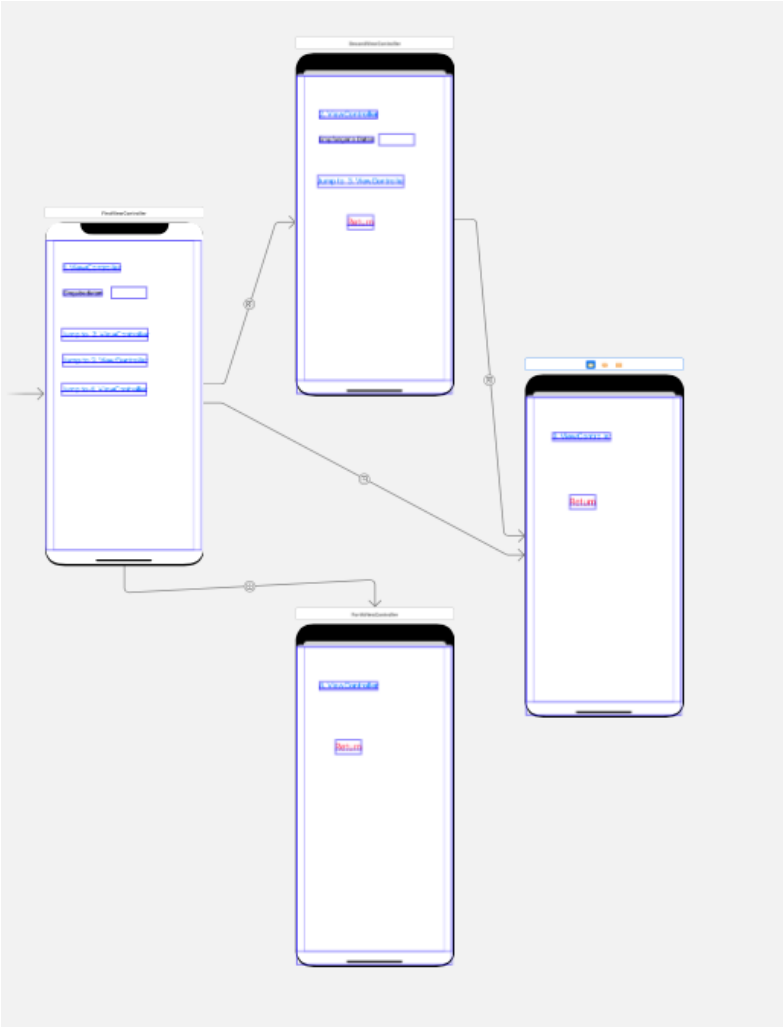
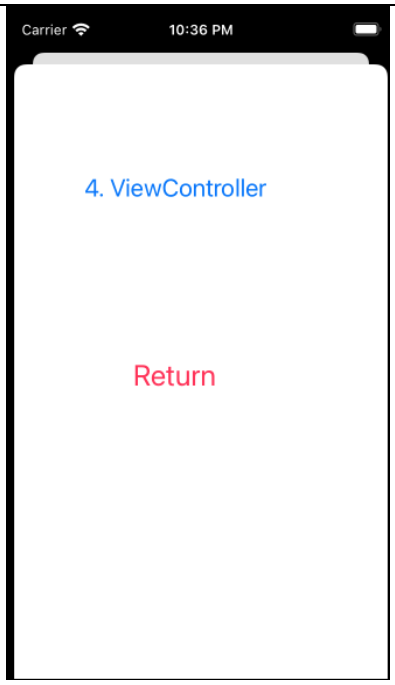


Abbildung 190 Aufruf-Struktur der App Segue

10.7 Erster Testlauf



10.8 Rücksprung organisieren

Der Rücksprung darf nicht wieder über den gleichen Mechanismus erfolgen. Dann würden die einzelnen ViewController weiterhin verbleiben. Deshalb muss man den Rücksprung „ordentlich“ absolvieren. Dazu gibt es den dritten Schalter von links, der für den Rücksprungverantwortlich ist.

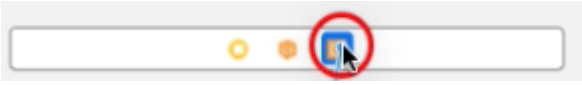


Abbildung 191 Rücksprung-Schalter

Um den Return korrekt abzuarbeiten benötigen die drei aufzurufenden ViewController noch folgende Methode:

```
@IBAction override func unwind(for unwindSegue: UIStoryboardSegue,
                               towards subsequentVC: UIViewController) {
     //  hier fehlt kein Quellcode
}
```

Quellcode für den zweiten, dritten und vierten ViewController

Nun können die Rücksprünge eingebaut werden:

- Anklicken des Schalters „Return“
- Die Taste Ctrl drücken
- Drag&Drop zum oberen dritten Schalter
- Loslassen und die Methode auswählen

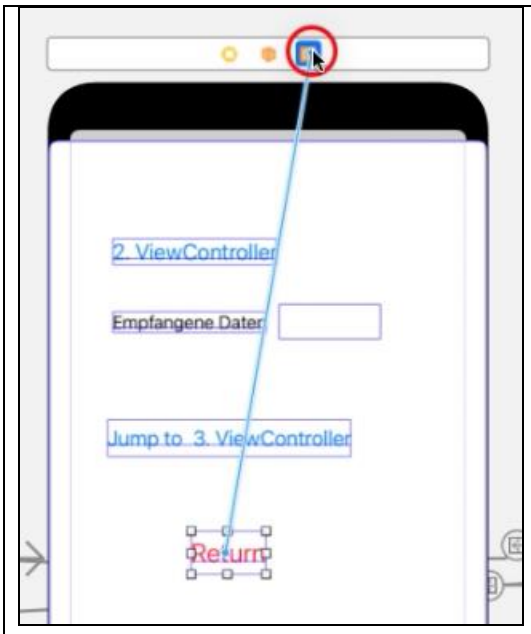


Abbildung 192 Link zum Exit-Schalter

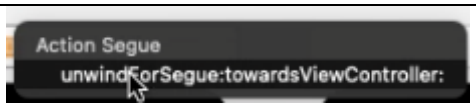
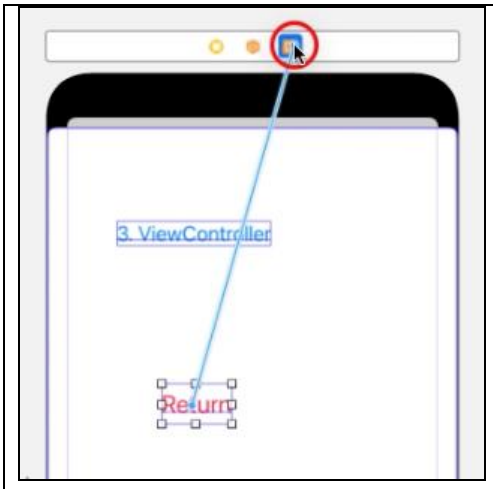
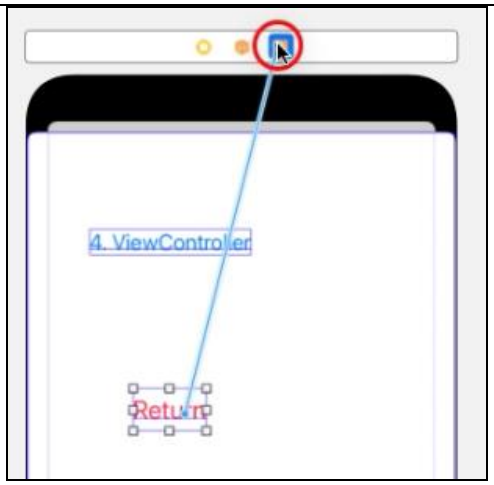


Abbildung 193 Auswahl der Methode

Nun die anderen beiden Exits:





10.8.1 Eintragen einer Delegate-Funktion

1. Eintragen einer Delegate-Funktion im neuen Quellcode.

```
@IBAction override func unwindForSegue(
  unwindSegue: UIStoryboardSegue,
  towardsViewController subsequentVC: UIViewController) {
    //  hier fehlt kein Quellcode
}
```

10.8.2 Verknüpfen des Schalters mit dem Exit-Button

Nachdem im Quellcode die Methode „unwindForSegue“ eingetragen wurde, funktioniert auch die Verknüpfung mit dem „Exit“-Schalter.

- Bei gedrückter Ctrl-Taste zieht man die linke Maus auf den Exit-Schalter im View des Projekts.
- Nun erscheint ein kleiner Dialog, in dem man die oben eingefügt Methode auswählt.

10.8.3 Quellcode des ersten ViewController.swift

```
import UIKit

class FistViewController: UIViewController {
    // Referenzen
    @IBOutlet var tinputdata: UITextField!

    override func viewDidLoad() {
        super.viewDidLoad()
    }
```

```
    // In a storyboard-based application, you will often want to do a
little preparation before navigation
    override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
        // Get the new view controller using segue.destination.
        // Pass the selected object to the new view controller.
        let input = tinputdata.text

        if let dest = segue.destination as? SecondViewController {
            dest.input = input!
        }
    }

}
```

10.8.4 Quellcode des zweiten ViewController.swift

```
import UIKit

class SecondViewController: UIViewController {

    @IBOutlet var tparam: UITextField!
    var input:String="-"

    override func viewDidLoad() {
        super.viewDidLoad()
       tparam.text = input
    }

    @IBAction override func unwind(for unwindSegue: UIStoryboardSegue,
towards subsequentVC: UIViewController) {
             //  hier fehlt kein Quellcode
     }
}
```

10.8.5 Quellcode des dritten ViewController.swift

```
import UIKit

class ThirdViewController: UIViewController {

    override func viewDidLoad() {
        super.viewDidLoad()

        // Do any additional setup after loading the view.
    }




    @IBAction override func unwind(for unwindSegue: UIStoryboardSegue,
towards subsequentVC: UIViewController) {
```

```
            //  hier fehlt kein Quellcode
     }


}
```

10.8.6 Quellcode des vierten ViewController.swift

//
// ForthViewController.swift
// Segue1
//
// Created by User1 on 13.09.21.
//

```
import UIKit

class ForthViewController: UIViewController {

    override func viewDidLoad() {
        super.viewDidLoad()
    }



    @IBAction override func unwind(for unwindSegue: UIStoryboardSegue,
towards subsequentVC: UIViewController) {
            //  hier fehlt kein Quellcode
     }
}
```

11 Navigations-Controller

Der Navigations-Controller fungiert im Prinzip als Hauptmenü. Mit seiner Hilfe kann man weitere Views, normale View, TabbedController, PageBasedController oder TableView aufrufen. **Dabei darf keine View zweimal verlinkt werden!**

Hinweis:

- Der Rücksprung erfolgt immer über einen Back-Schalter in der ButtonBar.
- Eine TableView mit Edit-, Delete oder New-Controllern würde damit nicht funktionieren, da man keinen Rückgabecode erhält.
- Jede Seite ist also selbstständig.

Ablauf:

1. Erstellen eines Projektes „Single View Application“
2. Einfügen eines Navigations-Controllers.
3. Der Navigations-Controller ist immer mit einer TableView verbunden. Diese wird für dieses Beispiel gelöscht und durch einen einfachen ersten ViewController ersetzt.
4. Die Verbindung ist dann eine „**root view controller**“.
5. Danach kann man beliebige ViewController einbauen und mit Schaltern oder mit Code Behind verknüpfen. Die Verbindung ist dann immer die **Show**-Variante

11.1 Projekt erstellen

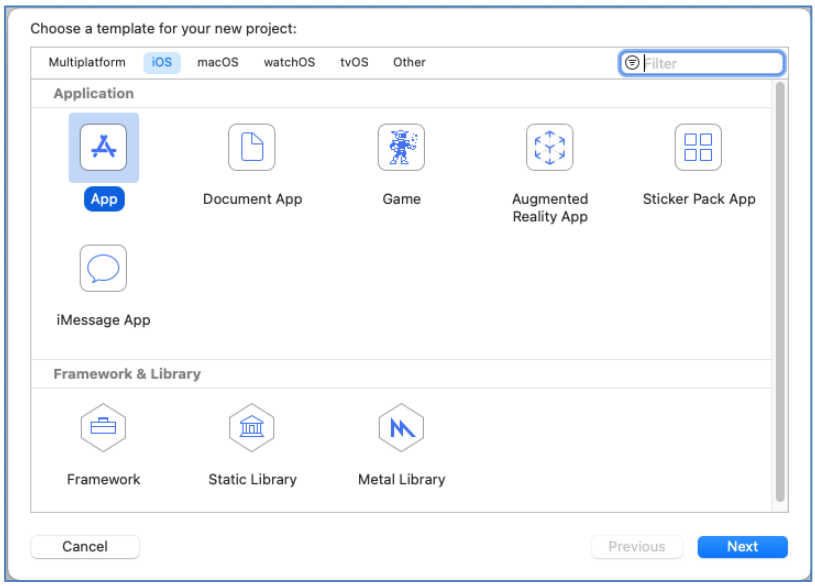


Abbildung 194 Dialog zum Erstellen eines Projektes

Nun wird der Name eingegeben: NaviController07

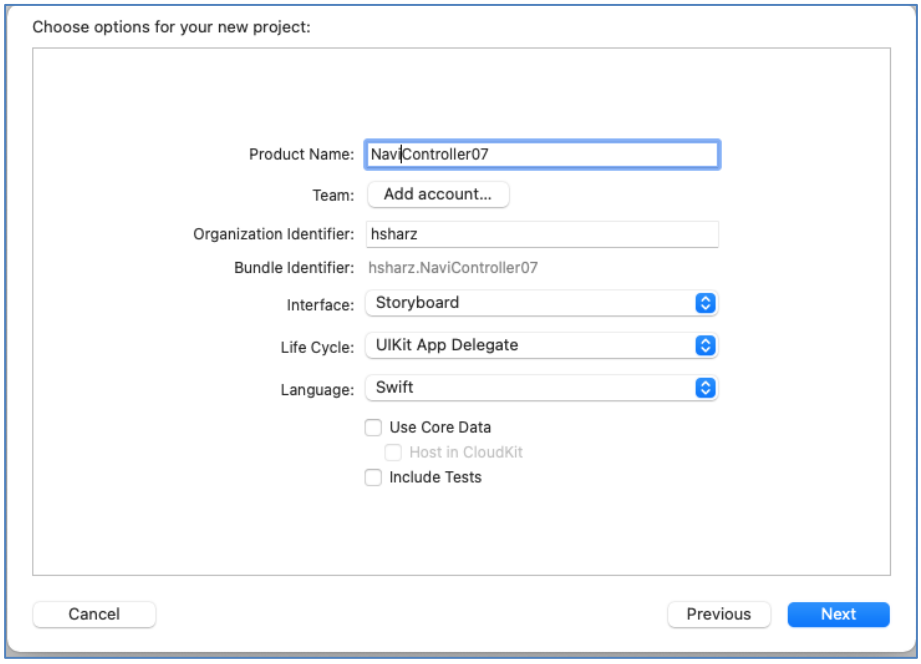


Abbildung 195 Eintragen des Namens

Danach wird das Projekt gespeichert:

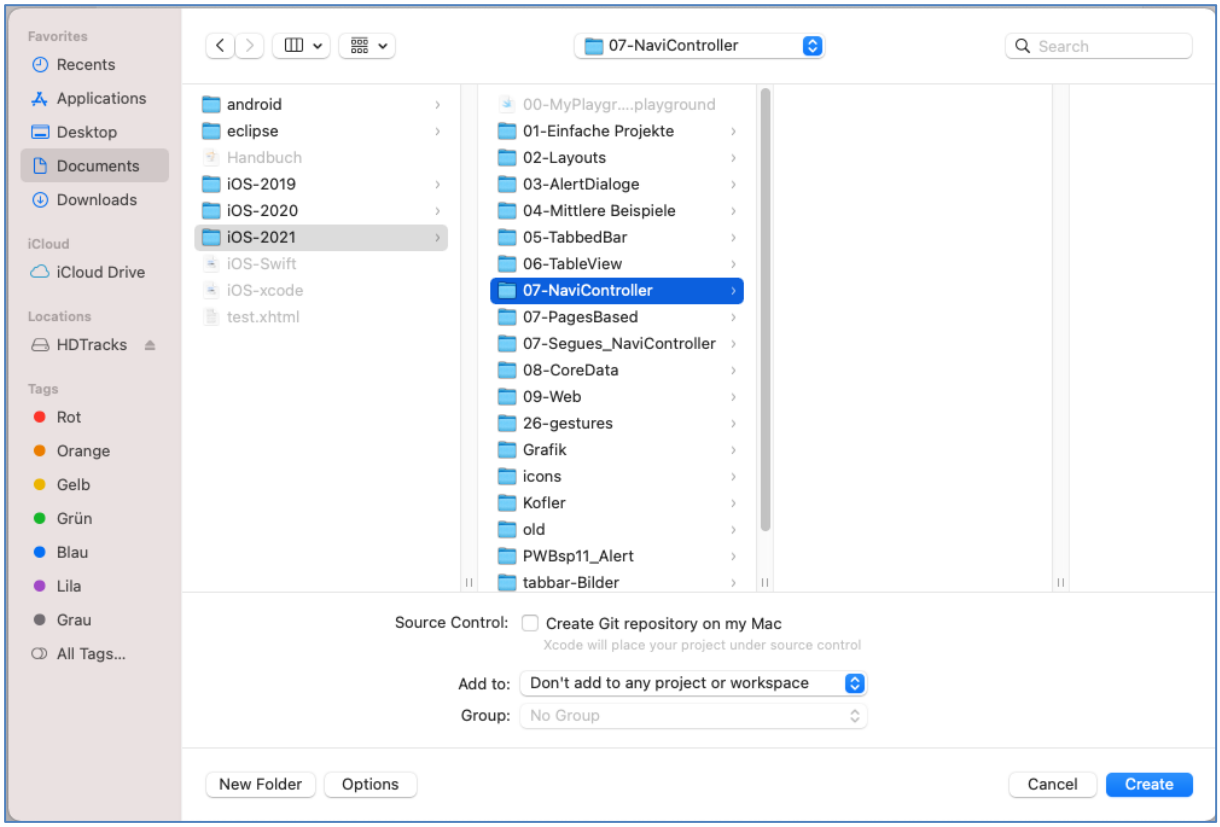


Abbildung 196 Projekt im Ordner speichern

Nun hat man kann bekannte Anfangsszenario.

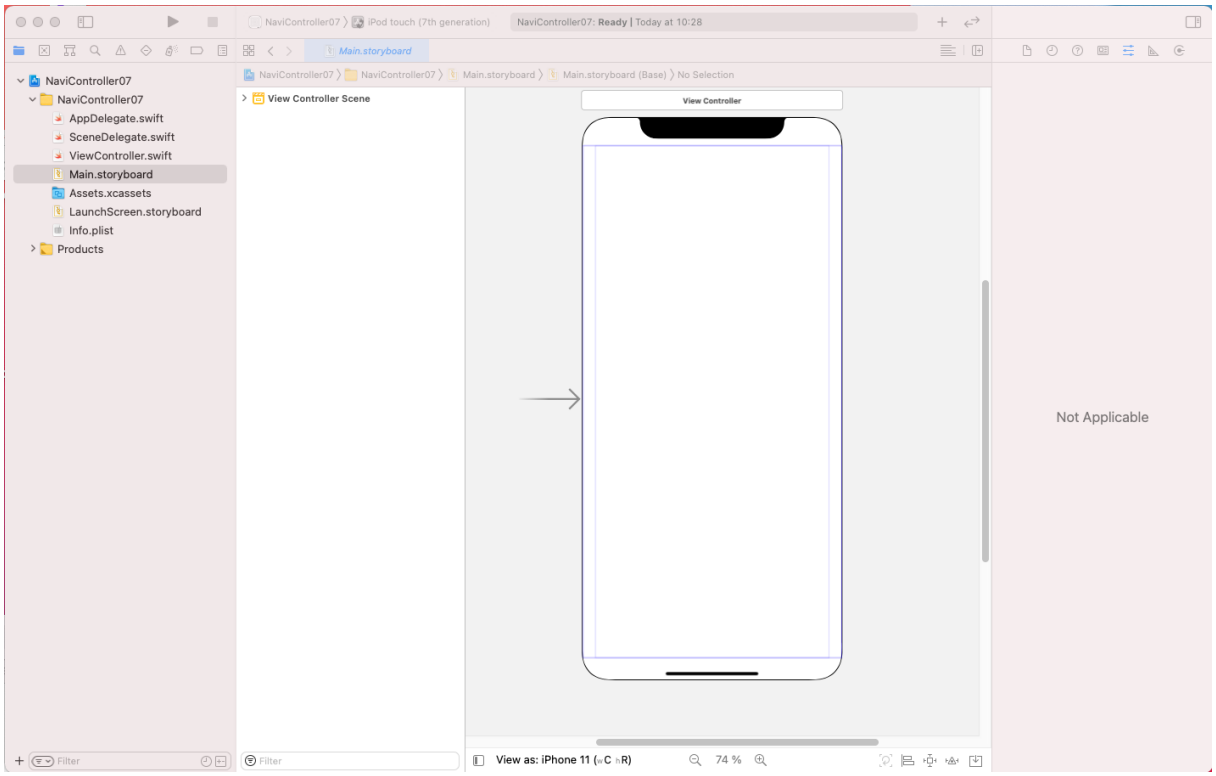


Abbildung 197 Anfangsszenario

Da ein Navigationscontroller benötigt wird muss man ihn über den UI-LibDialog, Shift+Cmd+L einfügen:

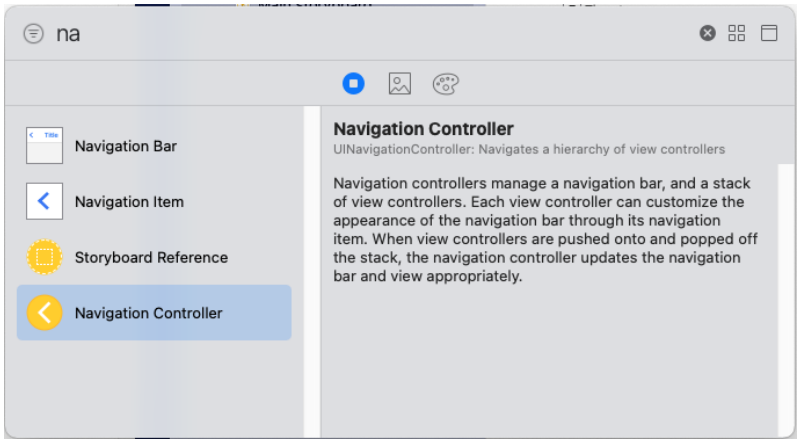


Abbildung 198 Aufruf des UI-LibDialog, Filter navi

Nun schiebt man den Navigationscontroller neben dem vorhandenen ViewCVontroller.
Das Projekt sollte jetzt so aussehen:

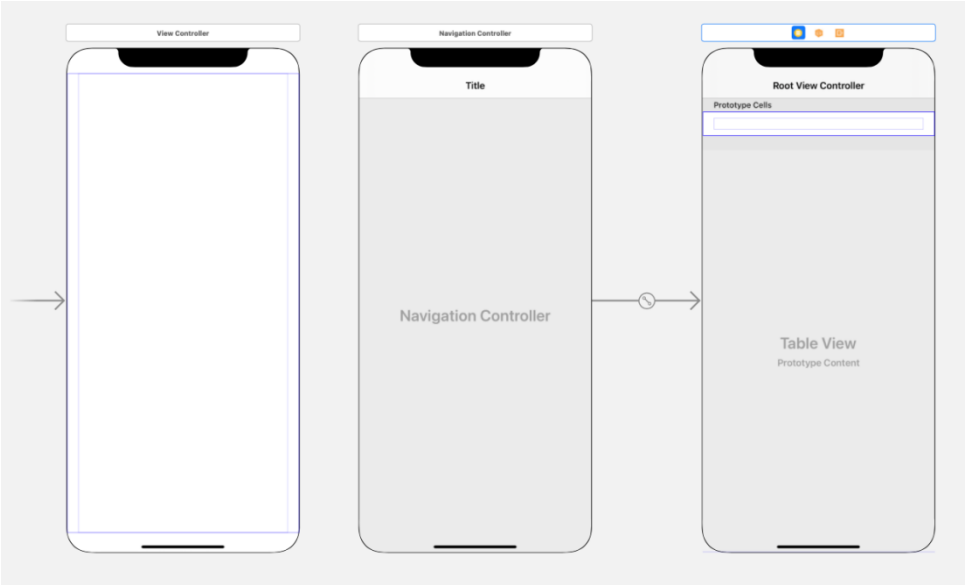


Abbildung 199 NavigationController mit zwei ViewControllern

11.2 Weitere ViewController einbauen

Im nächsten Schritt wird der Initialcontroller umgesetzt. Dazu klickt man auf den Pfeil, er müsste blau angezeigt werden. Dann verschiebt man ihn zum Naviagtionscontroller.

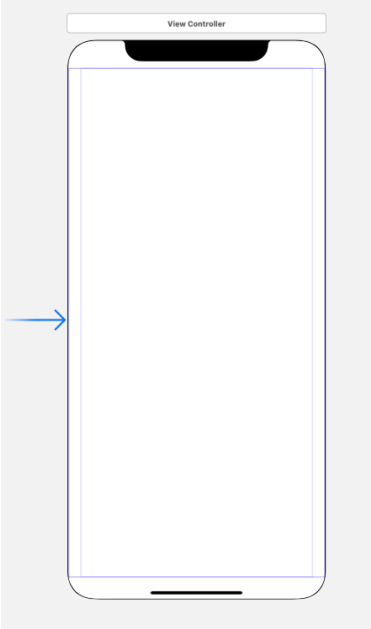


Abbildung 200 Initialcontroller vorher

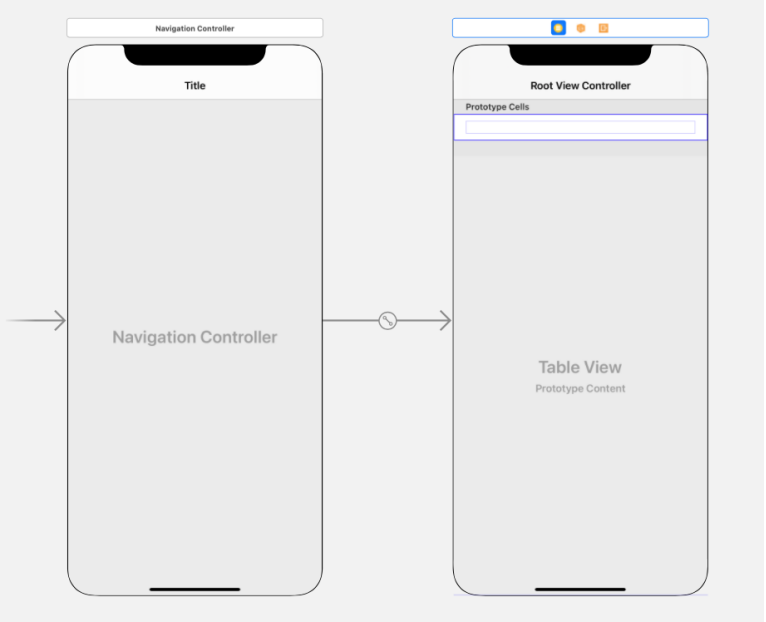


Abbildung 201 Initialcontroller später

Beim Starten des Projektes sollte eine leere Liste erscheinen:

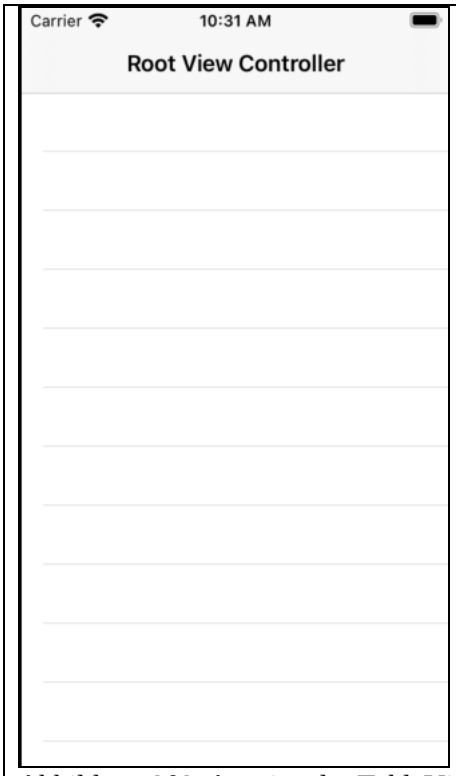


Abbildung 202 Anzeige der TableView (Test)

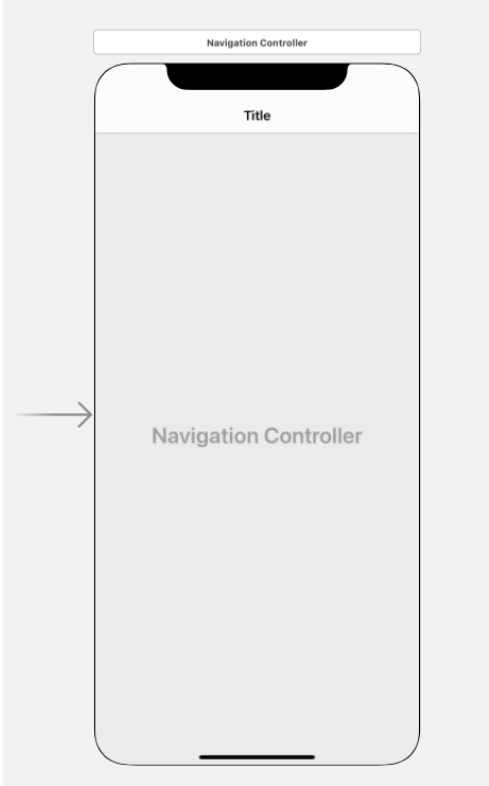


Abbildung 203 Nach dem Löschen der TableView

Um mit Schaltern für dieses Beispiel, weiterzuarbeiten, muss man **beide** ViewController löschen.

Nun werden **vier** normale ViewController in das Projekt eingefügt:

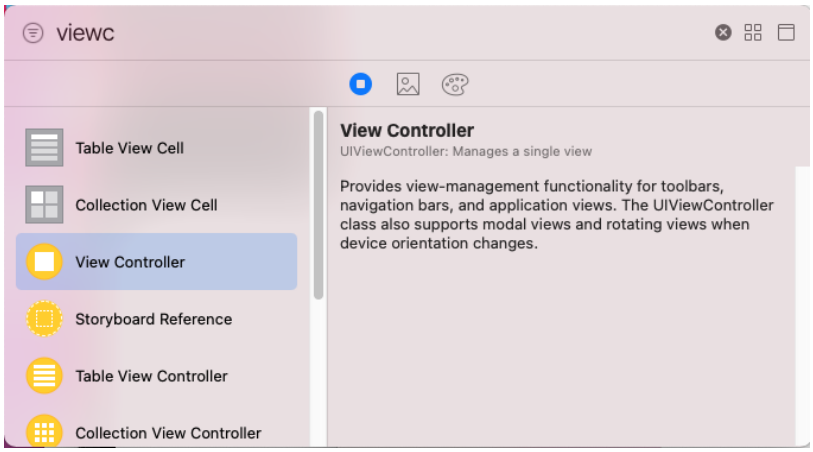


Abbildung 204 ViewController im UI-Lib-Dialog

Alle Controller werden in Reihe dargestellt:

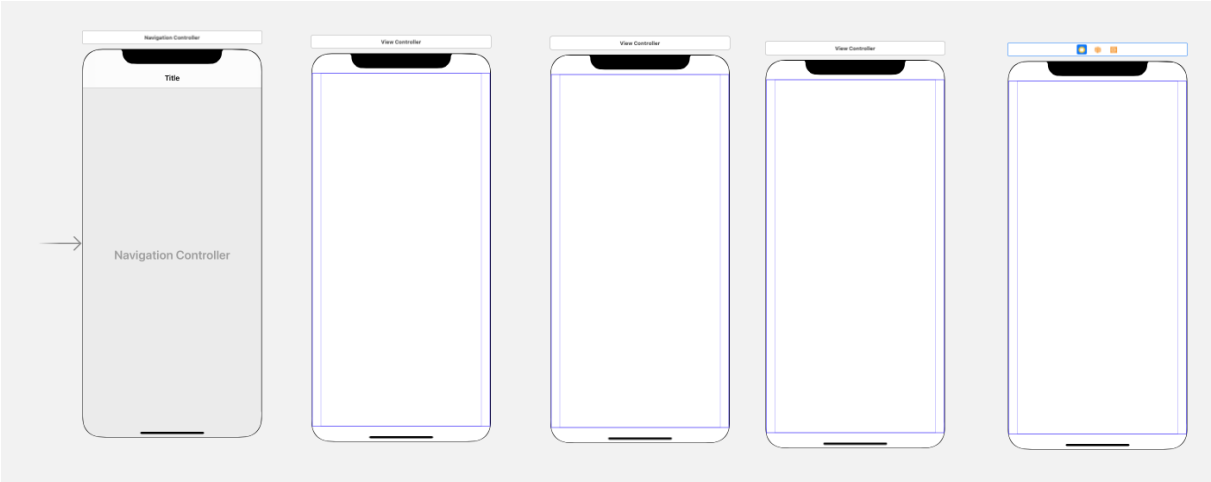


Abbildung 205 Vier ViewController

Umbenennen der ViewController (Title):

- FirstViewController
- SecondViewController
- ThirdViewController
- ForthViewController

11.3 Setzen der Links

Erstellen der Links:

- Anklicken des Navigationscontroller
- Ctrl-Taste drücken
- Zum ersten ViewController ziehen und loslassen
- Auswahl „root-view-Controller“

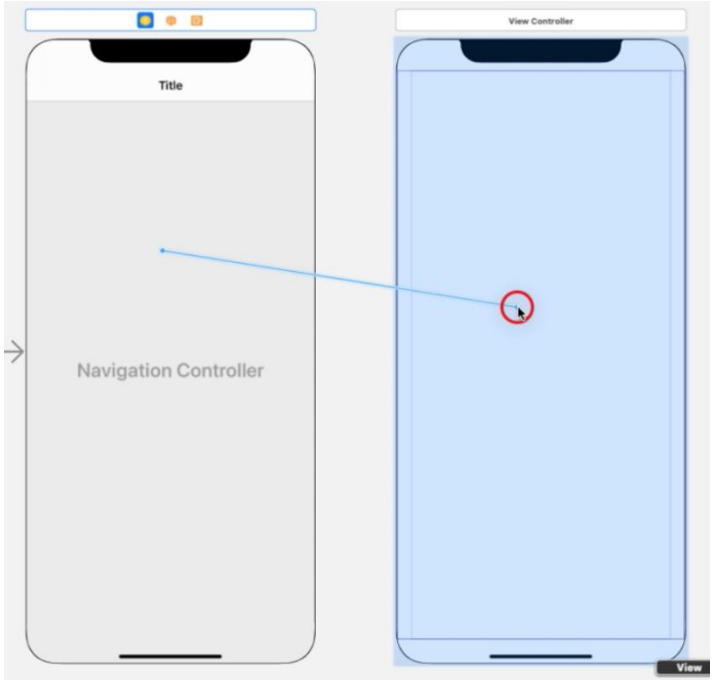


Abbildung 206 Den Navigationscontroller mit dem ersten View verbinden

Hier ist es wichtig den Eintrag „root view controller“ auszuwählen!

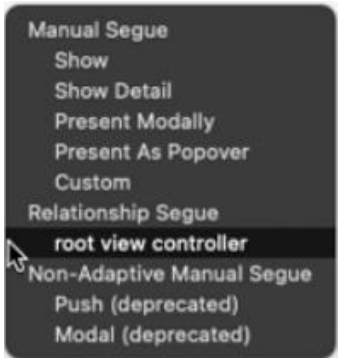


Abbildung 207 Navigationscontroller mit einer „root view controller“ Verbindung

Nun werden die vier Controller mit UI-Elementen bestückt:

Navigation:

- Von Eins nach Zwei, jeweils „show“ auswählen
- Von Zwei nach Drei, jeweils „show“ auswählen
- Von Eins nach Vier, jeweils „show“ auswählen

Link setzen:

Abbildung 208 Link vom ersten zum zweiten View setzen

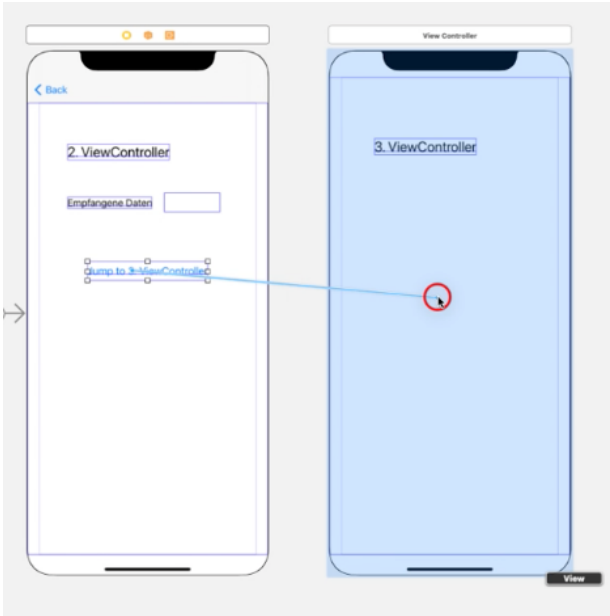


Abbildung 209 Link vom zweiten zum dritten View setzen

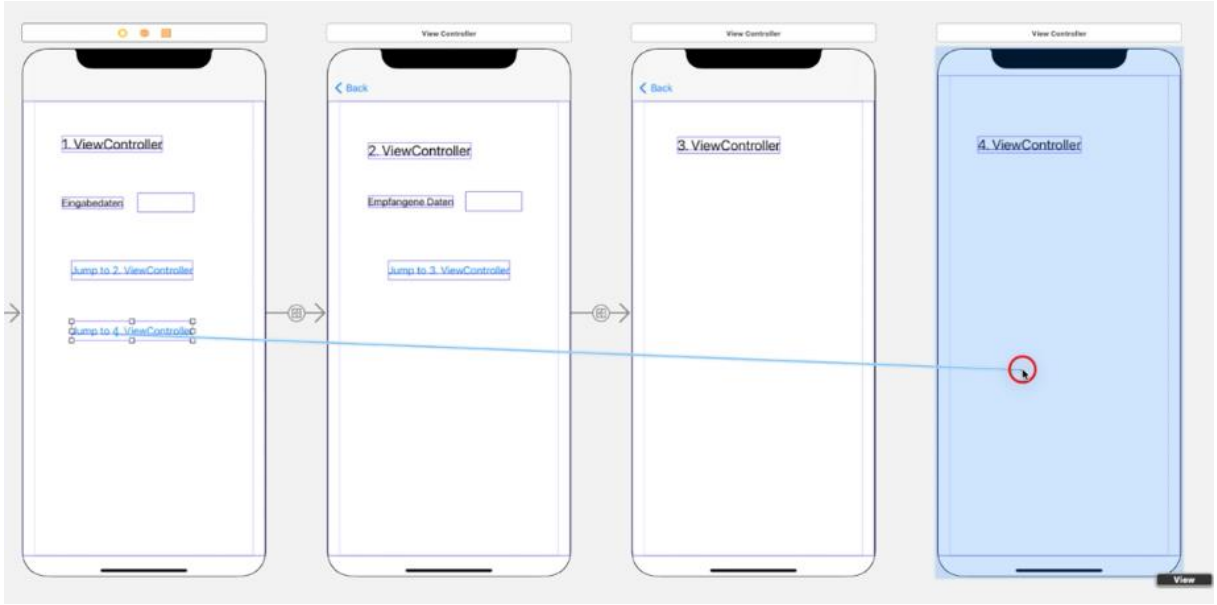


Abbildung 210 Link vom ersten zum vierten View setzen

Die untere Abbildung zeigt die komplette Struktur mit den UI-Elementen:

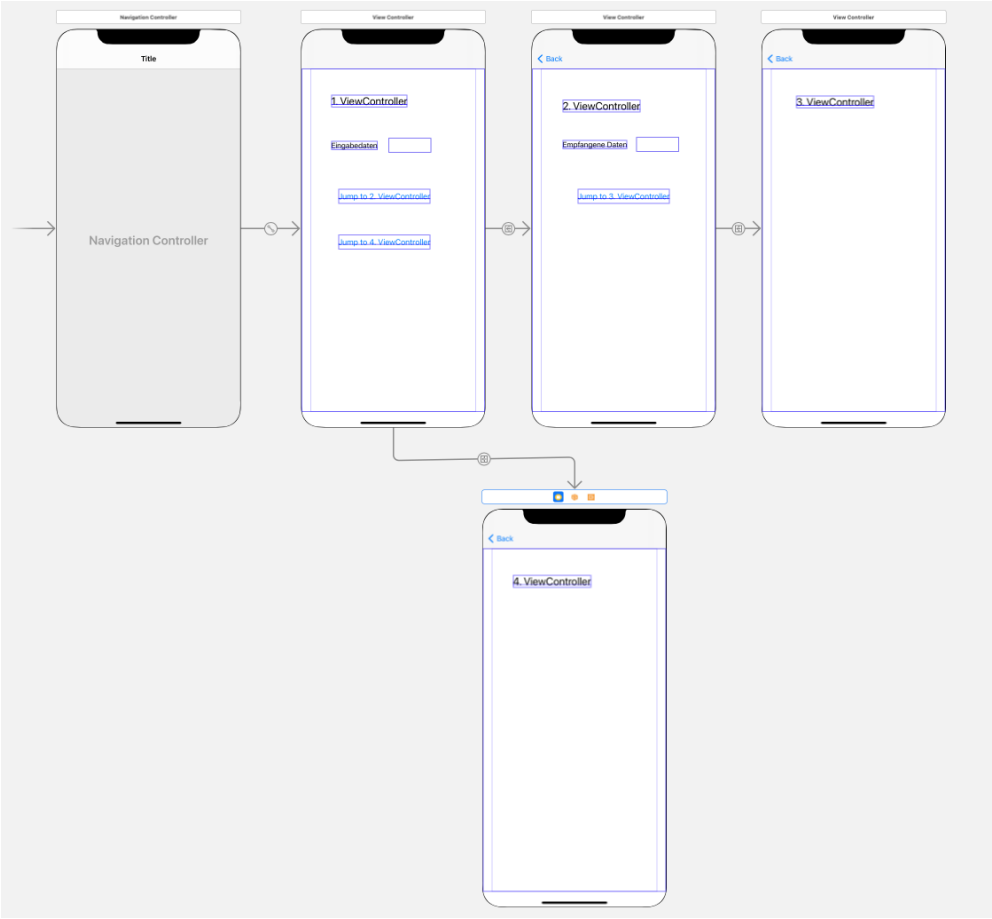
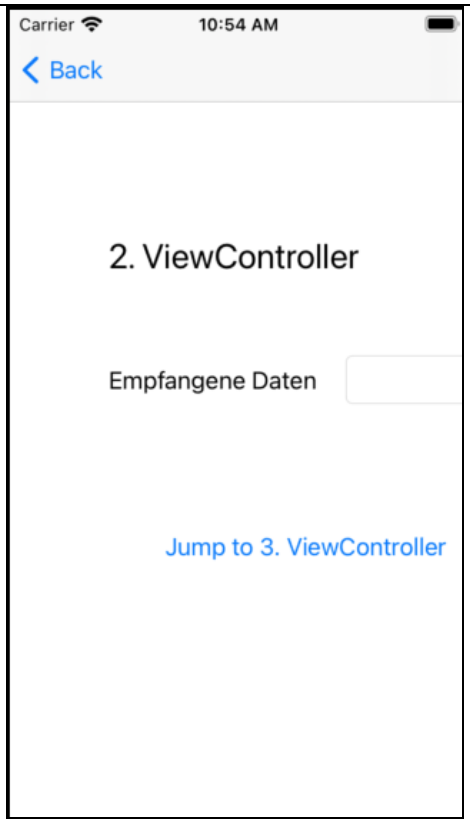
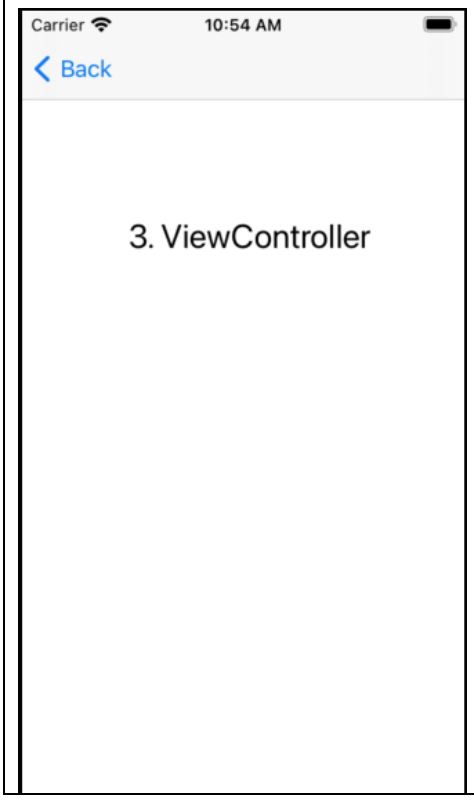
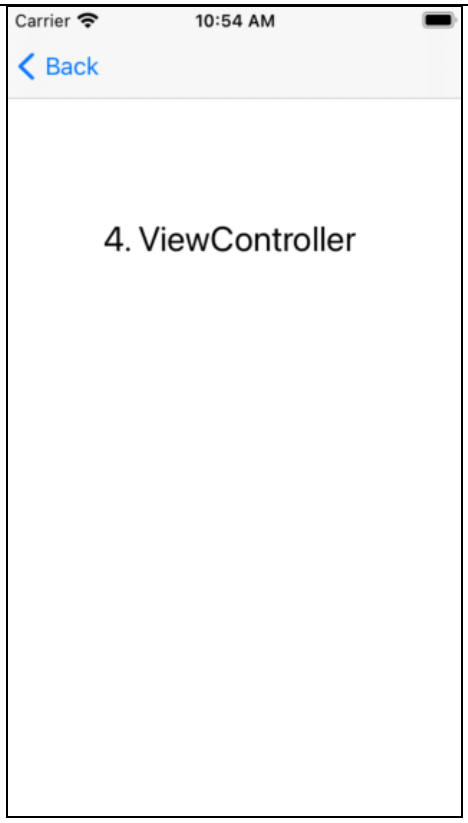


Abbildung 211 Komplette Struktur des Projektes

11.4 Anzeige eines Tests







11.5 Swift-Dateien für die ViewController erstellen

Nun müssen noch die Swift-Dateien erzeugt werden, die Klassen den ViewController zuordnet werden, die Referenzen erstellt werden und der Datentransfer vom ersten zum zweiten View implementiert werden.

Swift-Dateien:

Mit Cmd+N erscheint der „new file“-Dialog

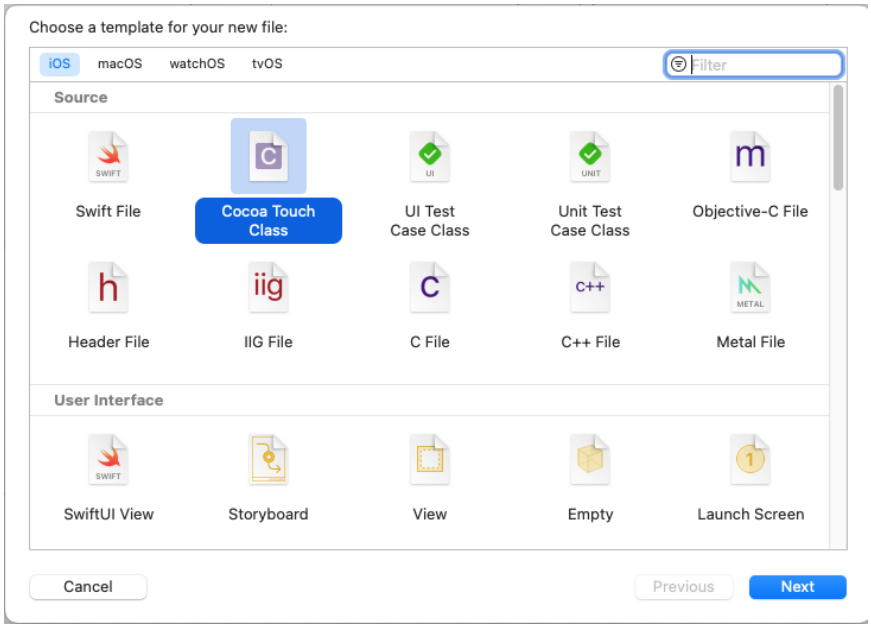


Abbildung 212 Der new file“-Dialog

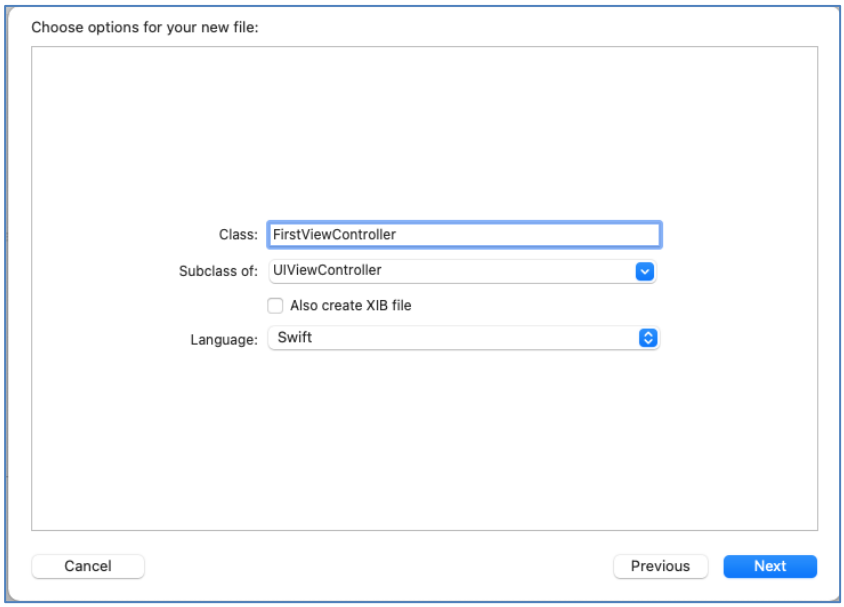


Abbildung 213 Eintragn des Namens "FirstViewController"

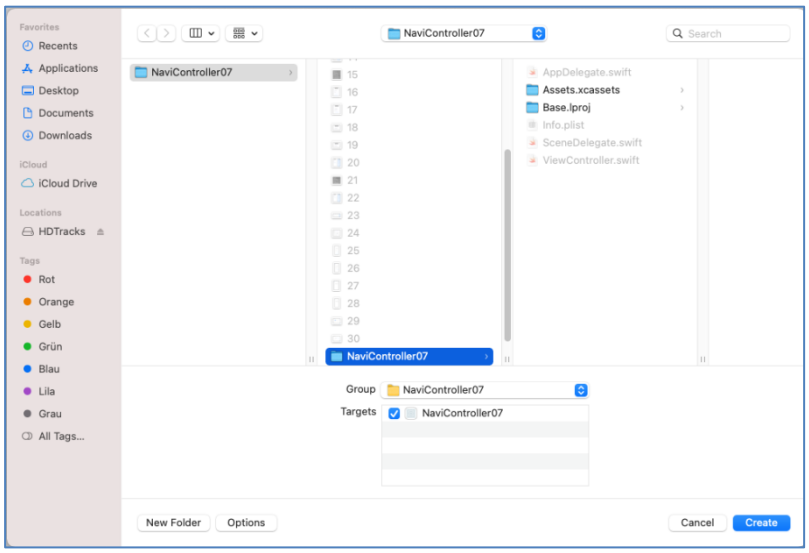


Abbildung 214 Speichern der neuen Datei

Mit Cmd+N erscheint der „new file“-Dialog

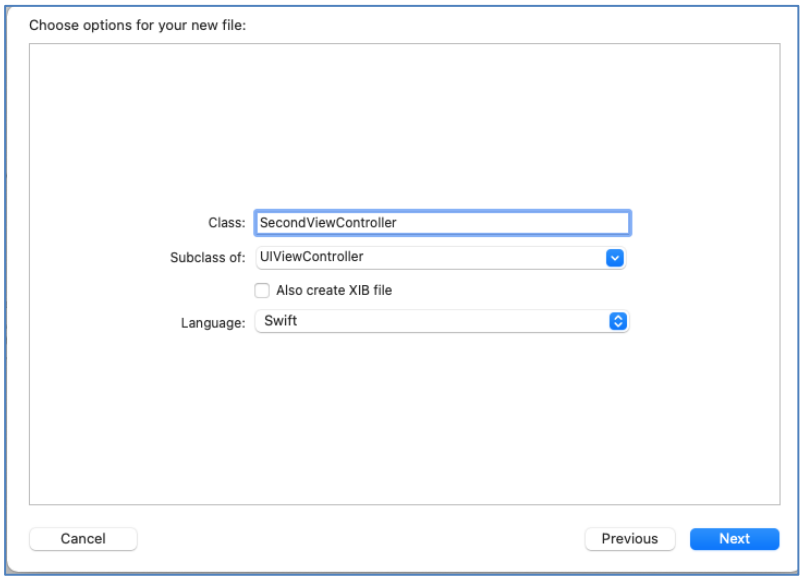


Abbildung 215 Eintragn des Namens "SecondViewController"

Speichern der Datei SecondViewController.swift im Ordner

Mit Cmd+N erscheint der „new file“-Dialog

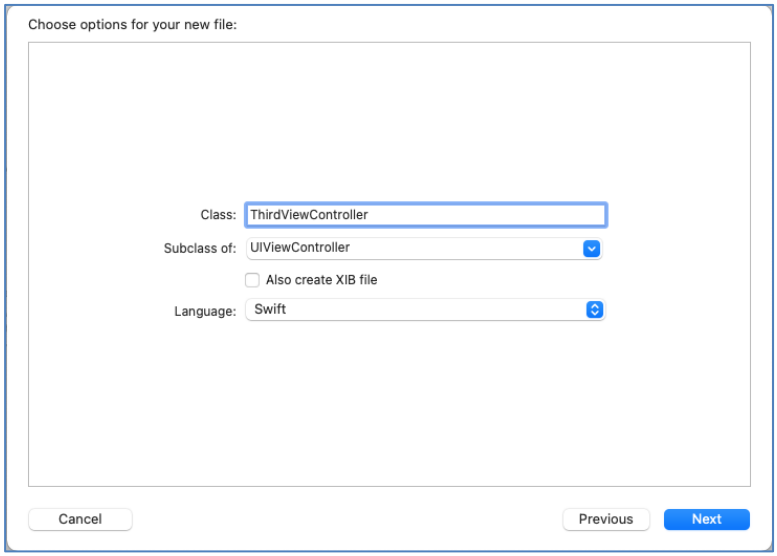


Abbildung 216 Eintragn des Namens "ThirdViewController"

Speichern der Datei ThirdViewController.swift im Ordner

Mit Cmd+N erscheint der „new file“-Dialog

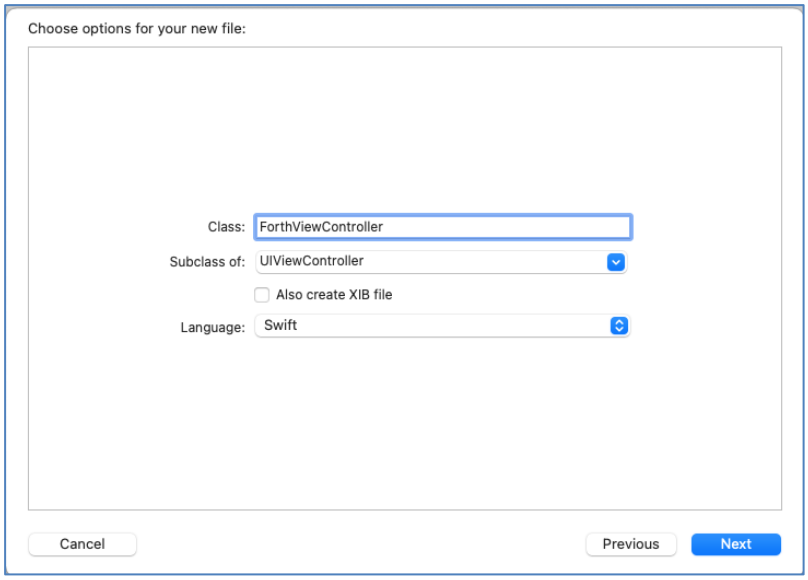


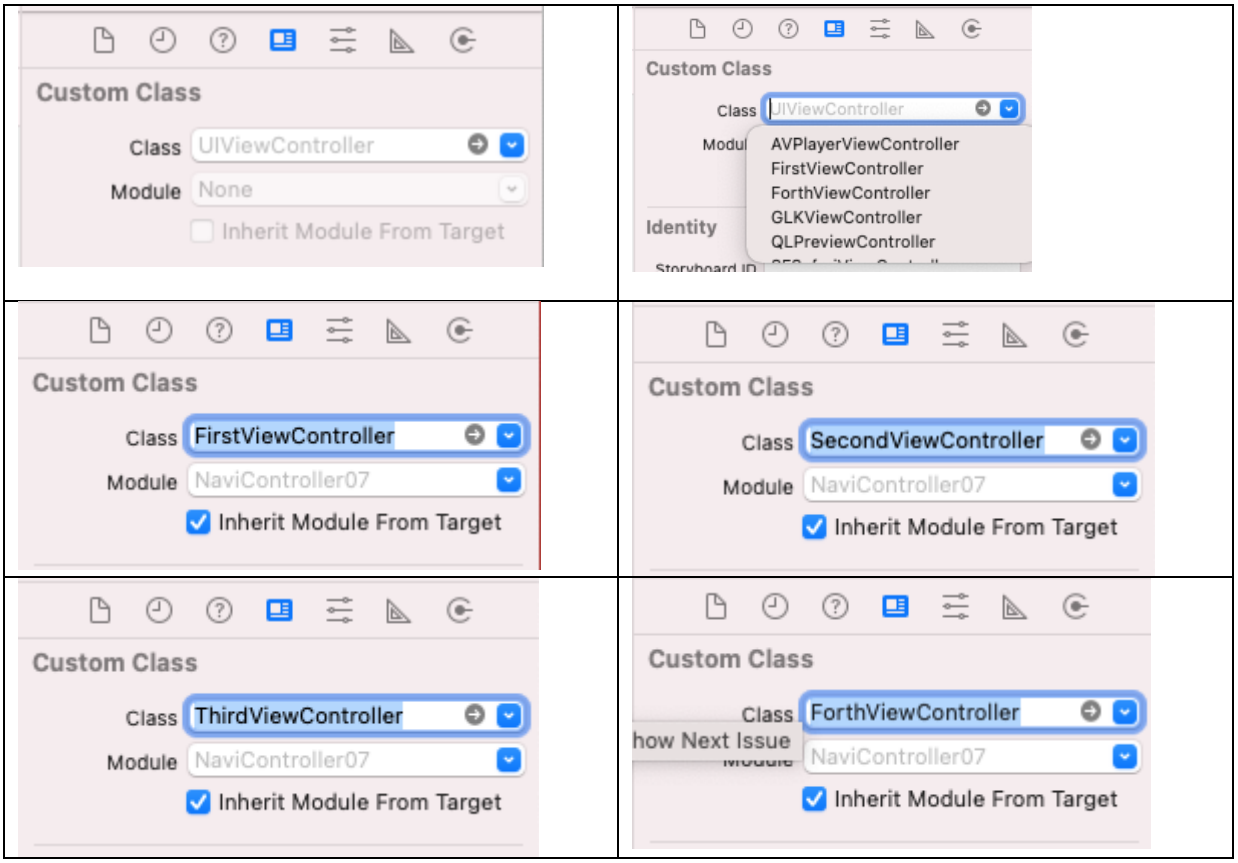
Abbildung 217 Eintragn des Namens "ForthViewController"

Speichern der Datei ForthViewController.swift im Ordner

Die Klassen den ViewController zuordnen:

- Man aktiviert den ersten ViewController:

- Rechts oben om Property-Feneter ist die Klasse des ViewController eingetragen.
- Jetzt noch leer.
- Man wählt die korrekte Klasse.aus



11.6 Referenzen

Für den ersten ViewController wird eine Referenz der Eingabezeile erzeugt:

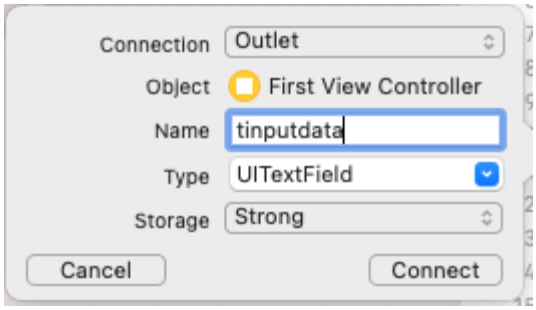


Abbildung 218 InputData für den 1. ViewController

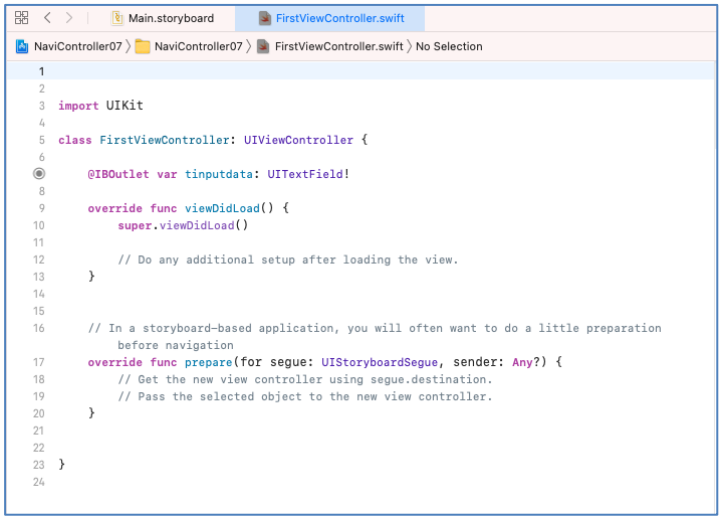


Abbildung 219 Erster ViewController (swift-Datei)

Für den zweiten ViewController wird auch eine Referenz der Textzeile eingefügt. Des Weiteren wird eine Variable, input, als Parameter erstellt.

```
@IBOutlet var tparam: UITextField!
var input:String="-"
```

11.7 Datentransfer

Erster ViewController:

```
override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
    let input = tinputdata.text
     // correct ViewController ?
    if let dest = segue.destination as? SecondViewController {
       dest.input = input!
    }
}
```

Zweiter ViewController:

```
override func viewDidLoad() {
    super.viewDidLoad()
    tparam.text = input
}
```

Sinnvoller wäre es natürlich eine Klasse oder eine Liste zu übergeben!

11.7.1 Anzeige des ersten ViewController.swift

```
import UIKit

class FirstViewController: UIViewController {

    @IBOutlet var tinputdata: UITextField!

    override func viewDidLoad() {
        super.viewDidLoad()
    }


    // In a storyboard-based application,
    // you will often want to do a little preparation before navigation
    override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
        // Get the new view controller using segue.destination.
        // Pass the selected object to the new view controller.
        let input = tinputdata.text

        if let dest = segue.destination as? SecondViewController {
            dest.input = input!
        }
    }


}
```

11.7.2 Anzeige des zweiten ViewController.swift

```
import UIKit

class SecondViewController: UIViewController {

    @IBOutlet var tparam: UITextField!
    var input:String="-"

    override func viewDidLoad() {
        super.viewDidLoad()
        tparam.text = input
    }


}
```

11.7.3 Anzeige des dritten ViewController.swift

```
//
//  ThirdViewController.swift
//  NaviController07
//
//  Created by User1 on 12.09.21.
//
```

```
import UIKit
```

```
class ThirdViewController: UIViewController {

    override func viewDidLoad() {
        super.viewDidLoad()
    }

}
```

11.7.4 Anzeige des vierten ViewController.swift

```
import UIKit

class ForthViewController: UIViewController {

    override func viewDidLoad() {
        super.viewDidLoad()
    }


}
```

12 TableView

12.1 Einfache TableView

Dieses Kapitel beschreibt die Vorgehensweise, um in einer TableView Texte in der „Zelle“ anzuzeigen.

12.1.1 Anlegen eines Projektes

Legen Sie ein normales Projekt „App“ an und speichern dieses.

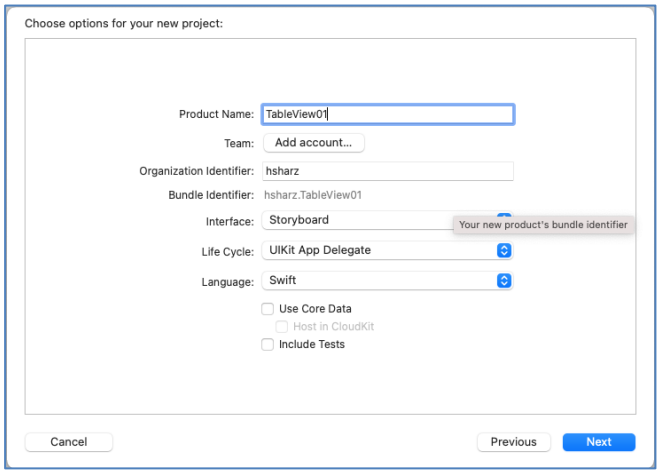


Abbildung 220 Anlegen einer App für eine TableView

Nun wird die App gespeichert:

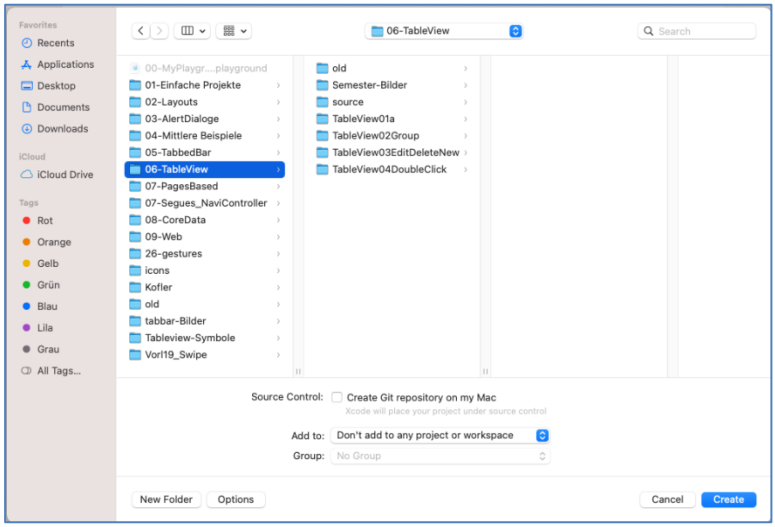


Abbildung 221 Speichern der App im Ordner TableView

Anzeige der App in XCode

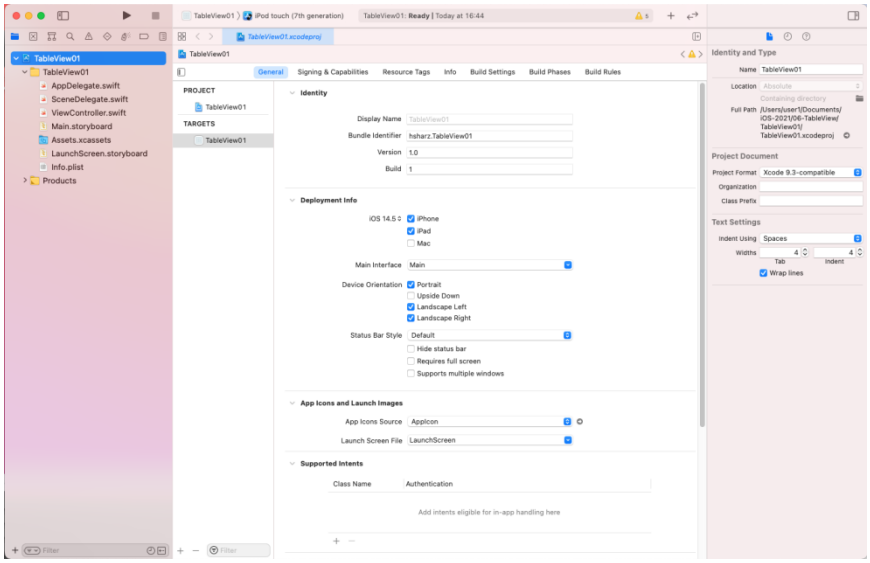


Abbildung 222 TableView01 in XCode

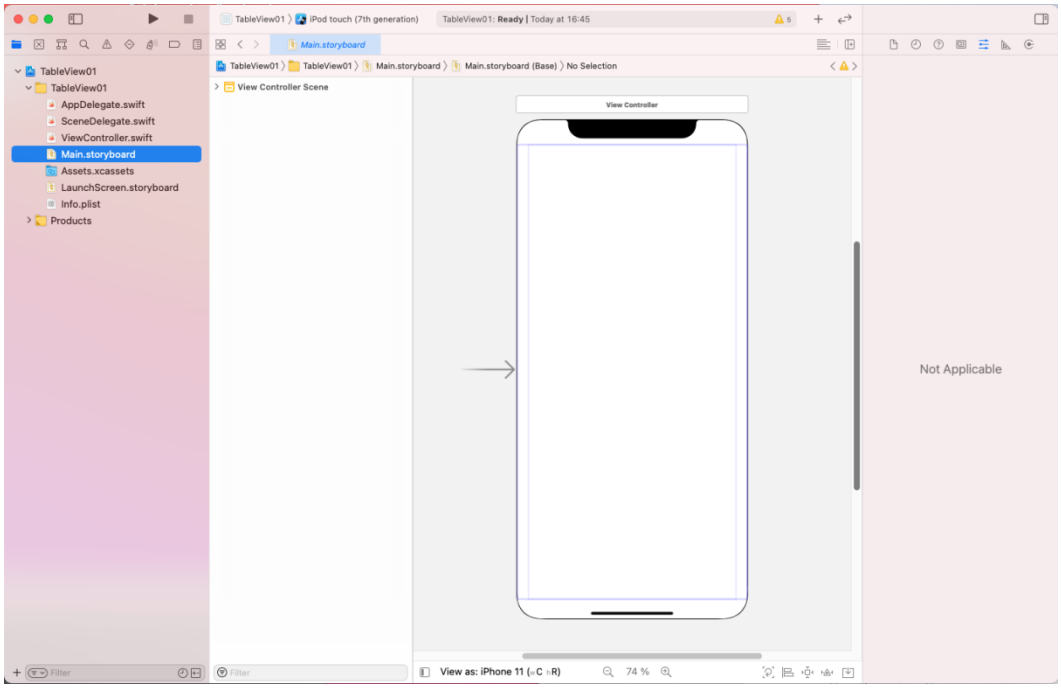


Abbildung 223 Anzeige des Main.StoryBoard

12.1.2 TableView einfügen

Fügen Sie nun aus dem Lib-Dialog einen TableView in den ViewController.

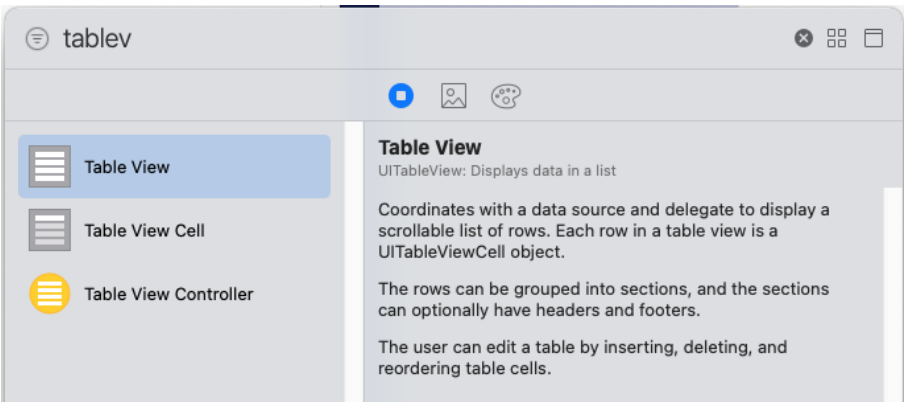


Abbildung 224 TableView Controller einfügen

Das Projekt müsste dann so aussehen:

Abbildung 225 Aktueller Stand in XCode

Abbildung 226 Test mit dem aktuellen Stand

Nun wird man eine Referenz der UITableView in der Swift-Klasse erstellt:

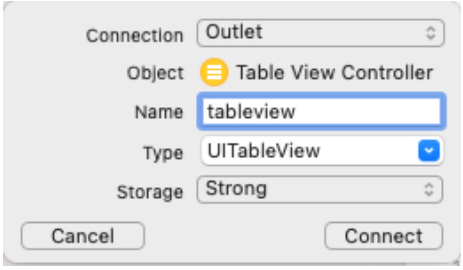


Abbildung 227 Referenz-Dialog für die TableView

12.1.3 Hilfsklassen für die TableView

In der Tabelle sollen Städte angezeigt werden. Dazu brauchen wir eine Klasse „City“ und eine Verwaltungsklasse „CitiesDB“.

Anlegen einer neuen Klasse „city.swift“

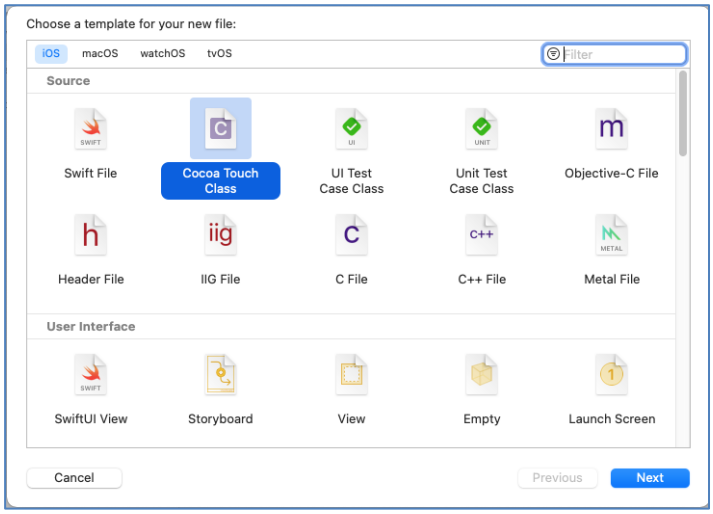


Abbildung 228 Einfüge-Dialog für die Klasse City.swift (Cmd+N)

Nun den Namen und den Typ eintragen:

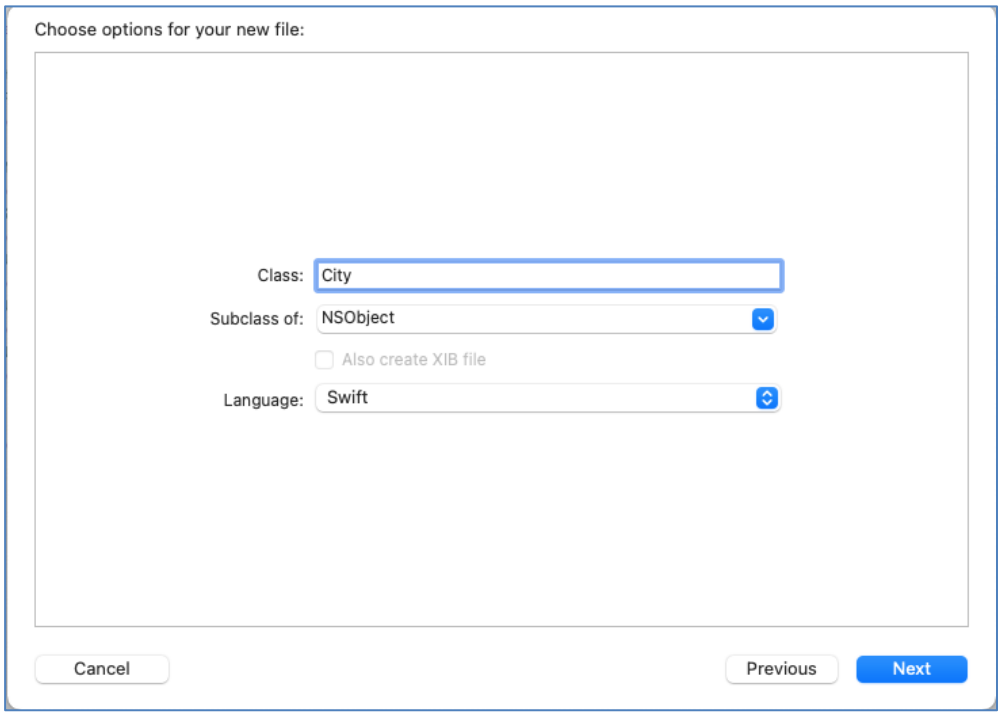


Abbildung 229 Name und Typ der Klasse City

Nun noch eine zweite Klasse „Enums.swift“ und eine dritte Klasse „CitiesDB.swift“ einfügen.
In der Klasse „Enums.swift“ werden die Aufzählungen, Enumeration eingetragen:

```
import UIKit


enum Continent:Int {        // Deklaration
  case  AFRICA=1, AMERICA, ASIA, AUSTRALIA, EUROPE, ANTARTKIS
} }
```

Quellcode für Enums.swift

Eine Enumeration ist im Quellcode wesentlich sicherer als ein int-Wert mit Konstanten. Das wäre ein sehr schlechter Stil.

```
import UIKit


class City: NSObject {
      // Attribute
   var continent: Continent=Continent.AFRICA
   var name : String = ""
   var remark : String = ""
   var visited:Visited = Visited.NONE

     // Konstruktor
   init(_ continent: Continent,_ name:String, _ remark:String, _ visited:Visited ){
      self.continent = continent
      self.name = name
      self.remark = remark
      self.visited = visited
   }  // init

}
```

Quellcode für City.swift

```
import UIKit

class CitiesDB: NSObject {

   public static func loadCitiesIntern()  -> Array<City> {
     var cities=[City]()
     cities.append( City( Continent.AMERICA ,  "New York",  "Manhattan", Visited.NONE)  )
     cities.append( City( Continent.EUROPE , "Mailand", "Mailander Scala", Visited.AVERAGE))
     cities.append( City( Continent.AUSTRALIA ,"Central Australien","Ayers Rock, Uluru",
Visited.NONE)  )
     cities.append( City( Continent.AMERICA ,"Peru",  "Machu Picchu",  Visited.UP)  )
     cities.append( City( Continent.EUROPE ,"Wernigerode",  "HS Harz",  Visited.DOWN)  )
     cities.append( City( Continent.AUSTRALIA ,"Queensland",  "The Great Barrier Reef",
Visited.NONE)  )
     cities.append( City( Continent.AMERICA ,"Rio de Janeiro", "Karneval",  Visited.DOWN)  )
     cities.append( City( Continent.EUROPE ,"Brüssel",   "Atomium",  Visited.UP)  )
     cities.append( City( Continent.AUSTRALIA ,"Sidney",  "Opernhaus",  Visited.UP)  )
     cities.append( City( Continent.AFRICA ,"Kairo",  "Pyramiden",  Visited.NONE)  )
     cities.append( City( Continent.AMERICA ,"Chile",  "Atacama Wüste",  Visited.UP)  )
```

```
     cities.append( City( Continent.ASIA ,"Tokio",  "Shinjuku Gyoen National Garden",
Visited.DOWN)  )
     cities.append( City( Continent.EUROPE ,"Kiel",  "Kieler Woche",  Visited.UP)  )
     cities.append( City( Continent.AFRICA ,"Hunsbergen","Apollo 11 Höhle",  Visited.NONE) )
     cities.append( City( Continent.EUROPE ,"Dresden",  "Das grüne Zimmer",  Visited.NONE) )
     cities.append( City( Continent.AFRICA ,"Marrakesch",  "Botanische Garten Jardin
Majorelle",  Visited.DOWN)  )
     cities.append( City( Continent.AMERICA ,"San Fransisco",  "Golden Gate Bridge",
Visited.UP)  )
     cities.append( City( Continent.EUROPE ,"Halberstadt",  "John Cage",  Visited.NONE)  )
     cities.append( City( Continent.ASIA ,"Peking",  "Verbotene Stadt",  Visited.NONE)  )
     cities.append( City( Continent.EUROPE ,"Venezuela",  "Angel Falls",  Visited.UP)  )
     cities.append( City( Continent.AMERICA ,"Bilbao",  "Guggenheim-Museum",  Visited.UP) )
     cities.append( City( Continent.ASIA ,"Chiang Mai",  "Der weiße Tempel",  Visited.UP) )
     cities.append( City( Continent.AMERICA ,"Rio de Janeiro",  "Zuckerhut",  Visited.NONE))
     return cities
}
```

Quellcode für CitiesDB.swift

12.1.4 TableView mit einer Liste füllen

Durch die Delegates des TableViewControllers sind alle wichtigen Methoden schon in der Datei „ViewController.swift“ eingetragen. Man muss also nur die Liste der Städte holen, die UITableView mit den Methoden verknüpfen (Interface) und die Anzahl der Gruppen, die Anzahl der Städte und den Aufbau der „Zelle“ eintragen.

Delegate

```
class ViewController: UIViewController, UITableViewDelegate, UITableViewDataSource{
}
```

Referenzen und Variablen:

```
  // Referenzen
@IBOutlet var tableview: UITableView!

var cities : [City] = []    // leeres Array, Liste
```

viewDidLoad:

```
override func viewDidLoad() {
    super.viewDidLoad()
    cities = CitiesDB.loadStaedteIntern()
    tableview.delegate=self
    tableview.dataSource=self
}
```

Anzahl der Gruppen:

```
func numberOfSections(in tableView: UITableView) -> Int {
   return 1
}
```

Anzahl der Städte:

```
func tableView(_ tableView: UITableView,
                              numberOfRowsInSection section: Int) -> Int {
   return cities.count

}
```

Eintragen des Textes in der Zelle:

```
func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath)
-> UITableViewCell {

  let cell: UITableViewCell = UITableViewCell(
       style: UITableViewCell.CellStyle.subtitle,
       reuseIdentifier: "reuseIdentifier")

        // Configure the cell...
  let row = (indexPath as NSIndexPath).row   // aktueller Index
  let city:City = cities[ row ]
  cell.textLabel?.text = city.name
  cell.detailTextLabel?.text = city.remark

  return cell
}
```

Beachten Sie, dass man mit dem Quellcode „UITableViewCell.CellStyle.subtitle“ die Zelle mit dem Style (Text, Subtext und Image) definiert.

Typen der cell-Instanz:

Insgesamt gibt es vier vordefinierte Formate:

Typ	Erläuterung
UITableViewCellStyle.**default**	Bild links, Titel rechts
UITableViewCellStyle.value1	Bild links, Titel in der Mitte, rechts Zusatzinformation
UITableViewCellStyle.value2	Zusatzinformation links, rechts Titel
UITableViewCellStyle.subtitle	Bild links, Titel rechts, darunter Zusatzinformation

Mögliche Elemente in der Zelle:

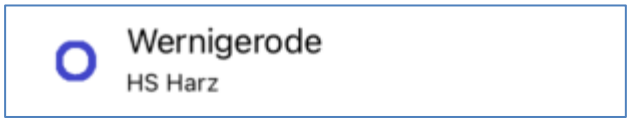
Typ	Zuweisung
textLabel	cell?.textLabel!.text ="City"
detailTextLabel	cell?.detailTextLabel!.text ="City"
imageView	cell?.imageView!.image = UIImage(named: "Bild6")

Beispiele:

default:

value1

value2



subtitle:

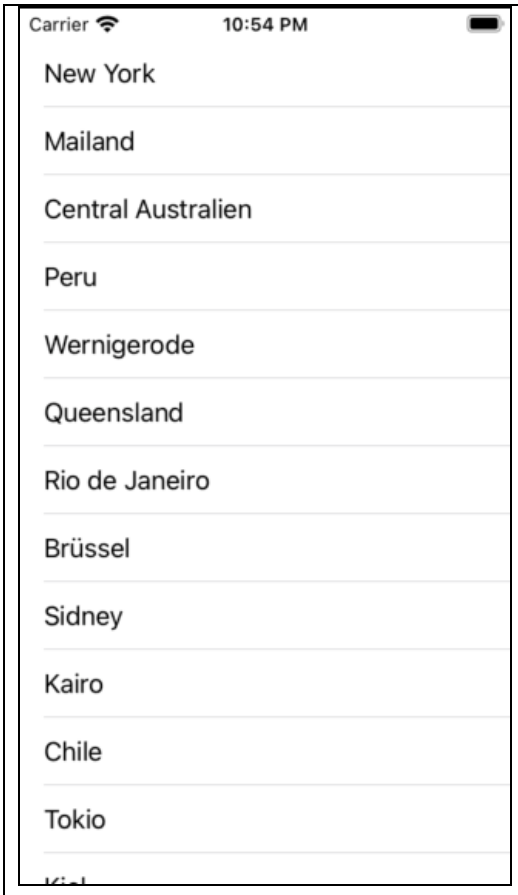


Abbildung 230 Mit dem Namen

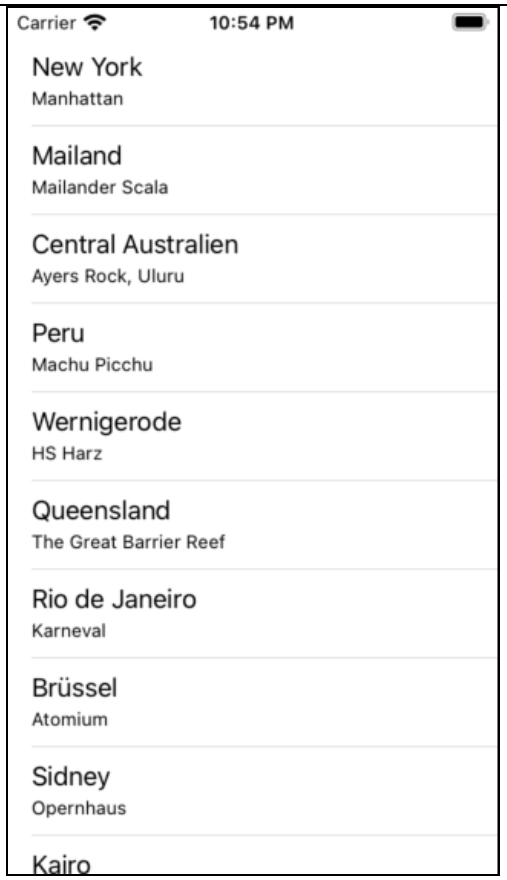


Abbildung 231 Mit Namen und Bemerkung

12.2 TableView mit Symbolen

Einfügen der Symbole aus der Datei „mycellsymbols.zip“ in den Projektordner „assert“ per Drag&Drop“. **Bitte beachten Sie, dass Sie immer das Bild ohne Nummer ins Projekt ziehen**.

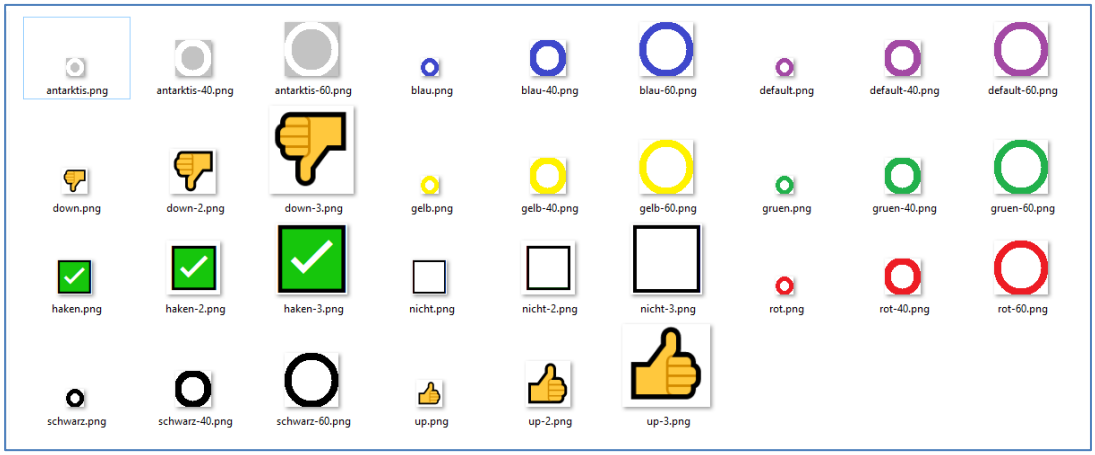


Abbildung 232 Symbole für die TableView

Ändern der Methode für die Zell-Darstellung:

```
func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath)
-> UITableViewCell {
   let cell: UITableViewCell =
        UITableViewCell(style: UITableViewCell.CellStyle.subtitle,
        reuseIdentifier: "reuseIdentifier")


        // Configure the cell...
   let row = (indexPath as NSIndexPath).row
   let city:City = cities[ row ]
   cell.textLabel?.text = city.name
   cell.detailTextLabel?.text = city.remark

    // Symbole, ja nach Kontigent
    switch city.continent {
      case Continent.AFRICA:
         cell.imageView?.image = UIImage(named: "schwarz")

      case Continent.AMERICA:
         cell.imageView?.image = UIImage(named: "rot")

      case Continent.ASIA:
         cell.imageView?.image = UIImage(named: "gelb")

      case Continent.AUSTRALIA:
         cell.imageView?.image = UIImage(named: "gruen")

      case Continent.EUROPE:
         cell.imageView?.image = UIImage(named: "blau")

      case Continent.ANTARKTIS:
         cell.imageView?.image = UIImage(named: "antarktis")

      default:
         cell.imageView?.image = UIImage(named: "default")
    }

    return cell
}
```

Ergebnis:

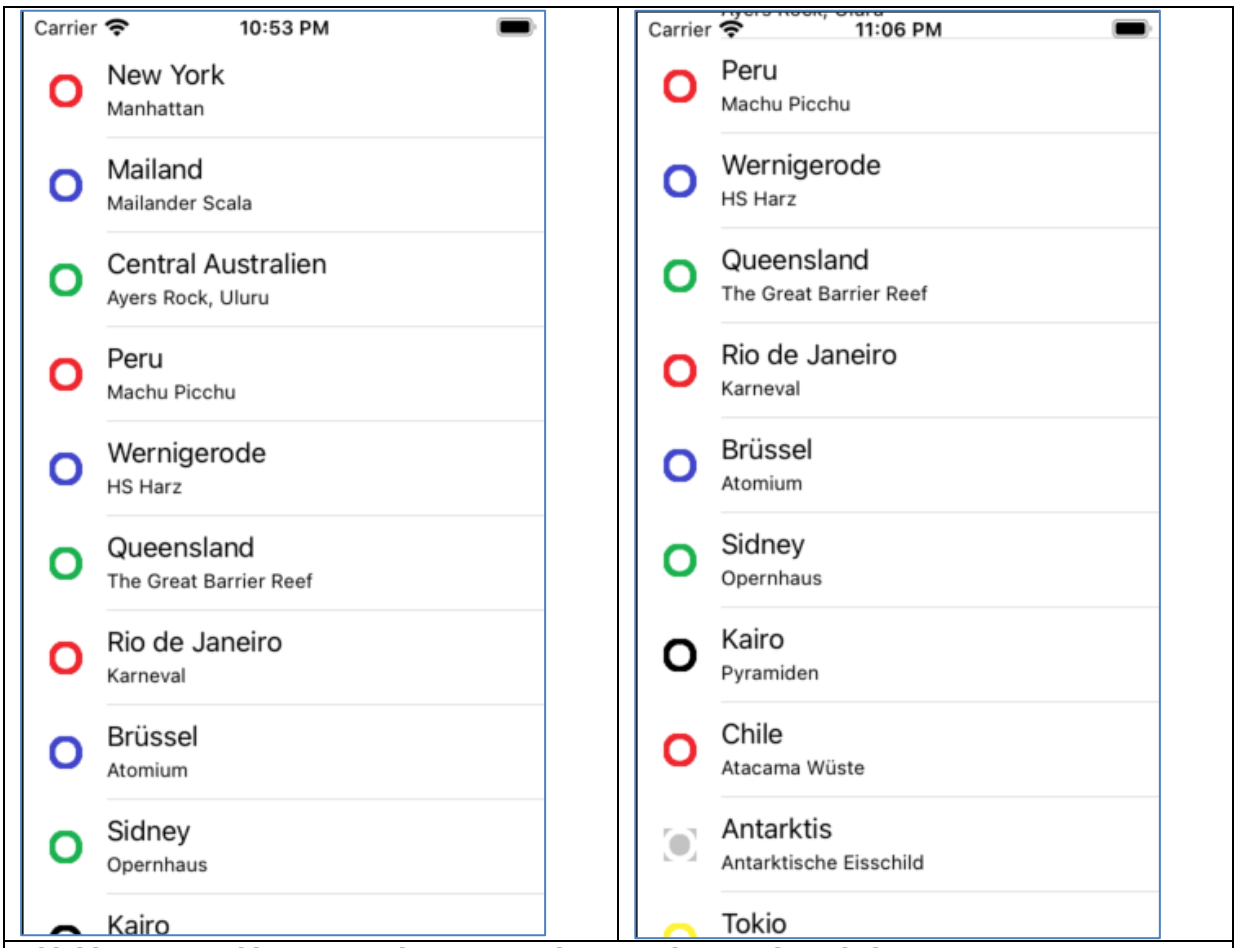


Abbildung 233 TableView mit dem Namen, der Bemerkung und Symbolen

12.3 TableView mit Gruppen (TableView02Group)

Dieses Kapitel beschreibt, wie man Gruppen in der Tabelle darstellen kann. Der Aufbau ist grundsätzlich mit dem ersten Beispiel identisch.

Vorschau

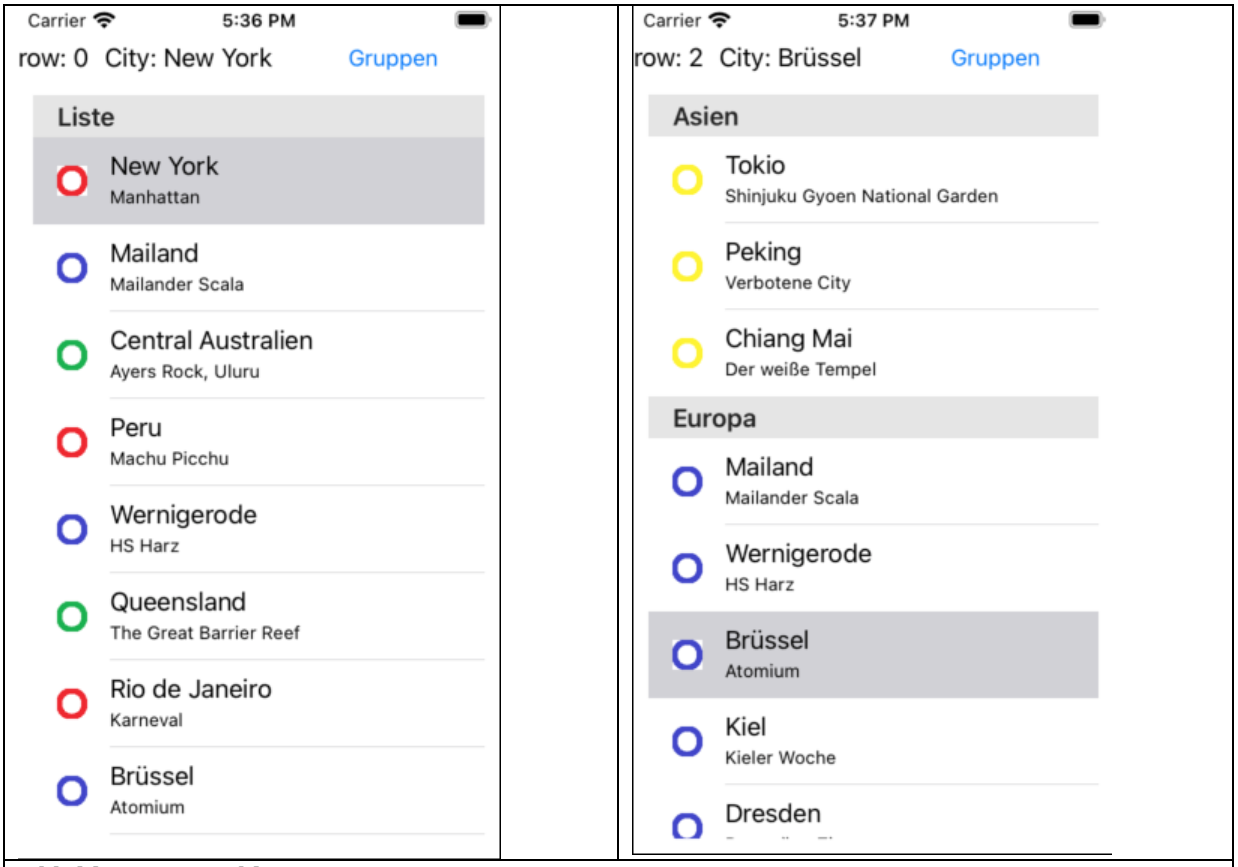


Abbildung 234 TableView mit Gruppen

12.3.1 Anlegen eines Projektes

Legen Sie ein normales Projekt „App“ an und speichern dieses.

12.3.2 Schalterleiste einfügen

Fügen Sie einen horizontalen Stackview oben in den ViewController ein. Als Constraint geben Sie ein:

- Top 0
- Left: 0
- Right: 0
- Bottom: 0

Fügen Sie zwei Labels und einen Button in den Stack. Die Labels dienen als Anzeige der aktuellen Zeile, der Schalter wechselt den Gruppenmodus.

12.3.3 TableView einfügen

Fügen Sie nun aus dem Lib-Dialog eine TableView in den ViewController. Als Constraint geben Sie ein:

- Top 10
- Left: 10
- Right: 10
- Bottom: 10

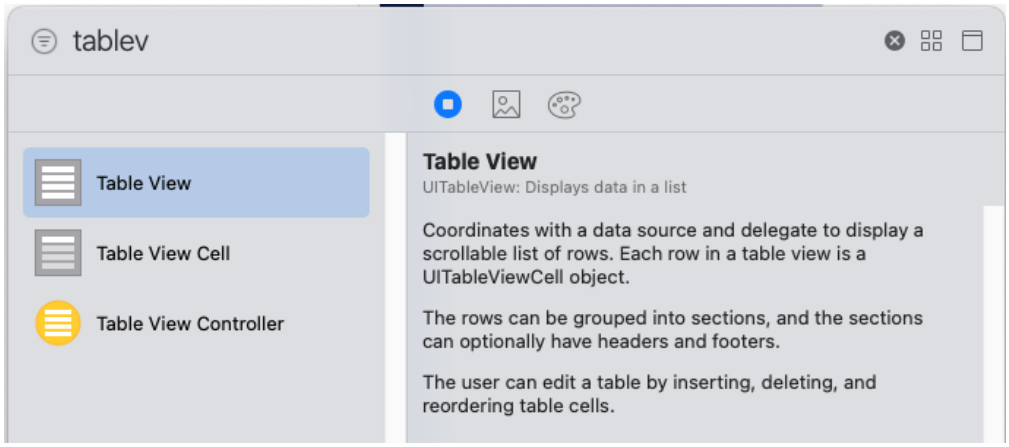


Abbildung 235 TableView einfügen

Das Projekt müsste dann so aussehen:

Abbildung 236 Aktueller Stand mit dem StackView und der TableView

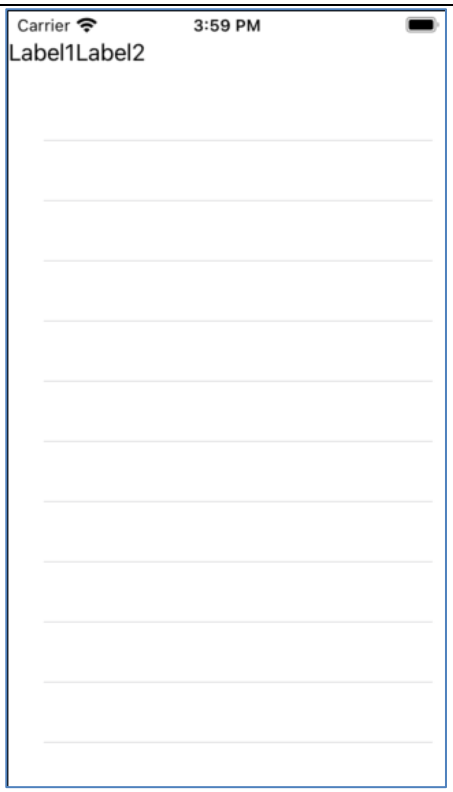


Abbildung 237 Leere TableView

Im letzten Schritt für die UI erstellt man die Referenzen der UITableView, der beiden Labels.

12.3.4 Hilfsklassen für die TableView

In der Tabelle sollen Städte angezeigt werden. Dazu brauchen wir eine Klasse „City“ und eine Verwaltungsklasse „CitiesDB“.

Anlegen einer neuen Klasse „City.swift“

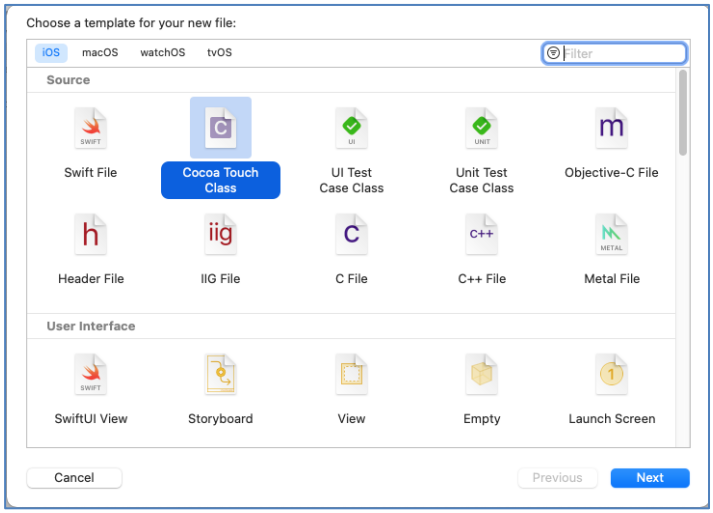


Abbildung 238 Einfüge-Dialog für die Klasse City.swift (Cmd+N)

Nun den Namen und den Typ eintragen:

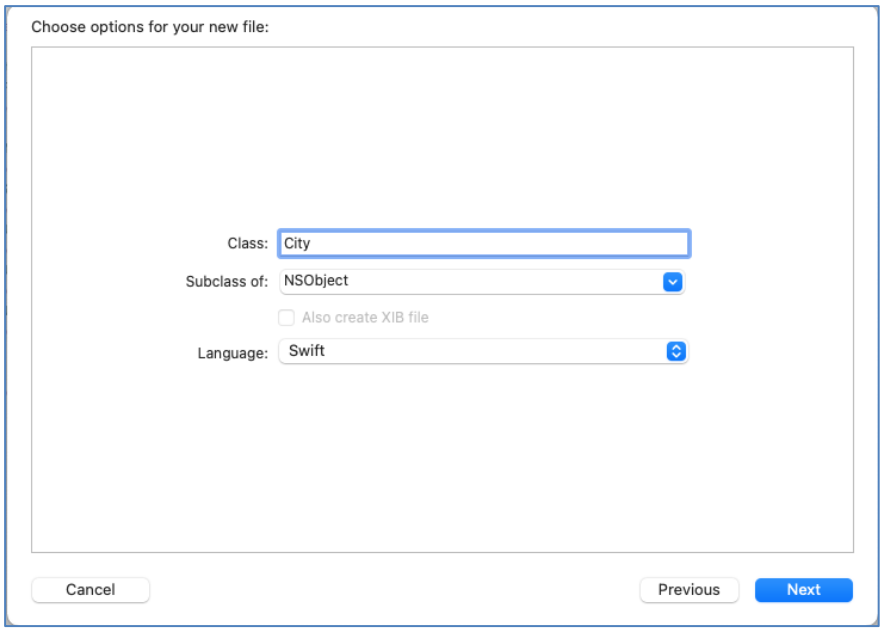


Abbildung 239 Name und Typ der Klasse City

Nun noch eine zweite Klasse „Enums.swift“ und eine dritte Klasse „CitiesDB.swift“ einfügen.
In der Klasse „Enums.swift“ werden die Aufzählungen, Enumeration eingetragen:

```
import UIKit


enum Continent:Int {          // Deklaration
  case  AFRICA=1, AMERICA, ASIA, AUSTRALIA, EUROPE, ANTARTKIS
} }

enum Visited : Int{          // Deklaration
```

```
  case  NONE=0, DOWN, AVERAGE, UP
}
```

Quellcode für Enums.swift

Eine Enumeration ist im Quellcode wesentlich sicherer als ein int-Wert mit Konstanten. Das wäre ein sehr schlechter Stil.

```
import UIKit

class City: NSObject {
      // Attribute
   var continent: Continent=Continent.AFRICA
   var name : String = ""
   var remark : String = ""
   var visited:Visited = Visited.NONE

     // Konstruktor
   init(_ continent: Continent,_ name:String, _ remark:String, _ visited:Visited){
      self.continent = continent
      self.name = name
      self.remark = remark
      self.visited = visited
   }  // init

}
```

Quellcode für City.swift

```
import UIKit

class CitiesDB: NSObject {

   public static func loadCitiesIntern()  -> Array<City> {
     var cities=[City]()
     cities.append( City( Continent.AMERICA ,  "New York",  "Manhattan", Visited.NONE)  )
     cities.append( City( Continent.EUROPE , "Mailand", "Mailander Scala", Visited.AVERAGE))
     cities.append( City( Continent.AUSTRALIA ,"Central Australien","Ayers Rock, Uluru",
Visited.NONE)  )
     cities.append( City( Continent.AMERICA ,"Peru",  "Machu Picchu",  Visited.UP)  )
     cities.append( City( Continent.EUROPE ,"Wernigerode",  "HS Harz",  Visited.DOWN)  )
     cities.append( City( Continent.AUSTRALIA ,"Queensland",  "The Great Barrier Reef",
Visited.NONE)  )
     cities.append( City( Continent.AMERICA ,"Rio de Janeiro", "Karneval",  Visited.DOWN)  )
     cities.append( City( Continent.EUROPE ,"Brüssel",   "Atomium",  Visited.UP)  )
     cities.append( City( Continent.AUSTRALIA ,"Sidney",  "Opernhaus",  Visited.UP)  )
     cities.append( City( Continent.AFRICA ,"Kairo",  "Pyramiden",  Visited.NONE)  )
     cities.append( City( Continent.AMERICA ,"Chile",  "Atacama Wüste",  Visited.UP)  )
     cities.append( City( Continent.ASIA ,"Tokio",  "Shinjuku Gyoen National Garden",
Visited.DOWN)  )
     cities.append( City( Continent.EUROPE ,"Kiel",  "Kieler Woche",  Visited.UP)  )
     cities.append( City( Continent.AFRICA ,"Hunsbergen","Apollo 11 Höhle",  Visited.NONE) )
     cities.append( City( Continent.EUROPE ,"Dresden",  "Das grüne Zimmer",  Visited.NONE) )
     cities.append( City( Continent.AFRICA ,"Marrakesch",  "Botanische Garten Jardin
Majorelle",  Visited.DOWN)  )
     cities.append( City( Continent.AMERICA ,"San Fransisco",  "Golden Gate Bridge",
Visited.UP)  )
     cities.append( City( Continent.EUROPE ,"Halberstadt",  "John Cage",  Visited.NONE)  )
     cities.append( City( Continent.ASIA ,"Peking",  "Verbotene Stadt",  Visited.NONE)  )
     cities.append( City( Continent.EUROPE ,"Venezuela",  "Angel Falls",  Visited.UP)  )
     cities.append( City( Continent.AMERICA ,"Bilbao",  "Guggenheim-Museum",  Visited.UP) )
```

```
    cities.append( City( Continent.ASIA ,"Chiang Mai",  "Der weiße Tempel",  Visited.UP) )
    cities.append( City( Continent.AMERICA ,"Rio de Janeiro",  "Zuckerhut",  Visited.NONE))
    return cities
}
```

Quellcode für CitiesDB.swift

12.3.5 Symbole für die TableView

Einfügen der Symbole aus der Datei „mycellsymbols.zip“ in den Projektordner „assert“ per Drag&Drop“. **Bitte beachten Sie, dass Sie immer das Bild ohne Nummer ins Projekt ziehen**.

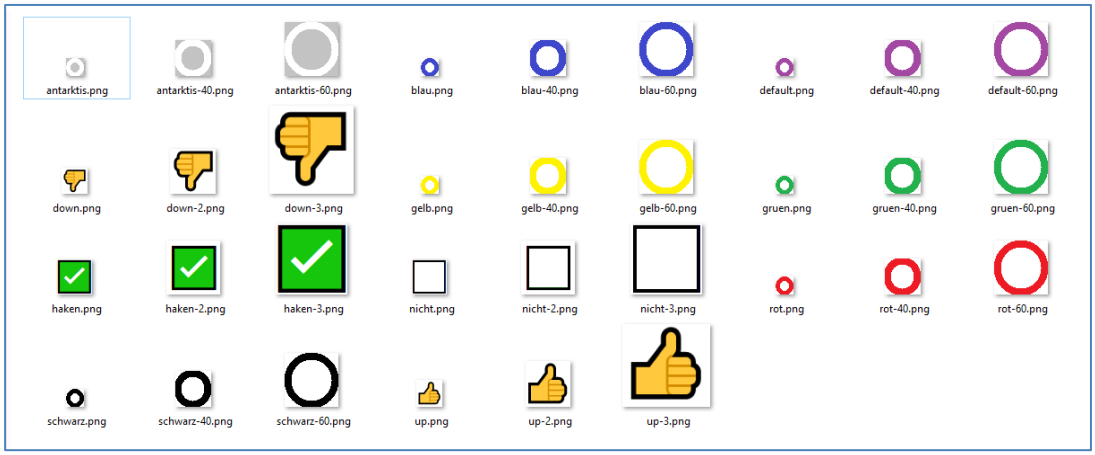


Abbildung 240 Symbole für die TableView

12.3.6 TableView mit einer Liste füllen

Da der ViewController kein TableViewController ist, müssen die Delegate manuell eingetragen werden. Danach muss also nur die Liste der Städte holen, die UITableView mit den Methoden verknüpfen (Interface) und die Anzahl der Gruppen, die Anzahl der Städte und den Aufbau der „Zelle“ eintragen.

Fügen Sie oben in der Klassendefinition folgende Protokolle, Delegate ein:
class ViewController: UIViewController , **UITableViewDelegate, UITableViewDataSource**

Methoden der TableView:

```
func numberOfSections(in tableView: UITableView) -> Int {
    return 1
}

func tableView(_ tableView: UITableView, numberOfRowsInSection section:
Int) -> Int {
    return cities.count
}

    // SelectecIndexChanged
```

```
func tableView(_: UITableView, didSelectRowAt: IndexPath) {
   // label setzen
}

     // Zelle aufbauen
func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath)
-> UITableViewCell {
      …
      return cell
}
```

Referenzen und Variablen:

```
  // Referenzen
@IBOutlet var tableview: UITableView!

@IBOutlet var label1: UILabel!
@IBOutlet var label2: UILabel!

var cities : [City] = []    // leeres Array, Liste
```

viewDidLoad:

```
func viewDidLoad() {
  super.viewDidLoad()

  cities = CitiesDB.loadCitiesIntern()

  tableview.delegate=self
  tableview.dataSource=self

}
```

Anzahl der Gruppen:

```
func numberOfSections(in tableView: UITableView) -> Int {
   return 1
}
```

Anzahl der Städte:

```
override func tableView(_ tableView: UITableView,
                            numberOfRowsInSection section: Int) -> Int {
   return cities.count

}
```

SelectedIndex:

```
// SelectecIndexChanged
func tableView(_: UITableView, didSelectRowAt: IndexPath) {
  let row = (didSelectRowAt as NSIndexPath).row
```

```
  let city:City = cities[row]
  label1.text="row: "+String(row)
  label2.text="City: "+city.name
}
```

Eintragen des Textes in der Zelle:

```
func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath)
-> UITableViewCell {

  let cell: UITableViewCell = UITableViewCell(
     style: UITableViewCell.CellStyle.subtitle,
     reuseIdentifier: "reuseIdentifier")


        // Configure the cell...
  let row = (indexPath as NSIndexPath).row
  let city:City = cities[ row ]
  cell.textLabel?.text = city.name
  cell.detailTextLabel?.text = city.remark

  switch city.continent {
     case Continent.AFRICA:
        cell.imageView?.image = UIImage(named: "schwarz")

     case Continent.AMERICA:
        cell.imageView?.image = UIImage(named: "rot")
     case Continent.ASIA:
        cell.imageView?.image = UIImage(named: "gelb")
     case Continent.AUSTRALIA:
        cell.imageView?.image = UIImage(named: "gruen")
     case Continent.EUROPE:
        cell.imageView?.image = UIImage(named: "blau")
     case Continent.ANTARTKIS:
        cell.imageView?.image = UIImage(named: "antarktis")

     default:
        cell.imageView?.image = UIImage(named: "default")
     }
  }
  return cell
}
```

Beachten Sie, dass man mit dem Quellcode „UITableViewCell.CellStyle.subtitle“ die Zelle mit dem Style (Text, Subtext und Image) definiert.

Typen der cell-Instanz:

Insgesamt gibt es vier vordefinierte Formate:

Typ	**Erläuterung**
UITableViewCellStyle.**default**	Bild links, Titel rechts
UITableViewCellStyle.value1	Bild links, Titel in der Mitte, rechts Zusatzinformation
UITableViewCellStyle.value2	Zusatzinformation links, rechts Titel

UITableViewCellStyle.subtitle	Bild links, Titel rechts, darunter Zusatzinformation

Mögliche Elemente in der Zelle:

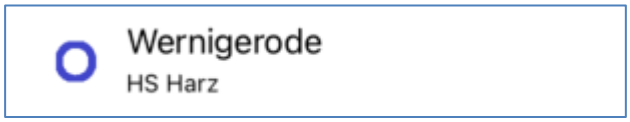
Typ	**Zuweisung**
textLabel	cell?.textLabel!.text ="Stadt"
detailTextLabel	cell?.detailTextLabel!.text ="Stadt"
imageView	cell?.imageView!.image = UIImage(named: "Bild6")

Beispiele:

default:

value1

value2



subtitle:

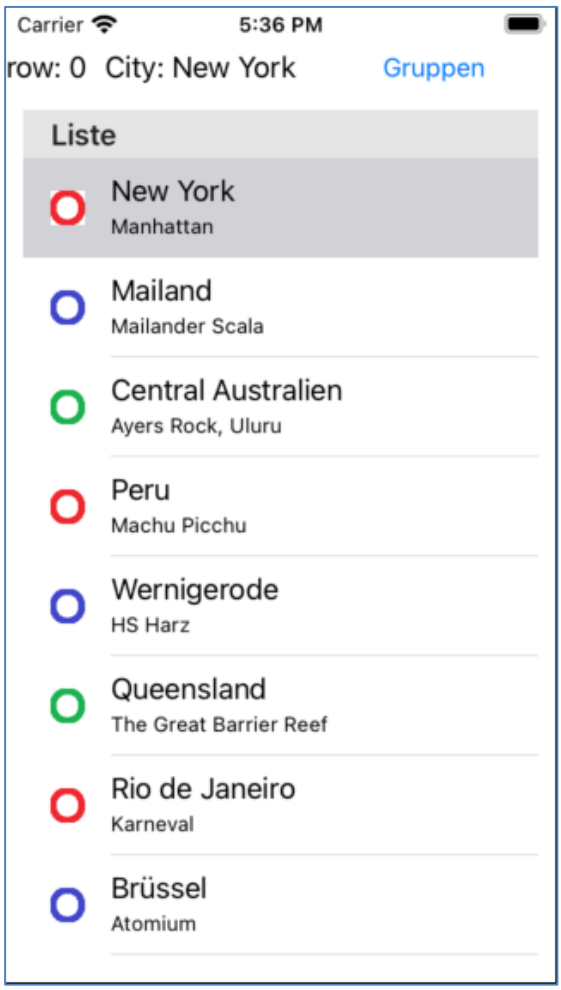


Abbildung 241 Aktueller Stand (Label Liste entfernen!)

Hier noch einmal der komplette Quellcode:

//

```
import UIKit

class ViewController: UIViewController , UITableViewDelegate,
UITableViewDataSource {

    var cities : [City] = []

    @IBOutlet var tableview: UITableView!
    @IBOutlet var bngroup: UIButton!


    @IBOutlet var label1: UILabel!
    @IBOutlet var label2: UILabel!

    @IBAction func bngroupclick(_ sender: UIButton) {
    }
    override func viewDidLoad() {
        super.viewDidLoad()
```

```
        cities = CitiesDB.loadCitiesIntern()

        tableview.delegate=self
        tableview.dataSource=self
    }

    func numberOfSections(in tableView: UITableView) -> Int {
       return 1
    }

    func tableView(_ tableView: UITableView, numberOfRowsInSection section:
Int) -> Int {
        return cities.count
    }



      // SelectecIndexChanged
     func tableView(_: UITableView, didSelectRowAt: IndexPath) {
         let row = (didSelectRowAt as NSIndexPath).row
        let city:City = cities[row]
        label1.text="row: "+String(row)
        label2.text="City: "+city.name
     }

    func tableView(_ tableView: UITableView, cellForRowAt indexPath:
IndexPath) -> UITableViewCell {


        let cell: UITableViewCell = UITableViewCell(style:
UITableViewCell.CellStyle.subtitle
                                                        , reuseIdentifier:
"reuseIdentifier")


        // Configure the cell...
        let row = (indexPath as NSIndexPath).row
        let city:City = cities[ row ]
        cell.textLabel?.text = city.name


        cell.detailTextLabel?.text = stadt.remark

        switch city.continent {
            case Continent.AFRICA:
                cell.imageView?.image = UIImage(named: "schwarz")

            case Continent.AMERICA:
                cell.imageView?.image = UIImage(named: "rot")
            case Continent.ASIA:
                cell.imageView?.image = UIImage(named: "gelb")
            case Continent.AUSTRALIA:
                cell.imageView?.image = UIImage(named: "gruen")
            case Continent.EUROPE:
                cell.imageView?.image = UIImage(named: "blau")
            case Continent.ANTARTKIS:
                cell.imageView?.image = UIImage(named: "antarktis")
```

```
            default:
                cell.imageView?.image = UIImage(named: "default")
        }

        return cell
    }


}
```

Nun erfolgt der Umbau auf die Gruppen:

12.3.7 Gruppenumbau

Defintionen:

```
var group:Bool = false

struct CitySection {
   var continent: Continent
   var cities: [City]
}
var sections = [CitySection]()
```

Es gibt nun eine „Klasse“, Struktur, die den Gruppennamen „kontinent“ hat und in der Liste die einzelnen Städte pro Kontinent speichert. Das Feld sections speichert alle Gruppenfelder.

viewDidLoad (neue Anweisungen):

```
override func viewDidLoad() {

   // Gruppiere die Staedte nach Kontinente
   // die Gruppierung wird mittels einer Hashtable durchgeführt!
   let groups = Dictionary(grouping: self.cities) {
         (city )  in return city.continent
   }

     // Eintragen in die GruppenArrays
    self.sections = groups.map {
        (key, value) in return CitySection(continent: key, cities: value)
    }
}
```

bngroupclick:

```
@IBAction func bngroupclick(_ sender: UIButton) {
    group = !group   // Umschalten
    tableview.reloadData()
}
```

Anzahl der Gruppen:

```
func numberOfSections(in tableView: UITableView) -> Int {
   if group {
```

```
      return sections.count
   }
   else{
      return 1
   }
}
```

Anzahl der Städte:

```
override func tableView(_ tableView: UITableView,
                              numberOfRowsInSection section: Int) -> Int {
   if group {
       // // section ist Index für das GruppenArray
      let sectionArray = self.sections[ section]
      return sectionArray.cities.count
   }
   else{
      return cities.count
   }
}
```

SelectedIndex:

```
// SelectecIndexChanged
func tableView(_: UITableView, didSelectRowAt: IndexPath) {
   if group {
      if let indexPath = tableview.indexPathForSelectedRow {
        let section = self.sections[indexPath.section]
        let city = section.cities[indexPath.row]
        label1.text="row: "+String(indexPath.row)
        label2.text="City: "+city.name
      }
   }
   else {
      if let indexPath = tableview.indexPathForSelectedRow {
        let city:City = cities[(indexPath as NSIndexPath).row]
        label1.text="row: "+String(indexPath.row)
        label2.text="City: "+city.name
      }
   }
}
```

Abfrage der Überschrift pro GRUPPE (neu):

```
func tableView(_ tableView: UITableView, titleForHeaderInSection section:
Int) -> String? {
   if group {
       // section ist Index für das GruppenArray
      let sectionArray = self.sections[ section]
      let continent = sectionArray.continent  // Gruppenname
      return getContinentText(continent)
   }
   else {
     return "Liste"
}
```

Eintragen des Textes in der Zelle:

```
func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath)
-> UITableViewCell {

  let cell: UITableViewCell = UITableViewCell(
     style: UITableViewCell.CellStyle.subtitle,
     reuseIdentifier: "reuseIdentifier")


        // Configure the cell...
  let row = (indexPath as NSIndexPath).row

  let city:City
  if group {
     let sectionsArray = self.sections[indexPath.section]
     city = sectionsArray.cities[row]
  }
  else {
    city = cities[ row ]
  }

  cell.textLabel?.text = city.name
  cell.detailTextLabel?.text = city.remark

  switch city.continent {
     case Continent.AFRICA:
        cell.imageView?.image = UIImage(named: "schwarz")

     case Continent.AMERICA:
        cell.imageView?.image = UIImage(named: "rot")
     case Continent.ASIA:
        cell.imageView?.image = UIImage(named: "gelb")
     case Continent.AUSTRALIA:
        cell.imageView?.image = UIImage(named: "gruen")
     case Continent.EUROPE:
        cell.imageView?.image = UIImage(named: "blau")
     case Continent.ANTARTKIS:
        cell.imageView?.image = UIImage(named: "antarktis")

     default:
        cell.imageView?.image = UIImage(named: "default")
     }
  }
  return cell
}
```

Hier noch einmal der komplette Quellcode

//

```
import UIKit

class ViewController: UIViewController , UITableViewDelegate, UITableViewDataSource
{

    var cities : [City] = []
    var group:Bool = false
```

```
    struct CitySection {
        var continent: Continent
        var cities: [City]
    }
    var sections = [CitySection]()

    @IBOutlet var tableview: UITableView!
    @IBOutlet var bngroup: UIButton!


    @IBOutlet var label1: UILabel!
    @IBOutlet var label2: UILabel!


    override func viewDidLoad() {
        super.viewDidLoad()

        cities = CitiesDB.loadStaedteIntern()

        tableview.delegate=self
        tableview.dataSource=self

        // Gruppiere die Staedte nach Kontinente
        let groups = Dictionary(grouping: self.cities) {
            (city )  in return city.continent
        }

        // Eintragen in die GruppenArrays
        self.sections = groups.map {
            (key, value) in return CitySection(continent: key, cities: value)
        }

    }

    @IBAction func bngroupclick(_ sender: UIButton) {
        group = !group
        tableview.reloadData()
    }


    func numberOfSections(in tableView: UITableView) -> Int {
        if group {
            return sections.count
        }
        else{
              return 1
          }
    }

    func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) ->
Int {
        if group {
            let sectionArray = self.sections[ section]  // section ist Index für
das GruppenArray
            return sectionArray.cities.count  // siehe definition der oberen
Structur
        }
        else{
              return cities.count
          }

    }
```

```
    // SelectecIndexChanged
   func tableView(_: UITableView, didSelectRowAt: IndexPath) {

      if group {
          if let indexPath = tableview.indexPathForSelectedRow {
             let section = self.sections[indexPath.section]
              let city = section.cities[indexPath.row]
              label1.text="row: "+String(indexPath.row)
              label2.text="City: "+city.name
          }
      }
      else {
          if let indexPath = tableview.indexPathForSelectedRow {
              let city:City = cities[(indexPath as NSIndexPath).row]
              label1.text="row: "+String(indexPath.row)
              label2.text="City: "+city.name
          }
      }

   }  // SelectecIndexChanged


    // Abfrage der Überschrift pro GRUPPE !!!!
func tableView(_ tableView: UITableView, titleForHeaderInSection section: Int) ->
String? {
    if group {
        let sectionArray = self.sections[ section]  // section ist Index für das
GruppenArray
        let continent = sectionArray.continent  // Gruppenname
        return getContinentText(continent)
    }
    else {
         return "Liste"
    }
}




    func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) ->
UITableViewCell {

        let cell: UITableViewCell = UITableViewCell(style:
UITableViewCell.CellStyle.subtitle
                                                  , reuseIdentifier:
"reuseIdentifier")


        // Configure the cell...
        let row = (indexPath as NSIndexPath).row

        let city:City
        if group {
            let sectionsArray = self.sections[indexPath.section]
            city = sectionsArray.cities[row]
        }
        else {
            city = cities[ row ]
         }


        cell.textLabel?.text = city.name
        cell.textLabel?.textColor = UIColor(named: "redColor")
```

```
        //cell.textLabel?.backgroundColor = [UIColor, clearColor];

        //cell.textLabel?.textColor = new UIColor("green")


        cell.detailTextLabel?.text = city.remark

        switch city.continent {
            case Continent.AFRICA:
                cell.imageView?.image = UIImage(named: "schwarz")
            case Continent.AMERICA:
                cell.imageView?.image = UIImage(named: "rot")
            case Continent.ASIA:
                cell.imageView?.image = UIImage(named: "gelb")
            case Continent.AUSTRALIA:
                cell.imageView?.image = UIImage(named: "gruen")
            case Continent.EUROPE:
                cell.imageView?.image = UIImage(named: "blau")
            case Continent.ANTARTKIS:
                cell.imageView?.image = UIImage(named: "antarktis")

            default:
                cell.imageView?.image = UIImage(named: "default")
        }

        return cell
    }
}
```

12.3.8 Ergebnis

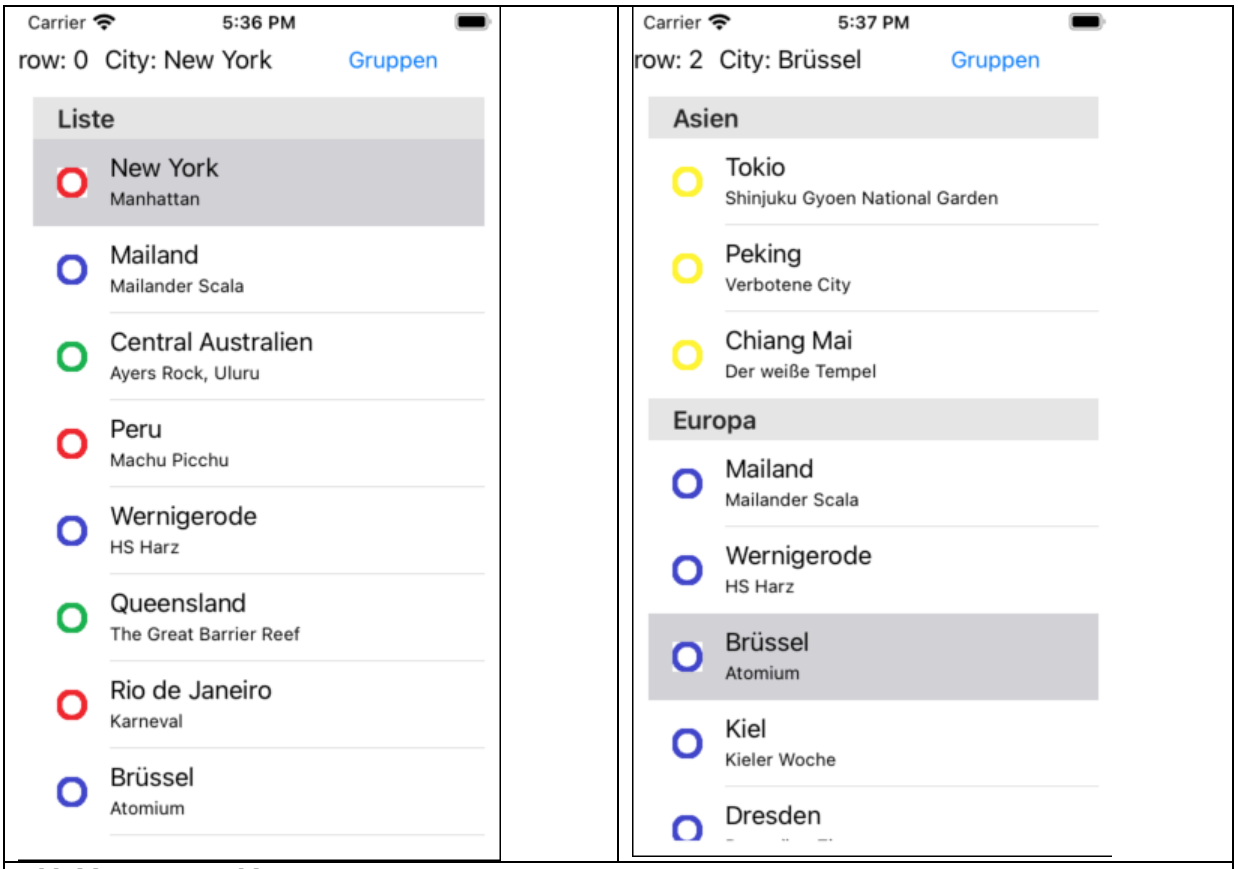


Abbildung 242 TableView mit Gruppen

12.3.9 Farbe setzen

Um das Aussehen besser zu gestalten, kann man die Farben der Label in der Zelle setzen. Dazu muss man aber die Farben definieren.

Vorgehensweise:

- Aktivieren des Projektseintrags „Assets.xcassets
- Aufruf des Menus Editor / Add new Asset / Color Set
- Nun wird eine neue Farbe eingetragen.
- Ändern des Namens (Beispiel redColor)
- Anklicken „Any Appearance“ zum Setzen der Farbe
    - Diese kann mit den Slider geändert werden
    - Oder mit dem Schalter „Show Color Panel“

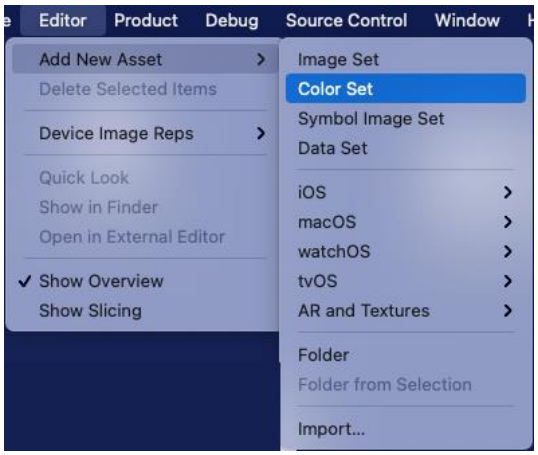


Abbildung 243 Menuaufruf zum Eintragen einer Farbe

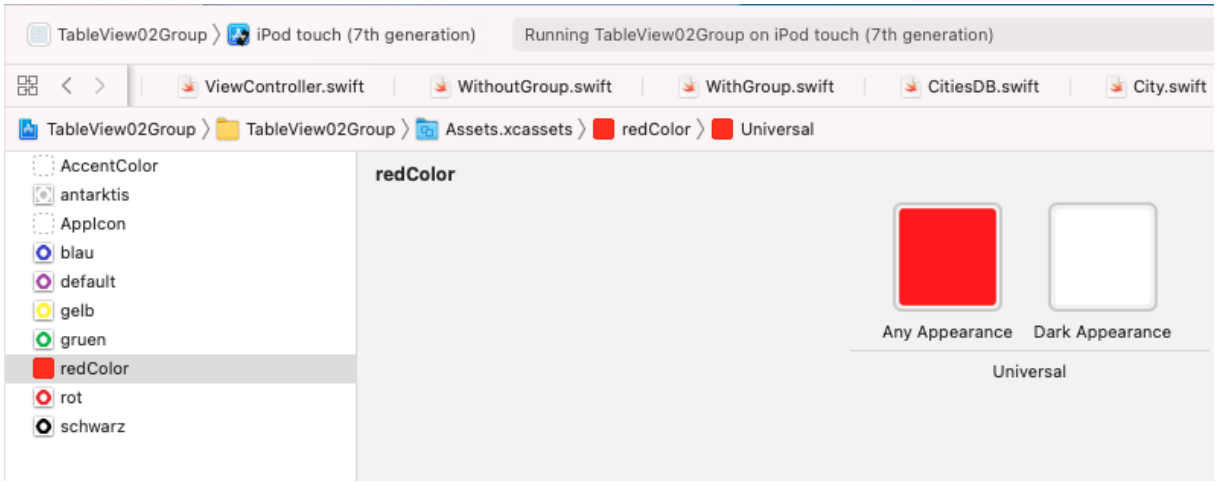


Abbildung 244 Fertige Farbe für die TableView

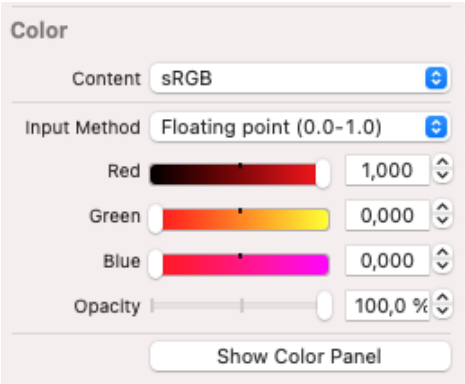


Abbildung 245 Ändern der Farbe mittels Slider (Red, Green, Blue)

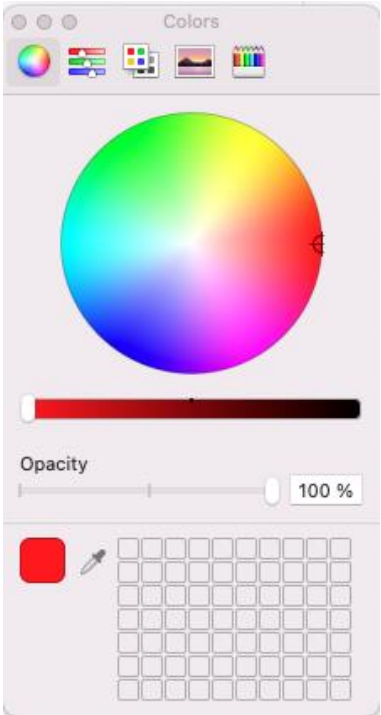


Abbildung 246 Anzeige der Color Panel

Setzen der Farbe in Swift:

```
cell.textLabel?.textColor = UIColor(named: "redColor")
cell.textLabel?.backgroundColor = UIColor(named: "yellowColor")
```

12.4 TableView mit Schaltern (Edit, Delete, New, TableView03)

Dieses Kapitel beschreibt, wie man die Einträge in einer Tabelle bearbeiten, löschen oder neue einfügen kann. Der Aufbau ist grundsätzlich mit dem ersten Beispiel identisch. Hier wird nur eine TableView in einem normalen UIViewController eingefügt.

Vorschau

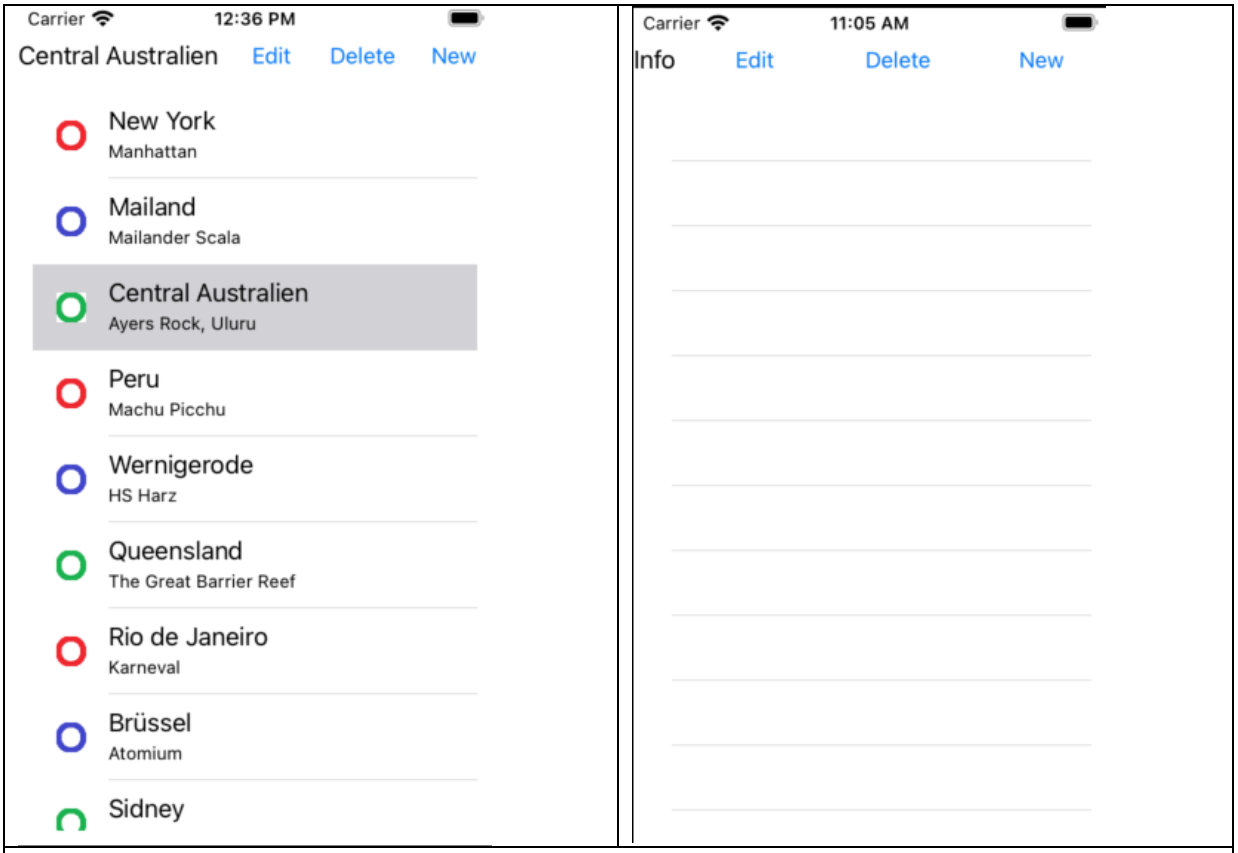
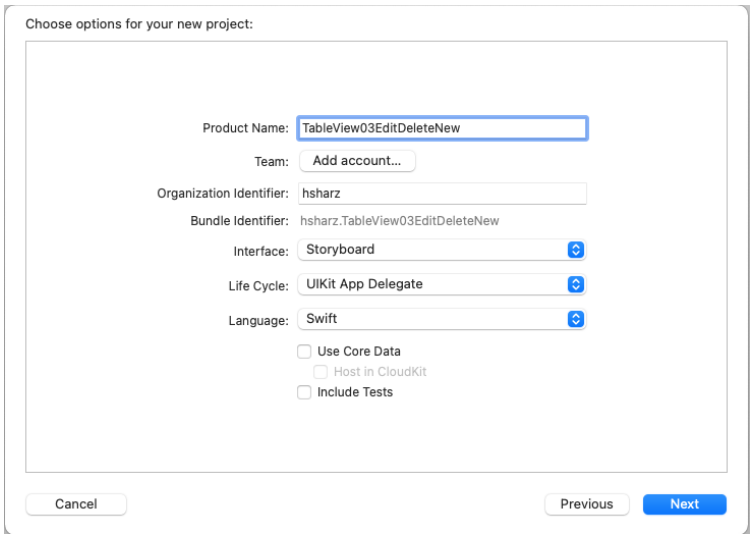


Abbildung 247 TableView mitSchaltern

12.4.1 Anlegen eines Projektes

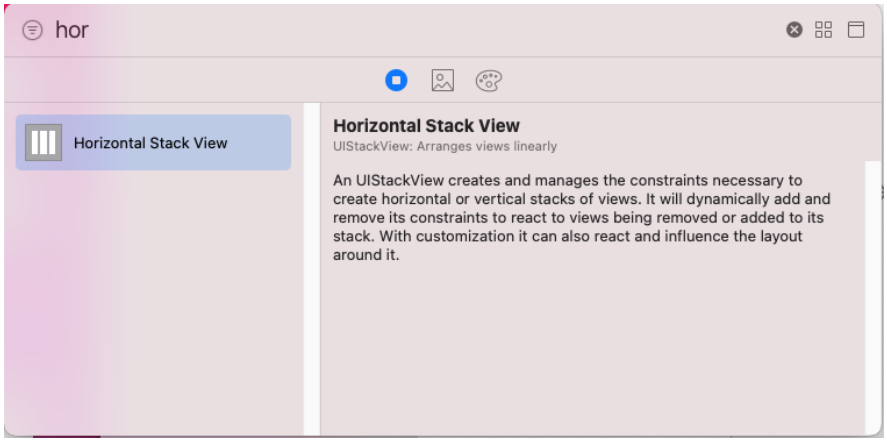
Legen Sie ein normales Projekt „App“ an und speichern Sie dieses.



12.4.2 Schalterleiste einfügen

Fügen Sie einen horizontalen Stackview oben in den ViewController ein. Als Constraint geben Sie ein:

- Top 0
- Left: 0
- Right: 0
- Bottom: 0



Fügen Sie ein Label und drei Buttons in den Stack. Das Label dient als Anzeige der aktuellen Zeile, die Schalter dienen zum Bearbeiten:

- Edit hier wird zu einem zweiten View umgeschaltet
- Delete hier wird ein NoYes-Dialog benutzt
- New hier wird zu einem zweiten View umgeschaltet

12.4.3 TableView-Controller einfügen

Fügen Sie nun aus dem Lib-Dialog eine TableView in den ViewController. Als Constraint geben Sie ein:

- Top 10
- Left: 10
- Right: 10
- Bottom: 10

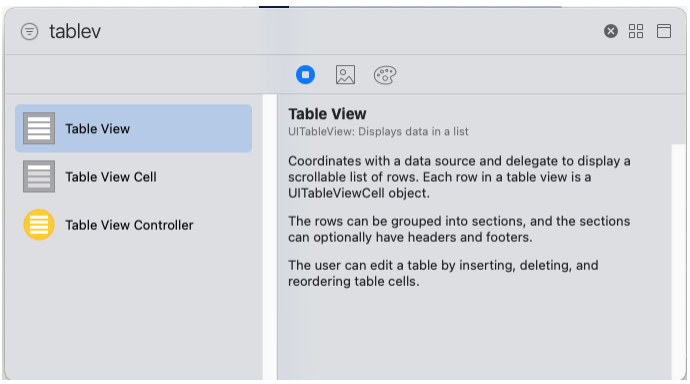


Abbildung 248 TableView einfügen

Das Projekt müsste dann so aussehen:

Abbildung 249 Aktueller Stand mit dem StackView und der TableView

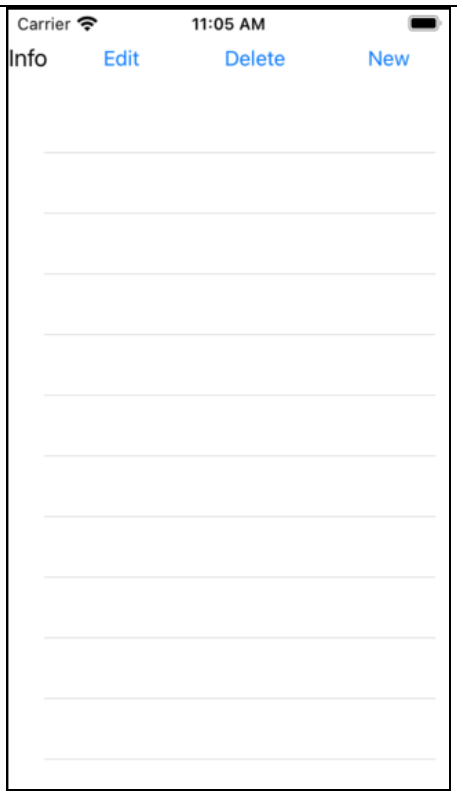


Abbildung 250 Leere TableView

Im letzten Schritt für die UI erstellt man die Referenzen der UITableView und der zwei Schalter. Für den Schalter „Delete“ wird eine EventMethode eingefügt.

12.4.4 Hilfsklassen für die TableView

In der Tabelle sollen Städte angezeigt werden. Dazu brauchen wir eine Klasse „City“ und eine Verwaltungsklasse „CitiesDB“.

Anlegen einer neuen Klasse „City.swift“

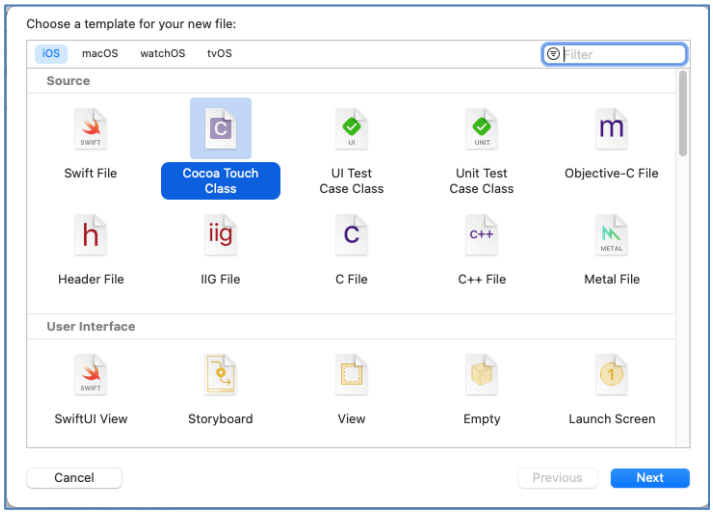


Abbildung 251 Einfüge-Dialog für die Klasse City.swift (Cmd+N)

Nun den Namen und den Typ eintragen:

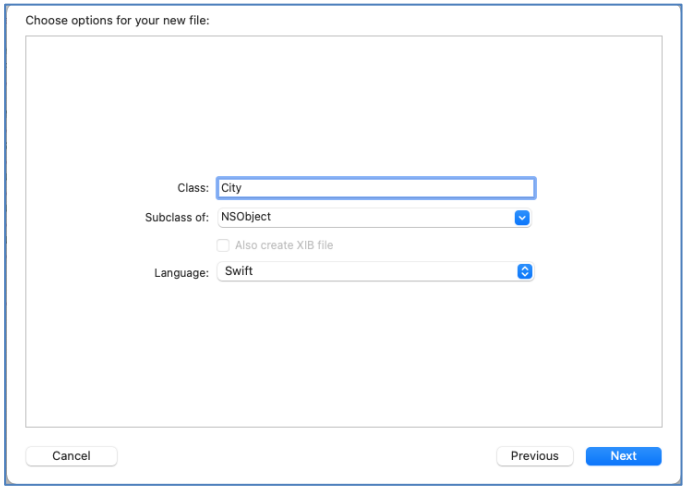


Abbildung 252 Name und Typ der Klasse City

Nun noch eine zweite Klasse „Enums.swift“ und eine dritte Klasse „CitiesDB.swift“ einfügen.
In der Klasse „Enums.swift“ werden die Aufzählungen, Enumeration eingetragen:

```
import UIKit


enum ChangeMode:Int {
     case EDIT
    case NEW
  //  case DELETE   mit yes-no Dialo
}


enum Continent:Int {        // Deklaration
  case  AFRICA=0, AMERICA, ASIA, AUSTRALIA, EUROPE, ANTARTKIS
}

let continente:[String] =
     ["Afrika","Amerika", "Asien", "Australien", "Europa", "Antartkis"]


func getContinentText(_ continent:Continent) -> String {
    switch (continent) {
      case Continent.AFRICA:
        return "Afrika"
      case Continent.AMERICA:
        return "Amerika"
      case Continent.ASIA:
        return "Asien"
      case Continent.AUSTRALIA:
        return "Australien"
      case Continent.EUROPE:
        return "Europa"
      case Continent.ANTARTKIS:
        return "Antartkis"
      default:
        return "unbekannt"
   }
}

func getContinent(_ continent:Int) -> Continent {
    switch (continent) {
       case 0:
         return Continent.AFRICA
       case 1:
         return Continent.AMERICA
       case 2:
         return  Continent.ASIA
       case 3:
         return Continent.AUSTRALIA
       case 4:
         return Continent.EUROPE
       case 5:
         return Continent.EUROPE
       default:
         return Continent.EUROPE
   }
```

```
}

enum Visited : Int{         // Deklaration
  case  NONE=0, DOWN, AVERAGE, UP
}
```

Quellcode für Enums.swift

Eine Enumeration ist im Quellcode wesentlich sicherer als ein int-Wert mit Konstanten. Das wäre ein sehr schlechter Stil.

```
import UIKit

class City: NSObject {
      // Attribute
   var continent: Continent=Continent.AFRICA
   var name : String = ""
   var remark : String = ""
   var visited:Visited = Visited.NONE

     // Konstruktor
   init(_ continent: Continent,_ name:String, _ remark:String, _ visited:Visited){
      self.continent = continent
      self.name = name
      self. remark = remark
      self. visited = visited
   }  // init

}
```

Quellcode für City.swift

```
import UIKit

class CitiesDB: NSObject {

   public static func loadCitiesIntern()  -> Array<City> {
     var cities=[City]()
     cities.append( City( Continent.AMERICA ,  "New York",  "Manhattan", Visited.NONE)  )
     cities.append( City( Continent.EUROPE , "Mailand", "Mailander Scala", Visited.AVERAGE))
     cities.append( City( Continent.AUSTRALIA ,"Central Australien","Ayers Rock, Uluru",
Visited.NONE)  )
     cities.append( City( Continent.AMERICA ,"Peru",  "Machu Picchu",  Visited.UP)  )
     cities.append( City( Continent.EUROPE ,"Wernigerode",  "HS Harz",  Visited.DOWN)  )
     cities.append( City( Continent.AUSTRALIA ,"Queensland",  "The Great Barrier Reef",
Visited.NONE)  )
     cities.append( City( Continent.AMERICA ,"Rio de Janeiro", "Karneval",  Visited.DOWN)  )
     cities.append( City( Continent.EUROPE ,"Brüssel",   "Atomium",  Visited.UP)  )
     cities.append( City( Continent.AUSTRALIA ,"Sidney",  "Opernhaus",  Visited.UP)  )
     cities.append( City( Continent.AFRICA ,"Kairo",  "Pyramiden",  Visited.NONE)  )
     cities.append( City( Continent.AMERICA ,"Chile",  "Atacama Wüste",  Visited.UP)  )
     cities.append( City( Continent.ASIA ,"Tokio",  "Shinjuku Gyoen National Garden",
Visited.DOWN)  )
     cities.append( City( Continent.EUROPE ,"Kiel",  "Kieler Woche",  Visited.UP)  )
     cities.append( City( Continent.AFRICA ,"Hunsbergen","Apollo 11 Höhle",  Visited.NONE) )
     cities.append( City( Continent.EUROPE ,"Dresden",  "Das grüne Zimmer",  Visited.NONE) )
     cities.append( City( Continent.AFRICA ,"Marrakesch",  "Botanische Garten Jardin
Majorelle",  Visited.DOWN)  )
```

```
     cities.append( City( Continent.AMERICA ,"San Fransisco",  "Golden Gate Bridge",
Visited.UP)  )
     cities.append( City( Continent.EUROPE ,"Halberstadt",  "John Cage",  Visited.NONE)  )
     cities.append( City( Continent.ASIA ,"Peking",  "Verbotene Stadt",  Visited.NONE)  )
     cities.append( City( Continent.EUROPE ,"Venezuela",  "Angel Falls",  Visited.UP)  )
     cities.append( City( Continent.AMERICA ,"Bilbao",  "Guggenheim-Museum",  Visited.UP) )
     cities.append( City( Continent.ASIA ,"Chiang Mai",  "Der weiße Tempel",  Visited.UP) )
     cities.append( City( Continent.AMERICA ,"Rio de Janeiro",  "Zuckerhut",  Visited.NONE))
     return cities
}
```

Quellcode für CitiesDB.swift

12.4.5 Symbole für die TableView

Einfügen der Symbole aus der Datei „mycellsymbols.zip“ in den Projektordner „assert“ per Drag&Drop“. **Bitte beachten Sie, dass Sie immer das Bild ohne Nummer ins Projekt ziehen**.

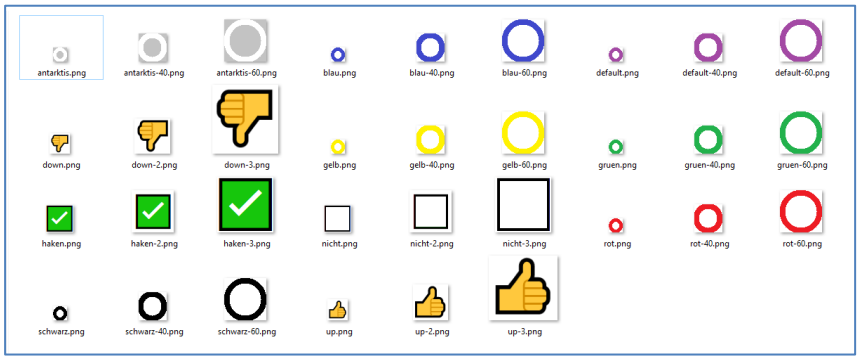


Abbildung 253 Symbole für die TableView

12.4.6 TableView mit einer Liste füllen

Da der ViewController kein TableViewController ist, müssen die Delegate manuell eingetragen werden. Danach muss also nur die Liste der Städte holen, die UITableView mit den Methoden verknüpfen (Interface) und die Anzahl der Städte und den Aufbau der „Zelle“ eintragen.

Fügen Sie oben in der Klassendefinition folgende Protokolle, Delegate, ein:
class ViewController: UIViewController , **UITableViewDelegate, UITableViewDataSource**

Methoden der TableView:

```
func numberOfSections(in tableView: UITableView) -> Int {
   return 1
}

func tableView(_ tableView: UITableView,numberOfRowsInSection section: Int) -> Int{
   return cities.count
}
```

```
      // SelectecIndexChanged
func tableView(_: UITableView, didSelectRowAt: IndexPath) {
   let row = (didSelectRowAt as NSIndexPath).row
   let city:City = cities[row]
   labelInfo.text="City: "+city.name
    // nun die beiden Schalter verfügbar machen
   bndelete.isEnabled=true
}

   // Zelle aufbauen
func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath)
-> UITableViewCell {

  let cell: UITableViewCell = UITableViewCell(
       style: UITableViewCell.CellStyle.subtitle,
       reuseIdentifier: "reuseIdentifier")


        // Configure the cell...
  let row = (indexPath as NSIndexPath).row
  let city:City = cities[ row ]
  cell.textLabel?.text = city.name
  cell.detailTextLabel?.text = city.remark

     // Image setzen
  switch city.continent {
    case Continent.AFRICA:
       cell.imageView?.image = UIImage(named: "schwarz")
    case Continent.AMERICA:
       cell.imageView?.image = UIImage(named: "rot")
    case Continent.ASIA:
       cell.imageView?.image = UIImage(named: "gelb")
    case Continent.AUSTRALIA:
       cell.imageView?.image = UIImage(named: "gruen")
    case Continent.EUROPE:
       cell.imageView?.image = UIImage(named: "blau")
    case Continent.ANTARTKIS:
       cell.imageView?.image = UIImage(named: "antarktis")
    default:
       cell.imageView?.image = UIImage(named: "default")
  }
  return cell
}
```

Neue Methoden der TableView für das Editieren:

```
   // diese Methode wird von NoYes-Dialog aufgerufen
  func deleteYes(){
    print("delete Yes")
    if let indexPath = tableview.indexPathForSelectedRow {
       let city:City = cities[indexPath.row]
      cities.remove(at: indexPath.row)
      tableview.reloadData()
     }
  }
```

```
   // diese Methode wird von NoYes-Dialog aufgerufen
  func deleteNo(){
    print("delete No")
  }

  @IBAction func bndeleteclick(_ sender: UIButton) {
    if let indexPath = tableview.indexPathForSelectedRow {
       let city:City = cities[indexPath.row]
      let mess:String = "Wollen Sie wirklich diesen Eintrag
löschen?\n\nCity:"+city.name+"\nKontinent:
"+getContinentText(city.continent)+"\nBem: "+city.remark

      Basis.showNoYesDialog(parent: self, _title: "Löschen eines Eintrags",
_message: mess, _yesFunc: deleteYes, _noFunc: deleteNo)
     }
  }


   // diese Methode wird vor dem Sprung zum InputController aufgerufen
  override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
      if let button:UIButton = sender as? UIButton {
         // Abfrage, welcher Schalter: Übergabe der Stadt
        if button==bnedit {
          if let indexPath = tableview.indexPathForSelectedRow {
           let city:City = cities[indexPath.row]
           if let dest = segue.destination as? InputViewController {
              dest.city=city
              dest.modus =   ChangeMode.EDIT
              labelInfo.text=city.name
           }
         }
       } // edit
      else if button==bnnew {
        if let dest = segue.destination as? InputViewController {
           dest.modus =   ChangeMode.NEW
           labelInfo.text="New"
        }
      } // new Stadt
     }  // if
    }  // prepare


  // Rücksprung vom ChangeViewController Abbruch
     @IBAction func unwindCancel(_ segue:UIStoryboardSegue) {
    // keine Aktion notwendig
    }  // unwindToEditViewCancel

  // Rücksprung vom ChangeViewController Save
   @IBAction func unwindOk(_ segue:UIStoryboardSegue) {
    if let src = segue.source as? InputViewController {
      print("in unwindOK")
      let city = src.city
      print(city.name)
      print(city.remark)
      print(city.continent)
      switch (src.modus) {
      case .EDIT:
        //
```

```
        print("edit")
        if let indexPath = tableview.indexPathForSelectedRow {
           let citytable:City = cities[indexPath.row]
          citytable.initCity(city)
          tableview.reloadRows(at: [indexPath], with:
UITableView.RowAnimation.right)
         }
      case .NEW:
        print("new")
        let citytable = City( Continent.EUROPE ,"",  "",  Visited.NONE)
        citytable.initCity(city)
        cities.append(citytable)
        tableview.reloadData()

         //
      } // switch (src.modus) {

  }  // unwindOk

  }


}
```

Referenzen und Variablen:

```
  // Referenzen
@IBOutlet var labelInfo: UILabel!

@IBOutlet var bnedit: UIButton!
@IBOutlet var bndelete: UIButton!
@IBOutlet var bnnew: UIButton!

@IBOutlet var tableview: UITableView!

var cities : [City] = []    // leeres Array, Liste
```

viewDidLoad:

```
override func viewDidLoad() {
   super.viewDidLoad()
   cities = CitiesDB.loadCitiesIntern()

   tableview.delegate=self
   tableview.dataSource=self

   labelInfo.text="-"
   bnedit.isEnabled=false
   bndelete.isEnabled=false
}
```

12.4.7 Löschen eines Eintrags

```
func deleteYes(){
   print("delete Yes")
   if let indexPath = tableview.indexPathForSelectedRow {
      cities.remove(at: indexPath.row)
      tableview.reloadData()
   }
}

func deleteNo(){
  print("delete No")
}

@IBAction func bndeleteclick(_ sender: Any) {
   if let indexPath = tableview.indexPathForSelectedRow {
      let city:City = cities[indexPath.row]
      let mess:String = "Wollen Sie wirklich diesen Eintrag löschen?\n\n" +
        "City:"+city.name+"\nKontinent: "+getContinentText(city.continent)+
        "\nBem: "+city.remark
      Basis.showNoYesDialog(parent: self, _title: "Löschen eines Eintrags",
         _message: mess, _yesFunc: deleteYes, _noFunc: deleteNo)
  }
}
```

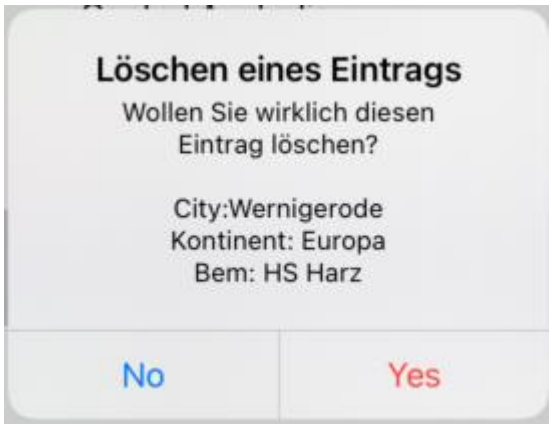


Abbildung 254 Abfrage zum Löschen eines Eintrags

12.4.8 Edit-ViewController

Nun einen zweiten ViewController aus dem Lib-Controller einfügen

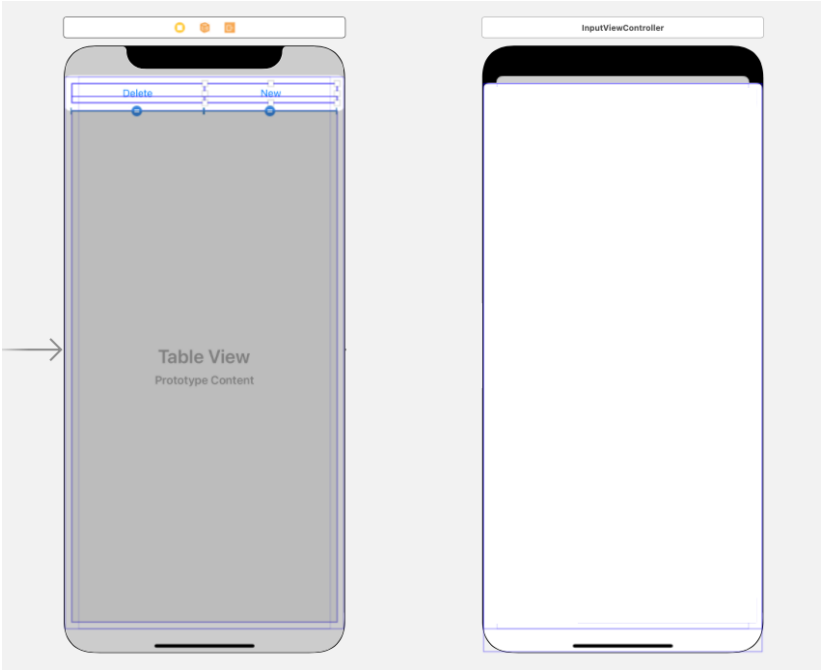
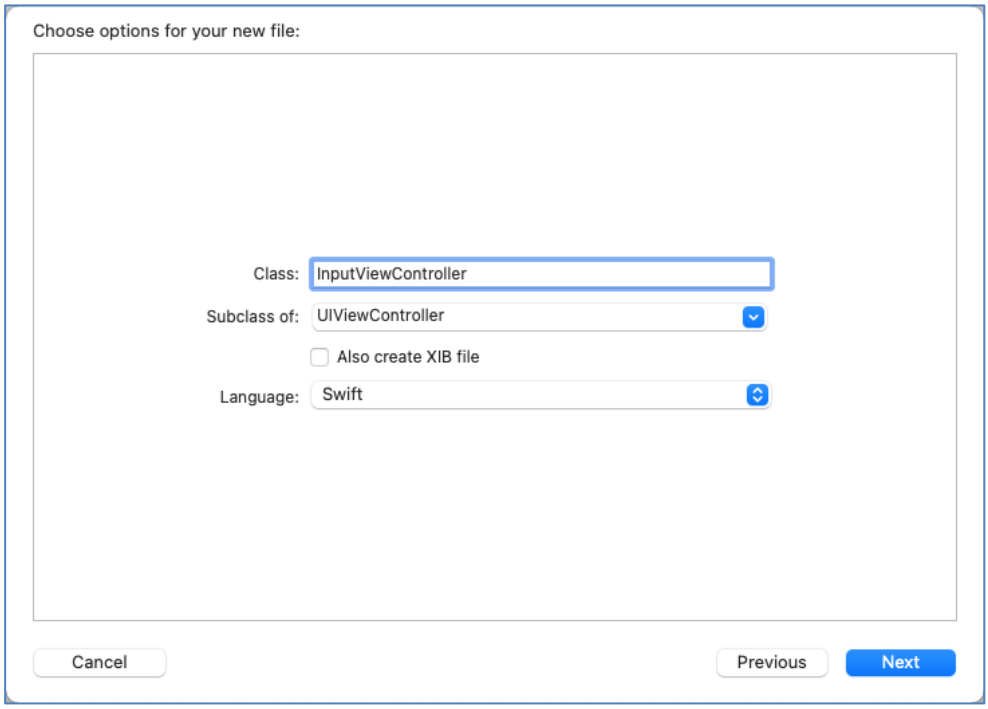


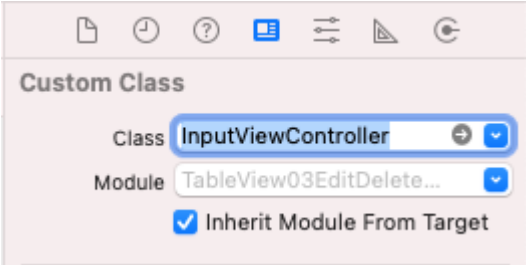
Abbildung 255 Anzeige beider ViewController

12.4.9 Swift-Klasse für den 2. View

Anlegen der Swift-Klasse für den Edit-ViewController mit Cmd+N



Nun den zweiten ViewController aktivieren, drei Schalter, ganz links und rechts obne im Property-Fenster die Klasse auswählen:



12.4.10 Schalter Edit

Ablauf:

- Aktivieren des Schalter „Edit“. Entweder in der Grafik oder im linken Property-Fenster.
- Ctrl-Taste drücken
- Ziehen der Maus zum zweiten ViewController
- Loslassen
- Auswahl „Show“

Abbildung 256 Link von Edit-Schalter zum 2. ViewController

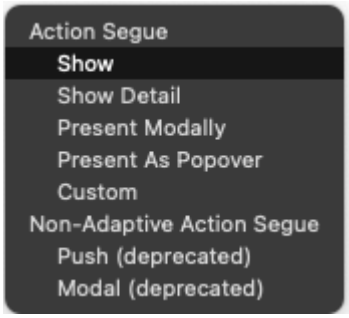


Abbildung 257 Show auswählen

Show (Push)

This segue displays the new content using the showViewController:sender: method of the target view controller. For most view controllers, this segue presents the new content modally over the source view controller. Some view controllers specifically override the method and use it to implement different behaviors. For example, a navigation controller pushes the new view controller onto its navigation stack. UIKit uses the targetViewControllerForAction:sender: method to locate the source view controller.

Show Detail (Replace)
This segue displays the new content using the showDetailViewController:sender: method of the target view controller. This segue is relevant only for view controllers embedded inside a UISplitViewController object. With this segue, a split view controller replaces its second child view controller (the detail controller) with the new content. Most other view controllers present the new content modally.

UIKit uses the targetViewControllerForAction:sender: method to locate the source view controller.

Present Modally
This segue displays the view controller modally using the specified presentation and transition styles. The view controller that defines the appropriate presentation context handles the actual presentation.

Present as Popover
In a horizontally regular environment, the view controller appears in a popover. In a horizontally compact environment, the view controller is displayed using a full-screen modal presentation.

Quelle:
https://developer.apple.com/library/archive/featuredarticles/ViewControllerPGforiPhoneOS/UsingSegues.html

12.4.11 Schalter New

Ablauf:

- Aktivieren des Schalter „New“. Entweder in der Grafik oder im linken Property-Fenster.
- Ctrl-Taste drücken
- Ziehen der Maus zum zweiten ViewController
- Loslassen
- Auswahl „Show“

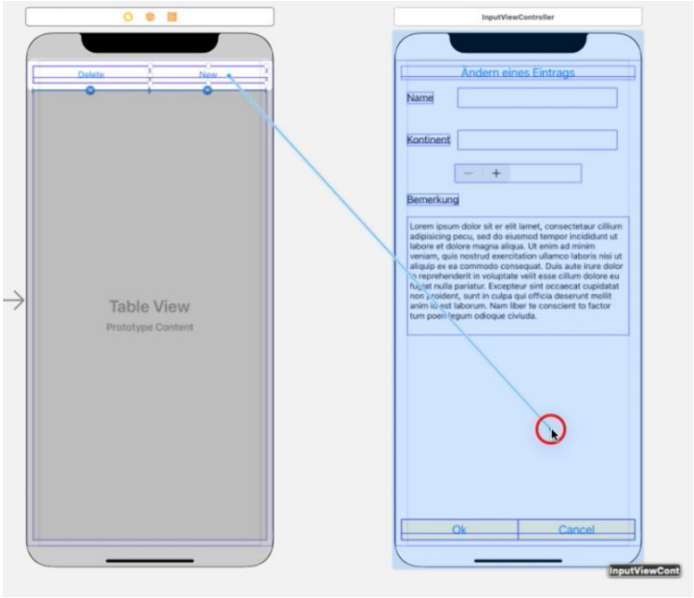


Abbildung 258 Link von New-Schalter zum 2. ViewController

12.4.12 UI-Aufbau des zweiten Controllers

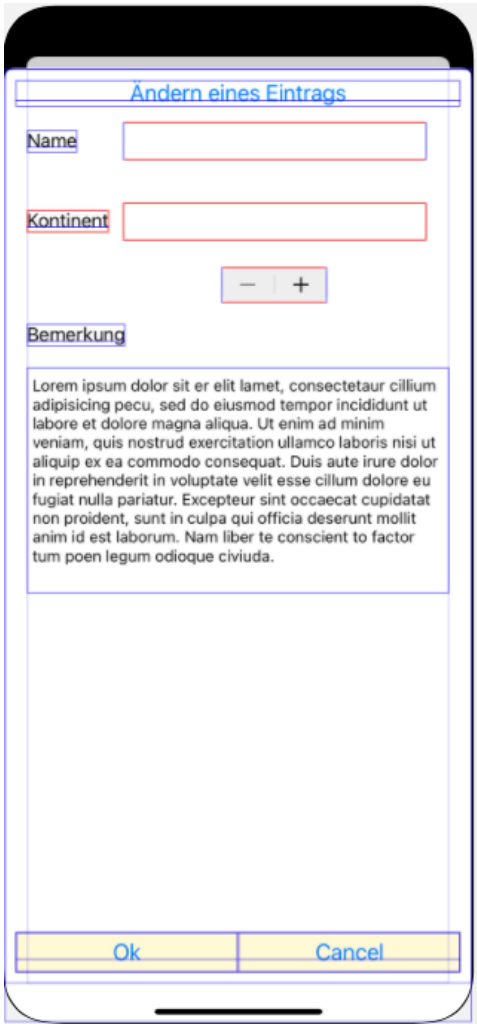


Abbildung 259 UI-Aufbau des zweiten ViewControllers

LabelCaption

- Top 10
- Left: 10
- Right: 10

Label Name:

- Top 20
- Left: 20

Textfield Name

- Top 13
- Left: 40
- Right: 40

Label Kontinent:

- Top: 50
- Left: 20

Textfield Kontinent:

- Top 37
- Left: 12
- Right: 40

Stepper Kontinent:

- Top 23
- Left: 100
- Right: 100

Label Bemerkung:

- Top: 18 (zum Label Kontinent)
- Left: 20

TextView Bemerkung:

- Top 18
- Left: 20
- Right: 20
- **Feste Höhe: 200**

Horizontale Box:

- Left: 10
- Right: 10
- Bottom 10

Schalter ok und Schalter cancel:

- **Gleiche Breite**

Referenzen erstellen:

- @IBOutlet var labelcaption: UILabel!
- @IBOutlet var tname: UITextField!
- @IBOutlet var tcontinent: UITextField!
- @IBOutlet var tremark: UITextView!
- @IBOutlet var steppercontinent: UIStepper!
- @IBOutlet var bnok: UIButton!
- @IBOutlet var bncancel: UIButton!

Event erstellen (nicht changedEnded nehmen):

- @IBAction func **tnamechanged**(_ sender: UITextField) {
- @IBAction func **stepperkontinentchanged**(_ sender: UIStepper) {

12.4.13 Code für den InputViewController

Variablen:

```
  // in dieser Variable findet die Parameterübergabe statt!
var city:City=City( Continent.AMERICA ,  "---",  "--------", Visited.NONE)
var modus:ChangeMode = ChangeMode.EDIT
```

Referenzen:

```
 @IBOutlet var labelcaption: UILabel!
 @IBOutlet var tname: UITextField!
 @IBOutlet var tcontinent: UITextField!
 @IBOutlet var tremark: UITextView!
 @IBOutlet var steppercontinent: UIStepper!
 @IBOutlet var bnok: UIButton!
 @IBOutlet var bncancel: UIButton!
```

Delegate für die TextView Änderung:

```
class InputViewController: UIViewController,  UITextViewDelegate{
  // es gibt leider kein OnChanged-Event
```

viewDidLoad:

```
override func viewDidLoad() {
    super.viewDidLoad()

    if modus==ChangeMode.EDIT {
        labelcaption.text = "Ändern von: "+city.name
        bnok.setTitle("Änderung speichern", for: UIControl.State.normal)
    }
    else {
        labelcaption.text = "Neuer Eintrag"
        bnok.setTitle("Eintrag einfügen", for: UIControl.State.normal)
    }
     // tcontinent wird über dem Stepüper gesetzt
    tcontinent.isEnabled = false
     // erstz aktiv bei einer Änderung
    bnok.isEnabled = false

    tremark.delegate=self
        // werte eintragen
    tname.text = city.name
    tremark.text = city.remark
    let conti:String = continente[city.kontinent.rawValue]
    tcontinent.text = conti
    steppercontinent.minimumValue=0
    steppercontinent.maximumValue=Double(continente.count-1)
    steppercontinent.value = Double(city.kontinent.rawValue)
    steppercontinent.stepValue=1
}
```

onChanged-Events:

```
@IBAction func tnamechanged(_ sender: UITextField) {
   bnok.isEnabled = true
}

@IBAction func stepperkontinentchanged(_ sender: UIStepper) {
    bnok.isEnabled = true
    let index:Int = Int(steppercontinent.value)
```

```
    let conti:String = continente[index]
    tcontinent.text = conti
}

func textViewDidChange(_ textView: UITextView) {
    //print(tremark.text!)
    bnok.isEnabled = true
}
```

12.4.14 Rücksprung

Im ersten ViewController wurden zwei „unwind“-Methoden eingebaut:

- unwindCancel
- unwindOk

Diese beiden Methoden werden durch die Schalter im InputViewController nach dem Beenden aufgerufen. Dazu muss man erstmal die prepare-Methode implementieren. Diese wird **vor** Rücksprung aufgerufen. Man benutzt die Methode, um die Daten zurückzuspeichern.

Prepare-Methode:

```
override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
    city.name=tname.text!
    city.remark=tremark.text!
    city.kontinent = getContinent(Int(steppercontinent.value))
}
```

Nun folgen noch die Links von den Schalter zum Exit-Schalter. Der Exit-Schalter ist der rechte der oberen drei Schalter. Man klickt den Ok-Schalter an, drückt die Ctrl-Taste und zieht die Linie zum rechten Schalter. Nach dem Loslassen der Maus wählt man die entsprechende Rücksprung-Methode aus.

12.4.15 Komplette Quellcodes

Hier noch einmal der komplette Quellcode des ersten ViewController

//

```
//
//  ViewController.swift
```

```
//  TableView03EditDeleteNew
//
//  Created by User1 on 05.09.21.
import UIKit

class ViewController: UIViewController, UITableViewDelegate,
UITableViewDataSource{
  @IBOutlet var labelInfo: UILabel!
  @IBOutlet var bnedit: UIButton!

  @IBOutlet var bndelete: UIButton!
  @IBOutlet var bnnew: UIButton!
  @IBOutlet var tableview: UITableView!

  var cities : [City] = []

  override func viewDidLoad() {
    super.viewDidLoad()
    cities = CitiesDB.loadCitiesIntern()

    tableview.delegate=self
    tableview.dataSource=self

    labelInfo.text="111"
    bnedit.isEnabled=false
    bndelete.isEnabled=false

  }


  func numberOfSections(in tableView: UITableView) -> Int {
     return 1
  }

  func tableView(_ tableView: UITableView, numberOfRowsInSection section:
Int) -> Int {
    return cities.count
  }



    // SelectecIndexChanged
   func tableView(_: UITableView, didSelectRowAt: IndexPath) {
     let row = (didSelectRowAt as NSIndexPath).row
    let city:City = cities[row]
    //label1.text="row: "+String(row)
    labelInfo.text="City: "+city.name
    bnedit.isEnabled=true
    bndelete.isEnabled=true
   }
  ^
```

```
  func tableView(_ tableView: UITableView, cellForRowAt indexPath:
IndexPath) -> UITableViewCell {

    let cell: UITableViewCell = UITableViewCell(style:
UITableViewCell.CellStyle.subtitle
                          , reuseIdentifier: "reuseIdentifier")

    // Configure the cell...
    let row = (indexPath as NSIndexPath).row
    let city:City = cities[ row ]
    cell.textLabel?.text = city.name
    cell.detailTextLabel?.text = stadt.remark

      switch city.kontinent {
      case Continent.AFRICA:
        cell.imageView?.image = UIImage(named: "schwarz")
      case Continent.AMERICA:
        cell.imageView?.image = UIImage(named: "rot")
      case Continent.ASIA:
        cell.imageView?.image = UIImage(named: "gelb")
      case Continent.AUSTRALIA:
        cell.imageView?.image = UIImage(named: "gruen")
      case Continent.EUROPE:
        cell.imageView?.image = UIImage(named: "blau")
      case Continent.ANTARTKIS:
        cell.imageView?.image = UIImage(named: "antarktis")

      default:
        cell.imageView?.image = UIImage(named: "default")
      }
    return cell
  }


  func deleteYes(){
    print("delete Yes")
    if let indexPath = tableview.indexPathForSelectedRow {
       let city:City = cities[indexPath.row]
      cities.remove(at: indexPath.row)
      tableview.reloadData()
     }
  }
  func deleteNo(){
    print("delete No")
  }

  @IBAction func bndeleteclick(_ sender: UIButton) {
    if let indexPath = tableview.indexPathForSelectedRow {
       let city:City = cities[indexPath.row]
      let mess:String = "Wollen Sie wirklich diesen Eintrag
löschen?\n\nCity:"+city.name+"\nKontinent:
"+getContinentText(city.kontinent)+"\nBem: "+city.remark
      Basis.showNoYesDialog(parent: self, _title: "Löschen eines Eintrags",
_message: mess, _yesFunc: deleteYes, _noFunc: deleteNo)
     }
  }
```

```
  override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
      if let button:UIButton = sender as? UIButton {
         // Abfrage, welcher Schalter: Übergabe der Stadt
        if button==bnedit {
          if let indexPath = tableview.indexPathForSelectedRow {
           let city:City = cities[indexPath.row]
           if let dest = segue.destination as? InputViewController {
              dest.city=city
              dest.modus =   ChangeMode.EDIT
              labelInfo.text=city.name
           }
         }
       } // edit
      else if button==bnnew {
        if let dest = segue.destination as? InputViewController {
           dest.modus =   ChangeMode.NEW
           labelInfo.text="New"
        }
      } // new Stadt
     }  // if
    }  // prepare


  // Rücksprung vom ChangeViewController Abbruch
     @IBAction func unwindCancel(_ segue:UIStoryboardSegue) {
    // keine Aktion notwendig
    }  // unwindToEditViewCancel

  // Rücksprung vom ChangeViewController Save
   @IBAction func unwindOk(_ segue:UIStoryboardSegue) {
    if let src = segue.source as? InputViewController {
      let city = src.city
      switch (src.modus) {
      case .EDIT:
        if let indexPath = tableview.indexPathForSelectedRow {
           let citytable:City = cities[indexPath.row]
          citytable.initCity(city)
          tableview.reloadRows(at: [indexPath], with:
UITableView.RowAnimation.right)
         }
      case .NEW:
        print("new")
        let citytable = City( Continent.EUROPE ,"",  "",  Visited.NONE)
        citytable.initCity(city)
        cities.append(citytable)
        tableview.reloadData()
      } // switch (src.modus) {
   }  // if
 } // unwindOk


}
```

Hier der Quellcode des InputViewControllers

```
import UIKit
```

```
class InputViewController: UIViewController,  UITextViewDelegate{

  var city:City=City( Continent.AMERICA ,  "---",  "--", Visited.NONE)
  var modus:ChangeMode = ChangeMode.EDIT


  @IBOutlet var labelcaption: UILabel!
  @IBOutlet var tname: UITextField!
  @IBOutlet var tcontinent: UITextField!
  @IBOutlet var tremark: UITextView!
  @IBOutlet var steppercontinent: UIStepper!
  @IBOutlet var bnok: UIButton!
  @IBOutlet var bncancel: UIButton!


  override func viewDidLoad() {
    super.viewDidLoad()

    if modus==ChangeMode.EDIT {
      labelcaption.text = "Ändern von: "+city.name
      bnok.setTitle("Änderung speichern", for: UIControl.State.normal)
    }
    else {
      labelcaption.text = "Neuer Eintrag"
      bnok.setTitle("Eintrag einfügen", for: UIControl.State.normal)
    }

    tcontinent.isEnabled = false
    bnok.isEnabled = false

    tremark.delegate=self
    // werte eintragen
    tname.text = city.name
    tremark.text = city.remark
    let conti:String = continente[city.kontinent.rawValue]
    tcontinent.text = conti
    steppercontinent.minimumValue=0
    steppercontinent.maximumValue=Double(continente.count-1)
    steppercontinent.value = Double(city.kontinent.rawValue)
    steppercontinent.stepValue=1

  }


  @IBAction func tnamechanged(_ sender: UITextField) {
    bnok.isEnabled = true
  }

  @IBAction func stepperkontinentchanged(_ sender: UIStepper) {
    bnok.isEnabled = true
    let index:Int = Int(steppercontinent.value)
    let conti:String = continente[index]
    tcontinent.text = conti
  }

  func textViewDidChange(_ textView: UITextView) {
    //print(tremark.text!)
    bnok.isEnabled = true
```

```
  }



  override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
    // Get the new view controller using segue.destination.
    // Pass the selected object to the new view controller.

    city.name=tname.text!
    city.remark=tremark.text!
    city.kontinent = getContinent(Int(steppercontinent.value))

  }


}
```

12.4.16 Laden und Speichern der Daten per UserDefaults

Damit die Städteliste persitent, dauerhaft gespeichert ist, muss man noch einige Änderungen am Quellcode vornehmen. Mit der UserDefault-Technik speichert man die Daten in eine Ini-Datei, ähnlich der Windows-Registry. Es ist keine Datenbank, aber der Aufwand ist wesentlich geringer und für kleine Daten vollkommen ausreichend.

Klasse City:

```
public func saveDefault(_ defaults:UserDefaults, _ i:Int) {
   defaults.set(continent.rawValue,forKey:"kontinent"+String(i))
   defaults.set(name,forKey:"name"+String(i))
   defaults.set(remark,forKey:"remark"+String(i))
   defaults.set(visited.rawValue,forKey:"visited"+String(i))
}  // saveDefault

public func loadDefault(_ defaults:UserDefaults, _ i:Int) {
   continent = Continent(rawValue:
defaults.integer(forKey:"kontinent"+String(i))) ?? Continent.AFRICA
   name = defaults.string(forKey:"name"+String(i))!
   remark = defaults.string(forKey:"remark"+String(i))!
   visited = Visited(rawValue:
defaults.integer(forKey:"visited"+String(i))) ?? Visited.NONE
}  // loadDefault
```

Klasse CitiesDB:

```
// dies ist die Standardmethode
// wenn noch keine Daten gespeichert sind, werden die Dummydaten genommen
public static func loadCities() -> Array<City> {
   var staedte=[City]()
   let defaults = UserDefaults.standard
   let n = defaults.integer(forKey:"staedteCount")

   if n>0 {
     staedte = loadCitiesDBIntern()
   }
```

```
    else {
      staedte = loadCitiesIntern()
   }
   return staedte
}  // loadStaedteDBIntern

 public static func loadCitiesDBIntern()  -> Array<City> {
    var staedte=[City]()
    let defaults = UserDefaults.standard
    let n = defaults.integer(forKey:"staedteCount")
    if n>0 {
       for i in 0...n-1 {
          let City = City(Continent.AFRICA ,"", "", Visited.NONE)
          City.loadDefault(defaults, i)
          staedte.append(City)
       } // for
    } // if
    return staedte
}  // loadStaedteDBIntern

public static func saveStaedte(staedte: Array<City>) {
    let defaults = UserDefaults.standard
    defaults.set(staedte.count,forKey:"staedteCount")
    var i:Int=0
    for City in staedte {
       City.saveDefault(defaults, i)
       i+=1
    }
}  // saveStaedte
```

Klasse ViewController (Änderung in rot):

```
override func viewDidLoad() {
   super.viewDidLoad()
   cities = CitiesDB.loadCities()
   tableview.delegate=self
   tableview.dataSource=self
   bnedit.isEnabled=false
   bndelete.isEnabled=false
}

// es gibt noch einen 4. Schalter „init", der die Liste wieder zurücksetzt
@IBAction func bninitclick(_ sender: UIButton) {
   cities = CitiesDB.loadCitiesIntern()
   CitiesDB.saveStaedte(staedte: cities)
   tableview.reloadData()
}

func deleteYes(){
   if let indexPath = tableview.indexPathForSelectedRow {
      cities.remove(at: indexPath.row)
      tableview.reloadData()
      CitiesDB.saveStaedte(staedte: cities)
   }
}  // deleteYes
   // Rücksprung vom ChangeViewController Save
@IBAction func unwindOk(_ segue:UIStoryboardSegue) {
   if let src = segue.source as? InputViewController {
```

```
      let city = src.city
      switch (src.modus) {
        case .EDIT:
           if let indexPath = tableview.indexPathForSelectedRow {
              let citytable:City = cities[indexPath.row]
              citytable.initCity(city)
              tableview.reloadRows(at: [indexPath],
                                   with: UITableView.RowAnimation.right)
              CitiesDB.saveStaedte(staedte: cities)
           }
        case .NEW:
             let citytable = City(Continent.EUROPE ,"",  "",  Visited.NONE)
             citytable.initCity(city)
             cities.append(citytable)
             tableview.reloadData()
             CitiesDB.saveStaedte(staedte: cities)
     } // switch (src.modus) {
}  // unwindOk
```

12.5 TableView mit DoubleClick (TableView04)

Dieses Kapitel beschreibt, wie man die Einträge in einer Tabelle bearbeiten, löschen oder neue einfügen kann. Der Aufbau ist grundsätzlich mit dem dritten Beispiel identisch. Hier wird aber mit einem Doppelklick für das Editieren gearbeitet.

Vorschau

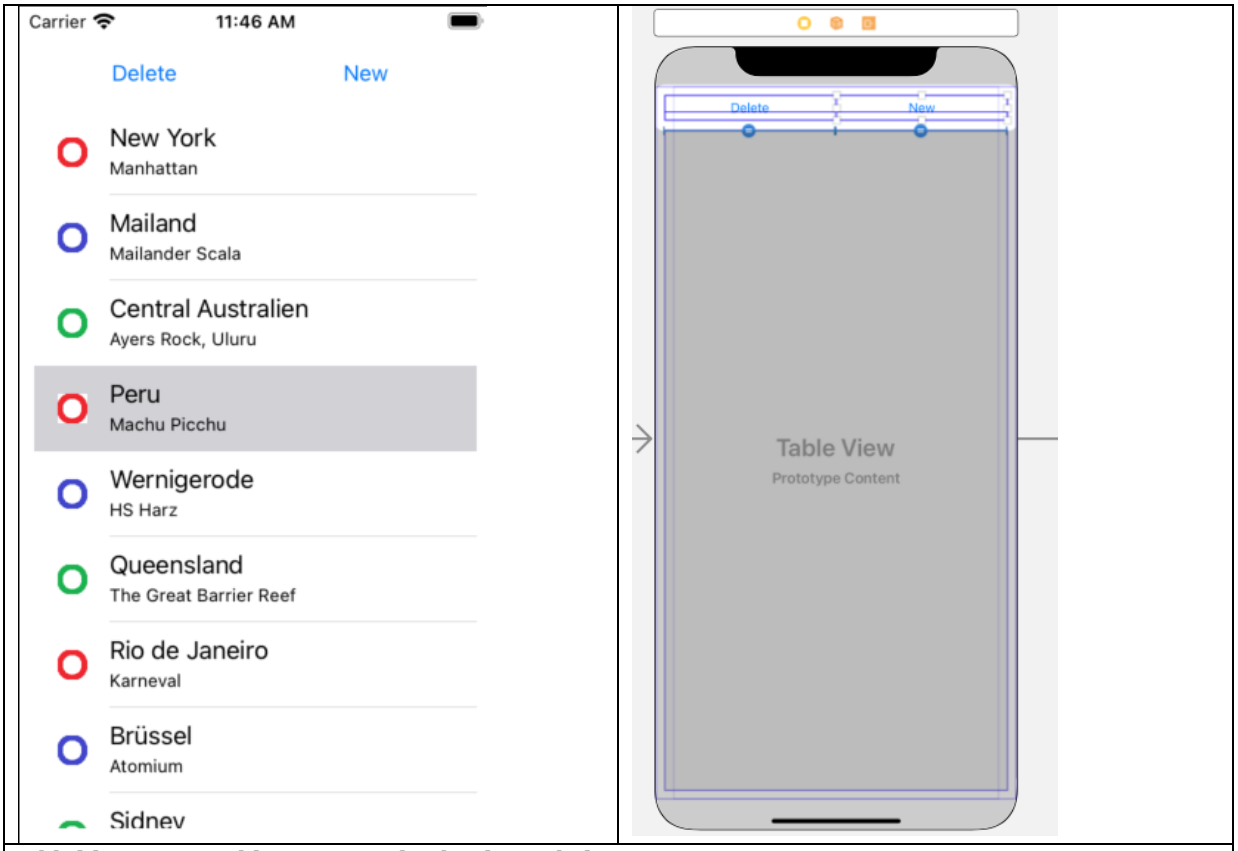


Abbildung 260 TableView mit den beiden Schaltern

12.5.1 Anlegen eines Projektes

Legen Sie ein normales Projekt „App“ an und speichern Sie dieses.
Der Name dieses Projektes ist „TableView04DoubleClick“.

12.5.2 Schalterleiste einfügen

Fügen Sie einen horizontalen Stackview oben in den ViewController ein. Als Constraint geben Sie ein:

- Top 0
- Left: 0
- Right: 0

- Bottom: 0

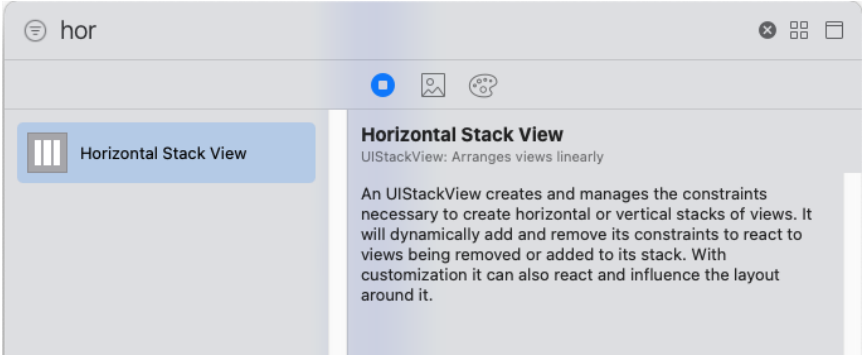


Abbildung 261 Horizontal Stack

Fügen Sie ein Label und drei Buttons in den Stack. Das Label dient als Anzeige der aktuellen Zeile, die Schalter dienen zum Bearbeiten:

- Delete hier wird ein NoYes-Dialog benutzt
- New hier wird zu einem zweiten View umgeschaltet

Prinzipiell könnte man auch drei Schalter einbauen, also eine „Edit"-Schalter wie im dritten Beispiel. Dann kann der Anwender den Schalter zum Ändern benutzen oder er „klickt" doppelt in einem Eintrag.

12.5.3 TableView-Controller einfügen

Fügen Sie nun aus dem Lib-Dialog eine TableView in den ViewController. Als Constraint geben Sie ein:

- Top 10
- Left: 10
- Right: 10
- Bottom: 10

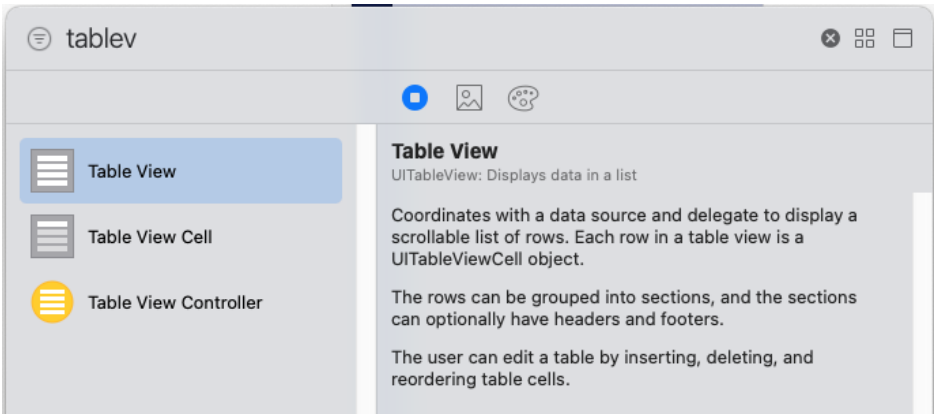


Abbildung 262 TableView einfügen

Das Projekt müsste dann so aussehen:

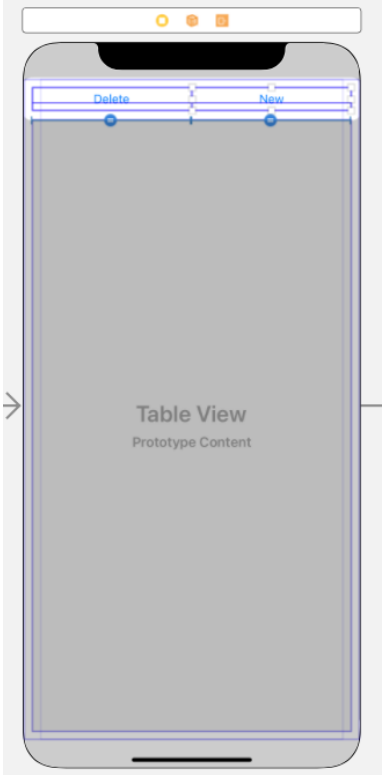


Abbildung 263 Aktueller Stand mit dem StackView und der TableView

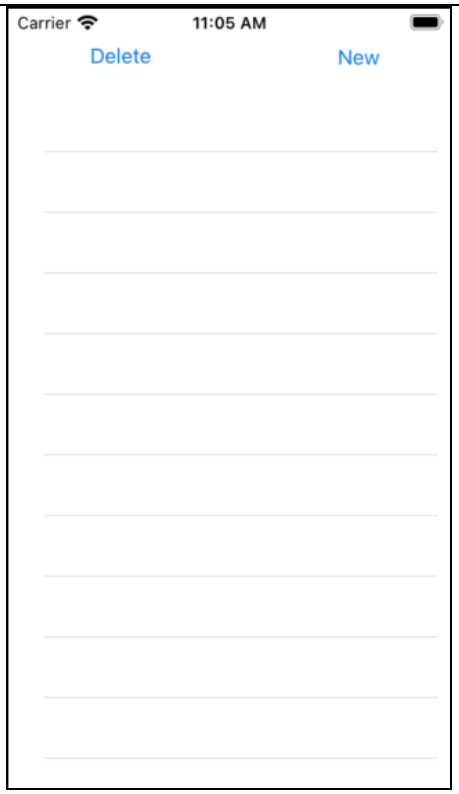


Abbildung 264 Leere TableView

Im letzten Schritt für die UI erstellt man die Referenzen der UITableView, abels und eine EventMethode für den Schalter.

12.5.4 Hilfsklassen für die TableView

In der Tabelle sollen Städte angezeigt werden. Dazu brauchen wir eine Klasse „City“ und eine Verwaltungsklasse „CitiesDB“.

Anlegen einer neuen Klasse „City.swift“

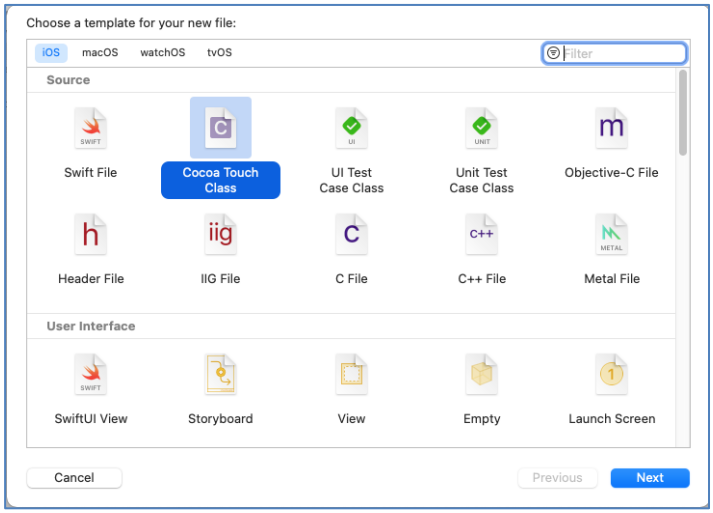


Abbildung 265 Einfüge-Dialog für die Klasse City.swift (Cmd+N)

Nun den Namen und den Typ eintragen:

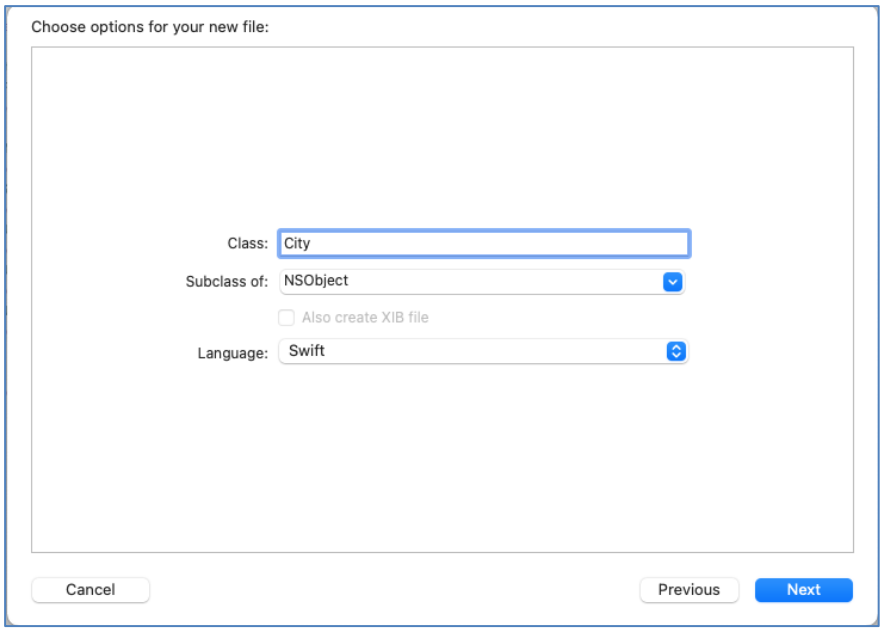


Abbildung 266 Name und Typ der Klasse City

Nun noch eine zweite Klasse „Enums.swift“ und eine dritte Klasse „CitiesDB.swift“ einfügen.
In der Klasse „Enums.swift“ werden die Aufzählungen, Enumeration eingetragen:

```
import UIKit


enum ChangeMode:Int {
     case EDIT
    case NEW
  //  case DELETE   mit yes-no Dialo
}
```

```
enum Continent:Int {         // Deklaration
  case  AFRICA=0, AMERICA, ASIA, AUSTRALIA, EUROPE, ANTARTKIS
}

let continente:[String] =
     ["Afrika","Amerika", "Asien", "Australien", "Europa", "Antartkis"]


func getContinentText(_ continent:Continent) -> String {
    switch (continent) {
      case Continent.AFRICA:
        return "Afrika"
      case Continent.AMERICA:
        return "Amerika"
      case Continent.ASIA:
        return "Asien"
      case Continent.AUSTRALIA:
        return "Australien"
      case Continent.EUROPE:
        return "Europa"
      case Continent.ANTARTKIS:
        return "Antartkis"
      default:
        return "unbekannt"
   }
}

func getContinent(_ continent:Int) -> Continent {
    switch (continent) {
       case 0:
         return Continent.AFRICA
       case 1:
         return Continent.AMERICA
       case 2:
         return  Continent.ASIA
       case 3:
         return Continent.AUSTRALIA
       case 4:
         return Continent.EUROPE
       case 5:
         return Continent.EUROPE
       default:
         return Continent.EUROPE
   }
}

enum Visited : Int{         // Deklaration
  case  NONE=0, DOWN, AVERAGE, UP
}
```

Quellcode für Enums.swift

Eine Enumeration ist im Quellcode wesentlich sicherer als ein int-Wert mit Konstanten. Das wäre ein sehr schlechter Stil.

```
import UIKit

class City: NSObject {
      // Attribute
   var continent: Continent=Continent.AFRICA
   var name : String = ""
   var remark : String = ""
   var visited:Visited = Visited.NONE

     // Konstruktor
   init(_ continent: Continent,_ name:String, _ remark:String, _ visited:Visited){
      self.continent = continent
      self.name = name
      self. remark = remark
      self. visited = visited
   }  // init

}
```

Quellcode für City.swift

```
import UIKit

class CitiesDB: NSObject {

   public static func loadCitiesIntern()  -> Array<City> {
     var cities=[City]()
     cities.append( City( Continent.AMERICA ,  "New York",  "Manhattan", Visited.NONE)  )
     cities.append( City( Continent.EUROPE , "Mailand", "Mailander Scala", Visited.AVERAGE))
     cities.append( City( Continent.AUSTRALIA ,"Central Australien","Ayers Rock, Uluru",
Visited.NONE)  )
     cities.append( City( Continent.AMERICA ,"Peru",  "Machu Picchu",  Visited.UP)  )
     cities.append( City( Continent.EUROPE ,"Wernigerode",  "HS Harz",  Visited.DOWN)  )
     cities.append( City( Continent.AUSTRALIA ,"Queensland",  "The Great Barrier Reef",
Visited.NONE)  )
     cities.append( City( Continent.AMERICA ,"Rio de Janeiro", "Karneval",  Visited.DOWN)  )
     cities.append( City( Continent.EUROPE ,"Brüssel",   "Atomium",  Visited.UP)  )
     cities.append( City( Continent.AUSTRALIA ,"Sidney",  "Opernhaus",  Visited.UP)  )
     cities.append( City( Continent.AFRICA ,"Kairo",  "Pyramiden",  Visited.NONE)  )
     cities.append( City( Continent.AMERICA ,"Chile",  "Atacama Wüste",  Visited.UP)  )
     cities.append( City( Continent.ASIA ,"Tokio",  "Shinjuku Gyoen National Garden",
Visited.DOWN)  )
     cities.append( City( Continent.EUROPE ,"Kiel",  "Kieler Woche",  Visited.UP)  )
     cities.append( City( Continent.AFRICA ,"Hunsbergen","Apollo 11 Höhle",  Visited.NONE) )
     cities.append( City( Continent.EUROPE ,"Dresden",  "Das grüne Zimmer",  Visited.NONE) )
     cities.append( City( Continent.AFRICA ,"Marrakesch",  "Botanische Garten Jardin
Majorelle",  Visited.DOWN)  )
     cities.append( City( Continent.AMERICA ,"San Fransisco",  "Golden Gate Bridge",
Visited.UP)  )
     cities.append( City( Continent.EUROPE ,"Halberstadt",  "John Cage",  Visited.NONE)  )
     cities.append( City( Continent.ASIA ,"Peking",  "Verbotene Stadt",  Visited.NONE)  )
     cities.append( City( Continent.EUROPE ,"Venezuela",  "Angel Falls",  Visited.UP)  )
     cities.append( City( Continent.AMERICA ,"Bilbao",  "Guggenheim-Museum",  Visited.UP) )
     cities.append( City( Continent.ASIA ,"Chiang Mai",  "Der weiße Tempel",  Visited.UP) )
     cities.append( City( Continent.AMERICA ,"Rio de Janeiro",  "Zuckerhut",  Visited.NONE))
     return cities
}
```

Quellcode für CitiesDB.swift

12.5.5 Symbole für die TableView

Einfügen der Symbole aus der Datei „mycellsymbols.zip“ in den Projektordner „assert“ per Drag&Drop“. **Bitte beachten Sie, dass Sie immer das Bild ohne Nummer ins Projekt ziehen**.

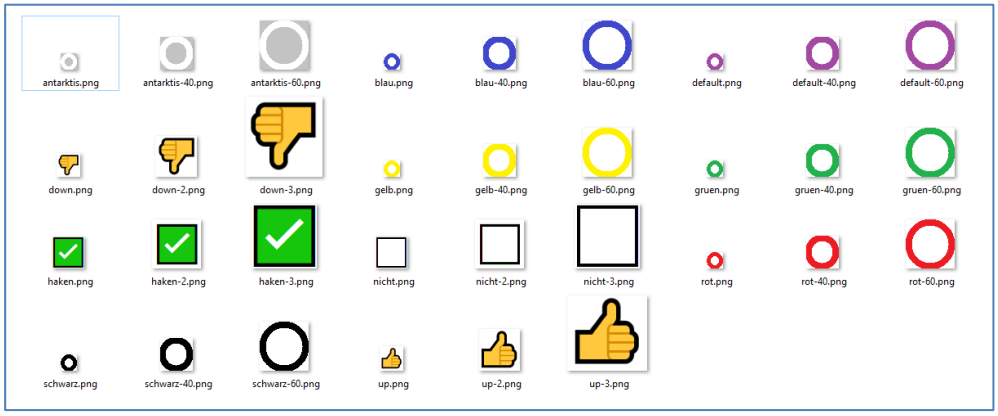


Abbildung 267 Symbole für die TableView

12.5.6 TableView mit einer Liste füllen

Da der ViewController kein TableViewController ist, müssen die Delegate manuell eingetragen werden. Danach muss also nur die Liste der Städte holen, die UITableView mit den Methoden verknüpfen (Interface) und die Anzahl der Städte und den Aufbau der „Zelle“ eintragen.

Fügen Sie oben in der Klassendefinition folgende Protokolle, Delegate, ein:
class ViewController: UIViewController , **UITableViewDelegate, UITableViewDataSource**

Methoden der TableView:

```
func numberOfSections(in tableView: UITableView) -> Int {
   return 1
}

func tableView(_ tableView: UITableView,numberOfRowsInSection section: Int) -> Int{
   return cities.count
}


      // SelectecIndexChanged
func tableView(_: UITableView, didSelectRowAt: IndexPath) {
   let row = (didSelectRowAt as NSIndexPath).row
   let city:City = cities[row]
   labelInfo.text="City: "+city.name
    // nun die beiden Schalter verfügbar machen
   bnedit.isEnabled=true
   bndelete.isEnabled=true
}
```

```
   // Zelle aufbauen
func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath)
-> UITableViewCell {

  let cell: UITableViewCell = UITableViewCell(
       style: UITableViewCell.CellStyle.subtitle,
       reuseIdentifier: "reuseIdentifier")


        // Configure the cell...
  let row = (indexPath as NSIndexPath).row
  let city:City = cities[ row ]
  cell.textLabel?.text = city.name
  cell.detailTextLabel?.text = city.remark

     // Image setzen
  switch city.continent {
    case Continent.AFRICA:
       cell.imageView?.image = UIImage(named: "schwarz")
    case Continent.AMERICA:
       cell.imageView?.image = UIImage(named: "rot")
    case Continent.ASIA:
       cell.imageView?.image = UIImage(named: "gelb")
    case Continent.AUSTRALIA:
       cell.imageView?.image = UIImage(named: "gruen")
    case Continent.EUROPE:
       cell.imageView?.image = UIImage(named: "blau")
    case Continent.ANTARTKIS:
       cell.imageView?.image = UIImage(named: "antarktis")
    default:
       cell.imageView?.image = UIImage(named: "default")
  }
  return cell
}
```

Neue Methoden der TableView für das Editieren:

```
   // diese Methode wird von NoYes-Dialog aufgerufen
  func deleteYes(){
    print("delete Yes")
    if let indexPath = tableview.indexPathForSelectedRow {
       let city:City = cities[indexPath.row]
      cities.remove(at: indexPath.row)
      tableview.reloadData()
     }
  }
   // diese Methode wird von NoYes-Dialog aufgerufen
  func deleteNo(){
    print("delete No")
  }

  @IBAction func bndeleteclick(_ sender: UIButton) {
    if let indexPath = tableview.indexPathForSelectedRow {
       let city:City = cities[indexPath.row]
```

```
      let mess:String = "Wollen Sie wirklich diesen Eintrag
löschen?\n\nCity:"+city.name+"\nKontinent:
"+getContinentText(city.continent)+"\nBem: "+city.remark

      Basis.showNoYesDialog(parent: self, _title: "Löschen eines Eintrags",
_message: mess, _yesFunc: deleteYes, _noFunc: deleteNo)
     }
  }


   // diese Methode wird vor dem Sprung zum InputController aufgerufen
  override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
      if let button:UIButton = sender as? UIButton {
         // Abfrage, welcher Schalter: Übergabe der Stadt
        if button==bnnew {
          if let dest = segue.destination as? InputViewController {
           dest.modus =   ChangeMode.NEW
          }
        }
        else {
         // Fehlermeldung
        }
      } // new Stadt
     }  // if
    }  // prepare


  // Rücksprung vom ChangeViewController Abbruch
     @IBAction func unwindCancel(_ segue:UIStoryboardSegue) {
    // keine Aktion notwendig
    }  // unwindToEditViewCancel

  // Rücksprung vom ChangeViewController Save
  // hier müssen beide „Events" eiongetragen (Edit und New)
   @IBAction func unwindOk(_ segue:UIStoryboardSegue) {
    if let src = segue.source as? InputViewController {
      print("in unwindOK")
      let city = src.city
      print(city.name)
      print(city.remark)
      print(city.continent)
      switch (src.modus) {
      case .EDIT:
        //
        print("edit")
        if let indexPath = tableview.indexPathForSelectedRow {
           let citytable:City = cities[indexPath.row]
          citytable.initCity(city)
          tableview.reloadRows(at: [indexPath], with:
UITableView.RowAnimation.right)
         }
      case .NEW:
        print("new")
        let citytable = City( Continent.EUROPE ,"",  "",  Visited.NONE)
        citytable.initCity(city)
        cities.append(citytable)
        tableview.reloadData()
```

```
          //
      } // switch (src.modus) {

  }  // unwindOk

  }


}
```

Referenzen und Variablen:

```
  // Referenzen
@IBOutlet var bndelete: UIButton!
@IBOutlet var bnnew: UIButton!

@IBOutlet var tableview: UITableView!

var cities : [City] = []    // leeres Array, Liste
```

viewDidLoad:

```
override func viewDidLoad() {
   super.viewDidLoad()
   cities = CitiesDB.loadCitiesIntern()

   tableview.delegate=self
   tableview.dataSource=self

   labelInfo.text="-"
   bnedit.isEnabled=false
   bndelete.isEnabled=false
}
```

12.5.7 Löschen eines Eintrags

```
func deleteYes(){
   print("delete Yes")
   if let indexPath = tableview.indexPathForSelectedRow {
      cities.remove(at: indexPath.row)
      tableview.reloadData()
   }
}

func deleteNo(){
  print("delete No")
}

@IBAction func bndeleteclick(_ sender: Any) {
   if let indexPath = tableview.indexPathForSelectedRow {
```

```
        let city:City = cities[indexPath.row]
        let mess:String = "Wollen Sie wirklich diesen Eintrag löschen?\n\n" +
          "City:"+city.name+"\nKontinent: "+getContinentText(city.continent)+
          "\nBem: "+city.remark
        Basis.showNoYesDialog(parent: self, _title: "Löschen eines Eintrags",
           _message: mess, _yesFunc: deleteYes, _noFunc: deleteNo)
    }
}
```

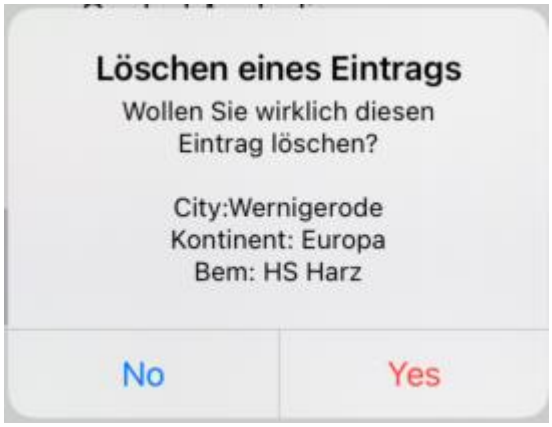


Abbildung 268 Abfrage zum Löschen eines Eintrags

12.5.8 Input-ViewController

Nun einen zweiten ViewController aus dem Lib-Controller einfügen

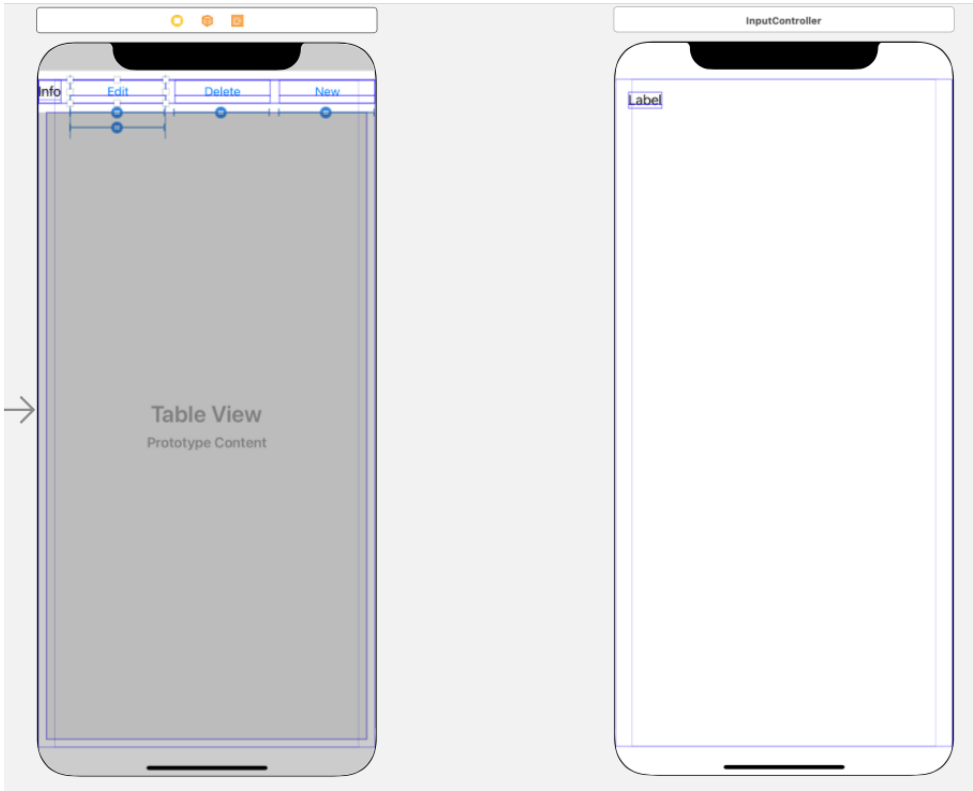
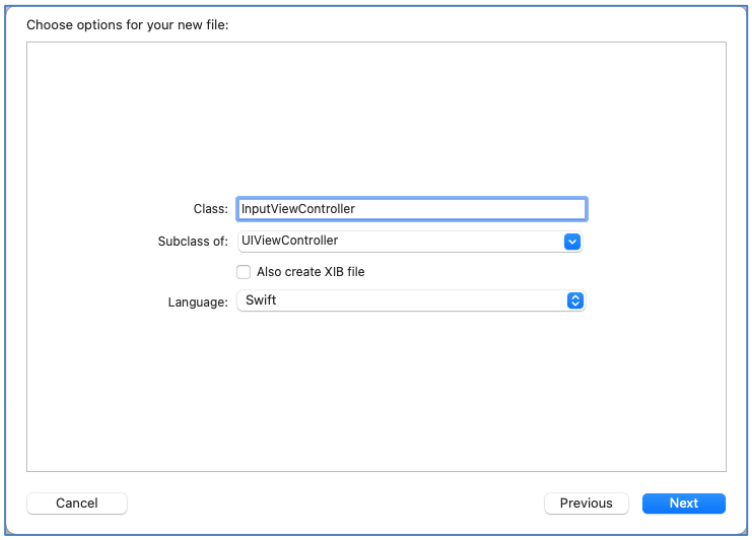


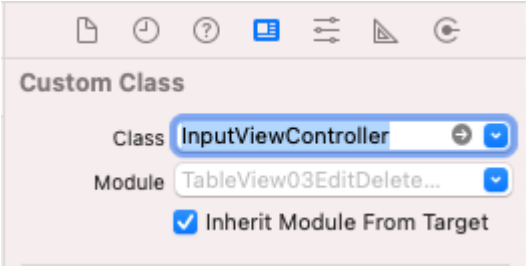
Abbildung 269 Anzeige beider ViewController (rechts mit Testlabel)

12.5.9 Swift-Klasse für den 2. View

Anlegen der Swift-Klasse für den Edit-ViewController mit Cmd+N



Nun den zweiten ViewController aktivieren, drei Schalter, ganz links und rechts obne im Property-Fenster die Klasse auswählen:



12.5.10 Schalter New

Ablauf:

- Aktivieren des Schalter „New“. Entweder in der Grafik oder im linken Property-Fenster.
- Ctrl-Taste drücken
- Ziehen der Maus zum zweiten ViewController
- Loslassen
- Auswahl „Show“

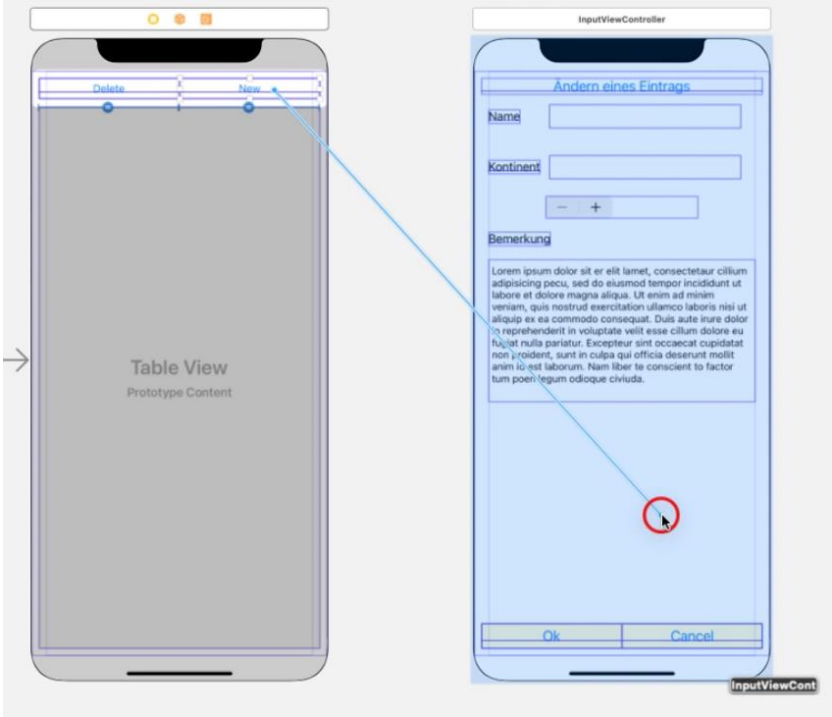


Abbildung 270 Link von New-Schalter zum 2. ViewController

12.5.11 Double-Click

Nun muss der Double-Click eingefügt werden. Dazu fügt man in „viewDidLoad“ folgenden Quellcode ein:

```
let tapGestureRecognizer2 = UITapGestureRecognizer(target: self,
            action: #selector(didTapView2(_:)))
tapGestureRecognizer2.numberOfTapsRequired = 2
tableview.addGestureRecognizer(tapGestureRecognizer2)
```

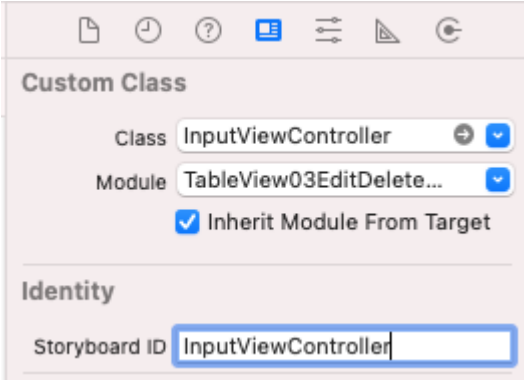
Dzu benötigt man noch die Event-Methode mit dem manuellen Aufruf des zweiten ViewControllers.

```
@IBAction func didTapView2(_ sender: UITapGestureRecognizer) {
  if let indexPath = tableview.indexPathForSelectedRow {
    let city:City = cities[indexPath.row]
    print("Doppel Tap: "+city.name)

    let storyBoard: UIStoryboard = UIStoryboard(name: "Main", bundle: nil)
    if let newViewController = storyBoard.instantiateViewController(
        withIdentifier: "InputViewController") as? InputViewController {
         let city:City = cities[indexPath.row]
         newViewController.city=city
         newViewController.modus =   ChangeMode.EDIT
         self.present(newViewController, animated: true, completion: nil)
    } // if
  }// if
}  // didTapView2
```

Der Rücksprung arbeitet dann wie im dritten Beispiel.

In der obigen Methode erzeugt man einen ViewController mit dem „Identifier“ **InputViewController**. Dieser Identifier ist bisher noch nie gesetzt wurden. Man aktiviert den InputController und geht in das rechte Property-Fenster. Dort setzt man den Identifier wie in der unteren Abbildung dargestellt.



12.5.12 UI-Aufbau des zweiten Controllers

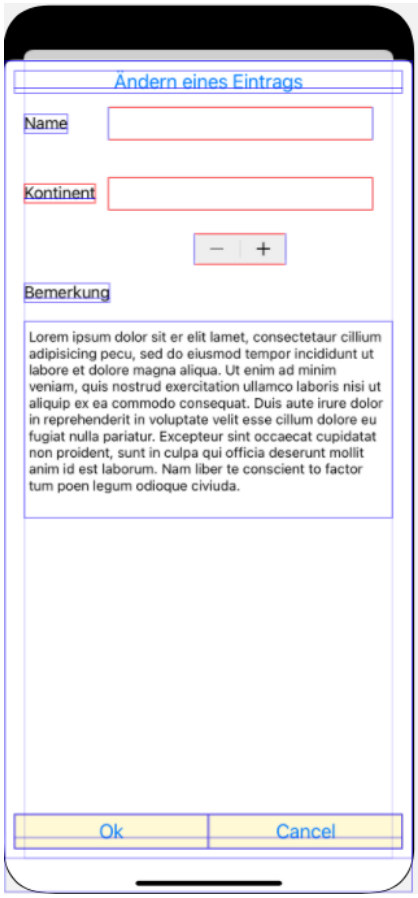


Abbildung 271 UI-Aufbau des zweiten ViewControllers

LabelCaption

- Top 10
- Left: 10
- Right: 10

Label Name:

- Top 20
- Left: 20

Textfield Name

- Top 13
- Left: 40
- Right: 40

Label Kontinent:

- Top: 50
- Left: 20

Textfield Kontinent:

- Top 37
- Left: 12

- Right: 40

Stepper Kontinent:
- Top 23
- Left: 100
- Right: 100

Label Bemerkung:
- Top: 18 (zum Label Kontinent)
- Left: 20

TextView Bemerkung:
- Top 18
- Left: 20
- Right: 20
- **Feste Höhe: 200**

Horizontale Box:
- Left: 10
- Right: 10
- Bottom 10

Schalter ok und Schalter cancel:
- **Gleiche Breite**

Referenzen erstellen:
- @IBOutlet var labelcaption: UILabel!
- @IBOutlet var tname: UITextField!
- @IBOutlet var tcontinent: UITextField!
- @IBOutlet var tremark: UITextView!
- @IBOutlet var steppercontinent: UIStepper!
- @IBOutlet var bnok: UIButton!
- @IBOutlet var bncancel: UIButton!

Event erstellen (nicht changedEnded nehmen):
- @IBAction func **tnamechanged**(_ sender: UITextField) {
- @IBAction func **stepperkontinentchanged**(_ sender: UIStepper) {

12.5.13 Code für den InputViewController

Variablen:

```
  // in dieser Variable findet die Parameterübergabe statt!
var city:City=City( Continent.AMERICA ,  "---",  "--------", Visited.NONE)
var modus:ChangeMode = ChangeMode.EDIT
```

Referenzen:

```
 @IBOutlet var labelcaption: UILabel!
```

```
 @IBOutlet var tname: UITextField!
 @IBOutlet var tcontinent: UITextField!
 @IBOutlet var tremark: UITextView!
 @IBOutlet var steppercontinent: UIStepper!
 @IBOutlet var bnok: UIButton!
 @IBOutlet var bncancel: UIButton!
```

Delegate für die TextView Änderung:

```
class InputViewController: UIViewController,  UITextViewDelegate{
  // es gibt leider kein OnChanged-Event
```

viewDidLoad:

```
override func viewDidLoad() {
    super.viewDidLoad()

    if modus==ChangeMode.EDIT {
        labelcaption.text = "Ändern von: "+city.name
        bnok.setTitle("Änderung speichern", for: UIControl.State.normal)
    }
    else {
        labelcaption.text = "Neuer Eintrag"
        bnok.setTitle("Eintrag einfügen", for: UIControl.State.normal)
    }
     // tcontinent wird über dem Stepüper gesetzt
    tcontinent.isEnabled = false
     // erstz aktiv bei einer Änderung
    bnok.isEnabled = false

    tremark.delegate=self
        // werte eintragen
    tname.text = city.name
    tremark.text = city.remark
    let conti:String = continente[city.kontinent.rawValue]
    tcontinent.text = conti
    steppercontinent.minimumValue=0
    steppercontinent.maximumValue=Double(continente.count-1)
    steppercontinent.value = Double(city.kontinent.rawValue)
    steppercontinent.stepValue=1
}
```

onChanged-Events:

```
@IBAction func tnamechanged(_ sender: UITextField) {
   bnok.isEnabled = true
}

@IBAction func stepperkontinentchanged(_ sender: UIStepper) {
    bnok.isEnabled = true
    let index:Int = Int(steppercontinent.value)
    let conti:String = continente[index]
    tcontinent.text = conti
}

func textViewDidChange(_ textView: UITextView) {
```

```
    //print(tremark.text!)
    bnok.isEnabled = true
}
```

12.5.14 Rücksprung

Im ersten ViewController wurden zwei „unwind“-Methoden eingebaut:

- unwindCancel
- unwindOk

Diese beiden Methoden werden durch die Schalter im InputViewController nach dem Beenden aufgerufen. Dazu muss man erstmal die prepare-Methode implementieren. Diese wird **vor** Rücksprung aufgerufen. Man benutzt die Methode, um die Daten zurückzuspeichern.

Prepare-Methode:

```
override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
    city.name=tname.text!
    city.remark=tremark.text!
    city.kontinent = getContinent(Int(steppercontinent.value))
}
```

Nun folgen noch die Links von den Schalter zum Exit-Schalter. Der Exit-Schalter ist der rechte der oberen drei Schalter. Man klickt den Ok-Schalter an, drückt die Ctrl-Taste und zieht die Linie zum rechten Schalter. Nach dem Loslassen der Maus wählt man die entsprechende Rücksprung-Methode aus.

12.5.15 Komplette Quellcodes

Hier noch einmal der komplette Quellcode des ersten ViewController

```
import UIKit

class ViewController: UIViewController, UITableViewDelegate,
UITableViewDataSource {
```

```
  @IBOutlet var tableview: UITableView!
  @IBOutlet var bndelete: UIButton!
  @IBOutlet var bnnew: UIButton!

  var cities : [City] = []

  override func viewDidLoad() {
    super.viewDidLoad()
    cities = CitiesDB.loadCitiesIntern()

    tableview.delegate=self
    tableview.dataSource=self

    let tapGestureRecognizer2 = UITapGestureRecognizer(target: self,
           action: #selector(didTapView2(_:)))
    tapGestureRecognizer2.numberOfTapsRequired = 2
    tableview.addGestureRecognizer(tapGestureRecognizer2)
  }


  @IBAction func didTapView2(_ sender: UITapGestureRecognizer) {
    if let indexPath = tableview.indexPathForSelectedRow {
     let city:City = cities[indexPath.row]
     print("Doppel Tap: "+city.name)

     let storyBoard: UIStoryboard = UIStoryboard(name: "Main", bundle: nil)
     if let newViewController = storyBoard.instantiateViewController(
       withIdentifier: "InputViewController") as? InputViewController {
        let city:City = cities[indexPath.row]
        newViewController.city=city
        newViewController.modus =   ChangeMode.EDIT
        self.present(newViewController, animated: true, completion: nil)
      }
     }
  }



  func numberOfSections(in tableView: UITableView) -> Int {
     // #warning Incomplete implementation, return the number of sections
     return 1
  }

  func tableView(_ tableView: UITableView, numberOfRowsInSection section:
Int) -> Int {
    return cities.count
  }



    // SelectecIndexChanged
   func tableView(_: UITableView, didSelectRowAt: IndexPath) {
     let row = (didSelectRowAt as NSIndexPath).row
    let city:City = cities[row]
    print("selectIndex: "+city.name)

   }
```

```
  func tableView(_ tableView: UITableView, cellForRowAt indexPath:
IndexPath) -> UITableViewCell {
    let cell: UITableViewCell = UITableViewCell(style:
UITableViewCell.CellStyle.subtitle
                          , reuseIdentifier: "reuseIdentifier")

    // Configure the cell...
    let row = (indexPath as NSIndexPath).row
    let city:City = cities[ row ]
    cell.textLabel?.text = city.name
    cell.detailTextLabel?.text = city.remark

      switch city.continent {
      case Continent.AFRICA:
        cell.imageView?.image = UIImage(named: "schwarz")

      case Continent.AMERICA:
        cell.imageView?.image = UIImage(named: "rot")
      case Continent.ASIA:
        cell.imageView?.image = UIImage(named: "gelb")
      case Continent.AUSTRALIA:
        cell.imageView?.image = UIImage(named: "gruen")
      case Continent.EUROPE:
        cell.imageView?.image = UIImage(named: "blau")
      case Continent.ANTARTKIS:
        cell.imageView?.image = UIImage(named: "antarktis")

      default:
        cell.imageView?.image = UIImage(named: "default")
      }

    return cell
  }



  func deleteYes(){
    print("delete Yes")
    if let indexPath = tableview.indexPathForSelectedRow {
       //let city:City = cities[indexPath.row]
      cities.remove(at: indexPath.row)
      tableview.reloadData()
     }
  }
  func deleteNo(){
    print("delete No")
  }
  @IBAction func bndeleteclick(_ sender: Any) {
    if let indexPath = tableview.indexPathForSelectedRow {
       let city:City = cities[indexPath.row]
      let mess:String = "Wollen Sie wirklich diesen Eintrag
löschen?\n\nCity:"+city.name+"\nKontinent:
"+getContinentText(city.continent)+"\nBem: "+city.remark
      Basis.showNoYesDialog(parent: self, _title: "Löschen eines Eintrags",
_message: mess, _yesFunc: deleteYes, _noFunc: deleteNo)
     }
  }
```

```
  override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
      if let button:UIButton = sender as? UIButton {
         // Abfrage, welcher Schalter: Übergabe der Stadt

      if button==bnnew {
        if let dest = segue.destination as? InputViewController {
           dest.modus =   ChangeMode.NEW

        }
      } // new Stadt
      else {

      }
     }  // if
    }  // prepare


  // Rücksprung vom ChangeViewController Abbruch
     @IBAction func unwindCancel(_ segue:UIStoryboardSegue) {
    print("cancel")
    // keine Aktion notwendig
    }  // unwindToEditViewCancel

  // Rücksprung vom ChangeViewController Save
   @IBAction func unwindOk(_ segue:UIStoryboardSegue) {
    if let src = segue.source as? InputViewController {
      print("in unwindOK")
      let city = src.city
      print(city.name)
      print(city.remark)
      print(city.continent)
      switch (src.modus) {
      case .EDIT:
        //
        print("edit")
        if let indexPath = tableview.indexPathForSelectedRow {
           let citytable:City = cities[indexPath.row]
          citytable.initCity(city)
          tableview.reloadRows(at: [indexPath], with:
UITableView.RowAnimation.right)
         }
      case .DELETE:
        print("Delete")

      case .NEW:
        print("new")
        let citytable = City( Continent.EUROPE ,"",  "",  Visited.NONE)
        citytable.initCity(city)
        cities.append(citytable)
        tableview.reloadData()
      } // switch (src.modus) {
   }  // if
  }  // unwindOk
}
```

Hier der Quellcode des InputViewControllers (komplett identisch mit dem dritten Beispiel)

```
import UIKit
```

```
class InputViewController: UIViewController,  UITextViewDelegate{

  var city:City=City( Continent.AMERICA ,  "---",  "--", Visited.NONE)
  var modus:ChangeMode = ChangeMode.EDIT


  @IBOutlet var labelcaption: UILabel!
  @IBOutlet var tname: UITextField!
  @IBOutlet var tcontinent: UITextField!
  @IBOutlet var tremark: UITextView!
  @IBOutlet var steppercontinent: UIStepper!
  @IBOutlet var bnok: UIButton!
  @IBOutlet var bncancel: UIButton!


  override func viewDidLoad() {
    super.viewDidLoad()

    if modus==ChangeMode.EDIT {
      labelcaption.text = "Ändern von: "+city.name
      bnok.setTitle("Änderung speichern", for: UIControl.State.normal)
    }
    else {
      labelcaption.text = "Neuer Eintrag"
      bnok.setTitle("Eintrag einfügen", for: UIControl.State.normal)
    }

    tcontinent.isEnabled = false
    bnok.isEnabled = false

    tremark.delegate=self
    // werte eintragen
    tname.text = city.name
    tremark.text = city.remark
    let conti:String = continente[city.kontinent.rawValue]
    tcontinent.text = conti
    steppercontinent.minimumValue=0
    steppercontinent.maximumValue=Double(continente.count-1)
    steppercontinent.value = Double(city.kontinent.rawValue)
    steppercontinent.stepValue=1

  }


  @IBAction func tnamechanged(_ sender: UITextField) {
    bnok.isEnabled = true
  }

  @IBAction func stepperkontinentchanged(_ sender: UIStepper) {
    bnok.isEnabled = true
    let index:Int = Int(steppercontinent.value)
    let conti:String = continente[index]
    tcontinent.text = conti
  }

  func textViewDidChange(_ textView: UITextView) {
    //print(tremark.text!)
```

```
    bnok.isEnabled = true
  }




  override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
    // Get the new view controller using segue.destination.
    // Pass the selected object to the new view controller.

    city.name=tname.text!
    city.remark=tremark.text!
    city.kontinent = getContinent(Int(steppercontinent.value))

  }


}
```

Prinzipiell könnte man mit dem Double-Click Event auch das Delete-Event mit berücksichtigen. Dann müssten man aber drei Schalter haben:

- Speichern
- Löschen
- Abbrechen

Im „New“-Event hat man dann aber nur noch **zwei** Schalter! Einer muss dann versteckt werden.

12.6 TableView mit selbstdefinierter Zelle (TableView05Cell)

Dieses Kapitel beschreibt, wie man für eine TableView eine selbstdefinierte Zelle benutzen kann. Das ist manchmal notwendig, wenn man zum Beispiel zwei IMageView darstellen will.

Vorschau

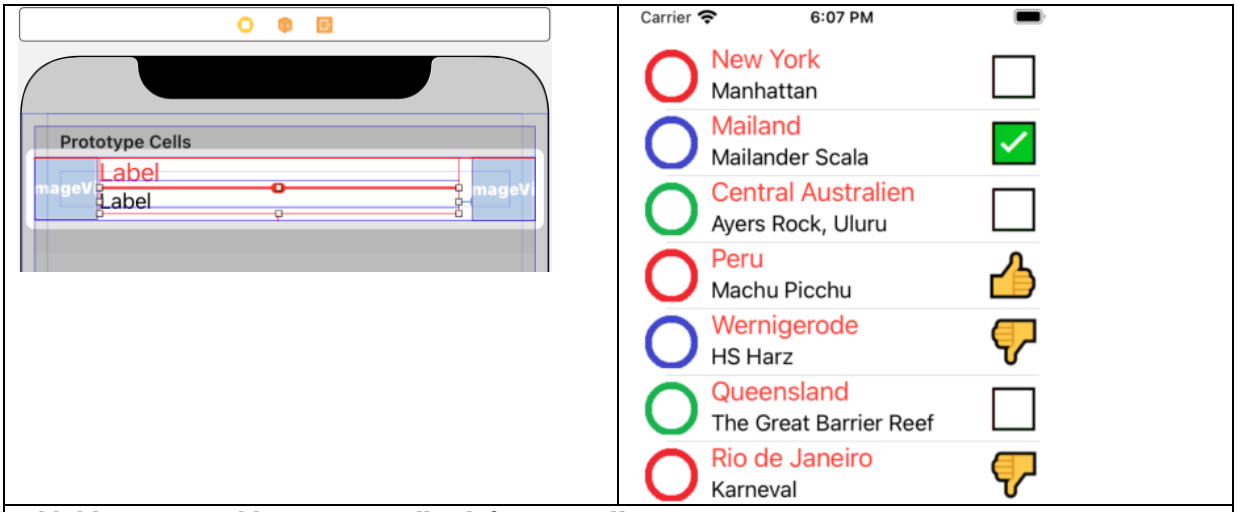


Abbildung 272 TableView mit selbstdefinerter Zelle

12.6.1 Anlegen eines Projektes

Legen Sie ein normales Projekt „App“ an und speichern Sie diese.

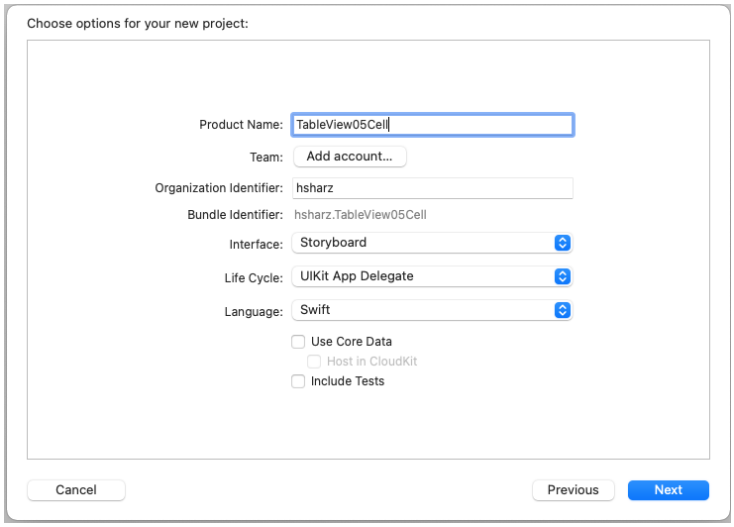


Abbildung 273 Anlegen der App TableView05Cell

12.6.2 TableView-Controller einfügen

Fügen Sie nun aus dem Lib-Dialog eine TableView in den ViewController. Als Constraint geben Sie ein:

- Top 10
- Left: 10
- Right: 10
- Bottom: 10

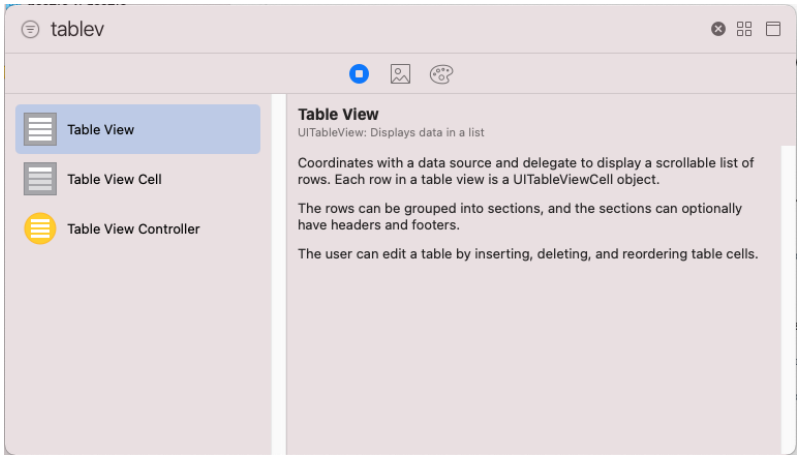
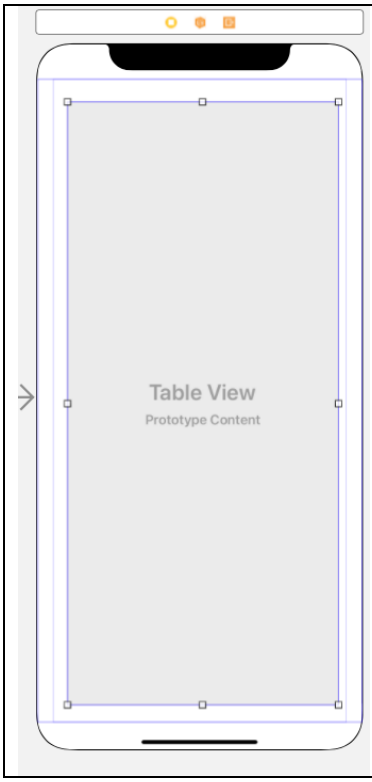


Abbildung 274 TableView einfügen

Das Projekt müsste dann so aussehen:



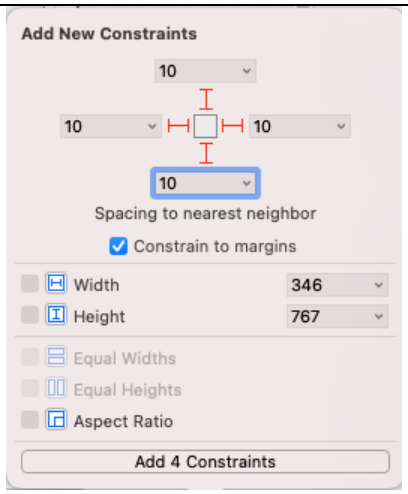


Abbildung 276 Constraints der TableView

Abbildung 275 TableView	

Im nächsten Schritt erstellt man eine Referenz der UITableView im ViewController.

12.6.3 Hilfsklassen für die TableView

In der Tabelle sollen Städte angezeigt werden. Dazu brauchen wir eine Klasse „City“ und eine Verwaltungsklasse „CitiesDB“.

Anlegen einer neuen Klasse „City.swift“

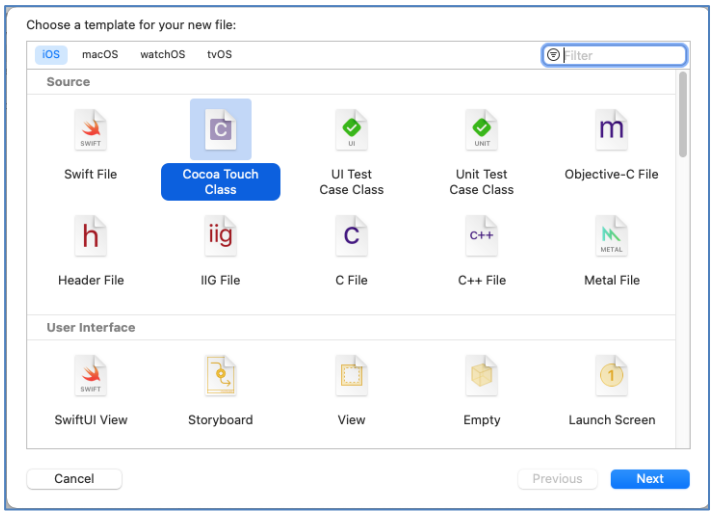


Abbildung 277 Einfüge-Dialog für die Klasse City.swift (Cmd+N)

Nun den Namen und den Typ eintragen:

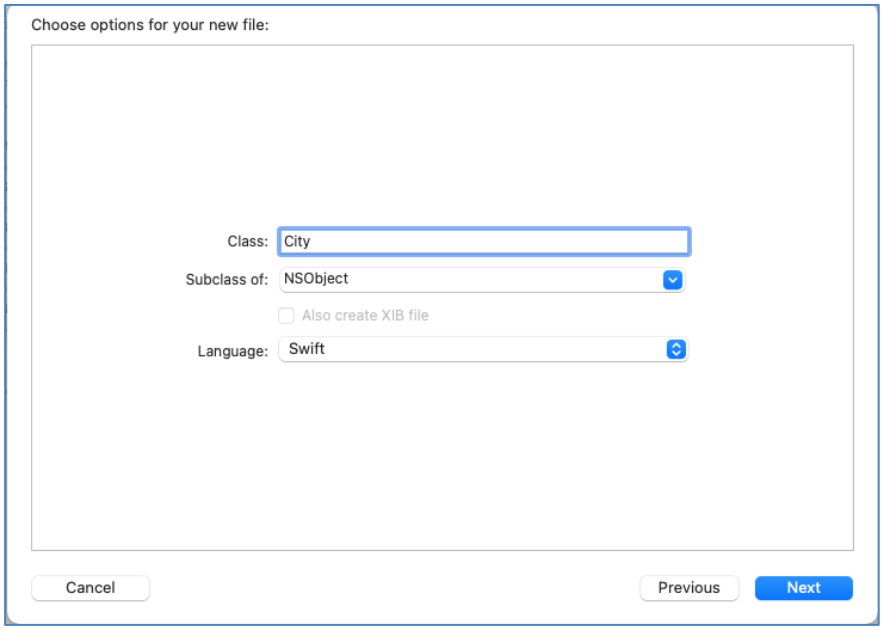


Abbildung 278 Name und Typ der Klasse City

Nun noch eine zweite Klasse „Enums.swift“ und eine dritte Klasse „CitiesDB.swift“ einfügen.
In der Klasse „Enums.swift“ werden die Aufzählungen, Enumeration eingetragen:

```
import UIKit


enum Continent:Int {         // Deklaration
  case  AFRICA=0, AMERICA, ASIA, AUSTRALIA, EUROPE, ANTARTKIS
} }

enum Visited : Int{         // Deklaration
  case  NONE=0, DOWN, AVERAGE, UP
}
```

Quellcode für Enums.swift

Eine Enumeration ist im Quellcode wesentlich sicherer als ein int-Wert mit Konstanten. Das wäre ein sehr schlechter Stil.

```
import UIKit

class City: NSObject {
      // Attribute
   var continent: Continent=Continent.AFRICA
   var name : String = ""
   var remark : String = ""
   var visited:Visited = Visited.NONE

     // Konstruktor
   init(_ continent: Continent,_ name:String, _ remark:String, _ visited:Visited){
      self.continent = continent
      self.name = name
```

```
        self.remark = remark
        self.visited = visited
    }   // init

}
```

Quellcode für City.swift

```
import UIKit

class CitiesDB: NSObject {

   public static func loadCitiesIntern()  -> Array<City> {
     var cities=[City]()
     cities.append( City( Continent.AMERICA ,  "New York",  "Manhattan", Visited.NONE)  )
     cities.append( City( Continent.EUROPE , "Mailand", "Mailander Scala", Visited.AVERAGE))
     cities.append( City( Continent.AUSTRALIA ,"Central Australien","Ayers Rock, Uluru",
Visited.NONE)  )
     cities.append( City( Continent.AMERICA ,"Peru",  "Machu Picchu",  Visited.UP)  )
     cities.append( City( Continent.EUROPE ,"Wernigerode",  "HS Harz",  Visited.DOWN)  )
     cities.append( City( Continent.AUSTRALIA ,"Queensland",  "The Great Barrier Reef",
Visited.NONE)  )
     cities.append( City( Continent.AMERICA ,"Rio de Janeiro", "Karneval",  Visited.DOWN)  )
     cities.append( City( Continent.EUROPE ,"Brüssel",   "Atomium",  Visited.UP)  )
     cities.append( City( Continent.AUSTRALIA ,"Sidney",  "Opernhaus",  Visited.UP)  )
     cities.append( City( Continent.AFRICA ,"Kairo",  "Pyramiden",  Visited.NONE)  )
     cities.append( City( Continent.AMERICA ,"Chile",  "Atacama Wüste",  Visited.UP)  )
     cities.append( City( Continent.ASIA ,"Tokio",  "Shinjuku Gyoen National Garden",
Visited.DOWN)  )
     cities.append( City( Continent.EUROPE ,"Kiel",  "Kieler Woche",  Visited.UP)  )
     cities.append( City( Continent.AFRICA ,"Hunsbergen","Apollo 11 Höhle",  Visited.NONE) )
     cities.append( City( Continent.EUROPE ,"Dresden",  "Das grüne Zimmer",  Visited.NONE) )
     cities.append( City( Continent.AFRICA ,"Marrakesch",  "Botanische Garten Jardin
Majorelle",  Visited.DOWN)  )
     cities.append( City( Continent.AMERICA ,"San Fransisco",  "Golden Gate Bridge",
Visited.UP)  )
     cities.append( City( Continent.EUROPE ,"Halberstadt",  "John Cage",  Visited.NONE)  )
     cities.append( City( Continent.ASIA ,"Peking",  "Verbotene Stadt",  Visited.NONE)  )
     cities.append( City( Continent.EUROPE ,"Venezuela",  "Angel Falls",  Visited.UP)  )
     cities.append( City( Continent.AMERICA ,"Bilbao",  "Guggenheim-Museum",  Visited.UP) )
     cities.append( City( Continent.ASIA ,"Chiang Mai",  "Der weiße Tempel",  Visited.UP) )
     cities.append( City( Continent.AMERICA ,"Rio de Janeiro",  "Zuckerhut",  Visited.NONE))
     return cities
}
```

Quellcode für CitiesDB.swift

12.6.4 Symbole für die TableView

Einfügen der Symbole aus der Datei „mycellsymbols.zip“ in den Projektordner „assert“ per Drag&Drop“. **Bitte beachten Sie, dass Sie immer das Bild ohne Nummer ins Projekt ziehen**.

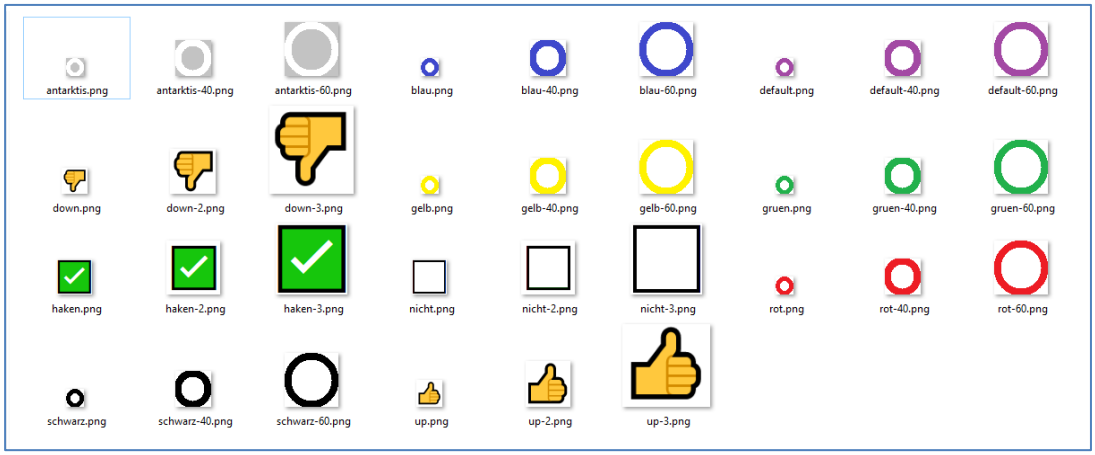


Abbildung 279 Symbole für die TableView

12.6.5 TableView mit einer Liste füllen

Da der ViewController kein TableViewController ist, müssen die Delegate manuell eingetragen werden. Danach muss also nur die Liste der Städte holen, die UITableView mit den Methoden verknüpfen (Interface) und die Anzahl der Gruppen, die Anzahl der Städte und den Aufbau der „Zelle“ eintragen.

Fügen Sie oben in der Klassendefinition folgende Protokolle, Delegate ein:
class ViewController: UIViewController , **UITableViewDelegate, UITableViewDataSource**

Methoden der TableView:

```
func numberOfSections(in tableView: UITableView) -> Int {
    return 1
}

func tableView(_ tableView: UITableView, numberOfRowsInSection section:
Int) -> Int {
    return cities.count
}

    // SelectecIndexChanged
func tableView(_: UITableView, didSelectRowAt: IndexPath) {
   // label setzen
}

     // Zelle aufbauen
func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath)
-> UITableViewCell {
      …
      return cell
}
```

Referenzen und Variablen:

```
  // Referenzen
@IBOutlet var tableview: UITableView!

@IBOutlet var label1: UILabel!
@IBOutlet var label2: UILabel!

var cities : [City] = []    // leeres Array, Liste
```

viewDidLoad:

```
func viewDidLoad() {
  super.viewDidLoad()

  cities = CitiesDB.loadCitiesIntern()

  tableview.delegate=self
  tableview.dataSource=self

}
```

Anzahl der Gruppen:

```
func numberOfSections(in tableView: UITableView) -> Int {
   return 1
}
```

Anzahl der Städte:

```
override func tableView(_ tableView: UITableView,
                              numberOfRowsInSection section: Int) -> Int {
   return cities.count

}
```

SelectedIndex:

```
// SelectecIndexChanged
func tableView(_: UITableView, didSelectRowAt: IndexPath) {
  let row = (didSelectRowAt as NSIndexPath).row
  let city:City = cities[row]
  label1.text="row: "+String(row)
  label2.text="City: "+city.name
}
```

Eintragen des Textes in der Zelle:

```
func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath)
-> UITableViewCell {

  let cell: UITableViewCell = UITableViewCell(
```

```
      style: UITableViewCell.CellStyle.subtitle,
      reuseIdentifier: "reuseIdentifier")


         // Configure the cell...
   let row = (indexPath as NSIndexPath).row
   let city:City = cities[ row ]
   cell.textLabel?.text = city.name
   cell.detailTextLabel?.text = city.remark

   switch city.continent {
      case Continent.AFRICA:
         cell.imageView?.image = UIImage(named: "schwarz")

      case Continent.AMERICA:
         cell.imageView?.image = UIImage(named: "rot")
      case Continent.ASIA:
         cell.imageView?.image = UIImage(named: "gelb")
      case Continent.AUSTRALIA:
         cell.imageView?.image = UIImage(named: "gruen")
      case Continent.EUROPE:
         cell.imageView?.image = UIImage(named: "blau")
      case Continent.ANTARTKIS:
         cell.imageView?.image = UIImage(named: "antarktis")

      default:
         cell.imageView?.image = UIImage(named: "default")
      }
   }
   return cell
}
```

Beachten Sie, dass man mit dem Quellcode „UITableViewCell.CellStyle.subtitle“ die Zelle mit dem Style (Text, Subtext und Image) definiert.

Typen der cell-Instanz:

Insgesamt gibt es vier vordefinierte Formate:

Typ	**Erläuterung**
UITableViewCellStyle.**default**	Bild links, Titel rechts
UITableViewCellStyle.value1	Bild links, Titel in der Mitte, rechts Zusatzinformation
UITableViewCellStyle.value2	Zusatzinformation links, rechts Titel
UITableViewCellStyle.subtitle	Bild links, Titel rechts, darunter Zusatzinformation

Mögliche Elemente in der Zelle:

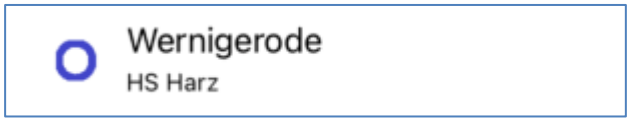
Typ	**Zuweisung**
textLabel	cell?.textLabel!.text ="Stadt"
detailTextLabel	cell?.detailTextLabel!.text ="Stadt"
imageView	cell?.imageView!.image = UIImage(named: "Bild6")

Beispiele:

default:

value1

value2



subtitle:

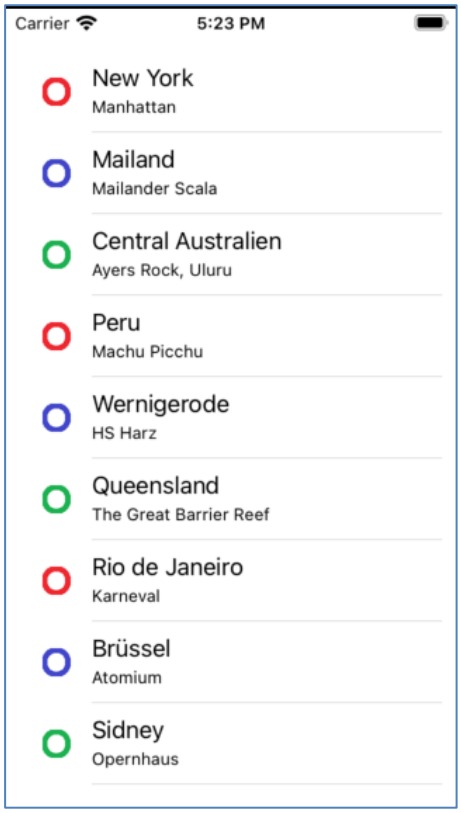


Abbildung 280 Aktueller Stand

Bis jetzt ist alles wie im ersten Beispiel. Nun wird eine Zelle in der TableView deklariert und die Zell-Elemente werden eingefügt:

- Zwei ImageView
- Zwei Labelfelder

12.6.6 Eigene Zelle definieren

Man aktiviert die TableView im Projektbaum. Dann ändert man die Anzahl der Prototyp Cells auf 1. Daraufhin ändert sich die Anzeige der Tableview:

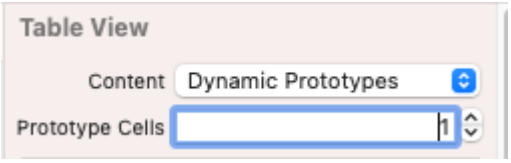


Abbildung 281 Prototyp Cells

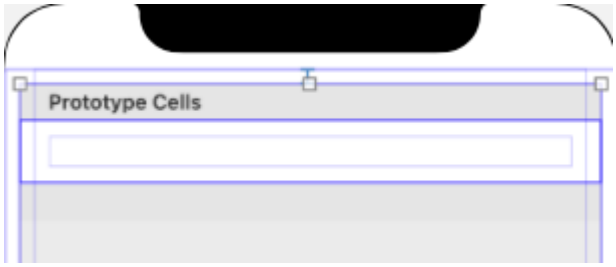


Abbildung 282 Anzeige des Platzes der leeren Zelle

Nun aktiviert man im Projektbaum die „Table View Cell“, um die Verbindung zur eigenen Cell-Klasse zu erhalten.

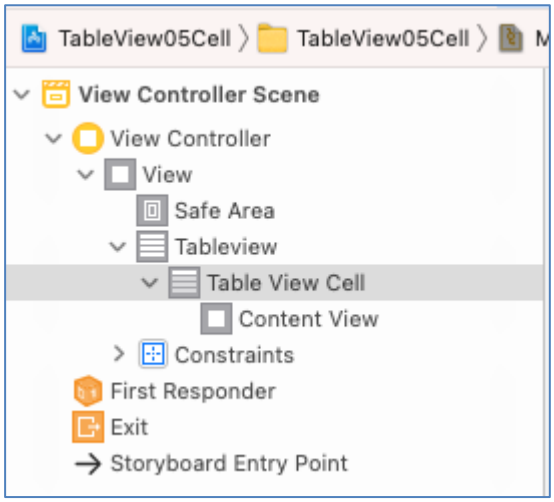
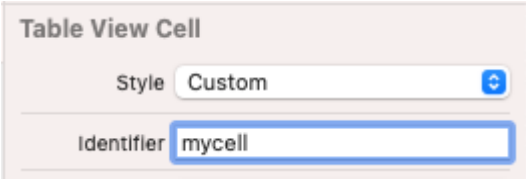


Abbildung 283 Table View Cell

Im rechten oberen Propertyfenster trägt man den Identifier der Cell-Klasse ein, hier „mycell“ ein. Dieser Identifier wird bei der Anzeige der Tabelle im ViewController benutzt!



Die Anzeige im linken Projektbaum sollte sich nun ändern:

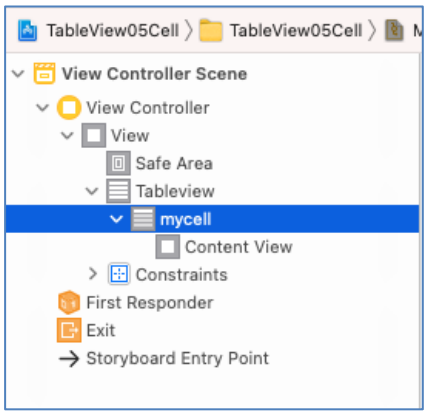


Abbildung 284 Identifier "mycell"

Nun zieht man aus dem GrafikLib-Dialog die Elemente und fügt die Constraints ein.

ImageView für den Kontinent:

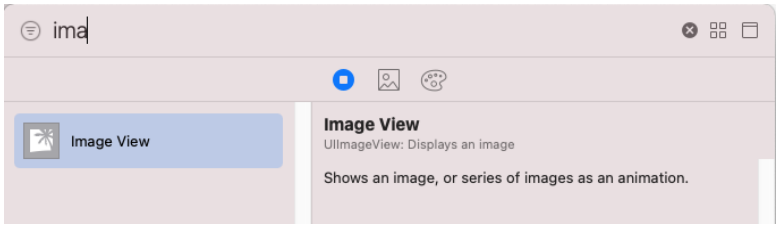


Abbildung 285 Contraints für das erste Bild

ImageView für den Besucherstatus (visited):

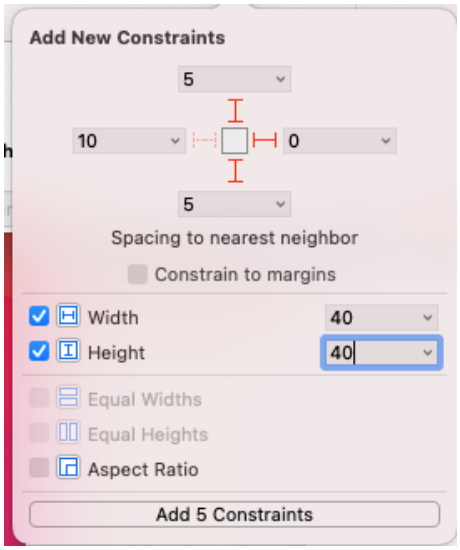


Abbildung 286 Contraints für das zweite Bild

Die Reihenfolge ist wichtig, da die beiden Labeltexte zwischen den Bilder platziert werden.

Labelfeld für den Städtenamen:

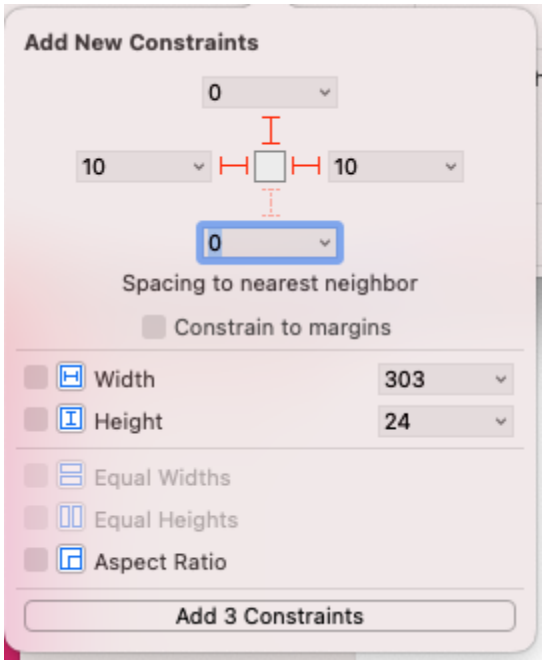


Abbildung 287 Contraints für den Städtenamen

Nun wird noch die Farbe rot gesetzt und die Schriftgröße auf 20pt gesetzt.

Labelfeld für die Bemerkung:

- Top: 2
- Left: 10
- Right: 10

Die „Zelle“ sollte jetzt so aussehen:

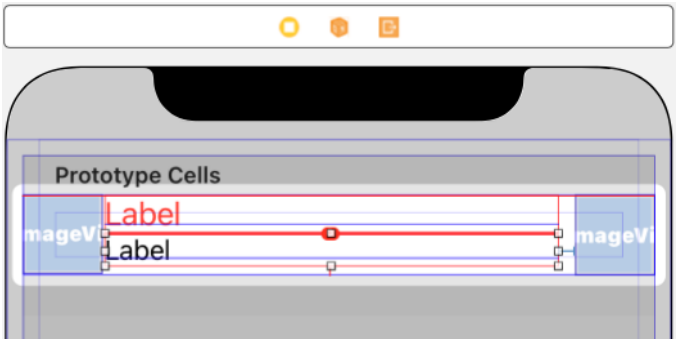


Abbildung 288 Anzeige der selbsdefinierten Zelle für eine TableView

Nun kann man noch die Titel ändern, so dass die Zelle im linken Projektbaum so aussieht:

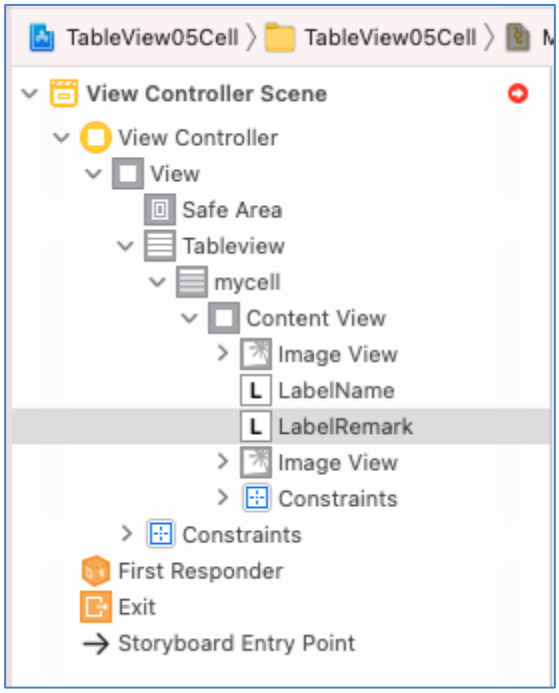


Abbildung 289 Anzeige der Zelle im linken Projektbaum

Nun fehlen noch drei Schritte:

- Anlegen der Klasse „MyCell“
- Erstellen der Referenzen von der TableView in die Klasse„MyCell“!!
- Ändern der Anzeige der TableView im ViewController

Mit Cmd+N wird eine neue Klasse erstellt:

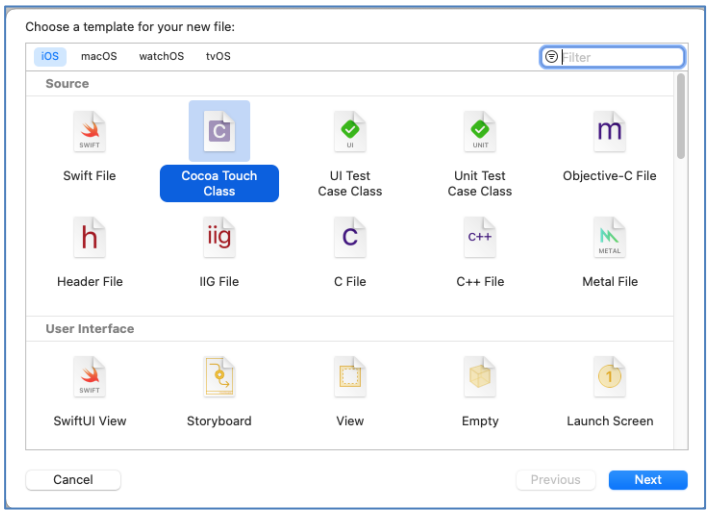


Abbildung 290 Klasse MyCell erstellen

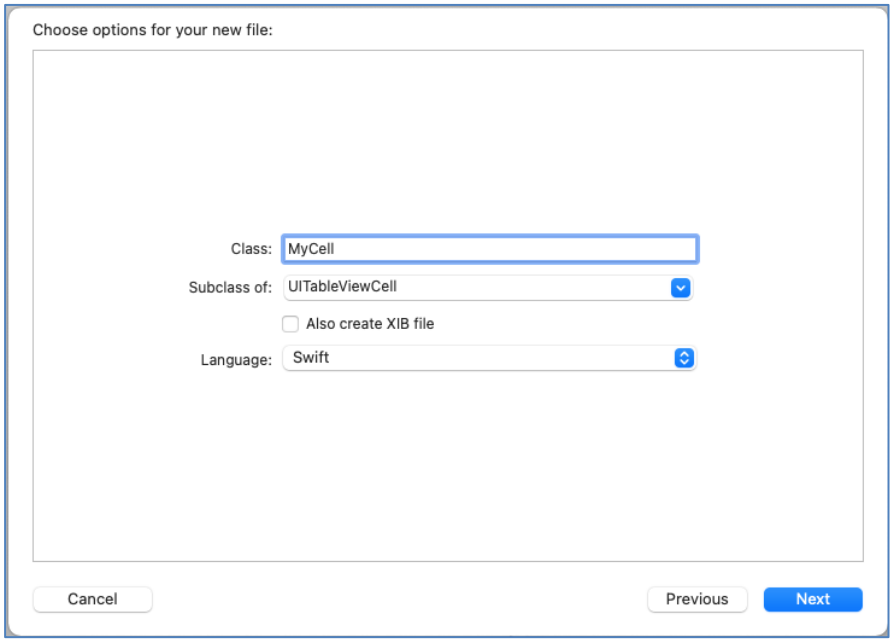


Abbildung 291 Name der Klasse MyCell eingeben

Nun verknüpft man die Zelle mit der Klasse:

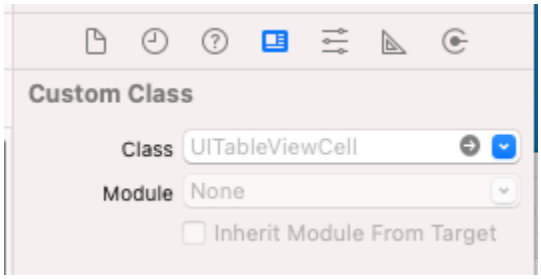


Abbildung 292 Eintragen der Cell-Klasse (1)

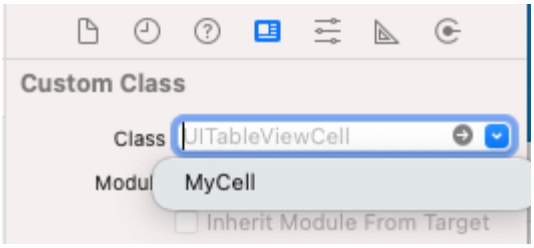


Abbildung 293 Eintragen der Cell-Klasse (2)

Anzeige der fertigen Klasse MyCell:

Main.storyboard | MyCell.swift

TableView05Cell › TableView05Cell › MyCell.swift › No Selection

```
1  //
2  //  MyCell.swift
3  //  TableView05Cell
4  //
5  //  Created by User1 on 11.09.21.
6  //
7
8  import UIKit
9
10 class MyCell: UITableViewCell {
       @IBOutlet var imgcontinent: UIImageView!
       @IBOutlet var title: UILabel!
       @IBOutlet var subtitle: UILabel!
       @IBOutlet var imgvisited: UIImageView!
15
16     override func awakeFromNib() {
17         super.awakeFromNib()
18         // Initialization code
19     }
20
21     override func setSelected(_ selected: Bool, animated:
           Bool) {
22         super.setSelected(selected, animated: animated)
23
24         // Configure the view for the selected state
25     }
26
27 }
28
```

Abbildung 294 Die Klasse MyCell

Nun muss noch die Anzeige der TableView-Cell auf mycell geäbdert werden:

Alte Lösung:

```
func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath)
-> UITableViewCell {
  let cell: UITableViewCell = UITableViewCell(style:
UITableViewCell.CellStyle.subtitle , reuseIdentifier: "reuseIdentifier")


  cell.textLabel?.text = city.name
  cell.detailTextLabel?.text = city.remark

  switch city.continent {
     case Continent.AFRICA:
        cell.imageView?.image = UIImage(named: "schwarz")
     case Continent.AMERICA:
```

```
        cell.imageView?.image = UIImage(named: "rot")
     case Continent.ASIA:
        cell.imageView?.image = UIImage(named: "gelb")
     case Continent.AUSTRALIA:
        cell.imageView?.image = UIImage(named: "gruen")
     case Continent.EUROPE:
        cell.imageView?.image = UIImage(named: "blau")
     case Continent.ANTARTKIS:
        cell.imageView?.image = UIImage(named: "antarktis")
     default:
        cell.imageView?.image = UIImage(named: "default")
  }
  return cell
}
```

Neue Lösung mit dem ImageView für visited:

```
func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath)
-> UITableViewCell {

  let cell = tableView.dequeueReusableCell( withIdentifier: "mycell", for:
indexPath) as! MyCell

  cell.title!.text = city.name
  cell.subtitle!.text = city.remark

  switch city.continent {
    case Continent.AFRICA:
      cell.imgcontinent!.image = UIImage(named: "schwarz")
    case Continent.AMERICA:
      cell.imgcontinent!.image = UIImage(named: "rot")
    case Continent.ASIA:
      cell.imgcontinent!.image = UIImage(named: "gelb")
    case Continent.AUSTRALIA:
      cell.imgcontinent!.image = UIImage(named: "gruen")
    case Continent.EUROPE:
      cell.imgcontinent!.image = UIImage(named: "blau")
    case Continent.ANTARTKIS:
      cell.imgcontinent!.image = UIImage(named: "antarktis")
    default:
      cell.imgcontinent!.image = UIImage(named: "default")
  }  // switch city.continent {

  switch (city.visited) {
    case Visited.NONE:
      cell.imgvisited.image = UIImage(named: "nicht")
    case Visited.DOWN:
      cell.imgvisited.image = UIImage(named: "down")
    case Visited.AVERAGE:
      cell.imgvisited.image = UIImage(named: "haken")
    case Visited.UP:
      cell.imgvisited.image = UIImage(named: "up")
    default:
      cell.imgcontinent.image = UIImage(named: "nicht")
  }
  return cell
}
```

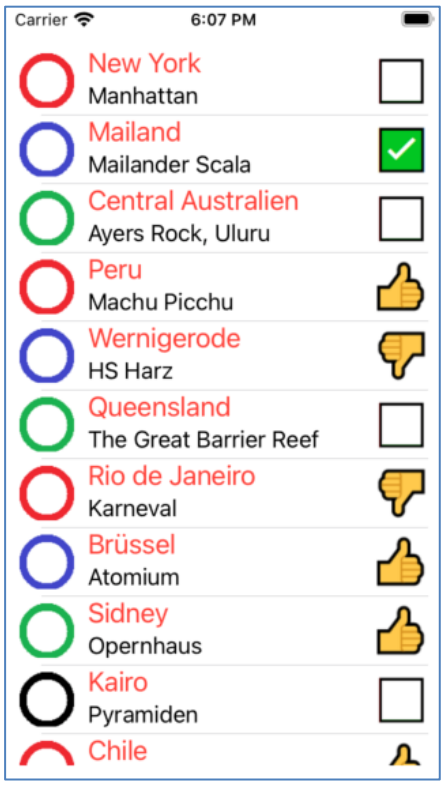


Abbildung 295 Anzeige der Musterlösung

Hier noch einmal der komplette Quellcode:

MyCell.swift

```
import UIKit

class MyCell: UITableViewCell {
    @IBOutlet var imgcontinent: UIImageView!
    @IBOutlet var title: UILabel!
    @IBOutlet var subtitle: UILabel!
    @IBOutlet var imgvisited: UIImageView!

    override func awakeFromNib() {
        super.awakeFromNib()
        // Initialization code
    }

    override func setSelected(_ selected: Bool, animated: Bool) {
        super.setSelected(selected, animated: animated)
    }

}
```

Anzeige des ViewControllers

```
import UIKit
```

```
class ViewController: UIViewController , UITableViewDelegate,
UITableViewDataSource{
    @IBOutlet var tableview: UITableView!

    var cities : [City] = []

    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view.
        cities = CitiesDB.loadCitiesIntern()

        tableview.delegate=self
        tableview.dataSource=self
      //  tableview.tableFooterView = UIView(frame: CGRect.zero)
    }





func numberOfSections(in tableView: UITableView) -> Int {
   // #warning Incomplete implementation, return the number of sections
   return 1
}

func tableView(_ tableView: UITableView, numberOfRowsInSection section:
Int) -> Int {
    return cities.count
}



  // SelectecIndexChanged
 func tableView(_: UITableView, didSelectRowAt: IndexPath) {
     let row = (didSelectRowAt as NSIndexPath).row
    let city:City = cities[row]
    print("selectIndex: "+city.name)

 }

func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath)
-> UITableViewCell {
    //let cell: UITableViewCell = UITableViewCell(style:
UITableViewCell.CellStyle.subtitle , reuseIdentifier: "reuseIdentifier")

    let cell = tableView.dequeueReusableCell( withIdentifier: "mycell",
for: indexPath) as! MyCell

    // Configure the cell...
    let row = (indexPath as NSIndexPath).row
    let city:City = cities[ row ]
    //cell.textLabel?.text = city.name
    //cell.detailTextLabel?.text = city.remark

    cell.title!.text = city.name
    cell.subtitle!.text = city.remark
```

```
        switch city.continent {
         case Continent.AFRICA:
            cell.imgcontinent!.image = UIImage(named: "schwarz")
         case Continent.AMERICA:
            cell.imgcontinent!.image = UIImage(named: "rot")
         case Continent.ASIA:
            cell.imgcontinent!.image = UIImage(named: "gelb")
         case Continent.AUSTRALIA:
            cell.imgcontinent!.image = UIImage(named: "gruen")
         case Continent.EUROPE:
            cell.imgcontinent!.image = UIImage(named: "blau")
         case Continent.ANTARTKIS:
            cell.imgcontinent!.image = UIImage(named: "antarktis")
         default:
            cell.imgcontinent!.image = UIImage(named: "default")
        } // switch

    switch (city.visited) {
       case Visited.NONE:
          cell.imgvisited.image = UIImage(named: "nicht")
        case Visited.DOWN:
          cell.imgvisited.image = UIImage(named: "down")
        case Visited.AVERAGE:
          cell.imgvisited.image = UIImage(named: "haken")
        case Visited.UP:
          cell.imgvisited.image = UIImage(named: "up")
        default:
          cell.imgcontinent.image = UIImage(named: "nicht")
    }

    return cell
}

}
```

Basis.swift:

```
import UIKit

class Basis {


  static  func   showOKDialog(parent:UIViewController, _title title:String,
_message message:String) {
       //Creating UIAlertController and
       //Setting title and message for the alert dialog
       let alertController = UIAlertController(title: title, message:
message, preferredStyle: .alert)  // alert actionSheet

       //the confirm action taking the inputs
       let okAction = UIAlertAction(title: "Ok", style: .default) { (_) in

        // self.ok()

       }
```

```
       //the cancel action doing nothing
      // let cancelAction = UIAlertAction(title: "Cancel", style: .cancel)
{ (_) in }



       //adding the action to dialogbox
       alertController.addAction(okAction)
      // alertController.addAction(cancelAction)

       //finally presenting the dialog box
    parent.present(alertController, animated: true, completion: nil)
     }  // showOKDialog




  static func   showNoYesDialog(parent:UIViewController,_title
title:String, _message message:String, _yesFunc  yesFunction : @escaping()-
>Void, _noFunc noFunction : @escaping()->Void) {
     //Creating UIAlertController and
     //Setting title and message for the alert dialog

    //let y=yesFunction
    //yesFunction(1)
    let alertController = UIAlertController(title: title, message: message,
preferredStyle: .alert)

     //the confirm action taking the inputs
     let noAction = UIAlertAction(title: "No", style: .default) { (_) in
       noFunction()
     }

     //the cancel action doing nothing
     let yesAction = UIAlertAction(title: "Yes", style: .destructive) { (_)
in
        yesFunction()
     }

     //adding the action to dialogbox
     alertController.addAction(noAction)
     alertController.addAction(yesAction)

     //finally presenting the dialog box
     parent.present(alertController, animated: true, completion: nil)
   }  // showYesNoDialog2



  static func  showYesNoDialog(parent:UIViewController,_title title:String,
_message message:String, _yesFunc  yesFunction : @escaping()->Void, _noFunc
noFunction : @escaping()->Void) {
    //Creating UIAlertController and
    //Setting title and message for the alert dialog

   //let y=yesFunction
   //yesFunction(1)
```

```
   let alertController = UIAlertController(title: title, message: message,
preferredStyle: .alert)

    //the confirm action taking the inputs
    let confirmAction = UIAlertAction(title: "Yes", style: .default) { (_)
in
    yesFunction()
    }

    //the cancel action doing nothing
    let cancelAction = UIAlertAction(title: "No", style: .destructive) {
(_) in

       noFunction()

    }



    //adding the action to dialogbox
    alertController.addAction(confirmAction)
    alertController.addAction(cancelAction)

    //finally presenting the dialog box
    parent.present(alertController, animated: true, completion: nil)
  }  // showYesNoDialog2




  static func  showYesNoCancelDialog(parent:UIViewController,_title
title:String, _message message:String, _yesFunc  yesFunction : @escaping()-
>Void, _noFunc noFunction : @escaping()->Void, _cancelFunc cancelFunction :
@escaping()->Void) {
       //Creating UIAlertController and
       //Setting title and message for the alert dialog

      //let y=yesFunction
      //yesFunction(1)
       let alertController = UIAlertController(title: title, message:
message, preferredStyle: .alert)

       //the confirm action taking the inputs
     let yesAction = UIAlertAction(title: "Yes", style: .default) { (_) in

       yesFunction()


       }

       //the cancel action doing nothing
       let noAction = UIAlertAction(title: "No", style: .destructive) { (_)
in

        noFunction()

       }
```

```
     //the cancel action doing nothing
     let cancelAction = UIAlertAction(title: "Cancel", style: .cancel) {
(_) in

           cancelFunction()

        }

       //adding the action to dialogbox
       alertController.addAction(yesAction)
       alertController.addAction(noAction)
       alertController.addAction(cancelAction)

       //finally presenting the dialog box
       parent.present(alertController, animated: true, completion: nil)
     }  // showYesNoCancelDialog


}
```

13 Master-Detail-Application

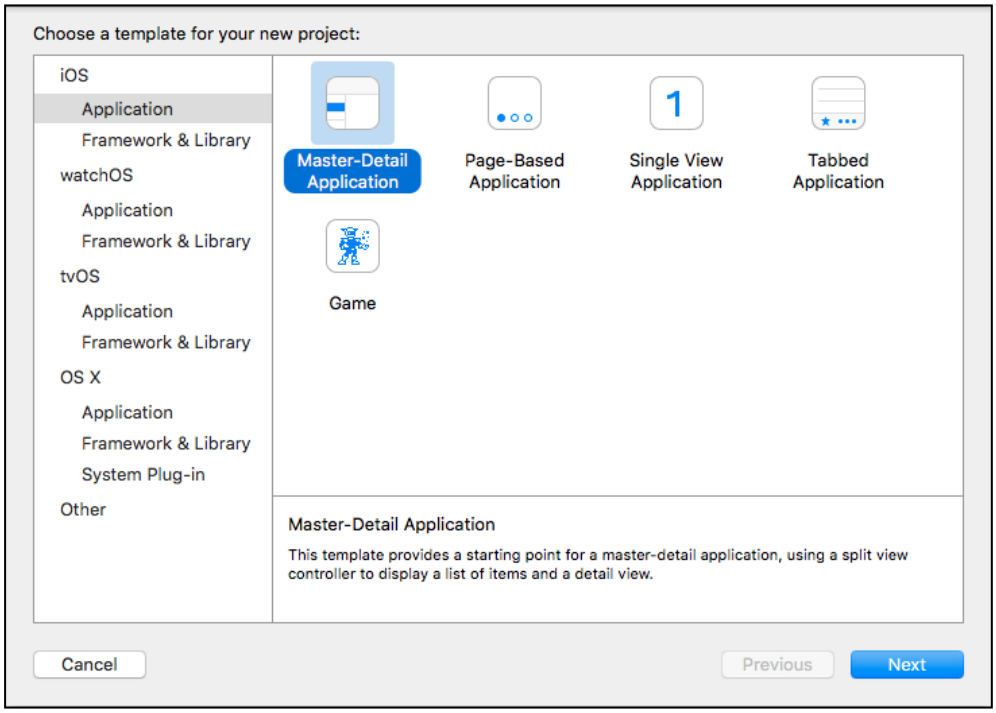


Abbildung 296 Master-Detail-Application

Dieses Framework liefert einen ansprechenden Rahmen für den TabellenView. Es hat vorgefertigt FÜNF grafische Hauptkomponten:

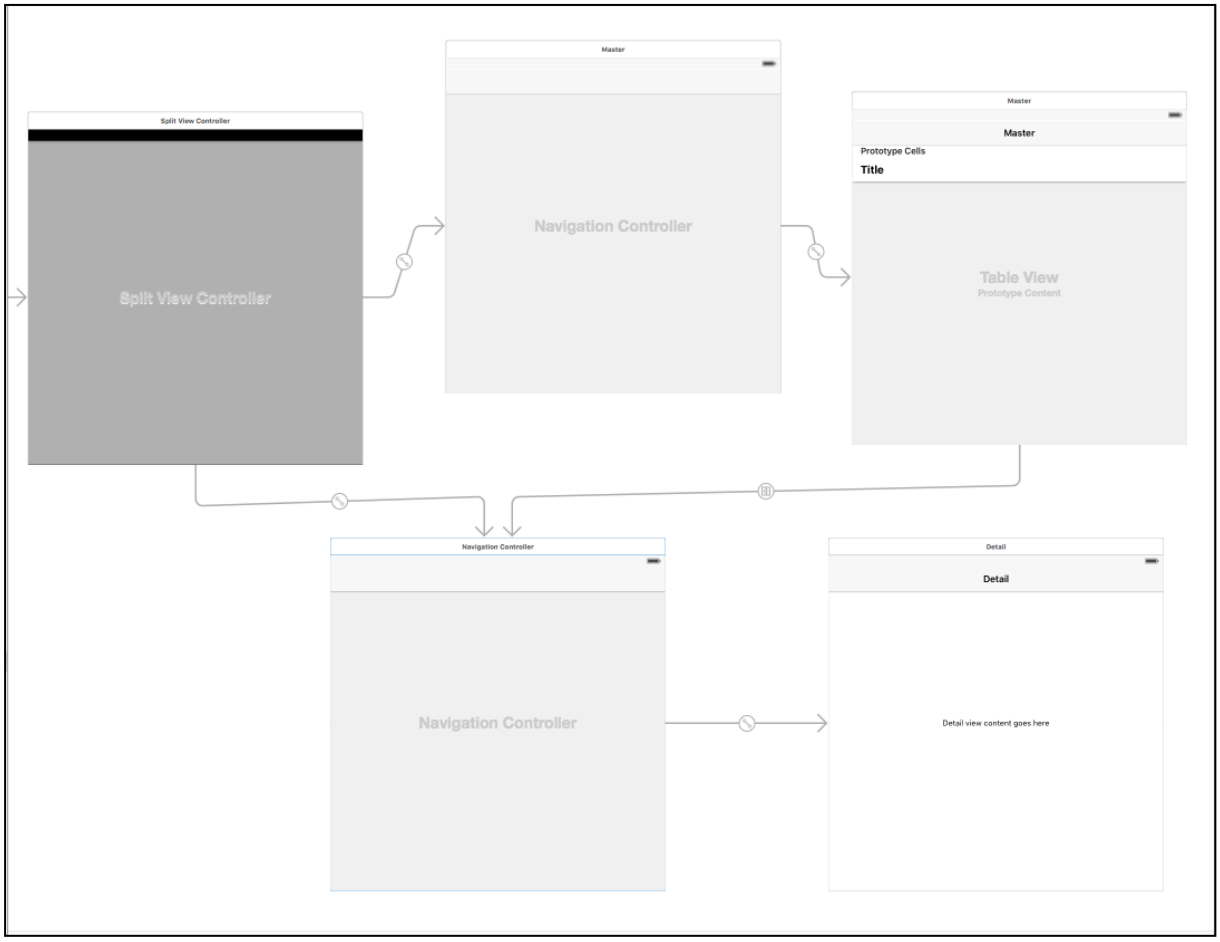


Abbildung 297 Komponenten einer Master-Detail-Application

Folgende Hauptkomponten existieren:

- Split-View-Controller
- Navigations-Controller
- **Table View**
- Navigations-Controller
- **Details View-Controller**

Für die Programmierung sind aber nur zwei Komponenten wichtig:

- **Table View**
  - Anzeige der „Menü“-Tabelleneinträge
- **Details View-Controller**
  - Anzeige der Details mit beliebigen Labeln, Editoren, Bildern etc.

Musterlösung:

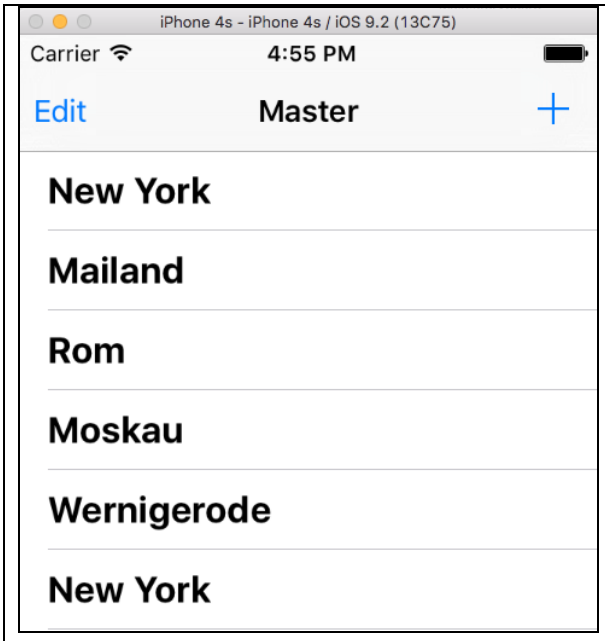


Abbildung 298 MenüView

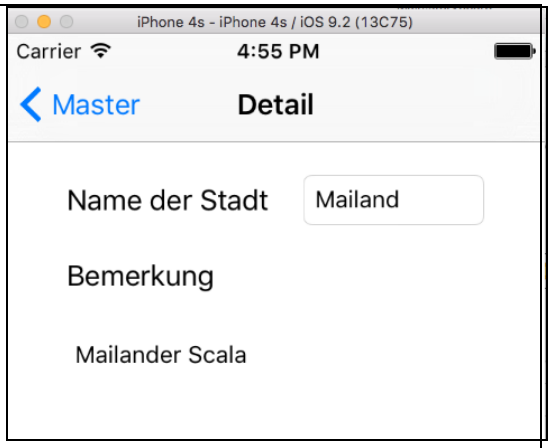


Abbildung 299 Anzeige der Details mit Editoren

Hinweis:

- Der Schalter „+“ erlaubt das Einfügen von neuen Daten

Erstellen des fertigen Projektes:

1. Anlegen einer neuen CocoaTouch Class

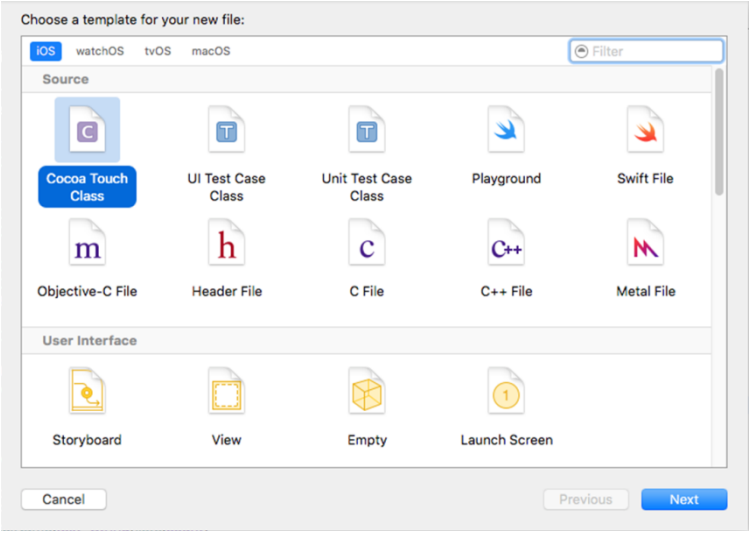
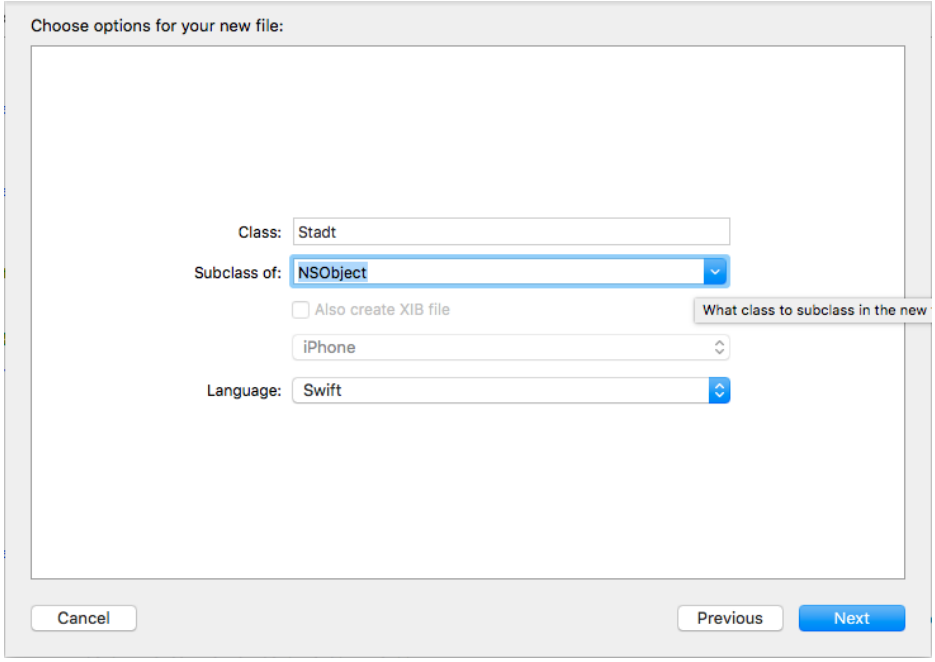


Abbildung 300 Neuer Eintrag "Cocoa Touch Class"

2. **Basis-Klasse auswählen: NSObject**



3. **Speichern unter dem Klassennamen, z. B. Stadt**

4. **Eintragen der Klassenstruktur:**

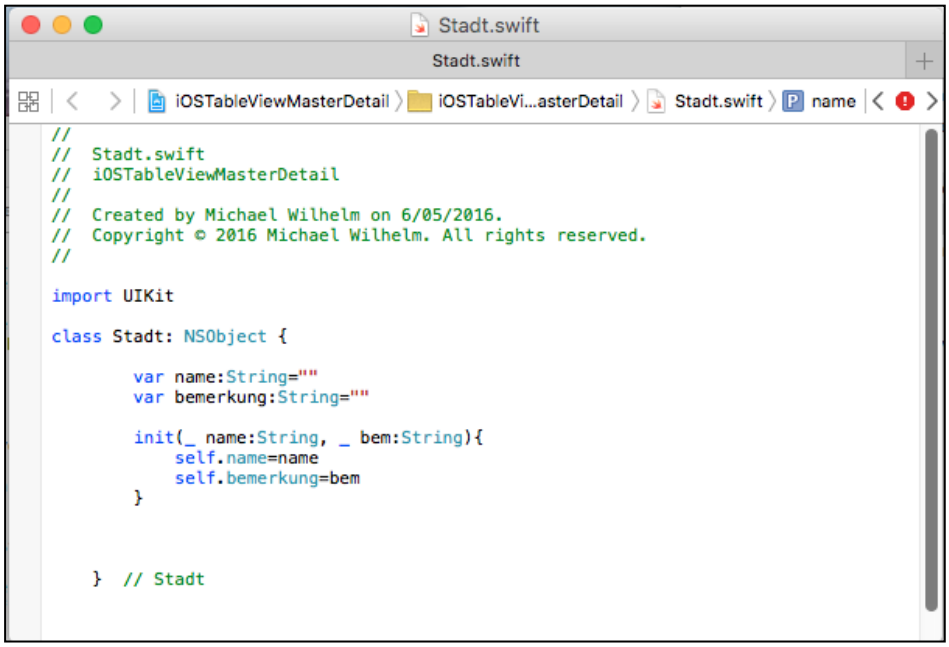


Abbildung 301 Projekt-Klasse „Stadt“

Damit kann man die obige Klasse im gesamten Projekt benutzen. Im nächsten Schritt wird der Maste-View so verändert, dass er mehrere Städte in einer Tabelle anzeigen kann.

5. Modifizieren des MasterViewController

Eintragen der Klassenglobalen variablen:

```
var staedte = [Stadt]()          // Liste der Städte
var neueStadtNr=0
```

Eintragen der Städte in die Liste:

```
override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after ...
        cities.append( City("New York", "Manhattan")  )
        cities.append( City("Mailand", "Mailander Scala")  )
        cities.append( City("Rom",  "Vatikan, Michelangelo")  )
        cities.append( City("Moskau", "Roter Platz")  )
        cities.append( City("Wernigerode", "HS Harz")  )
        cities.append( City("New York", "Manhattan")  )
```

Ändern von „object“ nach „staedte“:

```
var objects = [AnyObject]()  // Hauptarray  löschen

func insertNewObject(sender: AnyObject) {
    neueStadtNr++
    let stadt:String = "Stadt"+String(neueStadtNr)
    cities.append( City(stadt, stadt+stadt ) )

    ...
}

override func prepareForSegue(segue: UIStoryboardSegue,
        sender: AnyObject?) {
    if segue.identifier == "showDetail" {
      if let indexPath =
            self.tableView.indexPathForSelectedRow {
      let stadt:Stadt = staedte[indexPath.row]  // as! Stadt
      let detailviewcontroller =
         (segue.destinationViewController as!
             UINavigationController).topViewController as!
                 DetailViewController
            detailviewcontroller.detailStadt = stadt
        ...
         }
    }
}
```

Bitte beachten Sie, dass die “Übergabe-Variable” hier schon verändert wurde.

```
  detailviewcontroller.detailStadt = stadt
```

statt

```
  controller.detailItem = object
```

```
override func tableView(tableView: UITableView,
        numberOfRowsInSection section: Int) -> Int {
```

```
        return cities.count
    }

    override func tableView(tableView: UITableView,
         cellForRowAtIndexPath indexPath: NSIndexPath)
            -> UITableViewCell {
    let cell = tableView.dequeueReusableCellWithIdentifier(
                "Cell", forIndexPath: indexPath)
    let stadt:Stadt = staedte[indexPath.row] // as! NSDate
    cell.textLabel!.text = stadt.name
    return cell
  }

  override func tableView(tableView: UITableView,
      commitEditingStyle editingStyle:
      UITableViewCellEditingStyle, forRowAtIndexPath
          indexPath: NSIndexPath) {

    if editingStyle == .Delete {
       cities.removeAtIndex(indexPath.row)
       tableView.deleteRowsAtIndexPaths([indexPath],
                 withRowAnimation: .Fade)
    }
    else if editingStyle == .Insert {
    }
  }
```

Damit sind alle Änderungen im Master-View angeschlossen. Es fehlt nur noch die Änderung in den Details-Controller.

6. Modifizieren des DetailsViewController

a) Löschen des Labels

b) Eintragen von zwei Label
Eintragen eines Textfield
Eintragen eines TextView (Multi-Line-Editor)

c) Eintragen von Constraints

Label: „Name der Stadt“

- Top: 10 Pixel
- Left: 10 Pixel
- Right: 10 Pixel
- Bottom: 10 Pixel
- Ausrichtung: zentriert

TextField: Name der Stadt aus der Instanz

- Top: 10 Pixel
- Left: 10 Pixel

- Right: 10 Pixel
- Bottom: 10 Pixel

Label: Bemerkung

- Top: 10 Pixel
- Left: 10 Pixel
- Bottom: 10 Pixel

TextView: Bemerkung der Stadt aus der Instanz

- Top: 10 Pixel
- Left: 10 Pixel
- Right: 10 Pixel
- Bottom: 10 Pixel

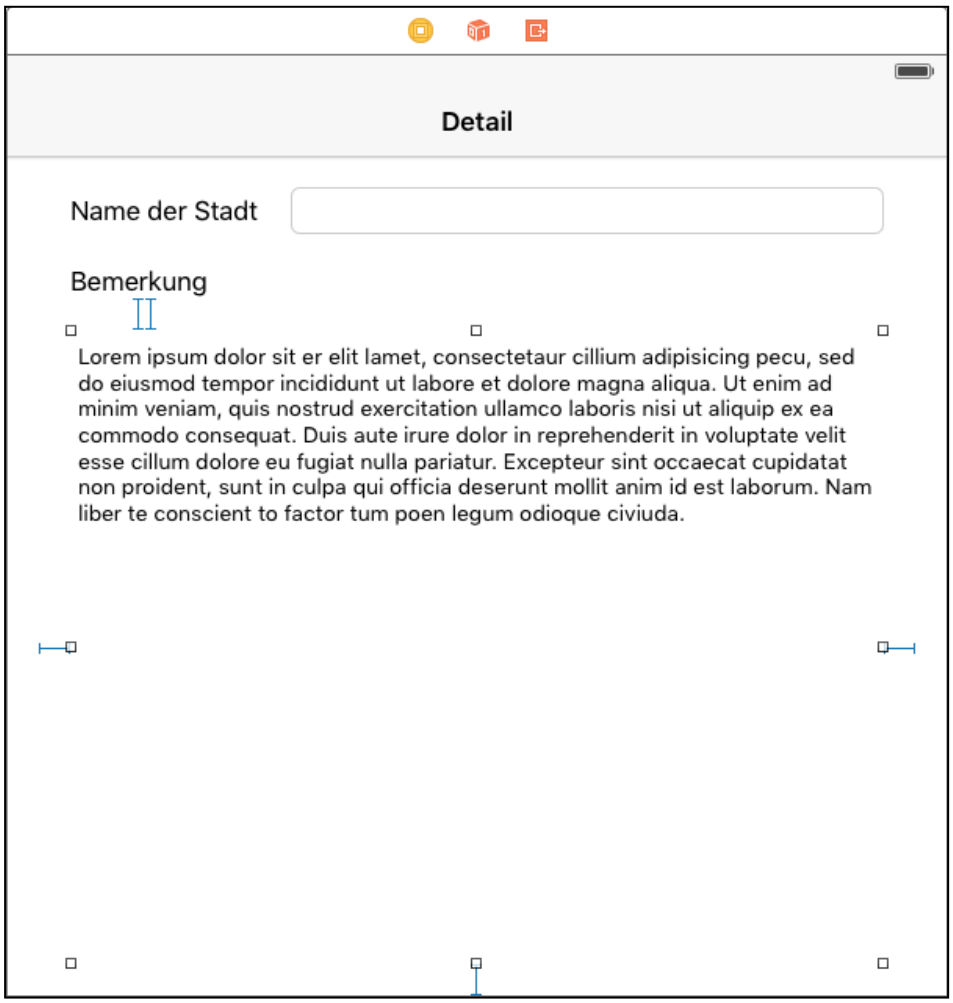


Abbildung 302 Constraints der DetailsViewController

7. Erstellen der beiden Outlets per Drag & Drop

```
@IBOutlet var tStadtName: UITextField!
@IBOutlet var tStadtBemerkung: UITextView!
```

8. Ändern der Anzeige-Programmierung (uiElemente, Klasse Stadt):

```
var detailStadt:Stadt? = nil {
        didSet {
            // Update the view.
            //self.configureView()  // wenn aktiv, Absturz
        }
    }

    func configureView() {
        // Update the user interface for the detail item.
        if let stadt:Stadt = self.detailStadt {
            tStadtName.text = stadt.name
            tStadtBemerkung.text = stadt.bemerkung
        }
    }

    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup ...
        // Testanweisungen
        //tStadtName.text = "stadt.name"
        //tStadtBemerkung.text = "stadt.bemerkung"
        self.configureView()
    }
```

14 Buttons

14.1 Buttons mit Symbolen

Jeder Button kann man einem Bild versehen werden. Dazu müssen die Bilder im Ordner „Images.xcassets“ eintragen werden.

Ablauf:

- Öffnen Sie das Fenster für den Eintrag „Assets.xcassets“.
- Öffnen Sie den Finder und den Ordner, in dem die Bilder eingetragen sind.
  - **Die Bilder müssen können beliebig farbig sein.**
  - **Es sollten drei Bilder vorhanden sein:**
    - 25×25 Pixel
    - 50×50 Pixel
    - 75×75 Pixel
- Ziehen Sie das **erste** Bild in den Assets.xcassets bzw. Images.xcassets.. Der linke Bereich in der oberen Abbildung.
- Ziehen Sie die beiden restlichen Bilder in den Bereichen 2x und 3x.
- Damit können im Eigenschaftsdialog der Schalter die Bilder auswählen.

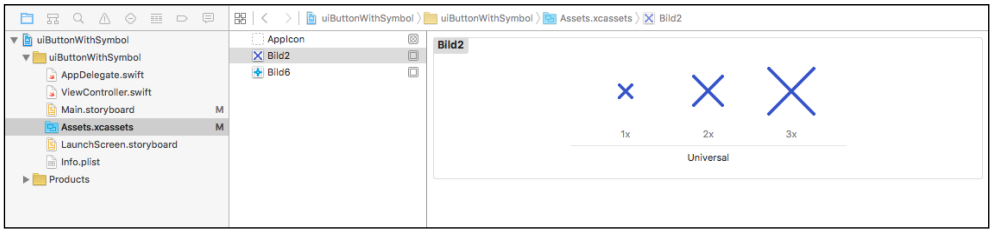


Abbildung 303 Anzeige der Symbole in den „Images-Ordner“ Assets.xcassts

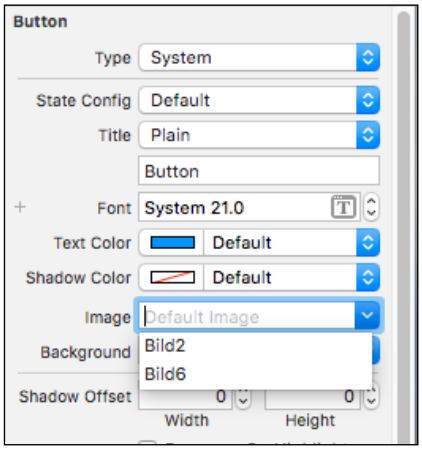


Abbildung 304 Auswahl eines Symbols in einem Button

Beachten Sie, dass die Texte nur sichtbar sind, wenn man ihnen eine separate Farbe gibt.

It’s not a error, …

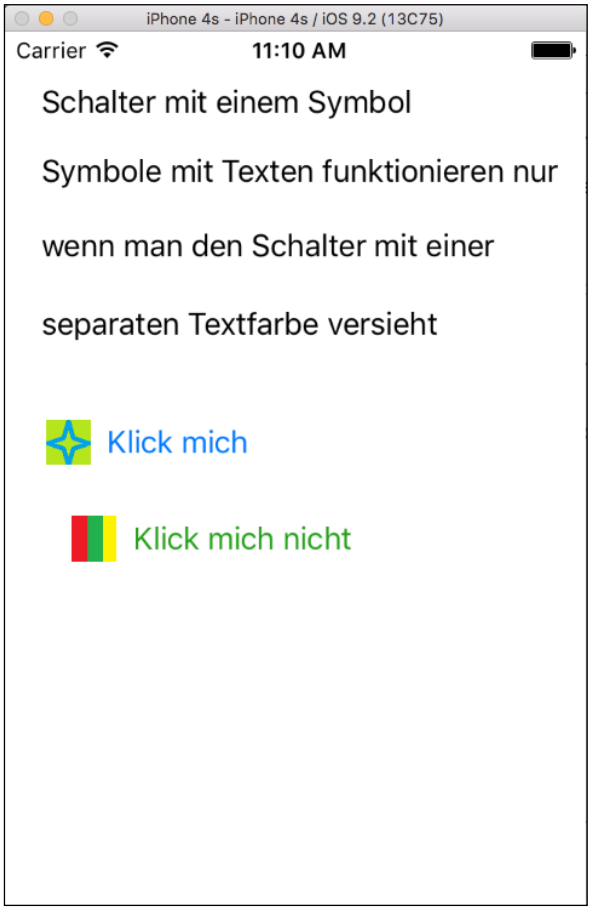


Abbildung 305 ViewController mit Bilder in Schaltern

14.2 Hintergrundfarbe von Buttons

```
button.backgroundColor = UIColor(red: 0.4, green: 1.0, blue: 0.2, alpha: 0.5)

button.backgroundColor = .white
button.backgroundColor = .red
button.backgroundColor = .green
button.backgroundColor = .blue


// siehe Farb-Asset
   bn.view.backgroundColor = UIColor(named: "myColor")
```

15 uiImageView

Die Anzeige von Bildern wird über die Komponente uiImageView realisiert.

15.1 „Assets.xcassets“

Die hier beschriebene Variante zeigt Bilder, die im Ordner „Assets.xcassets“ bzw. Images.xcassets. eingetragen wurden.

Ablauf:

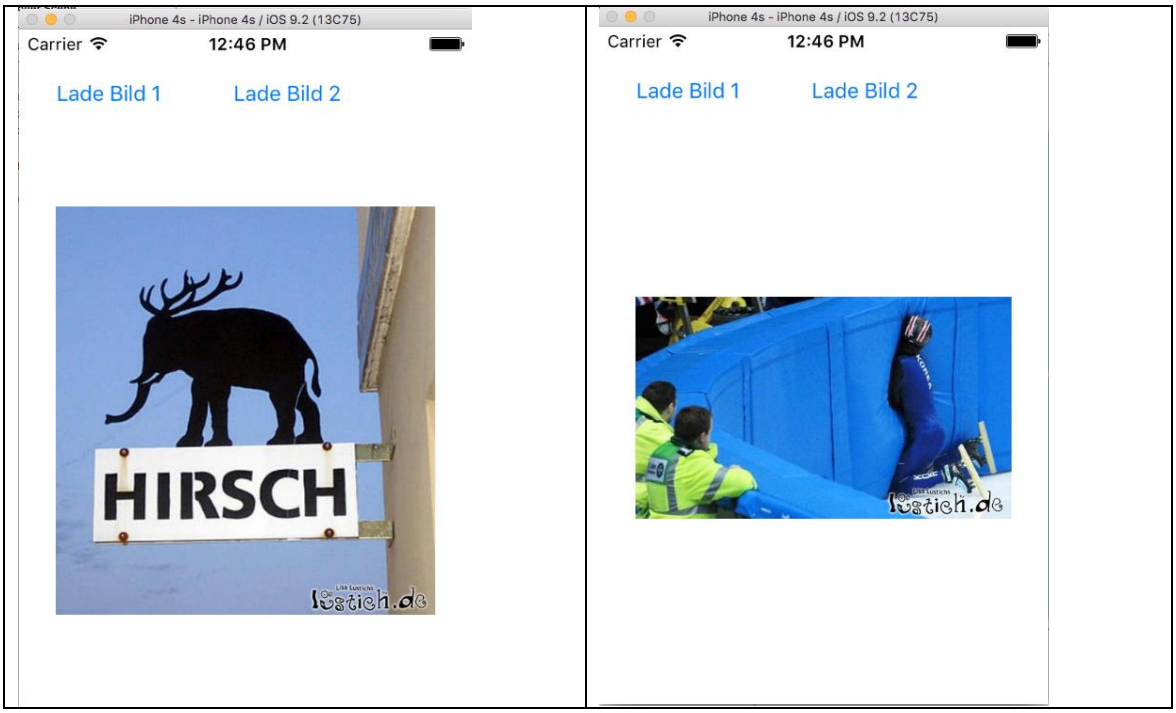
- Öffnen Sie das Fenster für den Eintrag „Assets.xcassets“ bzw. Images.xcassets.
- Öffnen Sie den Finder und den Ordner, in dem die Bilder eingetragen sind.
    - **Die Bilder müssen können beliebig farbig sein.**

- Ziehen Sie die Bilder in den Assets.xcassets bzw. Images.xcassets..
- Die Bilder werden über den Namen im Katalog adressiert werden. Also nicht über den Dateinamen.

Weitere Ablauf:

- Ziehen Sie einen uiImageView auf den ViewController.
- Setzen Sie die Eigenschaft der Darstellung des Bildes auf „Aspect Fit“.
- Die uiImageView-Komponente muss als Modus “Aspect to Fit” haben. Ansonsten wird das Bild verzerrt skaliert.

Abbildung 306 Aspect to Fit



Das Layout dieser „App“ ist etwas kompliziert, da die Labels ohne StackView eingetragen wurden.

- Label „lade Bild 1“
  - Top: 10 Pixel
  - Left: 10 Pixel
  - Right: 50 Pixel
  - Bottom: 50 Pixel
- Label „lade Bild 2“
  - Top: 10 Pixel
  - Left: 50 Pixel
  - Right: 10 Pixel
  - Bottom: 50 Pixel
- ImageView
  - Top: 50 Pixel **zum Top Layout Guide**
  - Left: 10 Pixel
  - Right: 10 Pixel
  - Bottom: 10 Pixel

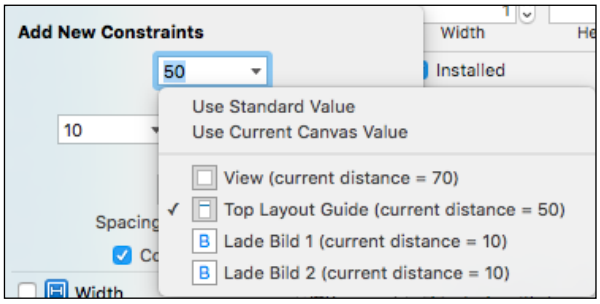


Abbildung 307 Constraints zum Top Layout Guide

16 Core-Daten

16.1 NSUserDefaults

- Ein NSUserDefaults speichert die Daten ähnlich einer Ini-Datei.
- Gespeichert werden Variablen mit einen Schlüssel/Key.
- Die Daten werden in der plist-Datei (properties) gespeichert.

Beispiele zum Speichern:

```
let nr:Int=41
let defaults=NSUserDefaults.standardUserDefaults()
defaults.setObject(nr,forKey: "nr")
defaults.setInteger(nr,forKey: "nr")
let str:String="Norman Bates"
let defaults=NSUserDefaults.standardUserDefaults()
defaults.setObject(str,forKey: "name")
```

Beispiel für das Laden:

```
var nr:Int=0
let defaults=NSUserDefaults.standardUserDefaults()
nr=defaults.integerForKey("nr")
var str:String=""
let defaults=NSUserDefaults.standardUserDefaults()
str=defaults.stringForKey("name")!
// a string can be nil !!!
```

Eine Zahl, Bool haben immer einen Wert. String oder Array sind „Optional-Werte“ (!)

Funktionen zum Laden:

- let nr:Int=defaults.integerForKey("nr")
- let b:Bool=defaults.boolForKey("withShadow")
- let myArray[Int]=defaults.arrayForKey("myArray")
- let value:Float=defaults.floatForKey("price")
- let value:Double=defaults.doubleForKey("price")
- let obj:AnyObject=defaults.objectForKey("custom")
- let strArray:[AnyObject]=defaults.stringArrayForKey("customs")

Funktionen zum Speichern:

- defaults.setObject(nr,forKey: "nr")
- defaults.setObject(str,forKey: "name")
- defaults.setBool(true,forKey: "withShadow")
- defaults.setInteger(1234,forKey: "age")
- defaults.setFloat(12.34,forKey: "price")
- defaults.setDouble(12.34,forKey: "price")
- defaults.setObject(myArray,forKey: "customs")
- defaults.setObject(myDictionary,forKey: "customs")

16.2 CoreData (database)

17 Grafik

Dieses Kapitel zeigt die Verwendung von einer eigener Grafikkomponente.

In Java erzeugt man eine eigene Klasse, die von JPanel abgeleitet wird. Danach überschreibt man die Methode paint. Der Parameter “Graphic g” ist die Referenz auf die „Leinwand“.

Beispiel einer Grafikprogrammierung in Java mit Swing:

```
class MyCanvas extends JPanel {

  public void paint (Graphics g) {
    int width = this.getWidth();
    int height = this.getHeight();
    g.drawLine(10,10,width-10,height-10);
    g.drawLine(10,40,10,100);
    g.drawLine(10,100,30,300);
    g.drawLine(30,300,300,50);
  }  // paint

}
```

Dieses Prinzip ist auch unter Swift realisiert. Nur etwas komplizierter. Der Grund liegt im Drag & Drop-Technik. Man benötigt eine eigene Grafikkomponente in der Komponenten-Liste. Da das nicht möglich, verwendet Swift eine andere Technik und setzt die Oberklasse einer uiView-Komponente um. Damit wird eine uiView-Klasse auf eine eigene Swift-Klasse gesetzt.

Ablauf:

1) **Erstellen der Grafikkomponente:**
   Aufruf einer neuen Datei mit Cmd+N
   Anklicken der Cocoa Touch Class
   Schalter „Next“

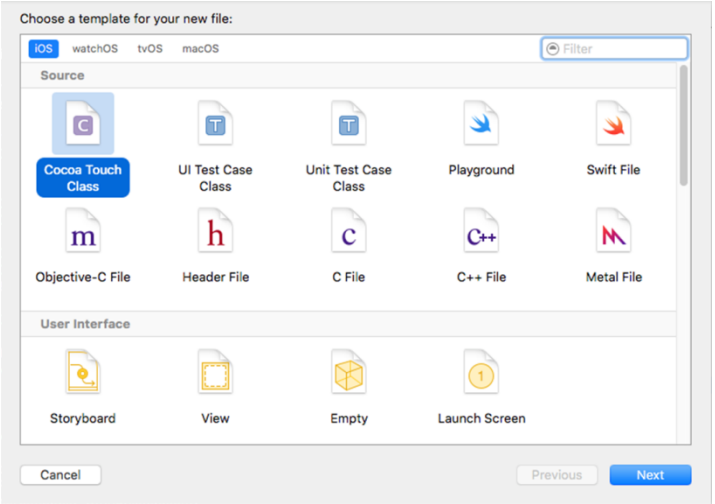


Abbildung 308 Erstellen der Grafik-Komponente

Nun die SubClass eintragen:

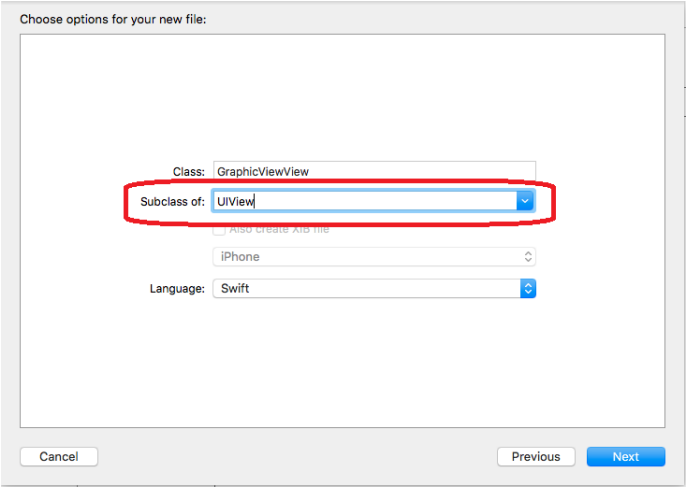


Abbildung 309 Eingabe der Sub-Class „uiView“

Die Klasse UIView ist völlig ausreichend für die Grafikprogrammierung.

Nun eine ganz normale uiView-Komponente per Drag & Drop in den View-Controller eintragen.
Die Verknüpfung mit der Subclass erfolgt der Einstellung:

- Öffnen des Identity Inspector
- Setzen der Custom-Class auf „GraphicViewView“

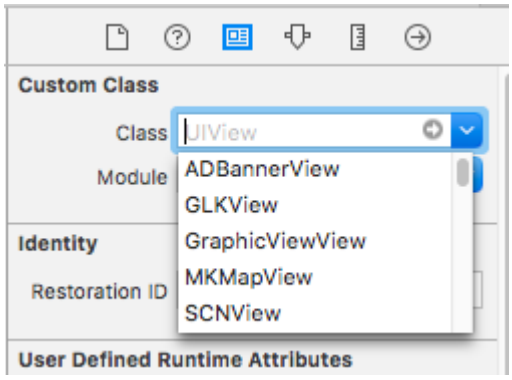


Abbildung 310 Setzen der Basisklasse auf GraphicViewView

Damit ist Verbindung zur Grafik-Komponente erfolgt. Setzen Sie nun den Modus der View-Komponente auf Redraw.

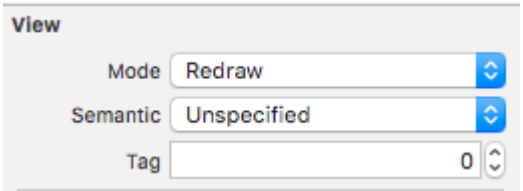


Abbildung 311 Modus auf Redraw setzen

Als letzten Schritt muss noch die Grafikprogrammierung eingetragen werden.

Alle Grafiken müssen im einem Block programmiert werde, da sonst die unterschiedlichen Farben, Stricharten und Linienbreiten nicht unterschieden werden können:

- Optionen setzen
- Grafikfunktionen
- Aufruf von: **CGContextStrokePath(context)**
- Optionen setzen
- Grafikfunktionen
- Aufruf von: **CGContextStrokePath(context)**

Quellcode der Grafikmethode:

```
//
//  GraphicViewView.swift
import UIKit

class GraphicViewView: UIView {


    // Only override drawRect: if you perform custom drawing.
    override func drawRect(rect: CGRect) {
        let context = UIGraphicsGetCurrentContext()
        let red = UIColor.redColor().CGColor
        CGContextSetLineWidth(context, 3.0)
        CGContextSetStrokeColorWithColor(context, red)
        CGContextMoveToPoint(context, 10.0, 10.0)
        CGContextAddLineToPoint(context, 200.0, 500.0)
        CGContextStrokePath(context)
    }
}
```

Erläuterung:

Holen der "Leinwand":
let context = UIGraphicsGetCurrentContext()

Holen einer roten Farbe:
let red = UIColor.redColor().CGColor

Setzen der Linienbreite:
CGContextSetLineWidth(context, 3.0)

Setzen der Linienfarbe, ginge auch direkt:
CGContextSetStrokeColorWithColor(context, red)

Linie zeichnen:
CGContextMoveToPoint(context, 10.0, 10.0)
CGContextAddLineToPoint(context, 200.0, 500.0)
// 32 oder 64-Bit FloatingPoint, je nach Betriebssystem

Komplette Grafik aktualiseren:

```
CGContextStrokePath(context)
```

Das Ergebnis ist in der unteren Abbildung zu sehen. Da die App zu klein ist, wird die Linie abgeschnitten.

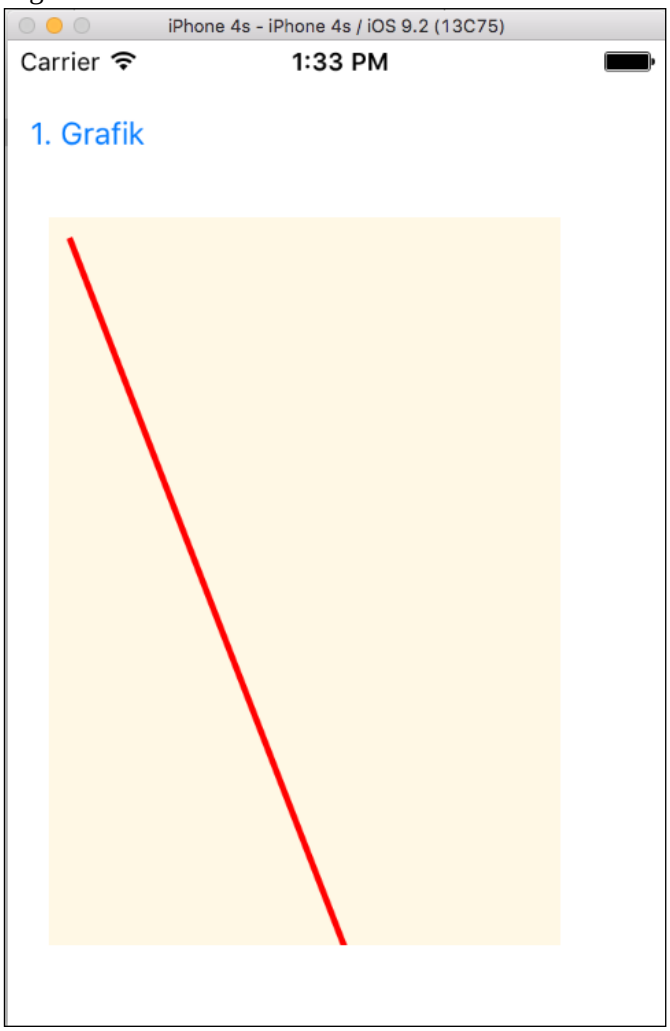


Abbildung 312 Anzeige der Grafikprogrammierung

Hinweis:

- Der Hintergrund wurde per Attribut-Inspector eingetragen.

17.1 Varianten

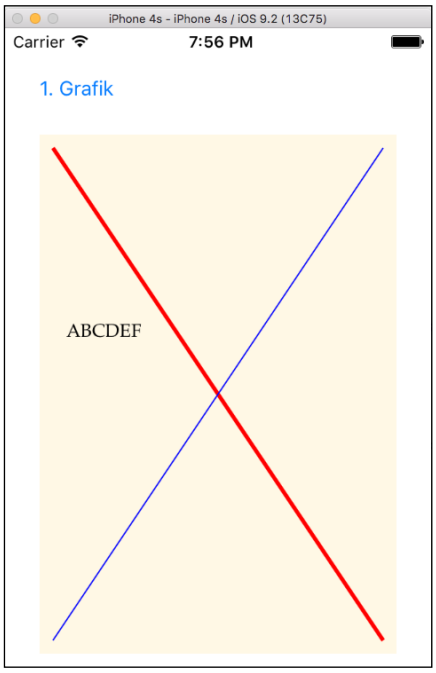


Abbildung 313 Test2, welches die aktuellen Abmessungen benutzt

```
func test2(rect: CGRect){
   let context = UIGraphicsGetCurrentContext()
   let red = UIColor.redColor().CGColor
   let blue = UIColor.blueColor().CGColor

   CGContextSetLineWidth(context, 3.0)
   CGContextSetStrokeColorWithColor(context, red)
   CGContextMoveToPoint(context, 10, 10)
   CGContextAddLineToPoint(context, rect.width-10, rect.height-10)
   CGContextStrokePath(context)  // schließt die Grafik ab

   CGContextSetLineWidth(context, 1.0)
   CGContextSetStrokeColorWithColor(context, blue)
   CGContextMoveToPoint(context, rect.width-10, 10)
   CGContextAddLineToPoint(context, 10, rect.height-10)
   CGContextStrokePath(context)  // schließt die Grafik ab


     // DrawText
     // Hashtable, dict   String:AnyObject
   let attrsDictionary =
     [NSFontAttributeName:font!,
     NSBaselineOffsetAttributeName:baselineAdjust]

   CGContextSetStrokeColorWithColor(
        context, UIColor.greenColor().CGColor)
```

```
        // zeichnen des Textes
      s.drawInRect(
          CGRectMake(20.0, 140.0, 300.0, 48.0),
          withAttributes: attrsDictionary)
    CGContextStrokePath(context)  // schließt die Grafik ab

  }  // Test2
```

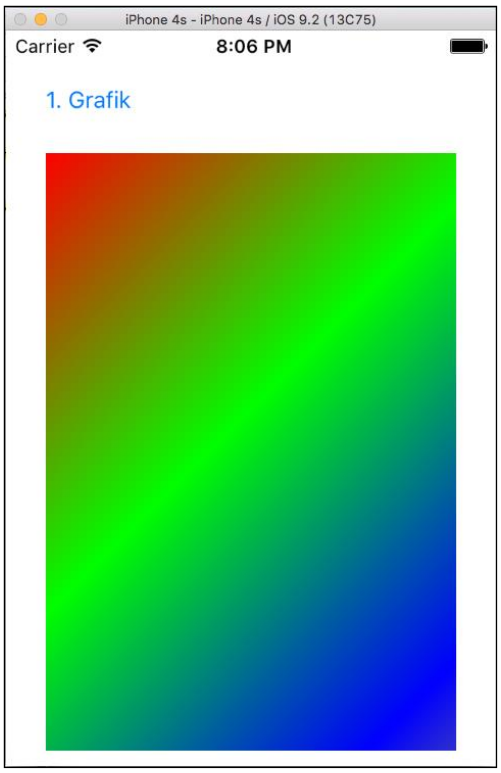


Abbildung 314 Farbgradient test3

```
  func test3(rect: CGRect) {
     let context = UIGraphicsGetCurrentContext()
     let locations: [CGFloat] = [ 0.0, 0.25, 0.5, 0.75 ]
     let colors = [
            UIColor.redColor().CGColor,
            UIColor.greenColor().CGColor,
            UIColor.blueColor().CGColor,
            UIColor.yellowColor().CGColor
        ]

      let colorspace = CGColorSpaceCreateDeviceRGB()
      let gradient = CGGradientCreateWithColors(
            colorspace, colors, locations)

      var startPoint = CGPoint()
      var endPoint =  CGPoint()
```

```
    startPoint.x = 0.0
    startPoint.y = 0.0
    endPoint.x = 600
    endPoint.y = 600

    CGContextDrawLinearGradient(
       context,
       gradient,
       startPoint,
       endpoint
     , CGGradientDrawingOptions.DrawsAfterEndLocation)
 }  // test3
```

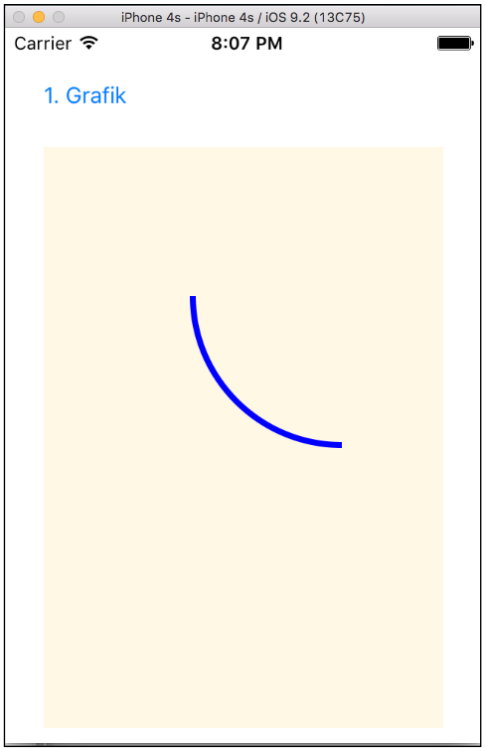


Abbildung 315 Bogen, Arc, Test4,

```
func test4(rect:CGRect) {
   let context = UIGraphicsGetCurrentContext()
   CGContextSetLineWidth(context, 4.0)
   CGContextSetStrokeColorWithColor(
      context,UIColor.blueColor().CGColor)

   CGContextMoveToPoint(context, 100, 100)
   CGContextAddArcToPoint(context, 100,200, 300,200, 100)
   CGContextStrokePath(context)  // schließt die Grafik ab
}  // Test4
```

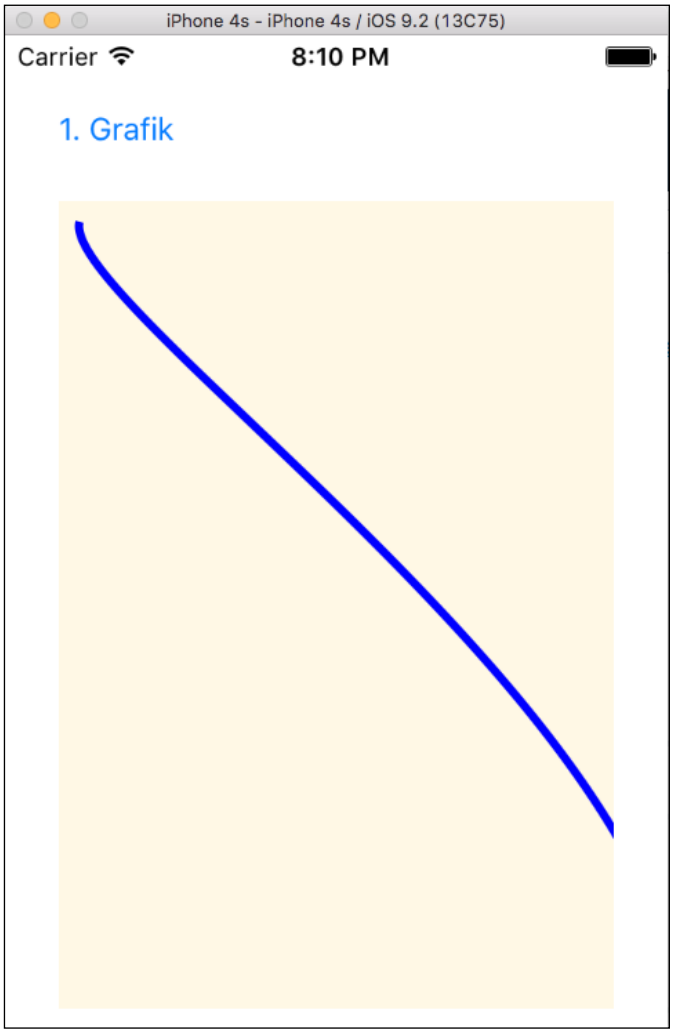


Abbildung 316 Bezier, Test5

```
func test5(rect: CGRect)    {
   let context = UIGraphicsGetCurrentContext()
   CGContextSetLineWidth(context, 4.0)
   CGContextSetStrokeColorWithColor(
      context, UIColor.blueColor().CGColor)

   CGContextMoveToPoint(context, 10, 10)
   CGContextAddCurveToPoint(context, 0, 50, 300, 250, 300, 400)
   CGContextStrokePath(context)  // schließt die Grafik ab
} // test5
```

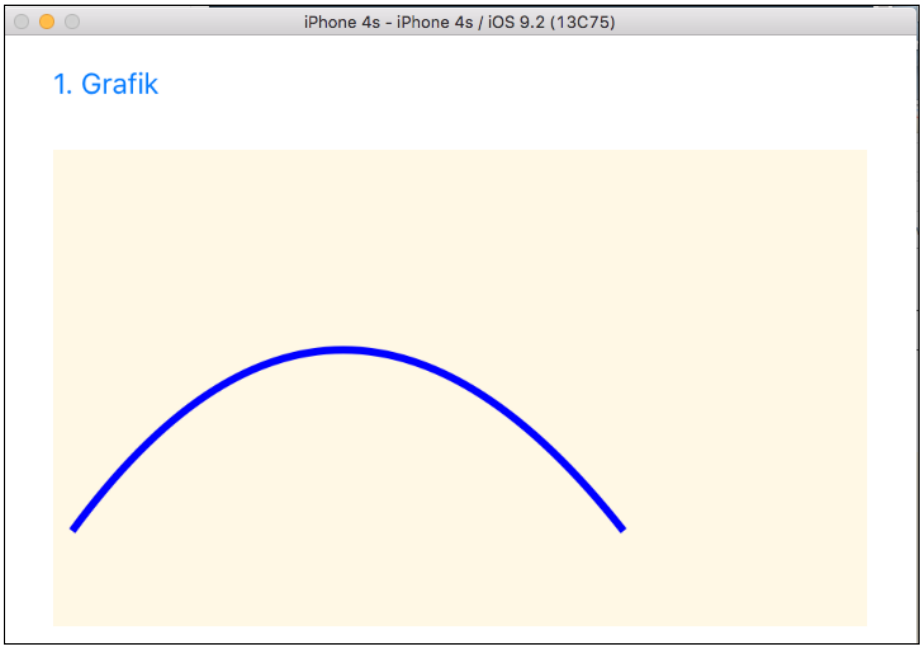


Abbildung 317 Bezier, Test6

```
func test6(rect: CGRect)    {
   let context = UIGraphicsGetCurrentContext()
   CGContextSetLineWidth(context, 4.0)
   CGContextSetStrokeColorWithColor(
      context, UIColor.blueColor().CGColor)
   CGContextMoveToPoint(context, 10, 200)
   CGContextAddQuadCurveToPoint(context, 150, 10, 300, 200)
   CGContextStrokePath(context)  // schließt die Grafik ab
}  // Test6
```

17.2 Grafik-Methoden

func UIRectFill(_ rect: CGRect)
Fills the specified rectangle with the current color.

func UIRectFillUsingBlendMode(_ rect: CGRect, _ blendMode: CGBlendMode)
Fills a rectangle with the current fill color using the specified blend mode.

func UIRectFrame(_ rect: CGRect)
Draws a frame around the inside of the specified rectangle.

func CGContextSetFont(_ c: CGContext?, _ font: CGFont?)
Sets the platform font in a graphics context.

func CGContextSetFontSize(_ c: CGContext?, _ size: CGFloat)
Sets the current font size.

func CGContextSetTextDrawingMode(_ c: CGContext?, _ mode: CGTextDrawingMode)
Sets the current text drawing mode.
Modi:
- CGBlendMode
- CGInterpolationQuality
- CGTextDrawingMode
- CGTextEncoding

func CGContextSetTextPosition(_ c: CGContext?, _ x: CGFloat, _ y: CGFloat)
Sets the location at which text is drawn.

func CGContextClearRect(_ c: CGContext?, _ rect: CGRect)
Paints a transparent rectangle.

func CGContextDrawPath(_ c: CGContext?, _ mode: CGPathDrawingMode)
Draws the current path using the provided drawing mode.

func CGContextStrokeRectWithWidth(_ c: CGContext?, _ rect: CGRect, _ width: CGFloat)
Paints a rectangular path, using the specified line width.

func CGContextStrokeRect(_ c: CGContext?, _ rect: CGRect)
Paints a rectangular path.

func CGContextStrokeLineSegments(
_ c: CGContext?, _ points: UnsafePointer<CGPoint>, _ count: Int)
Strokes a sequence of line segments.

points:
- An array of points, organized as pairs—the starting point of a line segment followed by the ending point of a line segment. For example, the first point in the array specifies the starting position of the first line, the second point specifies the ending position of the first line, the third point specifies the starting position of the second line, and so forth.

count
- The number of points in the points array.

Beispiel:
```
CGContextBeginPath (context);
for (k = 0; k < count; k += 2) {
    CGContextMoveToPoint(context, s[k].x, s[k].y);
    CGContextAddLineToPoint(context, s[k+1].x, s[k+1].y);
}
CGContextStrokePath(context);
```

func CGContextStrokeEllipseInRect(_ c: CGContext?, _ rect: CGRect)
Strokes an ellipse that fits inside the specified rectangle.

18 Sensoren

18.1 Kamera

Protokolle:

- `UINavigationControllerDelegate`
- `UIImagePickerControllerDelegate`

Globale Variable:

```
@IBOutlet  var uiImageView: UIImageView!
var  uiImagePickerController: UIImagePickerController!
```

Quellcode zum Auslösen der Kamera (Button_Event):

```
@IBAction func bnTakePictureClick(sender: UIButton) {
   uiImagePickerController = UIImagePickerController()
   uiImagePickerController.delegate = self
   uiImagePickerController.sourceType =
        UIImagePickerControllerSourceType.Camera

   presentViewController(
          uiImagePickerController,
          animated: true,
          completion: nil)       // insert Codeblock 2 the image
  }
```

Auflistung des UIImagePickerControllerSourceType

- Camera
- PhotoLibrary
- SavedPhotosAlbum

Hinweis:

- Die uiImageView-Komponente muss als Modus “Aspect to Fit” haben. Ansonsten wird das Bild verzerrt skaliert.

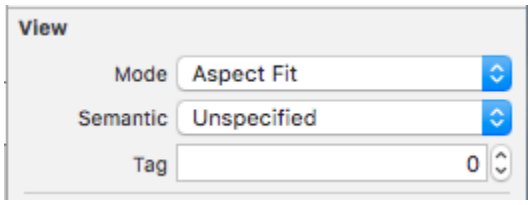


Abbildung 318 Aspect to Fit

Quellcode zum Anzeigen des Bildes:

```
   func imagePickerController(
       picker: UIImagePickerController!,
       didFinishPickingImage image: UIImage!,
       editingInfo: NSDictionary!) {
```

```
    let selectedImage : UIImage = image
    imageView.image=selectedImage
    self.dismissViewControllerAnimated(true, completion: nil)
  }
```

Abbildung 319 Kamera-App mit einem Bild, welches beim Start geladen wurde

18.2 Mikrofon

Import:

```
import AVFoundation
import MediaPlayer
```

Delegate:

- AVAudioPlayerDelegate
- AVAudioRecorderDelegate

```
class ViewController:UIViewController,
        AVAudioPlayerDelegate,
        AVAudioRecorderDelegate  {

  // globale Variablen
    var avAudioPlayer : AVAudioPlayer?
    var avAudioRecorder : AVAudioRecorder?

    // Schalter fuer das Deaktivieren
    @IBOutlet var bnStartRecord: UIButton!
    @IBOutlet var bnStop: UIButton!
    @IBOutlet var bnPlay: UIButton!

    @IBOutlet var labelStatus: UILabel!

    // uiEvents
    @IBAction func bnStartRecordClick(sender: UIButton) {
        if avAudioRecorder?.recording == false {
            bnStartRecord.enabled = false
            bnStop.enabled = true
            avAudioRecorder?.record()
            labelStatus.text = "Aufnahme läuft..."
        }  // if
    }
    @IBAction func bnStopClick(sender: UIButton) {
        bnStartRecord.enabled = true
        bnStop.enabled = false
        bnPlay.enabled = true
        if avAudioRecorder?.recording == true {
            avAudioRecorder?.stop()
            labelStatus.text = "Aufnahme beendet..."
        }
        else {
            avAudioPlayer?.stop()
            labelStatus.text = "Wiedergabe beendet..."
        }
    }


      // benutzt eine anonyme Methode
```

```
@IBAction func bnPlayClick(sender: UIButton) {
    if avAudioRecorder?.recording == false {
        bnStartRecord.enabled = false
        bnStop.enabled = true
        bnPlay.enabled = false
        labelStatus.text = "Wiedergabe läuft..."
        do {
            avAudioPlayer = try AVAudioPlayer(
                contentsOfURL: (avAudioRecorder?.url)!)
            avAudioPlayer?.play()
        }
        catch {
            print("Fehler bei der Wiedergabe")
        }
    }
}


private func getPathFilename() -> String {
    let paths = NSSearchPathForDirectoriesInDomains(
         .DocumentDirectory, .UserDomainMask, true)
    if let path = paths.first {
        return path+"/sound.caf"
    }
    else {
        return ""
    }
}


override func viewDidLoad() {
    super.viewDidLoad()
    bnStartRecord.enabled = true
    bnStop.enabled = false
    bnPlay.enabled = false

    avAudioPlayer?.delegate = self
    avAudioRecorder?.delegate = self

    let soundFilePath = getPathFilename()
    let soundFileURL = NSURL(fileURLWithPath: soundFilePath)
    let recordSettings =
    [AVEncoderAudioQualityKey: AVAudioQuality.Min.rawValue,
        AVEncoderBitRateKey: 16,
        AVNumberOfChannelsKey: 2,
        AVSampleRateKey: 44100.0]




    let audioSession = AVAudioSession.sharedInstance()
```

```
        do {
            try
                audioSession.setCategory(
                      AVAudioSessionCategoryPlayAndRecord)
        } catch {
            print("Fehler bei der Initialisierung")
        }

        do {
            try
                avAudioRecorder = AVAudioRecorder(
                    URL: soundFileURL,
                    settings: recordSettings as!
                                          [String: AnyObject])
            avAudioRecorder?.prepareToRecord()
        } catch {
            print("Fehler bei der Initialisierung(2)")
        }


    }  // Did Load

    // delegate Methoden
    func audioPlayerDidFinishPlaying(
                player: AVAudioPlayer!, successfully flag: Bool) {
         print("Wiedergabe beendet")
         labelStatus.text = "Wiedergabe beendet..."
    }

    func audioPlayerDecodeErrorDidOccur(
               player: AVAudioPlayer!, error: NSError!) {
        print("Ein Fehler ist während der Wiedergabe aufgetreten")
    }

    func audioRecorderDidFinishPlaying(
             player: AVAudioRecorder!, successfully flag: Bool) {
        print("Aufnahme beendet")
        labelStatus.text = " Aufnahme beendet..."
    }


    func audioRecorderDecodeErrorDidOccur(
            player: AVAudioRecorder!, error: NSError!) {
       print("Ein Fehler ist während der Aufnahme aufgetreten")
    }


}
```

18.3 Beschleunigungssensor

Die Sensoren bezüglich der Beschleunigung und der Lage sind sehr vielfältig. In diesem Kapitel wird ein einfaches Beispiel dargestellt. Auf meiner Homepage ist ein zweites komplettes Beispiel

Quellcode:

```
import UIKit
import CoreMotion             // neues Import-Modul


class ViewController: UIViewController {

    // uiAttribute
    @IBOutlet var tStatus: UITextView!

    // Verweis auf die Geraete
    var uiDynamicAnimator: UIDynamicAnimator? = nil
    let cmMotionManager = CMMotionManager()
    let nsOperationQueue = NSOperationQueue()


       // delegate-Methoden
    override func viewDidAppear(animated: Bool) {
       cmMotionManager.startDeviceMotionUpdatesToQueue(
          nsOperationQueue, withHandler: cmAccelerometerHandler)
    }

     // wird von den Geaeten aufgerufen
    func cmAccelerometerHandler(
            motion: CMDeviceMotion?, error: NSError?) {
        if error != nil {
            tStatus.text = "Error"
        }
        else {
            let gravity : CMAcceleration = motion!.gravity
            let x = CGFloat(gravity.x)
            let y = CGFloat(gravity.y)
            let z = CGFloat(gravity.z)
            tStatus.text = "x: " + String(x) + "\n" +
                    "y: " + String(y) + "\n" +
                    "z: " + String(z)
        }
    }

    override func viewWillDisappear(animated: Bool) {
        cmMotionManager.stopAccelerometerUpdates()
    }
```

```
    func initSensors() {
        uiDynamicAnimator = UIDynamicAnimator(
               referenceView: self.view)
    }

    override func viewDidLoad() {
        super.viewDidLoad()
        initSensors()
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can be recreated.
    }

}
```

Weitere Links:

- http://nshipster.com/cmdevicemotion/
- http://swiftdoc.org
- https://www.bignerdranch.com/blog/uidynamics-in-swift/

Einstellen des Intervalls:

- var deviceMotionUpdateInterval: NSTimeInterval
- manager.gyroUpdateInterval = 0.1

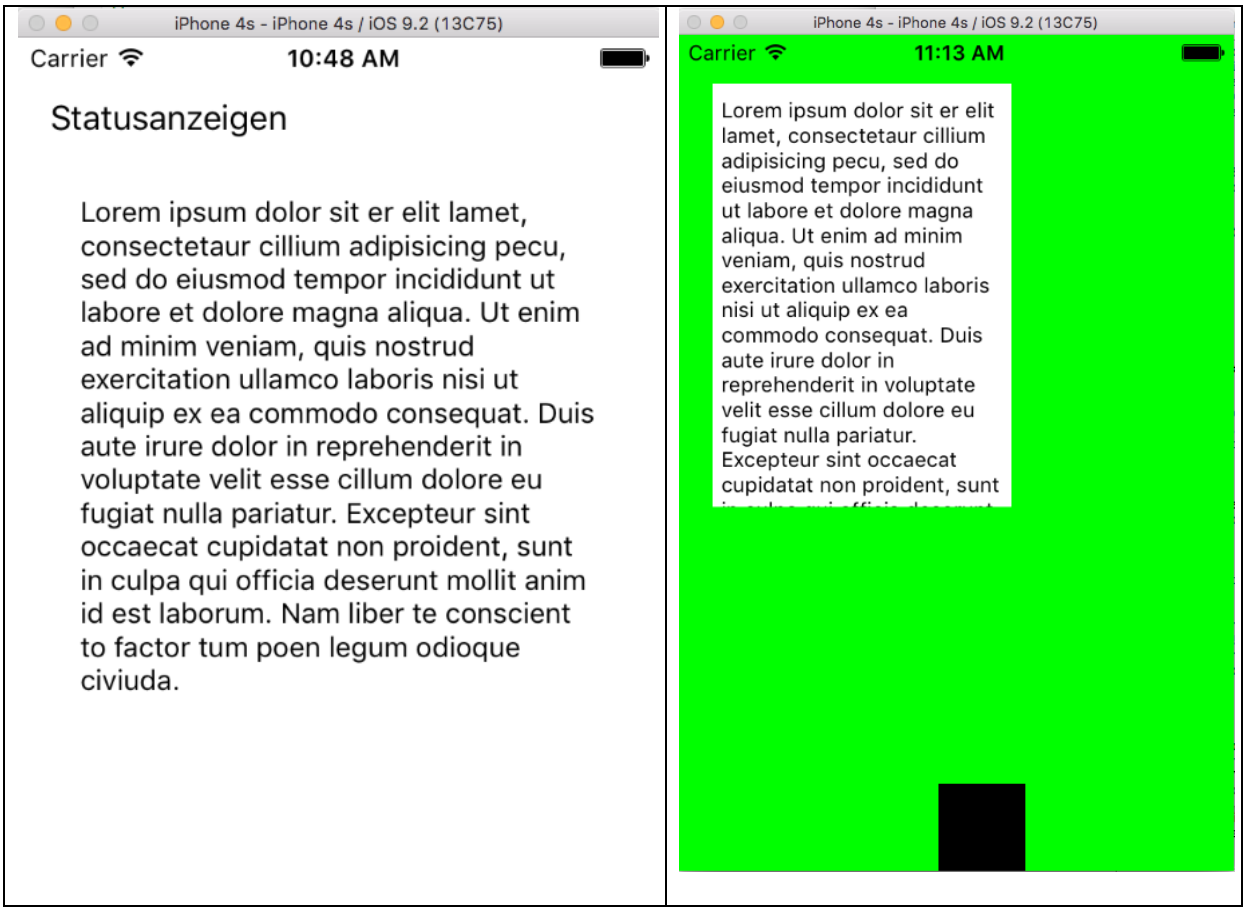


Abbildung 320 Hier sollten eigentliche Daten angezeigt werden, wurde noch nicht getestet.

Die rechte obere Abbildung zeigt das zweite Beispiel, hier wird das schwarze Viereck durch die Sensoren gesteuert.

18.4 GPS

Für die Benutzung des GPS-Sensors benötigt man zum einen das Framework „CoreLocation“ und einen Eintrag in der pList. Der Eintrag definiert, dass die App nachfragt, ob man den Sensor benutzen darf. Dieses Kapitel hat noch nicht geklappt.

18.4.1 Library einbinden

Um das Framework CoreLocation einzubinden, öffnen Sie Project-Navigator die Projektdatei. Dann öffnen Sie den Eintrag „General“. Nun scrollen Sie nach unten, bis zum Eintrag „Linked Framework and Libraries“ erscheint. Nun auf das Pluszeichen klicken.

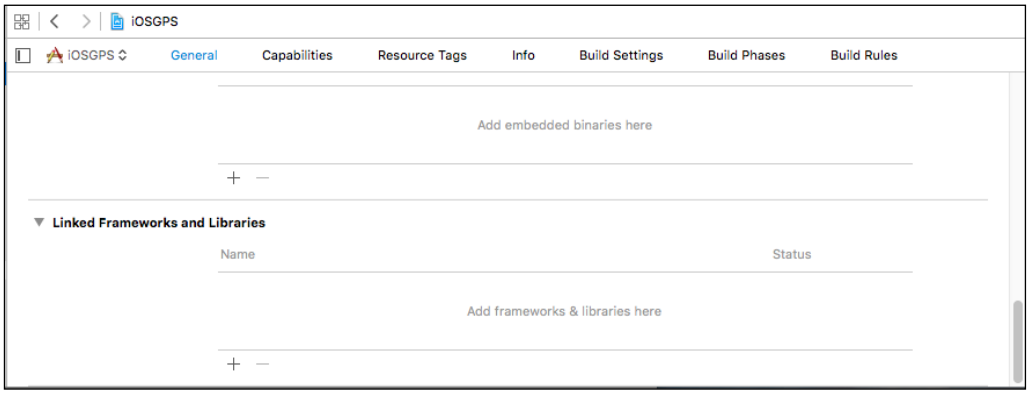


Abbildung 321 CoreLocationFramework hinzufügen, Teil 1

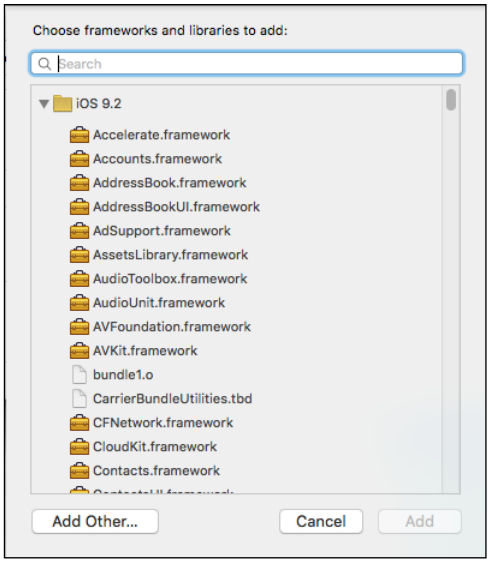


Abbildung 322 Suchen des Eintrags "CoreLocation", Teil 2

Nun gibt man den Suchbegriff ein:

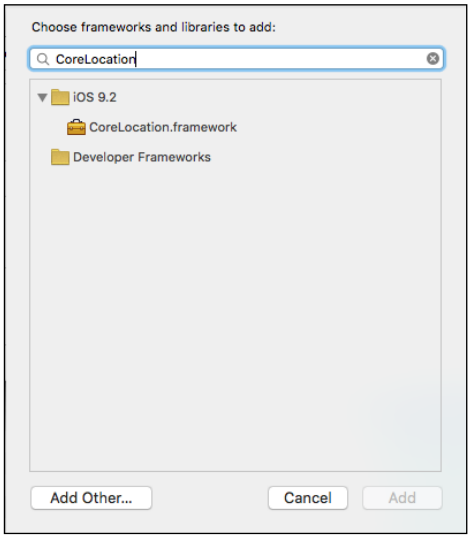


Abbildung 323 Eintragen von CoreLocation zum Suchen, Teil 3

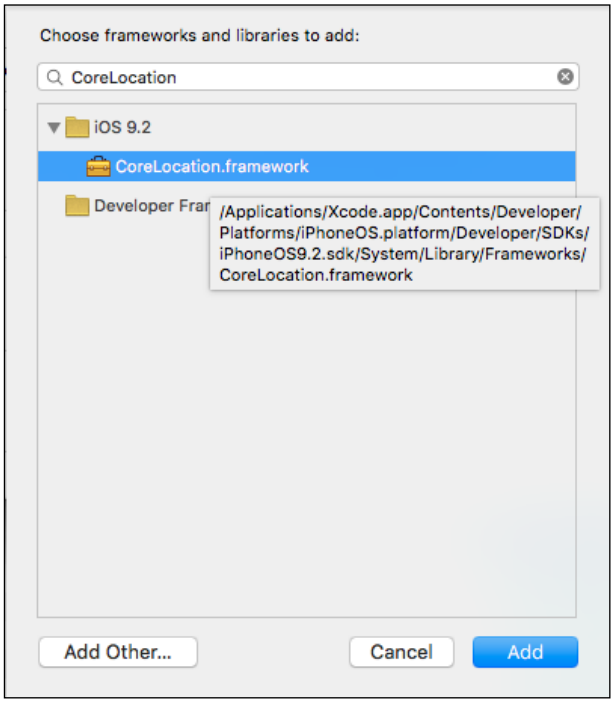


Abbildung 324 Anklicken von CoreLocation, Teil 4

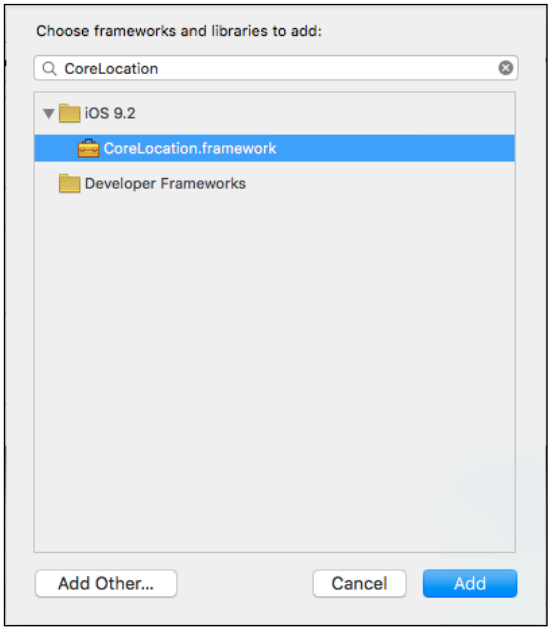


Abbildung 325 Am Schluss den Schalter „Add“ betätigen, Teil 4

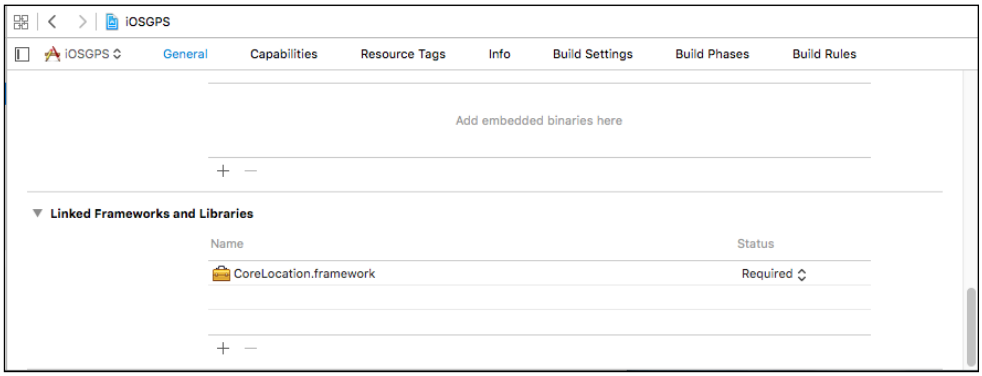


Abbildung 326 Ziel erreicht, das Framework ist verlinkt, Teil 5

18.4.2 Eintragen der Rechte-Abfrage

Für die Rechte-Abfrage öffnet man den Eintrag pList. Dann erscheint die untere Abbildung.

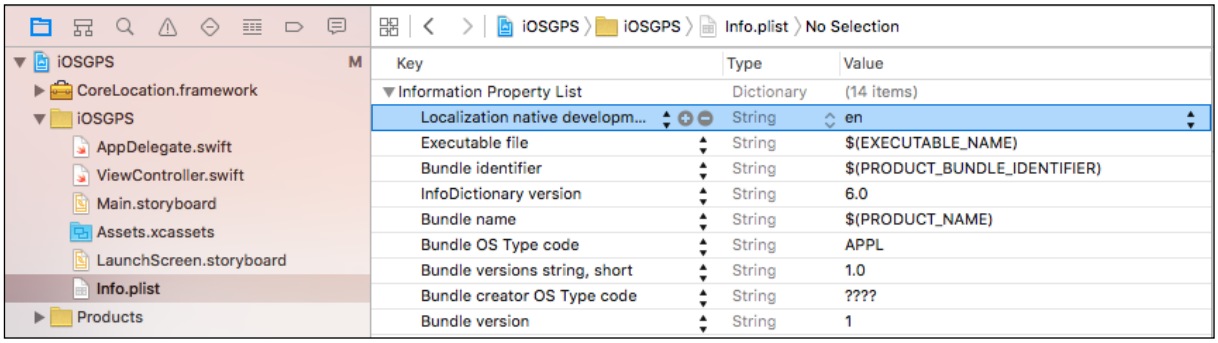


Abbildung 327 Eintragen der Rechte-Abfrage

In dieser soll man eine neuen Eintrag in der Spalte „Key“ hinzufügen:
NSLocationWhenInUseUsageDescription

Der Datentyp ist String. Als Value gibt man eine Beschreibung ein.

Das hat leider noch nicht geklappt.

18.4.3 Quellcode

Importmodule: CoreLocation
Delegate: CLLocationManagerDelegate

```
import UIKit
import CoreLocation

class ViewController: UIViewController, CLLocationManagerDelegate {

    // uiAttibute
    @IBOutlet var uiLocalisation: UILabel!
```

```
@IBOutlet var uiZipcode: UILabel!

@IBOutlet var uiCountry: UILabel!

@IBOutlet var uiArea: UILabel!

let clLocationManager = CLLocationManager()

// uiEvents

@IBAction func bnGetGPSDatasClick(sender: UIButton) {
    clLocationManager.delegate = self
    clLocationManager.desiredAccuracy =
                    kCLLocationAccuracyNearestTenMeters
      // Abfrage, nur wenn App im Vordergrund
    clLocationManager.requestWhenInUseAuthorization()
    clLocationManager.startUpdatingLocation()
}

  // Delegate Methoden
func locationManager(
        locManager: CLLocationManager,
        didUpdateLocations locations: [CLLocation]) {
    // innere Methode
  CLGeocoder().reverseGeocodeLocation(
        locManager.location!, completionHandler: {
     (placemarks, error) -> Void in

        if error != nil {
            self.uiLocalisation.text =
                "Fehler bei der Umwandlung der Geokoordianten"
              + error!.localizedDescription
        }
        else {
            if placemarks!.count > 0 {
                let clPlacemark = placemarks![0]
                self.setLocationInformation(clPlacemark)

            }
            else {
                self.uiLocalisation.text =
                  "Fehlerhafte Daten vom Geocoder"
            }
        }
    })
}
```

```
        // aus den Daten die ui-Attribute setzen
    func setLocationInformation(clPlacemark: CLPlacemark?) {
        if let placemark = clPlacemark {
            clLocationManager.stopUpdatingLocation()

        uiZipcode.text =
        (placemark.postalCode != nil) ? placemark.postalCode : ""

        uiCountry.text =
         (placemark.administrativeArea != nil) ?
              placemark.administrativeArea : ""

        uiArea.text = (placemark.country != nil) ?
              placemark.country : ""
        }
    }

    func locationManager(
          manager: CLLocationManager,
          didFailWithError error: NSError) {
      uiLocalisation.text =
          "Fehler bei der Umwandlung der Geokoordianten"
            + error.localizedDescription
    }
    // Rest gelöscht
}
```

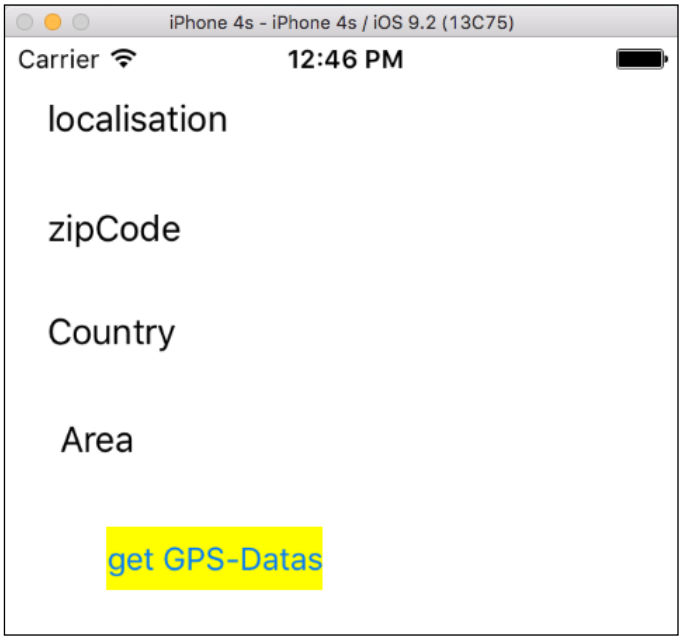


Abbildung 328 Ergebnis, aber noch nicht life getestet

19 Games

Neben den normalen Framework-Vorgaben, gibt es auch eine Game-Vorlage. Diese zeigt dann eindrucksvoll ihre Leistungesfähigkeit.

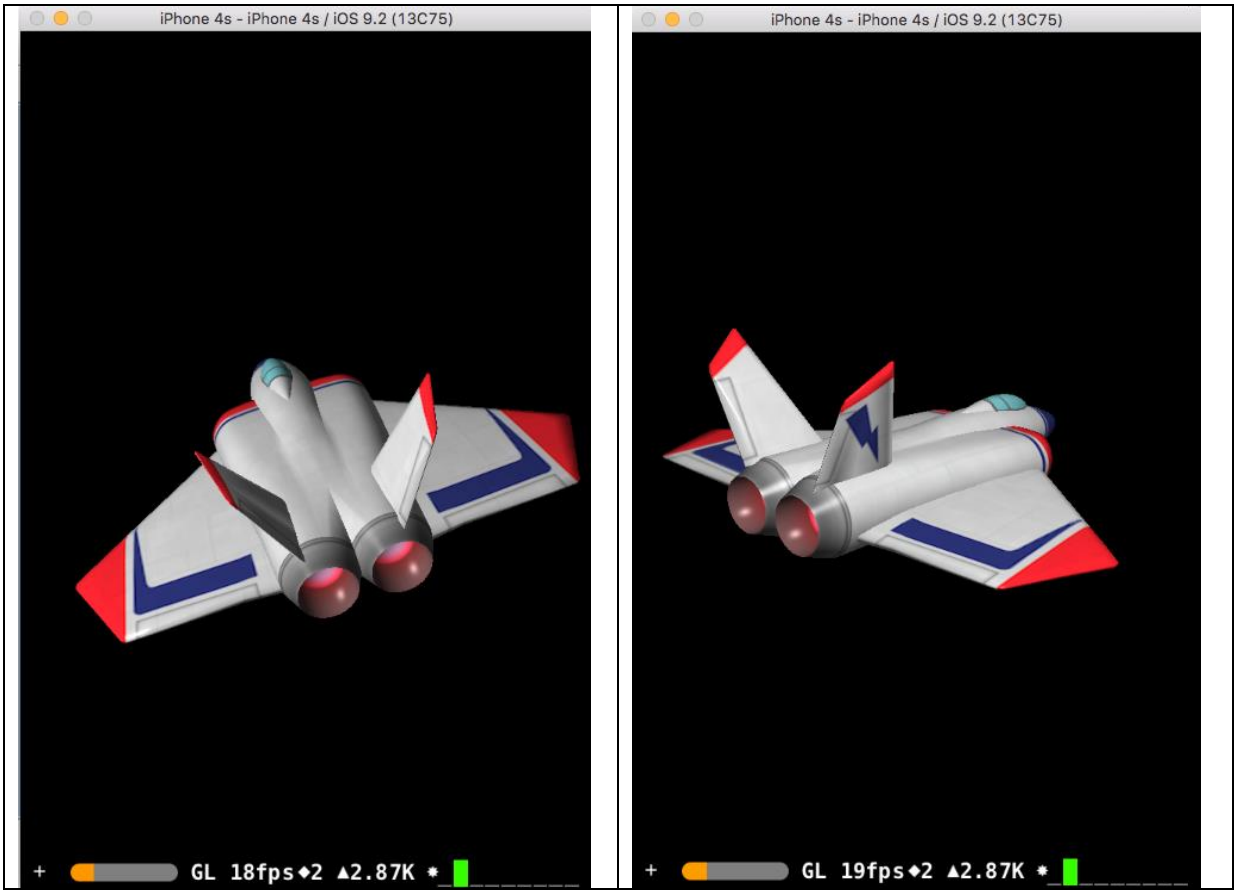


Abbildung 329 Anzeige des Beispielsgames

Vorlage aus XCode Version 7.x

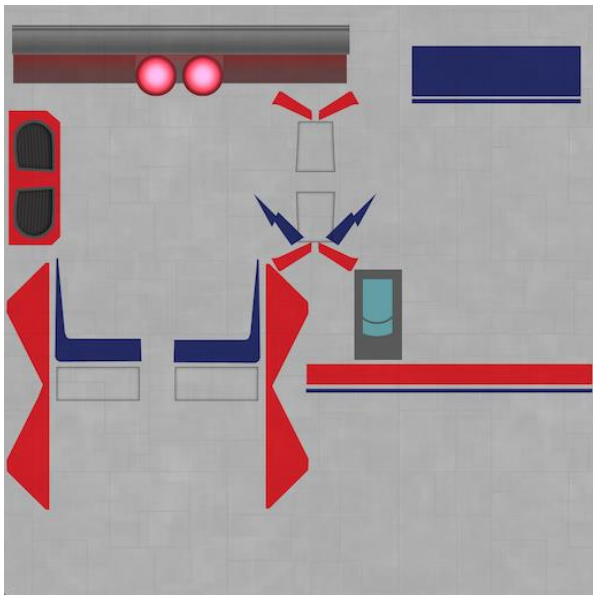


Abbildung 330 Texture des Flugzeugs

Quellcode:

```
//  GameViewController.swift
//  uiGame1

import UIKit
import QuartzCore
import SceneKit

class GameViewController: UIViewController {

    override func viewDidLoad() {
        super.viewDidLoad()

        // create a new scene
        let scene = SCNScene(named: "art.scnassets/ship.scn")!

        // create and add a camera to the scene
        let cameraNode = SCNNode()
        cameraNode.camera = SCNCamera()
        scene.rootNode.addChildNode(cameraNode)

        // place the camera
        cameraNode.position = SCNVector3(x: 0, y: 0, z: 15)

        // create and add a light to the scene
        let lightNode = SCNNode()
        lightNode.light = SCNLight()
        lightNode.light!.type = SCNLightTypeOmni
        lightNode.position = SCNVector3(x: 0, y: 10, z: 10)
        scene.rootNode.addChildNode(lightNode)
```

```
    // create and add an ambient light to the scene
    let ambientLightNode = SCNNode()
    ambientLightNode.light = SCNLight()
    ambientLightNode.light!.type = SCNLightTypeAmbient
    ambientLightNode.light!.color = UIColor.darkGrayColor()
    scene.rootNode.addChildNode(ambientLightNode)

    // retrieve the ship node
    let ship = scene.rootNode.childNodeWithName(
                          "ship", recursively: true)!

    // animate the 3d object
    ship.runAction(SCNAction.repeatActionForever(
         SCNAction.rotateByX(0, y: 2, z: 0, duration: 1)))

    // retrieve the SCNView
    let scnView = self.view as! SCNView

    // set the scene to the view
    scnView.scene = scene

    // allows the user to manipulate the camera
    scnView.allowsCameraControl = true

    // show statistics such as fps and timing information
    scnView.showsStatistics = true

    // configure the view
    scnView.backgroundColor = UIColor.blackColor()

    // add a tap gesture recognizer
    let tapGesture = UITapGestureRecognizer(
             target: self, action: "handleTap:")
    scnView.addGestureRecognizer(tapGesture)
}



func handleTap(gestureRecognize: UIGestureRecognizer) {
    // retrieve the SCNView
    let scnView = self.view as! SCNView

    // check what nodes are tapped
    let p = gestureRecognize.locationInView(scnView)
    let hitResults = scnView.hitTest(p, options: nil)
    // check that we clicked on at least one object
    if hitResults.count > 0 {
        // retrieved the first clicked object
        let result: AnyObject! = hitResults[0]

        // get its material
        let material = result.node!.geometry!.firstMaterial!
```

```
            // highlight it
            SCNTransaction.begin()
            SCNTransaction.setAnimationDuration(0.5)

            // on completion - unhighlight
            SCNTransaction.setCompletionBlock {
                SCNTransaction.begin()
                SCNTransaction.setAnimationDuration(0.5)

                material.emission.contents = UIColor.blackColor()

                SCNTransaction.commit()
            }

            material.emission.contents = UIColor.redColor()

            SCNTransaction.commit()
        }
    }

    override func shouldAutorotate() -> Bool {
        return true
    }

    override func prefersStatusBarHidden() -> Bool {
        return true
    }

    override func supportedInterfaceOrientations()
                        -> UIInterfaceOrientationMask {
        if UIDevice.currentDevice().userInterfaceIdiom == .Phone {
            return .AllButUpsideDown
        } else {
            return .All
        }
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Release any cached data, images, etc that aren't in
use.
    }

}
```

20 Testen einer App

Aufruf der Webseite:
http://help.adobe.com/en_US/as3/iphone/WS144092a96ffef7cc-371badff126abc17b1f-8000.html

20.1 Generating a certificate signing request

To obtain a developer certificate, you generate a certificate signing request file, which you submit at the Apple iPhone Dev Center site.
Generate a certificate signing request on Mac OS

On Mac OS, you can use the Keychain Access application to generate a code signing request. The Keychain Access application is in the Utilities subdirectory of the Applications directory. On the Keychain Access menu, select Certificate Assistant > Request a Certificate from a Certificate Authority.

1. Open Keychain Access.

2. On the Keychain Access menu, select Preferences.

3. In the Preferences dialog box, click Certificates. Then set Online Certificate Status Protocol and Certificate Revocation List to Off. Close the dialog box.

4. On the Keychain Access menu, select Certificate Assistant > Request a Certificate from a Certificate Authority.

5. Enter the e-mail address and name that matches your iPhone developer account ID. Do not enter a CA e-mail address. Select Request is Saved to Disk and then click the Continue button.

6. Save the file (CertificateSigningRequest.certSigningRequest).

7. Upload the CSR file to Apple at the iPhone developer site. (See “Apply for an iPhone developer certificate and create a provisioning profile”.)

Generate a certificate signing request on Windows

For Windows developers, it may be easiest to obtain the iPhone developer certificate on a Mac computer. However, it is possible to obtain a certificate on a Windows computer. First, you create a certificate signing request (a CSR file) using OpenSSL:

1. Install OpenSSL on your Windows computer. (Go to http://www.openssl.org/related/binaries.html.). You may also need to install the Visual C++ 2008 Redistributable files, listed on the Open SSL download page. (You do not need Visual C++ installed on your computer.)

2. Open a Windows command session, and CD to the OpenSSL bin directory (such as c:\OpenSSL\bin\).

3. Create the private key by entering the following in the command line:

```
openssl genrsa -out mykey.key 2048
```

Save this private key file. You will use it later.

When using OpenSSL, do not ignore error messages. If OpenSSL generates an error message, it may still output files. However, those files may not be usable. If you see errors, check your syntax and run the command again.

4. Create the CSR file by entering the following in the command line:

openssl req -new -key mykey.key -out CertificateSigningRequest.certSigningRequest -subj "/emailAddress=yourAddress@example.com, CN=John Doe, C=US"

Replace the e-mail address, CN (certificate name), and C (country) values with your own.

5. Upload the CSR file to Apple at the iPhone developer site. (See “Apply for an iPhone developer certificate and create a provisioning profile”.)

21 Delegates

21.1 PickerView-Delegates

21.1.1 Eine Spalte

```
class ViewController: UIViewController, UIPickerViewDelegate, UIPickerViewDataSource {

func numberOfComponents(_ pickerView:UIPickerView)-> Int{
return 1                // Anzahl der Spalten
}

func pickerView(_ pickerView:UIPickerView, numberOfRowsInComponent component:Int) -> Int{
return 42               // Anzahl der Zeilen
}

func pickerView(_ pickerView:UIPickerView, titleForRow row:Int, forComponent component:Int) ->
String?{
  return String(row+1)              // aktueller Wert der Zeile
}

func pickerView(_ pickerView: UIPickerView, didSelectRow row: Int, inComponent component: Int)
{
// selected Index
}

override func viewDidLoad() {
    super.viewDidLoad()
    self.uiPickerView.delegate=self
    uiPickerView.dataSource = self
}
```

21.1.2 Zwei Spalten im PickerView (Stunden, Minuten)

```
class ViewController: UIViewController, UIPickerViewDelegate, UIPickerViewDataSource {

func numberOfComponents(_ pickerView:UIPickerView)-> Int{
    // Anzahl der Spalten
    return 2;
}
```

```
func pickerView(_ pickerView:UIPickerView, numberOfRowsInComponent component:Int) -> Int{
    // Abfrage nach der Anzanl der Zeilen in Abhängigkeit der Spalten
    if component==0 {
       return 24
    }
    else {
        return 60
    }
}


func pickerView(_ pickerView:UIPickerView, titleForRow row:Int, forComponent component:Int) ->
String?{
    // aktueller Wert der Zeile, eventuelle Offset müssen hier berücksichtigt werden
    if component==0 {
       return row
    }
    else {
        return row
    }
}

func pickerView(_ pickerView: UIPickerView, didSelectRow row: Int, inComponent component: Int)
{
    // selected Index
}

override func viewDidLoad() {
    super.viewDidLoad()
    self.uiPickerView.delegate=self
    uiPickerView.dataSource = self
}
```

21.2 TableView-Delegates

```
class ViewController: UIViewController, UITableViewDelegate, UITableViewDataSource {

let staedte = ["NewYork", "Mailand", "Rom", "Wernigerode","Moskau",

"Magdeburg", "München", "Kiel", "Amsterdam", "London", "Paris", "Tokio", "Toronto"]


override func viewDidLoad() {
    super.viewDidLoad()
    tableview.delegate=self
    tableview.dataSource=self
}
```

```
func numberOfSectionsInTableView(tableView: UITableView) -> Int {
    return 1
}

func tableView(tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
    return cities.count   // Anzahl der Zeilen
}

func tableView(tableView: UITableView, cellForRowAtIndexPath indexPath: NSIndexPath) -> UITableViewCell {
    // Wert der aktuellen Zeile, Rückgabe eines TableViewCell
    let cell:UITableViewCell=UITableViewCell(style: UITableViewCellStyle.Subtitle,
                reuseIdentifier: "mycell")
    cell.textLabel!.text = "Hallo"
    return cell
}
```

22 Indexverzeichnis

A

B

C

D

G

I

K

L

M

N

O

P

R

S

T

U

Z