

UITableView mit Swift

Schneller Einstieg für Anfänger

von: codingtutor.de
Autor: Jan Brinkmann

I. Die UITableView kennenlernen

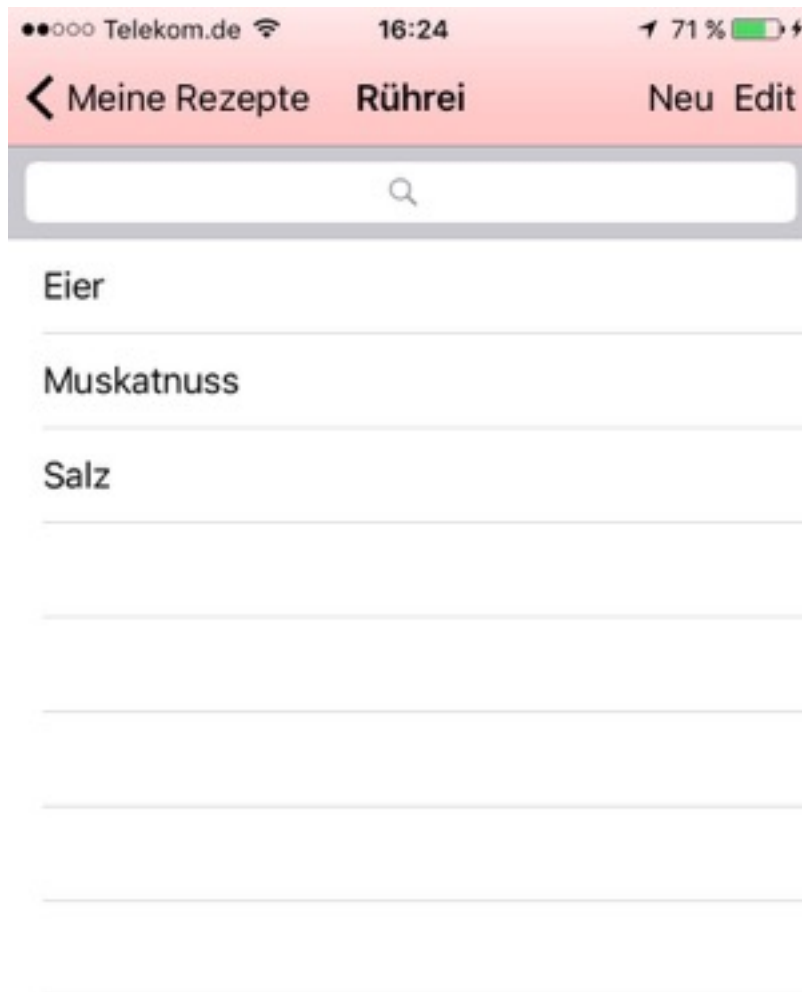
Streng genommen ist eine “TableView” ein View-Element, das Inhalte tabellarisch darstellt. Die Logik ist in der **Klasse UITableView** implementiert. Doch es gehören noch weitere Klassen, wichtige Enumerations, Protokolle und eine ganze Reihe Methodenaufrufe dazu. Nur über die Dokumentation der Klasse allein ist es schwer die Zusammenhänge im Detail zu verstehen. Bevor wir im zweiten Teil des Buches in die Praxis einsteigen, lernst Du die Bestandteile kennen. Zuerst gibt es eine allgemeine Übersicht. Eine konkretere Erklärung findest Du noch einmal einzeln in den jeweiligen Abschnitten.

Verschiedene Tabellenarten






Da viele Möglichkeiten angeboten werden, gibt es nicht die eine Standardansicht. In der Praxis triffst Du auf verschiedene Ausprägungen. Es gibt sie als schlichte Tabelle, unterteilt in Sektionen, mit dynamischen oder statischen Inhalten sowie mit oder ohne Index. Auch die Tabellenzeilen können sehr unterschiedlich aussehen, unterschiedlichen hoch sein und individuell umgesetzt werden. Dazu aber später im Abschnitt über die UITableViewCell mehr.

Um nicht zu sehr in die Theorie abzuschweifen und einen ersten Eindruck zu bekommen, ist es sinnvoll einige Beispiele zu sehen. Unter anderem die folgenden Screenshots werden Dir in sehr

ähnlicher Form im praktischen Beispiel im zweiten Teil dieses Buches wieder begegnen. Diese Ansichten sind alle Teil der App, die wir implementieren.



Einfache Tabelle, mit UISearchBar

Telekom.de 16:24 71 %		
Meine Rezepte		
Frühstück		
	Rührei 3 Zutaten	>
	Pancakes 6 Zutaten	>
Mittag		
	Nudeln mit Tomatensoße 6 Zutaten	>
	Tofu Salat 6 Zutaten	>

Eine Tabelle mit Sektionen

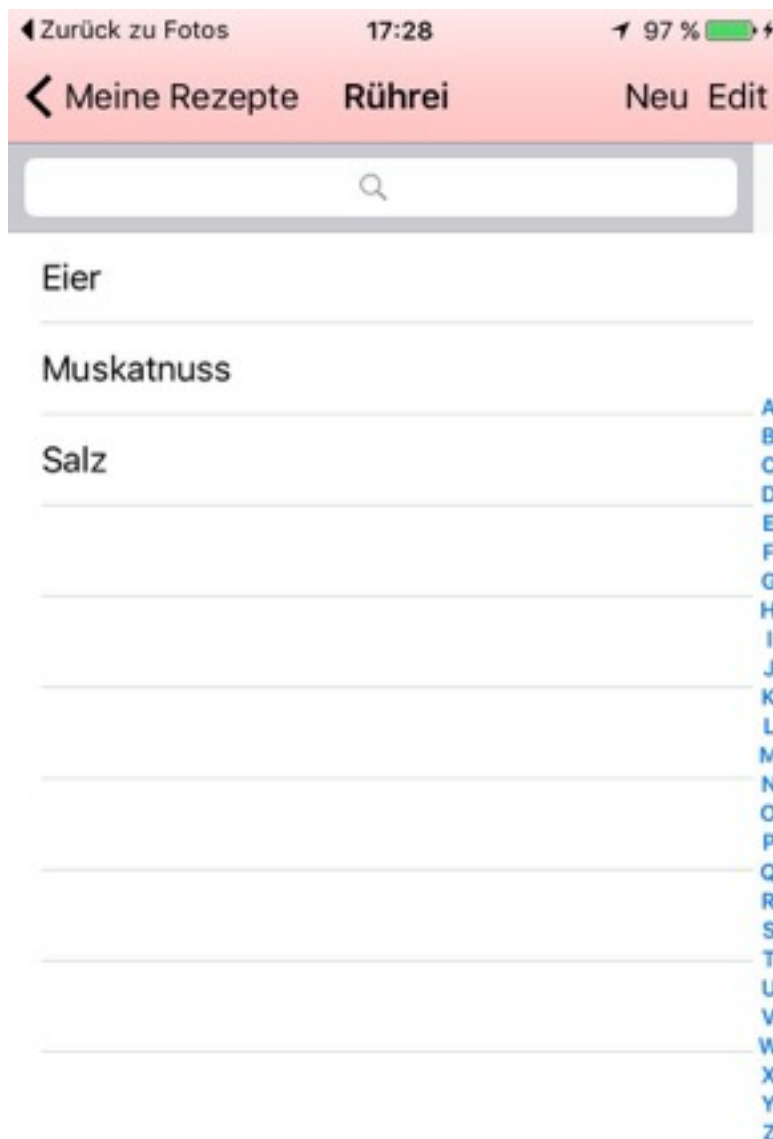


Tabelle mit Index

Nur die Spitze des Eisbergs

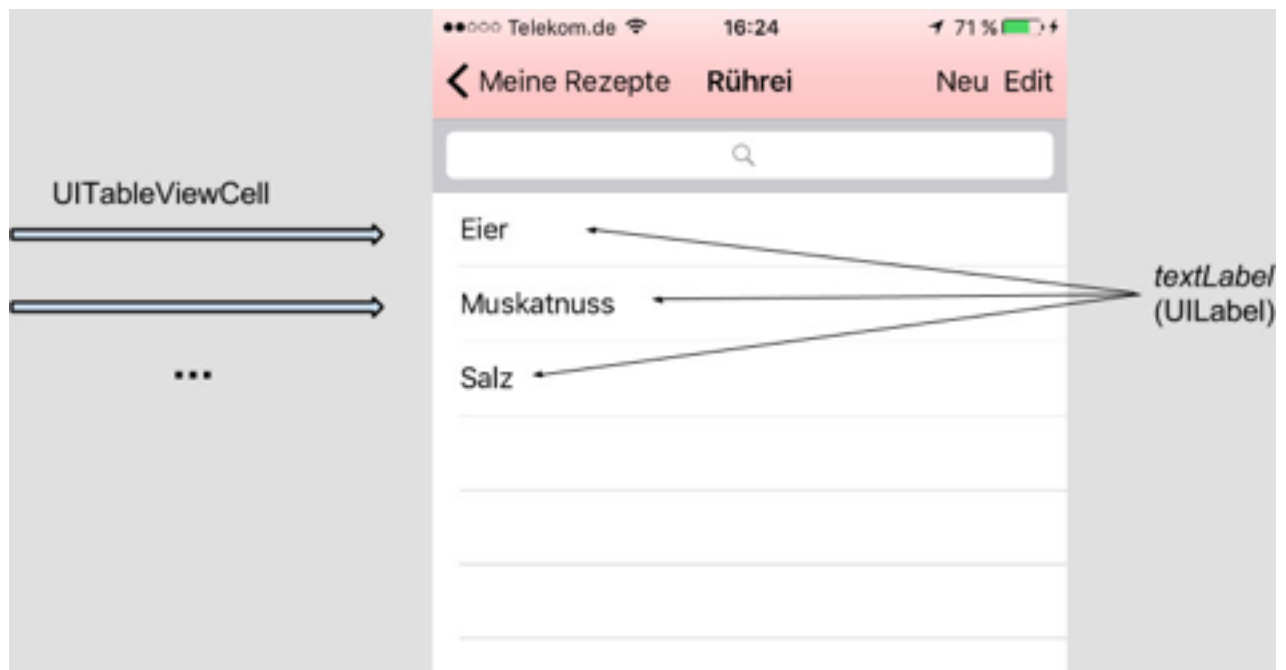
Wenn die Rede von "einer UITableView" ist, meint dies umgangssprachlich häufig eben genau eine Tabelle wie oben zu sehen - inkl. der sichtbaren Inhalte, der Zeilen und so weiter. Allerdings ist die Klasse *UITableView* in diesem Orchesta nicht

allein für die Darstellung verantwortlich. Sie ist auch Dirigent und Taktgeber. Die weiteren Bestandteile bleiben zumindest zunächst verborgen.

Um die Tabelle zu konfigurieren, anzupassen und zu verändern wird über die speziellen Outlets *datasource* und *delegate* ein Controller festgelegt. Der implementiert spezielle Methoden aus den Protokollen *UITableViewDataSource* und *UITableViewDelegate*. Die *UITableView* erkennt wenn sie vorhanden sind und ruft sie auf. Dabei fragt sie z.B. Wie viele Zeilen die Tabelle enthält, wie hoch jede Zeile sein oder welche *UITableViewCell* Instanz anzuzeigen ist. Über die Rückgabewerte nährt sich die Tabelle und wird schließlich verändert.

Pro Zeile ein Objekt

Jede Zeile wird von einer Instanz der Klasse *UITableViewCell* repräsentiert. Die kann verschiedene Informationen aufnehmen, zum Beispiel Text in einem *UILabel* mit dem Namen *textLabel*. Hier können Zeichenketten hinterlegt werden, die in der Tabelle aufgelistet werden sollen. In der einfachen Tabelle oben gibt es bspw. drei *UITableViewCell*-Objekte. In der enthaltenen Eigenschaft *textLabel* wird jeweils ein anderer Wert gespeichert.



UITableViewCell in einer Tabelle

Daraus ergibt sich die Liste der Zutaten: Eier, Muskatnuss und Salz. Woher die Anzahl der Einträge, die Objekte und die Liste der Inhalte stammt, siehst Du später im Beispiel.

Inhalte scrollen

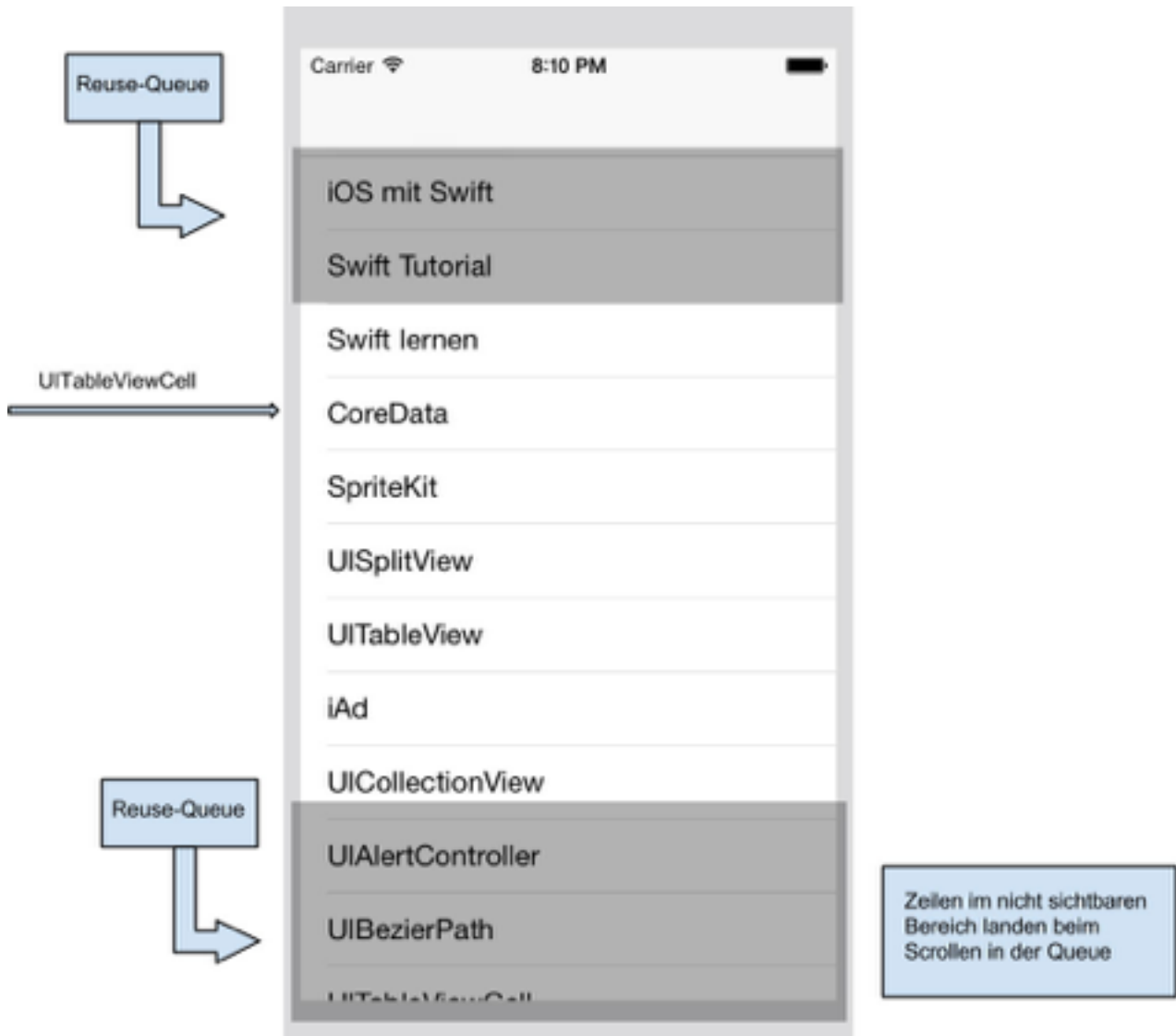
Zentraler Bestandteil der TableView ist die Möglichkeit die Inhalte zu scrollen. So kannst Du deutlich mehr Inhalte in einer Tabelle hinterlegen als gleichzeitig angezeigt werden. Was selbstverständlich klingt, ist technisch nicht ganz so trivial.

Hast Du schon mal mit "Schwung" durch eine Liste "gescrollt"? Die Zeilen fliegen nur so vorbei und Inhalte sind kaum für eine

Sekunde sichtbar. Wenn nun für jede Zeile immer wieder ein neues Objekt erstellt würde, gäbe es zwei Probleme. Zum einen verbrauchst die App immer mehr Speicher. Bei 217 Einträgen wären 217 Objekte notwendig, obwohl immer nur ein Bruchteil davon angezeigt wird. Das klingt nicht sehr effizient. Außerdem würde vielleicht die Animation ins Stocken geraten. Es kann gut sein, dass nicht jedes Objekt immer schnell genug erzeugt wird. Bei der Spielentwicklung spricht man von sogenannten "lags" (englisch für "Verzögerung"). Was bei Spielen den Spaß schnell verderben kann, wird bei einer normalen Anwendung mindestens noch als störend empfunden.

Die Reuse-Queue

Die Antwort ist die sogenannte Reuse-Queue (englisch "to reuse" bedeutet soviel wie "erneut verwenden"). Dazu werden Tabellenzeilen mit einem eindeutigen "Reuse-Identifizier" versehen. Wenn ein Eintrag im nicht sichtbaren Bereich verschwindet wird das Objekt nicht einfach gelöscht. Es wird zur erneuten Verwendung freigegeben. Der Trick besteht nun darin nicht einfach neue Objekte zu erzeugen. Stattdessen wird zunächst im Cache ein freies Objekt gesucht. Nur wenn keines vorhanden ist, wird eine neue Instanz erzeugt und der "Reuse-Identifizier" hinterlegt. Bildlich sieht das in etwa wie folgt aus:



Schematische Darstellung der Reuse-Queue

Der etwas dunklere Bereich soll hier den Bereich andeuten, der nicht mehr im sichtbaren Feld liegt. So würden bspw. die Zeilen "iOS mit Swift" oder "Swift Tutorial" oben im Cache abgelegt. Im unteren Bereich wären es die Einträge "UIAlertController", "UIBezierPath" und so weiter.

Im Beispiel oben sind nur sieben Einträge gleichzeitig sichtbar. Wenn nun eine Zeile vollständig aus dem sichtbaren Bereich verschwindet, wird sie zur Wiederverwendung freigegeben. Statt der 217 Objekte würden plötzlich jetzt nur noch acht verwendet. Bei sieben sichtbaren Zeilen muss mindestens eine in der Reserve bleiben. Das ist etwa 27 Mal weniger Speicherverbrauch!

Eine Ausnahme besteht, wenn die Zeile ausgewählt wurde und erst dann in den nicht sichtbaren Bereich verschwindet. Für das Grundverständnis ist das aber erstmal nicht relevant.

UITableView- oder UITableViewController?

Die UITableView steht für das Viewelement. Es stellt die Tabelle dar und wird, wie oben erwähnt, mit UITableViewCell Objekten gefüllt. In jedem UIViewController kann so eine Tabelle hinterlegt werden, weitestgehend auch in beliebiger Größe.

Um mit einer UITableView zu arbeiten, musst Du mehrere Komponenten verbinden. Zunächst bildet ein Viewcontroller die Grundlage. Darin wird die Tabelle, also die UITableView selbst, hinterlegt. Der Zugriff auf die Tabelle erfolgt aus dem Code heraus über ein Outlet, was schon der dritte Bestandteil ist.

Die nächste Säule sind die fest eingebauten Outlets *datasource* und *delegate*. Die verbindest Du mit dem Controller, in dem die Tabelle hinterlegt wurde. Die Klasse von einem Controller muss

dafür mit dem *UITableViewDataSource*- und dem *UITableViewDelegate*-Protokoll schließlich die letzten beiden Bestandteile implementieren.

Der Unterschied zum *UITableViewController* ist im wesentlichen, dass all diese Aufgaben bereits erledigt wurden. Es müssen lediglich wenige Methoden überschrieben werden, da der *UITableViewController* schließlich nicht raten kann, welche Daten angezeigt werden, wie viele dies sind und so weiter. Wenn die Tabellenzeilen ausgewählt werden können, müssen Methoden aus dem Delegate-Protokoll überschrieben und selbst implementiert werden. Auch hier weiß der Controller nicht, was in der jeweiligen App für eine Aktion folgen soll.

Außerdem ist der gesamte *ViewController* mit der Tabelle ausgefüllt.

Sections und Rows

Jede Zeile gehört zu einer Sektion. Selbst wenn Du das nicht aktiv angibst, hat Deine Tabelle automatisch eine Sektion. Und genau der werden die Zeilen zugeordnet. Dargestellt wird die *UITableView* in dem Fall als relativ einfache Liste.

Es gibt aber auch die Möglichkeit eine Tabelle zu unterteilen. Ein Beispiel ist unter iOS das Adressbuch. Hier hast Du für jeden Buchstaben eine eigene Sektion, um eine optische Trennung zu

erreichen. Gleichzeitig dient diese Gliederung auch zur Navigation über den Tabellenindex auf der rechten Seite.

Eine Sektion besteht aus einem möglichen Header (einem Titel über den Einträgen), einem Footer (ein weiterer Titel unterhalb der Einträge) sowie den Zeilen der Sektion selbst:

Header	
Row 0	
Row 1	
Row 2	
Footer	
Header	
Row 0	
Row 1	
...	
Footer	

Section 0

Section 1

Hier sind zwei Sektionen angedeutet, die sowohl Header als auch Footer darstellen. Darin enthalten sind jeweils wieder Zeilen.

Die Nummerierung der Sektionen und Zeilen beginnt ähnlich wie bei einem Array bei Null. Das hat ganz praktische Gründe, denn im Endeffekt werden die Inhalte auch nur in Datenstrukturen abgelegt. Wenn mehr als eine Zeilen

Um eine Tabellenzeile zu identifizieren, gibt es daher nicht nur die Nummer der Zeile. Hinzu kommt ein weiterer Wert, die Nummer der Sektion. Wie oben zu sehen, beginnt die Nummerierung der Zeilen in jeder Sektion neu. Das muss bei der Planung der eigenen Datenquelle in jedem Fall berücksichtigt werden. Bei acht Zeilen, die gleichmäßig auf zwei Sektionen verteilt sind, gibt es also schematisch folgenden Aufbau:

- Sektion 0
 - Zeile 0, Zeile 1, Zeile 2, ...
- Sektion 1
 - Zeile 0, Zeile 1, Zeile 2, ...
- ...

Analog zu dieser Struktur kann eine Datenquelle aufgebaut werden, z.B. als zweidimensionales Array.

Der NSIndexPath zur Adressierung

Im Zusammenhang mit der Adressierung der Tabellenzeilen gilt es die Klasse *NSIndexPath* zu kennen. Viele Methoden die aus dem Delegate- oder DataSource-Protokoll implementiert werden, bekommen genau das als Parameter übergeben. Daran kannst Du erkennen um welche Sektion und welche Zeile es geht. Ich habe die Erklärung in der Zeichnung weiter oben bewusst auf Englisch hinterlegt. Denn jedes *NSIndexPath*-Objekt bringt diese Informationen in den Eigenschaften *section* bzw. *row* mit. Genau diese Daten kannst Du auslesen und weiterverwenden. Wenn es ohnehin nur eine Sektion gibt, wird natürlich einfach nur mit der Zeile gearbeitet.

Die Datenquelle: UITableViewDataSource

Die Bestandteile der Tabelle helfen Dir nur, wenn Du Daten anzeigen kannst. Und da die in jeder Situation aus einer anderen Quelle stammen können, wird nicht bspw. einfach ein Array übergeben und angezeigt. Die UITableView arbeitet viel flexibler. Sie wird über das *dataSource*-Objekt konfiguriert. Dazu implementiert bspw. der Controller, in dem die Tabelle hinterlegt wurde, das *UITableViewDataSource* Protokoll und verschiedene Methoden daraus.

Auf den ersten Blick verwirrt der Methodenname, da er fast immer *tableView* lautet. Unterschiedlich sind die Parameter. Erkennen kannst Du die passende Methode daher nicht am

Namen, sondern am zweiten Parameter. Der hat einen externen Namen. Die UITableView ruft genau diese Methoden auf, um die Rahmendaten wie Anzahl der Sektionen und Zeilen, die eigentlichen Inhalte, Header und Footer für Sektionen und so weiter abzurufen. Das kannst Du nutzen, um Deine eigenen Inhalte anzubinden. Im DataSource-Protokoll gibt es dafür verschiedene Methoden. Die folgende Liste ist der offiziellen Dokumentation zum UITableViewDataSource-Protokoll entnommen:

Die Tabelle konfigurieren

- tableView(:cellForRowAtIndexPath:)
Holt das UITableViewCell Objekt für den NSIndexPath mit Nummer der Sektion x und Zeile y.
- tableView(:numberOfRowsInSection:)
Hier wird die Anzahl der Zeilen in der Section x abgefragt. Wenn es keine Sektionen gibt, dann wird die Gesamtzeile der Zeilen der Tabelle zurückgegeben.
- numberOfSectionsInTableView(:)
Gibt die Anzahl der Sektionen zurück. Das ist automatisch 1, wenn sie nicht implementiert wird.
- sectionIndexTitlesForTableView(:)
Gibt ein Array mit den Indexnamen zurück. Das sind im Adressbuch bspw. die Buchstaben von A bis Z.
- tableView(:sectionForSectionIndexTitle:atIndex:)
Hier wird einem Index die Position zugeordnet und zurückgegeben. Bei der Adressbuch-App würde z.B. einem Buchstaben zwischen A und Z die Sektion zuordnet.

- tableView(_:titleForHeaderInSection:)

Hier wird die Kopfzeile für die Sektion x abgefragt und zurückgegeben.

- tableView(_:titleForFooterInSection:)

Analog zur Kopfzeile wird hier der Titel für die Fußteile der Sektion x abgefragt und zurückgegeben.

Zwingend benötigt werden nur die ersten beiden Methoden. Die Methode muss wissen wie viele Tabellenzeilen in einer Sektion enthalten sind. Die Anzahl der Einträge liefert die Methode mit dem Parameter `numberOfRowsInSection`. Außerdem ist es natürlich wichtig, dass die jeweiligen Tabelleninhalte abgefragt werden können. Dazu dient der Aufruf mit dem Parameter `cellForRowAtIndexPath`.

Zeilen einfügen und löschen

- tableView(_:commitEditingStyle:forRowAtIndexPath:)

Die Methode wird aufgerufen, wenn eine Tabellenzeile mit dem roten Button entfernt wird. Sie dient dazu, den Vorgang in der Datenquelle vorzunehmen. Wir werden das später noch am Beispiel sehen. Um das Löschen per Wischen zu ermöglichen, muss diese Methode implementiert werden.

- tableView(_:canEditRowAtIndexPath:)

Über diese Methode können Zeilen einzeln oder ganz vom editieren ausgeschlossen werden.

Tabellenzeilen neu ordnen

- tableView(:canMoveRowAtIndexPath:)

Auch hier wird eine Berechtigung eingeholt. Einzelne oder alle Zeilen können von einer neuen Sortierung ausgeschlossen werden.

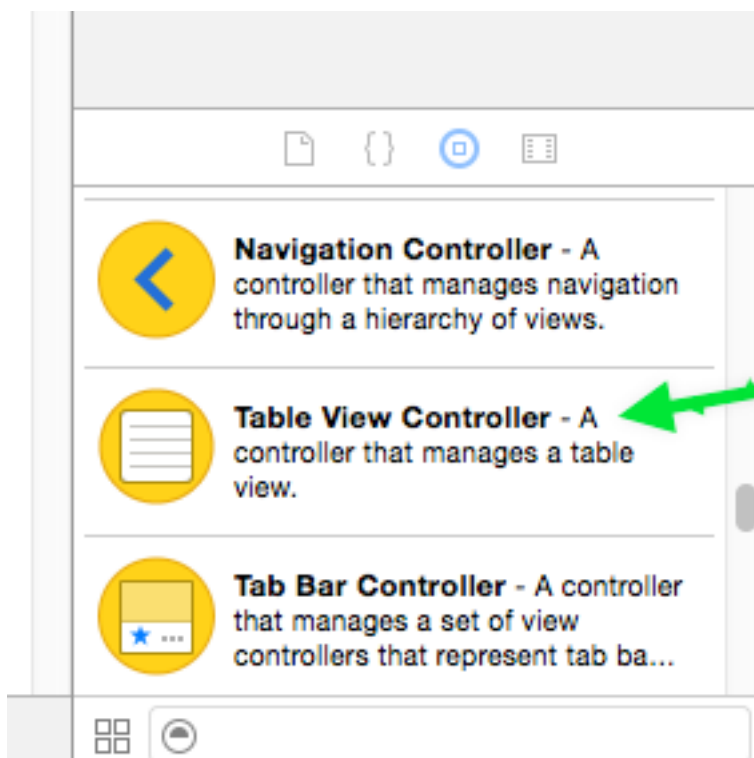
- tableView(:moveRowAtIndexPath:toIndexPath:)

Werden Zeilen neu geordnet, wird diese Methode aufgerufen. Sie ermöglicht es zum Beispiel ein Attribut für die Sortierung zu verändern, falls die Reihenfolge persistent gespeichert wird.

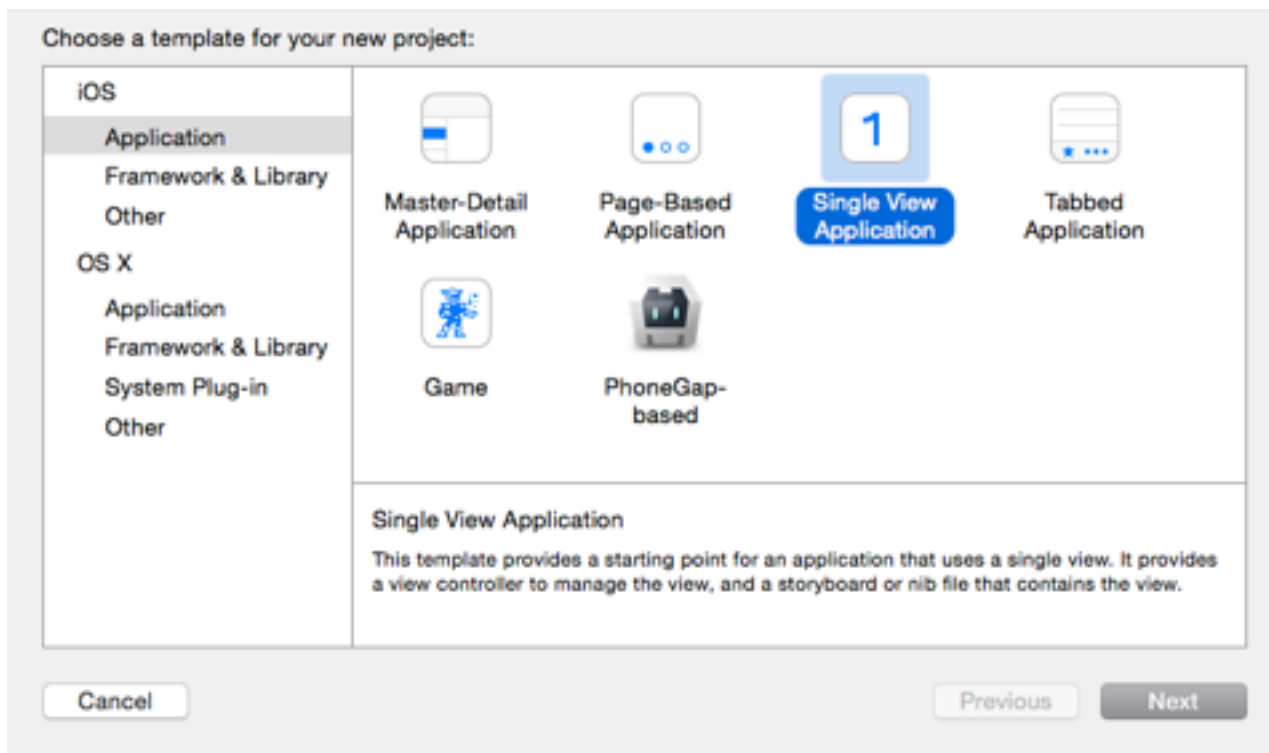
Diese Übersicht musst Du nicht auswendig lernen. Von meinen Kursteilnehmer werde ich aber häufiger gefragt woher die Methoden stammen, welche es gibt und wofür die gut sind. Wir werden die meisten davon noch in der Praxis wiedersehen. Sie dient erst mal nur dem besseren Überblick über die angebotenen Möglichkeiten.

II. UITableView Tutorial

Wenn Du in Deinen Apps eine TableView implementierst, gibt es mehrere Wege. Die erste Variante ist der UITableViewController selbst. In der Object Library findest Du im Interface Builder den Table View Controller:

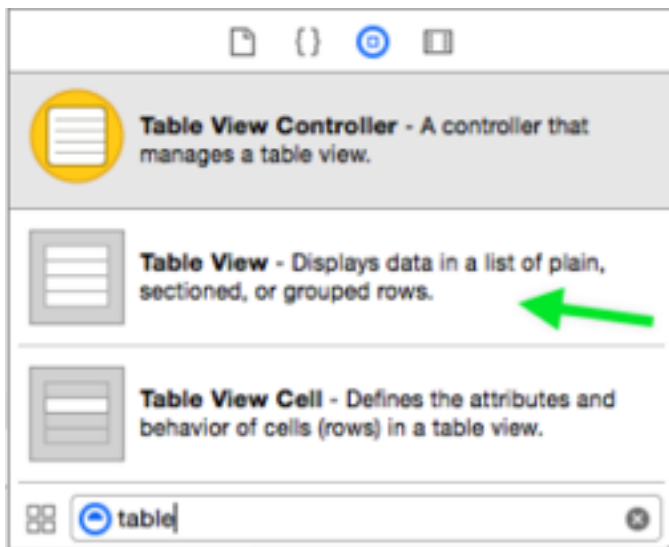


In diesem TableView Tutorial zeige ich den manuellen Weg. Das ermöglicht ein allgemeines Verständnis. Außerdem ist er auch auf diese Weise im UITableViewController vorbereitet. Das Projekt, dass ich hier vorstelle, basiert auf einer Single View Application. Über Xcode ein neues Projekt erzeugen und wie folgt wählen:

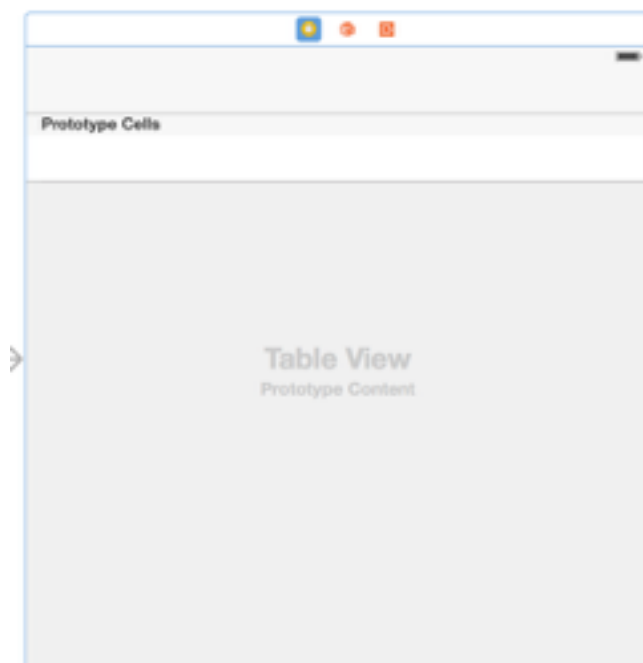


Storyboard: Eine TableView hinzufügen

Zunächst musst Du im Storyboard eine TableView in den noch leeren ViewController “legen”. Dazu suchst Du in der Object Library rechts unten (rechte Seitenleiste in Xcode einblenden) die Table View. Achte darauf, dass Du nicht aus Versehen den Table View Controller (mit dem gelben Icon) verwendest. Das gewünschte Element ist die *Table View*, mit dem grauen Icon. Siehe auch den folgenden Screenshot:

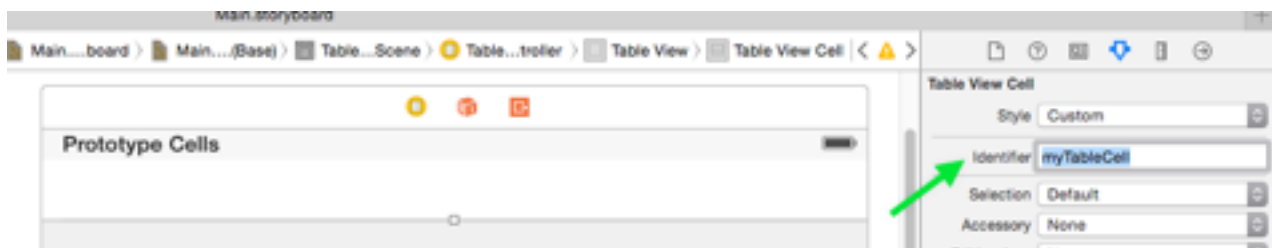


Einfach per Drag & Drop in den den leeren ViewController ziehen. Wenn Du erfolgreich warst, sieht es hinterher in etwa wie folgt aus:



Wenn Dir keine weiße Zeile unterhalb der Überschrift “Prototype Cells” angezeigt wird, musst Du die noch aktivieren. Dazu wählst Du die Tabelle aus (einfach den grauen Bereich anklicken) und öffnest auf der rechten Seite den Attribute Inspector. Dort wird die Anzahl der Prototype Cells auf eins gestellt. Jetzt erscheint die weiße Zeile.

Um den Reuse-Identifier zu setzen, musst Du den soeben erstellten Prototyp auswählen und rechts im Attribute Inspector wie oben beschrieben den Identifier auf myTableViewCell setzen:

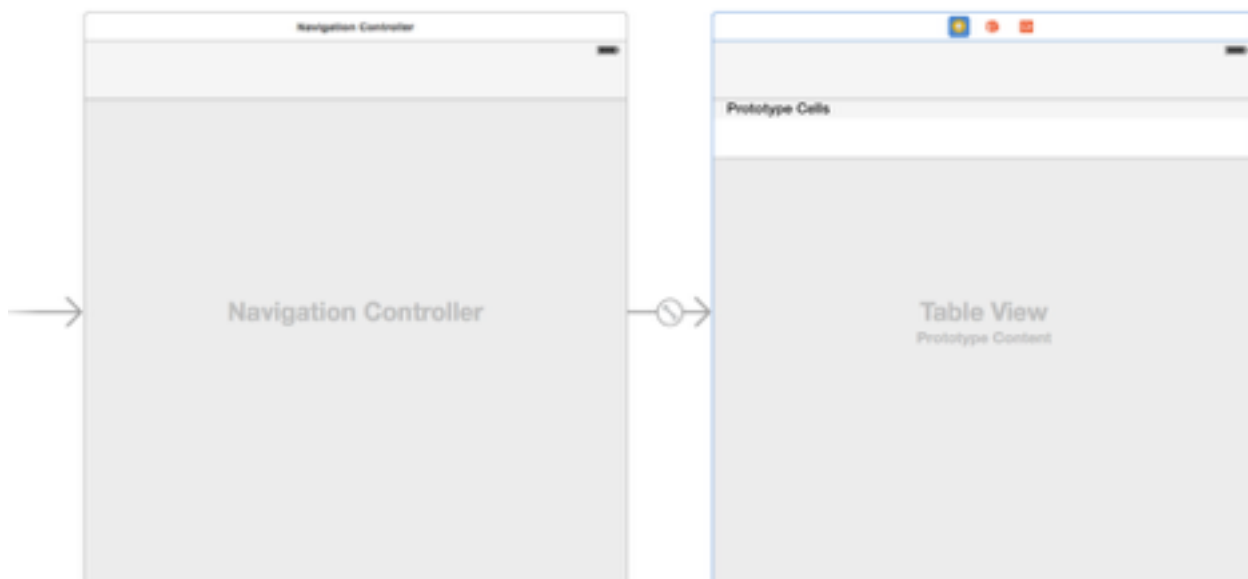


Der grafische Teil ist nun fast abgeschlossen. Allerdings empfehle ich bei Tabellen noch einen Navigation Controller (UINavigationController) zu verwenden. Der ist sehr nützlich, wenn die Tabellenzeilen ausgewählt werden können. Ein Beispiel: eine Liste mit Personen.

In der ersten Tabelle führst Du Namen auf. Wählt der Benutzer eine Person, öffnet sich eine Detailansicht. Der Navigation

Controller ermöglicht es dem Benutzer oben links einfach wieder zur Liste zurückzukehren. Ohne zusätzliche Programmlogik.

Um den Navigation Controller hinzuzufügen, musst Du den bereits vorhandenen ViewController auswählen. Dazu einfach das gelbe Icon wie oben im Screenshot wählen. Anschließend im Menu Editor -> Embed In -> Navigation Controller aufrufen. Dein Storyboard sollte nun in etwa wie folgt aussehen:

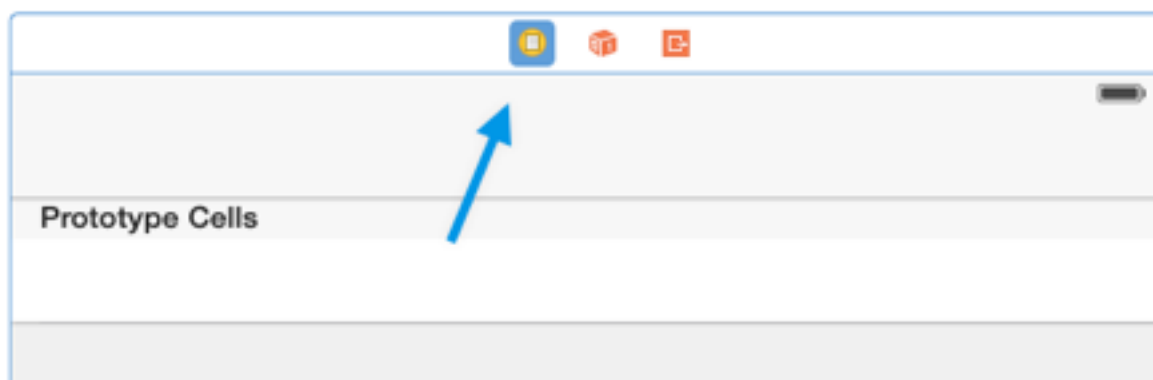


Ein IBOutlet für die Tabelle

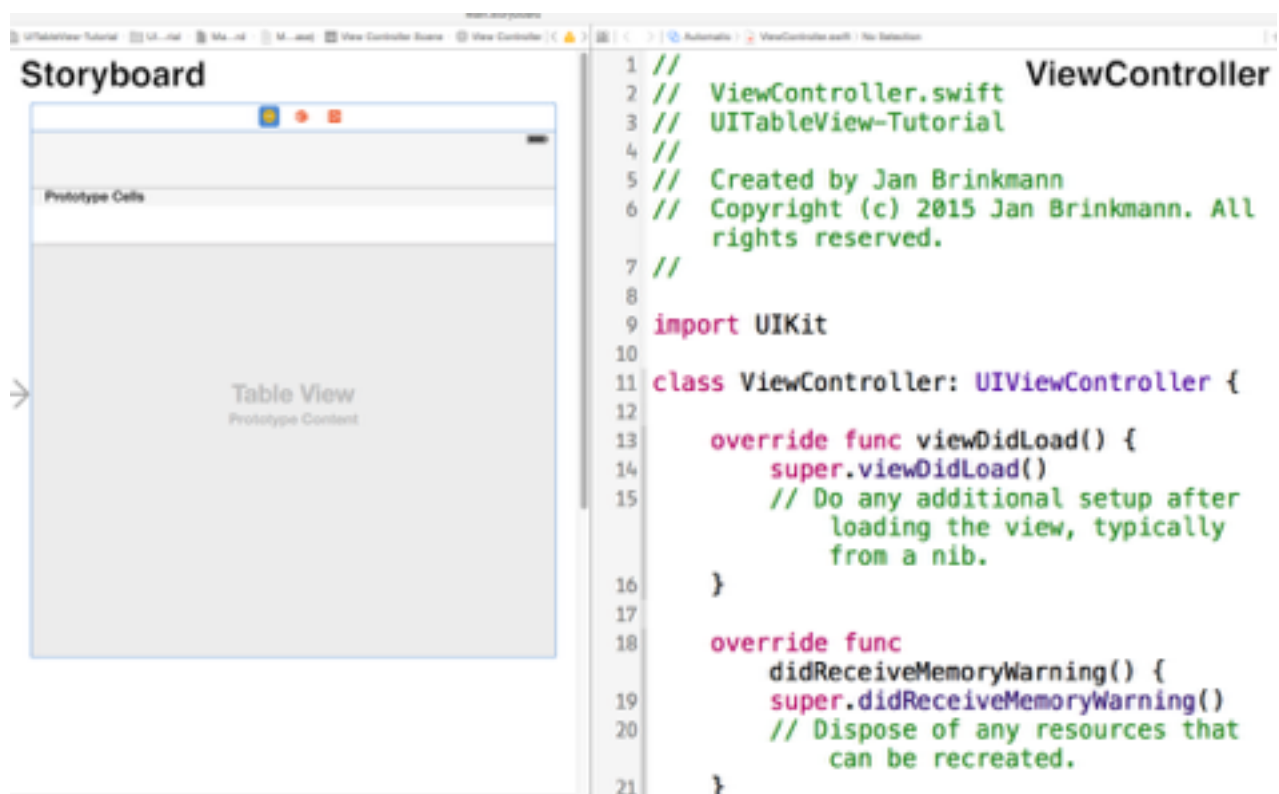
Im Quellcode, genau genommen im ViewController, greifst Du über Outlets auf Elemente wie die Table View zu. Diese Verbindung müssen wir herstellen. Öffne dazu das Storyboard und aktiviere den Assistant Editor. Xcode bietet dazu eine Schaltfläche oben rechts an (die beiden Ringe, die eine Schnittmenge andeuten):



Jetzt muss erneut der View Controller ausgewählt werden. Dazu wählst Du einfach das gelbe Icon über dem Controller mit der Table View aus:

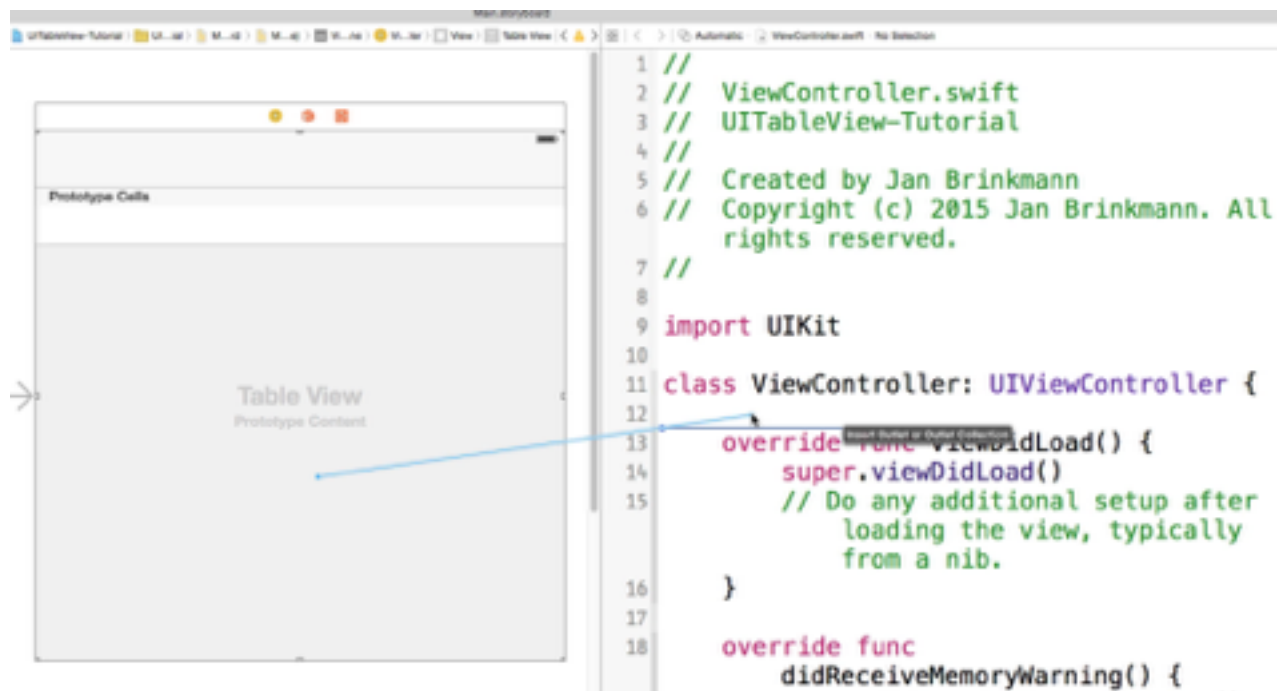


Auf der rechten Seite, also im Assistent Editor, sollte nun die Klasse ViewController geöffnet sein. Somit hast Du nun gleichzeitig auf das Storyboard und den Code Zugriff:

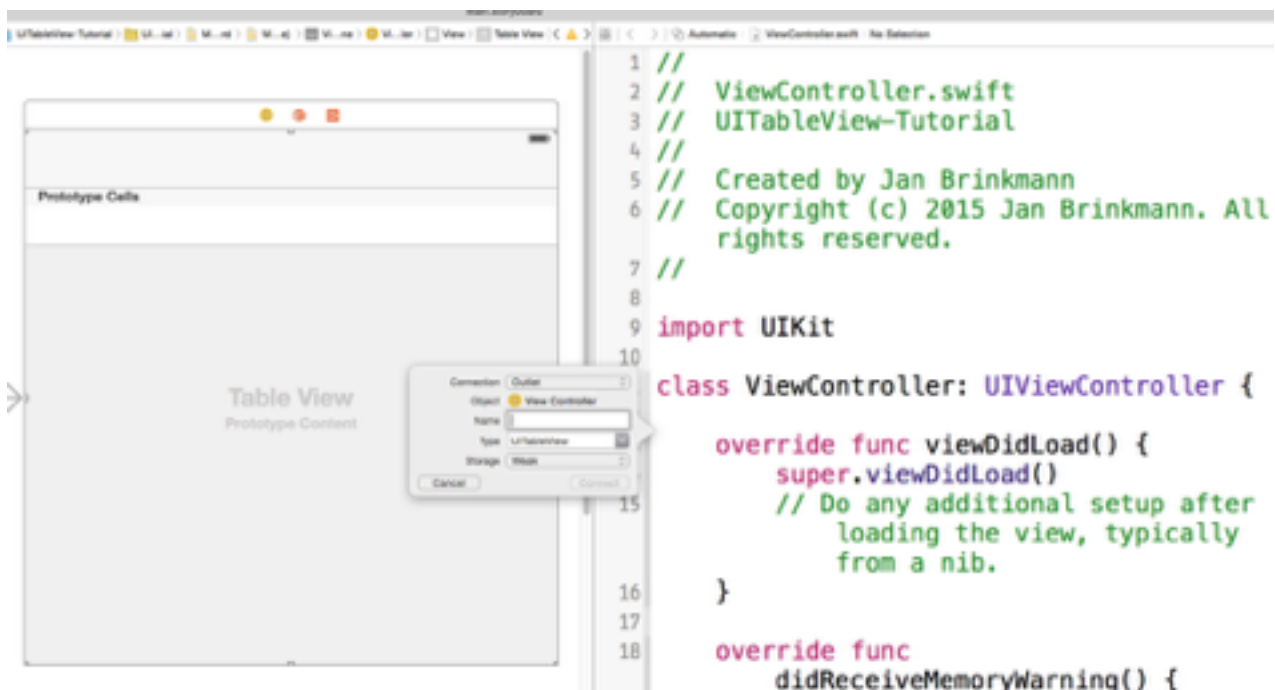


In dieser Ansicht kannst Du nun Outlets erstellen. Das ist denkbar einfach. Klick dazu einfach auf die graue Fläche der Table View, um diese auszuwählen. Anschließend drückst Du die Taste CTRL und hältst sie gedrückt. Nun mit der Maus auf die Table View klicken und nach rechts ziehen. Während dessen bleibt CTRL

weiterhin gedrückt. Wenn Du alles richtig gemacht hast, erscheint ein blauer Feil und deutet die gewünschte Aktion an:



Wichtig: Du musst den Cursor in die Definition der Klasse, also “zwischen” die geschwungenen Klammern, bewegen. Ein blauer Strich und der Hinweis “Insert Outlet or Outlet Connection” erscheinen. Nur dort kannst Du erfolgreich loslassen. Sobald Du dies tust, wird ein kleines Dialogfenster geöffnet. Hier kannst Du den Namen und weitere Eigenschaften festlegen und wählen, ob ein Outlet oder ein Action erstellt werden sollen. In diesem Fall ist das Outlet die richtige Wahl:



Somit muss für den Wert *Connection* Outlet gewählt werden. Achte darauf, dass bei dem Wert *Type* die UITableView steht. Ansonsten erzeugst Du ein Outlet für das falsche View Element. Das passiert schneller als Du glaubst. Die dabei auftretenden Folgefehler zu lösen, deuten nicht immer unmittelbar auf das falsche Element. Was jetzt noch fehlt ist der Name. Den gibst Du an, wie Du magst. Ich arbeite in diesem Beispiel mit der Bezeichnung *tableView*. So heißt das IBOutlet im UITableViewController. Nach der Bestätigung durch einen Klick auf Connect entsteht die neue Eigenschaft *tableView* in der Klasse ViewController.

UITableViewDataSource: Inhalte für die Tabelle

Im iOS-Umfeld wird viel über Delegates und Protokolle gearbeitet. Sie ermöglichen eine saubere Anbindung an externe Klassen, bspw. um wie bei der TableView Inhalte abzurufen.

Wichtig ist in diesem Zusammenhang das UITableViewDataSource Protokoll. Die TableView hat ein spezielles Outlet dataSource. Über dies kann ein Objekt als Datenquelle festgelegt werden. Einzige Voraussetzung: Die Klasse muss zum eben genannten Protokoll kompatibel sein. Und genau dafür sorgen wir nun.

Im ersten Schritt wird die Definition der Klasse ViewController erweitert:

```
class ViewController: UIViewController,
UITableViewDataSource {
//...
```

Das sorgt für eine Fehlermeldung, da wir in diesem Moment die erforderlichen Methoden noch nicht implementiert haben. Das folgt jetzt gleich. Wichtig sind noch ein paar Einträge für die Tabelle. Dazu lege ich einfach ein String-Array mit dem Namen entries als Eigenschaft an:

```
class ViewController: UIViewController,
UITableViewDataSource {
    let entries = ["iOS mit Swift", "Swift Tutorial",
"Swift lernen"]
    //...
```

Mit dem Protokoll müssen wir uns um die fehlenden Methoden kümmern, die Du jetzt implementierst:

Anzahl der Sektionen und Zeilen

Jede UITableView besteht aus Sektionen und Zeilen. Wie viele es sind, wird im Code berechnet. Die folgenden Methoden müssen zur Klasse ViewController hinzugefügt werden:

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

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

Was hier passiert ist fast selbsterklärend. Zunächst wird über `numberOfSectionsInTableView` die Anzahl der Sektionen bestimmt. Bei einer einfachen Tabelle ist dies statisch der Wert "1". Streng genommen musst Du Dich gar nicht mit der Methode befassen, wenn nur eine Sektion existiert. Der Wert ist dann automatisch eins.

Spannender wird nun die Methode *tableView*. Wie Du im ersten Abschnitt gesehen hast, gibt es sie vielen verschiedenen Ausführungen. Wird der Parameter *numberOfRowsInSection* wie hier übergeben, wird nach der Anzahl der Inhalte in der übergebenen Sektion gefragt. Dazu zählst Du einfach die Elemente im Array und gibst den Wert zurück.

Inhalte für Zeilen zurückgeben

Inhalte werden nicht als Array an die TableView übergeben. Stattdessen fragt die UITableView nach den konkreten Inhalten für einzelne Zeilen. Dabei bezieht sich die Anfrage immer auf einen NSIndexPath. Daraus geht hervor für welche Sektion und für welche Zeile Inhalte benötigt werden. Da wir ohnehin nur eine Sektion haben, ignorieren wir dies an der Stelle. Wichtig ist für uns nur der Wert row im Parameter indexPath. Den können wir als Index für unser Array weiterverwenden und weiterleiten:

```
func tableView(tableView: UITableView,
    cellForRowAtIndexPath indexPath: NSIndexPath) ->
UITableViewCell {

    let cell =
tableView.dequeueReusableCellWithIdentifier("myTableViewCell",
    forIndexPath: indexPath) as! UITableViewCell
    cell.textLabel?.text = entries[indexPath.row]

    return cell
}
```

Wie oben beschrieben wird hier ein Objekt aus der Queue gelesen. In jeder UITableViewCell gibt es das UILabel mit dem Namen textLabel. Genau dies nutzen wir und setzen die Eigenschaft Text auf den Eintrag in unserem Array. Anschließend geben wir die Variable cell zurück.

Wenn Du jetzt Deine App startest, solltest Du eine Tabelle mit den Werten aus dem Array entries sehen.

Den kompletten Quellcode findest Du hier im [Github Projekt](#).