

---

# Titanium Mobile: API Reference



**Titanium.UI.TableView Class**  
**Titanium.UI.TableViewRow Class**  
**Titanium.UI.TableViewSection Class**

Copyright © 2010 Appcelerator, Inc. All rights reserved.

Appcelerator, Inc. 444 Castro Street, Suite 818, Mountain View, California 94041

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, mechanical, electronic, photocopying, recording, or otherwise, without prior written permission of Appcelerator, Inc., with the following exceptions: Any person is hereby authorized to store documentation on a single computer for personal use only and to print copies of documentation for personal use provided that the documentation contains Appcelerator's copyright notice.

The Appcelerator name and logo are registered trademarks of Appcelerator, Inc. Appcelerator Titanium is a trademark of Appcelerator, Inc. All other trademarks are the property of their respective owners.

No licenses, express or implied, are granted with respect to any of the technology described in this document. Appcelerator retains all intellectual property rights associated with the technology described in this document.

Every effort has been made to ensure that the information in this document is accurate. Appcelerator is not responsible for typographical or technical errors. Even though Appcelerator has reviewed this document, APPCELERATOR MAKES NO WARRANTY OR REPRESENTATION, EITHER EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT, ITS QUALITY, ACCURACY, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. AS A RESULT, THIS DOCUMENT IS PROVIDED "AS IS," AND YOU, THE READER, ARE ASSUMING THE ENTIRE RISK AS TO ITS QUALITY AND ACCURACY. IN NO EVENT WILL APPCELERATOR BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY DEFECT OR INACCURACY IN THIS DOCUMENT, even if advised of the possibility of such damages. THE WARRANTY AND REMEDIES SET FORTH ABOVE ARE EXCLUSIVE AND IN LIEU OF ALL OTHERS, ORAL OR WRITTEN, EXPRESS OR IMPLIED. No Appcelerator dealer, agent, or employee is authorized to make any modification, extension, or addition to this warranty. Some states do not allow the exclusion or limitation of implied warranties or liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you. This warranty gives you specific legal rights, and you may also have other rights which vary from state to state.

# Contents

<b>Overview .....</b>	<b>4</b>
<b>Titanium.UI.TableView Class .....</b>	<b>5</b>
Instance Methods Summary .....	5
Object Properties .....	5
Events .....	6
Event Properties .....	7
Titanium.UI.TableView Class Instance Methods .....	8
<b>Titanium.UI.TableViewRow Class .....</b>	<b>15</b>
Instance Methods Summary .....	15
Object Properties .....	15
Events .....	16
Event Properties .....	16
Notes .....	16
Titanium.UI.TableViewRow Class Instance Methods .....	16
<b>Titanium.UI.TableViewSection Class .....</b>	<b>20</b>
Instance Methods Summary .....	20
Object Properties .....	20
Events .....	21
Event Properties .....	21
Titanium.UI.TableViewSection Class Instance Methods .....	22
<b>Example Programs .....</b>	<b>25</b>
Very simple example of a TableView .....	25
Example of adding rows to an existing TableView .....	26
Grouping rows in a TableView .....	28
More complex example illustrating class names for TableViewRows .....	30

## Overview

A `TableView` allows you to create a scrollable table of content in a list-based fashion. You create a `TableView` with the method `Titanium.UI.createTableView`. Typically, you'll add a `TableView` to a window or other view where it can be positioned according to the parameters provided when the `TableView` is created. A window can have 0, 1 or more `TableViews` associated with it. There are numerous methods documented here that can manipulate the `TableView` and its contents, typically in response to user-generated events.

# Titanium.UI.TableView Class

## Instance Methods Summary

Name	Description
<a href="#">add</a>	add a child to the view hierarchy
<a href="#">addEventListener</a>	add an event listener for the instance to receive view triggered events
<a href="#">animate</a>	animate the view
<a href="#">appendRow</a>	append a row to the table, optionally with animation
<a href="#">deleteRow</a>	delete an existing row, optionally with animation
<a href="#">deselectRow</a>	programmatically deselect a row
<a href="#">fireEvent</a>	fire a synthesized event to the view's listener
<a href="#">hide</a>	hide the view
<a href="#">insertRowAfter</a>	insert a row before another row, optionally with animation
<a href="#">insertRowBefore</a>	insert a row after another row, optionally with animation
<a href="#">removeEventListener</a>	remove a previously added event listener
<a href="#">scrollToIndex</a>	scroll to a specific row index and ensure that that row is on screen
<a href="#">scrollToTop</a>	scroll the table to a specific top position where 0 is the topmost y position in the table view
<a href="#">selectRow</a>	programmatically select a row
<a href="#">setData</a>	set the data in the table, optionally with animation
<a href="#">show</a>	make the view visible
<a href="#">toImage</a>	return a Blob image of the rendered view
<a href="#">updateRow</a>	update an existing row, optionally with animation

## Object Properties

Name	Type	Description
<code>allowsSelection</code>	boolean	true if the rows can be selected
<code>anchorPoint</code>	object	a dictionary with properties x and y to indicate the anchor point value. anchor specifies the position by which animation should occur. center is 0.5, 0.5
<code>animatedCenterPoint</code>	object	read-only object with x and y properties of where the view is during animation
<code>backgroundColor</code>	string	the background color of the table view
<code>backgroundGradient</code>	object	a background gradient for the view with the properties: type,startPoint,endPoint,startRadius,endRadius,backfillStart,backfillEnd,colors.
<code>backgroundImage</code>	string	the background image to render in the background of the table view
<code>borderColor</code>	string	the border color of the view
<code>borderRadius</code>	float	the border radius of the view
<code>borderWidth</code>	float	the border width of the view
<code>bottom</code>	float,string	property for the view bottom position. this position is relative to the views parent. can be either a float value or a string of the width.
<code>center</code>	object	a dictionary with properties x and y to indicate the center of the views position relative to the parent view
<code>data</code>	array	an array of dictionaries, row objects or section objects; returns an array of sections
<code>editable</code>	boolean	allow the table view to be editable (this must be true for swipe-to-delete)
<code>editing</code>	boolean	boolean to control the editing state of the table view

Name	Type	Description
filterAttribute	string	the filter attribute to be used when searching. this property maps to your data object or a property on the row object
filterCaseInsensitive	boolean	boolean to indicate if the search should be case sensitive or case insensitive (default)
footerTitle	string	the table view footer title
footerView	object	the table view footer as a view that will be rendered instead of a label
headerTitle	string	the table view header title
headerView	object	the table view header as a view that will be rendered instead of a label
height	float,string	property for the view height. can be either float value or a string of the width.
index	array	an array of objects (with title and index properties) to control the table view index
left	float,string	property for the view left position. this position is relative to the views parent. can be either a float value or a string of the width.
maxRowHeight	float	max row height for table view rows
minRowHeight	float	min row height for table view rows
moving	boolean	boolean to control the moveable state of the table view
opacity	float	the opacity from 0.0-1.0
right	float,string	property for the view right position. this position is relative to the views parent. can be either a float value or a string of the width.
rowHeight	float	default row height for table view rows
scrollable	boolean	true (default) if tableview can be scrolled
search	object	the search field to use for the table view
searchHidden	boolean	boolean to control the visibility of the search field
separatorColor	string	the separator color color as a hex or named value
separatorStyle	int	the separator style constant. For iPhone, Titanium.UI.iPhone.TableViewSeparatorStyle
size	object	the size of the view as a dictionary of width and height properties
style	int	iPhone only. the style of the table view. constant from Titanium.UI.iPhone.TableViewStyle
top	float,string	property for the view top position. this position is relative to the views parent. can be either a float value or a string of the width.
touchEnabled	boolean	a boolean indicating if the view should receive touch events (true, default) or forward them to peers (false)
transform	object	the transformation matrix to apply to the view
visible	boolean	a boolean of the visibility of the view
width	float,string	Property for the view width. Can either be `auto`, a float value or a string of the width.
zIndex	int	the z index position relative to other sibling views

## Events

When an event listener is added to an instance of `TableView`, `TableViewRow` or `TableViewSection`, the following triggered events can be received. Event properties provide detail about the specific event. Note that not all properties apply to every event type.

Event Type	When fired
click	a table row is clicked
dblclick	the device detects a double click against the view

Event Type	When fired
delete	a table row is deleted by the user Note: TableView only
doubletap	the device detects a double tap against the view
move	a table row is moved by the user Note: TableView only
scroll	the table view is scrolled (currently, iphone only) Note: TableView only
scrollEnd	the table view stops scrolling (currently, iphone only) Note: TableView only
singletap	the device detects a single tap against the view
swipe	the device detects a swipe (left or right) against the view
touchcancel	a touch event is interrupted by the device (for example, in circumstances such as an incoming call to allow the UI to clean up state)
touchend	a touch event is completed
touchmove	as soon as the device detects movement of a touch; event coordinates are always relative to the view in which the initial touch occurred
touchstart	as soon as the device detects a gesture
twofingertap	the device detects a two-finger tap against the view

## Event Properties

Event	Description	Triggered Event Types
contentOffset	dictionary with x and y properties containing the content offset	scroll, scrollEnd
contentSize	dictionary with width and height properties containing the size of the content (regardless of the display size in the case of scrolling)	scroll, scrollEnd
detail	boolean to indicate if the right area was clicked	click, delete, move
direction	direction of the swipe - either left or right	swipe
globalPoint	a dictionary with properties x and y describing the point of the event in screen coordinates	<i>all</i> except delete, move, scroll, scrollEnd
index	table view row index	click, delete, move
row	table view row object	click, delete, move
rowData	table view row data object	click, delete, move
searchMode	boolean to indicate if the table is in search mode	click, delete, move
section	table view section object	click, delete, move
size	dictionary with width and height properties containing the size of the visible table view	scroll, scrollEnd
source	the source object that fired the event	<i>all</i>
type	the name of the event fired	<i>all</i>
x	the x point of the event, in receiving view coordinates	<i>all</i> except delete, move, scroll, scrollEnd
y	the y point of the event, in receiving view coordinates	<i>all</i> except delete, move, scroll, scrollEnd

## Titanium.UI.TableView Class Instance Methods

---

### add

Add a child to the view hierarchy.

#### Arguments

Name	Type	Description
view	object	the view to add to this views hierarchy

#### Returns

void

---

### addEventListener

Adds an event listener for the instance to receive view triggered events.

#### Arguments

Name	Type	Description
name	string	Name of the event
callback	function	Callback function to invoke when the event is fired

#### Returns

void

#### Example

```
myTableView.addEventListener('click', function(e) {  
    alert('Click at index: '+e.index);  
});
```

---

### animate

Animates the view.

#### Arguments

Name	Type	Description
obj	object	Either a dictionary of animation properties, or an Animation object
callback	function	Callback function to be invoked upon completion of the animation

#### Returns

void

**Example**

```
myTableView.animate({animationStyle:Titanium.UI.iPhone.RowAnimationStyle.UP});
```

---

**appendRow**

Append a row to the table, optionally with animation.

**Arguments**

Name	Type	Description
row	object	row object to append, or dictionary object describing a row
properties	object	animation properties

**Returns**

void

**Example**

```
var myRow = Ti.UI.createTableViewRow();
myRow.title = "Coffee";
myTableView.appendRow(myRow, {});
```

---

**deleteRow**

Deletes an existing row from the table, optionally with animation.

**Arguments**

Name	Type	Description
row	object	row object to delete, or index of the row to delete
properties	object	animation properties

**Returns**

void

**Example**

```
myTableView.deleteRow(2, {animationStyle:Titanium.UI.iPhone.RowAnimationStyle.UP});
```

---

**deselectRow**

Programmatically deselect a row.

## Arguments

Name	Type	Description
row	int	Row index to deselect.

## Returns

void

## Example

```
myTableView.deselectRow(4);
```

---

## fireEvent

Fire a synthesized event to the view's listener.

## Arguments

Name	Type	Description
name	string	Name of the event.
event	object	event object

## Returns

void

## Example

```
myTableView.fireEvent('click', {index:2});
```

---

## hide

Hide the view

## Arguments

This function takes no arguments.

## Returns

void

## Example

```
myTableView.hide();
```

---

## insertRowAfter

Insert a row before another row, optionally with animation.

**Arguments**

Name	Type	Description
index	int	index
row	object	row to insert
properties	object	animation properties

**Returns**

void

**Example**

```
myTableView.insertRowAfter(2, myRow, {});
```

**insertRowBefore**

Insert a row after another row, optionally with animation.

**Arguments**

Name	Type	Description
index	int	index
row	object	row to insert
properties	object	animation properties

**Returns**

void

**Example**

```
myTableView.insertRowBefore(0, myRow, {});
```

**removeEventListener**

Remove a previously added event listener.

**Arguments**

Name	Type	Description
name	string	Name of the event.
callback	function	Callback function passed in addEventListener.

**Returns**

void

---

## scrollToIndex

Scroll to a specific row index and ensure that that row is on screen.

### Arguments

Name	Type	Description
index	int	Index.
properties	object	Animation properties. Position property controls the position constant to use for position (on iPhone, use constants from Titanium.UI.iPhone.TableViewScrollPosition).

### Returns

void

### Example

```
myTableView.scrollToIndex(3, {});
```

---

## scrollToTop

Scroll the table to a specific top position where 0 is the topmost y position in the table view.

### Arguments

Name	Type	Description
top	float	y position
properties	object	Optional dictionary with the key animated (default, true) as boolean to indicate if the scroll should be animated or immediate.

### Returns

void

---

## selectRow

Programmatically select a row.

### Arguments

Name	Type	Description
row	int	Row index to select.

### Returns

void

**Example**

```
myTableView.selectRow(2);
```

---

**setData**

Set the data in the table, optionally with animation.

**Arguments**

Name	Type	Description
data	array	Data array of dictionaries, row objects or section objects.
properties	object	Animation properties.

**Returns**

void

---

**show**

Make the view visible.

**Arguments**

This function takes no arguments.

**Returns**

void

**Example**

```
myTableView.show();
```

---

**toImage**

Return a Blob image of the rendered view.

**Arguments**

Name	Type	Description
f	function	Function to be invoked upon completion. If non-null, this method will be performed asynchronously. if null, it will be performed immediately.

**Returns**

void

**Example**

```
var myImage = myTableView.toImage();
```

---

## updateRow

Update an existing row, optionally with animation.

### Arguments

Name	Type	Description
row	object	row data to update
properties	object	animation properties

### Returns

void

### Example

```
myTableView.updateRow(3, {title:'New title'});
```

# Titanium.UI.TableViewRow Class

## Instance Methods Summary

Name	Description
<a href="#">add</a>	add a child to the view hierarchy
<a href="#">addEventListener</a>	add an event listener for the instance to receive view triggered events
<a href="#">animate</a>	animate the view
<a href="#">fireEvent</a>	fire a synthesized event to the views listener
<a href="#">hide</a>	hide the view
<a href="#">removeEventListener</a>	remove a previously added event listener
<a href="#">show</a>	make the view visible
<a href="#">toImage</a>	return a Blob image of the rendered view

## Object Properties

Name	Type	Description
anchorPoint	object	a dictionary with properties x and y to indicate the anchor point value. anchor specifies the position by which animation should occur. center is 0.5, 0.5
animatedCenterPoint	object	read-only object with x and y properties of where the view is during animation
backgroundColor	string	the background color of the table view
backgroundGradient	object	A background gradient for the view with the properties: type,startPoint,endPoint,startRadius,endRadius,backfillStart,backfillEnd,colors.
backgroundImage	string	the background image to render in the background of the table view
borderColor	string	the border color of the view
borderRadius	float	the border radius of the view
borderWidth	float	the border width of the view
bottom	float,string	property for the view bottom position. this position is relative to the views parent. can be either a float value or a string of the width.
center	object	a dictionary with properties x and y to indicate the center of the views position relative to the parent view
className	string	The class name of the table. Each table view cell must have a unique class name if the cell layout is different. However, use the same name for rows that have the same structural layout (even if the content is different) to provide maximum rendering performance.
color	string	default color of the row when not selected
hasCheck	boolean	render a system provided check mark in the right image area of the row cell
hasChild	boolean	render a system provided right arrow in the right image area of the row cell
hasDetail	boolean	render a system provided blue indicator icon in the right image area of the row cell
height	float	the height of the row. specify auto to calculate the row height based on the size of the child views of the row
indentionLevel	int	the indention level for the cell (defaults to 0)
layout	string	the layout algorithm to use for the layout. either absolute (default) or vertical.
left	float, string	Property for the view left position. This position is relative to the views parent. can be either a float value or a string of the width.
leftImage	string	image url to render in the left image area of the row cell
opacity	float	the opacity from 0.0-1.0

Name	Type	Description
right	float,string	Property for the view right position. This position is relative to the views parent. can be either a float value or a string of the width.
rightImage	string	image url to render in the right image area of the row cell
selectedBackground-Color	string	the background color to render when the row cell is selected
selectedBackgroundImage	string	the background image to render when the row cell is selected
selectedColor	string	color of the row during selection
selectionStyle	int	a selection style constant to control the selection color. For iPhone, use the constants from Titanium.UI.iPhone.TableViewCellStyle
size	object	the size of the view as a dictionary of width and height properties
title	string	The title cell value. Do not specify if using views as children of the row.
top	float,string	Property for the view top position. This position is relative to the views parent. can be either a float value or a string of the width.
touchEnabled	boolean	a boolean indicating if the view should receive touch events (true, default) or forward them to peers (false)
transform	object	the transformation matrix to apply to the view
visible	boolean	a boolean of the visibility of the view
width	float,string	Property for the view width. Can either be `auto`, a float value or a string of the width.
zIndex	int	the z index position relative to other sibling views

## Events

See [Events on page 6](#).

## Event Properties

See [Event Properties on page 7](#).

## Notes

Make sure you set the className on each row instance if using more than one type of row layout. You can use the same value for each instance of a row where the layout is the same — even if the value of the elements inside the row have different values. For example, if the text is the only thing different between two cells but the layout is the same, both row instances should have the same value for className.

You can listen for table row events on all rows by adding an event listener to the table view instance. Events automatically propagate to parent views.

## Titanium.UI.TableViewRow Class Instance Methods

### add

Add a child to the view hierarchy.

### Arguments

Name	Type	Description
view	object	the view to add to this views hierarchy

### Returns

void

---

## addEventListener

Adds an event listener for the instance to receive view triggered events.

### Arguments

Name	Type	Description
name	string	Name of the event
callback	function	Callback function to invoke when the event is fired

### Returns

void

---

## animate

Animates the view.

### Arguments

Name	Type	Description
obj	object	Either a dictionary of animation properties, or an Animation object
callback	function	Callback function to be invoked upon completion of the animation

### Returns

void

---

## fireEvent

Fire a synthesized event to the view's listener.

### Arguments

Name	Type	Description
name	string	Name of the event.
event	object	event object

### Returns

void

---

## hide

Hide the view

### Arguments

This function takes no arguments.

### Returns

void

---

## removeEventListener

Remove a previously added event listener.

### Arguments

Name	Type	Description
name	string	Name of the event.
callback	function	Callback function passed in addEventListener.

### Returns

void

---

## show

Make the view visible.

### Arguments

This function takes no arguments.

### Returns

void

---

## toImage

Return a Blob image of the rendered view.

### Arguments

Name	Type	Description
f	function	Function to be invoked upon completion. If non-null, this method will be performed asynchronously. if null, it will be performed immediately.

### Returns

void

# Titanium.UI.TableViewSection Class

## Instance Methods Summary

Name	Description
<a href="#">add</a>	add a child to the view hierarchy
<a href="#">addEventListener</a>	add an event listener for the instance to receive view triggered events
<a href="#">animate</a>	animate the view
<a href="#">fireEvent</a>	fire a synthesized event to the views listener
<a href="#">hide</a>	hide the view
<a href="#">remove</a>	remove a previously add view from the view hiearchy
<a href="#">removeEventListener</a>	remove a previously added event listener
<a href="#">rowAtIndex</a>	retrieve the row object at a specific index
<a href="#">show</a>	make the view visible
<a href="#">toImage</a>	return a Blob image of the rendered view

## Object Properties

Name	Type	Description
anchorPoint	object	a dictionary with properties x and y to indicate the anchor point value. anchor specifies the position by which animation should occur. center is 0.5, 0.5
animatedCenterPoint	object	read-only object with x and y properties of where the view is during animation
backgroundColor	string	the background color of the table view
backgroundGradient	object	A background gradient for the view with the properties: type,startPoint,endPoint,startRadius,endRadius,backfillStart,backfillEnd,colors.
backgroundImage	string	the background image to render in the background of the table view
borderColor	string	the border color of the view
borderRadius	float	the border radius of the view
borderWidth	float	the border width of the view
bottom	float,string	property for the view bottom position. this position is relative to the views parent. can be either a float value or a string of the width.
center	object	a dictionary with properties x and y to indicate the center of the views position relative to the parent view
footerTitle	string	the title of the section footer
footerView	object	a view to use instead of the default label when rendering the section footer
headerTitle	string	the title of the section header
headerView	object	a view to use instead of the default label when rendering the section header
height	float	the height of the row. specify auto to calculate the row height based on the size of the child views of the row
left	float, string	Property for the view left position. This position is relative to the views parent. can be either a float value or a string of the width.
opacity	float	the opacity from 0.0-1.0
right	float,string	Property for the view right position. This position is relative to the views parent. can be either a float value or a string of the width.
rowCount	int	the (readonly) number of rows in the section
selectedBackground-Color	string	the background color to render when the row cell is selected

Name	Type	Description
size	object	the size of the view as a dictionary of width and height properties
title	string	The title cell value. Do not specify if using views as children of the row.
top	float,string	Property for the view top position. This position is relative to the views parent. can be either a float value or a string of the width.
touchEnabled	boolean	a boolean indicating if the view should receive touch events (true, default) or forward them to peers (false)
transform	object	the transformation matrix to apply to the view
visible	boolean	a boolean of the visibility of the view
width	float,string	Property for the view width. Can either be `auto`, a float value or a string of the width.
zIndex	int	the z index position relative to other sibling views

## Events

See [Events on page 6](#).

## Event Properties

See [Event Properties on page 7](#).

## Titanium.UI.TableViewSection Class Instance Methods

---

### add

Adds a child TableView to the view hierarchy.

#### Arguments

Name	Type	Description
view	TableView	the TableView to add to this view hierarchy

#### Returns

Void

---

### addEventListener

Adds an event listener for the instance to receive view triggered events.

#### Arguments

Name	Type	Description
name	string	Name of the event
callback	function	Callback function to invoke when the event is fired

#### Returns

void

---

### animate

Animates the view.

#### Arguments

Name	Type	Description
obj	object	Either a dictionary of animation properties, or an Animation object
callback	function	Callback function to be invoked upon completion of the animation

#### Returns

void

---

### fireEvent

Fire a synthesized event to the view's listener.

**Arguments**

Name	Type	Description
name	string	Name of the event.
event	object	event object

**Returns**

void

**hide**

Hide the view

**Arguments**

This function takes no arguments.

**Returns**

void

**remove**

Remove a previously add view from the view hiearchy.

**Arguments**

Name	Type	Description
view	object	The view to remove from this views hiearchy.

**Returns**

Void

**removeEventListener**

Remove a previously added event listener.

**Arguments**

Name	Type	Description
name	string	Name of the event.
callback	function	Callback function passed in addEventListener.

**Returns**

void

---

## rowAtIndex

Retrieve the row object at a specific index.

### Arguments

This function takes no arguments.

### Returns

void

---

## show

Make the view visible.

### Arguments

This function takes no arguments.

### Returns

void

---

## toImage

Return a Blob image of the rendered view.

### Arguments

Name	Type	Description
f	function	Function to be invoked upon completion. If non-null, this method will be performed asynchronously. if null, it will be performed immediately.

### Returns

void

## Example Programs

The following example programs illustrate TableViews. Each of these programs replaces your app.js file in your Resources folder. You can make changes to these programs to experiment with the TableView, and the related classes and methods.

- [Very simple example of a TableView on page 25](#)
- [Example of adding rows to an existing TableView on page 26](#)
- [Grouping rows in a TableView on page 28](#)
- [More complex example illustrating class names for TableViewRows on page 30](#)

### Very simple example of a TableView

The following code illustrates the most basic use of a TableView, including an event listener. When you click on a row in the table view, the program displays an alert indicating which row was clicked.

This program works on both iPhone and Android.

```
// app.js

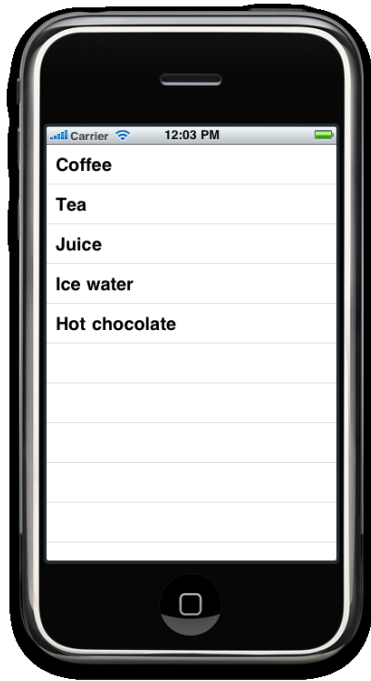
// Create a window
var myWindow = Titanium.UI.createWindow({fullscreen:false});

// Create the table view -- it's a collection of various beverages
var myData = [{title:'Coffee'}, {title:'Tea'}, {title:'Juice'}, {title:'Ice water'}];
var myTableView = Titanium.UI.createTableView({data:myData});

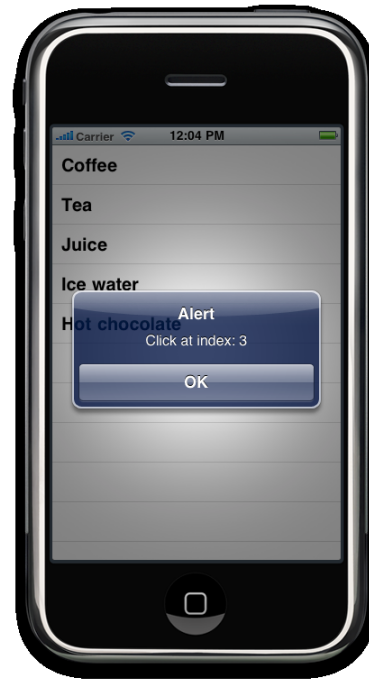
// Create one more row, and append it to the TableView
var myRow = Titanium.UI.createTableViewRow();
myRow.title = "Hot chocolate";
myTableView.appendRow(myRow);

// Add an event listener to respond to clicks in the TableView
myTableView.addEventListener('click', function(e) {
    // Report which item was clicked
    Titanium.UI.createAlertDialog({
        title:'Alert',
        message:('Click at index: '+e.index)
    }).show();
});

// Add the table to the window
myWindow.add(myTableView);
myWindow.open();
```



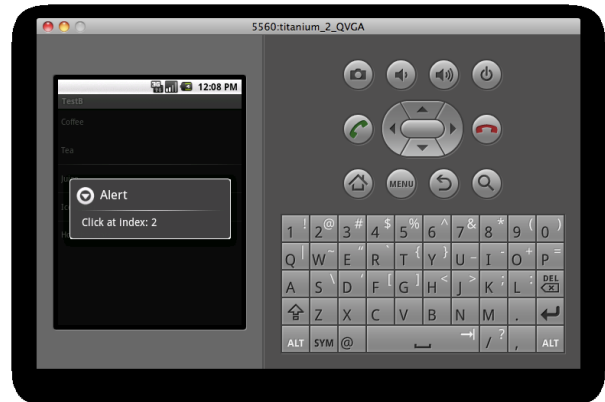
Basic TableView example, for iPhone



Click received in the TableView, for iPhone



Basic TableView example, for Android



Click received in the TableView, for Android

### Example of adding rows to an existing TableView

This example shows how you might programmatically add more rows to an existing TableView. In this case, we're doing this in the event listener for the TableView. When you click on a row, the program adds another row to the TableView. As you click several times, you'll see the TableView grow, and you can scroll it vertically.

This program works on both iPhone and Android.

```
// app.js
```

```
// Create a window
var myWindow = Titanium.UI.createWindow({fullscreen:false});

// Create the table view -- it's a collection of various beverages
var myData = [{title:'Coffee'}, {title:'Tea'}, {title:'Juice'}, {title:'Ice water'}];
var myTableView = Titanium.UI.createTableView({data:myData});

// Create one more row, and append it to the TableView
var myRow = Titanium.UI.createTableViewRow();
myRow.title = "Hot chocolate";
myTableView.appendRow(myRow, {animationStyle:Titanium.UI.iPhone.RowAnimationStyle.UP});

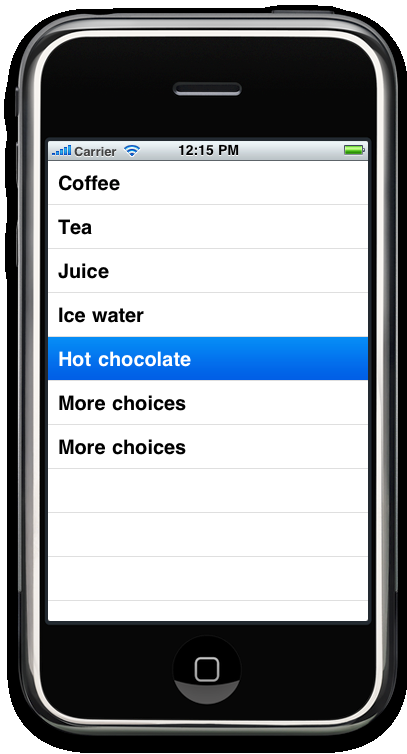
// Add an event listener to respond to clicks in the TableView
myTableView.addEventListener('click', function(e) {
    // Report which item was clicked
    Titanium.UI.createAlertDialog({
        title:'Alert',
        message:'Click at index: '+e.index
    }).show();

    // Show highlighting for the row that was clicked
    if (Titanium.Platform.name != 'android') {
        myTableView.selectRow(e.index);
    }

    // For example, append an additional row to the table each time
    myTableView.appendRow(
        Titanium.UI.createTableViewRow({title:'More choices'}),
        {animationStyle:Titanium.UI.iPhone.RowAnimationStyle.LEFT});
});

// Add the table to the window
myWindow.add(myTableView);
myWindow.open();

// Fire a synthesized event, which will be picked up by the event handler
myTableView.fireEvent('click', {index:2});
```



Two "More choices" rows added, for iPhone



Two "More choices" rows added, for Android

## Grouping rows in a TableView

There are several built-in options for drawing TableView rows as groups, or in sections as groups. This example shows how to do this. One line is commented out -- you can exchange that line for the line next to it to see how iPhone can draw the TableView in an alternate style.

This program works on both iPhone and Android.

```
// app.js

// Create a window
var myWindow = Titanium.UI.createWindow({fullscreen:false});

// Create the first table view section, a collection of hot beverages
var hotTableViewSection = Titanium.UI.createTableViewSection({
  headerTitle:'Hot beverages'});
hotTableViewSection.add(Titanium.UI.createTableViewRow({title:'Coffee'}));
hotTableViewSection.add(Titanium.UI.createTableViewRow({title:'Tea'}));
hotTableViewSection.add(Titanium.UI.createTableViewRow({title:'Hot chocolate'}));

// Create the second table view section, a collection of cold beverages
```

```
var coldTableViewSection = Titanium.UI.createTableViewSection({
    headerTitle:'Cold beverages'});
coldTableViewSection.add(Titanium.UI.createTableViewRow({title:'Chocolate milk'}));
coldTableViewSection.add(Titanium.UI.createTableViewRow({title:'Ice water'}));
coldTableViewSection.add(Titanium.UI.createTableViewRow({title:'Juice'}));

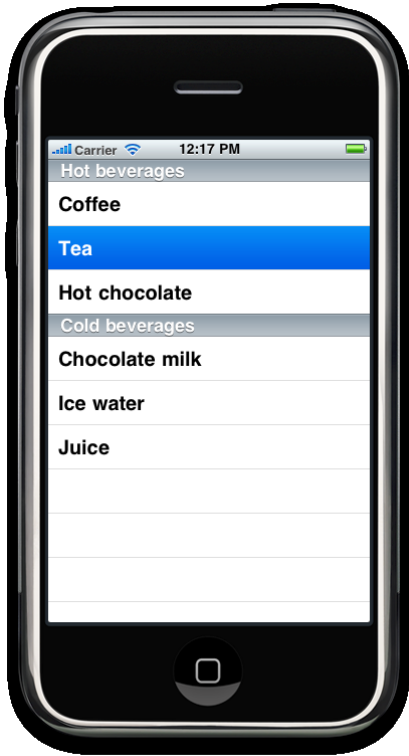
// Create a TableView; we'll add the row data later
// 2 different grouping styles. Comment out one of these 2 calls to createTableView
//var myTableView = Titanium.UI.createTableView({
//    style:Titanium.UI.iPhone.TableViewStyle.GROUPED});
var myTableView = Titanium.UI.createTableView({});

// Add the hot and cold sections to the TableView
myTableView.setData([hotTableViewSection, coldTableViewSection]);

// Add an event listener to respond to clicks in the TableView
myTableView.addEventListener('click', function(e) {
    // Report which item was clicked
    Titanium.UI.createAlertDialog({
        title:'Alert',
        message:'Click at index: '+e.index
    }).show();

    // Show highlighting for the row that was clicked
    if (Titanium.Platform.name != 'android') {
        myTableView.selectRow(e.index);
    }
});

// Add the table to the window
myWindow.add(myTableView);
myWindow.open();
```



TableView rows in 2 groups, for iPhone



TableView rows in 2 groups, for Android

## More complex example illustrating class names for TableViewRows

This much longer example for iPhone illustrates an important point about using TableView rows. The program is a rudimentary contact manager, with TableViewRows that display names from a database. Some of the names have pictures associated with them, and some don't. The program assumes that you have a small picture called "pic.jpg" in your Resources folder, along with app.js.

There is a `className` field associated with each TableViewRow that helps system efficiently allocate TableViewRow objects. Since freed objects can be reused, it's an important optimization to reuse row objects for the same types of data each time. The `className` for a TableViewRow can be used to mark rows that are similar, or dissimilar. In this example, each TableViewRow uses the `className` to distinguish these.

For an experiment, you can change the `noPic` classname to `Pic` — this will have the effect of using the "wrong" type of TableViewRow object for some of the rows. You'll see warning messages appear in Titanium Developer's console when you run the program.

This program also illustrates Buttons and more event listeners, and other techniques for managing data in a TableView. Logic that is common to more than one event listener is pulled out into two JavaScript functions.

The beginning part of the program creates a database and populates it with fake data, so we'll have something to do queries on for the main body of the program. This is of course only an example -- it would be pretty unlikely for a real program to hard-code a database like this. Except for this, the rest of the program is trying to be more realistic.

```

// app.js
// Simple demonstration of a contact viewer, getting contact info from a database

// Create a window
var myWindow = Titanium.UI.createWindow({fullscreen:false});

// Get an image from a file (everybody will have the same image)
var imageFile = Titanium.Filesystem.getFile('pic.jpg');
var pic = imageFile.read();

// Create a simple database of contacts. This is hard-coded because this demonstration
// is more about retrieving the data from the database, rather than creating contact info.
// First, open the database table. Each row in the table looks like this:
// id, numerical index for each row
// n, the friend's name
// az, the sort page for this friend (a=1, b=2, etc.)
// h, whether the friend has an associated pic (1=true)
// p, the name of the file where the pic is stored
var db = Titanium.Database.open('pals');
db.execute(
  'CREATE TABLE IF NOT EXISTS friends (id INTEGER, n STRING, az INTEGER, h INTEGER, p STRING)');

// Starting with an empty table, populate the table with contact information. Notice that some
// contacts have pictures associated with them, and some don't.
db.execute('DELETE FROM friends');
db.execute('INSERT INTO friends (id,n,az,h,p) VALUES(?,?,?,?,?)', 1, 'Aaron', 1, 1, 'pic.jpg');
db.execute('INSERT INTO friends (id,n,az,h) VALUES(?,?,?,?,?)', 2, 'Alice', 1, 0, );
db.execute('INSERT INTO friends (id,n,az,h,p) VALUES(?,?,?,?,?)', 3, 'Annie', 1, 1, 'pic.jpg');
db.execute('INSERT INTO friends (id,n,az,h,p) VALUES(?,?,?,?,?)', 4, 'Arnold',1, 1, 'pic.jpg');
db.execute('INSERT INTO friends (id,n,az,h) VALUES(?,?,?,?,?)', 5, 'Barbie',2, 0, );
db.execute('INSERT INTO friends (id,n,az,h) VALUES(?,?,?,?,?)', 6, 'Billy', 2, 0, );
db.execute('INSERT INTO friends (id,n,az,h,p) VALUES(?,?,?,?,?)', 7, 'Bobbie',2, 1, 'pic.jpg');
db.execute('INSERT INTO friends (id,n,az,h) VALUES(?,?,?,?,?)', 8, 'Bruno', 2, 0, );
db.execute('INSERT INTO friends (id,n,az,h,p) VALUES(?,?,?,?,?)', 9, 'Carlos',3, 1, 'pic.jpg');
db.execute('INSERT INTO friends (id,n,az,h) VALUES(?,?,?,?,?)', 10, 'Cindy', 3, 0, );
db.execute('INSERT INTO friends (id,n,az,h) VALUES(?,?,?,?,?)', 11, 'Claude',3, 0, );
db.execute('INSERT INTO friends (id,n,az,h,p) VALUES(?,?,?,?,?)',12, 'Connie',3, 1, 'pic.jpg');
db.execute('INSERT INTO friends (id,n,az,h,p) VALUES(?,?,?,?,?)',13, 'Denise',4, 1, 'pic.jpg');
db.execute('INSERT INTO friends (id,n,az,h,p) VALUES(?,?,?,?,?)',14, 'Devon', 4, 1, 'pic.jpg');
db.execute('INSERT INTO friends (id,n,az,h) VALUES(?,?,?,?,?)', 15, 'Donald',4, 0, );
db.execute('INSERT INTO friends (id,n,az,h,p) VALUES(?,?,?,?,?)',16, 'Doris', 4, 1, 'pic.jpg');

// Finished with the database for now
db.close();

// Create a TableView for the contact viewer, and add it to the window
var myTableView = Titanium.UI.createTableView({
  top:44 // Adjust downward to accommodate height of toolbar
});
myWindow.add(myTableView);
myWindow.open();

// Create a toolbar which will have next and back buttons, and
// some flexible space between them and the toolbar's title
var label = Titanium.UI.createButton({
  title:'Select Page',
  color:'#fff',// white text
  style:Titanium.UI.iPhone.SystemButtonStyle.PLAIN

```

```
});

var flexSpace = Titanium.UI.createButton({
    systemButton:Titanium.UI.iPhone.SystemButton.FLEXIBLE_SPACE
});

var next = Titanium.UI.createButton({
    title:'Next',
    style:Titanium.UI.iPhone.SystemButtonStyle.BORDERED
});

var back = Titanium.UI.createButton({
    title:'Back',
    style:Titanium.UI.iPhone.SystemButtonStyle.BORDERED
});

// The currentPage and newPage variables determine which contacts are displayed.
// In this simple example, there are 4 pages, with contacts starting with
// A, B, C, and D respectively. The A's are page 1, B's are on page 2, etc.
var currentPage = 0;// Current page displayed
var newPage = 0;// Next page to be displayed

// This is the logic to create rows for this page of contacts,
// and append them to the TableView. We call this from the event listeners
// for the back and next buttons to redraw the rows.
function appendrows() {
    // Get one page of contacts from the database (The A's, B's, C's or D's.)
    db = Titanium.Database.open('pals');
    var selectedFriends = db.execute('SELECT * FROM friends WHERE az =?', currentPage);

    // Iterate across all the contacts retrieved from the database,
    // creating a TableViewRow for each.
    while (selectedFriends.isValidRow()) {
        // You can retrieve a field in a row either by name or column position
        var friendName = selectedFriends.fieldByName('n');
        var picExists = selectedFriends.field(3);

        // Some rows will have pics, and others will not. We use the className to
        // mark them, which will enable Titanium to manage the row data structures,
        // so that reused row objects will always match the type of data that we
        // plan to store in them.
        if (picExists == 1) {
            // Get the image from the filename stored in the database
            var imageFile =
                Titanium.Filesystem.getFile(selectedFriends.field(4));
            var pic = imageFile.read();
            var myRow = Titanium.UI.createTableViewRow({
                title:friendName,
                className:'Pic'
            });
            var myPicView = Titanium.UI.createImageView({
                image:pic,
                width:40,
                height:40,
                left:250
            });
            // Add the pic to the row
            myRow.add(myPicView);
        }
    }
}
```

```
    else {
        myRow = Titanium.UI.createTableViewRow({
            title:friendName,
            className:'noPic'
        });
    }

    // Now append the row we just made to the TableView.
    // Depending on the logic above, this row might have a pic, or not
    myTableView.appendRow(myRow);

    // Iterate to the next selection from the database
    selectedFriends.next();
}

// Close the result set, and the database
selectedFriends.close();
db.close();
};

// Before appending replacement rows, we delete the rows that we're erasing.
// Notice that everything's hard-coded; we always have 4 rows per page.
// The rows are removed from the bottom, moving upwards. This is also
// called from the event listeners for the back and next buttons.
function deleterows() {
    if (newPage != currentPage) {
        if (currentPage != 0) { // Peel away old rows
            for (i=3;i>=0;i--) { // 4 rows, from 0 to 3
                myTableView.deleteRow(i);
            }
        }
    }
}

// These are the event listeners for the next and back buttons.
// These get called when the user clicks on one of the buttons, which
// will cause us to redraw the screen for a new page of contacts. Note
// the logic to suppress redrawing if you're at the beginning or end.
next.addEventListener('click', function(e) {
    if (currentPage < 4) {
        newPage = currentPage + 1;
    }
    if (newPage != currentPage) {
        deleterows();
        currentPage = newPage;
        appendrows();
    }
});

back.addEventListener('click', function(e) {
    if (currentPage > 1) {
        newPage = currentPage - 1;
    }
    if (newPage != currentPage) {
        deleterows();
        currentPage = newPage;
        appendrows();
    }
});
```

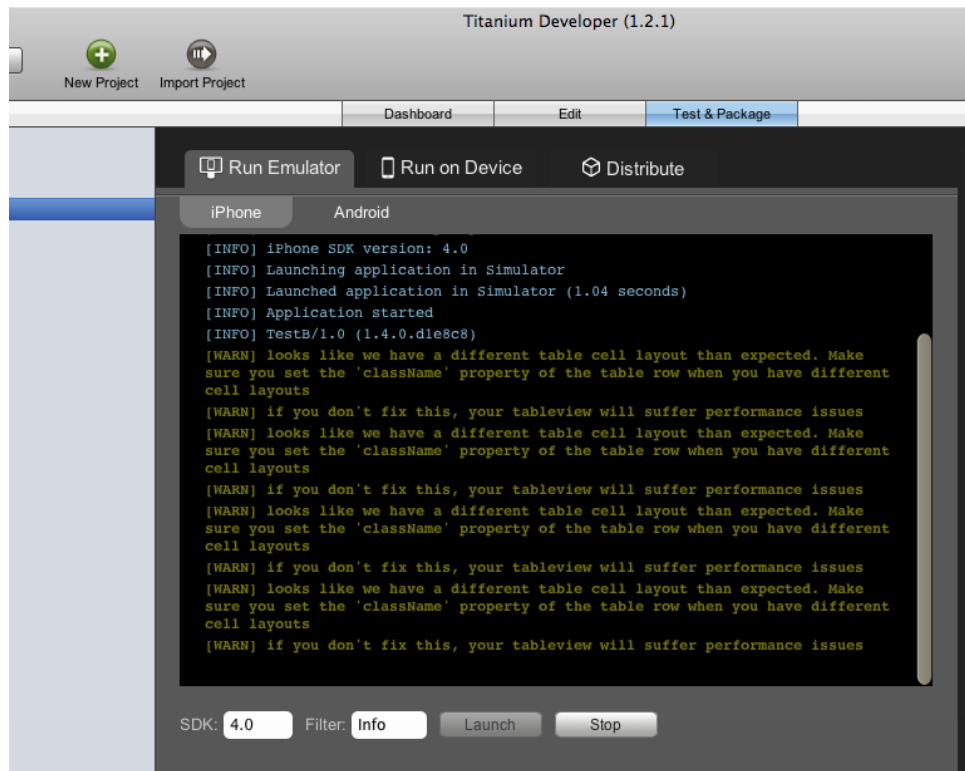
```
// Create the toolbar, and add it to the window
var toolbar = Titanium.UI.createToolbar({
    items:[back,flexSpace,label,flexSpace,next],
    top:0,
    borderTop:false,
    borderBottom:true
});
myWindow.add(toolbar);

// Add an event listener to the TableView to select a row when
// you click on it.
myTableView.addEventListener('click', function(e) {
    // Show that choice as selected
    if (Titanium.Platform.name != 'android') {
        myTableView.selectRow(e.index);
    }
});

// Initialize the display by synthesizing a click on the next
// button. This will trigger the next button's event listener,
// to call the logic that draws the first page of contacts.
next.fireEvent('click',{});
```



Simple contact manager for iPhone -- 3 rows have a picture, one does not



Error messages in Titanium Developer console with noPic modification

## Revision History

8/30/2010	Initial Release
10/05/2010	Revised and expanded code samples
10/12/2010	Corrected datatype information for the data properties