IOS115

Customizing TableViews

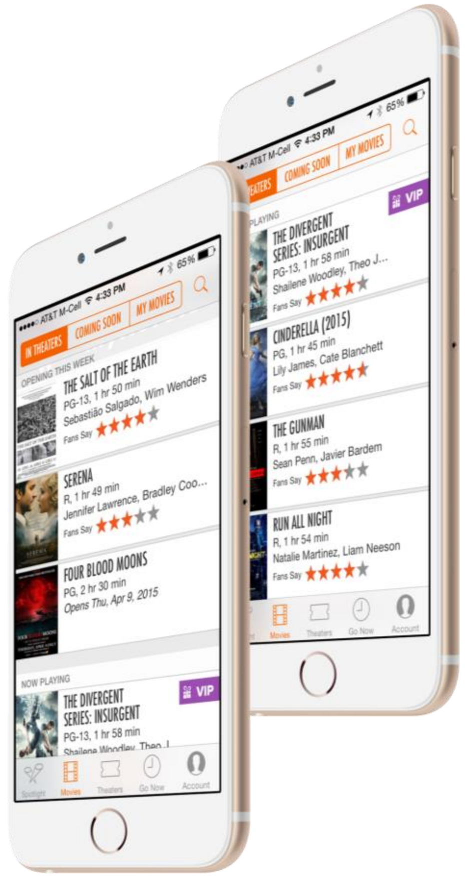
Download class materials from
university.xamarin.com

Microsoft

Xamarin University

Objectives

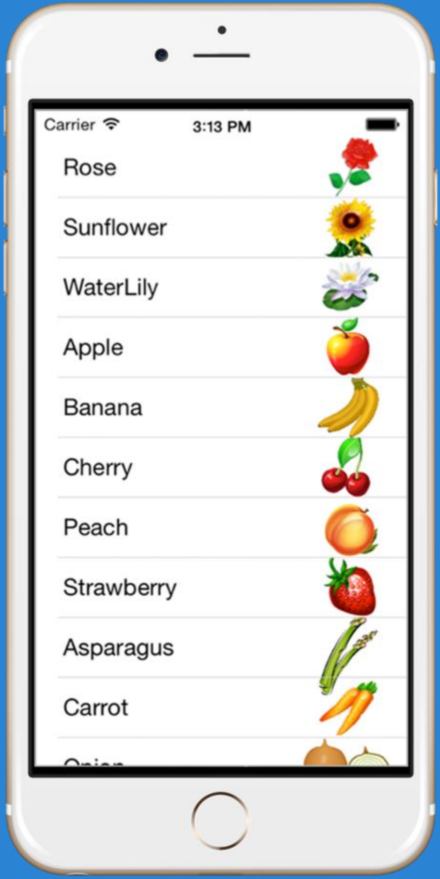
1. Customize table view cells in code
2. Customize table view cells in the designer
3. Group data in the table view



Customize table view cells in code

Tasks

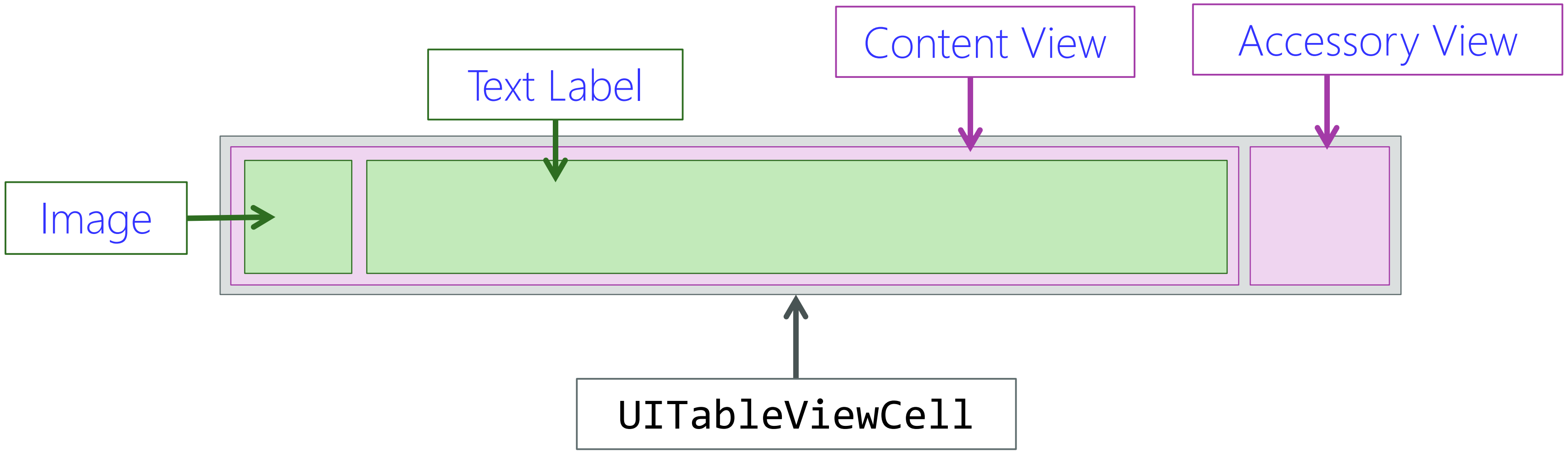
1. View the anatomy of a cell
2. Customize the default cell styles
3. Create a custom cell



Xamarin University

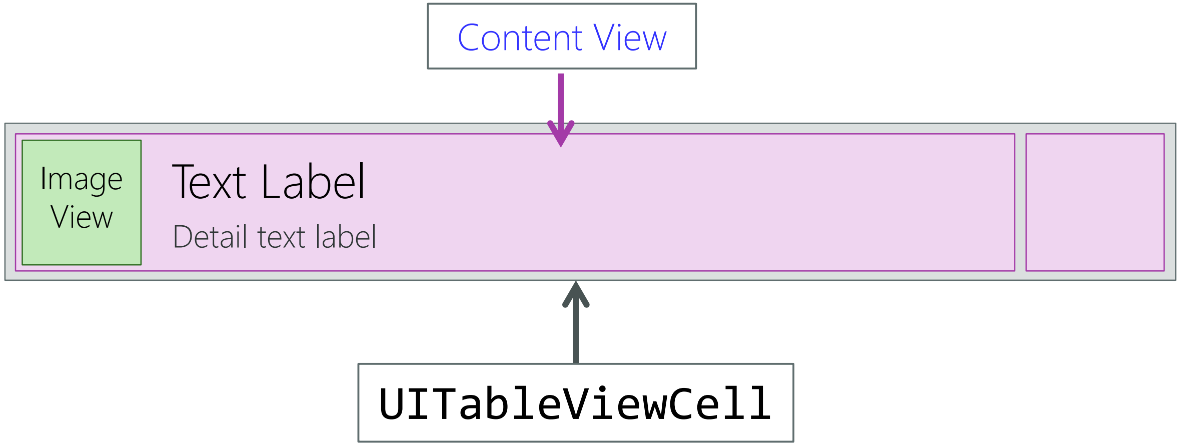
Anatomy of a default cell

❖ The default **`UITableViewCell`** is composed of the cell and several subviews, which allows for a high degree of customization out of the box



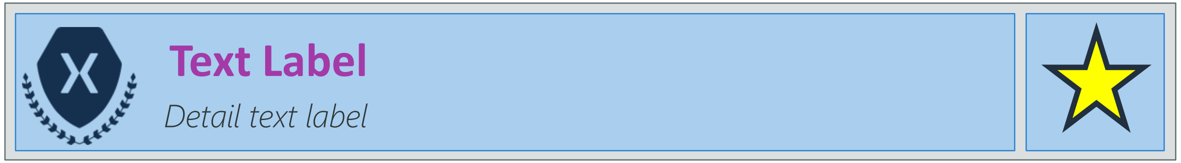
Subviews

- We can customize a cell by working with the default content view, taking advantage of the built-in classes to adjust fonts, colors, and change the accessory image



Customize the default views

- We can change the properties on the built-in subviews in the **`GetCell`** implementation

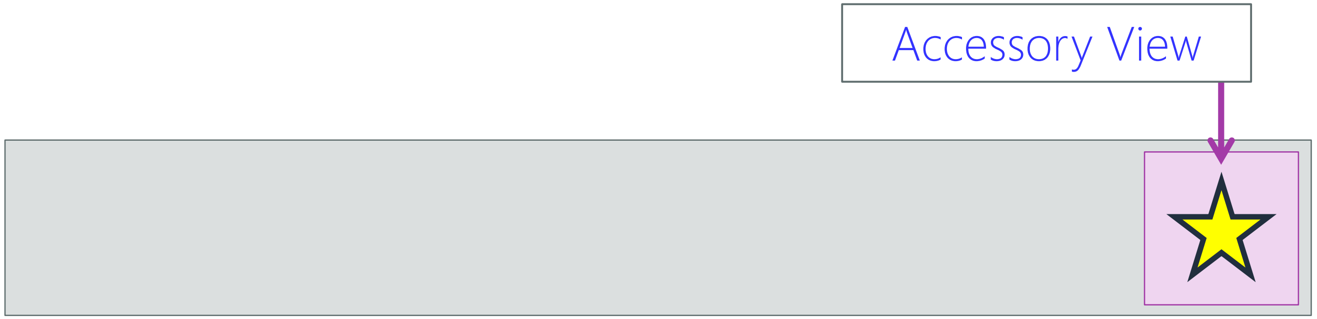


All default cells all contain the **`DetailTextLabel`**, and **`ImageView`** properties but they will be **`null`** for styles that don't support these visualizations

Xamarin University

Accessory view

❖ The accessory view is used to indicate state or behavior when a cell is tapped – you can customize the image or replace it with a custom **`UIView`**



The accessory view shows additional information like state (checkmark) or it indicates behavior (chevron for navigation). *It shouldn't be used for cell content.*

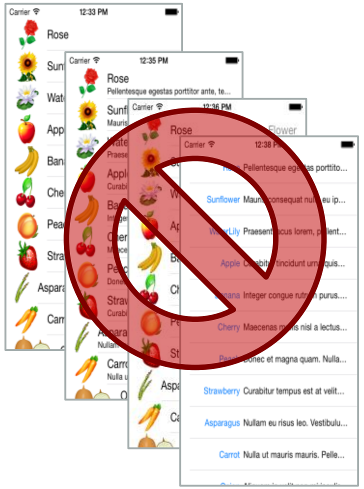
Individual Exercise

Customize a default table cell

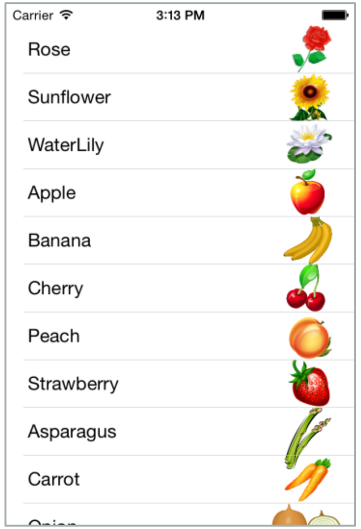
Xamarin University

Custom Table View cells

- Built-in cell styles cover common scenarios, but sometimes you need to display information in ways that are not supported by default - when this happens you can turn to a **custom table view cell**



What if we want the image on the right side?



Xamarin University

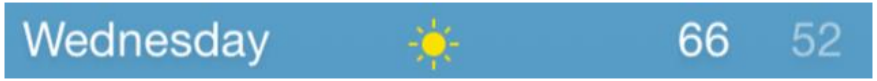
Creating a custom table view cell

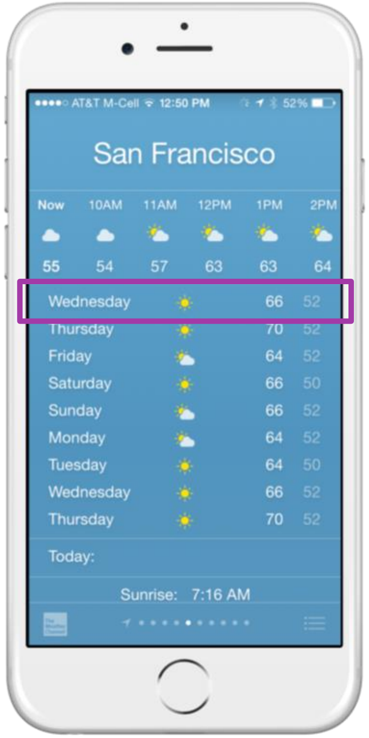
- Custom cells can be created either in code, or in the Storyboard designer

Completely customized cells

- Sometimes the data you want to display doesn't fit within the confines of the default cell

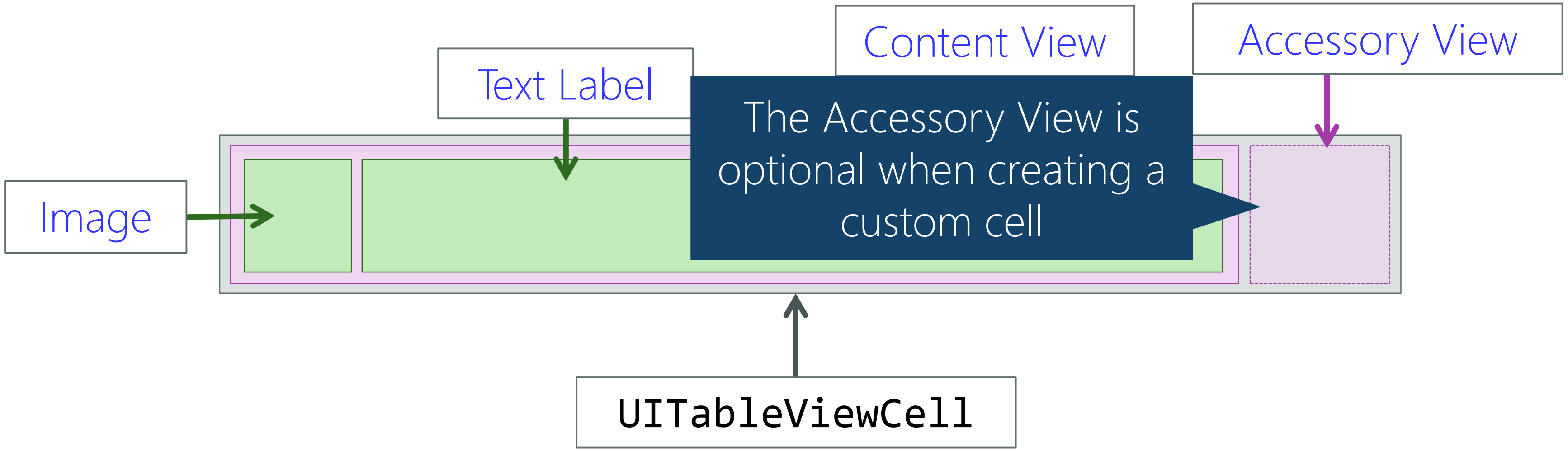
- When this happens, it is necessary to make a *custom cell*





Anatomy of a custom cell

❖ The content view is blank in a custom cell, it's up to you to populate it with custom controls and visuals



Steps to creating a custom cell

- There are three steps to creating a custom cell

Create a custom cell class

Add the custom UI views to the cell

Populate the custom views with data

Xamarin University

Create a custom cell class

❖ A custom cell class derives from **`UITableViewCell`** and defines the UI and behavior of the cell

```
public class PlantTableViewCell : UITableViewCell
{
    ...
}
```

Create a custom cell class

- If the cell is used within a Table View created in the designer, the constructor must but updated

```
public class PlantTableViewCell : UITableViewCell
{
    public PlantTableViewCell(IntPtr handle) : base (handle)
    {
    }
    ...
}
```

Constructor is passed a native handle and must forward the call to the base class

Add the custom UI visuals to the cell

- Create custom subview(s) to display data within the cell and add them to the **`ContentView`**

```
UILabel plantName;  // hold reference to update

public PlantTableViewCell (IntPtr handle) : base (handle)
{
    plantName = new UILabel();
    ContentView.AddSubview (plantName);
}
```

Xamarin University

Layout the cell

❖ Override the **`LayoutSubviews`** method to size and position the child views in your cell

```
UILabel plantName;  // hold reference to update

public override void LayoutSubviews()
{
    base.LayoutSubviews ();
    plantName.Frame = new CGRect(10, 18, 100, 20);
    ...
}
```

Xamarin University

Register the cell with the UITableView

- Register the cell for *reuse* using the **`RegisterClassForCellReuse`** method on your **`UITableView`**

```
public class PlantTVC : UITableViewController
{
    public PlantTVC ()
    {
        TableView.RegisterClassForCellReuse(
           typeof(PlantCellView), "PlantCellId");
        ...
    }
}
```

Recall that reusing cells optimizes the memory and performance of your application – you should always utilize this iOS feature

Visualize the data in the cell

- We expose a public method to update the content of the child views

```
public class PlantTableViewCell : UITableViewCell
{
    ...
    public void UpdateCell (Plant plant)
    {
        plantName.Text = plant.Name;
        plantImage.Image = LoadImageFromUrl(plant.ImageUrl);
    }
}
```

Visualize the data in the cell

❖ Call the update method on the custom cell from the **`GetCell`** method in the table view controller

```
public override UITableViewCell GetCell(UITableView tableView, NSIndexPath indexPath)
{
   Plant plant = plants[indexPath.Row];
   var cell = tableView.DequeueReusableCell("PlantCellId") as PlantTableViewCell;

   cell.UpdateCell(plant);

   return cell;
}
```

Individual Exercise

Create a custom table view cell in code

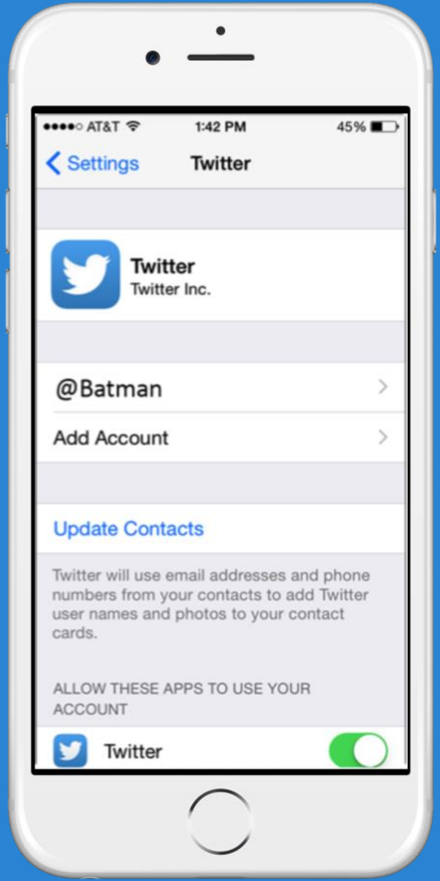
Xamarin University

Customize table view cells in the designer

Xamarin University

Tasks

1. Choose static or dynamic cells based on your app's data
2. Design a custom cell using the designer


●●●●○ AT&T 1:42 PM 45%

Settings Twitter

Twitter
Twitter Inc.

@Batman

Add Account

Update Contacts

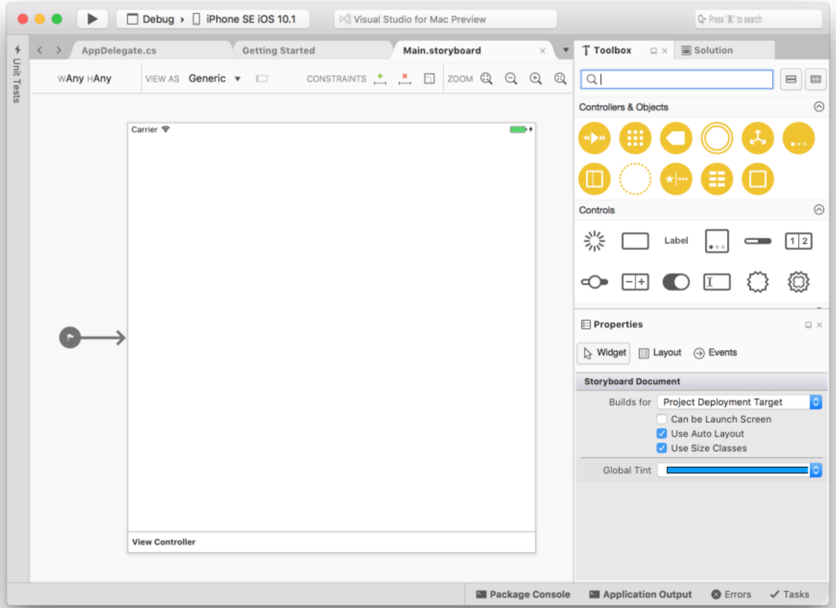
Twitter will use email addresses and phone numbers from your contacts to add Twitter user names and photos to your contact cards.

ALLOW THESE APPS TO USE YOUR ACCOUNT

Twitter

The iOS Designer

- The Xamarin.iOS designer allows you design, create and visualize your UI including Table Views and Table Cells



Storyboard Document

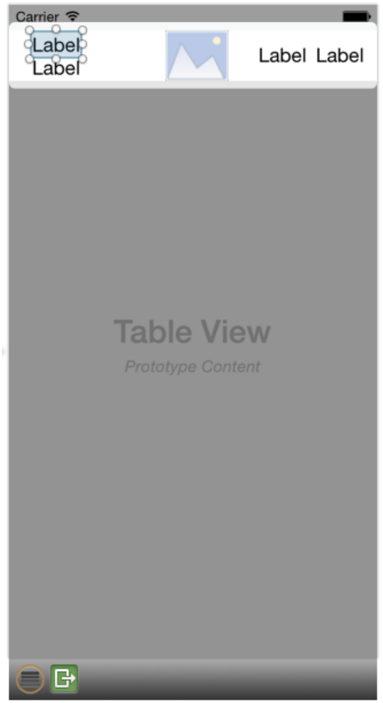
Builds for: Project Deployment Target

- [ ] Can be Launch Screen
- [x] Use Auto Layout
- [x] Use Size Classes

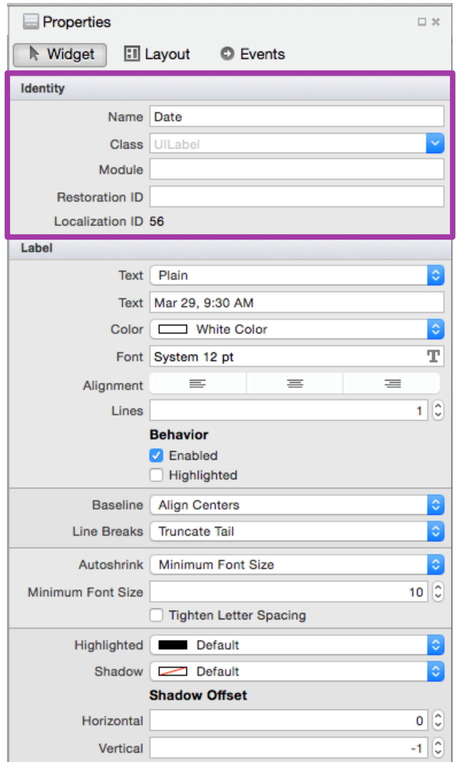
Global Tint

Style the custom cell in the designer

- When adding a **`UITableView`** to a storyboard, it will create an editable table view cell that can be customized



Changing properties



Properties

Widget | Layout | Events

Identity

Name	Date
Class	UILabel
Module	
Restoration ID	
Localization ID	56

Label

Text	Plain
Text	Mar 29, 9:30 AM
Color	White Color
Font	System 12 pt
Alignment	
Lines	1

Behavior

- [✓] Enabled
- [ ] Highlighted

Baseline	Align Centers
Line Breaks	Truncate Tail
Autoshrink	Minimum Font Size
Minimum Font Size	10

- [ ] Tighten Letter Spacing

Highlighted	Default
Shadow	Default

Shadow Offset

Horizontal	0
Vertical	-1

- Use the **Identity** tab to assign a **Name** and **Class** to your UI element
  - **Name** assigns a code-behind element to the control
  - **Class** creates the code-behind class used to customize the cell definition

Xamarin University

Two types of cells

- The iOS designer supports two types of table view cell designs

Static

Static cells are populated at design time and don't change

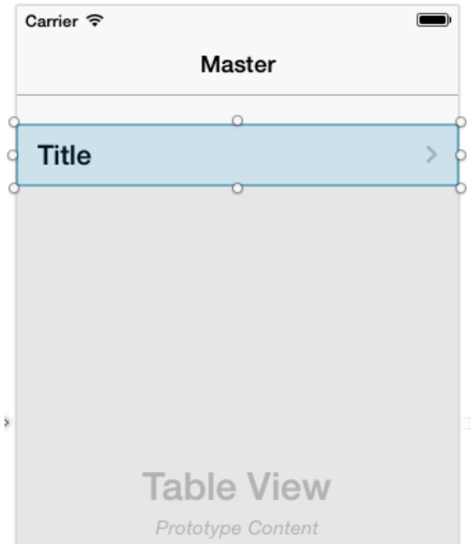
Dynamic

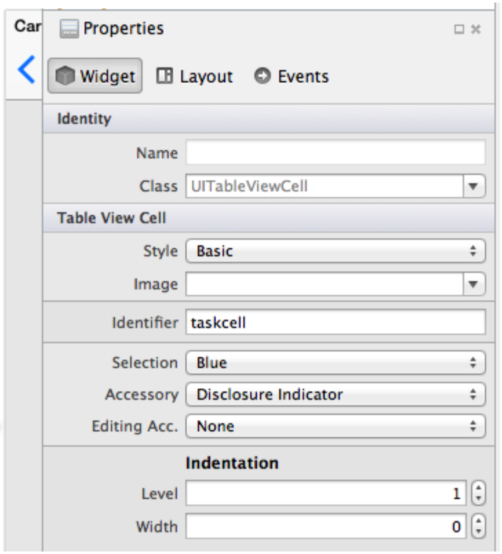
Dynamics cells are populated with runtime data

Dynamic prototype cells

- Custom cells which are populated with runtime data are represented in the designer using a *dynamic prototype* cell definition

Prototype cell





Identity

Name:
Class: UITableViewCell

Table View Cell

Style: Basic
Image:
Identifier: taskcell
Selection: Blue
Accessory: Disclosure Indicator
Editing Acc.: None

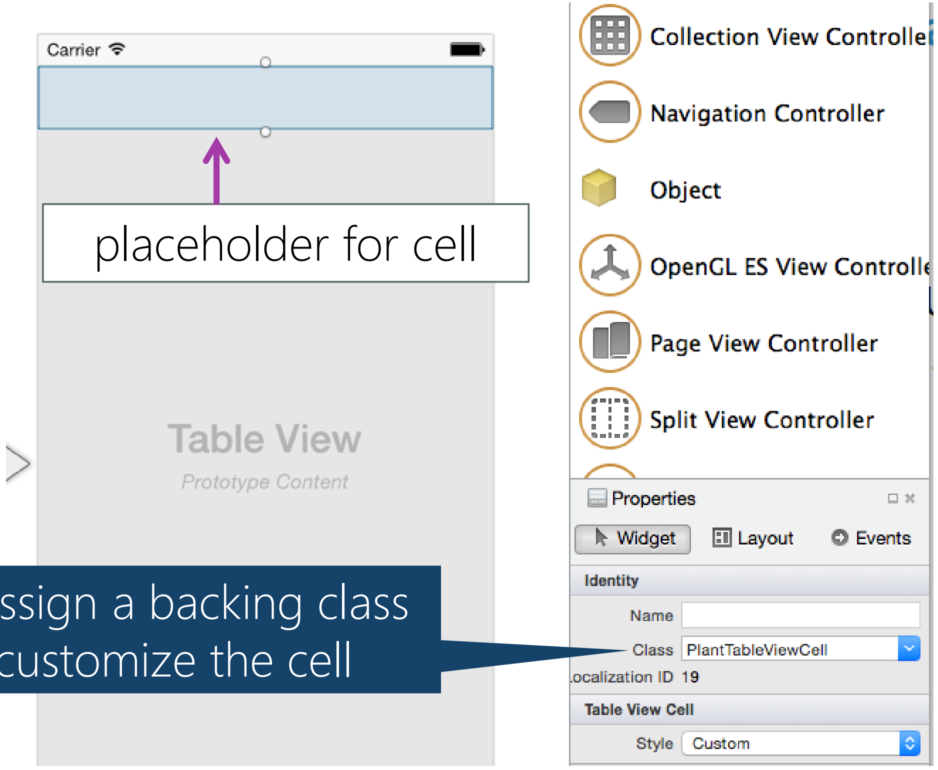
Indentation

Level: 1
Width: 0

Designing a cell in the designer

- When the Table View or Table View Controller is created in a Storyboard, then you can **click on the cell placeholder** to adjust the design

- While cell design is active, drag and drop sub-views into the cell container



Identity

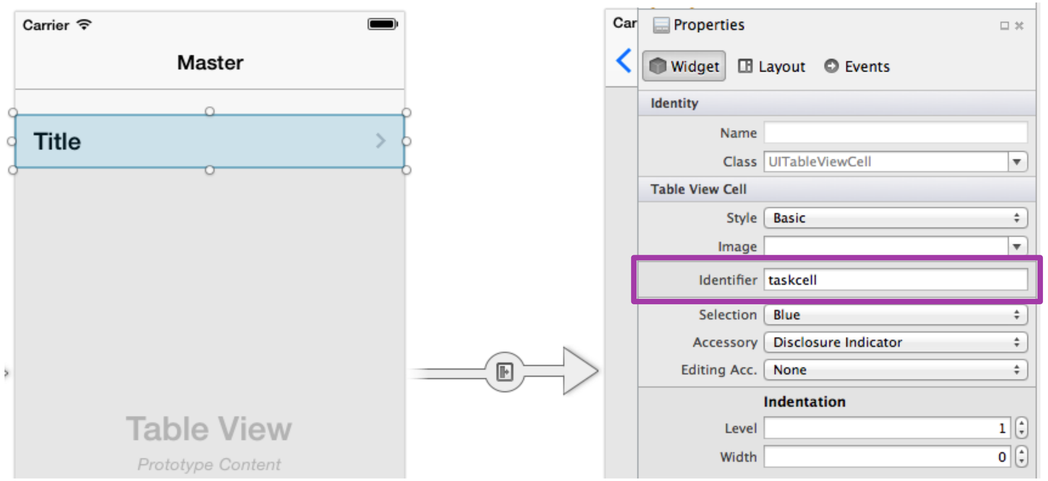
Name	
Class	PlantTableViewCell
ocalization ID	19

Table View Cell

Style	Custom

Set the reuse Identifier

- Set the reuse Identifier when using dynamic prototype cells to enable cell reuse



Identity

Name:
Class: UITableViewCell

Table View Cell

Style: Basic
Image:
Identifier: taskcell
Selection: Blue
Accessory: Disclosure Indicator
Editing Acc.: None

Indentation

Level: 1
Width: 0

Make sure the reuse identifier set in the storyboard matches the ID used in the `GetCell` method in your table view controller code-behind

Flash Quiz

Flash Quiz

① Cells which contain pre-defined data are referred to as:

a) Prototype cells

b) Dynamic cells

c) Static cells

Xamarin University

Flash Quiz

① Cells which contain pre-defined data are referred to as:

a) Prototype cells
b) Dynamic cells
c) Static cells

Flash Quiz

② When creating custom cells, the designer can do everything custom code can

a) True

b) False

Flash Quiz

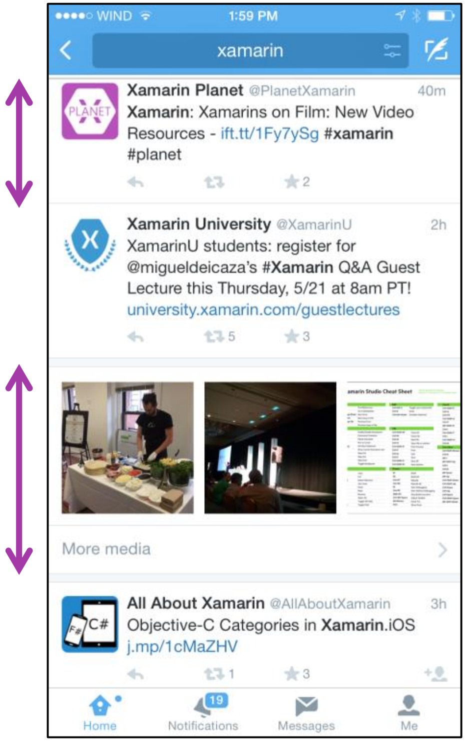
② When creating custom cells, the designer can do everything custom code can

a) True

b) <u>False</u>

Self-sizing rows

- By default, **`UITableView`** rows are automatically sized based on their content if defined with **Auto Layout**



Xamarin Planet @PlanetXamarin 40m
Xamarin: Xamarins on Film: New Video Resources - ift.tt/1Fy7ySg **#xamarin** #planet
2

Xamarin University @XamarinU 2h
XamarinU students: register for @migueldeicaza's **#Xamarin** Q&A Guest Lecture this Thursday, 5/21 at 8am PT! university.xamarin.com/guestlectures
5 3

More media

All About Xamarin @AllAboutXamarin 3h
Objective-C Categories in **Xamarin**.iOS j.mp/1cMaZHV
1 3

Turning off self-sizing rows

- Self-sizing rows can be turned off by setting the **`EstimatedRowHeight`** property to 0

```
public class MessagesTableViewController :UITableViewController
{
    public MyTableViewController()
    {
        TableView.EstimatedRowHeight = 0;
        ...
    }
}
```

The height of the row must now manually be set

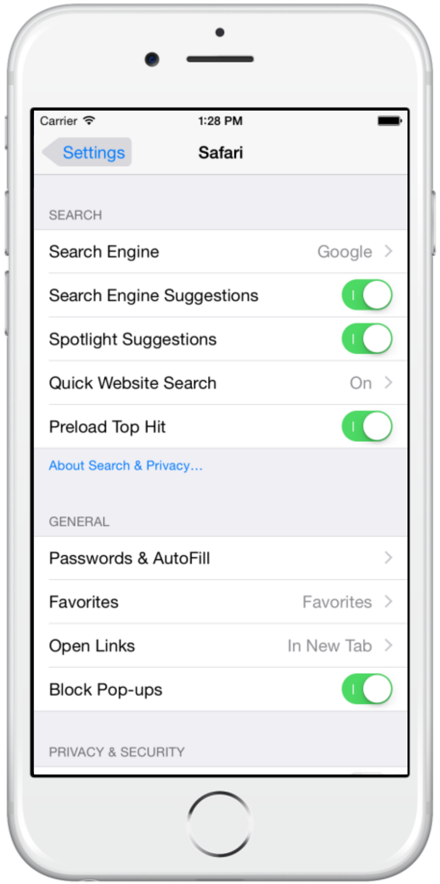
Individual Exercise

Create a prototype table view cell using the designer

Static table view cells

- When you want to display pre-defined data which does not change, you can use **static cells**, these are:
  - hard coded into the table view design
  - not assigned a reuse identifier
  - not populated by a table view source

- Typically used when the design and data of the cell is completely known at compile time



Carrier 1:28 PM

Settings Safari

SEARCH

Search Engine Google >

Search Engine Suggestions ✓

Spotlight Suggestions ✓

Quick Website Search On >

Preload Top Hit ✓

About Search & Privacy…

GENERAL

Passwords & AutoFill >

Favorites Favorites >

Open Links In New Tab >

Block Pop-ups ✓

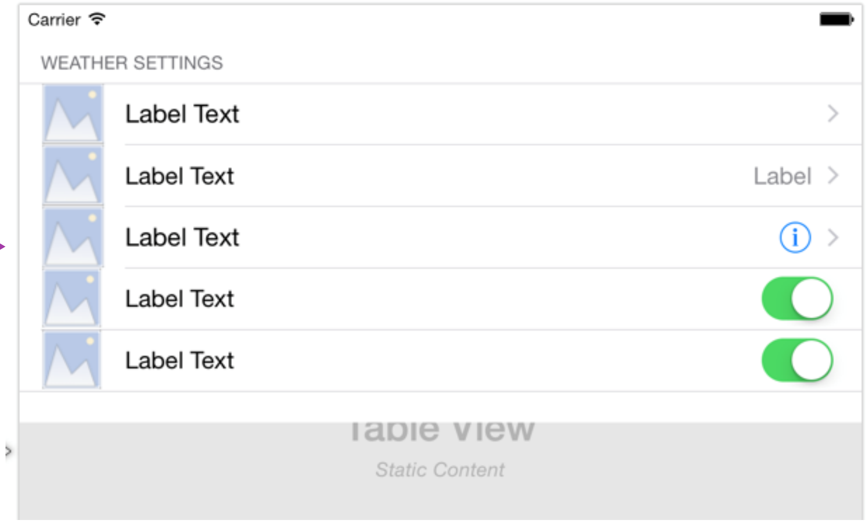
PRIVACY & SECURITY

Xamarin University

Static cells in the designer

- We can create and populate Static cells using the designer

Static cell



Xamarin University

Populating static cells

- To update the contents of the static cells at runtime, name the cells in the designer and access the child views in the view controller's code behind

```
partial class SettingsViewController : UITableViewController
{
    ...

    public override void ViewDidLoad ()
    {
        CellDefaultCity.DetailTextLabel.Text = "Vancouver";
        ...
    }
}
```

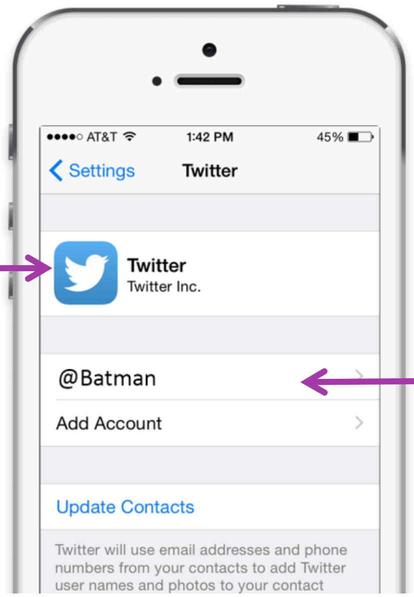
Group Exercise

Create static cells in the Designer

Mixing static and dynamic data?

- iOS does not allow you to mix static and dynamic prototype cells

Static cell content (never changes)



Twitter will use email addresses and phone numbers from your contacts to add Twitter user names and photos to your contact

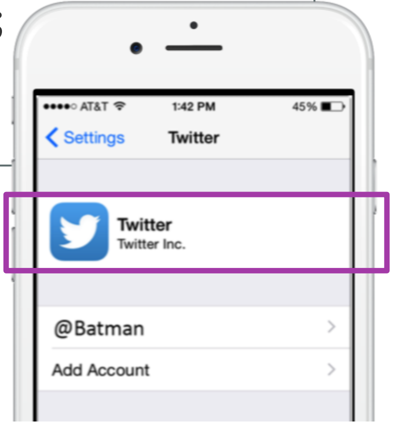
Dynamic cell content (decided at runtime)

Xamarin University

Simulating static content with dynamic cells

- Dynamic prototype cells can behave like static cells when the cell is returned without content from **`GetCell`**

```
public override  UITableViewCell GetCell (UITableView tableView,
                                          NSIndexPath indexPath)
{
    cell = tableView.DequeueReusableCell (CELL_ID, indexPath);
    return cell;
}
```

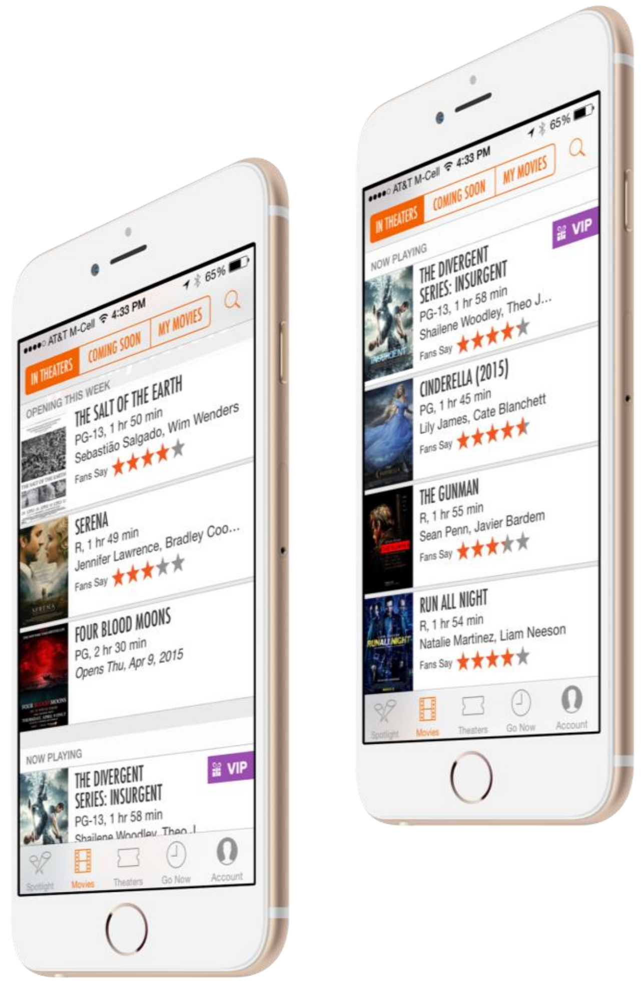


Group data in the Table View

Xamarin University

Tasks

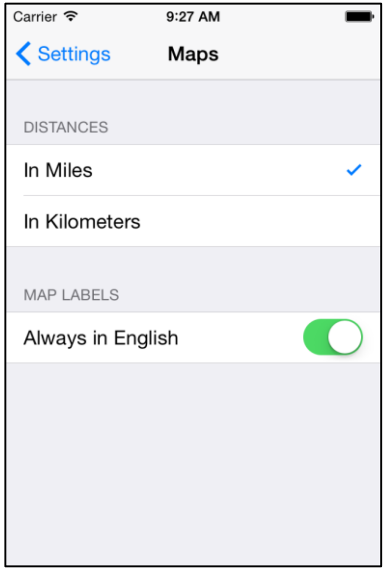
1. Changing the grouping visualization for a table view
2. Display an index
3. Add headers and footers
4. Customize headers and footers



Organizing the table view data

- The Table View has several built-in features which can be used to organize the data display and make it more accessible to the user

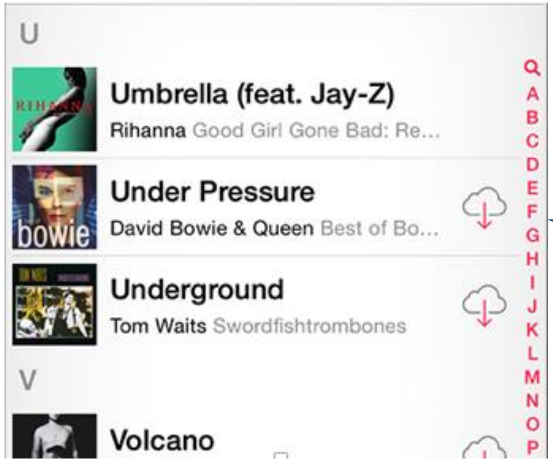
Table view cells can be organized into logical groups with headers



Xamarin University

Organizing the table view data

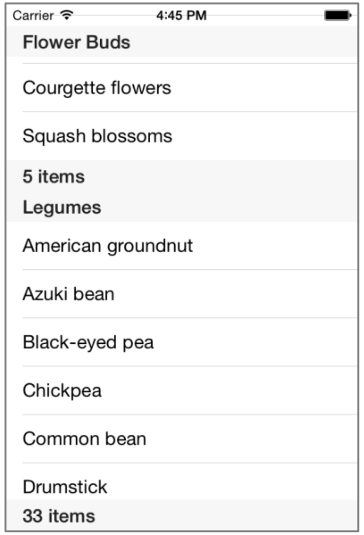
❖ The Table View has several built-in features which can be used to organize the data display and make it more accessible to the user


U
Umbrella (feat. Jay-Z)
Rihanna Good Girl Gone Bad: Re...
Under Pressure
David Bowie & Queen Best of Bo...
Underground
Tom Waits Swordfishtrombones
V
Volcano
A B C D E F G H I J K L M N O P

Can add an index for quicker navigation

Compare Plain vs. Grouped tables

- There is no behavioral difference between Plain and Grouped tables, you will choose one or the other based on how you want your UI to appear



Carrier 4:45 PM

Flower Buds

Courgette flowers

Squash blossoms

5 items

Legumes

American groundnut

Azuki bean

Black-eyed pea

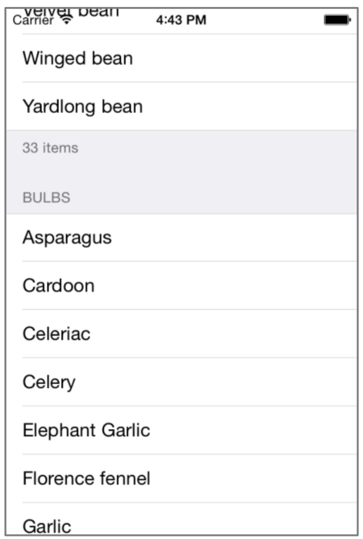
Chickpea

Common bean

Drumstick

33 items

Plain



Velvet bean

Carrier 4:43 PM

Winged bean

Yardlong bean

33 items

BULBS

Asparagus

Cardoon

Celeriac

Celery

Elephant Garlic

Florence fennel

Garlic

Grouped

Xamarin University

Set the table style

❖ The table style can be set in code or by using the designer

```
var tblView = new UITableView (frame: rc,
                               style: UITableViewStyle.Plain);
```

The style must be set when the table view is created and cannot be updated

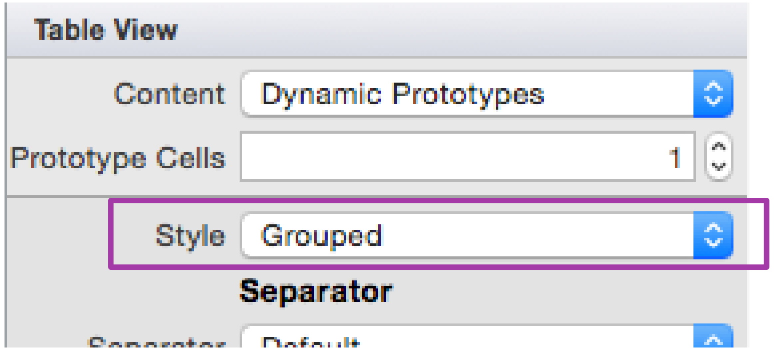


Table View

Content: Dynamic Prototypes

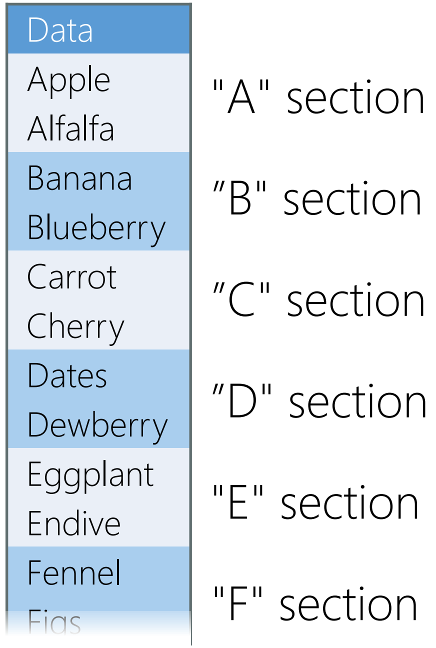
Prototype Cells: 1

Style: Grouped

Separator

What is a section?

- A *section* is a logical group in a list of data – the Table View displays each section in its own group
- You decide what the sections will be based on your data and its organization

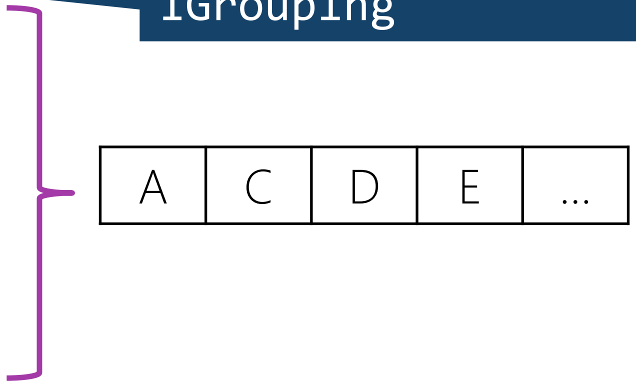


Section the data

- Sectioning organizes the data into logical groups (i.e. alphabetically)

Data	List position	Section index	Section label
Almond	0	0	A
Apple	1	0	A
Arugula	2	0	A
Avocado	3	0	A
Celery	4	1	C
Coconut	5	1	C
Dates	6	2	D

this is commonly stored in a `Dictionary<K,V>` or an `IGrouping`



Xamarin University

Providing grouped data to the table view

- The Table View Source must implement two additional methods to support a grouped Table View

`NumberOfSections`

`RowsInSection`

NumberOfSections

- The **NumberOfSections** method should return the number of groups to display – e.g. how many keys are in the dictionary, or how many partitions the data is split into

```
Dictionary<string, string[]> groupedFruit;

public override nint NumberOfSections (UITableView tableView)
{
    return groupedFruit.Keys.Length; // # of groups
}
```

RowsInSection

- The **`RowsInSection`** method identifies the number of rows (items) in a given section (group)

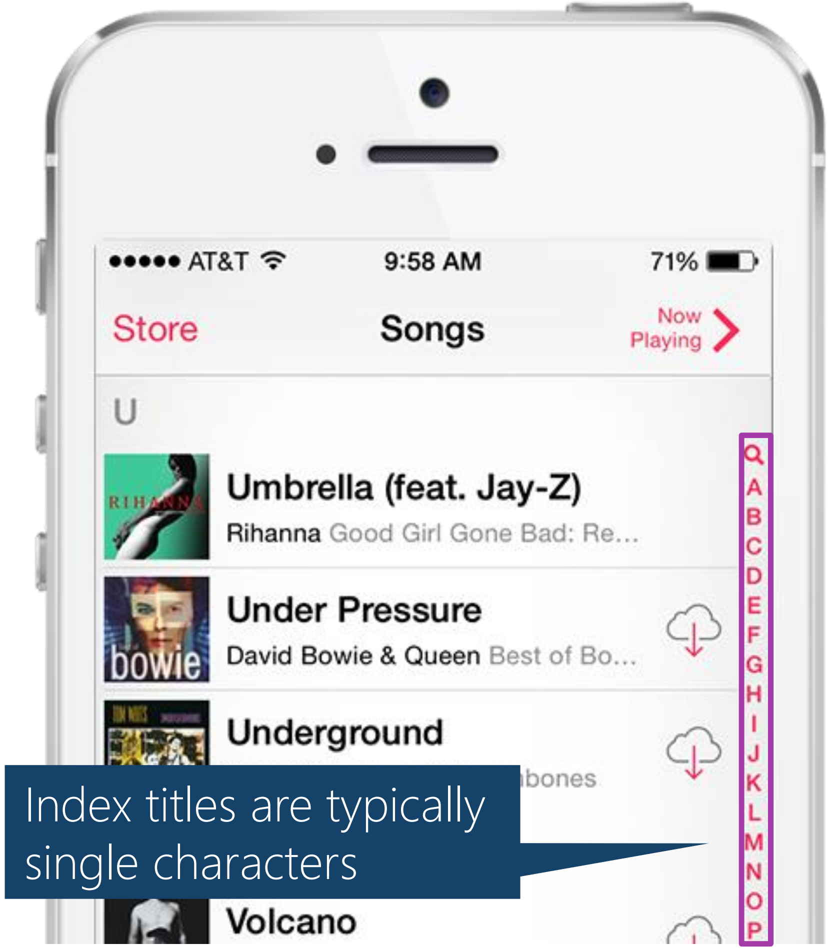
```
Dictionary<string, string[]> groupedFruit;

public override nint RowsInSection (UITableView tableview,
                                    nint section)
{
   // # of fruits in group
   keys = indexedTableItems.Keys.ToArray ();
   return groupedFruit[keys[section]].Count();
}
```

Xamarin University

Creating an index

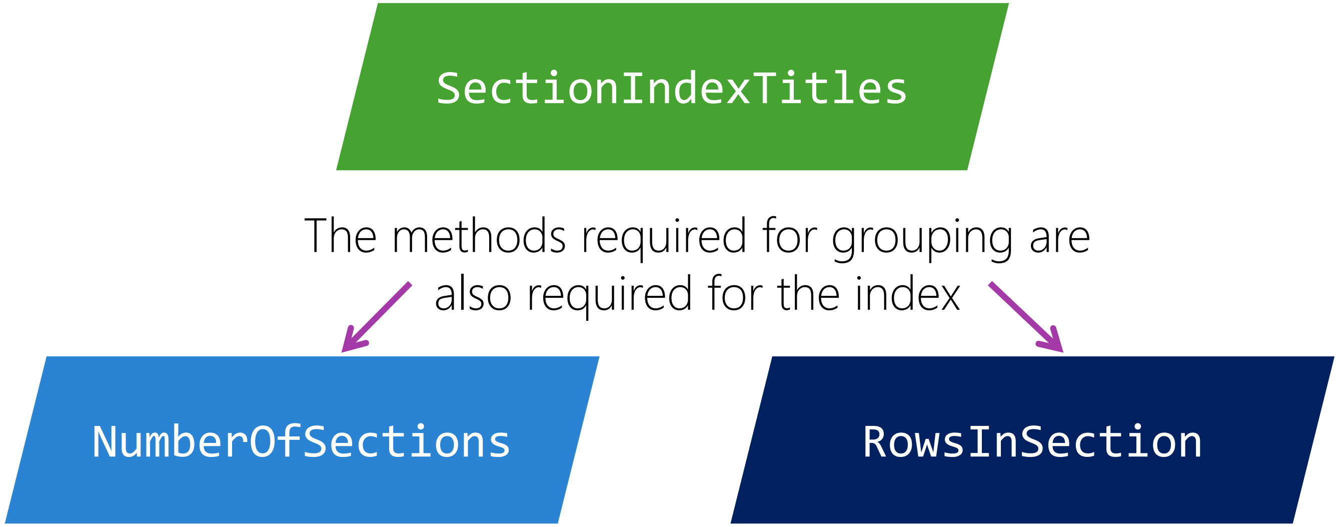
❖ An *index* can be added to the right side of a Table View for quicker navigation



Index titles are typically single characters

Populating the index

- To populate the index we need to override **`SectionIndexTitles`** in the table view controller


SectionIndexTitles
The methods required for grouping are also required for the index
NumberOfSections
RowsInSection

SectionIndexTitles

- The **`SectionIndexTitles`** method returns the array of strings that will be used to display the index

```
public override string[] SectionIndexTitles(UITableView tableView)
{
   return groupedFruit.Keys.ToArray();
}
```

Flash Quiz

Xamarin University

Flash Quiz

① An index can be used in which type of table view?

a) Plain
b) Grouped
c) Both

Flash Quiz

① An index can be used in which type of table view?

a) Plain

b) Grouped

c) Both

Flash Quiz

② You must set the Table View style to **`Grouped`** if **`NumberOfSections`** returns a value greater than 1

a) True

b) False

Flash Quiz

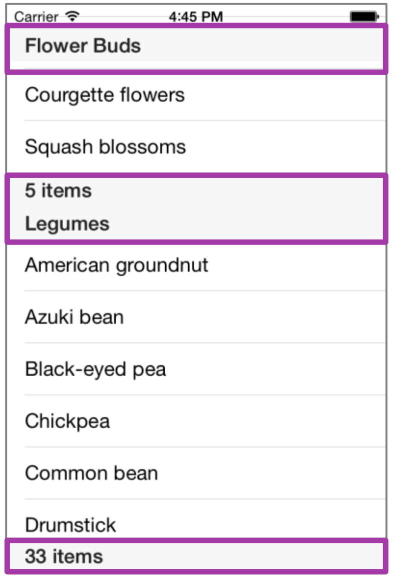
② You must set the Table View style to **`Grouped`** if **`NumberOfSections`** returns a value greater than 1

a) True

b) False

Headers and footers

- Table View supports both headers and footers on grouped sections



Carrier 4:45 PM

Flower Buds

Courgette flowers

Squash blossoms

5 items

Legumes

American groundnut

Azuki bean

Black-eyed pea

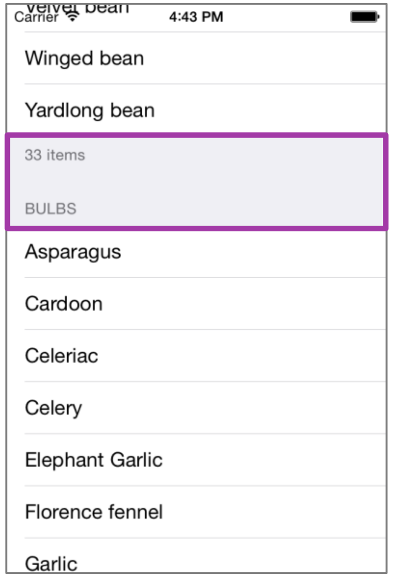
Chickpea

Common bean

Drumstick

33 items

Plain



Velvet bean

Carrier 4:43 PM

Winged bean

Yardlong bean

33 items

BULBS

Asparagus

Cardoon

Celeriac

Celery

Elephant Garlic

Florence fennel

Garlic

Grouped

Add headers and footers

- Displaying headers and footers requires additional methods

`TitleForHeader`

`TitleForFooter`

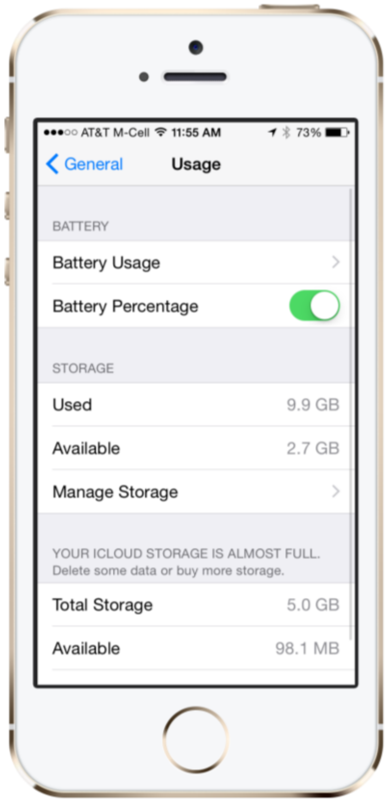
`GetViewForHeader`

`GetViewForFooter`

TitleForHeader

- **`TitleForHeader`** should return the string to show for the given section

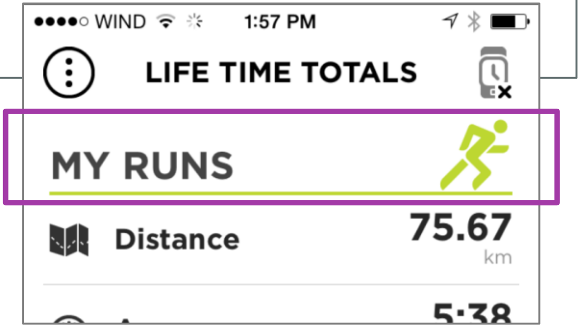
```
public override string TitleForHeader (
     UITableView tableView, nint section)
{
    return keys[section];
}
```



Customize the header

- You can customize the view for the header by using the **`GetViewForHeader`** method on the Table View source class

```
public override UIView GetViewForHeader (UITableView tableView,
                                         nint section) {
    if(section == 0)
        return BuildCustomHeaderView ("MY RUNS", "runner.png");
    ...
}
```



Individual Exercise

Create a grouped table with an index

Thank You!

Please complete the class survey in your profile:
university.xamarin.com/profile

Microsoft