Introduction to iPhone Programming

using Dashcode and Xcode

About the Author

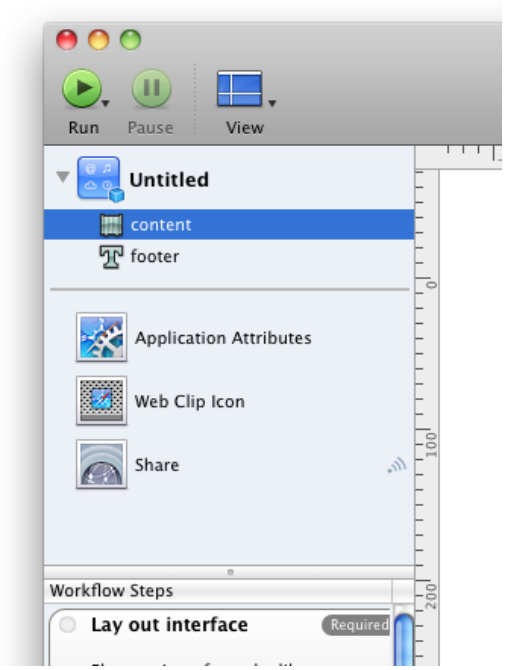
Wei-Meng Lee (**weimenglee@learn2develop.net**) is a technologist and founder of Developer Learning Solutions (http://www.learn2develop.net), a technology company specializing in hands-on training on the latest Microsoft and Mac OS X technologies. He is also an established author with Wrox and O'Reilly.

Wei-Meng first started the iPhone programming course in Singapore and it has since received many positive feedbacks. His hands-on approach to iPhone programming makes understanding the subject much easier than reading books, tutorials, and documentations from Apple. Wei-Meng is currently working with Wrox on "*Beginning iPhone SDK Programming with Objective-C"*, due to be published in Jan 2010.

Lab 1 – Getting Started with Dashcode

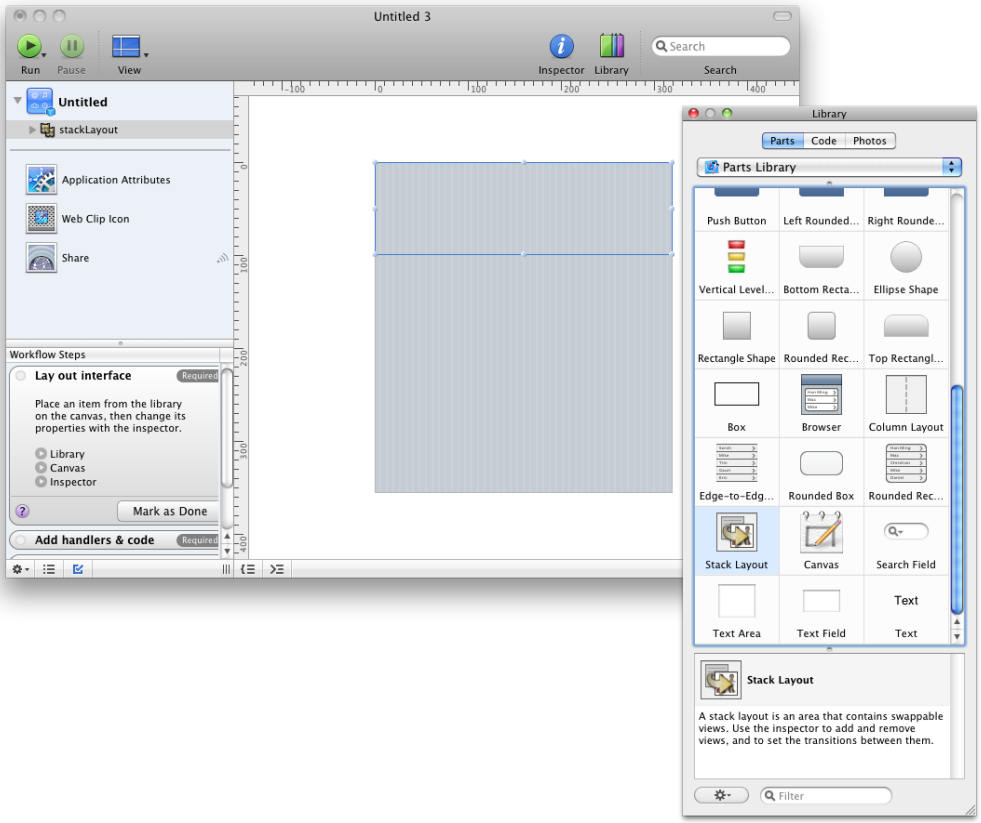
In this lab, you will learn how to get started with developing Web applications for the iPhone. Using the Dashcode tool (part of the iPhone SDK), you will create a currency convertor application that makes use of a client side database to store currency rates.

1. Launch Dashcode and create a new **Custom** project. Notice that by default, Dashcode has created a **content** and a **footer** parts for you. Parts are the various views that you see on your web applications, such as buttons, text, etc. For this lab, you will create a simple currency convertor web application for the iPhone.

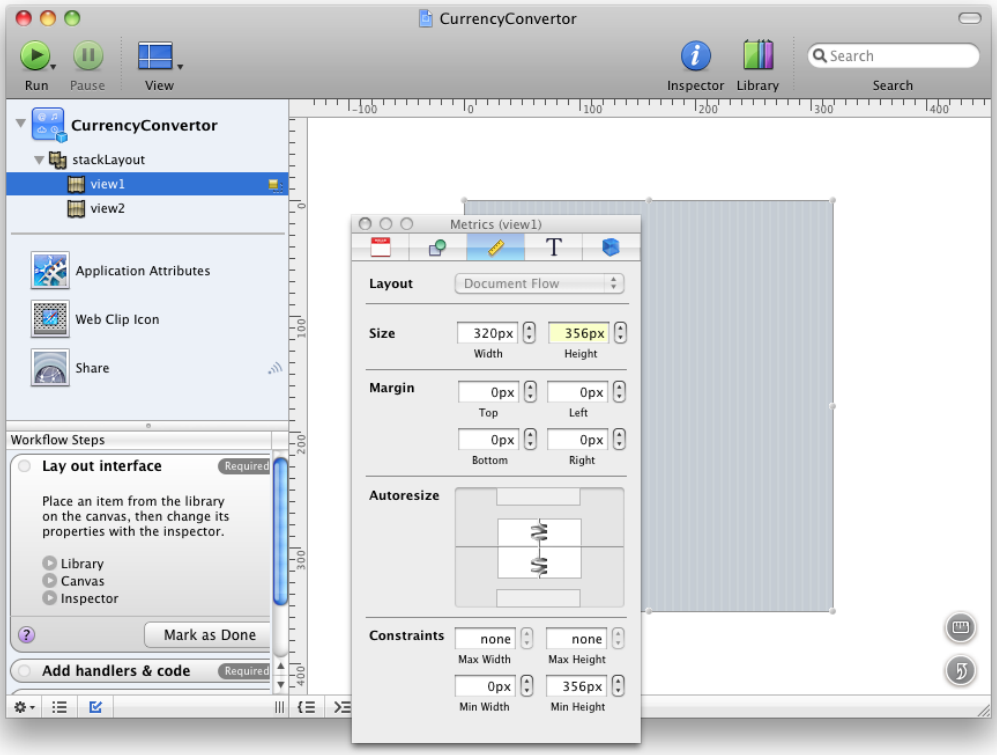


2. Select each of these parts and press the delete key. You shall delete these two parts and add your own parts manually.

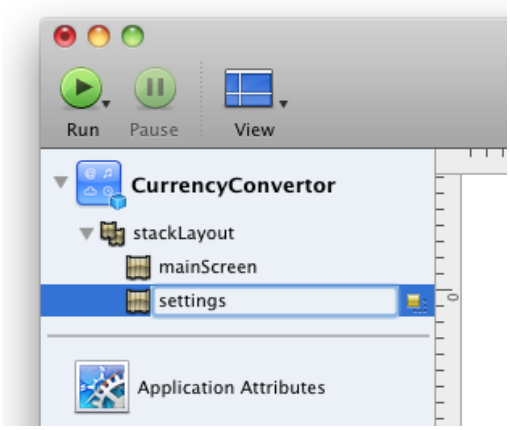
3. Using the **Library** (**Window→Show Library**), drag-and-drop a **Stack Layout** part to the design surface:



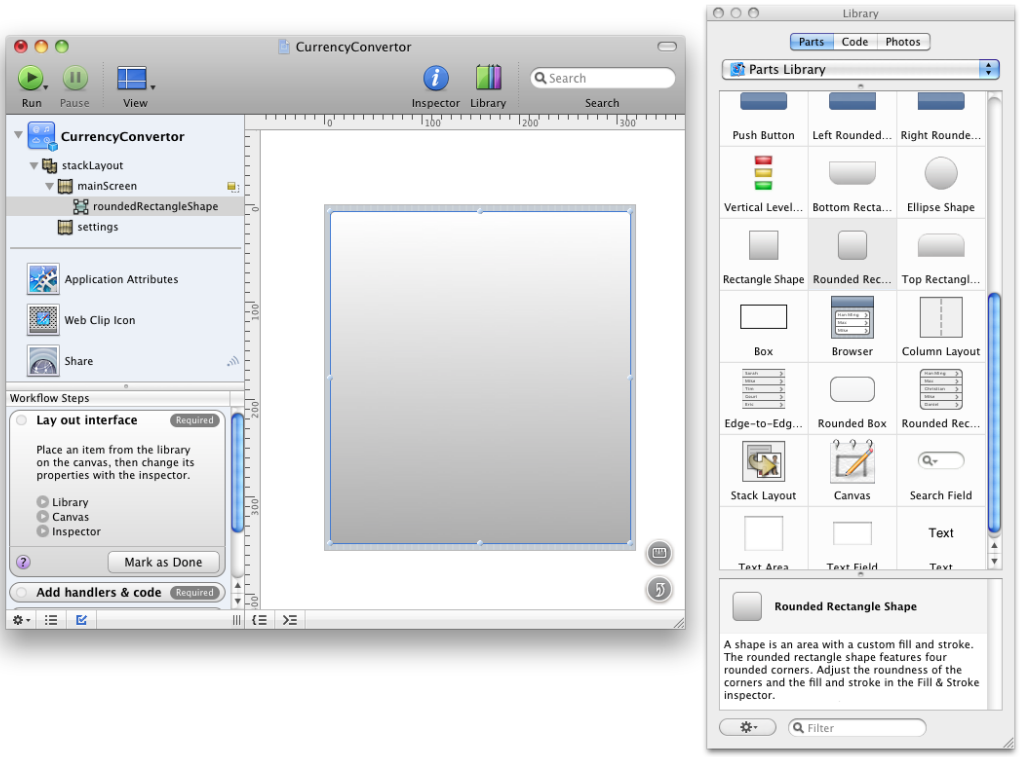
4. Expand the **stackLayout** part and you should see that it contains two subviews - **view1** and **view2**. Select **view1** and change its size to **320px** by **356px** via the **Inspector window** (**Window→Show Inspector**). Do the same for **view2**.



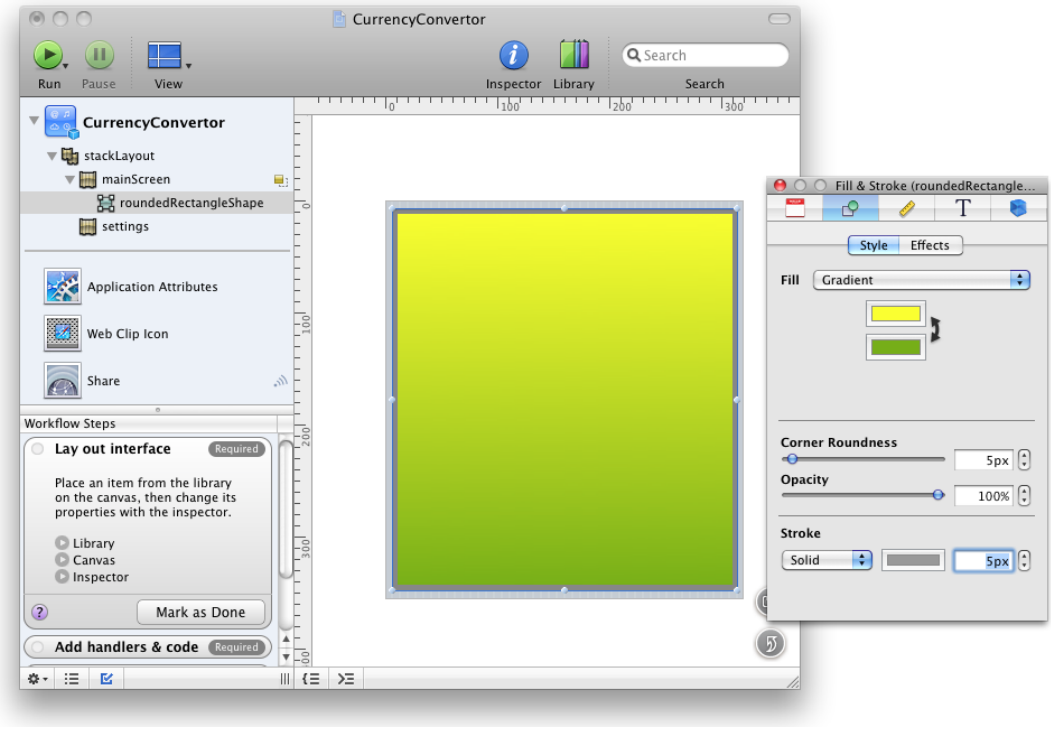
5. Double-click on **view1** and rename it as **mainScreen**. Do the same for **view2** and rename it as **settings**:



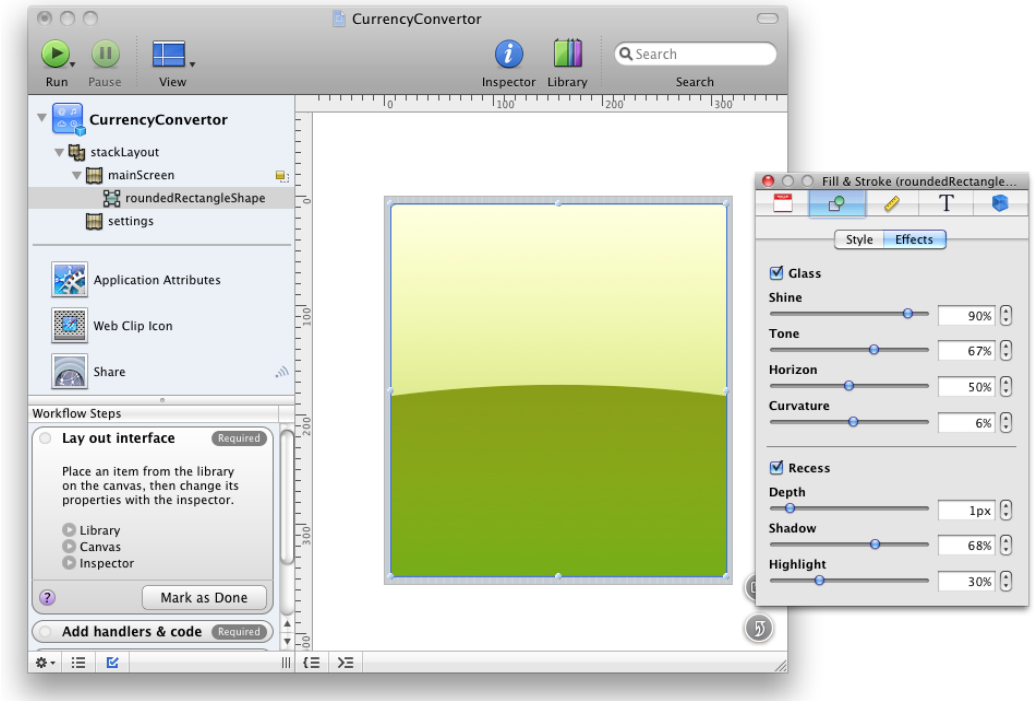
6. In the **Library**, drag-and-drop the **Rounded Rectangle Shape** part onto the **mainScreen** view:



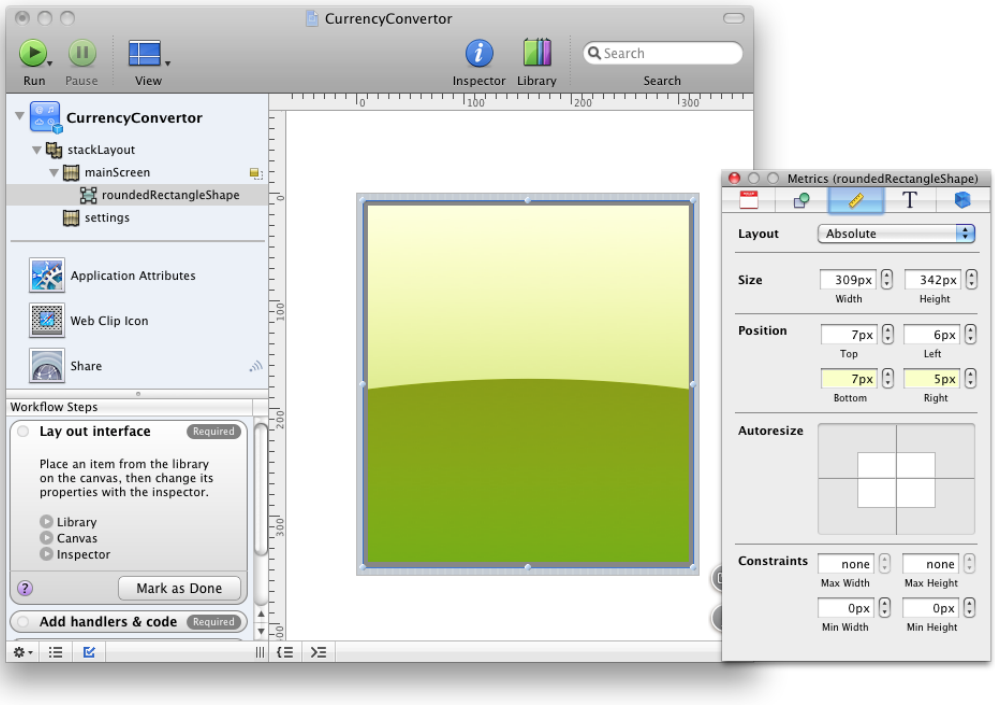
7. In its **Inspector window**, select the **Fill & Stroke** tab and in the **Style** tab select **Gradient** fill and select two colors:



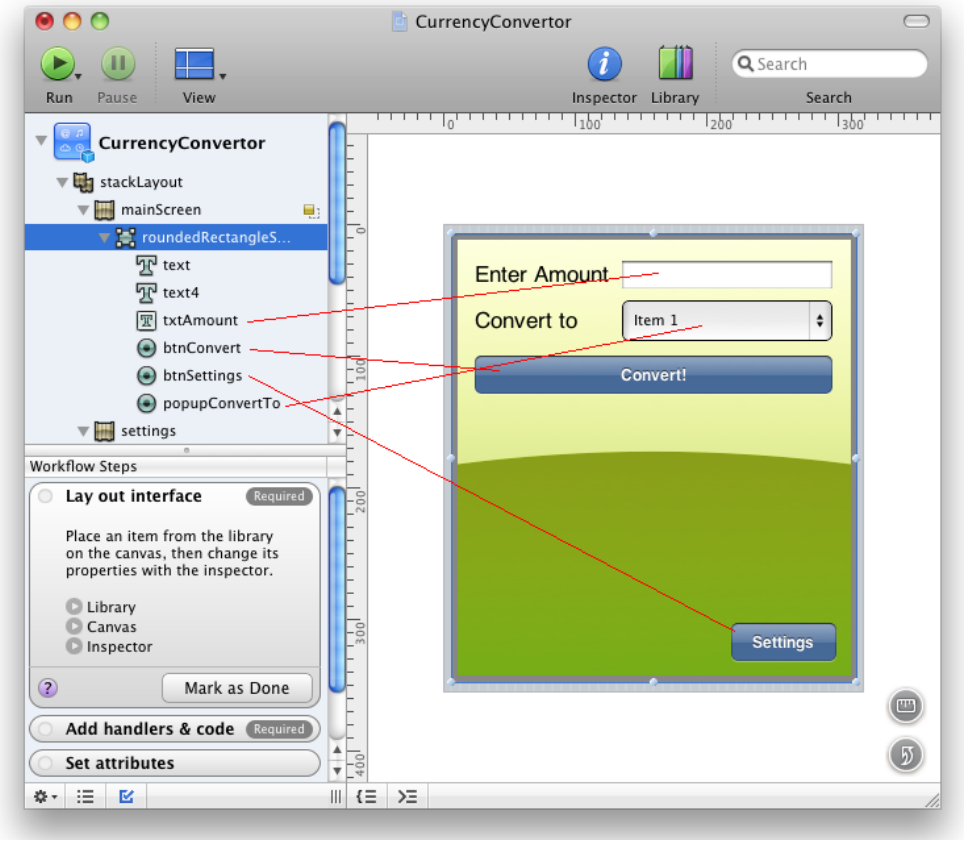
8. Select the **Effects** tab and check the **Glass** and **Recess** checkboxes:



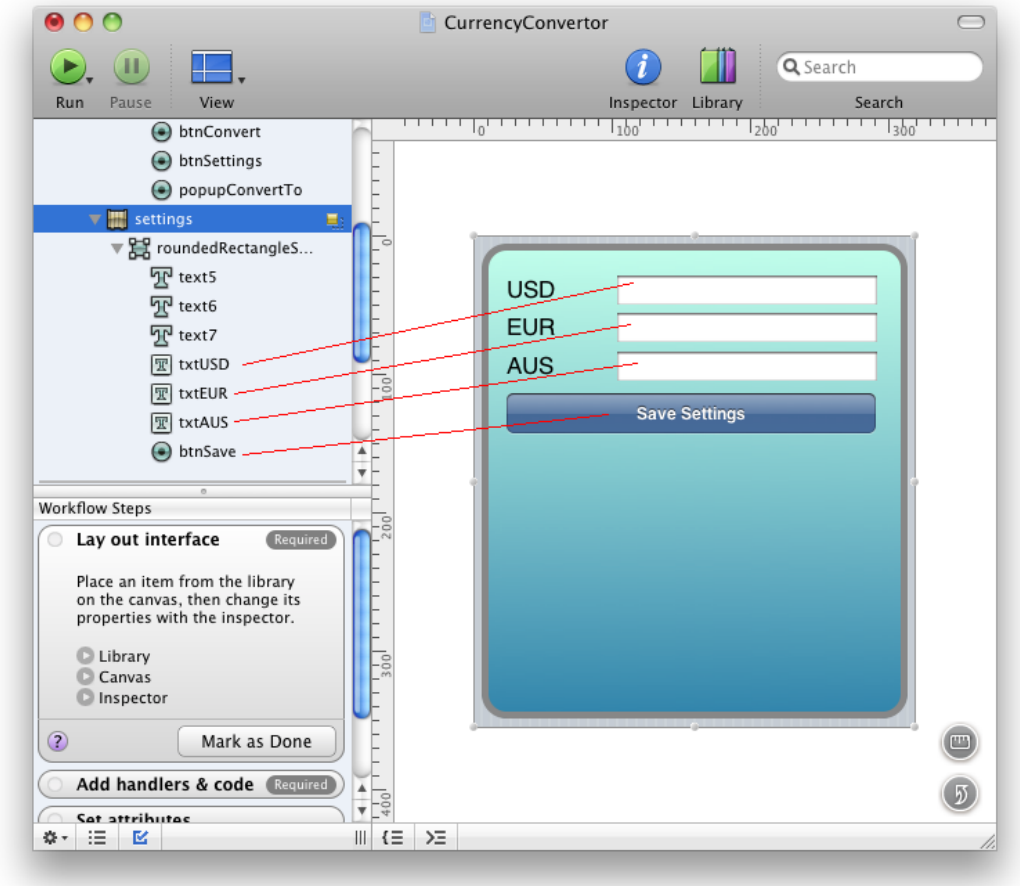
9. Select the **Metrics** tab and select the **Absolute** layout:



10. Add the following parts to the **Rounded Rectangle Shape** part and name them as shown:
    - Text
    - TextField
    - Pop-up Menu
    - Push Button



11. Select the **settings** subview and repeat the same steps you have performed above. The following figure shows the parts added to the **settings** subview.

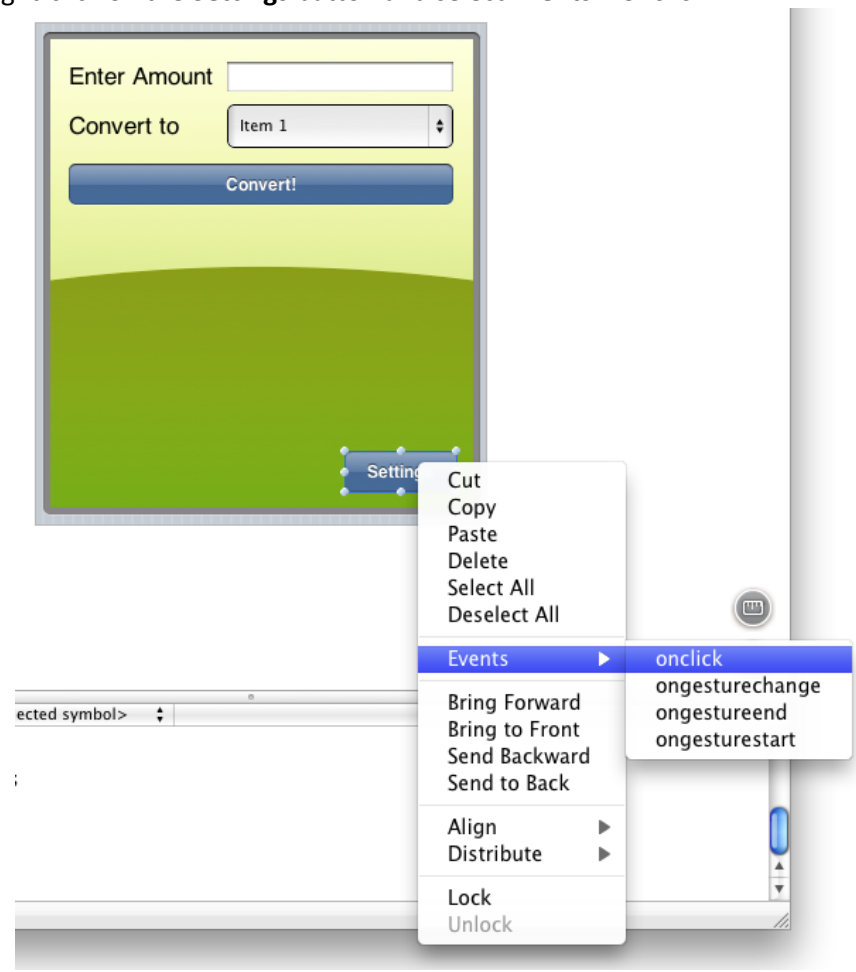


12. You are now ready to view the application on the iPhone Simulator. Press Command-r to view the application on the iPhone Simulator. Notice that the application is displayed by mobile Safari on the iPhone.

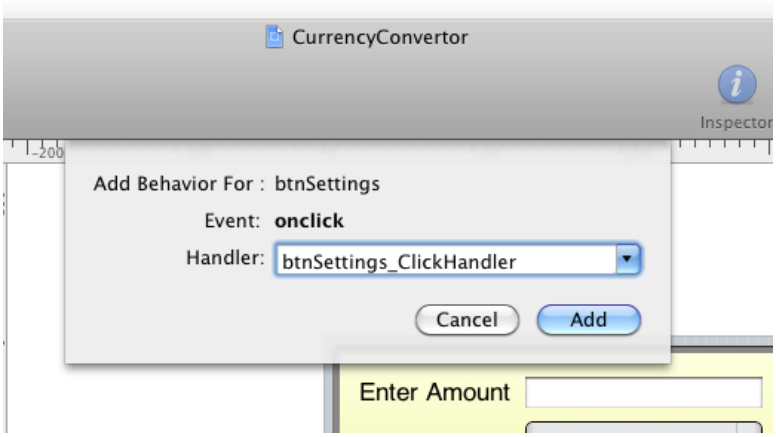
Notice that you can only see the **mainScreen** subview. To see the **settings** subview, you need to write some code to navigate to it from the **mainScreen** subview.

Coding the Application

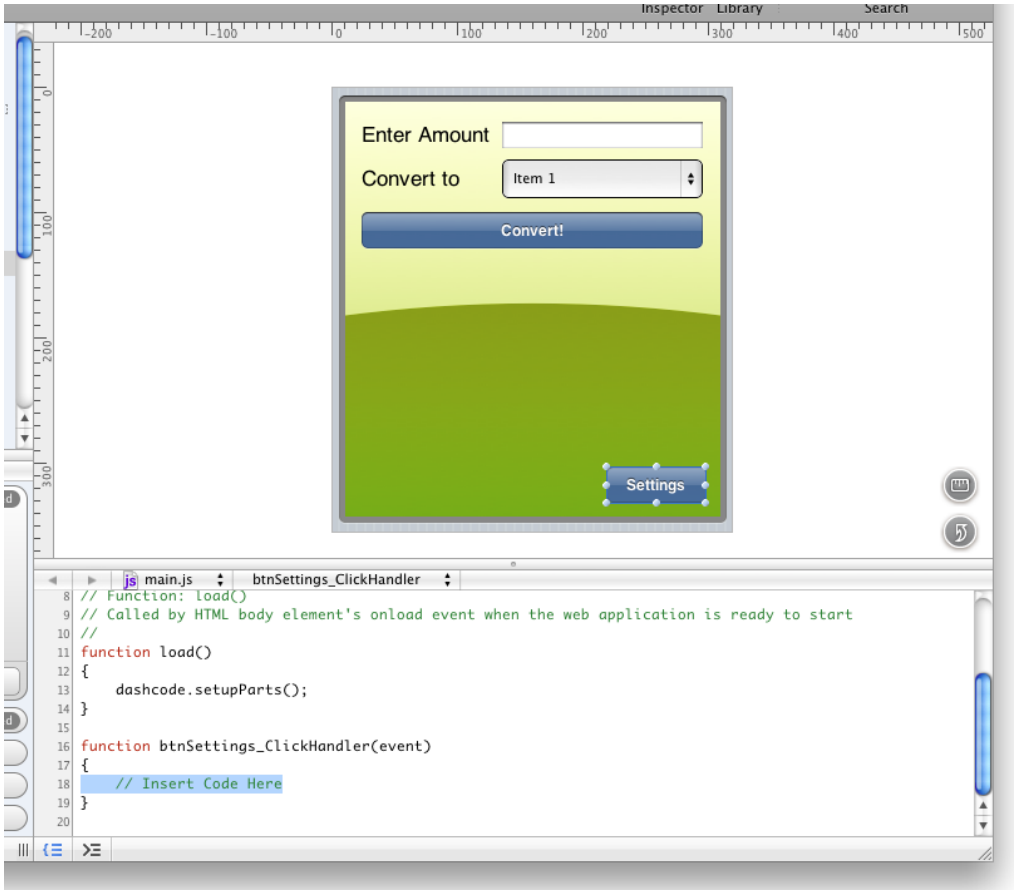
1. So you are now ready to write some code. With the **mainScreen** subview selected, right-click on the **Settings** button and select **Events→onclick**:



2. You will be asked to name the event handler for this event. Name it as shown below:



3. Notice that the code editor now appears at the bottom of the designer:



4. Enter the following code:

```
function btnSettings_ClickHandler(event)
{
    var views = document.getElementById('stackLayout');
    var settings = document.getElementById('settings');
    if (views && views.object && settings) {
        views.object.setCurrentView(settings);
    }
}
```

5. Select the **settings** subview and right-click on the **Save Settings** button and select **Events→onclick**. Name the handler as **btnSave_ClickHandler**. Enter the following code:

```
function btnSave_ClickHandler(event)
{
    var views = document.getElementById('stackLayout');
    var front = document.getElementById('mainScreen');
    if (views && views.object && front) {
        views.object.setCurrentView(front, true);
    }
}
```

6. Test the application again by pressing Command-r. This time, you will be able to navigate to the **settings** view by tapping on the **Settings** button in the **mainScreen** subview:

Database Access

1. In the **main.js** file, add the following lines of code for performing database operations:

```
var database = null;                                // The client-side database
var DB_tableName = "CurrencyKeyValueTable";    // database name

// Function: initDB() - Init and create the local database, if possible
function initDB()
{
    try {
        if (window.openDatabase) {
            database = openDatabase("ExchangeRatesDB", "1.0",
                                    "Exchange Rates Database", 1000);
            if (database) {
                database.transaction(function(tx) {
                    tx.executeSql("SELECT COUNT(*) FROM " +
                    DB_tableName, [],
                    function(tx, result) {
                        loadRates();
                    },
                    function(tx, error) {
                        // Database doesn't exist. Let's create one.
                        tx.executeSql("CREATE TABLE " + DB_tableName +
```

```
                        " (id INTEGER PRIMARY KEY," +
                        "  key TEXT," +
                        "  value TEXT)", [], function(tx, result) {
                            initRates();
                            loadRates ();
                        });
                    });
                });
            }
        }
    } catch(e) {
        database = null;
    }
}

// Function: initRates() - Initialize the default exchange rates
function initRates()
{
    if (database) {
        database.transaction(function (tx) {
            tx.executeSql("INSERT INTO " + DB_tableName +
                " (id, key, value) VALUES (?, ?, ?)", [0, 'USD', 1.44]);
            tx.executeSql("INSERT INTO " + DB_tableName +
                " (id, key, value) VALUES (?, ?, ?)", [1, 'EUR', 2.05]);
            tx.executeSql("INSERT INTO " + DB_tableName +
                " (id, key, value) VALUES (?, ?, ?)", [2, 'AUS', 1.19]);
        });
    }
}

// Function: loadRates() - Load the currency exchange rates from DB
function loadRates()
{
    var element;
    var popUpElement = document.getElementById('popupConvertTo');

    if (database) {
        database.transaction(function(tx) {
            tx.executeSql("SELECT key, value FROM " + DB_tableName, [],
            function(tx, result) {
                for (var i = 0; i < result.rows.length; ++i) {
                    var row = result.rows.item(i);
                    var key = row['key'];
                    var value = row['value'];

                    //---populate the pop-up menu part---
                    popUpElement.options[i].text = key;
                    popUpElement.options[i].value = value;

                    if (key == 'USD') {
                        element = document.getElementById('txtUSD');
                    }
                    else {
                        if (key == 'EUR') {
                            element = document.getElementById('txtEUR');
                        }
                        else if (key == 'AUS') {
                            element = document.getElementById('txtAUS');
                        }
                    }
                    element.value = value;
                }
            },
            function(tx, error) {
                showError('Failed to retrieve stored information from
database - ' +
                    error.message);
```

```
            });
        });
    }
    else {
        loadDefaultRates();
    }
}

// Function: saveRates() - Save the currency exchange rates into DB
function saveRates()
{
    if (database) {
        var elementUSD = document.getElementById('txtUSD');
        var elementEUR = document.getElementById('txtEUR');
        var elementAUS = document.getElementById('txtAUS');

        database.transaction(function (tx) {
            tx.executeSql("UPDATE " + DB_tableName + " SET key = 'USD',
                value = ? WHERE id = 0", [elementUSD.value]);
            tx.executeSql("UPDATE " + DB_tableName + " SET key = 'EUR',
                value = ? WHERE id = 1", [elementEUR.value]);
            tx.executeSql("UPDATE " + DB_tableName + " SET key = 'AUS',
                value = ? WHERE id = 2", [elementAUS.value]);
        });
    }
    loadRates();
}

// Function: deleteTable() - Delete currency exchange table from DB
function deleteTable()
{
    try {
        if (window.openDatabase) {
            database = openDatabase("ExchangeRatesDB", "1.0",
                                    "Exchange Rates Database");
            if (database) {
                database.transaction(function(tx) {
                    tx.executeSql("DROP TABLE " + DB_tableName, []);
                });
            }
        }
    } catch(e) {
    }
}

// Function: loadDefaultRates() - Load the default exchange rates
function loadDefaultRates()
{
    var popUpElement = document.getElementById('popupConvertTo');
    var element = document.getElementById('txtUSD');
    element.value = "1.44";
    popUpElement.options[0].text = "USD";
    popUpElement.options[0].value = element.value;

    element = document.getElementById('txtEUR');
    element.value = "2.05";
    popUpElement.options[1].text = "EUR";
    popUpElement.options[1].value = element.value;

    element = document.getElementById('txtAUS');
    element.value = "1.19";
    popUpElement.options[2].text = "AUS";
    popUpElement.options[2].value = element.value;
}
```

The database code above is pretty straightforward - store the exchange rates inside the database and populate the pop-up menu part when the rates are retrieved.

2. Modify the **load()** function as follows:

```
//
// Function: load()
// Called by HTML body element's onload event when the web application is
ready to
// start
//
function load()
{
    dashcode.setupParts();

    initDB();
    if (!database) {
        loadDefaultRates();
    }
}
```

3. Modify the **btnSave_ClickHandler()** function as follows:

```
function btnSave_ClickHandler(event)
{
    saveRates();
    var views = document.getElementById('stackLayout');
    var front = document.getElementById('mainScreen');
    if (views && views.object && front) {
        views.object.setCurrentView(front, true);
    }
}
```

4. Press Command-R to test the application. When the application is loaded, the pop-up menu will now display the three different currencies:


Carrier
9:26 AM
USD
Convert!
Previous
Next
AutoFill
Done
✓ USD
EUR
AUS

5. When you tap on the **Settings** button, the exchange rates would also be displayed in the **settings** subview:

Performing the Conversion

1. You are now ready to perform the actual conversion of the currencies. In Dashcode, select the **mainScreen** subview and right-click on the **Convert!** Button and select **Events→onclick**.

2. Name the event handler as **btnConvert_ClickHandler** and code it as follows:

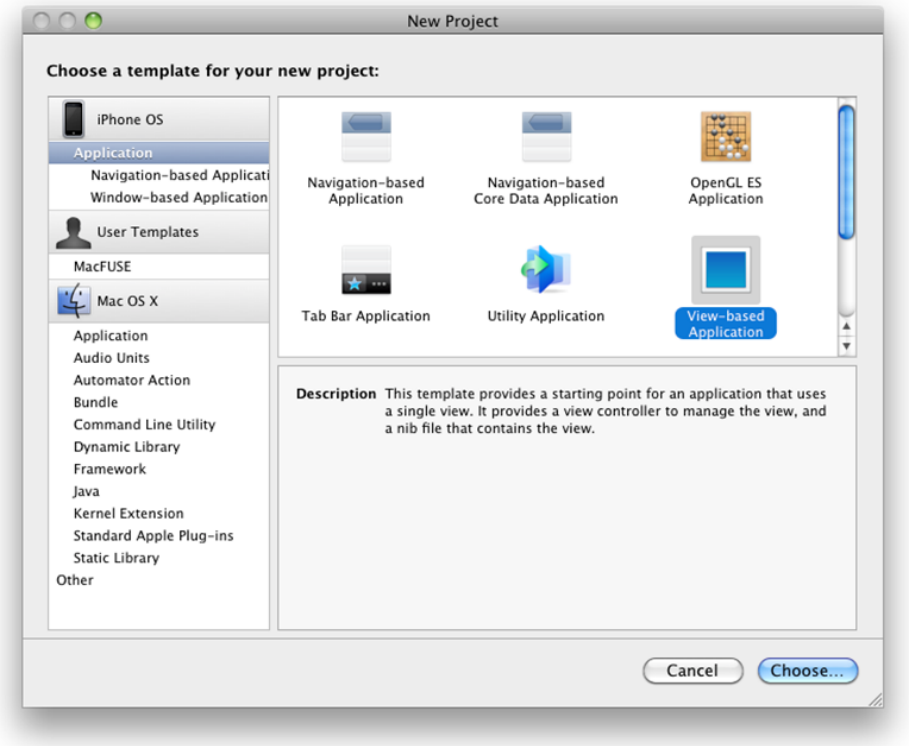
```
function btnConvert_ClickHandler(event)
{
    var amount = document.getElementById("txtAmount").value;
    var rates = document.getElementById("popupConvertTo").value;
    var result = amount * rates;
    alert(result);
}
```

3. Press Command-r to test the application. Enter an amount and select the currency to convert. Tapping on the **Convert!** button will now display the amount converted:

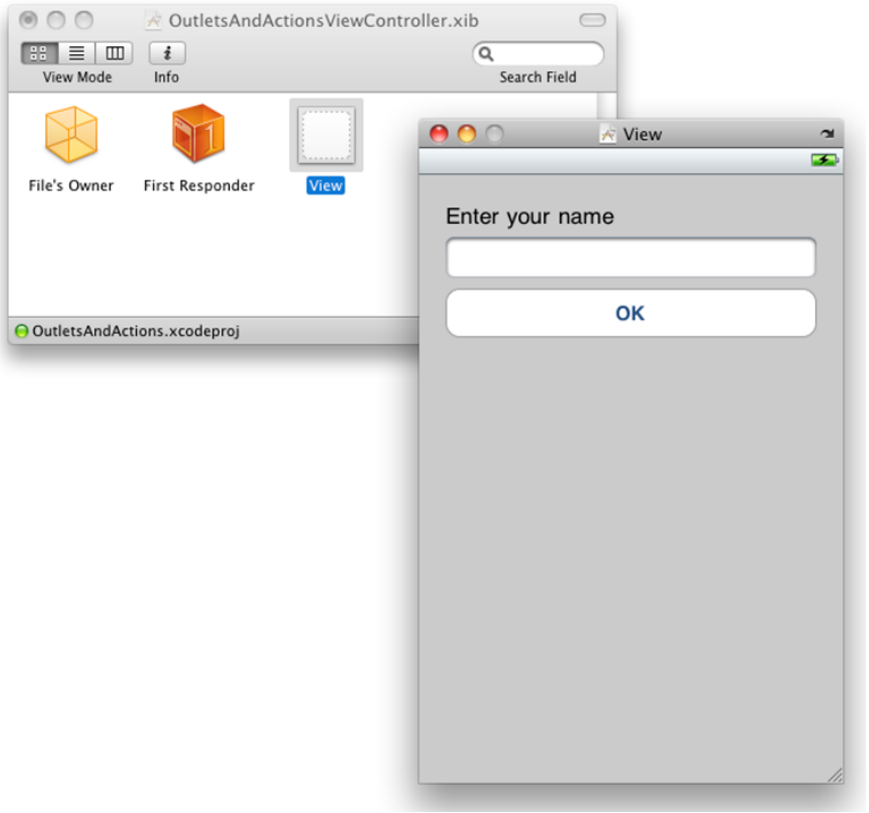
Lab 2 – Getting Started with Xcode

In this lab, you will learn about the fundamentals of iPhone native application programming - understanding the concepts of actions and outlets. You will then build a simple application that displays the current time.

1. Using Xcode, create a **View-based Application** project and name it as **OutletsAndActions**.

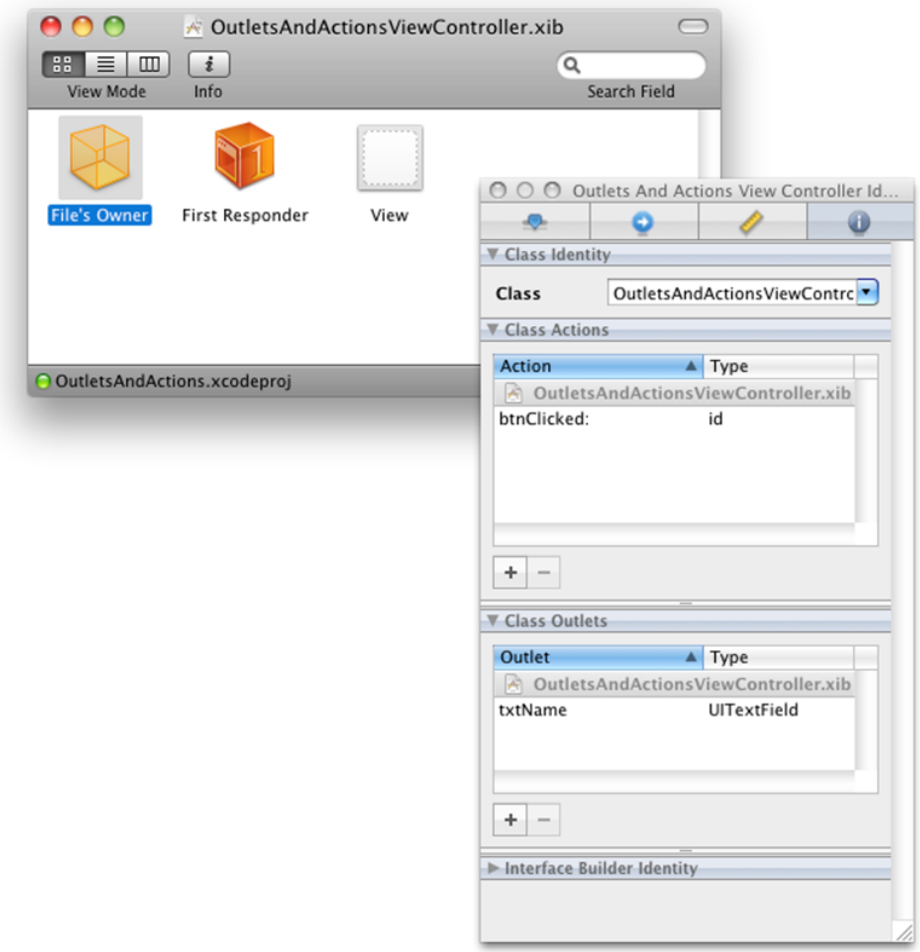


2. Edit the **OutletsAndActionsViewController.xib** file by double-clicking on it to open it in Interface Builder. When Interface Builder is loaded, double-click on the **View** item in the **OutletsAndActionsViewController.xib** window to visually display the View. Populate the three views onto the View window - **Label**, **TextField**, and **Button**.



Creating Actions and Outlets

1. In the **OutletsAndActionsViewController.xib** window, select the **File's Owner** item and view its **Identity Inspector** window. Observe that there are two sections here - **Class Actions** and **Class Outlets**.

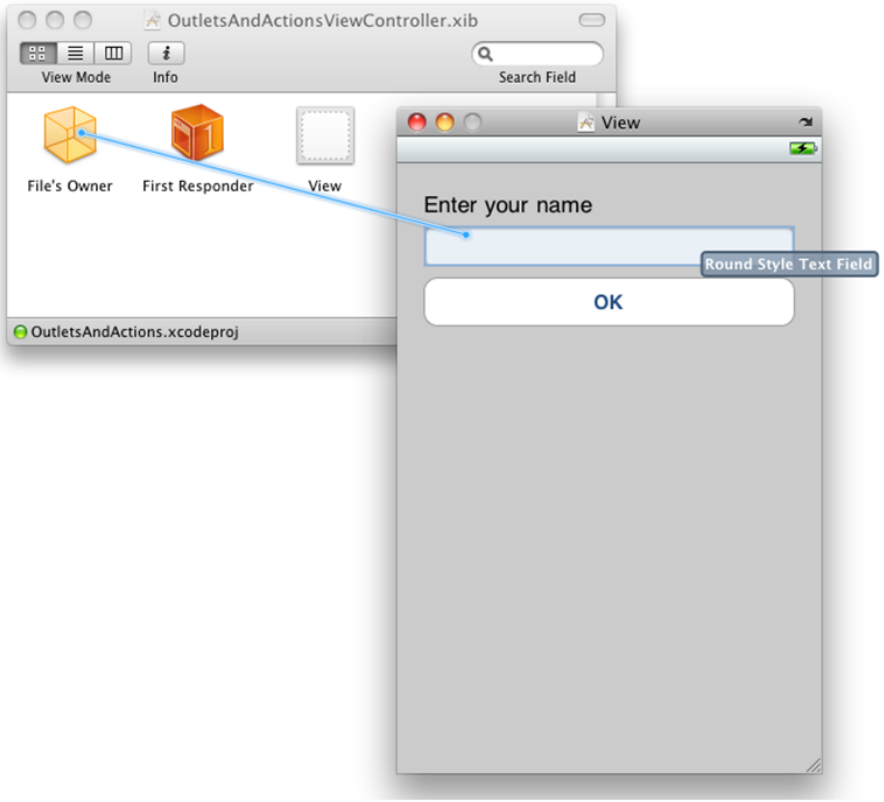


2. To add an action, click on the "+" button under the **Class Actions** section and name the action as **btnClicked:** (take note of the :). This action will be used to handle the event that will be raised when the button is pressed.

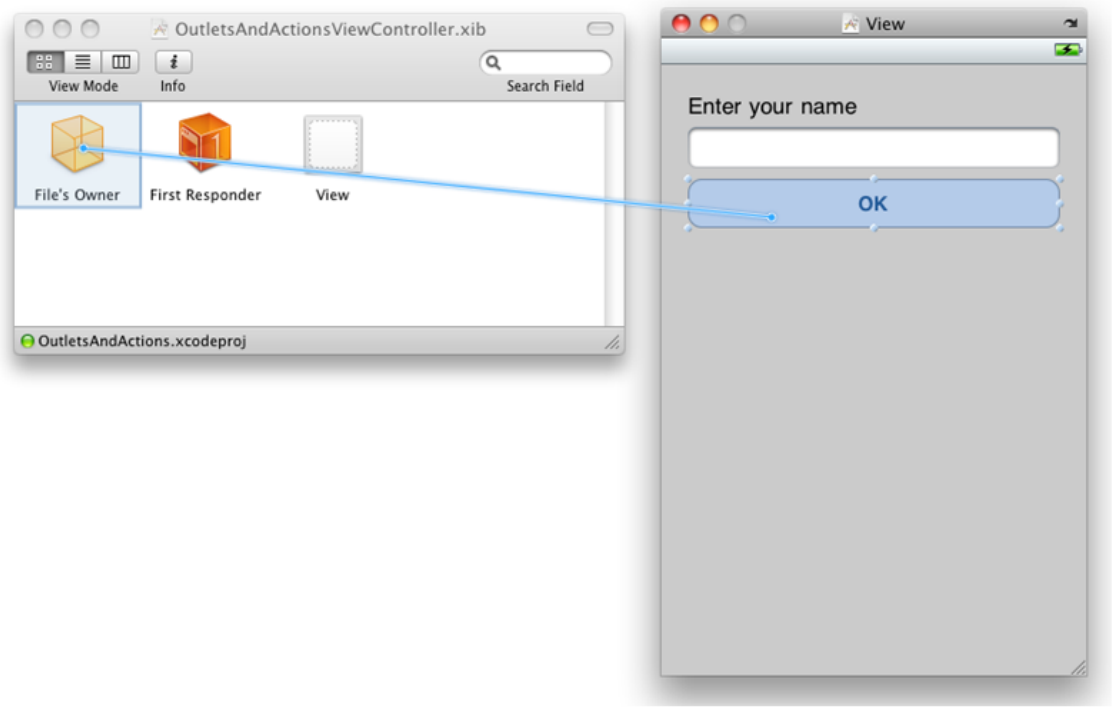
3. Likewise, for the outlet, click on the "+" button and name the outlet as **txtName**. For the outlet, you need to specify the type of view you are referring to. In this case, you will use this outlet to connect to the TextField view programmatically. Hence, specify the type as **UITextField**.

Connecting Actions and Outlets

1. In the **OutletsAndActionsViewController.xib** window, control-click and drag the File's Owner item to the TextField view. A popup will appear; select the outlet named **txtName**.



2. To connect an action, you control-click and drag a view to the File's Owner item. Hence, for the OK Button view, control-click and drag the OK Button view to the File's Owner item. Select the action named **btnClicked:**.



Defining the Actions and Outlets in the View Controller class

1. In the **OutletsAndActionsViewController.h** file, define the following:

```
#import <UIKit/UIKit.h>
@interface OutletsAndActionsViewController : UIViewController {

    //---declaring the outlet---
    IBOutlet UITextField *txtName;

}

//---expose the outlet as a property---
@property (nonatomic, retain) UITextField *txtName;

//---declaring the action---
-(IBAction) btnClicked: (id) sender;

@end
```

2. In the **OutletsAndActionsViewController.m** file, define the following:

```
#import "OutletsAndActionsViewController.h"

@implementation OutletsAndActionsViewController

//---synthesize the property---
@synthesize txtName;

//---displays an alert view when the button is clicked---
-(IBAction) btnClicked:(id) sender {

    NSString *str = [[NSString alloc] initWithFormat:@"Hello, %@",
```

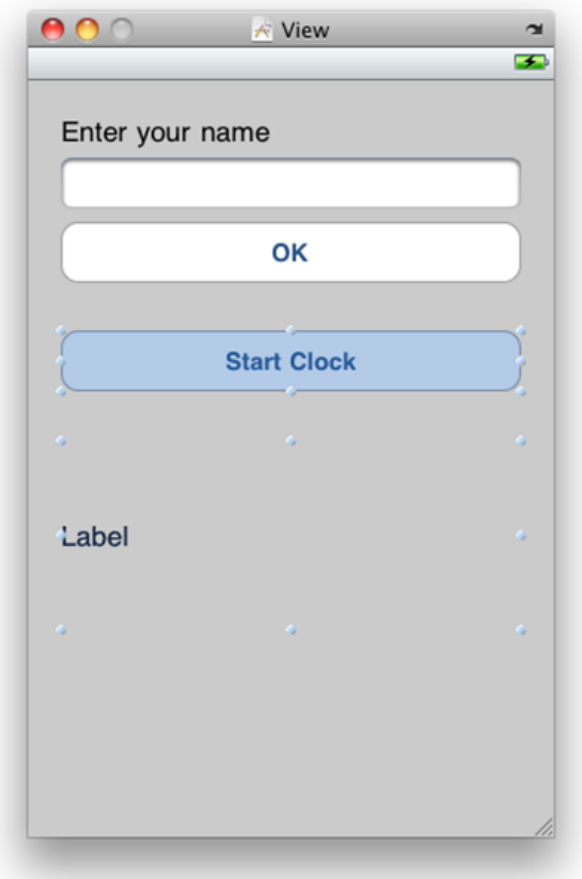
```
                          txtName.text];
    UIAlertView *alert = [[UIAlertView alloc] initWithTitle:@"Hello!"
                                message:str delegate:self
                                cancelButtonTitle:@"Done"
                                otherButtonTitles:nil];
    [alert show];
    [str release];
    [alert release];
}

- (void)dealloc {
    //---release the outlet---
    [txtName release];
    [super dealloc];
}
```

3. Press Command-R to test the application on the iPhone Simulator. When the application is loaded, tap on the TextField view to bring up the keyboard. Enter your name and then press the OK button. You will see the message as shown below:

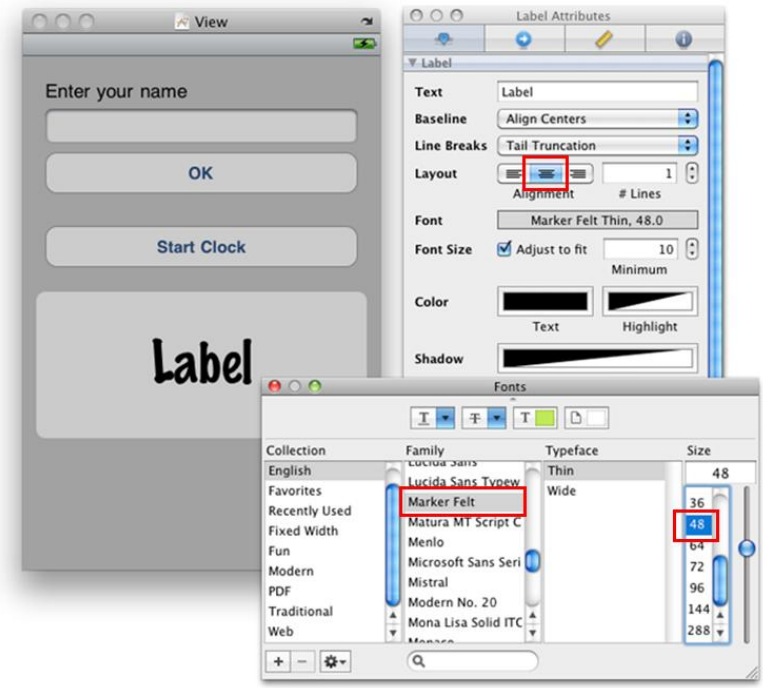
A More Sophisticated Example

Now that you had a good understanding of outlets and actions, let's work on a slightly more sophisticated example so that the application does something useful. You will now modify the application so that you can display the current time continuously, updating it every second. You also have the option to stop and resume the clock if you wish.

1. In the **View** window, add two additional views - **Button** and **Label**:



2. Change the font of the Label view by selecting it and then pressing Command-t. Change its font as shown below.



3. Save the file in Interface Builder and go back to Xcode.

4. In the **OutletsAndActionsViewController.h** file, add the following declarations:

```
#import <UIKit/UIKit.h>
@interface OutletsAndActionsViewController : UIViewController {
    IBOutlet UITextField *txtName;

    //---add the following declarations---
    IBOutlet UIButton *btnStartStop;
    IBOutlet UILabel *lblClock;
    NSDateFormatter *formatter;
    NSDate *date;
    NSTimer *timer;
}

@property (nonatomic, retain) UITextField *txtName;

//---add the following properties declarations---
@property (nonatomic, retain) UIButton *btnStartStop;
@property (nonatomic, retain) UILabel *lblClock;

-(IBAction) btnClicked: (id) sender;

//---add the following declaration---
-(IBAction) btnStartStopClicked: (id) sender;

@end
```

5. Perform the following connections:
   - Connect the **lblClock** outlet to the Label view (that one you just added)
   - Connect the **btnStartStop** outlet to the Button view (that one you just added)
   - Connect the **Start Clock** button to the **btnStartStopClicked:** action.

6. Save the file in Interface Builder.

Coding the Application

1. Back in Xcode, define the following in the **OutletsAndActionsViewController.m** file:

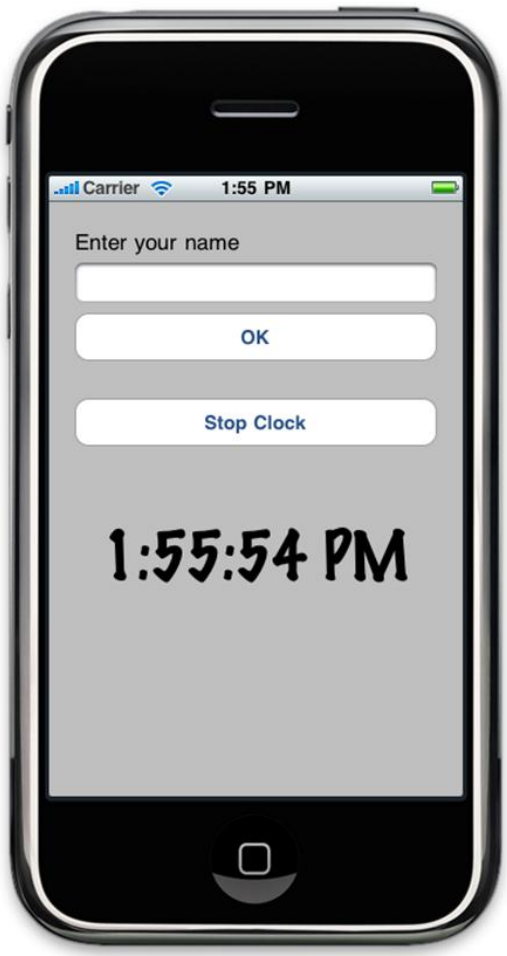
```
#import "OutletsAndActionsViewController.h"

@implementation OutletsAndActionsViewController

@synthesize txtName;

//---synthesize all the properties---
@synthesize btnStartStop;
@synthesize lblClock;

- (void)viewDidLoad {
    //---initialize the NSDateFormatter object---
    formatter =[[NSDateFormatter alloc] init];
    [super viewDidLoad];
}

-(IBAction) btnStartStopClicked: (id) sender {
    if ([[btnStartStop titleForState:UIControlStateNormal]
    isEqualToString:@"Start Clock"])
    {
        //---start the timer---
        timer = [NSTimer scheduledTimerWithTimeInterval:(1)
                 target:self
                 selector:@selector(updateTime)
                 userInfo:nil
```

```
                repeats:YES];

        //---change the caption to "Stop Clock"---
        [btnStartStop setTitle:@"Stop Clock" forState:UIControlStateNormal];
    }
    else
    {
        //---stop the timer---
        [timer invalidate];

        //---change the caption back to "Start Clock"---
        [btnStartStop setTitle:@"Start Clock"
             forState:UIControlStateNormal];
    }
}
```

```
//---called every second---
-(void)updateTime{
    //---display the time---
    date = [NSDate date];
    [formatter setTimeStyle:NSDateFormatterMediumStyle];
    lblClock.text = [formatter stringFromDate:date];
}
```

```
- (void)dealloc {
    [txtName release];
    //---release all the outlets and objects---
    [btnStartStop release];
    [lblClock release];
    [formatter release];
    [date release];
    [super dealloc];
}
```

2. To test the application, press Command-r in Xcode. Tapping the **Start Clock** button will start the clock, which updates itself every one second.


Carrier
1:55 PM
Enter your name
OK
Stop Clock
1:55:54 PM