iOS Developer's Guide

Version 1.0

Tuesday May 7, 2013

Revision Sign-Off

By signing the following, the team member asserts that he has read the entire document and has, to the best of his knowledge, found the information contained herein to be accurate, relevant, and free of typographical error.

Name	Signature	Date
Alex Anduss		
Baer Bradford		
Greg Kolesar		

Revision History

The following is a history of document revisions.

Version	Date Revised	Comments
0.1	Apr. 28, 2013	Initial draft.
1.0	May 7, 2013	Version 1.0

Table of Contents

1. Introduction

1.1 Purpose

The purpose of this document is to aid any developers with this system through modifying the app, deploying test versions to devices, and deploying updates to the App Store.

1.2 Overview

GoodNEWS Fort Worth is a program created by the University of North Texas Health Science Center in conjunction with Fort Worth City Leaders to promote healthy living in the city. In the past, the program had been a series of seminars that were used to assist citizens with their lifestyle. The app will be a great way for the GoodNEWS team to reach out and help the citizens of Fort Worth better their lifestyles. The app will also allow the citizens to respond to surveys and the GoodNEWS team can use the results to make Fort Worth a healthier place.

2. iOS Device Requirements

This section lists the requirements needed to run the GoodNEWS Fort Worth App.

2.1 iOS Version

iOS Version 6.0 and higher are required to use this app. iOS 6 is required due to the use of Auto-Layout, a feature available in iOS 6+. This feature allows the developer to write for one screen size and it will attempt to infer and adapt for the other screen sizes.

2.2 iPhones

The iPhones that support iOS 6.0 are the iPhone 3GS, iPhone 4, iPhone 4S, iPhone 5.

2.3 iPod Touches

The iPod Touch devices that support iOS 6 are the iPod Touch 4th Generation and the iPod Touch 5th Generation. The easiest way to tell if you iPod Touch can run iOS 6 is to see if it has a front facing camera. If it does, it is a 4th or 5th generation.

2.4 iPads

The iPad 2, iPad 3rd Generation, iPad 4th Generation, and the iPad Mini will all support use of this App. The App does not support the iPad's full resolution, but you can run it in a sandboxed mode (See Section 3.2).

2.4 Network

The App does require constant connection to a network, it can be a 3G, 4G, or LTE cellular network, or a Wi-Fi network. If for some reason you lose network capability, you will be logged out of your account for security and data integrity issues.

2.5 Wikipedia's List of iOS devices and specifications

http://en.wikipedia.org/wiki/List_of_iOS_devices

3. Data Model

This section lays out the Data Model as seen in **Figure 3.1**.

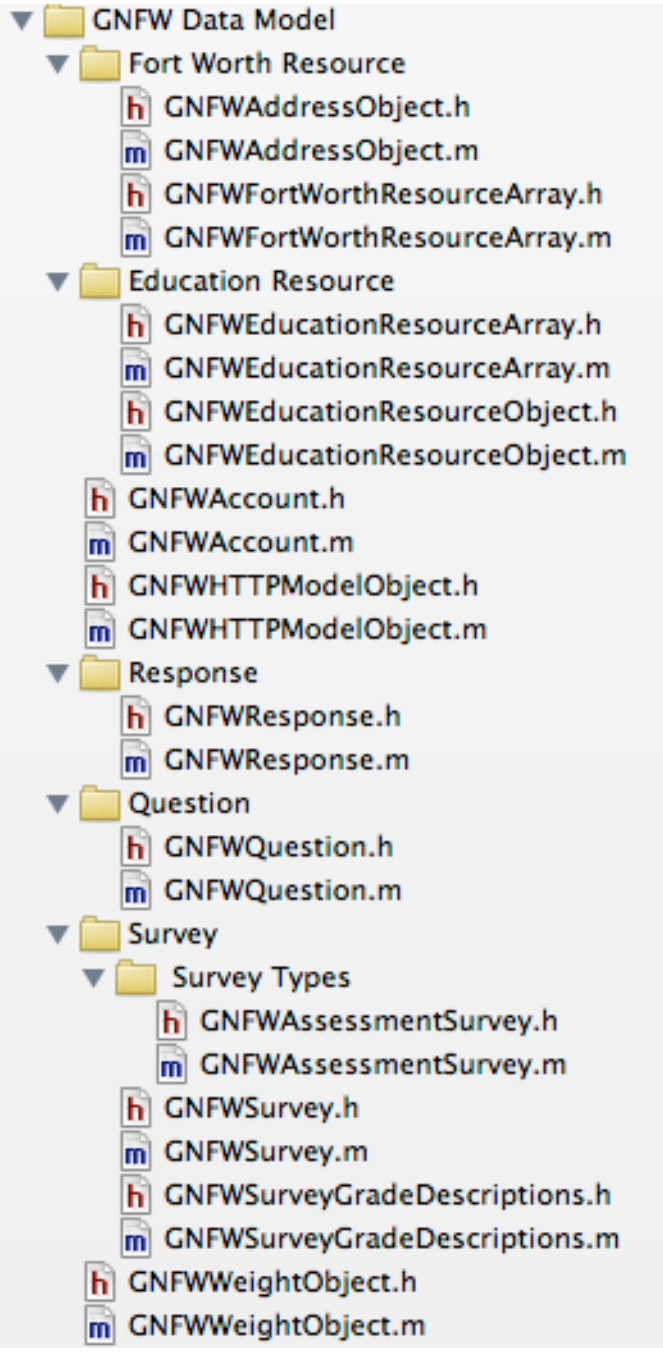


Figure 3.1

3.1 Account Singleton

When the user creates their account or logs in, all of the important user data is stored in a single account object, or the account singleton. When a user creates their account and all of the information is submitted to the CMS, there is a call made to grab all of the account information. The JSON Parser automatically creates a dictionary of this object which is then stored in the _accountDictionay ivar (instance variable) . **Figure 3.2** will show the header files and the getter methods that can be used to get information from the singleton.

```
#import <Foundation/Foundation.h>
@class GNFWWeightObject;
@interface GNFWAccount : NSObject
{
   NSMutableDictionary* _accountDictionary;
}

@property (nonatomic,strong) GNFWWeightObject* weightObject;

//Setter Methods
-(void)setUserId:(NSNumber*)userId;
-(void)setEmailAddress:(NSString*)emailAddress;
-(void)setPassword:(NSString*)password;
-(void)setGender:(NSString*)gender;
-(void)setAge:(NSString*)age;
-(void)setEducationLevel:(NSString*)educationLevel;
-(void)setIncomeRange:(NSString*)incomeRange;
-(void)setEthnicGroup:(NSString*)ethnicGroup;
-(void)setAdmin:(NSNumber*)isAdmin;
-(void)setWeightManagementEnrollment:(NSNumber*)enrolled;
-(void)setDictionary:(NSDictionary*)dictionary;

//Getter Methods
-(NSDictionary*)accountDictionary;
-(NSString*)emailAddress;
-(NSString*)password;
-(BOOL)enrolledInWeightManagement;
-(int)userId;

-(void)printAccount;
-(void)saveToUserDefaults;
-(void)readFromUserDefaults;
-(void)logoutAccount;

+(GNFWAccount*)account;
@end
```

Figure 3.2 GNFWAccount.h

I made the decision to use methods instead of properties for the data, because I used the dictionary to hold all of my data. In hindsight I should have made the dictionary a property, but this way also works. By using the methods to set and get the data, I could access the dictionary much easier. I decided to use the dictionary, because I could just pass the whole dictionary to the JSON parser which formatted it properly for the CMS. **Figure 3.3** demonstrates a few method calls to get information from the singleton.

```
NSMutableAttributedString* subject = [[NSMutableAttributedString alloc] initWithString:[NSString
    stringWithFormat:@"%@ is logged in",[[GNFWAccount account] emailAddress]]];
```

Figure 3.3 Sample Calls to the Singleton

The call will always start with [GNFWAccount account] which gets the instance of the singleton, and then any of the methods in GNFWAccount.h.

3.2 Survey

There are three parts to the data structure of surveying. The main survey class is GNFWSurvey.h. This is seen in **Figure 3.4.**

```
#import <Foundation/Foundation.h>

@class GNFWQuestion;
@class GNFWSurveyGradeDescriptions;

@interface GNFWSurvey : NSObject
{
   int _surveyId;
   int _surveyType;
   NSString* _name;
   NSMutableArray* _questions;
   NSMutableArray* _gradeDescriptions;
}

@property (nonatomic) BOOL assessmentFinished;
@property (nonatomic) BOOL assessmentStarted;
@property (nonatomic) int lastQuestionAnswered;

-(void)addQuestion:(GNFWQuestion*)question;
-(void)addGradeDescription:(GNFWSurveyGradeDescriptions*)surveyGradeDescription;
-(NSString*)surveyName;
-(NSArray*)questions;
-(NSArray*)gradeDescriptions;
-(int)surveyID;
@end
```

Figure 3.4 GNFWSurvey.h

GNFWSurvey.h is the basic survey object. It contains the survey data, and the questions. While the user is taking a survey, the object also keeps track of how the survey is progressing, like if it has started, finished, or what was the last question the user was on. This object is used for the Community Vote and Poll section since there is no need to grade them. For the Track and Progress and Health Assessment surveys, there is the

GNFWAssessmentSurvey.h object. This is a subclass of GNFWSurvey.h which means it inherits everything GNFWSurvey does, but allows for more. It leads to simpler code. GNFWAssessmentSurvey handles all of the grading of a survey that was just taken, it also stores the average grade, most recent grade, and the most recent time that a survey was taken. The final surveying object is the GNFWSurveyGradeDescriptions object. This object is used to store the upper and lower bound of a survey grade range and it also hold the description that goes along with these grades. The grade descriptions are stored in the GNFWSurvey object's _gradeDescriptions ivar.

3.3 Question

The GNFWQuestion object is a simple object as seen in **Figure 3.5**.

```
#import <Foundation/Foundation.h>
@class GNFWResponse;
@class GNFWQuestionView;
@class GNFWSurvey;
@interface GNFWQuestion : NSObject
{
   int _questionId;
   NSString* _questionText;
   NSMutableArray* _responses;
   GNFWSurvey* _parentSurvey;
}

@property (nonatomic) int score;
@property (nonatomic,strong) GNFWResponse* selectedResponse;

-(id)initWithId:(int)questionId questionText:(NSString*)questionText parentSurvey:(GNFWSurvey*)
    parentSurvey;
-(GNFWSurvey*)parentSurvey;
-(void)addResponse:(GNFWResponse*)response;
-(int)selectedResponseScore;
-(NSString*)questionText;
-(NSArray*)responses;
@end
```

Figure 3.5 GNFWQuestion.h

This object holds an array of responses, the question text, and it also has the parent survey which makes it easy to traverse the web of survey-question-responses. It also holds a pointer to the selected response which makes it easy for the developer to get the response id for the CMS.

3.4 Response

The response object is also simple. See **Figure 3.6**.

```
#import <Foundation/Foundation.h>
@class GNFWQuestion;
@interface GNFWResponse : NSObject
{
   int _responseId;
   NSString* _responseText;
   int _pointValue;
   GNFWQuestion* _parentQuestion;
}

-(id)initWithId:(int)responseId text:(NSString*)responseText pointValue:(int)pointValue
    parentQuestion:(GNFWQuestion*)parentQuestion;
-(int)pointValue;
-(GNFWQuestion*)parentQuestion;
-(NSString*)responseText;
-(int)responseId;
@end
```

Figure 3.6 GNFWResponse.h

This object holds the response id, response text, point value of the response for grading, and also the parent question, which also can be used to get the parent survey.

3.5 Fort Worth Resource

Fort Worth Resources are slightly complicated due to the need for Geoencoding. There is a handy wrapper class for it to keep the ugly code from being exposed to the developer.

```
#import <Foundation/Foundation.h>
@class GNFWAddressObject;
@interface GNFWFortWorthResourceArray : NSObject
{
   NSMutableArray* _fortWorthResourceArray;
}

@property (readonly,strong) NSString* sectionName;
@property (readonly) int sectionId;

-(id)initWithSectionName:(NSString*)sectionName sectionId:(NSInteger)sectionId;
-(void)addResourceWithName:(NSString*)businessName businessId:(NSNumber*)businessId;
-(void)updateResourceWithName:(NSString*)name link:(NSString*)link phoneNumber:(NSString*)phoneNumber
    streetName:(NSString*)streetName city:(NSString*)city state:(NSString*)state zipCode:(NSNumber*)
    zipCode description:(NSString*)description;
-(NSArray*)fortWorthResourceArray;
-(NSString*)nameForObjectAtIndex:(int)index;
-(GNFWAddressObject*)objectAtIndex:(int)index;
-(int)numberOfObjectsInArray;
@end
```

Figure 3.7 GNFWFortWorthResourceArray.h

The GNFWFortWorthResourceArray class holds the array of all the Fort Worth Resources for a certain section. The object is first initialized in the web service call when the user clicks on Fort Worth Resources. To make it simple for the developer to add objects to the array, they do not have to create an AddressObject themselves, the developer just has to call to add a resource and then it will create the AddressObject and

add it to the array. Due to some last minute changes, the updateResourceWithName method is no longer used. It was removed to improve efficiency with the CMS call and memory management on the devices.

```
#import <Foundation/Foundation.h>
#import <CoreLocation/CoreLocation.h>
#import <MapKit/MapKit.h>
#import <AddressBook/AddressBook.h>
#import <AddressBook/AddressBook.h>

@protocol GNFWAddressObjectProtocol <NSObject>
-(void)geolocationFinishedWithAnnotation:(MKPointAnnotation*)annotation;
@end

@interface GNFWAddressObject : NSObject
{
   BOOL _geoencodeFinished;
   BOOL _locationGeoencoded;
   BOOL _locationRequested;
   NSDictionary* _addressDictionary;
   id<GNFWAddressObjectProtocol> _delegate;
   MKPointAnnotation* _annotationPoint;
}

@property (readonly) NSNumber* resourceId;
@property (readonly,strong) NSString* streetName;
@property (readonly,strong) NSString* city;
@property (readonly,strong) NSString* state;
@property (readonly,strong) NSString* phoneNumber;
@property (readonly,strong) NSURL* link;
@property (readonly) NSNumber* zipCode;
@property (readonly,strong) NSString* businessName;
@property (readonly,strong) NSString* description;
@property (readonly,strong) NSString* fullAddressString;


-(id)initWithBusinessName:(NSString*)businessName businessId:(NSNumber*)businessId;
-(id)initWithStreetName:(NSString*)streetName city:(NSString*)city state:(NSString*)state zipCode:
    (NSNumber*)zipCode businessName:(NSString*)businessName phoneNumber:(NSString*)phoneNumber link:
    (NSURL*)link description:(NSString*)description;
-(MKPointAnnotation*)requestAnnotation:(id)sender;
-(NSDictionary*)addressDictionary;

@end
```

Figure 3.8 GNFWAddressObject.h

The GNFWAddressObject is created when the CMS call is made to retrieve a particular Fort Worth Resource. This object was tricky to create, because some of the data values may come back null, and dictionaries do not handle null values well, so I used properties instead, and since everything is set one in the init method, all of the null checking is handled there. Also, when this object is initialized with a full address, an asynchronous call is made to Geoencode the location. The asynchronous call then notifies the object when it has finished Geoencoding so it can be used with the mapping feature.

3.6 Health Education Resource

A Health Education Resource object consists of a Name and a URL.

4. iOS Web Service

This section will lay out how to use the web service. **Figure 4.1** shows all of the classes contained in the web service.

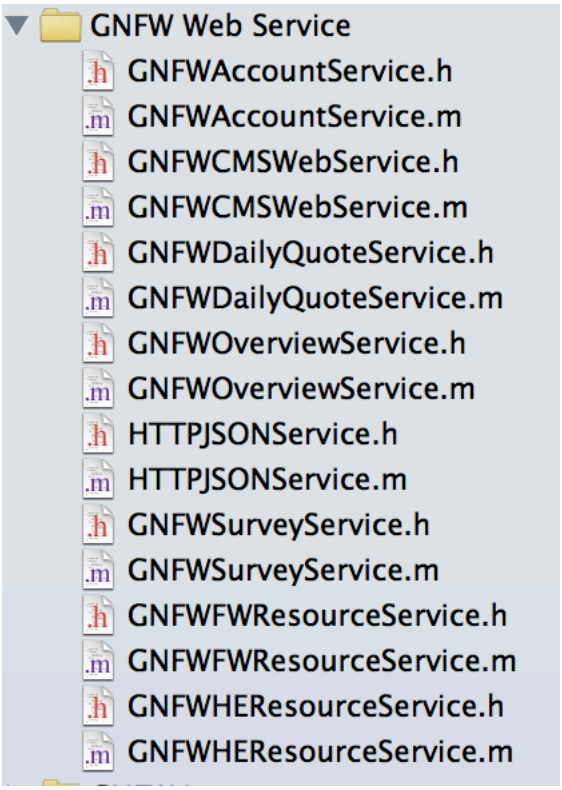


Figure 4.1 Web Service Class list

4.1 Web Service Singleton

Using the web service singleton is important to the flow of the program. The singleton and the classes that interact with it support many features including asynchronous calls, fire-and-forget calls, and callbacks. It is simple to use the singleton object. The first thing you want to do when creating a new class for a different service is to create the property in the interface file, **Figure 4.2** shows the existing properties and the new one will just follow that format.

```
#import <Foundation/Foundation.h>
@class HTTPJSONService;
@class GNFWDailyQuoteService;
@class GNFWAccount;
@class GNFWAccountService;
@class GNFWOverviewService;
@class GNFWSurveyService;
@class GNFWFWResourceService;
@class GNFWHEResourceService;

@interface GNFWCMSWebService : NSObject
@property (nonatomic, strong) HTTPJSONService* httpService;
@property (nonatomic, strong) GNFWDailyQuoteService* dailyQuoteService;
@property (nonatomic, strong) GNFWAccountService* accountService;
@property (nonatomic, strong) GNFWOverviewService* overviewService;
@property (nonatomic, strong) GNFWSurveyService* surveyService;
@property (nonatomic, strong) GNFWFWResourceService* fwResourceService;
@property (nonatomic, strong) GNFWHEResourceService* heResourceService;
//Shared web service instance
-(void)closeWebService;
+(GNFWCMSWebService*)sharedWebService;
@end
```

Figure 4.2 GNFWCMSWebService.h

As you can see there is not much to it, just declare the class and the variable name you would like to give it. You can really see the magic of the singleton work in the implementation file, see **Figure 4.3**.

```
#import "GNFWCMSWebService.h"
#import "HTTPJSONService.h"
#import "GNFWDailyQuoteService.h"
#import "GNFWAccount.h"
#import "GNFWAccountService.h"
#import "GNFWOverviewService.h"
#import "GNFWSurveyService.h"
#import "GNFWFWResourceService.h"
#import "GNFWHEResourceService.h"

static GNFWCMSWebService* sWebService = nil;

@implementation GNFWCMSWebService
@synthesize httpService = _httpService;
@synthesize dailyQuoteService = _dailyQuoteService;
@synthesize overviewService = _overviewService;
@synthesize surveyService = _surveyService;
@synthesize fwResourceService = _fwResourceService;
@synthesize heResourceService = _heResourceService;

-(id)init
{
   self = [super init];
   if (self)
   {
      _httpService = [[HTTPJSONService alloc] init];
      _dailyQuoteService = [[GNFWDailyQuoteService alloc]initWithCMSService:self];
      _accountService = [[GNFWAccountService alloc]initWithCMSService:self];
      _overviewService = [[GNFWOverviewService alloc]initWithCMSService:self];
      _surveyService = [[GNFWSurveyService alloc]initWithCMSService:self];
      _fwResourceService = [[GNFWFWResourceService alloc]initWithCMSService:self];
      _heResourceService = [[GNFWHEResourceService alloc]initWithCMSService:self];
   }
   return self;
}

#pragma mark - Singleton
+(GNFWCMSWebService*)sharedWebService
{
   if(nil == sWebService)
   {
      sWebService = [[GNFWCMSWebService alloc]init];
   }
   return sWebService;
}

-(void)closeWebService
{
   sWebService = nil;
}
@end
```

Figure 4.3 GNFWCMSWebService.m

By initializing all of the separate service classes in the singleton, you are able to access the methods contained in these classes from anywhere. This is important, because it only creates a single instance of these objects, and the developer does not need to worry about ARC (Automatic Reference Counting), or garbage collection, from destroying your instances of these objects before they can call back.

4.2 Create new GET call

Getting information from the content management system efficiently is quintessential to the functionality of the app. To ensure that a stable connection is established, a single method was created in the HTTPJSONService class to get the data from the CMS. The JSONService was already created in the singleton, and it contains an exposed method which takes the URL from the CMS API, the http method, which in this case will be @"GET", there is no data to go in, so that will get a nil, the sender will be *self*, and the handler, which you will see in **Figure 4.4**. By putting *self* in the sender field, it passes the current instance of the object making the call, this will allow the developer to call back the object when the web call has finished.

```
-(void)getDailyQuoteFromCMS:(id)senderController
{
    [_cmsService.httpService callToCMSWithURL:@"http://cms.goodnews-trueliving.com/API/DailyQuote"
                                   httpMethod:@"GET"
                                         data:nil
                                       sender:self
                                      handler:^(NSData* receivedData, NSError* error, id context, id sender)
    {
        if(error == nil)
        {
            NSDictionary* jsonDictionary = [NSJSONSerialization JSONObjectWithData:receivedData
                                                                            options:NSJSONWritingPrettyPrinted
                                                                              error:NULL];

            if([jsonDictionary isKindOfClass:[NSDictionary class]])
            {
                [senderController receivedDailyQuote:[jsonDictionary objectForKey:@"DailyQuote"]];
            }
            else
            {
                [senderController receivedDailyQuote:nil];
            }
        }
        else
        {
            [senderController receivedDailyQuote:nil];
        }
    }];

}
```

Figure 4.4 Example GET Call

Figure 4.4 shows an example GET call for the daily quote. Here you can see how the handler works. The handler is a C block, it uses a C struct with Objective-C data types to sendback information. As you can see, it includes the recievedData, which is the JSON data received from the server. The NSError object is either created by the JSONService call, or it is user defined for a few cases, if there is no error, it will come back as nil, and all of the errors are handled on the back end. The context is not used by any calls, but it can be used to store any additional data that may need to come back. Finally, the sender is the original class that made the call, and it allows for callbacks. The block works just like a method, when the web call finishes, everything will return here. Again, you can check to see if there was an error, and handle anything that may be needed there, and you can use the built in JSON parser with the recievedData to generate appropriate Objective-C objects. When you get the JSON object back from the parser, it will be an

NSDictionary with strings for keys, and the objects will be NSStrings, NSNumbers, NSNulls, and NSArrays.

4.3 Create new POST call

Creating a POST call is very similar to a GET call. The only difference is for httpMethod, you need to put @"POST" or @"PUT", and you need to include the data from the JSON parsed dictionary. **Figure 4.5** shows the general format of a POST call.

```
-(void)updateAccountWithSender:(id)sender
{
   NSError* errora = nil;

   NSData* data = [NSJSONSerialization dataWithJSONObject:[[GNFWAccount account] accountDictionary]
                                                  options:NSJSONWritingPrettyPrinted
                                                    error:&errora];
   [_cmsService.httpService callToCMSWithURL:@"http://cms.goodnews-trueliving.com/API/UserAccount/Edit"
                                  httpMethod:@"POST"
                                        data:data
                                      sender:sender
                                     handler:^(NSData* receivedData, NSError* error, id context, id sender)
    {
       if (error == nil)
       {
          NSDictionary* jsonDictionary = [NSJSONSerialization JSONObjectWithData:receivedData
                                                                          options:NSJSONWritingPrettyPrinted
                                                                            error:NULL];

          int status = [[jsonDictionary objectForKey:@"Status"] integerValue];
          switch (status)
          {
             case -1:
                //TODO: Handle Unhandled Exception from CMS
                break;
             case 0:
                  [[GNFWAccount account] saveToUserDefaults];
                break;
             case 1:
                break;
             default:
                break;
          }
       }
    }];
}
```

Figure 4.5 Example POST Call

This is a good example of a POST call. The first thing you will need to do is serialize the NSDictionary into JSON data, this data will be stored in an NSData object. This data is then included in the call to the CMS. It will return the same way as it did with the GET call, so see the end of **Section 4.2** for more information.

5. Storyboard

This section will walk the user through adding and removing views from the Storyboard as well as create new segues and interacting with separate nib (user interface) files, **Figure 5.1** shows the complete storyboard.

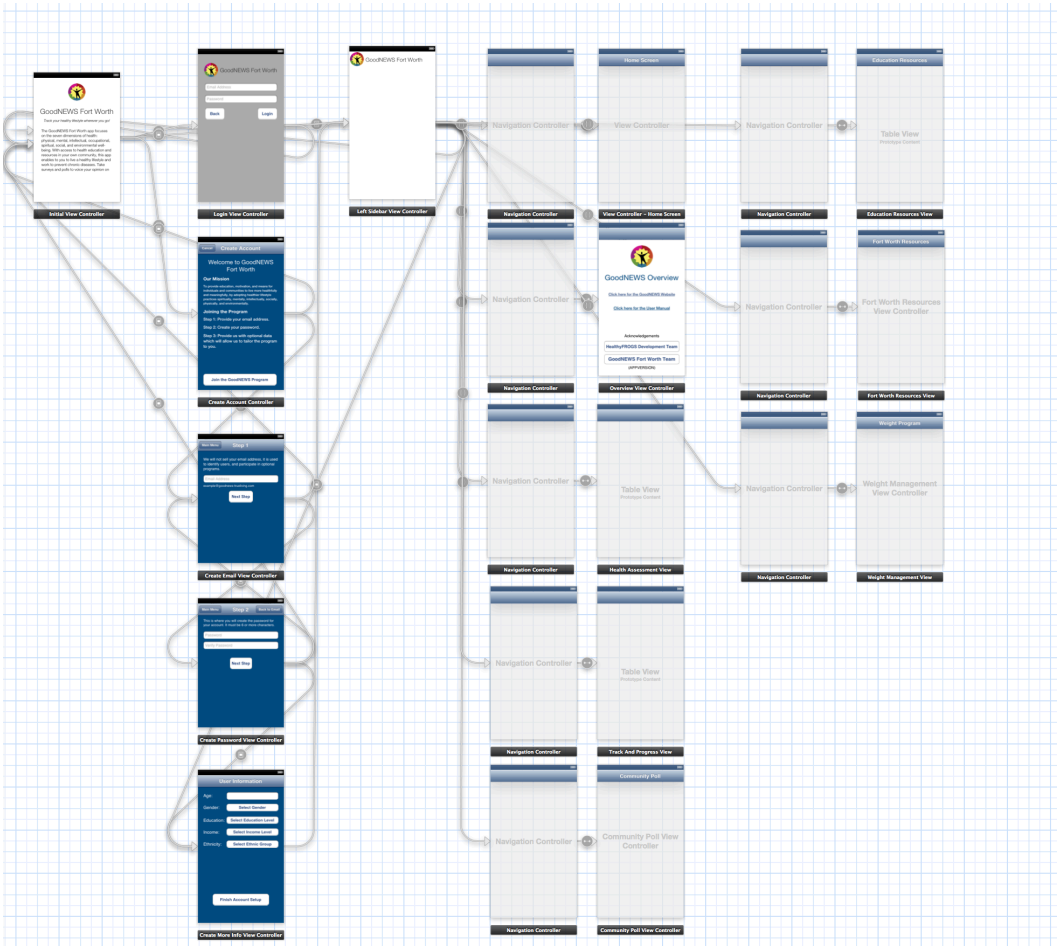


Figure 5.1 MainStoryboard.Storyboard

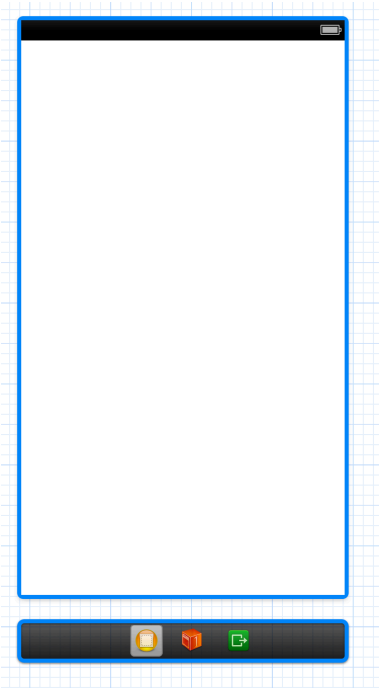
5.1 Adding to the Storyboard

Adding views to the storyboard is a simple process, there will be a step by step guide with images on how to add items to the storyboard, associate the views with a view controller, and hook up IBObjects and IBActions.

Step 1:

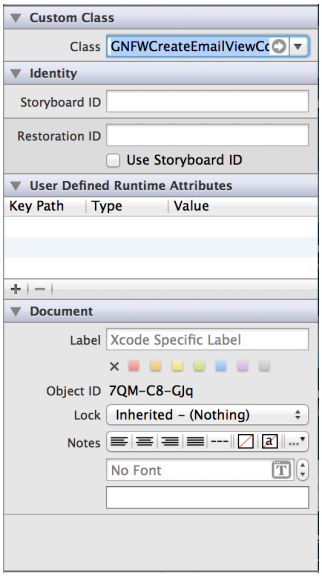
These are the options to choose from for view controllers, each one serves a specific function. To create a new one, just click and drag to the storyboard grid to create a new one.

Step 2:



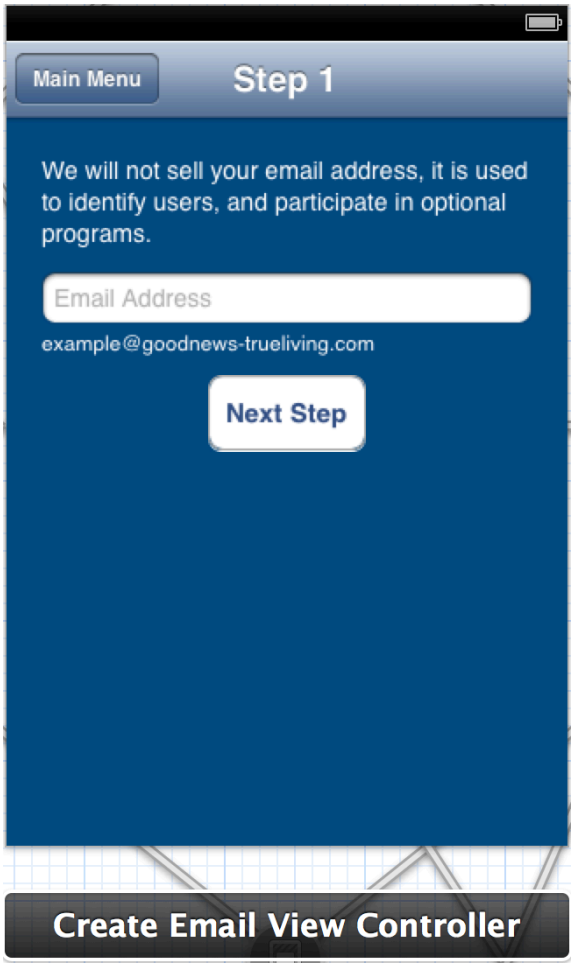
This is what a standard view controller will look like.

Step 3:



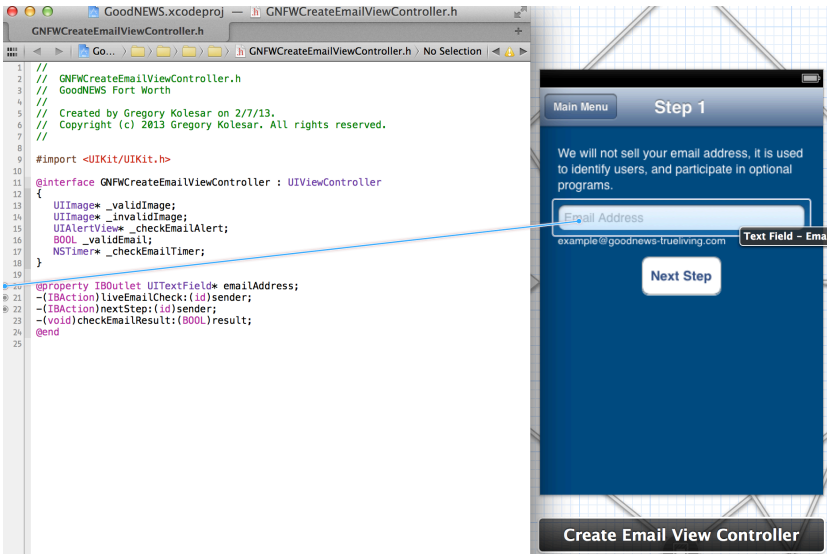
On this screen, in the right side bar, 3rd option on the top, you will see an option for custom class. This is important, because it will allow you to interact with any text fields, buttons, etc. that you put on the view. To do this, you need to create a new subclass of the view you chose.

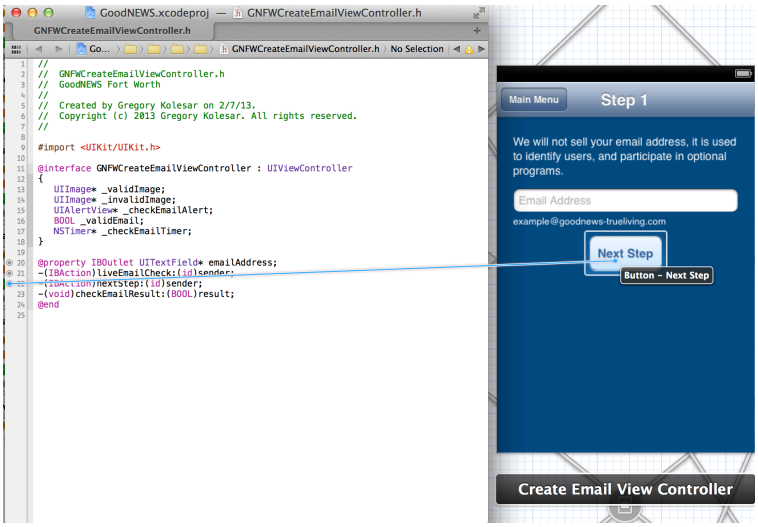
Step 4:



Here, 2 labels, a button, and a text field are added, and since it is linked to a view controller, we can now hook up these objects to interact with the code.

Step 5:





To connect the Interface Builder with the objects in the code, simply create an IBAction, which is just a method, or an IBOutlet which is a UI object. A little circle will appear in the margin, and click in the circle and move it to the corresponding object. When you do this, it will associate to the object in code allowing the user to interact with it. The code can modify text on objects, get values from the objects, and more.

5.2 Removing a view from the Storyboard

To remove a view from the storyboard, just select it, and press delete on your keyboard. It will automatically remove all links to the code.

5.3 Create/Remove Segues

Segues are important in storyboards, because they guide the flow of the application. The developer can use segues to transition between views when a certain action occurs. They are created in the story board and can be either activated on a button push, **Figure 5.1**. A segue can get a name label, **Figure 5.2**, and be called from code, **Figure 5.3**. To setup a segue to be called from code, all the developer needs to do is control + click on the blank space in the view and drag as shown in **Figure 5.1.**

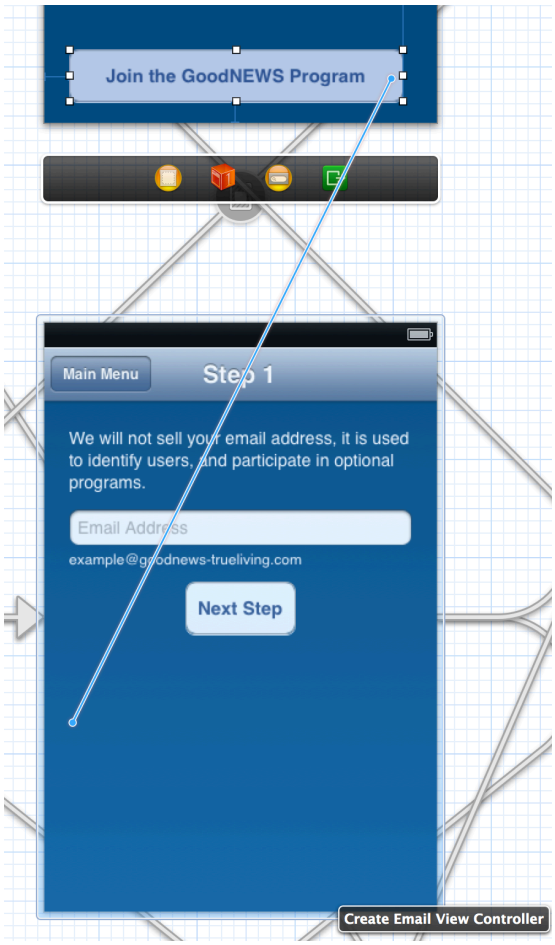


Figure 5.1 Connecting a Segue to a button push, just control + click on the button and drag to the corresponding view controller.

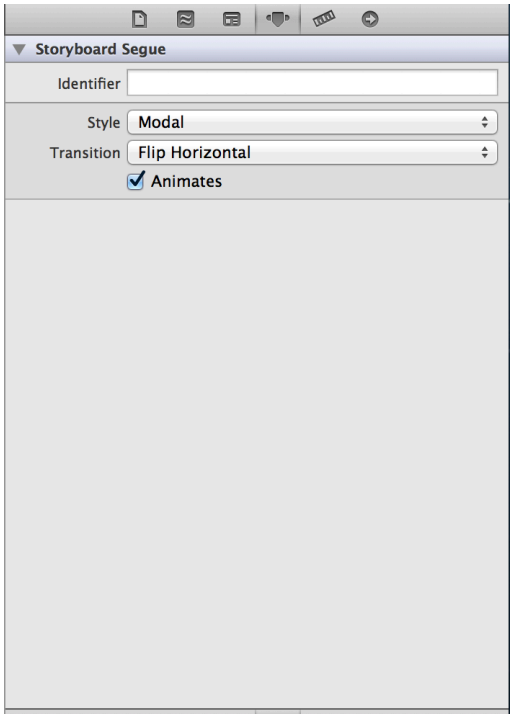


Figure 5.2 Segue Identifier, this is used to call the segue transition from code, also the style of the transition can be set from here.

```
-(void)userSettings
{
    [self performSegueWithIdentifier:@"accountsettings" sender:self];
}
```

Figure 5.3 Use the Identifier set in Figure 5.2 to call the transition

5.4 Use a nib file with the Storyboard

Sometimes it is easier to use a separate nib file (Interface Builder File) with the storyboard, a few reasons could be issues with Auto Layout, or you need to access it out of sequence from the storyboard, the map view and web browser are 2 examples from this program. A nib file is created the same way as a storyboard, except it is a single view controller. Calling the nib from code is easy just enter the name of the nib file without the ".xib" at the end, see **Figure 5.4**.

```
_resourceView = [[[NSBundle mainBundle] loadNibNamed:@"GNFWFortWorthResource" owner:self options:nil]objectAtIndex:0];
```

Figure 5.4 Code to get a nib file

6. Adding/Removing from the Main Menu

This section will give an overview to the developer on how to add/remove and modify components on the main menu.

6.1 The Main Menu Arrays

Look in the GNFWLeftNavView, and in the init method, there are 2 arrays, _data and _images. The data array holds the text that will be displayed on the main menu, and the images array holds the little images that are visible. The images need to be imported into the project before they can be used, so drag the image to the project navigator on the left side. To put text and an image into the menu, put the NSString you want in the data array, and put the image name as an NSString in the same index in the array.

```
-(id)initWithStyle:(UITableViewStyle)style
{
    self = [super initWithStyle:style];
    if (self)
    {
        _data = [NSArray arrayWithObjects:@"Health Assessment",
                 @"Track and Progress",
                 @"Education Resources",
                 @"Fort Worth Resources",
                 @"Community Poll",
                 @"GoodNEWS Overview",
                 @"",
                 @"Weight Management", nil];
        _images = [NSArray arrayWithObjects:@"bar_graph.png",
                   @"heart.png",
                   @"education.png",
                   @"fortworth.png",
                   @"community.png",
                   @"tutorial.png",
                   @"",
                   @"scale.png", nil];
    }
    return self;
}
```

Figure 6.1 Main Menu Information Arrays

7. Hard Coded GoodNEWS Text Locations

This section will inform the developer to where they can find the locations in the app where the text is hard coded.

7.1 GNFWCreateMoreInfoViewController.m

This class contains the text the user sees when they are entering the demographic data, if you need to change the choices, it is done here in the arrays.

```
-(void)selectGender:(id)sender
{
   [self dismissKeyboard];
   _actionSheet = [[GNFWPickerActionSheet alloc] initWithTitle:@"Gender"
                                                      delegate:self
                                                          data:[[NSArray alloc] initWithObjects:@"Male",
                                                                                                @"Female",@"Other",@"No Response", nil]
                                                        button:_genderButton];
   [self.view addSubview:_actionSheet.view];
}

-(void)selectEducationLevel:(id)sender
{
      [self dismissKeyboard];
   _actionSheet = [[GNFWPickerActionSheet alloc]initWithTitle:@"Education Level"
                                                     delegate:self data:[[NSArray alloc]
                                                                         initWithObjects:@"Some High School",
                                                                         @"High School Diploma", @"GED",
                                                                         @"Some College",@"Bachelor's Degree",
                                                                         @"Higher Level College",@"No Response", nil]
                                                       button:_educationButton];
   [self.view addSubview:_actionSheet.view];
}

-(void)selectIncomeLevel:(id)sender
{
      [self dismissKeyboard];
   _actionSheet = [[GNFWPickerActionSheet alloc]initWithTitle:@"Income Level"
                                                     delegate:self data:[[NSArray alloc]
                                                                         initWithObjects:@"Less than $25,000",
                                                                         @"$25,000-$50,000", @"$50,001-$100,000",
                                                                         @"$100,001+",@"No Response", nil]
                                                       button:_incomeButton];
   [self.view addSubview:_actionSheet.view];
}

-(void)selectEthnicGroup:(id)sender
{
      [self dismissKeyboard];
   _actionSheet = [[GNFWPickerActionSheet alloc]initWithTitle:@"Ethnicity"
                                                     delegate:self data:[[NSArray alloc]
                                                                         initWithObjects:@"African-American",
                                                                         @"Asian", @"Caucasian",
                                                                         @"Hispanic",@"Middle Eastern",
                                                                         @"Native American",@"Other"
                                                                         ,@"Pacific Islander",@"No Response", nil]
                                                       button:_ethnicGroupButton];
   [self.view addSubview:_actionSheet.view];
}
```

Figure 7.1 GNFWCreateMoreInfoViewController.m

7.2 UNTHSC.xib

This nib file contains the data for the UNTHSC screen, it has the pictures and text. The developer can modify the nib file, and they will see the changes in the app.

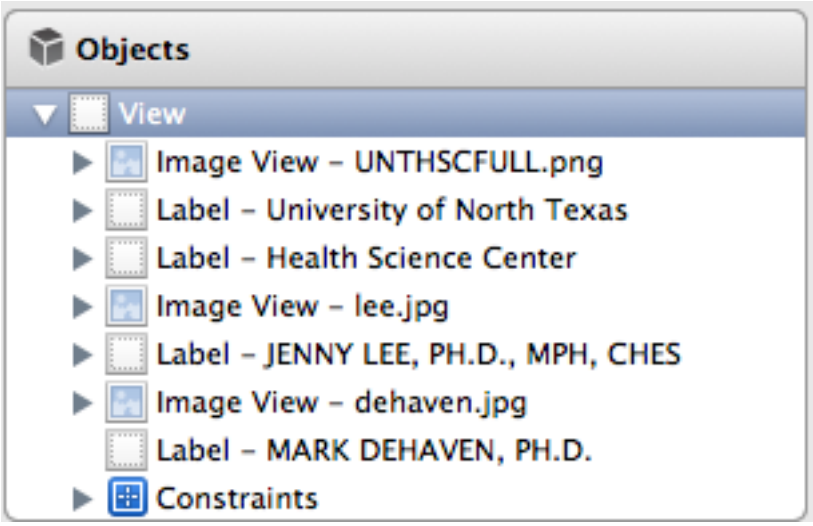


Figure 7.2 UNTHSC NIB objects

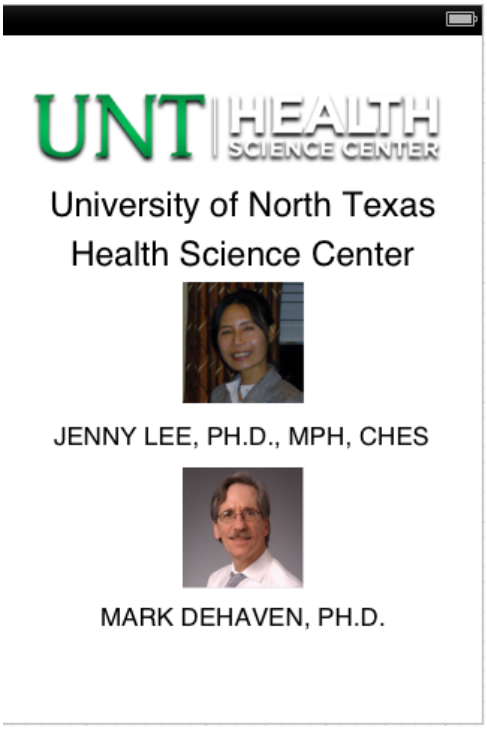


Figure 7.3 UNTHSC NIB File View

7.3 WMEnrollmentView.xib

This nib file contains the data for the Weight Management enrollment screen it has the pictures, description, and enroll button. The developer can modify the nib file, and they will see the changes in the app.

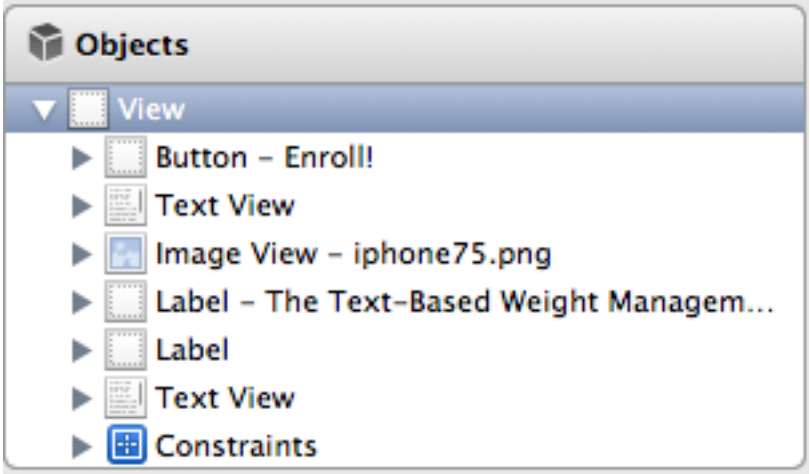


Figure 7.4 WMEnrollment NIB Objects

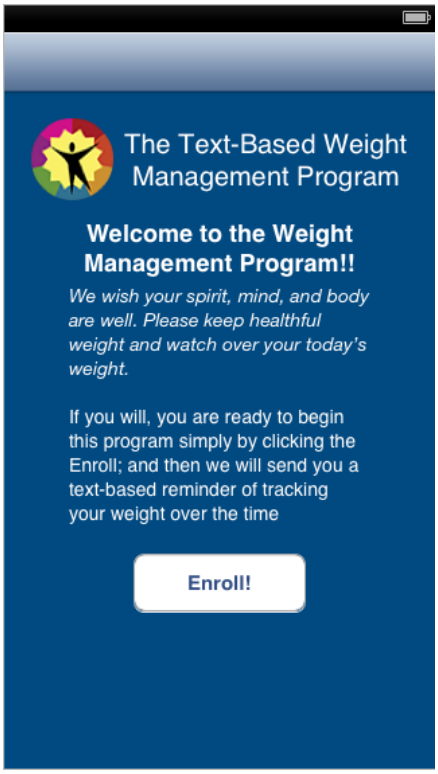


Figure 7.5 WMEnrollment NIB View

7.4 Initial Screen

The Initial Screen contains an overview of what the GoodNEWS program is about. The text can be changed via the storyboard, and is contained within a TextView.

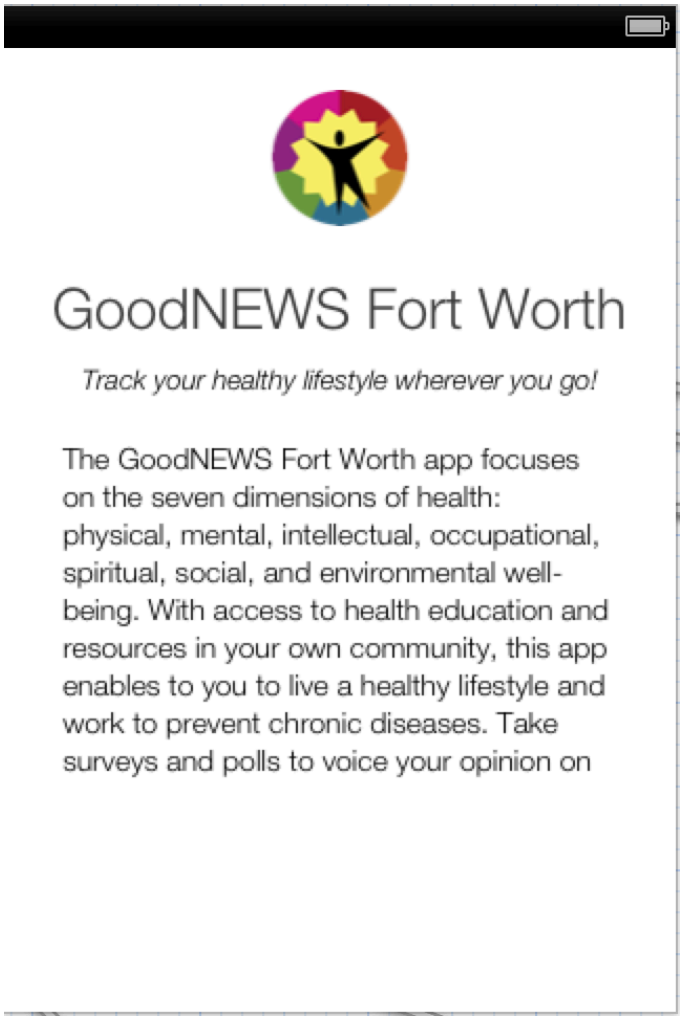


Figure 7.6 Initial screen TextView

8. Submitting to the App Store

Apple had great documentation on their website on how to submit an app. Follow the link below to visit Apple's tutorial.
http://developer.apple.com/library/ios/#documentation/ToolsLanguages/Conceptual/YourFirstAppStoreSubmission/AboutYourFirstAppStoreSubmission/AboutYourFirstAppStoreSubmission.html

This same process can be used to submit updates to the app store.