UITableView

A view that presents data using rows arranged in a single column.

Declaration

```
class UITableView : UIScrollView
```

Overview

A table view displays a list of items in a single column. `UITableView` is a subclass of `UIScrollView`, which allows users to scroll through the table, although `UITableView` allows vertical scrolling only. The cells comprising the individual items of the table are `UITableViewCell` objects; `UITableView` uses these objects to draw the visible rows of the table. Cells have content—titles and images—and can have, near the right edge, accessory views. Standard accessory views are disclosure indicators or detail disclosure buttons; the former leads to the next level in a data hierarchy and the latter leads to a detailed view of a selected item. Accessory views can also be framework controls, such as switches and sliders, or can be custom views. Table views can enter an editing mode where users can insert, delete, and reorder rows of the table.

A table view is made up of zero or more sections, each with its own rows. Sections are identified by their index number within the table view, and rows are identified by their index number within a section. Any section can optionally be preceded by a section header, and optionally be followed by a section footer.

Table views can have one of two styles, `UITableView.Style.plain` and `UITableView.Style.grouped`. When you create a `UITableView` instance you must specify a table style, and this style cannot be changed. In the plain style, section headers and footers float above the content if the part of a complete section is visible. A table view can have an index that appears as a bar on the right hand side of the table (for example, "A" through "Z"). You can touch a particular label to jump to the target section. The grouped style of table view provides a default background color and a default background view for all cells. The background view provides a visual grouping for all cells in a particular section. For example, one group could be a person's name and title, another group for phone numbers that the person uses, and another group for email accounts and so on. See the Settings application for examples of grouped tables. Table views in the grouped style cannot have an index.

Many methods of `UITableView` take `NSIndexPath` objects as parameters and return values. `UITableView` declares a category on `NSIndexPath` that enables you to get the represented row index (`row` property) and section index (`section` property), and to construct an index path from a given row index and section index (`init(row:section:)` method). Especially in table views with multiple sections, you must evaluate the section index before identifying a row by its index number.

A `UITableView` object must have an object that acts as a data source and an object that acts as a delegate; typically these objects are either the application delegate or, more frequently, a custom `UITableViewController` object. The data source must adopt the `UITableViewDataSource` protocol and the delegate must adopt the `UITableViewDelegate` protocol. The data source provides information that `UITableView` needs to construct tables and manages the data model when rows of a table are inserted, deleted, or reordered. The delegate manages table row configuration and selection, row reordering, highlighting, accessory views, and editing operations.

When sent a `setEditing(_:animated:)` message (with a first parameter of `true`), the table view enters into editing mode where it shows the editing or reordering controls of each visible row, depending on the `editingStyle` of each associated `UITableViewCell`. Clicking on the insertion or deletion control causes the data source to receive a `tableView(_:commit:forRowAt:)` message. You commit a deletion or insertion by calling `deleteRows(at:with:)` or `insertRows(at:with:)`, as appropriate. Also in editing mode, if a table-view cell has its `showsReorderControl` property set to `true`, the data source receives a `tableView(_:moveRowAt:to:)` message. The data source can selectively remove the reordering control for cells by implementing `tableView(_:canMoveRowAt:)`.

`UITableView` caches table-view cells for visible rows. You can create custom `UITableViewCell` objects with content or behavioral characteristics that are different than the default cells; A Closer Look at Table View Cells explains how.

`UITableView` overrides the `layoutSubviews()` method of `UIView` so that it calls `reloadData()` only when you create a new instance of `UITableView` or when you assign a new data source. Reloading the table view clears current state, including the current selection. However, if you explicitly call `reloadData()`, it clears this state and any subsequent direct or indirect call to `layoutSubviews()` does not trigger a reload.

State Preservation

If you assign a value to a table view's `restorationIdentifier` property, it attempts to preserve the currently selected rows and the first visible row. The table's data source may adopt the `UIDataSourceModelAssociation` protocol, which provides a way to identify a row's contents independent of that row's position in the table. If the table's data source adopts the `UIDataSourceModelAssociation` protocol, the data source will be consulted when saving state to convert the index paths for the top visible row and any selected cells to identifiers. During restoration, the data source will be consulted to convert those identifiers back to index paths and reestablish the top visible row, and reselect the cells. If the table's data source does not implement the `UIDataSourceModelAssociation` protocol, the scroll position will be saved and restored directly, as will the index paths for selected cells.

For more information about how state preservation and restoration works, see App Programming Guide for iOS.

Topics

Initializing a UITableView Object

`init(frame: CGRect, style: UITableView.Style)`

Initializes and returns a table view object having the given frame and style.

`init?(coder: NSCoder)`

Providing the Table View Data

`var dataSource: UITableViewDataSource?`

The object that acts as the data source of the table view.

`protocol UITableViewDataSource`

The `UITableViewDataSource` protocol is adopted by an object that mediates the application’s data model for a `UITableView` object. The data source provides the table-view object with the information it needs to construct and modify a table view.

`protocol UITableViewDataSourcePrefetching`

A protocol that provides advance warning of the data requirements for a table view, allowing the triggers of asynchronous data load operations.

Customizing the Table View Behavior

`var delegate: UITableViewDelegate?`

The object that acts as the delegate of the table view.

`protocol UITableViewDelegate`

The delegate of a `UITableView` object must adopt the `UITableViewDelegate`protocol. Optional methods of the protocol allow the delegate to manage selections, configure section headings and footers, help to delete and reorder cells, and perform other actions.

Configuring a Table View

`var style: UITableView.Style`

Returns the style of the table view.

`func numberOfRows(inSection: Int) -> Int`

Returns the number of rows (table cells) in a specified section.

`var numberOfSections: Int`

The number of sections in the table view.

`var rowHeight: CGFloat`

The height of each row (that is, table cell) in the table view.

`var separatorStyle: UITableViewCell.SeparatorStyle`

The style for table cells used as separators.

`var separatorColor: UIColor?`

The color of separator rows in the table view.

`var separatorEffect: UIVisualEffect?`

The effect applied to table separators.

`var backgroundView: UIView?`

The background view of the table view.

`var separatorInset: UIEdgeInsets`

Specifies the default inset of cell separators.

`var separatorInsetReference: UITableView.SeparatorInsetReference`

An indicator of how the separator inset value should be interpreted.

`enum UITableView.SeparatorInsetReference`

Constants indicating how to interpret the separator inset value of a table view.

`var cellLayoutMarginsFollowReadableWidth: Bool`

A Boolean value that indicates whether the cell margins are derived from the width of the readable content guide.

Creating Table View Cells

`func register(UINib?, forCellReuseIdentifier: String)`

Registers a nib object containing a cell with the table view under a specified identifier.

`func register(AnyClass?, forCellReuseIdentifier: String)`

Registers a class for use in creating new table cells.

`func dequeueReusableCell(withIdentifier: String, for: IndexPath) -> UITableViewCell`

Returns a reusable table-view cell object for the specified reuse identifier and adds it to the table.

`func dequeueReusableCell(withIdentifier: String) -> UITableViewCell?`

Returns a reusable table-view cell object located by its identifier.

Accessing Header and Footer Views

`func register(UINib?, forHeaderFooterViewReuseIdentifier: String)`

Registers a nib object containing a header or footer with the table view under a specified identifier.

`func register(AnyClass?, forHeaderFooterViewReuseIdentifier: String)`

Registers a class for use in creating new table header or footer views.

`func dequeueReusableHeaderFooterView(withIdentifier: String) -> UITableViewHeaderFooterView?`

Returns a reusable header or footer view located by its identifier.

`var tableHeaderView: UIView?`

Returns an accessory view that is displayed above the table.

`var tableFooterView: UIView?`

Returns an accessory view that is displayed below the table.

`var sectionHeaderHeight: CGFloat`

The height of section headers in the table view.

`var sectionFooterHeight: CGFloat`

The height of section footers in the table view.

`func headerView(forSection: Int) -> UITableViewHeaderFooterView?`

Returns the header view associated with the specified section.

`func footerView(forSection: Int) -> UITableViewHeaderFooterView?`

Returns the footer view associated with the specified section.

Accessing Cells and Sections

`func cellForRow(at: IndexPath) -> UITableViewCell?`

Returns the table cell at the specified index path.

`func indexPath(for: UITableViewCell) -> IndexPath?`

Returns an index path representing the row and section of a given table-view cell.

`func indexPathForRow(at: CGPoint) -> IndexPath?`

Returns an index path identifying the row and section at the given point.

`func indexPathsForRows(in: CGRect) -> [IndexPath]?`

An array of index paths each representing a row enclosed by a given rectangle.

`var visibleCells: [UITableViewCell]`

The table cells that are visible in the table view.

`var indexPathsForVisibleRows: [IndexPath]?`

An array of index paths each identifying a visible row in the table view.

Estimating Element Heights

`var estimatedRowHeight: CGFloat`

The estimated height of rows in the table view.

`var estimatedSectionHeaderHeight: CGFloat`

The estimated height of section headers in the table view.

`var estimatedSectionFooterHeight: CGFloat`

The estimated height of section footers in the table view.

Scrolling the Table View

`func scrollToRow(at: IndexPath, at: UITableView.ScrollPosition, animated: Bool)`

Scrolls through the table view until a row identified by index path is at a particular location on the screen.

`func scrollToNearestSelectedRow(at: UITableView.ScrollPosition, animated: Bool)`

Scrolls the table view so that the selected row nearest to a specified position in the table view is at that position.

Managing Selections

`var indexPathForSelectedRow: IndexPath?`

An index path identifying the row and section of the selected row.

`var indexPathsForSelectedRows: [IndexPath]?`

The index paths representing the selected rows.

`func selectRow(at: IndexPath?, animated: Bool, scrollPosition: UITableView.ScrollPosition)`

Selects a row in the table view identified by index path, optionally scrolling the row to a location in the table view.

`func deselectRow(at: IndexPath, animated: Bool)`

Deselects a given row identified by index path, with an option to animate the deselection.

`var allowsSelection: Bool`

A Boolean value that determines whether users can select a row.

`var allowsMultipleSelection: Bool`

A Boolean value that determines whether users can select more than one row outside of editing mode.

`var allowsSelectionDuringEditing: Bool`

A Boolean value that determines whether users can select cells while the table view is in editing mode.

`var allowsMultipleSelectionDuringEditing: Bool`

A Boolean value that controls whether users can select more than one cell simultaneously in editing mode.

Inserting, Deleting, and Moving Rows and Sections

`func insertRows(at: [IndexPath], with: UITableView.RowAnimation)`

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: UITableView.RowAnimation)`

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.

`func insertSections(IndexSet, with: UITableView.RowAnimation)`

Inserts one or more sections in the table view, with an option to animate the insertion.

`func deleteSections(IndexSet, with: UITableView.RowAnimation)`

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.

`func performBatchUpdates((() -> Void)?, completion: ((Bool) -> Void)? = nil)`

Animates multiple insert, delete, reload, and move operations as a group.

`func beginUpdates()`

Begins a series of method calls that insert, delete, or select rows and sections of the table view.

`func endUpdates()`

Concludes a series of method calls that insert, delete, select, or reload rows and sections of the table view.

Managing Drag Interactions

`var dragDelegate: UITableViewDragDelegate?`

The delegate object that manages the dragging of items from the table view.

`protocol UITableViewDragDelegate`

The interface for initiating drags from a table view.

`var hasActiveDrag: Bool`

A Boolean value indicating whether rows were lifted from the table view and have not yet been dropped.

`var dragInteractionEnabled: Bool`

A Boolean value indicating whether the table view supports drags and drops between apps.

Managing Drop Interactions

`var dropDelegate: UITableViewDropDelegate?`

The delegate object that manages the dropping of content into the table view.

`protocol UITableViewDropDelegate`

The interface for handling drops in a table view.

`var hasActiveDrop: Bool`

Managing the Editing of Table Cells

`var isEditing: Bool`

A Boolean value that determines whether the table view is in editing mode.

`func setEditing(Bool, animated: Bool)`

Toggles the table view into and out of editing mode.

Reloading the Table View

`var hasUncommittedUpdates: Bool`

A Boolean value indicating whether the table view contains drop placeholders or is reordering its rows as part of handling a drop.

`func reloadData()`

Reloads the rows and sections of the table view.

`func reloadRows(at: [IndexPath], with: UITableView.RowAnimation)`

Reloads the specified rows using an animation effect.

`func reloadSections(IndexSet, with: UITableView.RowAnimation)`

Reloads the specified sections using a given animation effect.

`func reloadSectionIndexTitles()`

Reloads the items in the index bar along the right side of the table view.

Accessing Drawing Areas of the Table View

`func rect(forSection: Int) -> CGRect`

Returns the drawing area for a specified section of the table view.

`func rectForRow(at: IndexPath) -> CGRect`

Returns the drawing area for a row identified by index path.

`func rectForFooter(inSection: Int) -> CGRect`

Returns the drawing area for the footer of the specified section.

`func rectForHeader(inSection: Int) -> CGRect`

Returns the drawing area for the header of the specified section.

Prefetching Data

If your table view relies on an expensive data loading process, you can improve your user experience by prefetching data before it is needed for display. Assign an object that conforms to the `UITableViewDataSourcePrefetching` protocol to the `prefetchDataSource` property to receive notifications of when to prefetch data for cells.

`var prefetchDataSource: UITableViewDataSourcePrefetching?`
The object that acts as the prefetching data source for the table view, receiving notifications of upcoming cell data requirements.

Configuring the Table Index

`var sectionIndexMinimumDisplayRowCount: Int`
The number of table rows at which to display the index list on the right edge of the table.

`var sectionIndexColor: UIColor?`
The color to use for the table view’s index text.

`var sectionIndexBackgroundColor: UIColor?`
The color to use for the background of the table view’s section index while not being touched.

`var sectionIndexTrackingBackgroundColor: UIColor?`
The color to use for the table view’s index background area.

Managing Focus

`var remembersLastFocusedIndexPath: Bool`
A Boolean value that indicates whether the table view should automatically return the focus to the cell at the last focused index path.

Constants

`enum UITableView.Style`
The style of the table view.

`enum UITableView.ScrollPosition`
The position in the table view (top, middle, bottom) to which a given row is scrolled.

`enum UITableView.RowAnimation`
The type of animation when rows are inserted or deleted.

Section Index Icons
Requests icon to be shown in the section index of a table view.

Default Dimension
The default value for a given dimension.

Notifications

`class let selectionDidChangeNotification: NSNotification.Name`

Posted when the selected row in the posting table view changes.

Instance Properties

`var insetsContentViewsToSafeArea: Bool`

Relationships

Inherits From

`UIScrollView`

Conforms To

`CVarArg`
`Equatable`
`Hashable`
`NSCoding`
`UIAccessibilityIdentification`