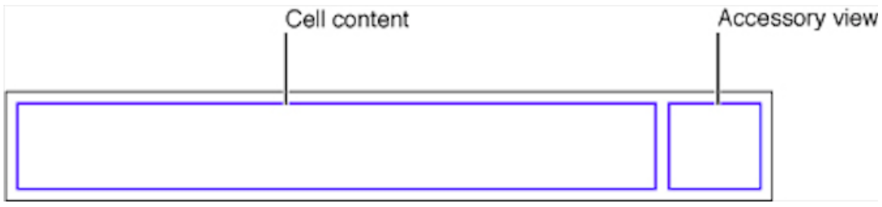
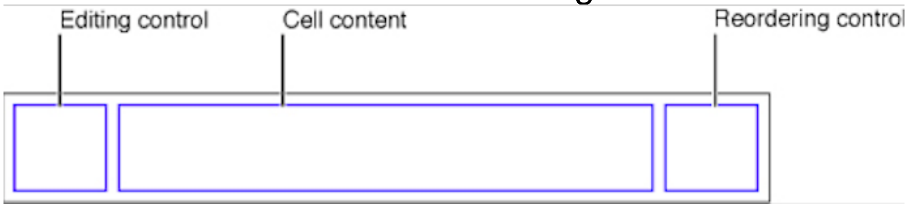
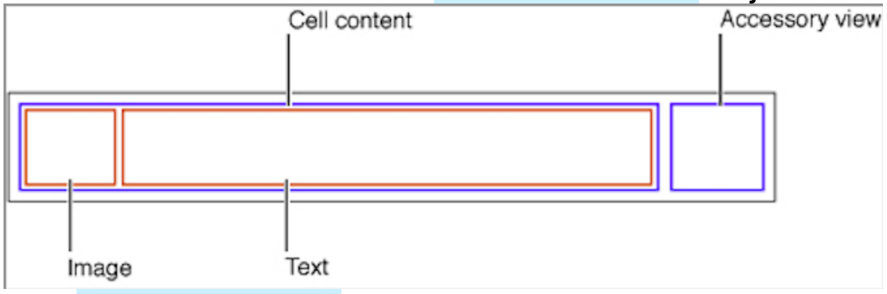
# A Closer Look at Table View Cells

Characteristics of Cell Objects
1. Normally, most of a cell object is reserved for its content: text, image, or any other kind of distinctive identifier.

2. Parts of a table view cell


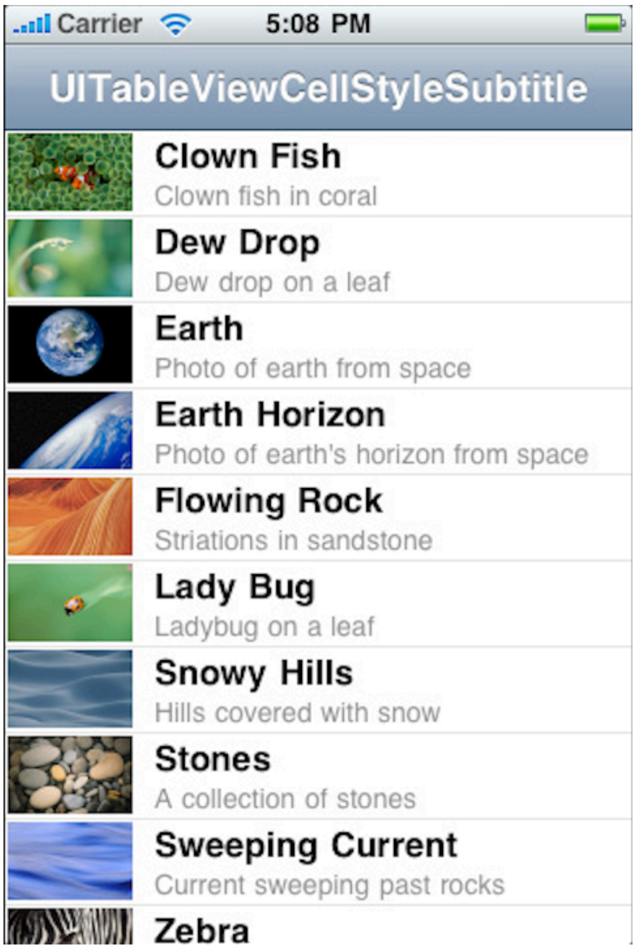
3. Parts of a table view cell in editing mode


4. If a cell object is reusable, you assign it a reuse identifier in the storyboard. At runtime, the table view store cell objects in an internal queue.

Using Cell Objects in Predefined Styles
1. Default cell content in a UITableViewCell object.


The UITableViewCell class defines three properties for this cell content:
=> textLabel—A label for the title (a UILabel object)
=> detailTextLabel—A label for the subtitle if there is additional detail (a UILabel object)
=> imageView—An image view for an image (a UIImageView object)

2. A table view with rows showing both images and text



3. Configuring a UITableViewCell object with both image and text

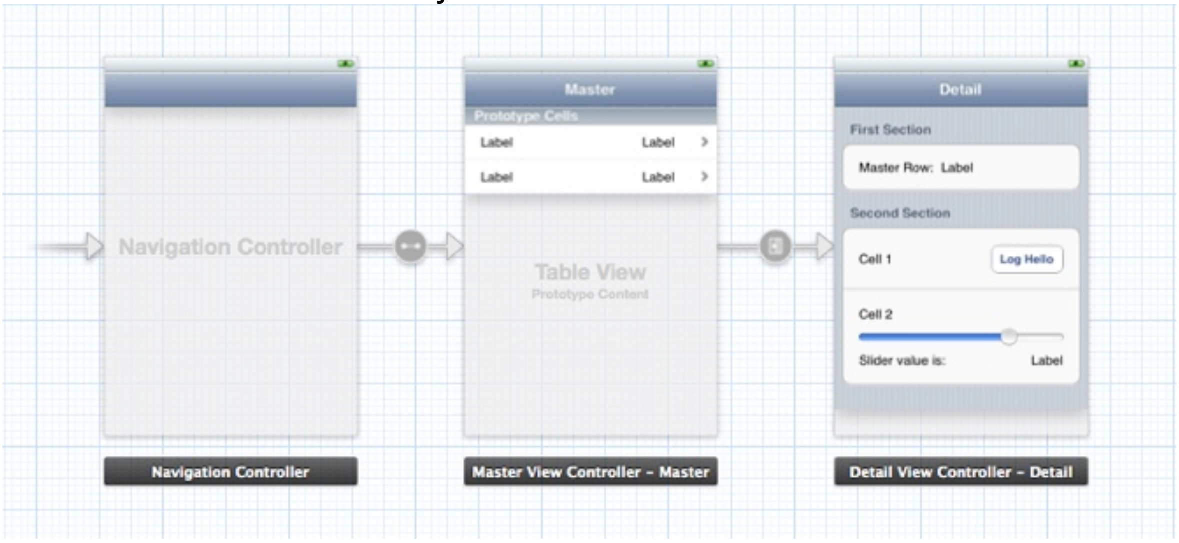
```
- (UITableViewCell *)tableView:(UITableView *)tableView
cellForRowAtIndexPath:(NSIndexPath *)indexPath {

    UITableViewCell *cell = [tableView
dequeueReusableCellWithIdentifier:@"MyIdentifier"];
    if (cell == nil) {
        cell = [[UITableViewCell alloc]
initWithStyle:UITableViewCellStyleSubtitle
reuseIdentifier:@"MyIdentifier"];
        cell.selectionStyle = UITableViewCellSelectionStyleNone;
    }
    NSDictionary *item = (NSDictionary *)[self.content
objectAtIndex:indexPath.row];
    cell.textLabel.text = [item objectForKey:@"mainTitleKey"];
    cell.detailTextLabel.text = [item objectForKey:@"secondaryTitleKey"];
    NSString *path = [[NSBundle mainBundle] pathForResource:[item
objectForKey:@"imageKey"] ofType:@"png"];
    UIImage *theImage = [UIImage imageWithContentsOfFile:path];
    cell.imageView.image = theImage;
    return cell;
}
```

4. When you configure a UITableViewCell object, you can also set various other properties, including (but not limited to) the following:
=> selectionStyle
=> accessoryType and accessoryView
=> editingAccessoryType and editingAccessoryView
=> showsReorderControl
=> backgroundView and selectedBackgroundView
=> indentationLevel and indentationWidth

Customizing Cells
1. Two ways to customize cells:
=> Add subviews to a cell's content view.
=> Create a custom subclass of UITableViewCell.

> Loading Table View Cells from a Storyboard
1. Table view cells in a storyboard



>> The Technique for Dynamic Row Content
1. The data source can use two different ways to access the subviews of the cells:
=> Use the tag property.
=> Use outlets.

2. Adding data to a cell using tags

```
- (UITableViewCell *)tableView:(UITableView *)tableView
```

```
cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
    UITableViewCell *cell = [tableView
dequeueReusableCellWithIdentifier:@"MyIdentifier"];

    UILabel *label;

    label = (UILabel *)[cell viewWithTag:1];
    label.text = [NSString stringWithFormat:@"%d", indexPath.row];

    label = (UILabel *)[cell viewWithTag:2];
    label.text = [NSString stringWithFormat:@"%d",
NUMBER_OF_ROWS - indexPath.row];

    return cell;
}
```

3. Adding data to a cell using outlets

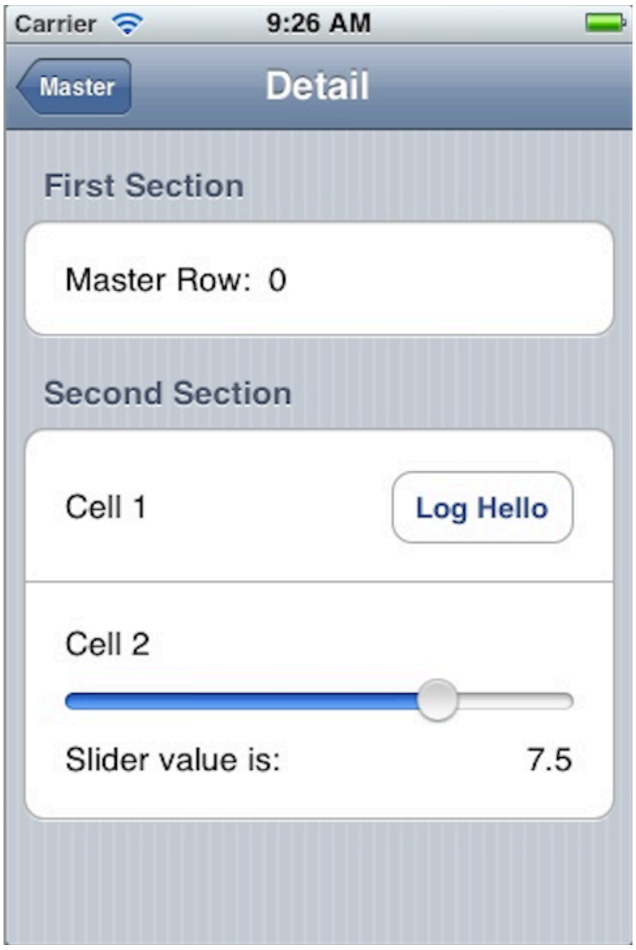
```
- (UITableViewCell *)tableView:(UITableView *)tableView
cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
    MyTableViewCell *cell = [tableView
dequeueReusableCellWithIdentifier:@"MyIdentifier"];

    cell.firstLabel.text = [NSString stringWithFormat:@"%d",
indexPath.row];
    cell.secondLabel.text = [NSString stringWithFormat:@"%d",
NUMBER_OF_ROWS - indexPath.row];

    return cell;
}
```

>> The Technique for Static Row Content

1. Table view rows drawn with multiple cells



> Programmatically Adding Subviews to a Cell’s Content View

1. Adding subviews to a cell’s content view

```
#define MAINLABEL_TAG 1
#define SECONDLABEL_TAG 2
#define PHOTO_TAG 3

- (UITableViewCell *)tableView:(UITableView *)tableView
cellForRowAtIndexPath:(NSIndexPath *)indexPath {

    static NSString *CellIdentifier = @"ImageOnRightCell";

    UILabel *mainLabel, *secondLabel;
    UIImageView *photo;
    UITableViewCell *cell = [tableView
dequeueReusableCellWithIdentifier:CellIdentifier];
    if (cell == nil) {
        cell = [[UITableViewCell alloc]
initWithStyle:UITableViewCellStyleDefault reuseIdentifier:CellIdentifier];
        cell.accessoryType =
UITableViewCellAccessoryDetailDisclosureButton;

        mainLabel = [[[UILabel alloc] initWithFrame:CGRectMake(0.0, 0.0, 
220.0, 15.0)]];
        mainLabel.tag = MAINLABEL_TAG;
        mainLabel.font = [UIFont systemFontOfSize:14.0];
        mainLabel.textAlignment = UITextAlignmentRight;
        mainLabel.textColor = [UIColor blackColor];
        mainLabel.autoresizingMask = 
UIViewAutoresizingFlexibleLeftMargin |
UIViewAutoresizingFlexibleHeight;
        [cell.contentView addSubview:mainLabel];

        secondLabel = [[[UILabel alloc] initWithFrame:CGRectMake(0.0, 
20.0, 220.0, 25.0)]];
        secondLabel.tag = SECONDLABEL_TAG;
        secondLabel.font = [UIFont systemFontOfSize:12.0];
        secondLabel.textAlignment = UITextAlignmentRight;
        secondLabel.textColor = [UIColor darkGrayColor];
        secondLabel.autoresizingMask = 
UIViewAutoresizingFlexibleLeftMargin |
UIViewAutoresizingFlexibleHeight;
        [cell.contentView addSubview:secondLabel];

        photo = [[[UIImageView alloc] initWithFrame:CGRectMake(225.0, 
0.0, 80.0, 45.0)]];
        photo.tag = PHOTO_TAG;
        photo.autoresizingMask = UIViewAutoresizingFlexibleLeftMargin |
UIViewAutoresizingFlexibleHeight;
        [cell.contentView addSubview:photo];
    } else {
        mainLabel = (UILabel *)[cell.contentView 
viewWithTag:MAINLABEL_TAG];
        secondLabel = (UILabel *)[cell.contentView 
viewWithTag:SECONDLABEL_TAG];
        photo = (UIImageView *)[cell.contentView 
viewWithTag:PHOTO_TAG];
    }
    NSDictionary *aDict = [self.list objectAtIndex:indexPath.row];
    mainLabel.text = [aDict objectForKey:@"mainTitleKey"];
    secondLabel.text = [aDict objectForKey:@"secondaryTitleKey"];
    NSString *imagePath = [[NSBundle mainBundle] pathForResource:
[aDict objectForKey:@"imageKey"] ofType:@"png"];
    UIImage *theImage = [UIImage imageWithContentsOfFile:imagePath];
    photo.image = theImage;

    return cell;
}
```

Enhancing the Accessibility of Table View Cells
1. Concatenating labels of a table cell

```
@implementation WeatherTableViewController
// This is a view that provides weather information. It contains a city
subview and a temperature subview, each of which provides a separate
label.
- (UITableViewCell *)tableView:(UITableView *)tableView
cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
    UITableViewCell *cell = [tableView
dequeueReusableCellWithIdentifier:@"Cell" forIndexPath:indexPath];

    // set up the cell here...

    NSString *cityLabel = [self.weatherCity accessibilityLabel];
    NSString *temperatureLabel = [self.weatherTemp accessibilityLabel];

    // Combine the city and temperature information so that VoiceOver
users can get the weather information with one gesture.
    [cell setAccessibilityLabel:[NSString stringWithFormat:@"%@, %@",
cityLabel, temperatureLabel]];
    return cell;
}
@end
```

Cells and Table View Performance
Ensure that your application does the following three things:
=> Reuse cells. Object allocation has a performance cost, especially if the allocation has to happen repeatedly over a short period—say, when the user scrolls a table view. If you reuse cells instead of allocating new ones, you greatly enhance table view performance.
=> Avoid relayout of content. When reusing cells with custom subviews, refrain from laying out those subviews each time the table view requests

a cell. Lay out the subviews once, when the cell is created.
=> Use opaque subviews. When customizing table view cells, make the subviews of the cell opaque, not transparent.