FREE eBook

LEARNING uitableview

Free unaffiliated eBook created from
Stack Overflow contributors.

#uitableview

Table of Contents

About

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: uitableview

It is an unofficial and free uitableview ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official uitableview.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapter 1: Getting started with uitableview

Remarks

This section provides an overview of what uitableview is, and why a developer might want to use it.

It should also mention any large subjects within uitableview, and link out to the related topics. Since the Documentation for uitableview is new, you may need to create initial versions of those related topics.

Examples

UITableView in detail

What is UITableView?

`UITableView` is a most frequently used user interface object that presents data in a **scrollable list** of multiple rows in a single column that can also be divided into sections.It allows vertical scrolling only and is a subclass of `UIScrollView`.

Why do we use UITableView?

We can use `UITableView` to present a **list of options** that can be selected, to navigate through **hierarchically structured data**, present an **indexed list of items**, to display detail information and controls in visually distinct groupings making use of **sections**.

We can see use of `UITableView` in our contacts, mailing lists etc. It is not only just used to present textual data but also images along with texts can be listed such as in YouTube app.

Detailed instructions on getting UITableView set up or installed using Story Board.

1. Create a simple project for Single View application.
2. In the Object Library, select the “Table View” object and drag it into the view of your view controller. Simply running the project you will see a blank page with lines.
3. Now if you simply want a scrollable view with contents, then drag a `UIView` into the `UITableView`, adjust its size and drag rest of the UIElements into that view as per requirements. But if you want to present a list of similar format, we use `UITableViewCell`.
4. The `UITableViewCell` class defines the attributes and behavior of the cells that appear in UITableView objects. This class includes properties and methods for setting and managing cell content and background (including text, images, and custom views), managing the cell selection and highlight state, managing accessory views, and initiating the editing of the cell contents.
5. The best part of using `UITableViewCell` is reusability. The purpose of dequeueReusableCellWithIdentifier is to use less memory.For e.g. if you have a list of 1000 entries and only 10 entries are visible at a time then only the visible cells are allocated in

memory, rest are reused as when the user scrolls the list. All you need to do is drag a `UITableViewCell` and drop to tableView. Then click on cell-> Go to attribute inspector-> Set tableView style to custom and identifier to anything unique you would like say myCell.

6. Now we need to conform to data source so that object gives data to another object. For example, the `UITableViewDataSource` protocol has methods such as cellForRowAtIndexPath and `numberOfRowsInSection` dictating what should be displayed in the table. Whereas delegate type object responds to actions that another object takes. For example, the `UITableViewDelegate` protocol has methods such as `didSelectRowAtIndexPath` for performing actions upon a user selecting a particular row in a table. 7.When you conform to datasource you need to implement its required method i.e

```
 - (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)section {
        // Here you need to give the total number of items you want to list. For example if
you want list of 2 numbers, then:

           return 2;

 }
```

and

```
- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath
*)indexPath {

//Here you need to dequeue the reusable cell which we discussed in point 5. Then you can
modify your cell here according to you and customize it here as per your requirement. Here the
call comes for numberOfRows number of times.

static NSString *cellIdentifier = @"cellID";

UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:
cellIdentifier];
if (cell == nil) {
    cell = [[UITableViewCell alloc]initWithStyle:
    UITableViewCellStyleDefault reuseIdentifier:cellIdentifier];
}
if(indexPath.row){
    [cell.textLabel setText:"1"];
}else{
    [cell.textLabel setText:"2"];
}

return cell;
}
```

7. Also about NSIndexPath:- The NSIndexPath class represents the path to a specific node in a tree of nested array collections. This path is known as an index path. Its objects are always of length 2. They're used for example to index a table view cell. The first index in a NSIndexPath object is called the section, the second is the row. An index path object with section 0 and row 0 indicates the first row in the first section. Use `[NSIndexPath indexPathForRow:inSection:]` to quickly create an index path.

Delegate & Data Source Methods:

Each table view must have a delegate and a data source.

Delegate Methods

None of the delegate methods are actually required, however you'll need to implement tableView:didSelectRowAtIndexPath: to handle touches on a table cell:

And other methods are...

```
// Display customization

- (void)tableView:(UITableView *)tableView willDisplayCell:(UITableViewCell *)cell
forRowAtIndexPath:(NSIndexPath *)indexPath;

// Variable height support

// If these methods are implemented, the above -tableView:heightForXXX calls will be deferred
until views are ready to be displayed, so more expensive logic can be placed there.

- (CGFloat)tableView:(UITableView *)tableView heightForRowAtIndexPath:(NSIndexPath
*)indexPath;
- (CGFloat)tableView:(UITableView *)tableView heightForHeaderInSection:(NSInteger)section;
- (CGFloat)tableView:(UITableView *)tableView heightForFooterInSection:(NSInteger)section;

// Section header & footer information. Views are preferred over title should you decide to
provide both

- (UIView *)tableView:(UITableView *)tableView viewForHeaderInSection:(NSInteger)section;   //
custom view for header. will be adjusted to default or specified header height
- (UIView *)tableView:(UITableView *)tableView viewForFooterInSection:(NSInteger)section;   //
custom view for footer. will be adjusted to default or specified footer height

// Accessories (disclosures).

- (void)tableView:(UITableView *)tableView
accessoryButtonTappedForRowWithIndexPath:(NSIndexPath *)indexPath;

// Selection

// Called before the user changes the selection. Return a new indexPath, or nil, to change the
proposed selection.
- (NSIndexPath *)tableView:(UITableView *)tableView willSelectRowAtIndexPath:(NSIndexPath
*)indexPath;
// Called after the user changes the selection.
- (void)tableView:(UITableView *)tableView didSelectRowAtIndexPath:(NSIndexPath *)indexPath;


// Editing

// Allows customization of the editingStyle for a particular cell located at 'indexPath'. If
not implemented, all editable cells will have UITableViewCellEditingStyleDelete set for them
when the table has editing property set to YES.
- (UITableViewCellEditingStyle)tableView:(UITableView *)tableView
editingStyleForRowAtIndexPath:(NSIndexPath *)indexPath;

// Controls whether the background is indented while editing.  If not implemented, the default
is YES.  This is unrelated to the indentation level below.  This method only applies to
grouped style table views.
```

```
- (BOOL)tableView:(UITableView *)tableView shouldIndentWhileEditingRowAtIndexPath:(NSIndexPath
*)indexPath;



// Indentation

- (NSInteger)tableView:(UITableView *)tableView indentationLevelForRowAtIndexPath:(NSIndexPath
*)indexPath; // return 'depth' of row for hierarchies
```

Data Source Required Methods:

The following methods are required from the data source: tableView:numberOfRowsInSection: and tableView:cellForRowAtIndexPath:. If your table is a grouped table, you must also implement numberOfSectionsInTableView:.

Required Methods:-

```
- (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)section;

// Row display.

- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath
*)indexPath;
```

Optional methods:-

```
- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView;              // Default is
1 if not implemented

- (NSString *)tableView:(UITableView *)tableView titleForHeaderInSection:(NSInteger)section;
// fixed font style. use custom view (UILabel) if you want something different
- (NSString *)tableView:(UITableView *)tableView titleForFooterInSection:(NSInteger)section;

// Editing

// Individual rows can opt out of having the -editing property set for them. If not
implemented, all rows are assumed to be editable.
- (BOOL)tableView:(UITableView *)tableView canEditRowAtIndexPath:(NSIndexPath *)indexPath;

// Moving/reordering

// Allows the reorder accessory view to optionally be shown for a particular row. By default,
the reorder control will be shown only if the datasource implements -
tableView:moveRowAtIndexPath:toIndexPath:
- (BOOL)tableView:(UITableView *)tableView canMoveRowAtIndexPath:(NSIndexPath *)indexPath;

// Index

- (NSArray<NSString *> *)sectionIndexTitlesForTableView:(UITableView *)tableView;
// return list of section titles to display in section index view (e.g. "ABCD...Z#")
- (NSInteger)tableView:(UITableView *)tableView sectionForSectionIndexTitle:(NSString *)title
atIndex:(NSInteger)index;  // tell table which section corresponds to section title/index
(e.g. "B",1))

// Data manipulation - insert and delete support
```

```
// After a row has the minus or plus button invoked (based on the UITableViewCellEditingStyle
for the cell), the dataSource must commit the change
// Not called for edit actions using UITableViewRowAction - the action's handler will be
invoked instead
- (void)tableView:(UITableView *)tableView
commitEditingStyle:(UITableViewCellEditingStyle)editingStyle forRowAtIndexPath:(NSIndexPath
*)indexPath;

// Data manipulation - reorder / moving support

- (void)tableView:(UITableView *)tableView moveRowAtIndexPath:(NSIndexPath *)sourceIndexPath
toIndexPath:(NSIndexPath *)destinationIndexPath;
```

Anatomy of a Table Cell

The default UITableViewCell has several standard data views and subviews.

- **cell.textLabel** - the UILabel for the cell
- **cell.detailTextLabel** - a smaller UILabel that appears below the text label
- **cell.imageView** - a UIImageView that the left side of the cell

The optional accessory view may contain one of the following icons. The default accessoryType is **UITableViewCellAccessoryNone**.

Read Getting started with uitableview online: https://riptutorial.com/uitableview/topic/6178/getting-started-with-uitableview

Credits

S. No	Chapters	Contributors
1	Getting started with uitableview	Community, Ishika, Rahul