IOS215

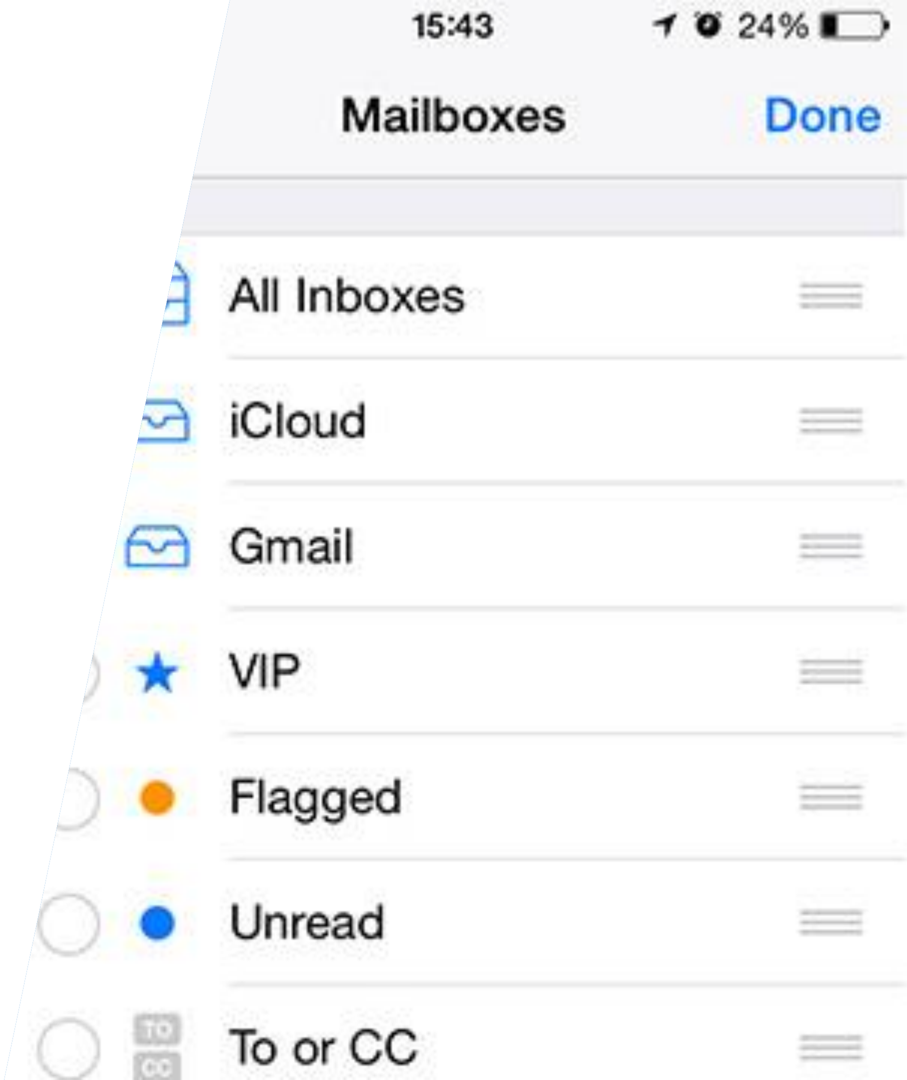
Editing Table Views

Download class materials from
university.xamarin.com

Microsoft Xamarin University

Objectives

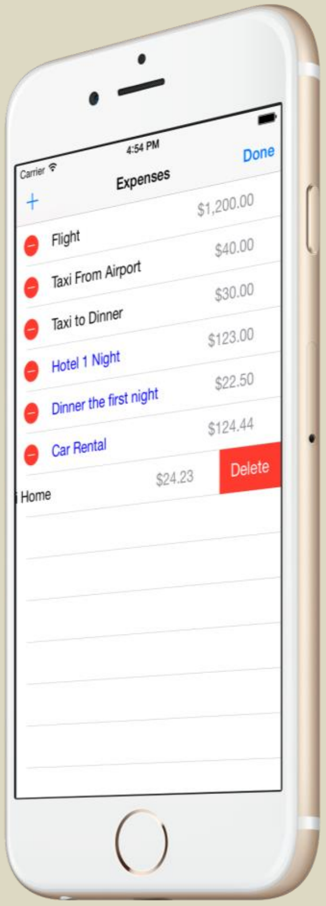
1. Work with built-in editing operations
2. Add support for modern editing operations
3. Integrate a search bar



Work with built-in editing operations

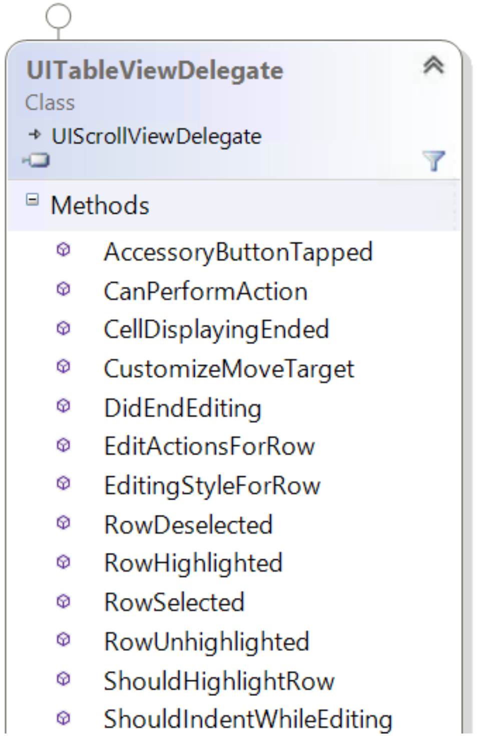
Tasks

1. Enable interactive editing
2. Enable internal editing
3. Re-order rows



Reminder: managing interactions

- `UITableViewDelegate` protocol provides notifications for interactions with the Table View
  - Row selection and highlighting
  - Editing actions
  - Swipe actions
  - Reordering rows
  - …
- Methods can be overridden through associated `UITableViewSource` or `UITableViewController`



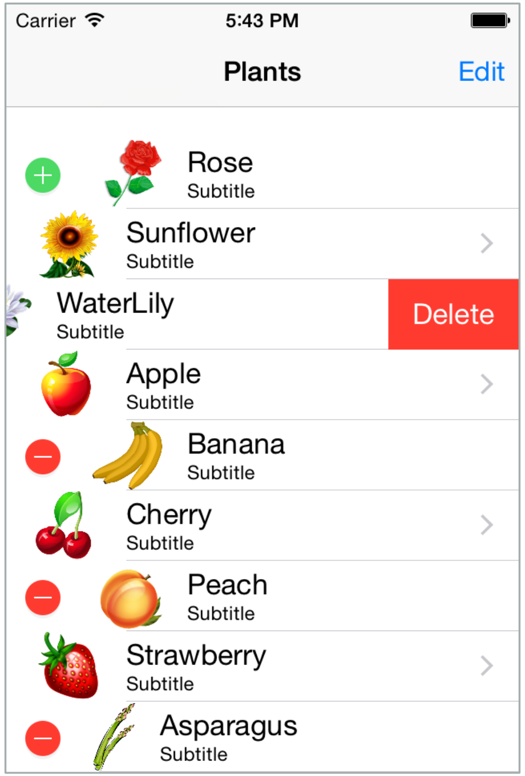
UITableViewDelegate
Class
UIScrollViewDelegate

Methods
- AccessoryButtonTapped
- CanPerformAction
- CellDisplayingEnded
- CustomizeMoveTarget
- DidEndEditing
- EditActionsForRow
- EditingStyleForRow
- RowDeselected
- RowHighlighted
- RowSelected
- RowUnhighlighted
- ShouldHighlightRow
- ShouldIndentWhileEditing

Interactive editing

- Table View supports an interactive **edit** mode which allows the user to add, delete and re-order rows through the UI

- App can decide on a row-by-row basis what editing operations are available

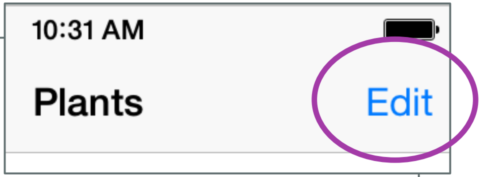


Xamarin University

Activating Edit mode

- Use the built-in Edit button when a navigation bar is present (i.e. when a `UINavigationController` is our root controller)

```
public override void ViewDidLoad()
{
    ...
    this.NavigationItem.RightBarButtonItem = this.EditButtonItem;
}
```



Best practice to use the built-in button definition if possible - when tapped, the button will automatically put the Table View into edit mode

Activating Edit mode

- Call **SetEditing** method on the Table View or **UIViewController** to turn on edit mode; this is what the built in **edit** button calls

```
void OnTableEdit(UIBarButtonItem sender)
{
    this.SetEditing(true, true);
}
```

On/Off

Pass true to *animate* the transition to/from edit mode

Detecting the current editing state

- `SetEditing` method is virtual and can be overridden to detect or influence the transition between interactive editing mode and normal mode

```
public override void SetEditing(bool editing, bool animated)
{
   base.SetEditing(editing, animated);
   ... // Custom code here
}
```

To quickly check whether a Table View is in editing mode, you can look at the `UITableView.Editing` boolean property

Implementing interactive editing

- Can override **`CanEditRow`** on the data source – this is called for each visible row to decide if any editing operations are available

```
public override bool CanEditRow(UITableView tableView,
                                NSIndexPath indexPath)
{
   PlantData plant = fruit[indexPath.Row];
   return plant.CanRemoveFromGarden;
}
```

return `true` / `false` whether the given row can be altered

Implementing interactive editing

- Use **`EditingStyleForRow`** override to decide how the row may be edited (the default is delete) – this is also called for each visible row to determine how it will be drawn in edit mode

```
public override UITableViewCellEditingStyle EditingStyleForRow(
                 UITableView tableView, NSIndexPath indexPath)
{
   if (indexPath.Row == 0)
      return UITableViewCellEditingStyle.Insert;
   return UITableViewCellEditingStyle.Delete;
}
```

Supports Insert, Delete and None

Individual Exercise

Turn on editing features on a Table View

Xamarin University

Handling edits to the data

- Table View calls the data source's **`CommitEditingStyle`** to notify application about edits performed by the user; that method must:

Perform the editing action to the data source

Refresh the Table View

Performing the editing action

- Must insert or remove the underlying data item from the collection being managed by the data source

```
public override void CommitEditingStyle(UITableView tableView,
     UITableViewCellEditingStyle editingStyle, NSIndexPath indexPath)
{
   if (editingStyle == UITableViewCellEditingStyle.Delete) {
      plants.RemoveAt(indexPath.Row);
      ...
   }
   else if (editingStyle == UITableViewCellEditingStyle.Insert) {
      plants.Insert(indexPath.Row, new Plant() { Name = "New Sapling" });
      ...
   }
}
```

Xamarin University

Updating the Table View

❖ Use the **InsertRows** and **DeleteRows** methods to add new visual cells to the Table View – make sure the **GetCell** method knows about the cells!

```
public override void CommitEditingStyle(UITableView tableView,
     UITableViewCellEditingStyle editingStyle, NSIndexPath indexPath)
{
   if (editingStyle == UITableViewCellEditingStyle.Delete) {
      plants.RemoveAt(indexPath.Row);
      TableView.DeleteRows(new[] { indexPath }, UITableViewRowAnimation.Automatic);
   }
   else if (editingStyle == UITableViewCellEditingStyle.Insert) {
      plants.Insert(indexPath.Row, new Plant() { Name = "New Sapling" });
      TableView.InsertRows(new[] { indexPath }, UITableViewRowAnimation.Fade);
   }
}
```

Xamarin University

Performing logic-driven updates

- If rows are being altered due to internal logic (vs. the user), *and* the number of rows is low, you should use the specific APIs to insert/delete rows

```
private void RemovePlant(Plant plant)
{
   int row = plants.IndexOf(plant);
   plants.RemoveAt(row);
   using (NSIndexPath indexPath = NSIndexPath.FromRowSection(row, 0)) {
      TableView.DeleteRows(new[] { indexPath },
            UITableViewRowAnimation.Middle);
   }
}
```

Must create `NSIndexPath` to identify row(s) being altered

A common technique is to create an *observable* table view source which updates the Table View when the underlying collection is changed

Xamarin University

Refreshing specific rows

- If the underlying data for a row is changed internally, call **`ReloadRows`** to get the Table View to refresh the row(s) if it is visible

```
void OnPlantUpdated(Plant plant)
{
   int row = plants.IndexOf(plant);
   using (NSIndexPath indexPath = NSIndexPath.FromRowSection(row, 0)) {
      this.TableView.ReloadRows(new[] { indexPath });
   }
}
```

Reloading the table view

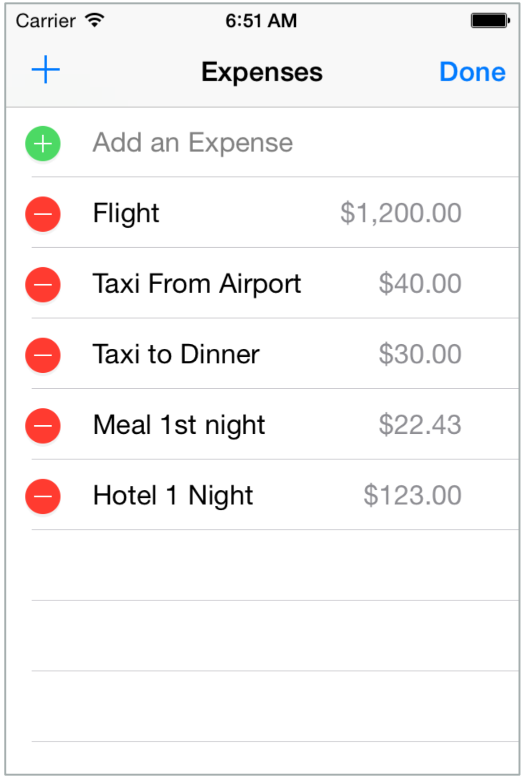
- If the entire data set is changed, you can force the Table View to reload everything using **`ReloadData`**

```
// Reload all from the data source (entire TableView is reloaded)
this.TableView.ReloadData();
```

What about inserting rows?

- Insert operations are a bit trickier to manage in the UI because you will often need to insert a "fake" row in the data to represent an insert

- Alternatively, a lot of applications add a button into the navigation bar to "add" a new row

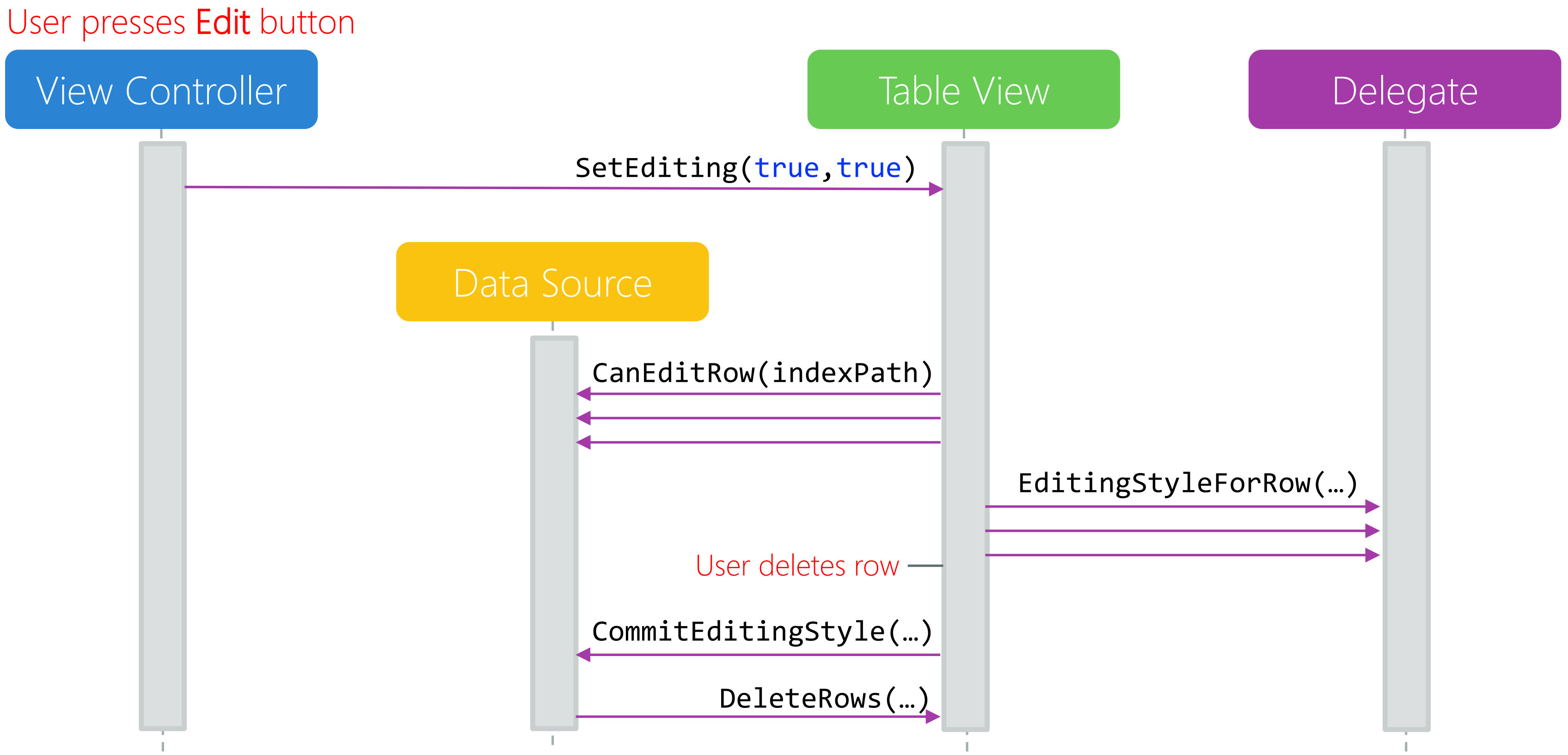


Flight	$1,200.00
Taxi From Airport	$40.00
Taxi to Dinner	$30.00
Meal 1st night	$22.43
Hotel 1 Night	$123.00

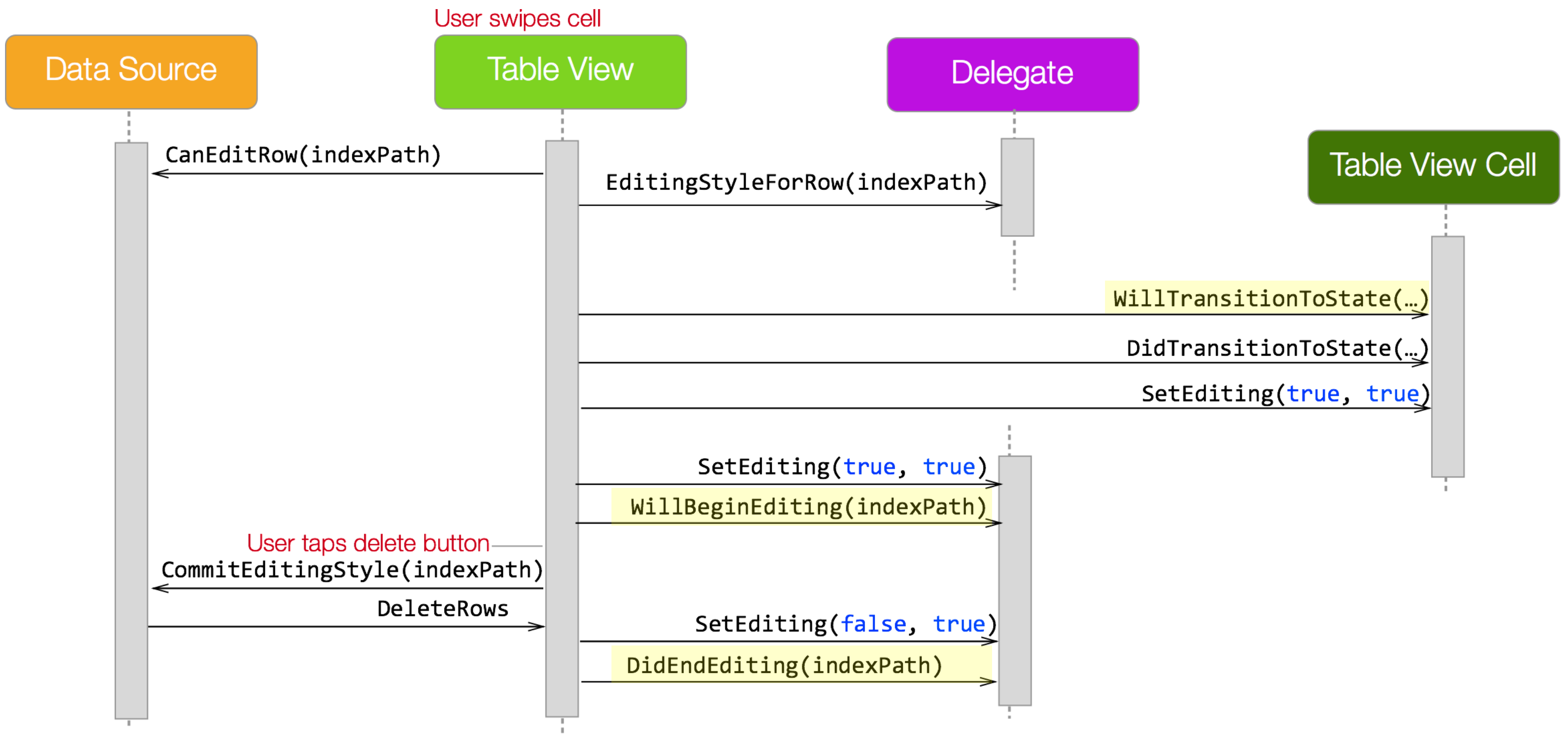
Individual Exercise

Committing editing operations

Recap: the big picture

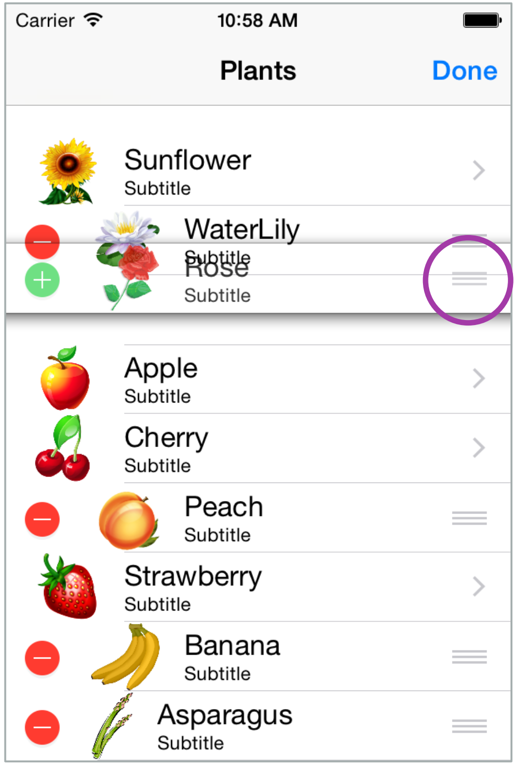


Swipe to Delete gesture workflow



Re-ordering rows

- Table View supports an interactive *re-ordering* capability which can be added to edit mode by overriding **`CanMoveRow`** and returning true for each row that can be moved
- This feature can only be turned on when the row is editable – you cannot have a non-editable, movable row
- For supported rows, a drag adorner is added to the right that the user can *grab* and move



Supporting row movement

- The UI is automatically changed when reorder occurs, must override **`MoveRow`** to update the underlying data positions

```
public override void MoveRow(UITableView tableView, NSIndexPath
sourceIndexPath, NSIndexPath destinationIndexPath)
{
    var plant = plants[sourceIndexPath.Row];
    plants.RemoveAt(sourceIndexPath.Row);
    plants.Insert(destinationIndexPath.Row, plant);
    // Table already updated automatically
}
```

This method *must* be overridden or the system will assume that reordering is not supported – even if **`CanMoveRows`** returns **`true`**

Xamarin University

Adjusting the row movement

- You can override **`CustomizeMoveTarget`** to enforce row ordering rules, passed proposed index and can return different index

```
public override NSIndexPath CustomizeMoveTarget(UITableView tableView,
       NSIndexPath sourceIndexPath, NSIndexPath proposedIndexPath)
{
   int validRow = ... // Perform logic to validate position
   return NSIndexPath.FromRowSection(validRow, 0);
}
```

Flash Quiz

Flash Quiz

① To turn on interactive editing you can _______ (Select all that apply)

a) Set UITableView.Editing = true
b) Override CanEditRow
c) Call UIViewController.SetEditing
d) Override EnterEditMode

Flash Quiz

① To turn on interactive editing you can _______ (Select all that apply)

a) <u>Set UITableView.Editing = true</u>

b) Override CanEditRow

c) <u>Call UIViewController.SetEditing</u>

d) Override EnterEditMode

Flash Quiz

② You must edit the underlying data source *and* the Table View when committing an editing operation

a) True

b) False

Flash Quiz

② You must edit the underlying data source *and* the Table View when committing an editing operation

a) True [✓]

b) False

Flash Quiz

③ To support interactive row movement, you should override the ______ and ________ methods

a) CanEditRow, CommitEditingStyle

b) CanMoveRow, MoveRow

c) CanMoveRow, CustomizeMoveTarget

Flash Quiz

③ To support interactive row movement, you should override the ______ and ________ methods

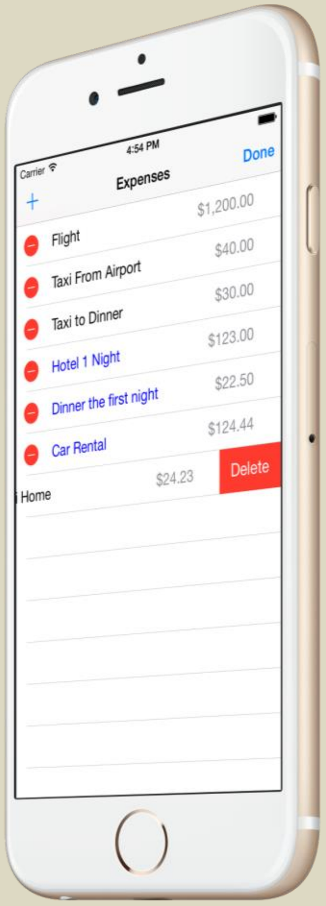
a) CanEditRow, CommitEditingStyle

b) <u>CanMoveRow, MoveRow</u>

c) CanMoveRow, CustomizeMoveTarget

Summary

1. Enable interactive editing
2. Enable internal editing
3. Re-order rows



Add support for modern editing operations

Xamarin University

Tasks

1. Add Table View Actions
2. Add Pull to refresh

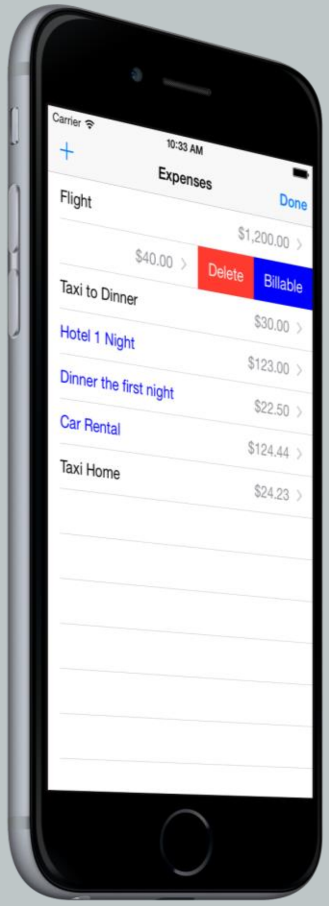
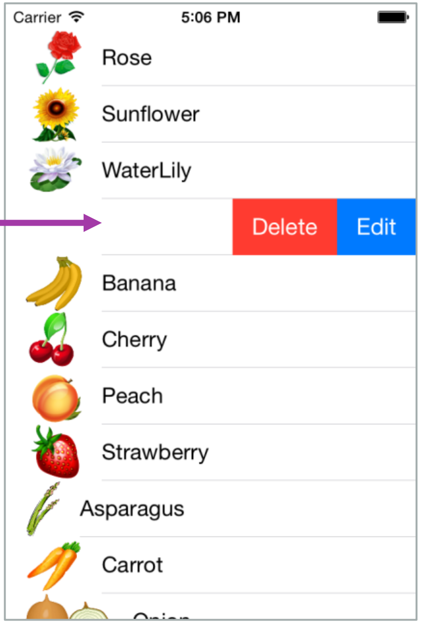


Table View actions

- Rows support **editing actions** which are shown using a *swipe-to-the-left* gesture

Displays buttons in a row, with a "More" action sheet for overflow



Adding Table View editing actions

- Editing actions are supplied through the **`EditActionsForRow`** override on the delegate; this is called when the row is swiped to determine if there are valid actions to display for this row

```
public override UITableViewRowAction[] EditActionsForRow(UITableView tableView,
                                                         NSIndexPath indexPath)
{
   var actions = new[] {
     UITableViewRowAction.Create(UITableViewRowActionStyle.Normal, "Edit", OnEdit),
     UITableViewRowAction.Create(UITableViewRowActionStyle.Destructive,"Delete",
                                                                     OnDelete),
   };
   actions[0].BackgroundColor = UIColor.Blue;
   return actions;
}
```

Adding Table View editing actions

- Editing actions are supplied through the **`EditActionsForRow`** override on the delegate; this is called when the row is swiped to determine if there are valid actions to display for this row

```
public override UITableViewRowAction[] EditActionsForRow(UITableView tableView,
                                                          NSIndexPath indexPath)
{
   var actions = new[] {
     UITableViewRowAction.Create(UITableViewRowActionStyle.Normal, "Edit", OnEdit),
     UITableViewRowAction.Create(UITableViewRowActionStyle.Destructive,"Delete",
                                                                   OnDelete),
   };
   actions[0].BackgroundColor = UIColor.Blue;
   return actions;
}
```

Each action has a style, text and function to execute

Xamarin University

Adding Table View editing actions

- Editing actions are supplied through the **`EditActionsForRow`** override on the delegate; this is called when the row is swiped to determine if there are valid actions to display for this row

```
public override UITableViewRowAction[] EditActionsForRow(UITableView tableView,
                                                         NSIndexPath indexPath)
{
   var actions = new[] {
     UITableViewRowAction.Create(UITableViewRowActionStyle.Normal, "Edit", OnEdit),
     UITableViewRowAction.Create(UITableViewRowActionStyle.Destructive,"Delete",
                                                                     OnDelete),
   };
   actions[0].BackgroundColor = UIColor.Blue;
   return actions;
}
```

You can also customize some aspects of the button UI

Caching off table view actions

- **`EditActionsForRow`** is called uniquely for each row; it is more efficient to create a single set of row actions if they are always the same

```
UITableViewRowAction[] actions;
public override UITableViewRowAction[] EditActionsForRow(UITableView tableView,
                                                          NSIndexPath indexPath)
{
   if (actions == null) {
      actions = new[] {
         UITableViewRowAction.Create(UITableViewRowActionStyle.Normal, "Edit", OnEdit),
         UITableViewRowAction.Create(UITableViewRowActionStyle.Destructive,"Delete", OnDelete),
      };
      actions[0].BackgroundColor = UIColor.Blue;
   }
   return actions;
}
```

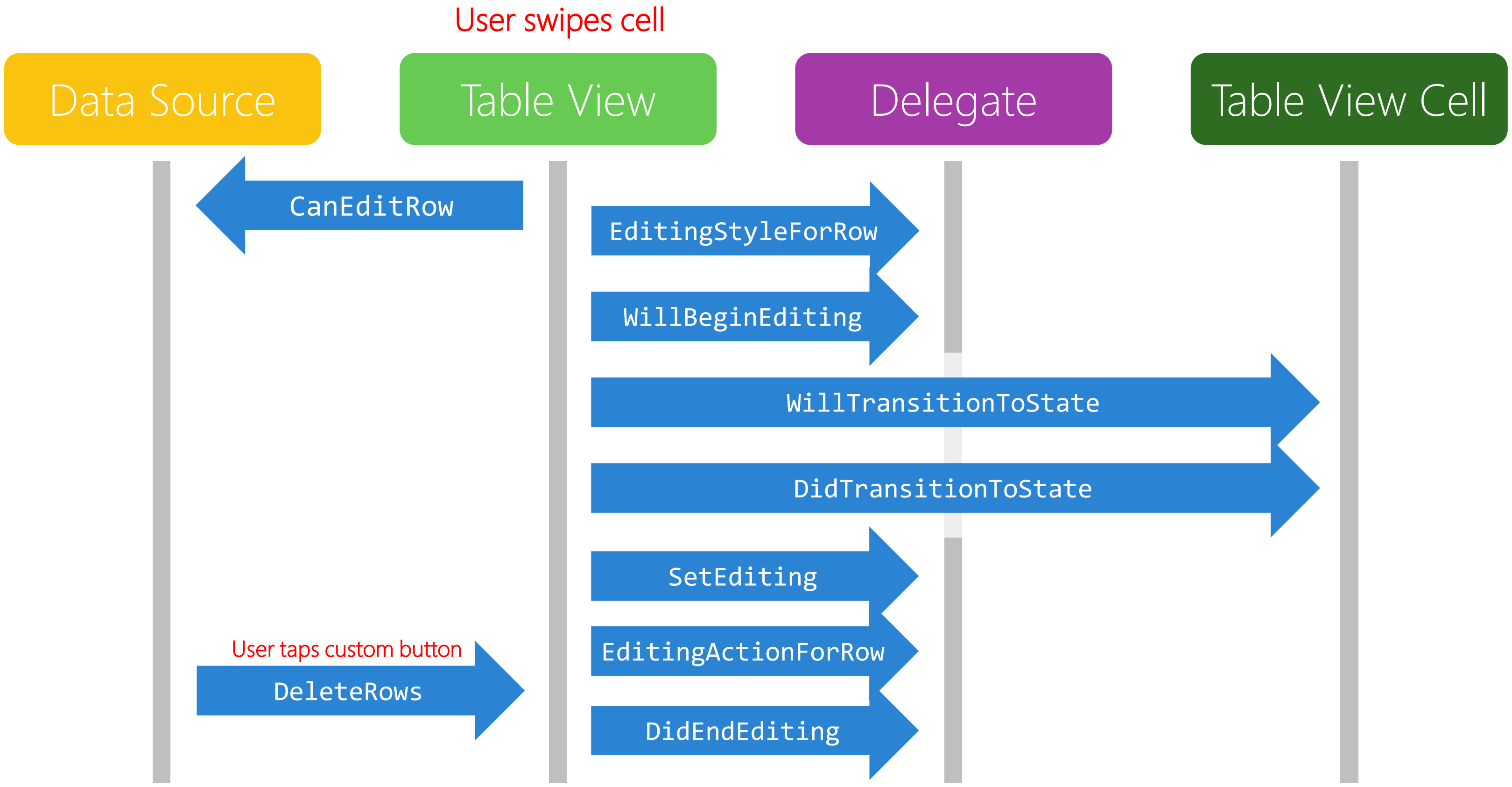
Responding to a row action

- Each action is associated with a .NET delegate function that is invoked if the user selects that action on the row

```
private void OnDelete(UITableViewRowAction action, NSIndexPath index)
{
    plants.RemoveAt(index.Row); // remove from data
    TableView.DeleteRows(new[] { indexPath },
                         UITableViewRowAnimation.Automatic);
}
```

Xamarin University

Row action "editing" mode



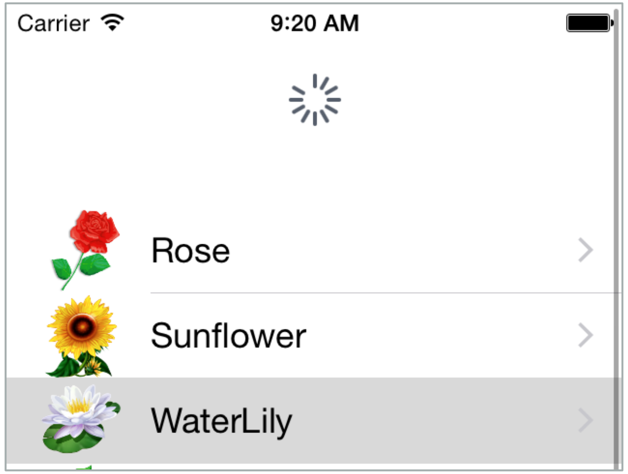
Individual Exercise

Add support for swipe-gesture edit actions

Xamarin University

Pull to refresh

- Popular addition to Table Views that display external data is a "pull-to-refresh" gesture to get new data from the external source



Refresh is activated by "pulling down" on the Table View – indicator is shown while the data is being updated

Implementing Pull to refresh

- Support built into **`UITableViewController`**, just has to be enabled

```
public override void ViewDidLoad()
{
    base.ViewDidLoad();
    ...
    this.RefreshControl = new UIRefreshControl();
}
```

1

Assign a new **`UIRefreshControl`** to the **`RefreshControl`** property of the Table View Controller

Xamarin University

Implementing Pull to refresh

- Support built into **`UITableViewController`**, just has to be enabled

```
public override void ViewDidLoad()
{
    base.ViewDidLoad();
    ...
    this.RefreshControl = new UIRefreshControl();
    this.RefreshControl.ValueChanged += (sender, e) => {
        // Refresh contents of underlying data
        ReloadPlantArrayFromWebService();


          ...
    };
}
```

2

Hook the **`ValueChanged`** event and update the underlying data source when it is raised

Xamarin University

Implementing Pull to refresh

- Support is built into **`UITableViewController`**, just has to be enabled

```
public override void ViewDidLoad()
{
   base.ViewDidLoad();
   ...
   this.RefreshControl = new UIRefreshControl();
   this.RefreshControl.ValueChanged += (sender, e) => {
      // Refresh contents of underlying data
      ...
      BeginInvokeOnMainThread(() => {
          this.RefreshControl.EndRefreshing();
      });
   };
}
```

3

When the data has been refreshed, **signal that the refresh is complete to hide the refresh UI**

Xamarin University

Implementing Pull to refresh

- Support is built into **`UITableViewController`**, just has to be enabled

```
public override void ViewDidLoad()
{
   base.ViewDidLoad();
   ...
   this.RefreshControl = new UIRefreshControl();
   this.RefreshControl.ValueChanged += (sender, e) => {
        // Refresh contents of underlying data
        ...
        BeginInvokeOnMainThread(() => {
             this.RefreshControl.EndRefreshing();
             TableView.ReloadData();
        });
...
```

4

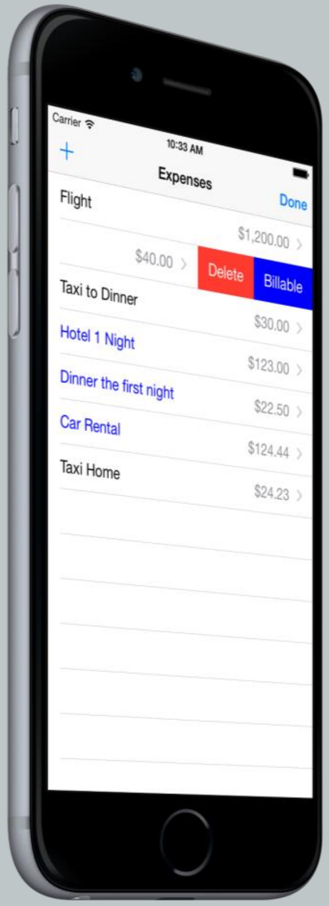
Finally, must refresh the Table View – very common to just use **`ReloadData`**

Individual Exercise

Add support for the "pull-to-refresh" gesture

Summary

1. Add Table View Actions
2. Add Pull to refresh

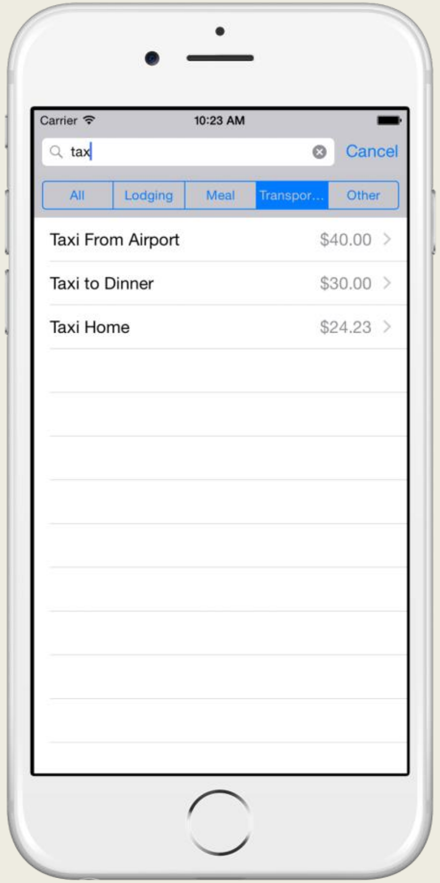


Integrate a search bar

Xamarin University

Tasks

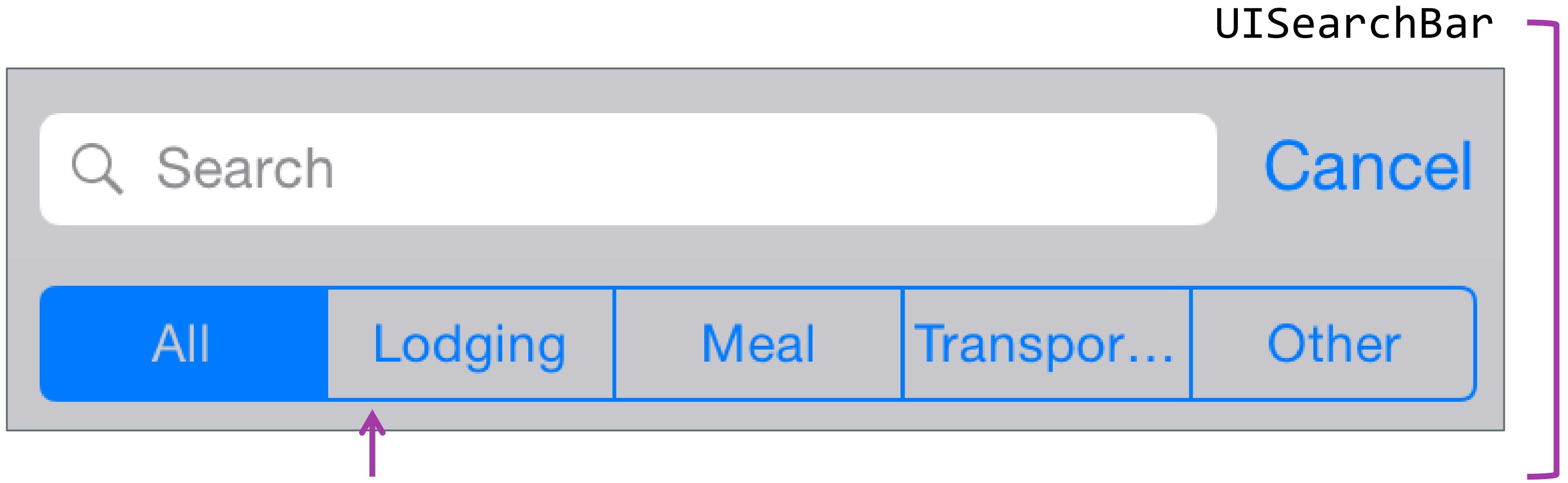
1. Add a Search bar to a Table View
2. Limit search scope



Taxi From Airport	$40.00
Taxi to Dinner	$30.00
Taxi Home	$24.23

Adding a Search Bar

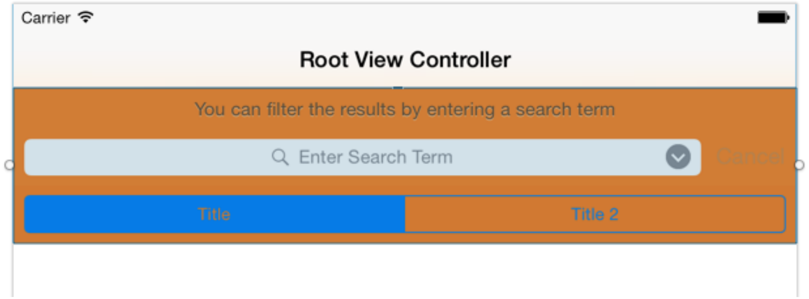
- iOS supports a built-in textual search UI which can accept a search term and optional scope for the search



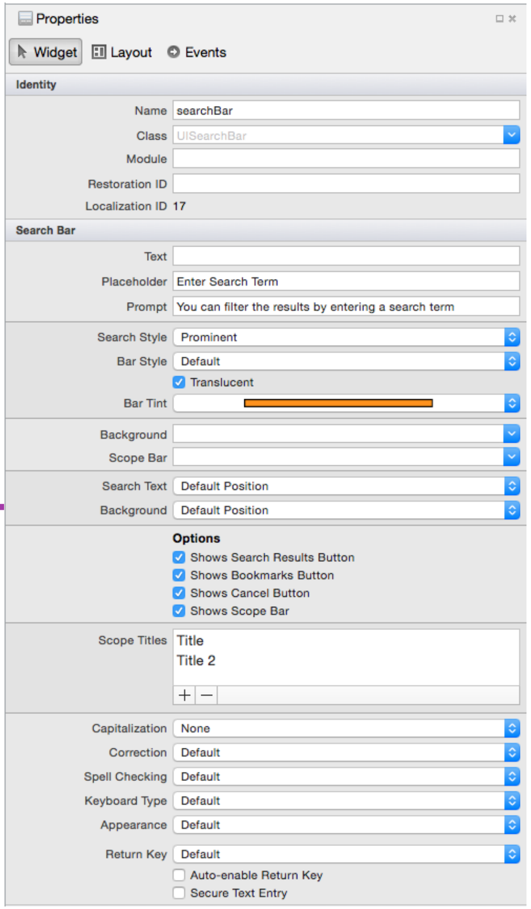
Scope Bar allows search to be restricted to a set of categories

Working with UISearchBar

- Can add **`UISearchBar`** into your UI through designer or code



Properties can be set to adjust the visualization and behavior of the search bar



Identity

Name	searchBar
Class	UISearchBar
Module	
Restoration ID	
Localization ID	17

Search Bar

Text	
Placeholder	Enter Search Term
Prompt	You can filter the results by entering a search term
Search Style	Prominent
Bar Style	Default
	☑ Translucent
Bar Tint	🟧
Background	
Scope Bar	
Search Text	Default Position
Background	Default Position

Options

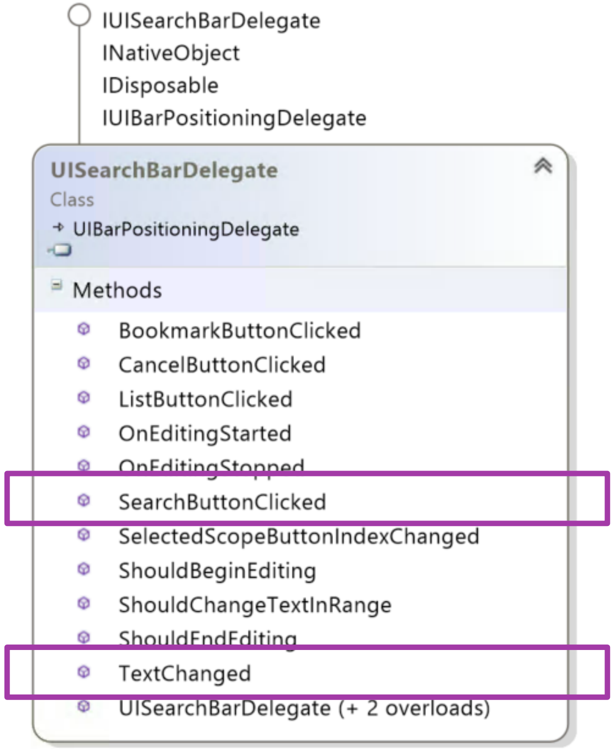
- ☑ Shows Search Results Button
- ☑ Shows Bookmarks Button
- ☑ Shows Cancel Button
- ☑ Shows Scope Bar

Scope Titles	Title Title 2
Capitalization	None
Correction	Default
Spell Checking	Default
Keyboard Type	Default
Appearance	Default
Return Key	Default

- ☐ Auto-enable Return Key
- ☐ Secure Text Entry

UISearchBar notifications

- Assign the **`WeakDelegate`** property to a **`UISearchBarDelegate`** implementation to handle the search logic
- Can trigger off the search button being tapped on the keyboard, or when the text is changed in the search box



Adding search support

- Apple has provided a presentation controller in **`UISearchController`** which will handle all the UI logic of showing and hiding the search bar automatically.

Usable anytime you need a search experience

Fully adaptive

Fully customizable

Using UISearchController [Step 1]

- Create a new **`UISearchController`** to manage the search experience, this is typically done in **`ViewDidLoad`**

```
UISearchController searchController;

public override void ViewDidLoad()
{
    ...
    searchController = new UISearchController(
                           (UIViewController)null);
    searchController.DimsBackgroundDuringPresentation = false;
    ...
```

Can pass a **`ViewController`** instance to use for results, or **`null`** to use the owner – must cast to get the right constructor!

Xamarin University

Using UISearchController [Step 2]

- Set the **`TableHeaderView`** property to put the **`UISearchBar`** above the Table View

```
UISearchController searchController;

public override void ViewDidLoad()
{
    ...
    TableView.TableHeaderView = searchController.SearchBar;



}
```

Using UISearchController [Step 2]

- Set the **`TableHeaderView`** property to put the **`UISearchBar`** above the Table View

```
UISearchController searchController;

public override void ViewDidLoad()
{
    ...
    TableView.TableViewHeader = searchController;
    searchController.SearchBar.SizeToFit();


}
```

Xamarin University

Using UISearchController [Step 3]

- Might need to set **`DefinesPresentationContext`** to **`true`** on the parent controller to ensure the search bar content is properly presented

```
UISearchController searchController;

public override void ViewDidLoad()
{
    ...
    TableView.TableViewHeader = searchController;
    searchController.SearchBar.SizeToFit();
    DefinesPresentationContext = true;
    ...
}
```

Using UISearchController [Step 4]

❖ Search notifications are reported through the `SearchResultsUpdater` property – this must be set to a `IUISearchResultsUpdating` implementation

```
searchController.SearchResultsUpdater = this;
```

```
public void UpdateSearchResultsForSearchController(
            UISearchController searchController)
{
   string textToSearchFor = searchController.SearchBar.Text;
   // ... Do filtering using text
}
```

Using UISearchController [Step 5]

- Search controller can *reuse* the existing table view – must ensure that **`GetCell`** and **`RowsInSection`** are working against the filtered data; can determine whether we are filtering by the **`Active`** property in the `UpdateSearchResultsForSearchController` callback

```
public void UpdateSearchResultsForSearchController(
                UISearchController searchController)
{
   if (searchController.Active)
     // Setup filtered data
   else
     // Reset to full dataset
}
```

Individual Exercise

Add search support to a Table View Controller

Configuring the UISearchBar

- Can use the **`SearchBar`** property to configure the search UI (colors, features, etc.) as well as wire up the **`WeakDelegate`** to handle notifications directly from the search bar itself

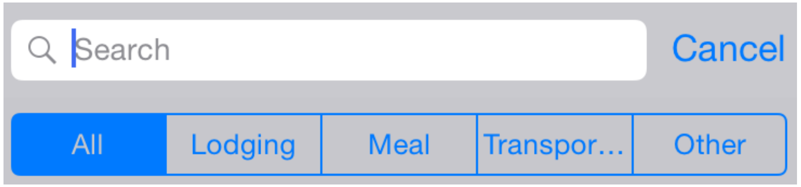
```
UISearchController searchController;

public override void ViewDidLoad()
{
    ...
    searchController.SearchBar.BarTintColor = UIColor.Green;
    searchController.SearchBar.WeakDelegate = this;
}
```

Providing searching categories

- Scope Bar is a built-in UI element included with the **`UISearchBar`**
- Can set specific categories using **`ScopeButtonTitles`** property
- Scope bar is shown and hidden automatically, but can be configured to always show with **`ShowsScopeBar`**

```
searchController
    .SearchBar
    .ScopeButtonTitles = new[] {
          "All",
          "Lodging",
          "Meal",
          "Transportation",
          "Other"
};
```



Responding to scope changes

- Changes to the scope are reported through the search bar's delegate

```
class SearchDelegate : UISearchBarDelegate {
   public Action<string,string> DoFilter { get; set; }
   public override void  SelectedScopeButtonIndexChanged(
            UISearchBar searchBar, int selectedScope) {
       string scope = searchBar.ScopeButtonTitles[selectedScope];
       string text = searchBar.Text;
       DoFilter(text, scope);
   }
}
```

```
searchController.SearchBar.WeakDelegate = new SearchDelegate() {
                                                   DoFilter = ... };
```

Xamarin University

Responding to scope changes

❖ Changes to the scope are reported through the search bar's delegate

```
searchController.SearchBar.WeakDelegate = this;
```

```
[Export ("searchBar:selectedScopeButtonIndexDidChange:")]
public virtual void ScopeButtonChanged(UISearchBar searchBar,
                                       int selectedScope)
{
   string scope = searchBar.ScopeButtonTitles[selectedScope];
   string text = searchBar.Text;
   // ... Provide filter based on text + scope
}
```

Xamarin University

Responding to scope changes

- Changes to the scope are reported through the search bar's delegate

```
searchController.SearchBar.WeakDelegate = this;
```

```
[Export ("searchBar:selectedScopeButtonIndexDidChange:")]
public virtual void ScopeButtonChanged(UISearchBar searchBar,
                                       int selectedScope)
{
    stri                    opeButtonTitles[selectedScope];
    stri                    t;
    // .                     on text + scope
}
```

Export signature must match the Objective-C definition in the Apple SDK

Responding to scope changes

❖ Changes to the scope are reported through the search bar's delegate

```
searchController.SearchBar.WeakDelegate = this;
```

```
[Export ("searchBar:selectedScopeButtonIndexDidChange:")]
public virtual void ScopeButtonChanged(UISearchBar searchBar,
                                       int selectedScope)
{
                           r.ScopeButtonTitles[selectedScope];
                           .Text;
                           ased on text + scope
}
```

Method should be virtual and have a compatible C# signature to the export

Xamarin University

Responding to scope changes

- Retrieve the currently selected scope from the **ScopeBar** to filter the results when the text is changed

```
public void UpdateSearchResultsForSearchController(
          UISearchController searchController)
{
   string textToSearchFor = searchController.SearchBar.Text;
   int index = searchController.SearchBar.SelectedScopeButtonIndex;
   string scope = searchController.SearchBar.ScopeButtonTitles[index];
   // ... Do filtering using text + scope
}
```

Flash Quiz

Flash Quiz

① To add search capabilities to a Table View, you should use ________

a) UISearchController

b) UISearchViewController

c) UISearchDisplayController

d) UISearchTableViewController

Xamarin University

Flash Quiz

① To add search capabilities to a Table View, you should use ________

a) UISearchController

b) UISearchViewController

c) UISearchDisplayController

d) UISearchTableViewController

Flash Quiz

② You can use the UISearchBar control without a search controller

a) True

b) False

Flash Quiz

② You can use the UISearchBar control without a search controller

a) [✓ True]

b) False

Xamarin University

Flash Quiz

③ To support searching capabilities with `UISearchController`, you must implement which interface?

a) IUISearchResultsUpdating

b) IUISearchBarDelegate

c) IUISearchDisplayDelegate

Flash Quiz

③ To support searching capabilities with **`UISearchController`**, you must implement which interface?

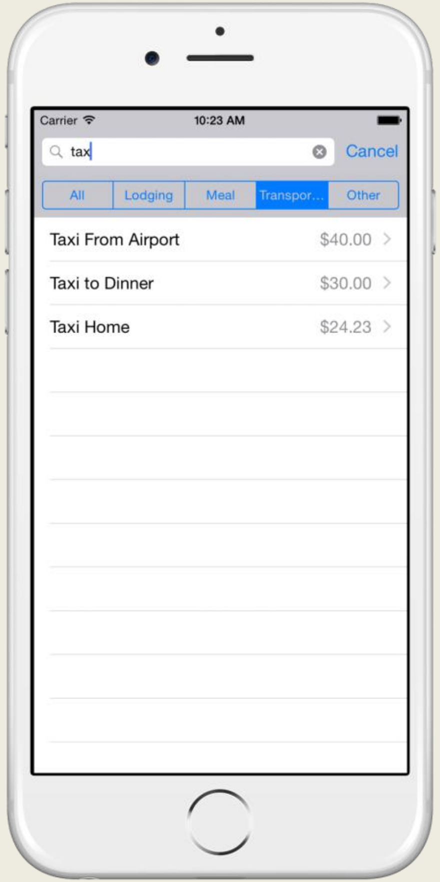
a) IUISearchResultsUpdating

b) IUISearchBarDelegate

c) IUISearchDisplayDelegate

Summary

1. Add a Search bar to a Table View
2. Limit search scope



Taxi From Airport $40.00
Taxi to Dinner $30.00
Taxi Home $24.23

Homework

Scope your searches with a scope bar

Xamarin University

Thank You!

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

Microsoft