iPhone Programming
Table/Collection Views

Tim Gegg-Harrison

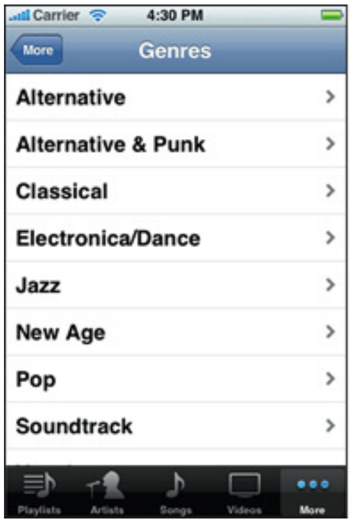
Table Views

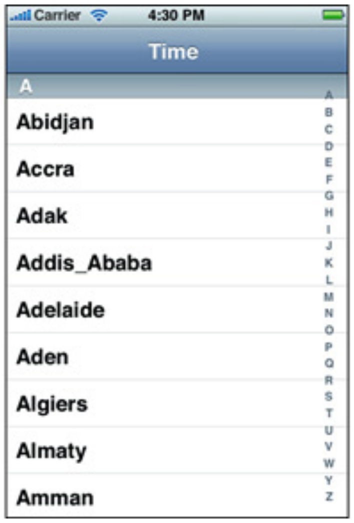
- `UITableView` is a subclass of `UIScrollView`
  - Table view allows for a single column and zero or more rows
- `UITableViewController` is a subclass of `UIViewController` that creates and manages a `UITableView` instance
  - Conforms to `UITableViewDelegate` and `UITableViewDataSource` protocols

Table Views…

Table views have many purposes:

- To let users navigate through hierarchically structured data
- To present an indexed list of items
- To display detail information and controls in visually distinct groupings
- To present a selectable list of options





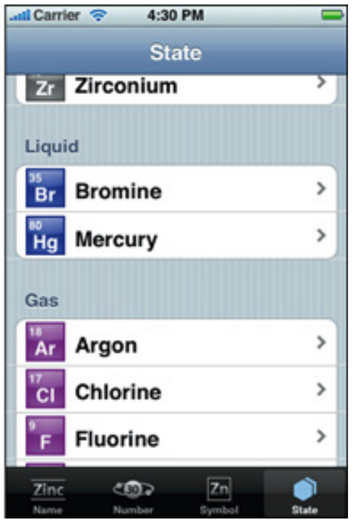


Table Views…

- A table view has only one column and allows vertical scrolling only.
- It consists of rows in sections.
  - Each section can have a header and a footer that displays text or an image.
  - Many table views have only one section with no visible header or footer.
- The `UIKit` framework identifies rows and sections through their index number:
  - Sections are numbered 0 through n – 1 from the top of a table view to the bottom.
  - Rows are numbered 0 through n – 1 within a section.
- A table view can have its own header and footer, distinct from any section.
  - The table header appears before the first row of the first section, and the table footer appears after the last row of the last section.

Table Views…

- When the table view goes into editing mode, the editing control for each cell object (if it's configured to have such a control) appears on its left side.
- The editing control can be either a deletion control (a red minus sign inside a circle) or an insertion control (a green plus sign inside a circle). The cell's content is pushed toward the right to make room for the editing control.

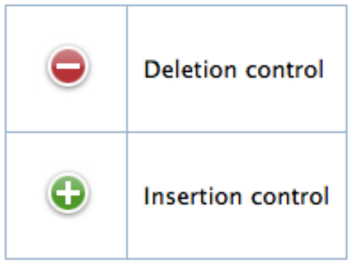


Table Views…

- If a cell object is reusable—the typical case—you assign it a reuse identifier (an arbitrary string).
  - At runtime, the table view stores cell objects in an internal queue.
  - When the table view asks the data source to configure a cell object for display, the data source can access the queued object by sending a `dequeueReusableCellWithIdentifier:` message to the table view, passing in a reuse identifier.
  - The data source sets the content of the cell and any special properties before returning it.
  - This reuse of cell objects is a performance enhancement because it eliminates the overhead of cell creation.
- With multiple cell objects in a queue, each with its own identifier, you can have table views constructed from cell objects of different types.
  - For example, some rows of a table view can have content based on the image and text properties of a `UITableViewCell` in a predefined style, while other rows can be based on a customized `UITableViewCell` that defines a special format for its content.

Table Views…

- Note that when the user selects a cell, you should respond by deselecting the previously selected cell (by calling the **deselectRow(at:animated:)** function) as well as by performing any appropriate action.

- If you respond to the the selection of a cell by pushing a new view controller onto the stack, you should deselect the cell (with animation) when the view controller is popped off the stack.

Table Views…

- Inserting, deleting, and moving rows in the table:

```
func insertRows(at: [IndexPath], with: UITableViewRowAnimation)
```
Inserts rows in the table view at the locations identified by an array of index paths, with an option to animate the insertion.

```
func deleteRows(at: [IndexPath], with: UITableViewRowAnimation)
```
Deletes the rows specified by an array of index paths, with an option to animate the deletion.

```
func moveRow(at: IndexPath, to: IndexPath)
```
Moves the row at a specified location to a destination location.

Table Views…

- Inserting, deleting, and moving sections in the table:

```
func insertSections(IndexSet, with: UITableViewRowAnimation)
    Inserts one or more sections in the table view, with an option to animate the insertion.

func deleteSections(IndexSet, with: UITableViewRowAnimation)
    Deletes one or more sections in the table view, with an option to animate the deletion.

func moveSection(Int, toSection: Int)
    Moves a section to a new location in the table view.
```

UICollectionView

- Introduced in iOS 6
- Manages an ordered collection of data
- UICollectionView takes the familiar patterns of UITableView and generalizes them to make any layout possible

UICollectionView

- Like UITableView, UICollectionView is a UIScrollView subclass that manages a collection of ordered items
- Items are managed by a *data source (*UICollectionViewDataSource*)*, which provides a representative cell view at a particular index path
- Unlike UITableView, however, UICollectionView is not constrained to a vertical, single-column layout
  - Instead, a collection view has a *layout* object, which determines the position of each subview, similar to a data source in some respects

UICollectionViewCell

- This is equivalent to UITableViewCell – it has an indexPath property which defines which row and section it belongs to; and various other properties to define the visual appearance

- Unlike UITableViewCell, UICollectionViewCell doesn't have any predefined types – you have to setup the cell from scratch

UICollectionViewLayout

- Most applications will opt to use or subclass UICollectionViewFlowLayout
- Flow layouts cover the broad class of layouts with some notion of linearity, whether that's a single row or column or a grid
- You must then implement delegate methods as appropriate to the specific layout
- You can then customize further by overriding properties or subclassing completely

UICollectionViewDataSource

- Much like UITableViewDataSource, the UICollectionViewDataSource is responsible for providing cells to the collection view on demand

- It's defined by the UICollectionViewDataSource protocol – there are several required methods and a whole range of optional ones

UICollectionViewDelegate

- UICollectionViewDelegate works in much the same way as UITableViewDelegate
- It's responsible for handling user interaction amongst other things, and is defined by the UICollectionViewDelegate protocol