

App-Entwicklung für iOS und OS X

SS 2016
Stephan Gimbel



UITextField

Wie ein UILabel, nur editierbar

Zum schreiben auf einem iPhone in einem zweiten UI (Keyboard ist klein).

Mehr ein mainstream UI Element auf einem iPad.

Nicht vom UI im Simulator täuschen lassen (da das physikalische Keyboard zum editieren verwendet werden kann).

Attributed Text, Text Farbe, Alignment, Schriftart, etc. wie bei einem UILabel.

Keyboard taucht auf, wenn UITextField zum “First Responder” wird

Passiert automatisch wenn der User darauf tapped.

Oder wir machen es zum First Responder durch Aufruf von
becomeFirstResponder.

Um das Keyboard auszublenden, Aufruf von resignFirstResponder für
das UITextField.

UITextField

Verwendung eines Delegate mit Return Key, etc.

```
func textFieldShouldReturn(sender: UITextField)
```

Oft wird in dieser Funktion sender.resignFirstResponder ausgeführt.

UITextField ist ein UIControl

Es kann target/action verwendet werden um über Änderungen zu benachrichtigen.

Wie bei einem Button, existieren verschiedene UIControlEvents um einen Action auszuführen.

Rechts-click auf UITextField im Storyboard um die verschiedenen Optionen zu sehen.

Keyboard

Art des Keyboards

Setzen des Property definiert im UITextInputTraits Protokoll (welches UITextField implementiert)

```
var UITextAutocapitalizationType autocapitalizationType // Wörter, Sätze, etc.  
var UITextAutocorrectionType autocorrectionType // ja oder nein  
var UIReturnKeyType returnKeyType // Go, Search, Google, Done, etc.  
var BOOL secureTextEntry // für Passwörter z.B.  
var UIKeyboardType keyboardType // ASCII, URL, PhonePad, etc.
```

UITextField ist ein UIControl

Es kann target/action verwendet werden um über Änderungen zu benachrichtigen.

Wie bei ein Button, existieren verschiedene UIControlEvents um einen Action auszuführen.

Rechts-click auf UITextField im Storyboard um die verschiedenen Optionen zu sehen.

Keyboard

Das Keyboard taucht über den anderen Views auf

Daher muss die View Positionierung angepasst werden (vor allem um das Text Field selbst sichtbar zu halten).

Wird mittels `UIKeyboard{Will,Did}{Show,Hide}Notifications` gemacht welche von UIWindow geschickt werden.

Wir haben noch nicht über NSNotifications gesprochen, ist aber ziemlich einfach.

Wir registrieren eine Funktion die aufgerufen wird, wenn ein bestimmter "Event" auftritt...

```
NSNotificationCenter.defaultCenter().addObserver(self,  
                                              selector: #selector(ClassName.theKeyboardAppeared),  
                                              name: UIKeyboardDidShowNotification,  
                                              object: view.Window)
```

Der Event ist `UIKeyboardDidShowNotification`.

Das Object ist der Auslöser des Events (MVC's View's Window)

`fun theKeyboardAppeared(notification: NSNotification)` wird aufgerufen sobald der Event auftritt.

`notification.userInfo` hat weitere Details.

`UITableViewController` "hört zu" und scrollt Table automatisch, wenn er ein `UITextField` hat.

UITextField

Andere UITextField Properties

```
var clearsOnBeginEditing: Bool  
var adjustFontSizeToFitWidth: Bool  
var minimumFontSize: CGFloat // immer setzen, wenn  
// adjustFontSizeToFitWidth  
// gesetzt wurde  
var placeholder: String // in Grau zeichnen wenn leer  
var background/disabledBackground: UIImage  
var defaultTextAttributes: Dictionary // für ganzen Text  
Oft wird in dieser Funktion sender.resignFirstResponder ausgeführt.
```

Andere UITextField Funktionalität

UITextFields haben left und right Overlays.

Das Layout kann detailliert kontrolliert werden (Border, Left/Right View, Clear Button).

Andere Keyboard Funktionalität

Keyboards können Accessory Views haben, die über dem Keyboard auftauchen (Custom Toolbar, etc.)

```
var inputAccessoryView: UIView // UITextField Funktion
```

Table Views

UITableView ist eine wichtige Klasse um Daten in einem Table darzustellen
Ein-dimensionaler Table.

Subclass von UIScrollView.

Kann statisch oder dynamisch sein (z.B. eine Liste von Items).

Viele Möglichkeiten zur Anpassung mittels dataSource und delegate Protokoll.

Sehr effizient, auch bei großen Datenmengen.

Mehrdimensionale Tables

Normalerweise mittels UINavigationController mit mehreren MVCs wo ein View ein UITableView ist.

Arten von UITableViews

Plan oder Grouped

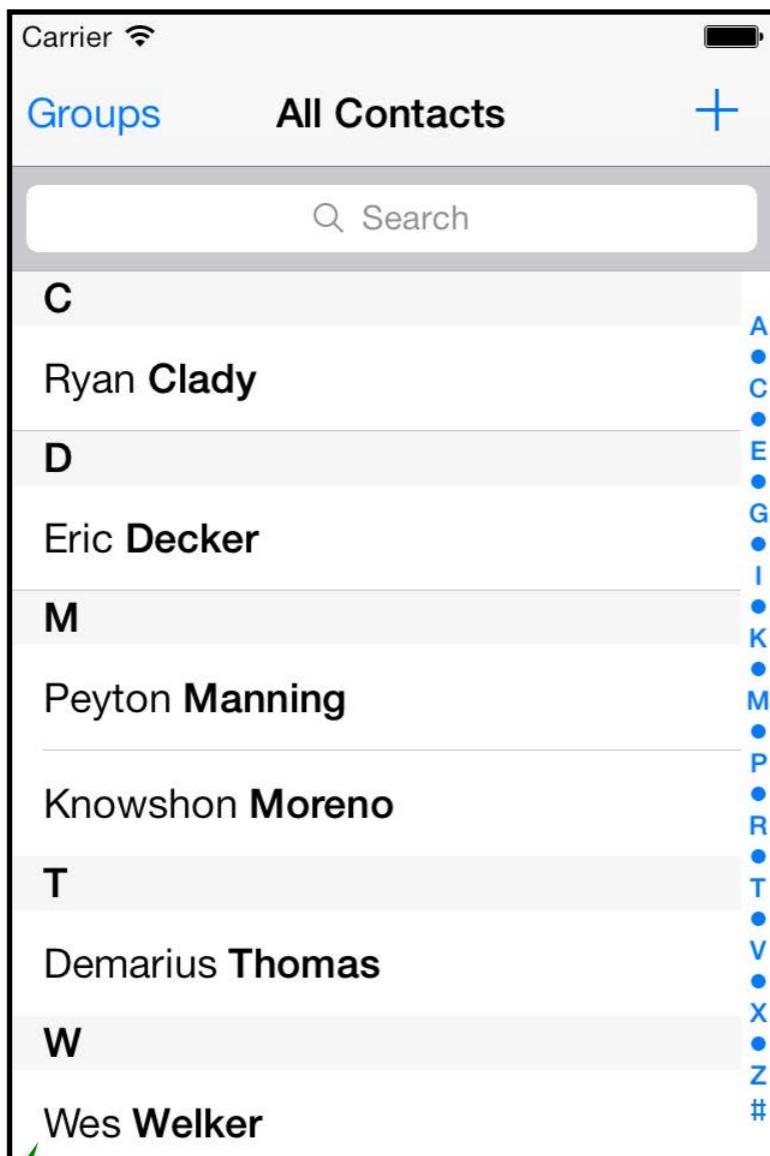
Static oder Dynamic

Mit Sections oder ohne

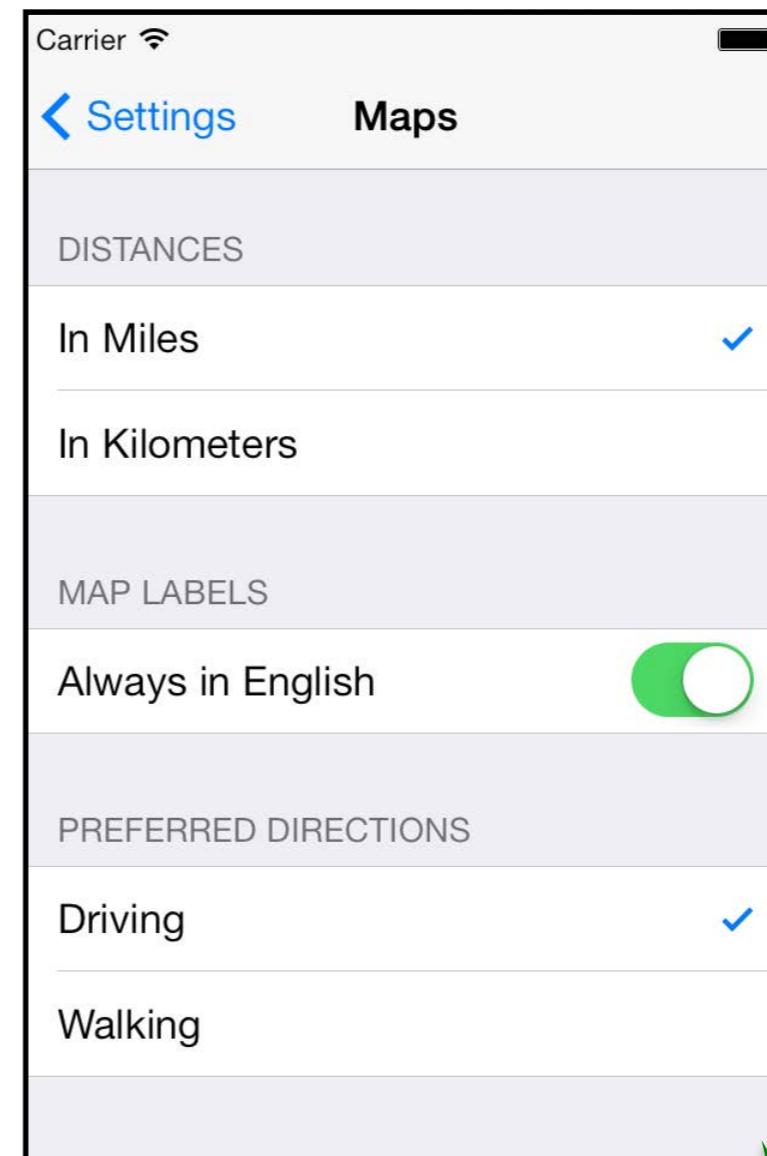
Unterschiedliche Formate für jede Zeile im Table (inkl. vollständig customized)

Table Views

`UITableViewStyle.Plain`



`.Grouped`



Dynamic (List) &
Plain (ungrouped)

Static &
Grouped

Table Views

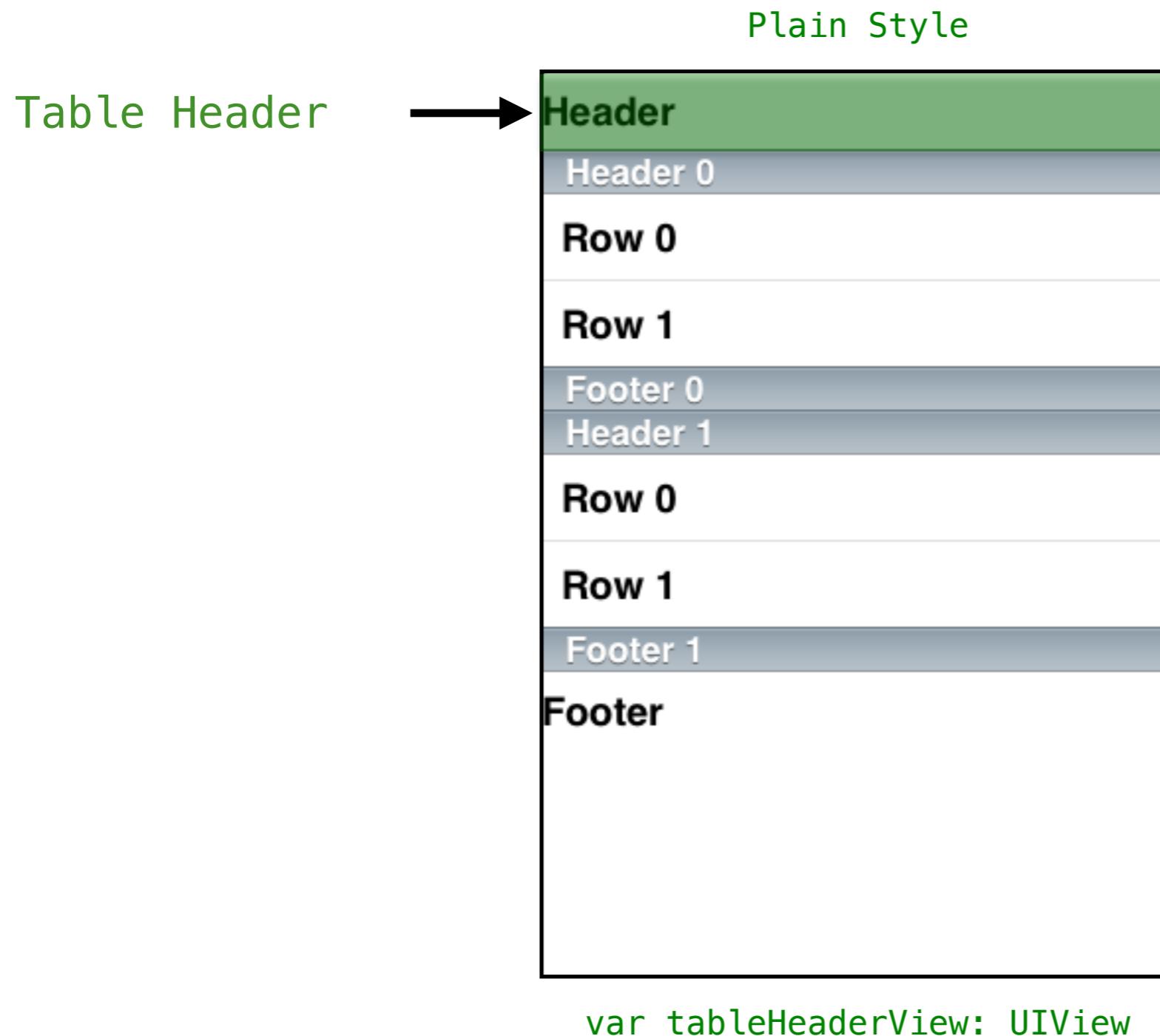


Table Views

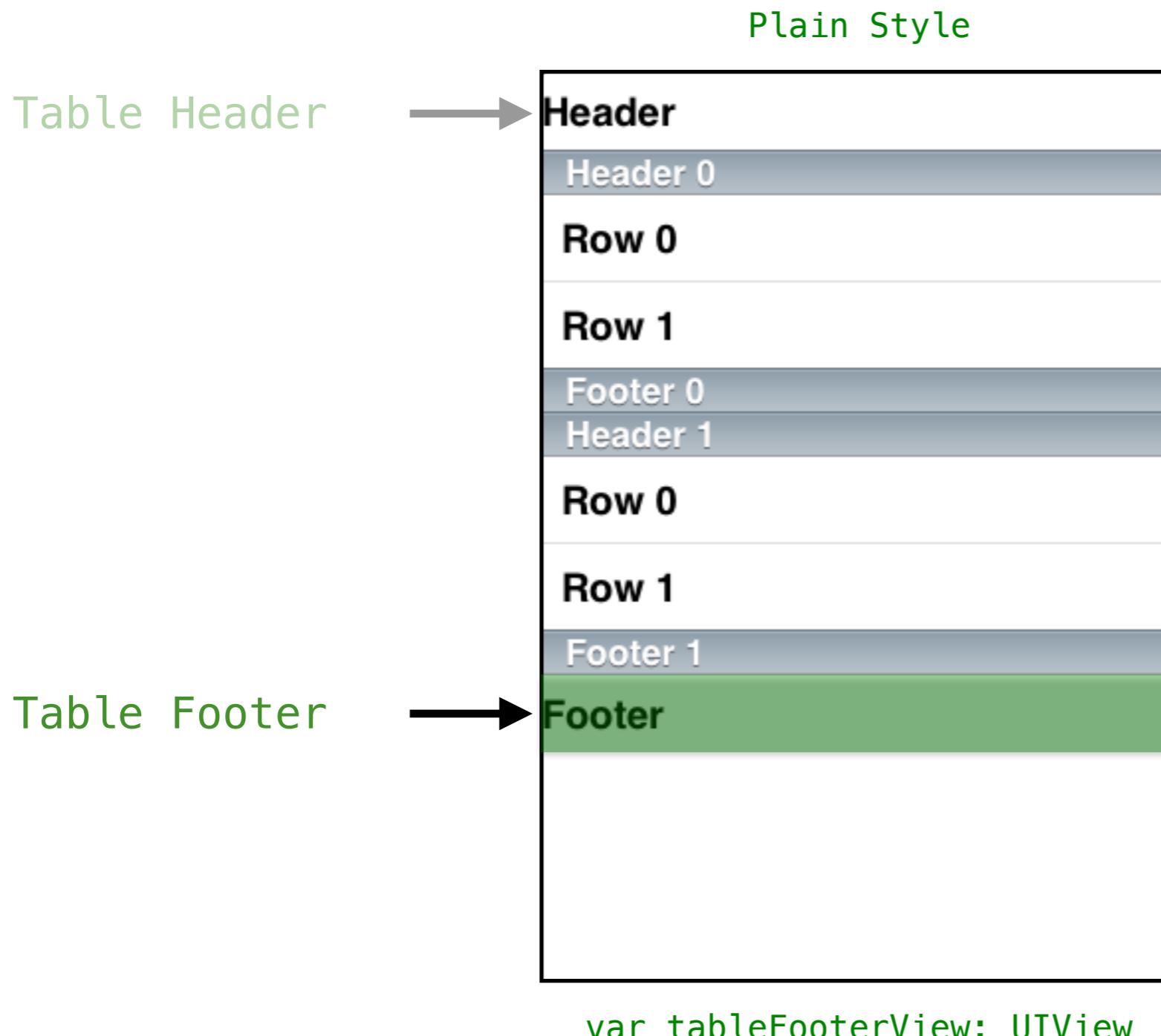


Table Views

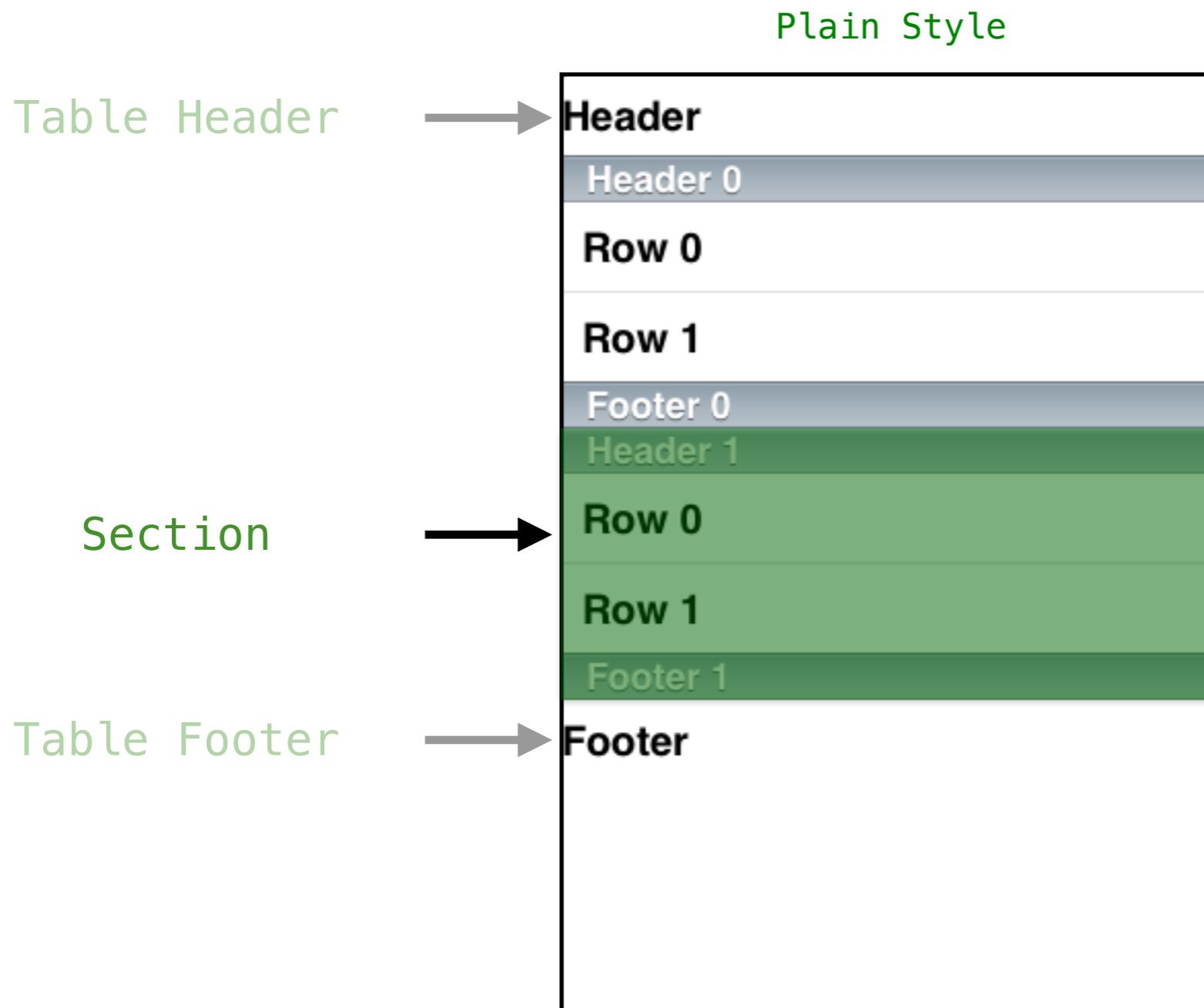
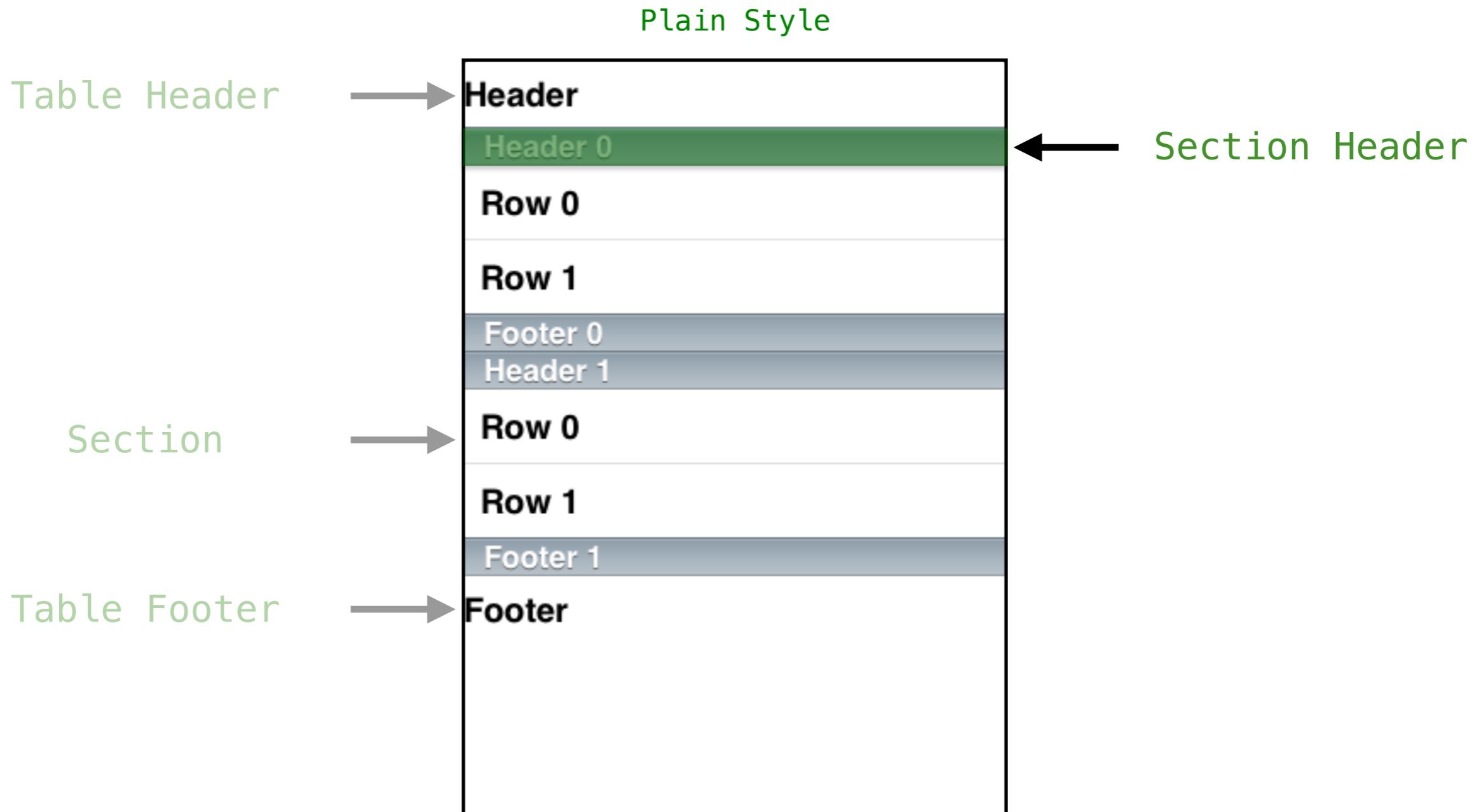
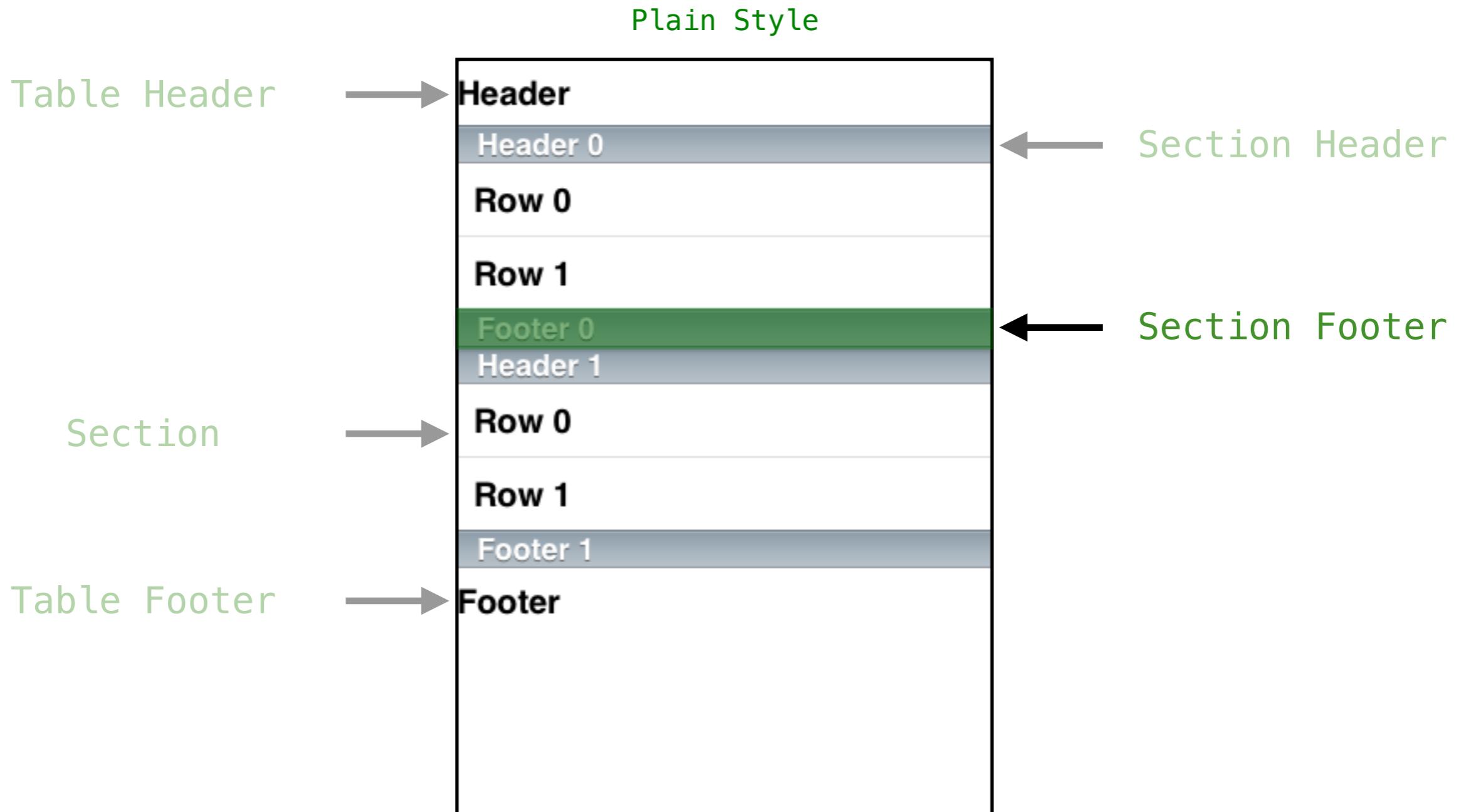


Table Views



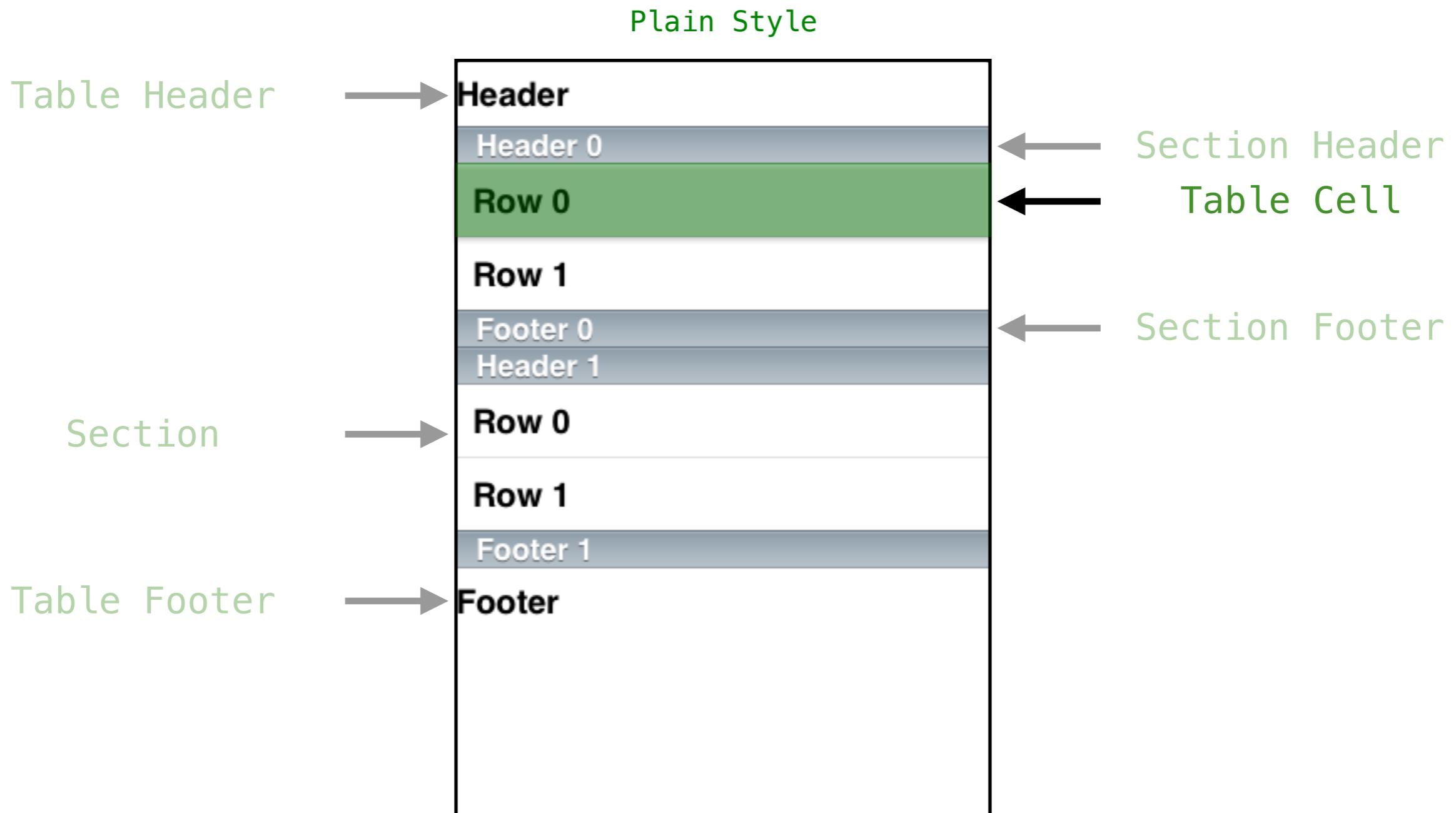
`UITableViewDataSource's tableView(UITableView, titleForHeaderInSection: Int)`

Table Views



`UITableViewDataSource's tableView(UITableView, titleForFooterInSection: Int)`

Table Views



`UITableViewDataSource's tableView(UITableView, cellForRowAtIndexPath: NSIndexPath)`

Table Views

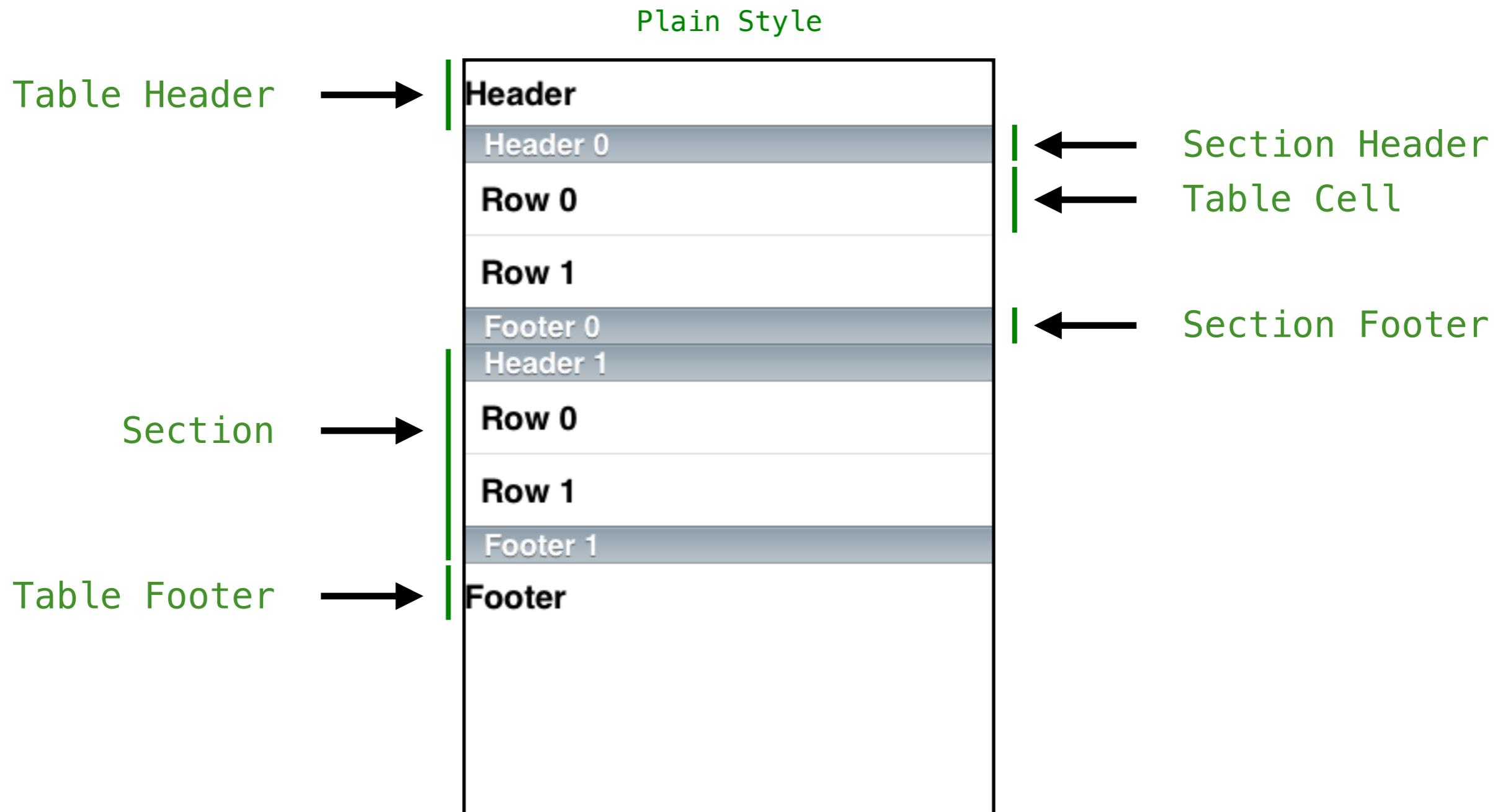


Table Views

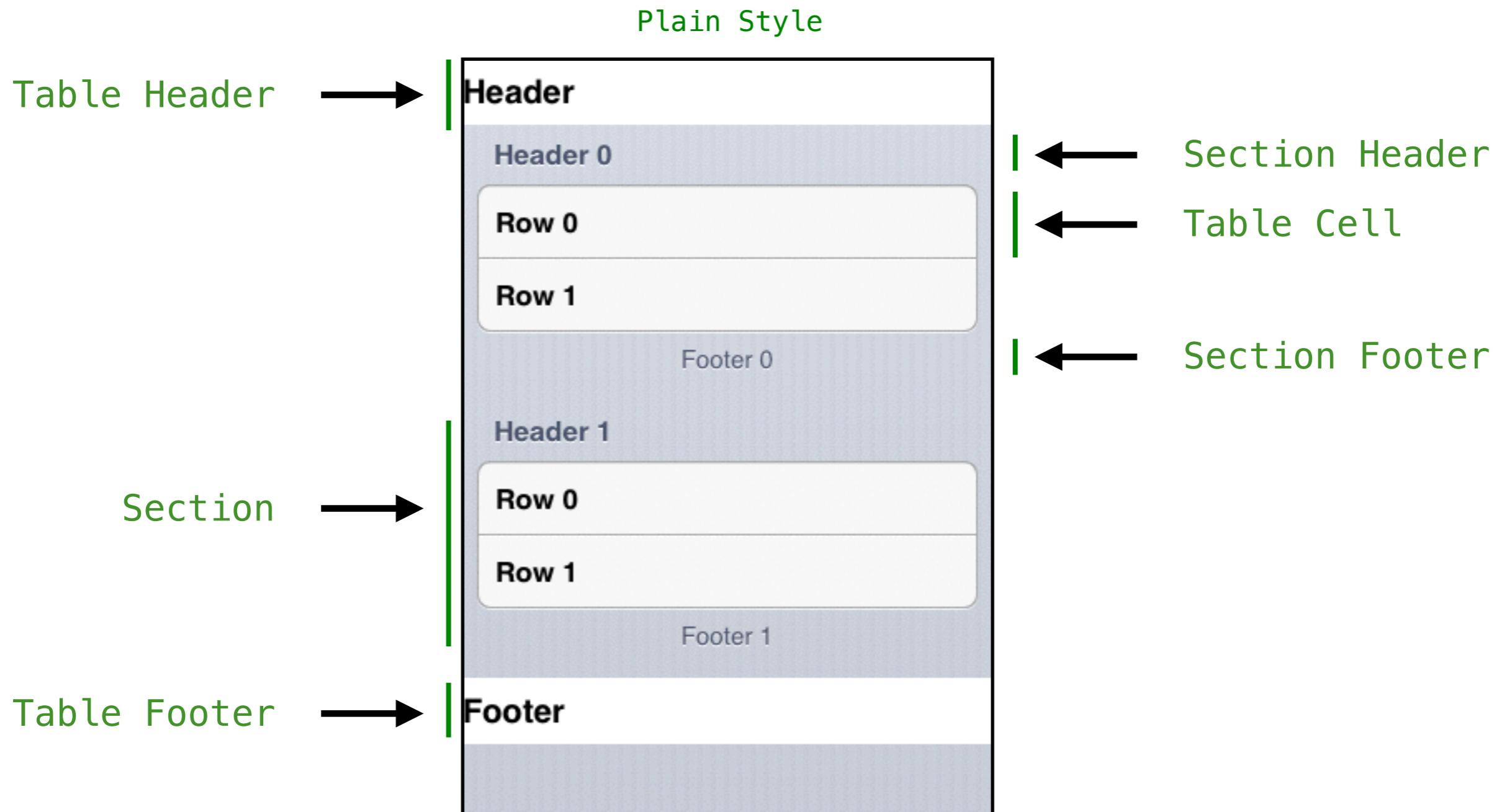
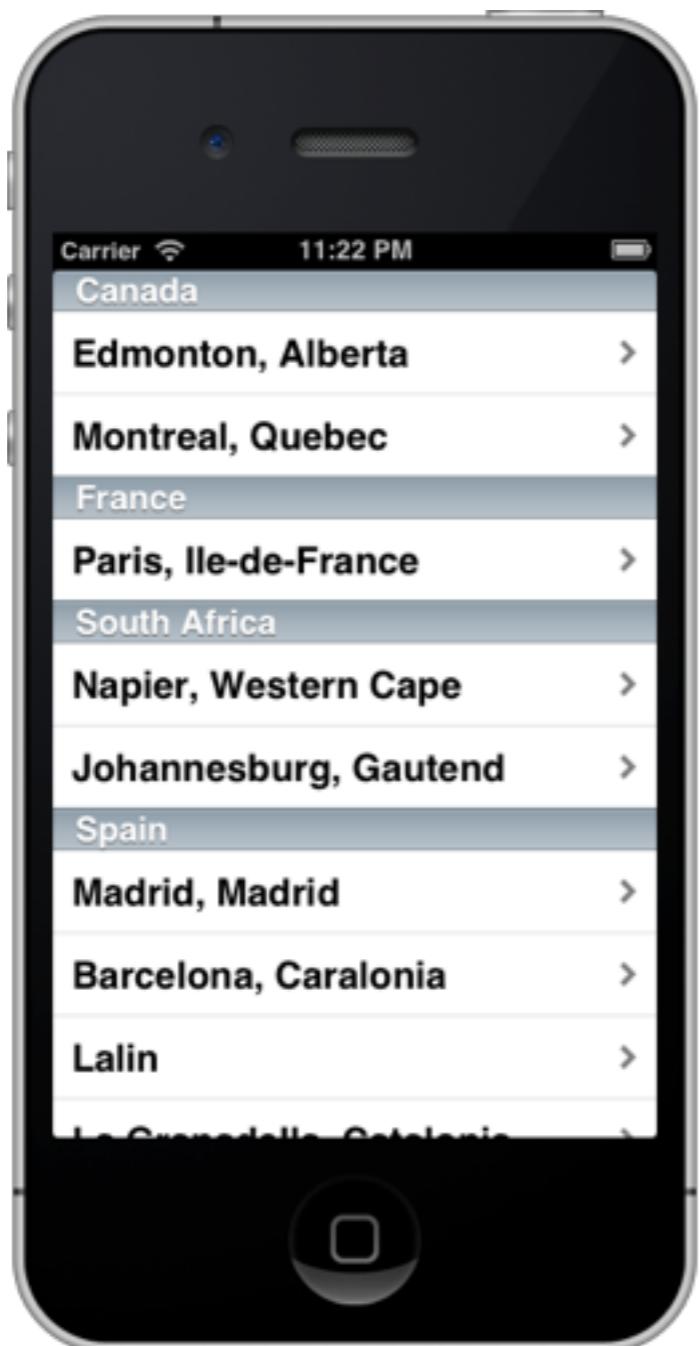
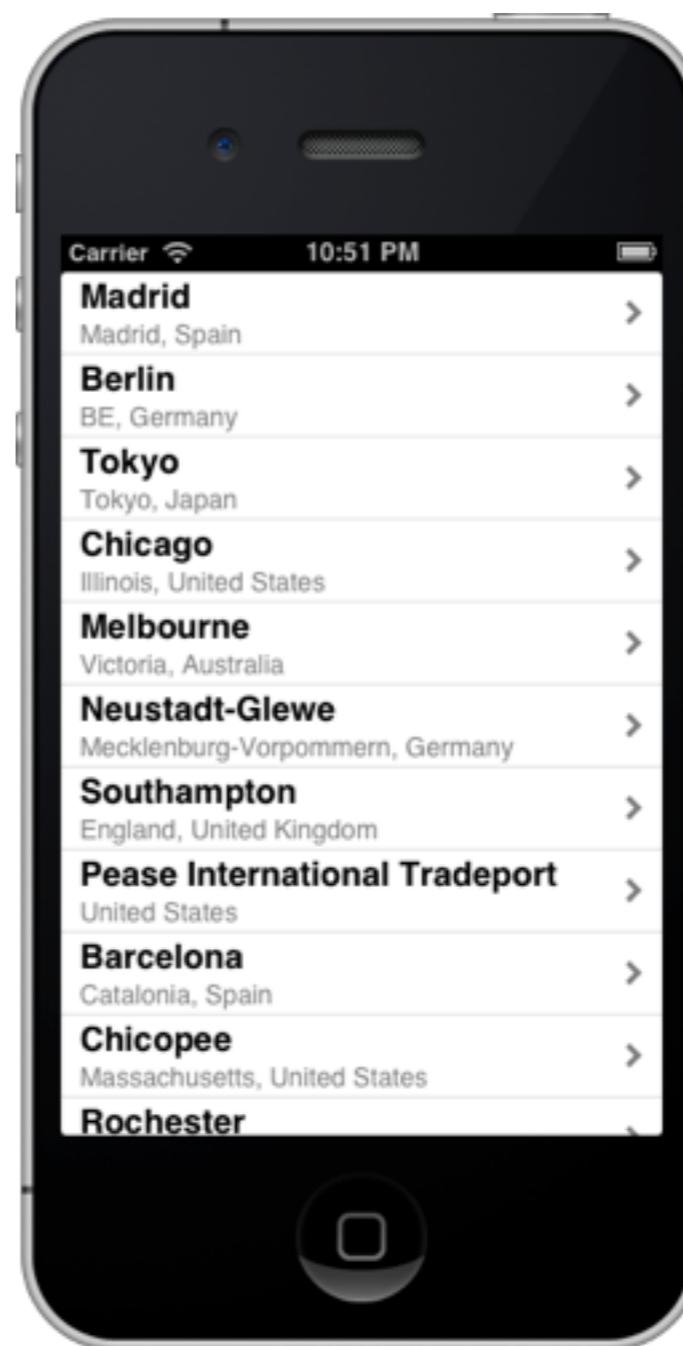


Table Views

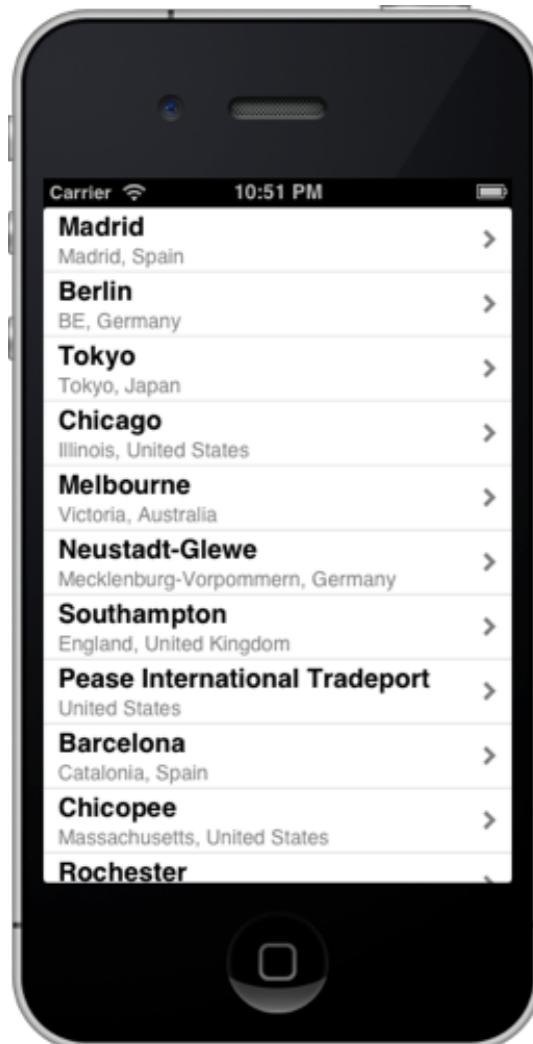


Sections



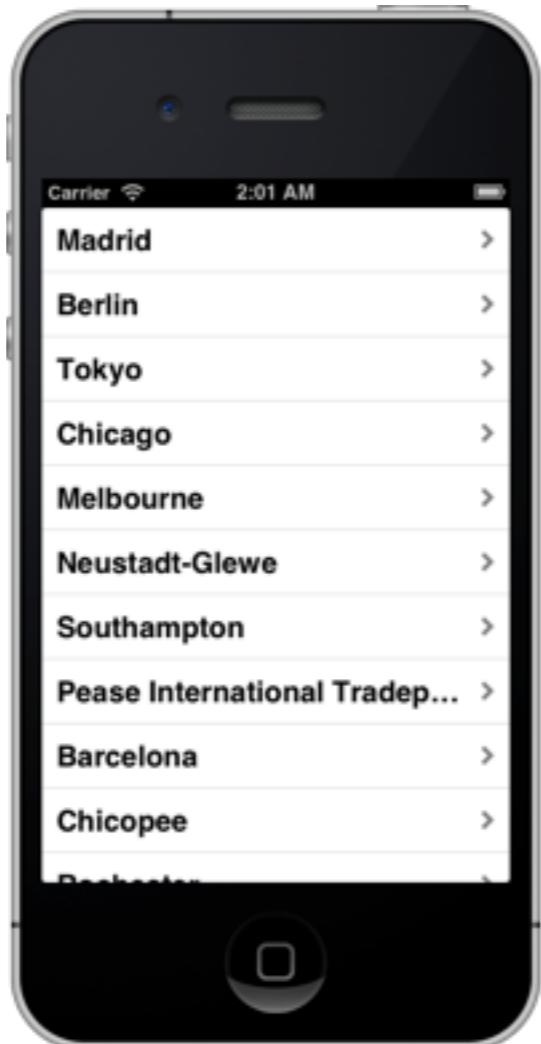
Keine Sections

Table Views



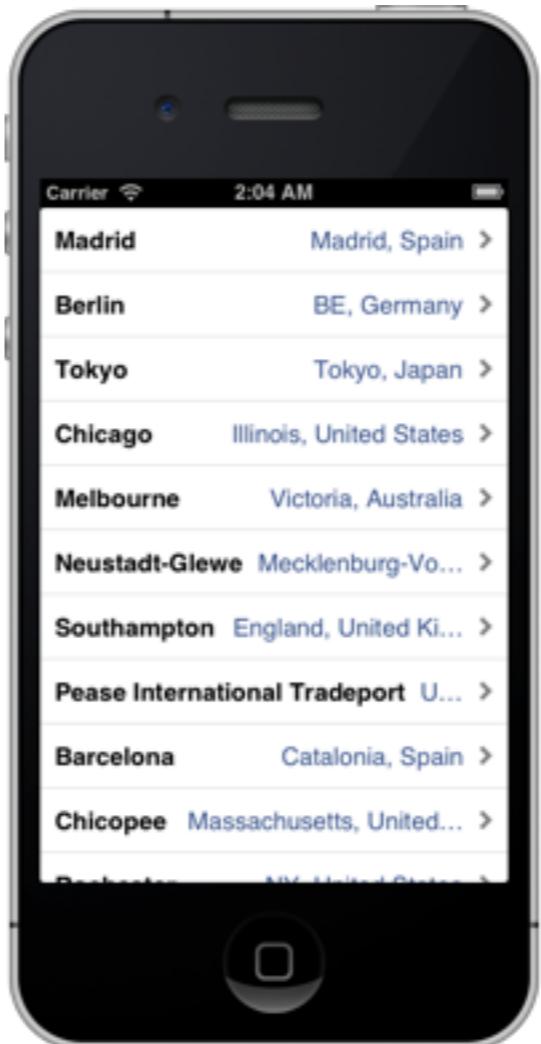
Subtitle

`UITableViewCellStyle.Subtitle`



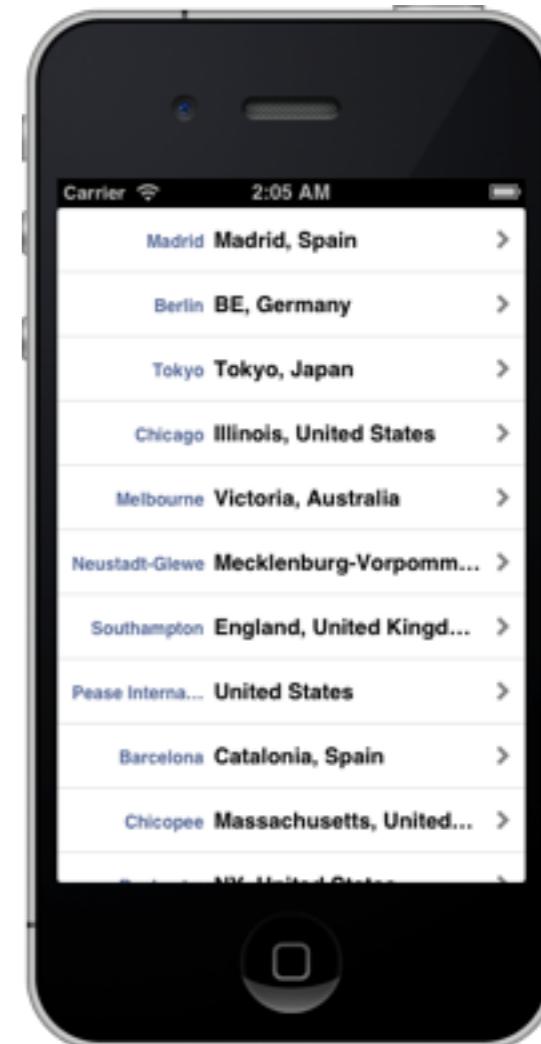
Basic

`.Default`



Right Detail

`.Value1`



Left Detail

`.Value2`

Table Views

Die Klasse `UITableViewController` stellt einen `UITableView` in einem MVC zur Verfügung.

The screenshot shows the Xcode interface with a storyboard scene selected. The storyboard preview shows a single table view. The Xcode toolbar at the top indicates the project is named "TableViewExample.xcodeproj" and the storyboard is "MainStoryboard_iPhone.storyboard". The bottom of the screen shows the Xcode status bar with various icons. On the right side, the "Quick Help" panel is open for the `UIViewController` class, providing a detailed description of its role in iOS app development. The "Objects" library on the far right lists several controller types: View Controller, Table View Controller, Collection View Controller, and Navigation Controller.

Table View Example View Controller Scene > Table View Example View Controller

TableViewExampleView
Controller : UIViewController

Description The `UIViewController` class provides the fundamental view-management model for all iOS apps. You rarely instantiate `UIViewController` objects directly. Instead, you instantiate subclasses of the `UIViewController` class based on the specific task each subclass performs. A view controller manages a set of views that make up a portion of your app's user interface. As part of the controller layer of your app, a view controller coordinates its efforts with model objects and other controller objects—including other view controllers—so your app presents a single coherent user interface.

Availability iOS (2.0 and later)

Declared `UIViewController.h`

Reference [UIViewController Class Reference](#)

Guides [View Controller Catalog for iOS](#), [View Controller Programming Guide for iOS](#)

Objects

View Controller – A controller that supports the fundamental view-management model in iPhone OS.

Table View Controller – A controller that manages a table view.

Collection View Controller – A controller that manages a collection view.

Navigation Controller – A controller that manages a navigation stack.

Table Views

The screenshot shows the Xcode interface with a storyboard project named "TableViewExample.xcodeproj". The storyboard scene is titled "Table View Example View Controller Scene". A UITableView is placed within a UIViewController. A callout bubble highlights the UITableView and contains the text: "Am sinnvollsten wenn der UITableView den gesamten self.view füllt. (tatsächlich ist self.view in einem UITableViewController der UITableView)". Another callout bubble highlights the UIViewController and contains the text: "Die Klasse UITableViewController stellt einen UITableView in einem MVC zur Verfügung."

Die Klasse UITableViewController stellt einen UITableView in einem MVC zur Verfügung.

**Am sinnvollsten wenn der UITableView den gesamten self.view füllt.
(tatsächlich ist self.view in einem UITableViewController der UITableView)**

Table View Example View Controller Scene > Table View Example View Controller

TableViewExampleView
Controller : UIViewController

Description The UIViewController class provides the fundamental view-management model for all iOS apps. You rarely instantiate UIViewController objects directly. Instead, you instantiate subclasses of the UIViewController class based on the specific task each subclass performs. A view controller manages a set of views that make up a portion of your app's user interface. As part of the controller layer of your app, a view controller coordinates its efforts with model objects and other controller objects—including other view controllers—so your app presents a single coherent user interface.

Availability iOS (2.0 and later)

Declared UIViewController.h

Reference UIViewController Class Reference

Guides View Controller Catalog for iOS, View Controller Programming Guide for iOS

Objects

View Controller – A controller that supports the fundamental view-management model in iPhone OS.

Table View Controller – A controller that manages a table view.

Collection View Controller – A controller that manages a collection view.

Navigation Controller – A controller that manages a navigation stack.

Table Views

The screenshot shows the Xcode interface with a storyboard project named "TableViewExample.xcodeproj". The storyboard contains a single scene titled "Table View Example View Controller Scene" which displays a blank white view. A callout bubble from the top-left of this view contains the text: "Die Klasse UITableViewController stellt einen UITableView in einem MVC zur Verfügung." To the right of the storyboard, the Xcode interface includes the Quick Help panel for the UITableViewController class, which provides a detailed description of its role in the iOS view management model.

Die Klasse UITableViewController stellt einen UITableView in einem MVC zur Verfügung.

Am sinnvollsten wenn der UITableView den gesamten self.view füllt.
(tatsächlich ist self.view in einem UITableViewController der UITableView)

Kann durch Drag&Drop in das Storyboard hinzugefügt werden.

Table Views

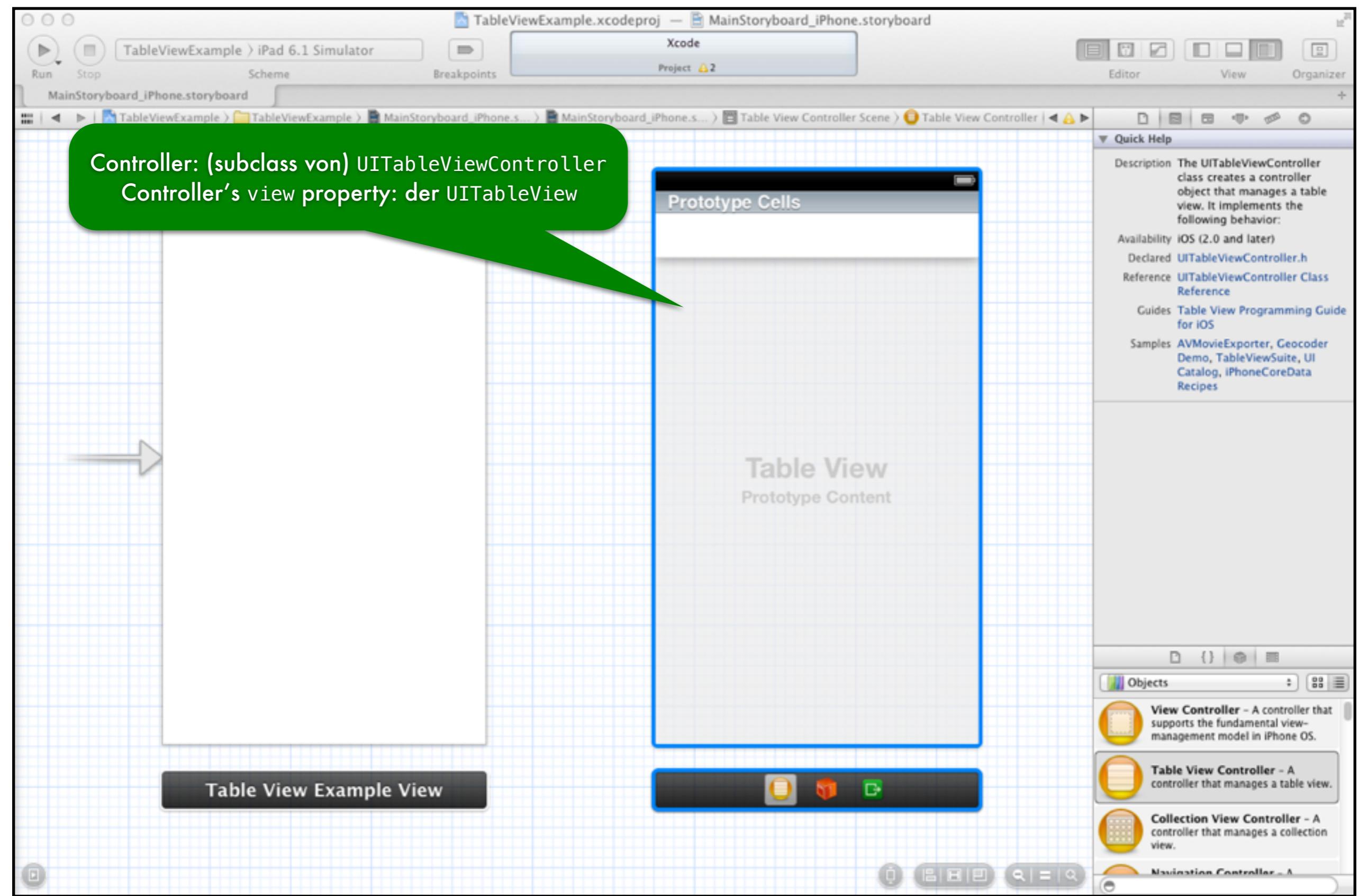


Table Views

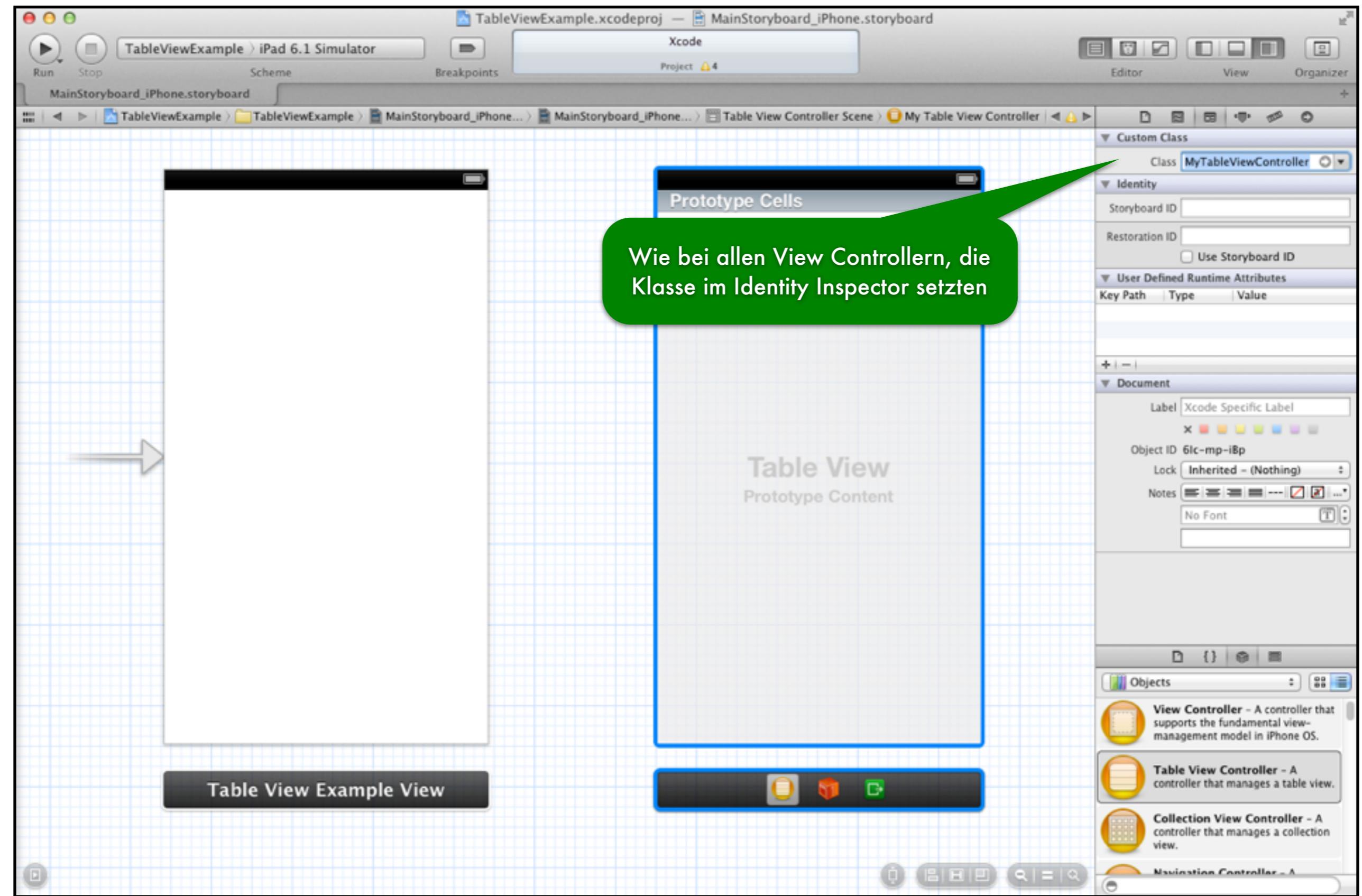


Table Views

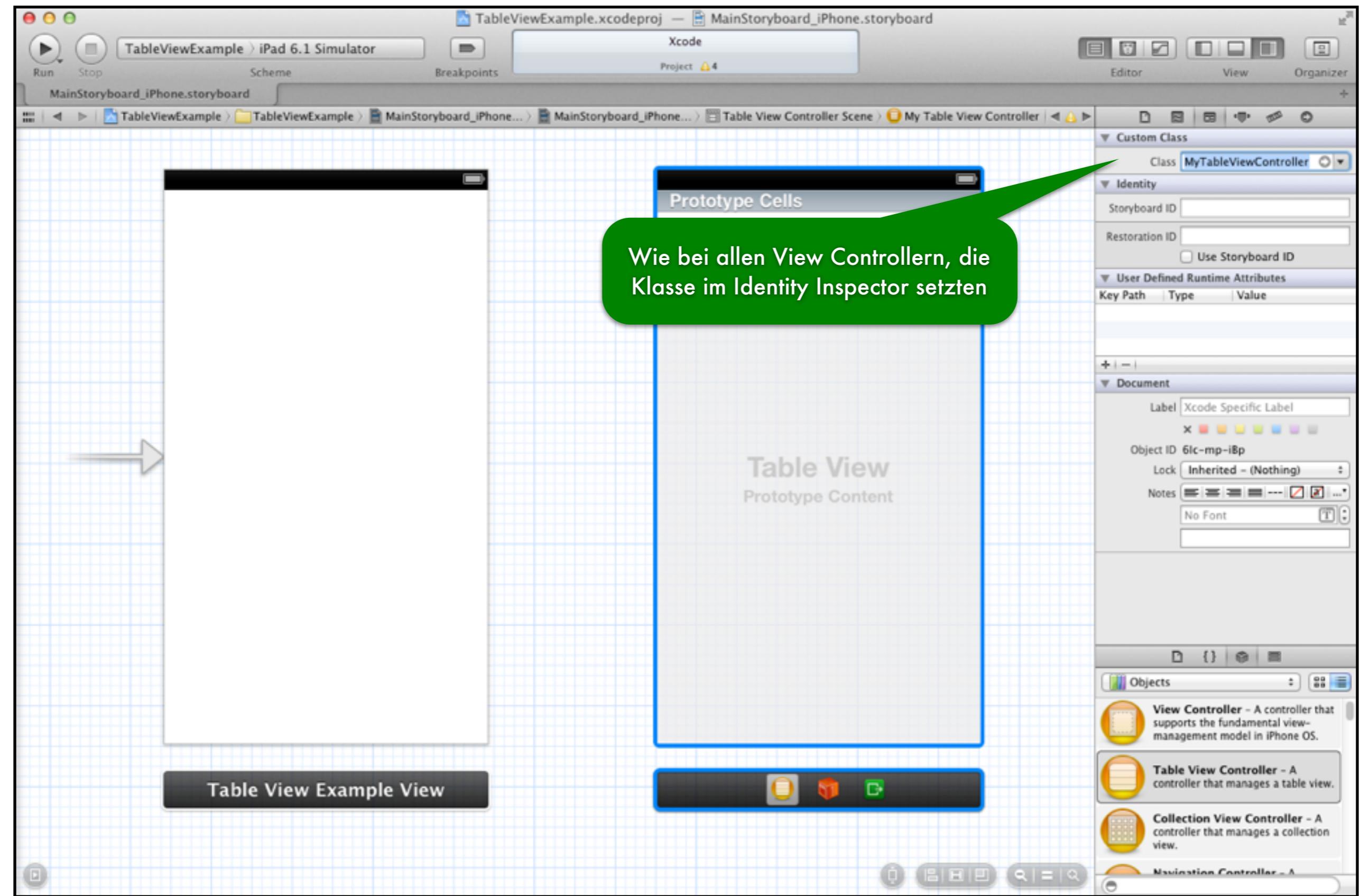


Table Views

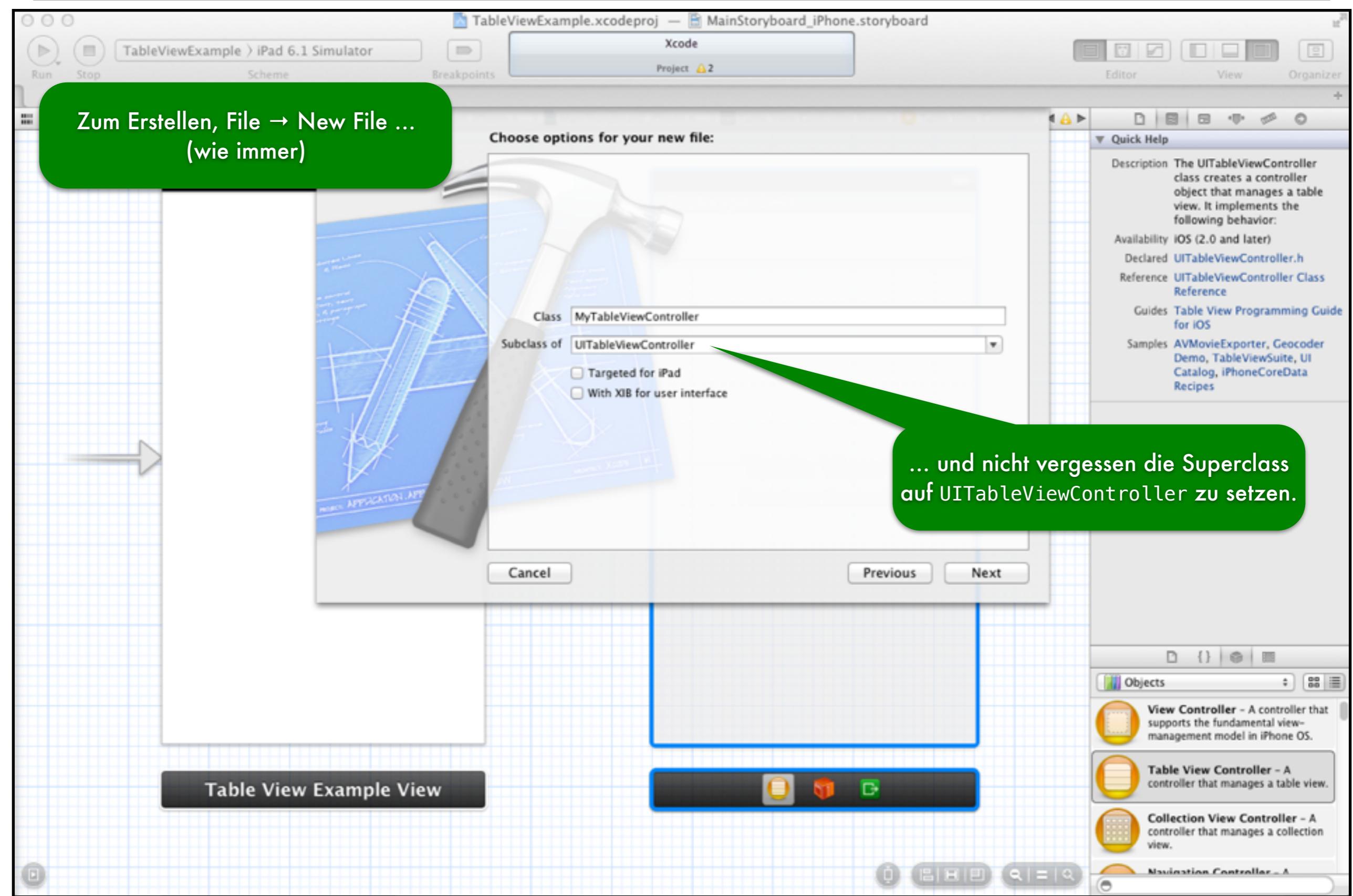


Table Views

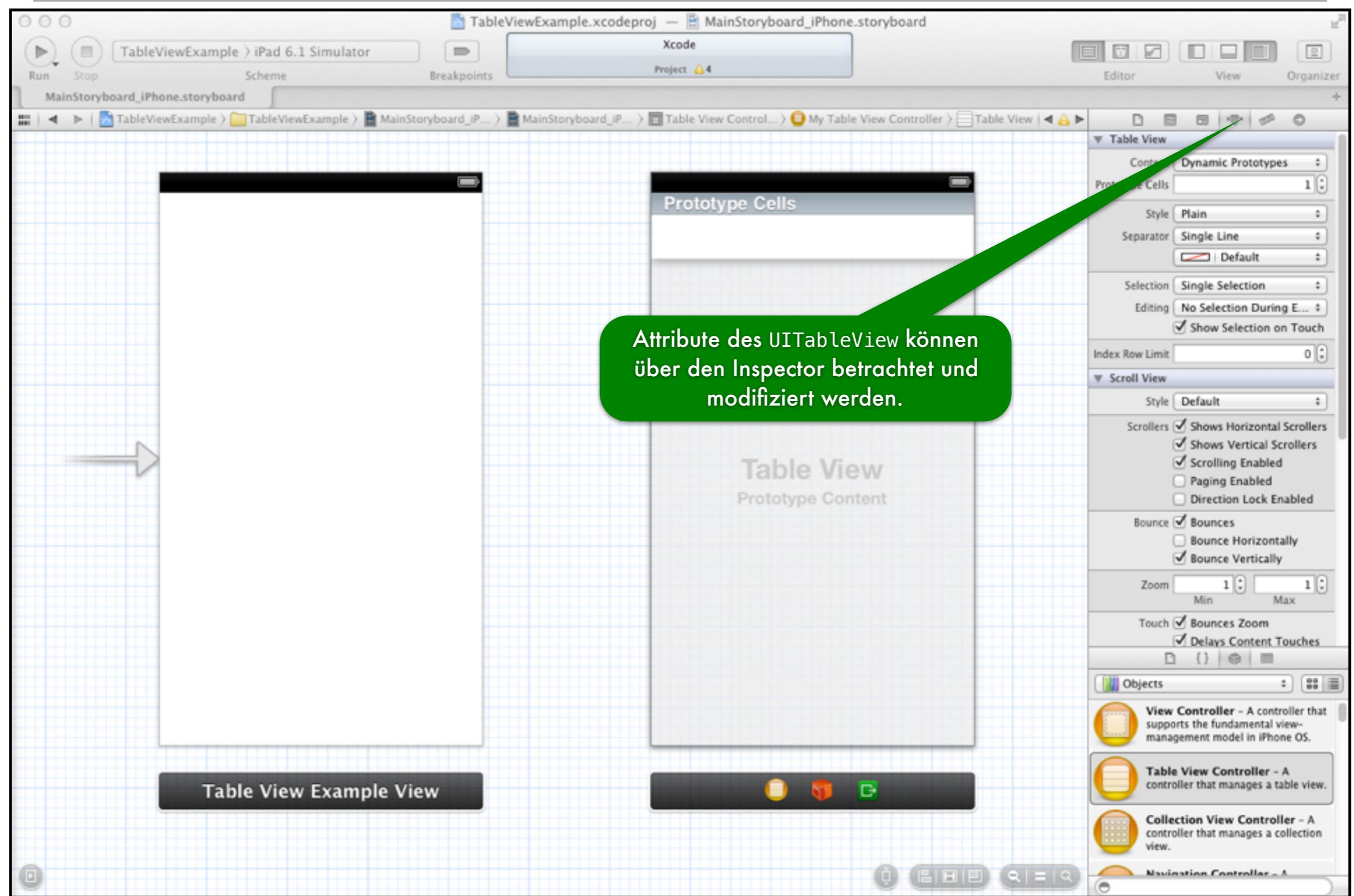


Table Views

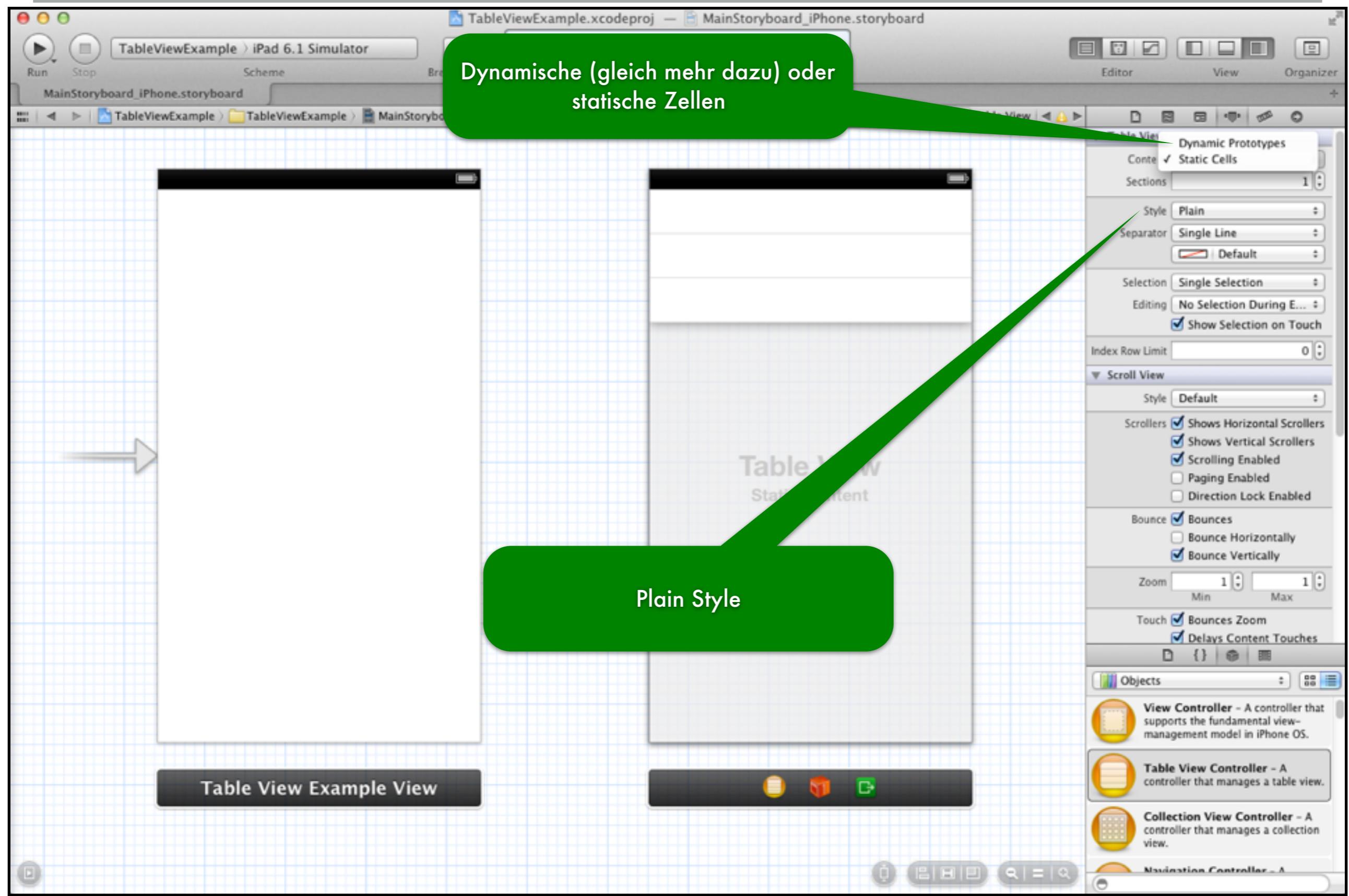


Table Views

The screenshot shows the Xcode interface with a storyboard file named MainStoryboard_iPhone.storyboard open. A green callout bubble points from the text "Grouped Style" to the "Style" setting in the Attributes Inspector, which is set to "Grouped". Another green callout bubble points from the text "Statisch bedeutet, dass die Zellen nur im Storyboard erstellt werden. Können aber beliebig editiert werden, inkl. Dragging von Buttons, Labels, etc. in die Zelle (und hooking von Outlets in den Controller)" to the "Content" section of the Attributes Inspector, which is set to "Static Cells". The storyboard preview shows a table view with three static cells. The bottom navigation bar contains icons for "Table View Example View", "File", "Edit", "Run", "Stop", and "Breakpoints". The bottom toolbar includes icons for device orientation, search, and other development tools.

Grouped Style

Statisch bedeutet, dass die Zellen nur im Storyboard erstellt werden.
Können aber beliebig editiert werden, inkl. Dragging von Buttons,
Labels, etc. in die Zelle (und hooking von Outlets in den Controller)

Table View Example View

Table View
Static Content

TableViewExample.xcodeproj — MainStoryboard_iPhone.storyboard

Editor View Organizer

Table View

Content Static Cells

Sections 1

Style Grouped

Separator Single Line Etched

Selection Single Selection

Editing No Selection During E...

Show Selection on Touch

Index Row Limit 0

Scroll View

Style Default

Scrollers Shows Horizontal Scrollers

Shows Vertical Scrollers

Scrolling Enabled

Paging Enabled

Direction Lock Enabled

Bounce Bounces

Bounce Horizontally

Bounce Vertically

Zoom 1 Min 1 Max

Touch Bounces Zoom

Delays Content Touches

Objects

View Controller - A controller that supports the fundamental view-management model in iPhone OS.

Table View Controller - A controller that manages a table view.

Collection View Controller - A controller that manages a collection view.

Navigation Controller - A

Table Views

The screenshot shows the Xcode interface with a storyboard project named 'TableViewExample.xcodeproj'. In the storyboard, there is a 'Table View' element with a single 'Prototype Content' cell. A green callout bubble points from the text to this prototype cell. Another green callout bubble points from the text to the 'Table View' element in the storyboard.

Ein interessanterer Table ist der dynamische...

... der meist nur im Plain Style verwendet wird.

Table View Example View

MainStoryboard_iPhone.storyboard

Editor View Organizer

TableViewExample > iPad 6.1 Simulator Scheme Breakpoints

TableViewExample > MainStoryboard_iPhone.storyboard

Table View

- Content Dynamic Prototypes
- Prototype Cells 1
- Style Plain
- Separator Single Line
- Selection Single Selection
- Editing No Selection During E...
- Show Selection on Touch
- Index Row Limit 0

Scroll View

- Style Default
- Scrollers Shows Horizontal Scrollers
- Shows Vertical Scrollers
- Scrolling Enabled
- Paging Enabled
- Direction Lock Enabled
- Bounce Bounces
- Bounce Horizontally
- Bounce Vertically
- Zoom 1 Min 1 Max
- Touch Bounces Zoom
- Delays Content Touches

Objects

- View Controller - A controller that supports the fundamental view-management model in iPhone OS.
- Table View Controller - A controller that manages a table view.
- Collection View Controller - A controller that manages a collection view.
- Navigation Controller - A

Table Views

Die Zellen dienen nun als Template und werden beliebig oft wiederholt, abhängig davon wie vielen Zeilen benötigt werden um die Daten im Model des MVC anzuzeigen.

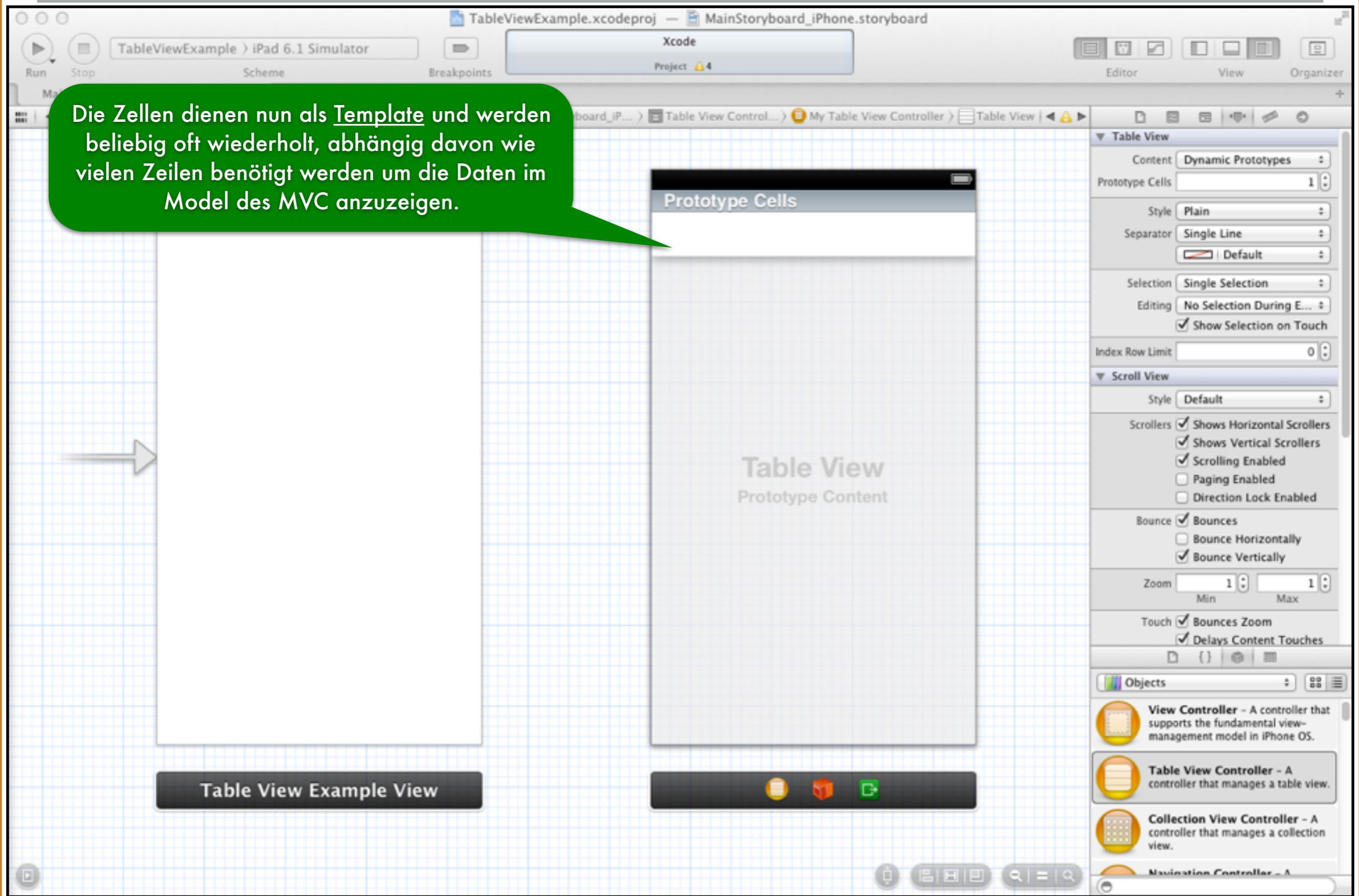


Table Views

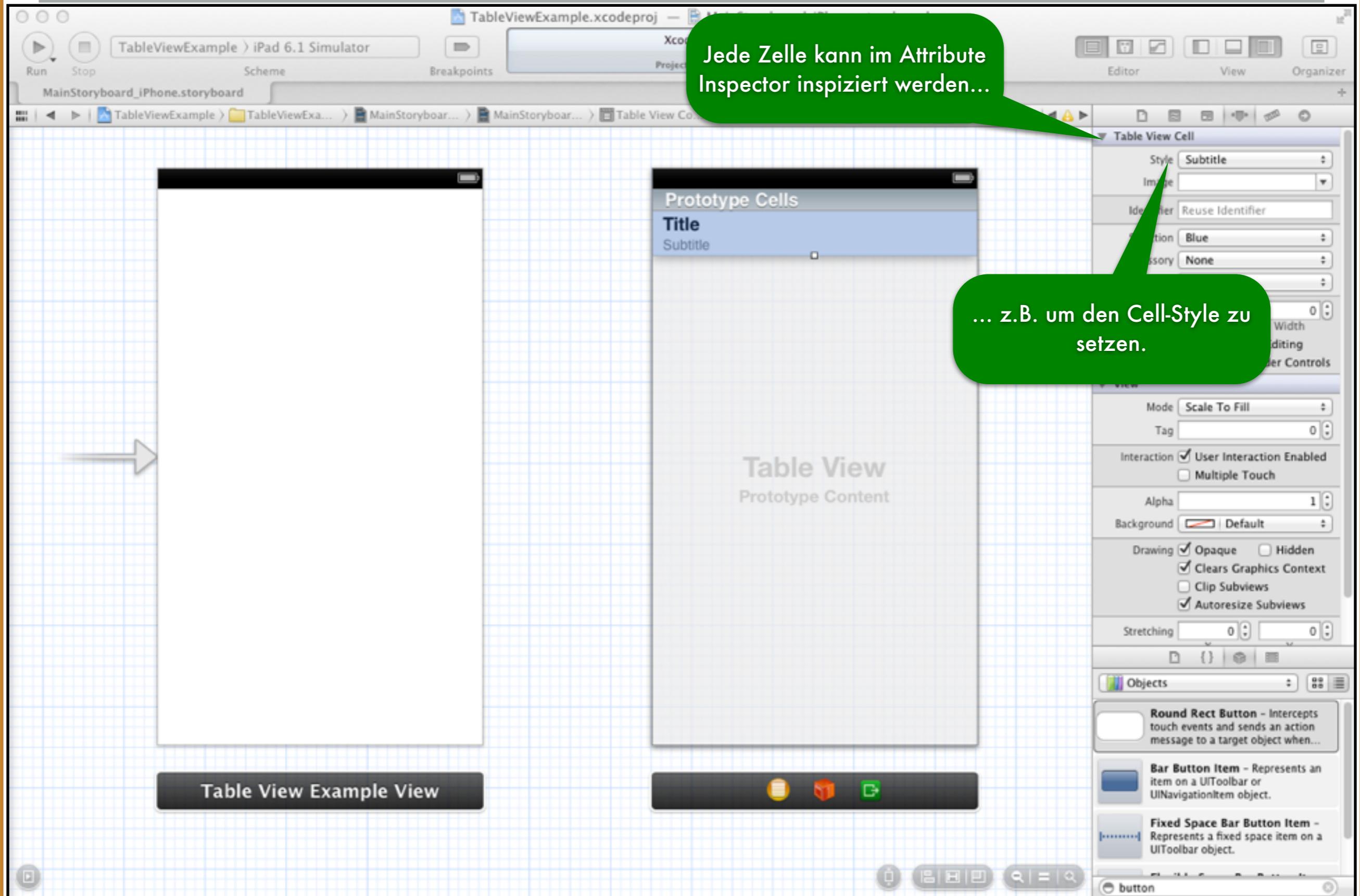


Table Views

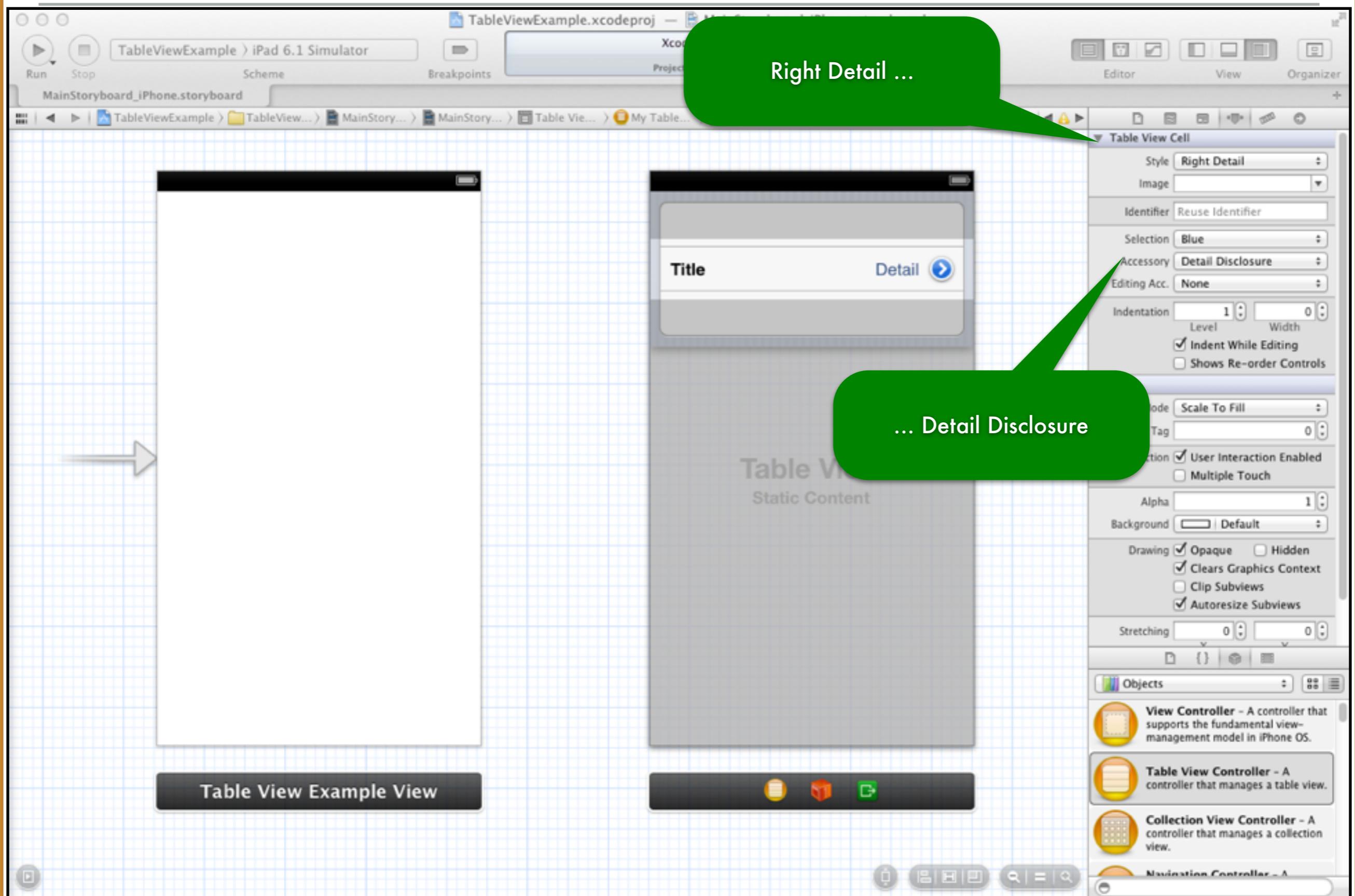


Table Views

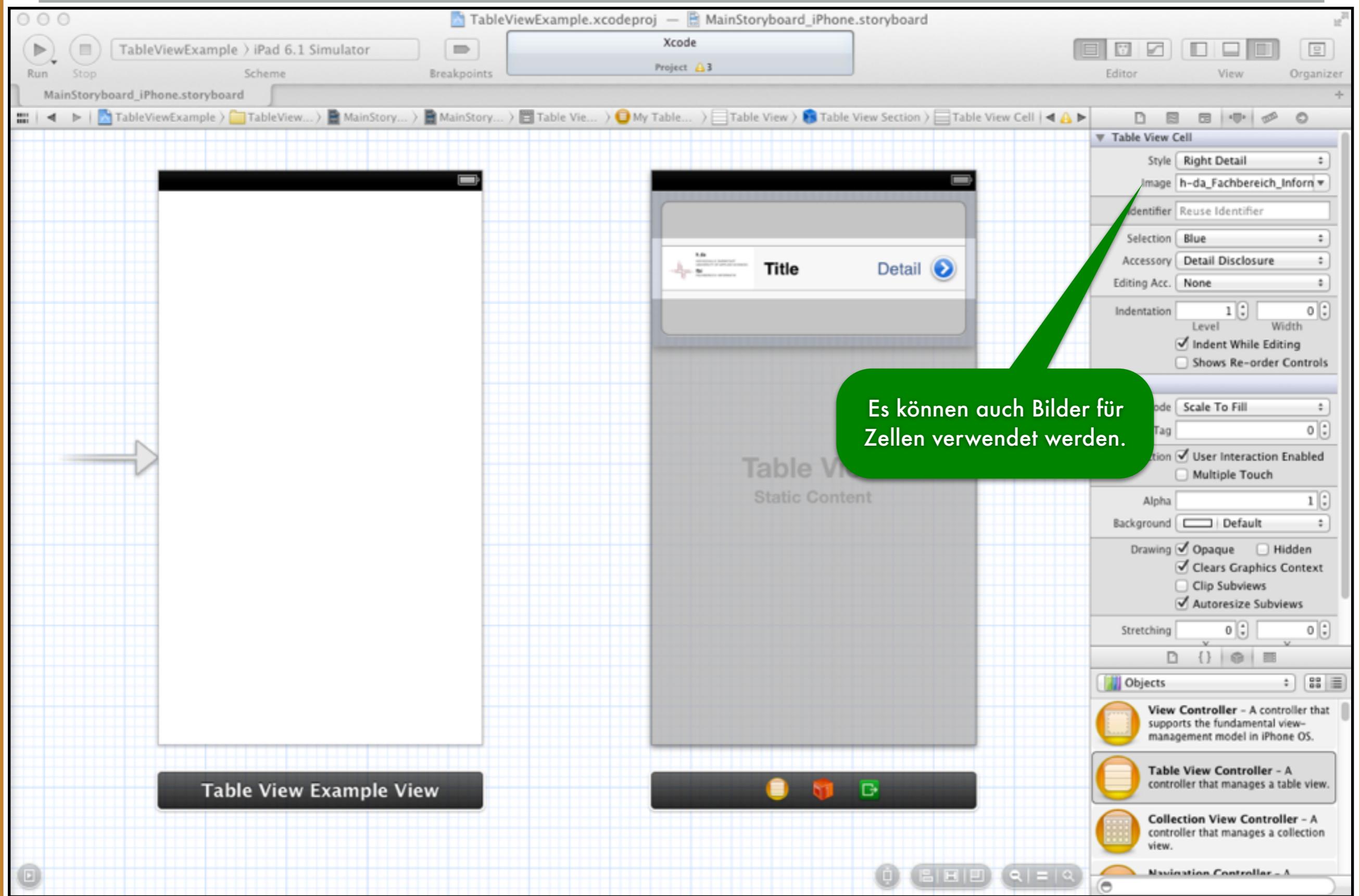


Table Views

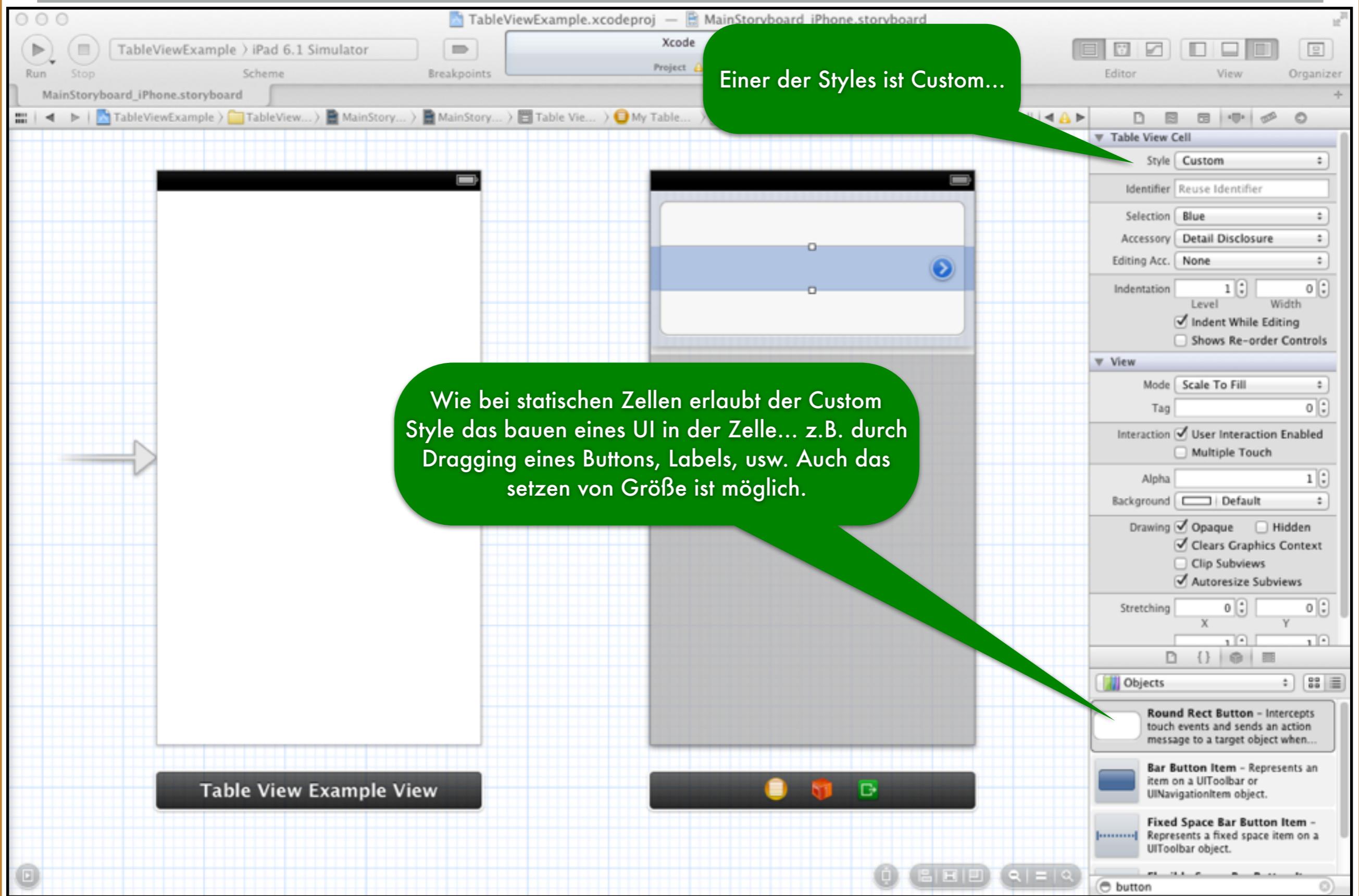


Table Views

The screenshot shows the Xcode interface with a storyboard project named 'TableViewExample.xcodeproj'. The storyboard file is 'MainStoryboard_iPhone.storyboard'. A green callout bubble points from the text below to the storyboard preview. The storyboard contains a 'Table View' with 'Static Content'. The table view has three sections: 'My Table...', 'Table View', and 'Table View Cell'. The 'Table View Cell' section is selected, and its properties are shown in the right-hand sidebar. The properties pane shows the following settings:

- Table View Cell**:
 - Style: Custom
 - Identifier: Reuse Identifier
 - Selection: Blue
 - Accessory: Detail Disclosure
 - Editing Acc.: None
 - Indentation: Level 1, Width 0
 - Indent While Editing
 - Shows Re-order Controls
- View**:
 - Mode: Scale To Fill
 - Tag: 0
 - Interaction:
 - User Interaction Enabled
 - Multiple Touch
 - Alpha: 1
 - Background: Default
 - Drawing:
 - Opaque
 - Hidden
 - Clears Graphics Context
 - Clip Subviews
 - Autoresizes Subviews
 - Stretching: X 0, Y 0
- Objects**: A list of objects including 'Round Rect Button', 'Bar Button Item', and 'Fixed Space Bar Button Item'.

Text in callout bubble:

Um custom Layouts zu verwenden ist es wichtig die passenden Autolayout Constraints zu setzen um z.B. die Höhe der Zelle an den Inhalt anzupassen.

Table View Example View

Table Views

The screenshot shows the Xcode interface with a storyboard open. A green callout bubble points from the text below to the selected table view cell in the storyboard preview.

Um Outlets zu verwenden, muss eine Subclass von UIView erstellt werden. Diese Klasse in den Zellen ist: UITableViewCell

The storyboard preview shows a table view with a single section labeled "Table View" and "Static Content". The cell's background color is blue. The Xcode right-hand sidebar displays the "Table View Cell" settings, including style (Custom), selection (Blue), and accessory (Detail Disclosure). The "View" section shows the mode as "Scale To Fill" and interaction as "User Interaction Enabled". The "Objects" section lists "Round Rect Button", "Bar Button Item", and "Fixed Space Bar Button Item".

Table View Protocol

Wie werden die Verbindungen in Code erstellt?

Verbindungen zum Code werden mittels UITableView's `dataSource` und `delegate` gemacht.

Das `delegate` wird verwendet um zu kontrollieren wie der Table angezeigt wird (Look & Feel).

Die `dataSource` stellt die Daten zur Verfügung, die in der Zelle angezeigt werden.

UITableViewController setzt sich automatisch selbst als delegate und dataSource des UITableViews.

Die eigene UITableViewController Subclass hat auch ein Property, welches auf den UITableView zeigt...

```
var tableView: UITableView // self.view in UITableViewController
```

Table View Protocol

Wann muss dataSource implementiert werden?

Wenn die Daten im Table dynamisch sind (z.B. nicht statische Zellen)

Es existieren drei wichtige Funktionen im Protokoll...

Wie viele Sections hat der Table?

Wie viele Zeilen (rows) sind in jeder Section?

Erhalten eines Views zum zeichnen jeder Zelle in einer Zeile in einer Section.

Zuerst die letzte (da die ersten beiden relativ klar sind) ...

Table View Protocol

Zur Verfügung stellen eines UIViews zum zeichnen jeder Zelle...

Es muss eine UITableViewCell sein (welches eine Subclass von UIView ist) oder eine Subclass davon.

Selbst wenn wir 10.000 Zeilen haben, so haben nur die sichtbaren eine UITableViewCell.

Dies bedeutet aber, dass UITableViewCells wiederverwendet werden, wenn Zeilen auftauchen oder verschwinden.

Dies hat Auswirkungen für Multithreaded Situationen, also ist hier Vorsicht geboten.

Table View Protocol

Der UITableView fragt seine UITableViewDataSource nach einer UITableViewCell für eine Zeile...

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

NSIndexPath ist ein Container,
welcher section und row überibt.

Table View Protocol

Der UITableView fragt seine UITableViewDataSource nach einer UITableViewCell für eine Zeile...

```
func tableView(tv: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {  
    let data = myInternalDataStructure[indexPath.section][indexPath.row]  
  
}
```

Table View Protocol

Der UITableView fragt seine UITableViewDataSource nach einer UITableViewCell für eine Zeile...

```
func tableView(tv: UITableView, cellForRowAt indexPath: NSIndexPath) -> UITableViewCell {  
    let data = myInternalDataStructure[indexPath.section][indexPath.row]  
  
    let cell = ... // create UITableViewCell and load it up with data  
  
    return cell  
}
```

Table View Protocol

Der UITableView fragt seine UITableViewDataSource nach einer UITableViewCell für eine Zeile...

```
func tableView(tv: UITableView, cellForRowAt indexPath: NSIndexPath) -> UITableViewCell {  
    let data = myInternalDataStructure[indexPath.section][indexPath.row]  
    let dequeued: AnyObject = tv.dequeueReusableCellWithIdentifier("MyCell", forIndexPath: indexPath)  
}
```

Die Funktion gibt eine UITableViewCell zurück, entweder eine die off-Screen gegangen ist oder durch erstellen einer Kopie eines Prototypes aus dem Storyboard.

Gibt an welcher Prototype zu kopieren oder zu nutzen ist.
(nicht vergessen den Identifier im Storyboard zu setzen)

Table View Protocol

Der UITableView fragt seine UITableViewDataSource nach einer UITableViewCell für eine Zeile...

```
func tableView(tv: UITableView, cellForRowAt indexPath: NSIndexPath) -> UITableViewCell {  
    let data = myInternalDataStructure[indexPath.section][indexPath.row]  
    let dequeued: AnyObject = tv.dequeueReusableCellWithIdentifier("MyCell", forIndexPath: indexPath)  
    let cell = dequeued as! ...  
  
    return cell  
}
```

Die dequeued UITableViewCell kommt als AnyObject zurück. Es muss also in die passende Subclass von UITableViewCell gecastet werden. Danach kann sie befüllt und zurückgegeben werden.

Table View Protocol

Der UITableView fragt seine UITableViewDataSource nach einer UITableViewCell für eine Zeile...

```
func tableView(tv: UITableView, cellForRowAt indexPath: NSIndexPath) -> UITableViewCell {  
    let data = myInternalDataStructure[indexPath.section][indexPath.row]  
    let dequeued: AnyObject = tv.dequeueReusableCellWithIdentifier("MyCell", forIndexPath: indexPath)  
    let cell = dequeued as! UITableViewCell  
    cell.textLabel?.text = "Title"  
    cell.detailedTextLabel?.text = "Subtitle"  
  
    return cell  
}
```

Für eine Subtitle Cell...

Table View Protocol

Der UITableView fragt seine UITableViewDataSource nach einer UITableViewCell für eine Zeile...

```
func tableView(tv: UITableView, cellForRowAt indexPath: NSIndexPath) -> UITableViewCell {  
    let data = myInternalDataStructure[indexPath.section][indexPath.row]  
    let dequeued: AnyObject = tv.dequeueReusableCellWithIdentifier("MyCell", forIndexPath: indexPath)  
    let cell = dequeued as! UITableViewCell  
    cell.textLabel?.text = "Title"  
    cell.detailedTextLabel?.text = "Subtitle"  
  
    return cell  
}
```

Für eine Subtitle Cell...

Table View Protocol

Woher weiss ein dynamischer Table wie viele Zeilen existieren?

Und natürlich auch wie viele sections?

Mittels dieser UITableViewDataSource Protokoll Funktionen...

```
func numberOfSectionsInTableView(sender: UITableView) -> Int
```

```
func tableView(sender: UITableView, numberOfRowsInSection: Int) -> Int
```

Anzahl der sections ist 1 per Default

Anders ausgedrückt, wenn numbersOfSectionsInTableView nicht implementiert wird, ist es 1.

Keinen Default für numberOfRowsInSection

Diese Funktion ist required im Protokoll (genau wie cellForRowIndexPath)

Und statische Tables?

Keine dieser dataSource Funktionen für statische Tables implementieren.

UITableViewController macht das für uns.

Wir können die Daten direkt im Storyboard editieren.

Table View Protocol

Zusammengefasst...

Laden von Daten in den Table View ist einfach...

1. Setzen des Table View's dataSource im Controller (automatisch für UITableViewController)
2. Implementieren von `numberOfSectionsInTableView` und `numberOfRowsInSection`
3. Implementieren von `cellForRowAtIndexPath` um eine gefüllte UITableViewCell zurück zu geben.

Section Titles sind ebenfalls Teil der Table "Daten"

Daher über die UITableViewDataSource Funktion zurück geben...

```
func tableView(tableView, titleForHeaderInSection section: Int) -> String
```

Wenn der String nicht ausreichend ist, dann kann das UITableView delegate einen UIView zur Verfügung stellen.

Es existieren eine Vielzahl weiterer Funktionen in diesem Protokoll

Besprechen wir nicht in der Veranstaltung.

Dienen meist zum Editieren des Tables (delete/insert/move)

Dies verändert meist das Model (wir sprechen vom
UITableViewDataSource Protokoll...)